Aplicación para reconocimiento visual y datación de jeroglíficos egipcios

Trabajo de Fin de Máster

Asignatura: 105132

Máster en Investigación en Ingeniería del Software y Sistemas Informáticos, UNED

E.T.S. Ingenieros en Informática

Director:

Dr. Carlos Cerrada Somolinos

Estudiante:

Jaime Duque Domingo

Aplicación para reconocimiento visual y datación de jeroglíficos egipcios

Asignatura: 105132

Trabajo Fin de Máster en Ingeniería de Software y Sistemas Informáticos. Itinerario de Ingeniería de Sistemas Informáticos

Máster en Investigación en Ingeniería del Software y Sistemas Informáticos (E.T.S. Ingenieros en Informática)

Director:

Dr. Carlos Cerrada Somolinos

Estudiante:

Jaime Duque Domingo

Resumen

Este trabajo estudia un mecanismo novedoso de reconocimiento e identificación de cartuchos egipcios, representando nombres de faraones, mediante la localización de los jeroglíficos que lo componen.

Se utilizan distintas técnicas aprendidas en la asignatura "Sistemas de Percepción Visual" del "Máster en Investigación en Ingeniería del Software y Sistemas Informáticos". Así mismo se aplican otros conocimientos adquiridos durante la realización del máster. El trabajo corresponde al área de Investigación en Sistemas Informáticos.

Las técnicas utilizadas en el reconocimiento clásico de textos latinos no se comportan bien ante textos con ruido donde es difícil extraer los jeroglíficos para su cálculo de momentos o reconocimiento mediante redes neuronales.

El trabajo se organiza en distintos bloques recogiendo el estado del arte actual de distintas técnicas de reconocimiento, el nuevo método desarrollado para el reconocimiento indicando las técnicas utilizadas, el análisis y diseño de la aplicación, y finalmente las pruebas que se han desarrollado sobre la aplicación.

La aplicación utiliza un servicio web de reconocimiento que recibe imágenes y devuelve de qué faraón y a qué dinastía pertenece. Esta técnica permite realizar modificaciones sobre el algoritmo sin que los clientes se vean alterados. Además permite aumentar la potencia de cálculo del algoritmo, permitiendo que los cartuchos se reconozcan en unos pocos segundos.

El documento ha sido elaborado para explicar de la mejor manera posible el trabajo, y el algoritmo novedoso utilizado en el reconocimiento de objetos.

El conocimiento personal de todos los jeroglíficos egipcios, junto con el diccionario del egipcio antiguo es complicado. El trabajo abre un mundo de posibilidades de cara al reconocimiento de textos completos en jeroglífico, lo que podría facilitar a arqueólogos y estudiosos su trabajo en un futuro.

Abstract

This work provides a new method to recognize and identify egyptian cartouches, looking for the hieroglyphs of which they are composed. These cartouches represent the throne name of the kings.

Several techniques are used, which have been studied in the subjet “Sistemas de Percepción Visual”, of “Máster en Investigación en Ingeniería del Software y Sistemas Informáticos”. Also other techniques learnt during the master have been applied. This work is part of the area “Investigación en Sistemas Informáticos”.

The common methods used to recognise latin texts don´t work very well when there is a big noise and it is difficult to extract the hieroglyphs to calculate moments or use neural networks.

This work is composed of different blocks, including the state-of-art of several techniques of artificial vision, the new method developed, the analysis and the design, and finally the tests that have been done.

The program uses a service web that receives an image, and that returns the name and the dinasty of the King. Using a service web allows to modify the program without the need to update the clients. It also allows to use the power of a powerful computer, running faster the algorithms. The document explains how this new algorithm works.

This work gives new possibilities to recognise the hieroglyphs, and opens a new way to recognise longer texts and inscriptions. With a grammar and a dictionary it would be possible, helping researchers and archeologists in the future.

Índice

Índice de Ilustraciones

Introducción

Este trabajo estudia un mecanismo novedoso de reconocimiento e identificación de cartuchos egipcios, representando nombres de faraones, mediante la localización de los jeroglíficos que lo componen.

Se utilizan distintas técnicas aprendidas en la asignatura "Sistemas de Percepción Visual" del "Máster en Investigación en Ingeniería del Software y Sistemas Informáticos". Así mismo se aplican otros conocimientos adquiridos durante la realización del máster. El trabajo corresponde al área de Investigación en Sistemas Informáticos.

Las técnicas utilizadas en el reconocimiento clásico de textos latinos no se comportan bien ante textos con ruido donde es difícil extraer los jeroglíficos para su cálculo de momentos o reconocimiento mediante redes neuronales.

El trabajo se organiza en distintos bloques recogiendo el estado del arte actual de distintas técnicas de reconocimiento, el nuevo método desarrollado para el reconocimiento indicando las técnicas utilizadas, el análisis y diseño de la aplicación, y finalmente las pruebas que se han desarrollado sobre la aplicación.

La aplicación utiliza un servicio web de reconocimiento que recibe imágenes y devuelve de qué faraón y a qué dinastía pertenece. Esta técnica permite realizar modificaciones sobre el algoritmo sin que los clientes se vean alterados. Además permite aumentar la potencia de cálculo del algoritmo, permitiendo que los cartuchos se reconozcan en unos pocos segundos.

El documento ha sido elaborado para explicar de la mejor manera posible el trabajo, y el algoritmo novedoso utilizado en el reconocimiento de objetos.

El conocimiento personal de todos los jeroglíficos egipcios, junto con el diccionario del egipcio antiguo es complicado. El trabajo abre un mundo de posibilidades de cara al reconocimiento de textos completos en jeroglífico, lo que podría facilitar a arqueólogos y estudiosos su trabajo en un futuro.

Metodología

Para la realización de este trabajo se han utilizado múltiples fuentes bibliográficas relativas al campo de la visión artificial, así como fuentes relativas al desciframiento de jeroglíficos egipcios.

La primera fase del trabajo consistió en un estudio de proyectos parecidos, métodos y algoritmos existentes de visión artificial, y la búsqueda de un conjunto de jeroglíficos sobre los que se pudieran realizar las pruebas necesarias.

La segunda parte del trabajo se centró en la búsqueda de un método para conseguir el objetivo de reconocimiento de cartuchos reales. Este trabajo llevó mucho tiempo intentando conseguir un algoritmo que fuera lo más eficiente posible.

La tercera parte del trabajo consistió en probar el algoritmo sobre distintos ejemplos. Estas pruebas mostraron debilidades existentes que obligaron a múltiples ajustes del código.

Este documento pretende mostrar la memoria de los diversos pasos del trabajo de investigación, explicando el método seleccionado así como el estado del arte de las distintas técnicas estudiadas.

1. Formulación del problema

La antigua civilización egipcia ha sido una de las culturas más fascinantes y duraderas que ha existido. Ha pasado menos tiempo desde nuestros días hasta la época en la que vivió Cleopatra, que desde Cleopatra hasta el faraón que construyó la gran pirámide, Kheops. Durante 3000 años, la civilización egipcia mantuvo una forma de vida y cultura bastante similar. La historia de Egipto se extiende durante treinta dinastías, en cada una de las cuales reinaron distintos faraones. Aproximadamente hubo unos 150 faraones, alguno de los cuales pudieron reinar de forma paralela en el Norte y el Sur del país, llamado respectivamente el Bajo y Alto Egipto.

La escritura egipcia es bastante compleja, si bien se basa en un alfabeto de unos 800 caracteres de uno, dos o tres fonemas. A diferencia de lo que se pensaba inicialmente, los jeroglíficos no son pictogramas, sino que son fonemas. No obstante, algunos de ellos puedan ser pictogramas que ayuden a la comprensión en la lectura. De particular interés son los nombres de los faraones o reyes. Para distinguir un nombre real los egipcios lo redondeaban con un rectángulo con las esquinas curvas, al que los franceses denominaron cartucho, por el parecido con sus cartuchos de munición utilizados durante la expedición de los soldados de Napoleón durante el siglo XVIII.

Ilustración 1: Cartucho de Ptolomeo

Gracias a los cartuchos reales, Champollion pudo comenzar a descifrar los jeroglíficos. Los nombres de los faraones habían perdurado en la memoria y en los documentos griegos y romanos, si bien la escritura egipcia se olvidó a partir de los primeros siglos del cristianismo. Champollion consiguió descifrar algunos nombres como el de Ramsés, Tutmosis, Ptolomeo o Cleopatra, y gracias a la piedra de Rosetta, sembró las bases de la traducción del resto del texto, y no sólo de nombres reales. Esto se pudo conseguir ya que la piedra de Rosetta reflejaba un decreto escrito en 3 lenguas diferentes: jeroglífico egipcio, demótico egipcio y griego antiguo. El conocimiento del demótico y del griego le permitió a Champollion iniciar las bases de la traducción, a pesar del estado incompleto de la piedra.

Los nombres reales no sólo nos sirven para saber quién era el faraón que aparece en una inscripción. Una aplicación muy interesante es la datación de los textos. Aunque no siempre es correcto, ya que algunos faraones tenían la costumbre de suplantar inscripciones más antiguas cambiando los nombres que aparecían por el suyo, si es válido en la mayoría de los casos. Por ejemplo, la piedra de Rosetta nos habla sobre Ptolomeo IV. Gracias a esto podemos intuir que la piedra es del año 196 a.C. aproximadamente.

Tanto para aficionados a la egiptología, como para profesionales, sería excelente tener una herramienta, que podría ser móvil, que nos permitiera fotografiar un cartucho real, y decirnos a qué faraón y a qué fecha correspondería aproximadamente. Este trabajo busca dicha finalidad. Con un teléfono móvil podríamos ir a Egipto, y ser capaces de traducir los nombres y averiguar a qué periodo corresponden los monumentos. Esto se llevaría a cabo utilizando distintas técnicas de visión artificial. Se probarían distintos métodos de transformación de imágenes y reconocimiento, con el fin de estimar el mejor método posible.

Una vía abierta sería la traducción de más textos, lo cual podría ser un reto a mayores más a allá del máster, por ejemplo en doctorado. La traducción de textos consistiría en el reconocimiento visual de los 800 caracteres egipcios, y su cruce contra un diccionario egipcio-español, así como contra una base gramatical. La complejidad de la gramática unida a la distinta composición de escritura, ya que los egipcios podían por ejemplo escribir de derecha a izquierda, de arriba abajo, o de izquierda a derecha, hace este proyecto mayor y más complejo.

2. Objetivos

2.1. Objetivo general

Este proyecto es ante todo un proyecto de investigación. Dada la complejidad del tema, es diferente a otro tipo de proyectos donde desde el primer momento existe una certeza de éxito. Se pretende mantener una base de datos de unos 50 faraones principales, e intentar desarrollar una aplicación capaz de fotografiar un cartucho egipcio y devolver el nombre del faraón, y la datación o fecha de dicho faraón. Para conseguir este objetivo se compararán y probarán diferentes técnicas, seleccionando la que proporcione mejores resultados de reconocimiento. El proyecto hará uso de las distintas técnicas vistas en la asignatura "Sistemas de Percepción Visual". Se plantea implementar la aplicación como un servicio web consumible tanto por una aplicación web cliente como por una APP Android (aplicación para móviles).

2.2. Objetivos específicos

1. Plan de proyecto de las distintas fases, incluyendo una fase de investigación previa.
2. Investigar distintos métodos de pre-tratamiento de la imagen, con el fin de normalizarla y segmentarla obteniendo el cartucho a identificar. De esta forma se eliminaría el resto de la imagen no válida.
3. Investigar los distintos métodos posibles de reconocimiento: comparación binaria, momentos de Hu, redes neuronales, etc.
4. Seleccionar un método de reconocimiento, y realizar una aplicación móvil que nos permita utilizarlo, realizando el procesamiento dentro del dispositivo móvil (no por Internet)

3. Investigación

3.1. Introducción

La mayor dificultad de la visión artificial consiste en crear algo diferente a cualquier programa tradicional, y que nos permita dar respuesta a problemas que normalmente sólo son capaces de resolver humanos. Los programas tradicionales se limitan a seguir una secuencia de instrucciones prefijadas de acuerdo a unos requisitos y entradas posibles conocidas. En la visión artificial las entradas no son conocidas a priori, y debemos ser capaces de dar respuestas en función de probabilidades.

Los jeroglíficos egipcios representan la escritura de una lengua que perduró miles de años. Durante los siglos evolucionó, aunque se mantuvo más estable incluso que las lenguas actuales. Los egipcios de la época del emperador romano Trajano eran capaces de leer aproximadamente la escritura de la época de las pirámides, hecha 2500 años antes.

Antes de abordar los métodos de reconocimiento de los jeroglíficos, debemos ver cuáles son sus puntos importantes de cara a un reconocimiento visual. Dichos puntos nos guiarán en las posibles aproximaciones que realicemos a su reconocimiento.

Los principales puntos a tener en cuenta son:

- A diferencia de lo que se pensaba inicialmente, los jeroglíficos representan fonemas. Dichos fonemas pueden ser mono-consonánticos, bi-consonánticos, tri-consonánticos e incluso representar pictogramas aclaratorios. Desde nuestro punto de vista deberemos dar cabida a todos, ya que aunque utilicemos los caracteres fonéticos para describirlos, si encontramos un pictograma tendremos que considerarlo al poder ser necesario en la traducción.

- Escriben en distintos materiales: piedra, madera, papiro, fayenza, oro, plata, bronce, piedras preciosas, etc.

- Utilizan una escritura diferente en papiro: hierático. El hierático es una escritura esqueletizada que les permitía escribir más rápido al tratarse de documentos en papiro.

- Podían escribir de izquierda a derecha, de derecha a izquierda, o de arriba abajo. La dirección del texto depende de hacia dónde miran los jeroglíficos. Por ejemplo, si aparece una persona sentada mirando hacia la derecha, la dirección de lectura será de derecha a izquierda. Además podían combinar de izquierda a derecha, o de derecha a izquierda, con de arriba abajo. Así mismo podían agrupar ciertos caracteres de formas diferentes. Desde el punto de vista de nuestra

aplicación consideraremos la escritura de izquierda a derecha, de arriba abajo, y consideraremos posibles agrupaciones.

Ilustración 2: Escritura de arriba a abajo, y de izquierda a derecha

- Su escritura sobre piedra y otros materiales resultaba ser bastante perfecta. Si examinamos distintos relieves veremos que normalmente los mismos jeroglíficos son muy similares. Esto es un punto fuerte que tomaremos en consideración en nuestro sistema de reconocimiento. Esta situación ocurría ya que los principales escultores solían utilizar modelos de referencia elaborados por escribas, y los escribas representaban menos de un 10% de la población.

- Al escribir en piedra utilizaban las técnicas de bajo relieve o sobre relieve. En bajo relieve grababan los jeroglíficos sobre una pared lisa. En sobre relieve tallaban el resto de la pared. A veces cometían errores, pero estos los corregían utilizando yeso y pintando. La gran mayoría de los jeroglíficos estaban pintados. En otros materiales como los metales preciosos o madera se utilizaba la técnica de bajo relieve. Se podía pintar encima, o bien integrar piedras preciosas.

- Muchos jeroglíficos estaban pintados, aunque hoy en día dicha pintura ha desaparecido en la mayoría de los relieves en piedra. En madera o dentro de algunas tumbas es habitual encontrar aún escritura pintada. El tema de la pintura es muy importante, ya que como veremos es uno de los puntos que nos hace rechazar el método de la prominencia (salience) basada en contraste. En el trabajo "Automatic Egyptian Hieroglyph Recognition by Retrieving Images as Texts" de la Universidad de Ámsterdam, se planteaba utilizar este método para el reconocimiento, de la misma que se utiliza en algunos sistemas de reconocimiento de texto, si bien nosotros lo descartamos inicialmente al considerar que cuando hay ruido puede resaltar ciertas regiones de la imagen no válidas, provocando confusión.

- Uno de los problemas principales de los textos es que han llegado bastante deteriorados. Los siglos han ido destruyendo poco a poco las inscripciones. Dicha destrucción se puede haber producido de manera natural: erosión de arena, agua, terremotos, viento; o bien producida por el hombre: los propios egipcios en determinados capítulos de la historia borraban inscripciones de faraones anteriores a base de picar las inscripciones, o bien los cristianos que optaron por

eliminar determinadas imágenes por considerarlas herejía, o finalmente durante la guerra francesa existía la afición de disparar haciendo blanco en diferentes monumentos.

- Existen cientos de miles de textos diseminados por todo el mundo. Y aun así se calcula que una gran mayoría del material arqueológico se encuentra enterrado.

- Los nombres de los faraones eran compuestos. Al igual que hoy tenemos apellidos, ellos tenían un nomen y un prenomen, e incluso en algunos casos podían tener distintos nombres en función de su nacimiento, coronación, e incluso de un cambio de nombre: Tut-ankh-aton pasó a llamarse Tut-ankh-amon al volver a Tebas y recuperar el culto al dios Amón y abandonar el culto a Atón.

Como podemos ver, algunos de los puntos que hemos comentado nos guiarán en la investigación para obtener un método válido de reconocimiento, y que aunque inicialmente se centre en el reconocimiento de cartuchos reales, se pueda extender al reconocimiento de otros tipos de textos.

Antes de empezar con el trabajo en cuestión, debemos centrarnos en con qué textos queremos trabajar. Como hemos dicho existen cientos de miles de textos, pero de cara a nuestro trabajo tenemos que buscar un texto que se adapte a nuestras necesidades. Como el trabajo se centra exclusivamente en el reconocimiento de nombres reales, el texto seleccionado será la Lista Real de Abydos, con una copia disponible en el Museo Británico de Londres. La lista real de Abydos fue elaborada en la época de Ramsés II e incluía la relación de los faraones que habían existidos previamente. No todos los faraones fueron incluidos, ya que algunos fueron considerados non-gratos por determinados hechos de la historia, o simplemente por ser mujeres. Akhenaton, Tutankhamon o Hatshepsut son ejemplos de faraones excluidos de la lista. La Lista de Abydos incluye 76 nombres de faraones desde la primera dinastía hasta la dinastía XIX. La historia de Egipto de divide en 30 dinastías y distintos períodos. Así por ejemplo la edad dorada, el Imperio Nuevo, se encuentra en las dinastías XVIII, XIX, o XX. En dicho período aparecen nombres como Akhenaton, Tutankhamon, Ramsés II, Seti I. La primera dinastía corresponde a la unificación de Egipto bajo el faraón Menes hacia el año 3300 a.C. Existen dudas sobre si dicho rey era el propio rey Narmer, con la conocida paleta de Narmer en el Museo Egipcio del Cairo. Por otro lado, la IV dinastía corresponde a la de los grandes constructores de las pirámides: Keops, Kefrén y Micerinos, si bien dichos nombres vienen del griego, y no de su origen real en egipcio.

Ilustración 3: Lista Real de Abydos

Una vez que hemos decidido utilizar la Lista de Abydos, empezaremos a probar distintos métodos para ver cuál nos ofrece mejores resultados.

3.2. TRANSFORMACIÓN A ESCALA DE GRISES

El primer paso para poder operar con la imagen consiste en su transformación a una escala de 256 niveles de gris. Esto nos permite trabajar con un único valor por píxel en vez de tres valores que tiene el modelo RGB (Red Green Blue). La obtención de la escala de grises es simple. Basta con calcular la intensidad de cada píxel, y dicho valor será el nivel de gris.

$$I = \frac{R+G+B}{3}$$

Otra posible obtención del nivel de gris más utilizada consiste en aplicar a cada color una diferente porción del nivel gris. Así, la intensidad vendría definida por:

$$I = 0.299 \text{ x } R + 0.587 \text{ x } G + 0.114 \text{ x } B$$

Podemos ver cómo quedaría la siguiente imagen transformándola en escala de grises.

Ilustración 4: Transformación a escala de grises

3.3. Filtro Mediana

Es de particular interés el filtro de la mediana. Dicho filtro, como veremos posteriormente, permite reducir el ruido sin alterar los bordes principales de la imagen. Su aplicación es bastante simple, y consistiría en calcular la mediana de los 8 píxeles vecinos de un píxel dado junto con este. A partir de los 9 píxeles se ordenan las intensidades de menor a mayor y se elige la quinta, que equivale a la mediana.

Las ventajas son que atenúa el ruido impulsional (sal y pimienta), elimina efectos engañosos, y preserva bordes de la imagen. Los inconvenientes son que pierde detalles (puntos, líneas finas), que redondea las esquinas de los objetos, y que provoca un desplazamiento de los bordes

Ilustración 5: Imagen original en grises y aplicación de filtro mediana

3.4. Filtro Máximo y Mínimo

El filtro máximo también se conoce como filtro de erosión, debido a la propiedad que posee de adelgazar líneas. Asigna al píxel actual el valor máximo de entre los valores de sus vecinos. Los píxeles oscuros aislados más oscuros serán sustituidos por valores más altos con la consiguiente reducción de los píxeles cercanos al negro. Su ventaja es que elimina el ruido pimienta (píxeles negros), aunque sus inconvenientes son que sólo funciona cuando el ruido es exclusivamente tipo pimienta, y que tiende a aclarar la imagen.

El filtro mínimo a diferencia del anterior tiende a ensanchar las líneas negras de la imagen, por lo que también es conocido como filtro de dilatación. Selecciona el menor valor de entre sus vecinos para un píxel dado. Su ventaja es que elimina el ruido sal (píxeles blancos), aunque sus inconvenientes son que sólo funciona cuando el ruido es exclusivamente tipo sal y que tiende a oscurecer la imagen.

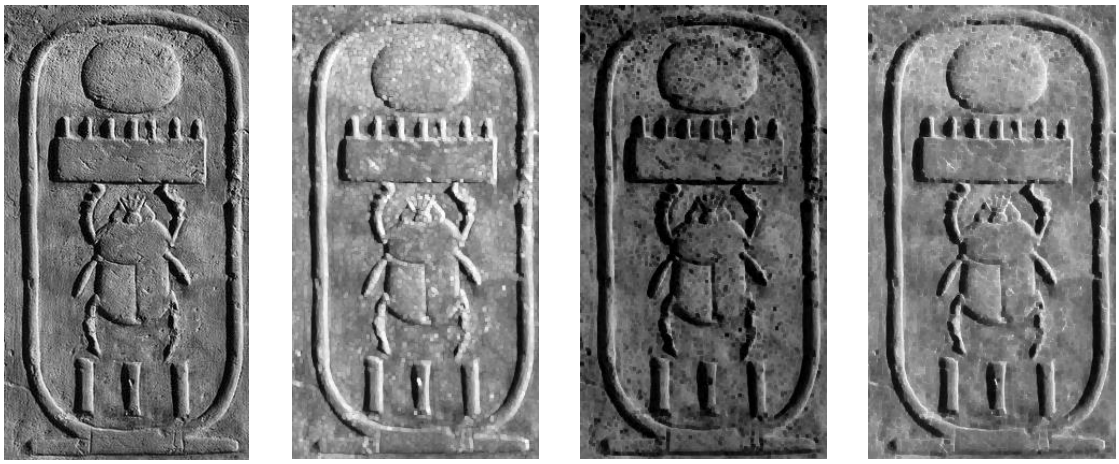


Ilustración 6: Imagen original en grises y aplicación de filtro máximo/mínimo y combinación de ambos

3.5. Operador de Sobel

Un primer intento de reconocer los bordes de los jeroglíficos y del cartucho se ha realizado utilizando el operador de Sobel. Si bien es cierto que en determinadas pruebas su resultado era bastante bueno, ha sido problemático utilizarlo debido entre otros factores a que no realiza supresión máxima como el algoritmo de Canny, o que tampoco intenta cerrar contornos abiertos, por lo que muchas veces los bordes son poco prácticos.

Matemáticamente, el operador utiliza dos kernels de 3×3 elementos para aplicar convolución a la imagen original para calcular aproximaciones a las derivadas, un kernel para los cambios horizontales y otro para las verticales. Si definimos A como la imagen original, el resultado, que son las dos imágenes

Gx y Gy que representan para cada punto las aproximaciones horizontal y vertical de las derivadas de intensidades, es calculado como:

$$G_x = \begin{vmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{vmatrix} * A \qquad G_y = \begin{vmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{vmatrix} * A$$

En cada punto de la imagen, los resultados de las aproximaciones de los gradientes horizontal y vertical pueden ser combinados para obtener la magnitud del gradiente, mediante:

$$G = \sqrt{G_x^2 + G_y^2}$$

Con esta información, podemos calcular también la dirección del gradiente, siendo 0 para bordes verticales con puntos más oscuros al lado izquierdo.

$$\Theta = \arctan\left(\frac{G_y}{G_x}\right)$$

A modo de ejemplo veamos qué bordes se obtienen sobre la siguiente imagen de grises después de aplicar un filtro de mediana para reducir ruido, y un filtro de máximos para aumentar contraste. Así mismo la imagen se ha binarizado para un umbral T=40.

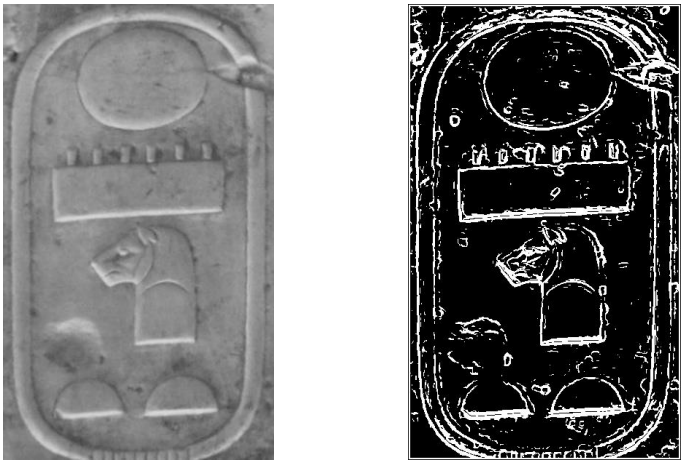


Ilustración 7: Aplicación del operador de Sobel

3.6. ALGORITMO DE CANNY

El operador de detección de bordes de Canny fue desarrollado por John F. Canny (Universidad de Berkeley, EEUU) en 1986 y se basa en un algoritmo de múltiples fases para detectar un amplio rango de bordes. Es sin duda el operador más utilizado en la detección de bordes. Es el algoritmo utilizado en nuestro trabajo para la detección de bordes, ya que sus resultados han sido más favorables que otros operadores como Sobel. Además, nuestro trabajo final requiere de bordes con supresión no máxima, algo

que nos proporciona Canny. Su comportamiento ante el ruido y otros factores, así como el resultado producido hace que lo elijamos para el trabajo. En el trabajo se han probado dos algoritmos de Canny, si bien finalmente se ha optado por utilizar el plug-in de ImageJ desarrollado por Thomas Boudier y Joris Meys. Los algoritmos probados son:

- Edge Detection by Canny-Deriche filtering, por Thomas Boudier y Joris Meys:

  http://imagejdocu.tudor.lu/doku.php?id=plugin:filter:edge_detection:start

- Algoritmo de Canny en Java, por Tom Gibara:

  http://www.tomgibara.com/computer-vision/canny-edge-detector

El algoritmo de Canny cumple los siguientes puntos:

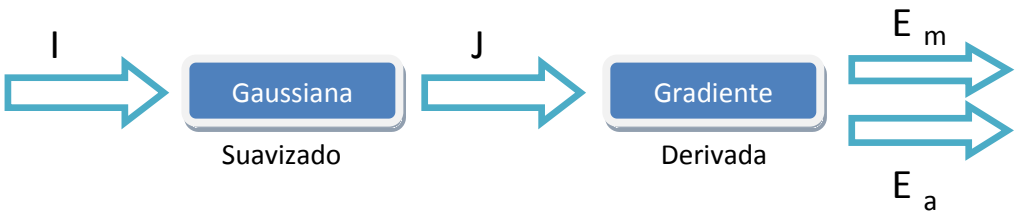
- Buena detección: el algoritmo debe marcar el mayor número real en los bordes de la imagen como sea posible.
- Buena localización: los bordes de marca deben estar lo más cerca posible del borde de la imagen real.
- Respuesta mínima: El borde de una imagen sólo debe ser marcado una vez, y siempre que sea posible, el ruido de la imagen no debe crear falsos bordes.

Las partes del algoritmo son:

- Obtención del gradiente.
- Supresión no máxima al resultado del gradiente.
- Histéresis de umbral a la supresión no máxima.
- Cierre de contornos abiertos.

Vemos a continuación qué hacemos en cada parte.

1. Obtención del gradiente:



Para obtener el gradiente se tiene que suavizar la imagen. Se aplica a la imagen I un suavizado gaussiano G o filtro gaussiano (también se puede aplicar otros filtros): J = I * G. Para calcular el kernel gaussiano G se utiliza la siguiente ecuación:

$$G(x,y)=\frac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}}$$

La imagen J se obtiene mediante convolución de I con G en cada píxel. Para cada píxel de la imagen J(i,j) se calculan las componentes del gradiente Jx y Jy, así como la magnitud de los bordes:

$$e_s(i,j)=\sqrt{J_x^2(i,j)+J_y^2(i,j)}$$

Finalmente se estima la orientación de la normal de los bordes:

$$e_0(i,j)=\arctan\left(\frac{J_y}{J_x}\right)$$

2. Supresión no máxima al resultado del gradiente:

El objetivo de este paso es obtener bordes de 1 píxel de grosor al considerar únicamente píxeles cuya magnitud es máxima en bordes gruesos y descartar aquellos cuyas magnitudes no alcancen ese máximo.

- Para todo punto se obtiene la dirección más cercana dk a 0º, 45º,90º y 135º en Ea(i,j).
- Si Em(i,j) es menor que uno de sus dos vecinos en la dirección dk,IN(i,j)=0. Si no IN(i,j)=Em(i,j).

3. Histéresis de umbral a la supresión no máxima:

Permite eliminar máximos procedentes de ruido. Como entrada tendremos IN, Ea, y dos umbrales T1 y T2 tal que T2>T1.

- Para todo punto en IN, y explorando en un orden:
    - Localizar el siguiente punto tal que IN(i,j) > T2
    - Seguir las cadenas de máximos locales a partir de IN(i,j) en ambas direcciones perpendiculares a la normal al borde siempre que IN>T1.
    - Marcar los puntos explorados.

La salida es un conjunto de bordes conectados de contornos de la imagen, así como la magnitud y orientación.

4. Cierre de contornos abiertos (Algoritmo de Deriche y Cocquerez)**:**

La imagen de entrada es una imagen de contornos binarizada (1= borde; 0=no borde).

- Para cada punto de borde de un extremo abierto se le asigna un código que determina las direcciones de búsqueda para el cierre del contorno.
- Para los píxeles marcados con este código se marca como píxel de borde el de máximo gradiente en las tres direcciones posibles.
- Se repiten los pasos hasta que se cierren todos los contornos.

Resultados

Los resultados obtenidos son bastante buenos. Después de jugar con los distintos umbrales de histéresis hemos elegido los valores T1=1f, T2=4.5f, representados en Java mediante las siguientes líneas de código:

```
detector.setLowThreshold(1f);

detector.setHighThreshold(4.5f);
```

Vemos a continuación el resultado de aplicar Canny a una imagen en escala de grises sobre la que hemos aplicado un filtro mediano de suavización y un filtro máximo de realce de contraste previamente. Como podemos observar los resultados son bastante buenos. Sin embargo no siempre es así, ya que en ciertos casos se ha detectado que se generan muchos bordes pequeños procedentes de ruido. Además, ciertos contornos no se cierran adecuadamente, lo cual nos provoca problemas si queremos aplicar el algoritmo de rellenado de regiones.

Ilustración 8: Aplicación del algoritmo de Canny (Tom Gibara)

3.7. ALGORITMO DE SUSAN

El algoritmo de SUSAN (Smallest Univalue Segment Assimilating Nucleus) es otro potente método de obtención de bordes de objetos. Fue propuesto por Smith & Brady en 1995. Se basa en la propiedad de que cada punto en una imagen está asociado al brillo de su área. Para implementarlo se aplica una máscara circular alrededor de cada punto que compara la intensidad de los píxeles vecinos con el píxel central (núcleo de la máscara). El área con intensidad similar al núcleo se llama área de SUSAN. Se repite el procedimiento para cada pixel de la imagen.

El área USAN varía dentro de la imagen dependiendo de su localización con respecto a propiedades especiales de la imagen. El área USAN es máximo dentro de un área, pero es menor en los bordes, e incluso menor en las esquinas. Esta es la propiedad en la que se basa el algoritmo de detección de esquinas.

El algoritmo es el siguiente:

- Determinar una máscara circular de aproximadamente 37 puntos alrededor del núcleo, para cada punto de la imagen.
- Calcular la diferencia de brillo entre cada pixel de la máscara y su núcleo.
- Sumar el número de píxeles (n) dentro de la máscara circular que tienen niveles de intensidad similares al núcleo (diferencia pequeña).
- Comparar n con g, el umbral geométrico que puede ser el valor máximo que puede tener n entre 2 (n_max/ 2).
- En las esquinas, el área USAN será siempre menor que la mitad del tamaño del área de la máscara, y se considerará un mínimo local.

Vemos a continuación el resultado de la aplicación del detector de bordes de SUSAN. El detector se comporta correctamente, si bien para nuestro trabajo utilizamos el algoritmo de Canny.

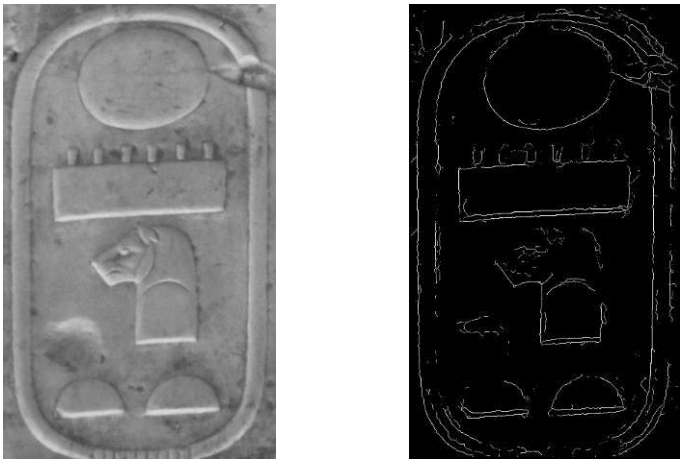


Ilustración 9: Aplicación del algoritmo de SUSAN

3.8. REGIONES POR FRONTERA

Un primer intento de obtención de extracción de los jeroglíficos consistía en la obtención de las regiones que formaba cada jeroglífico. La utilización de umbrales no era viable al tener los jeroglíficos el mismo color que el resto del cartucho, por lo que sólo se pudo recurrir al algoritmo de regiones por frontera o a métodos basados en prominencia (saliente).

Básicamente el algoritmo se basa en la utilización del gradiente y la Laplaciana. Se calcula el gradiente y la Laplaciana en cada punto de la imagen mediante convolución, y con dichos datos se evalúa la siguiente fórmula (según un umbral T):

$$S(x,y)=\begin{cases}0 & \text{si } G[f(x,y)]<\mathrm{T}\\ + & \text{si } G[f(x,y)]\geq \mathrm{T} \text{ y } L[f(x,y)]\geq 0\\ - & \text{si } G[f(x,y)]\geq \mathrm{T} \text{ y } L[f(x,y)]<0\end{cases}$$

Una región es tal que tiene la forma (...)(-,+)(0 o +)(+,-)(...). Para calcular el gradiente utilizamos el operador de Sobel visto anteriormente. Para calcular la Laplaciana utilizamos el siguiente operador simétrico mediante convolución:

$$L=\begin{vmatrix}0 & -1 & 0\\ -1 & 4 & -1\\ 0 & -1 & 0\end{vmatrix}$$

Se aplicó el algoritmo, con algunas pequeñas variantes, sobre la imagen de grises después de utilizar un filtro de mediana y de máximos con el fin de suavizar y realzar el contraste. El resultado produjo unos resultados bastante malos como podemos ver en la imagen siguiente.

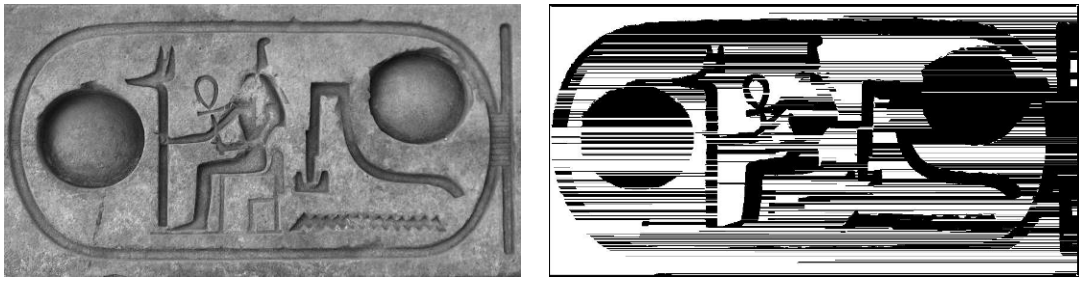


Ilustración 10: Primera prueba del algoritmo de obtención de regiones por frontera

Después de diversas pruebas se readaptó el algoritmo utilizando los bordes de Sobel como delimitadores de las regiones, y rechazando bordes cuyo tamaño era inferior a un umbral del 20% del mínimo entre la altura y la anchura de la imagen. Así mismo se excluyó el borde de los cartuchos y se eliminaron regiones de un tamaño pequeño. Para la eliminación de los bordes de los cartuchos se utilizó

un procedimiento que eliminaba las líneas laterales mayores que iban desde prácticamente arriba hasta abajo, o bien desde prácticamente la izquierda hasta la derecha. Podemos ver el resultado obtenido en la siguiente imagen.

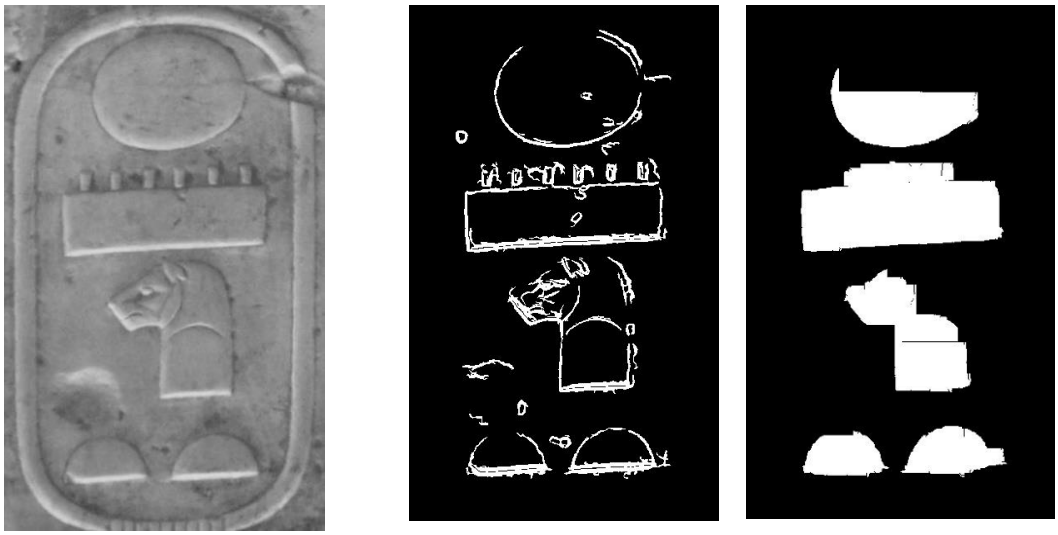


Ilustración 11: Obtención de regiones por frontera

Aparentemente los resultados no parecen malos, ya que la utilización de momentos invariantes, como los de Hu, nos permitiría reconocer los distintos jeroglíficos, siendo de esta manera también insensibles a la posición de giro. Sin embargo el comportamiento fue bueno en unas pocas imágenes, ya que por lo general los cartuchos tenían excesivo ruido, o los jeroglíficos muy próximos, lo que hacía que el sistema devolviera jeroglíficos en sitios donde no había, o bien jeroglíficos fusionados al estar muy próximos. Esta prueba, que inicialmente parecía la mejor solución, era muy problemática y finalmente fue descartada.

3.9. Modelos de atención a la visualización (salience)

Estos métodos basan su funcionamiento en intentar extraer aquellas zonas de la imagen que resultan visualmente interesantes. Lo pueden hacer de diversas formas, pero principalmente se utilizan dos: mediante diferencias de color existente entre distintos puntos de la imagen, o bien mediante la curvatura de los distintos bordes. Teóricamente siguiendo la curvatura de un borde, si coincide con otro deberían estar conectados. Algunos métodos de prominencia son extremadamente complejos, si bien hemos implementado alguno, y hemos realizado pruebas con distintos métodos disponibles en Internet.

Los métodos basados en el color no son interesantes en nuestro caso, ya que como apreciamos en las imágenes los jeroglíficos tienen el mismo color que el resto del cartucho. Sin embargo, los métodos basados en curvatura si son válidos ya que pueden devolver bordes basándose en la curvatura de los

jeroglíficos. No obstante, después de diversas pruebas su resultado no fue el esperado, produciéndose debido al excesivo ruido que en algunos cartuchos había. Esto originaba que se unificaran bordes con poco sentido. En el trabajo "Automatic Egyptian Hieroglyph Recognition by Retrieving Images as Texts" de la Universidad de Ámsterdam, se planteaba utilizar este método para el reconocimiento, de la misma que se utiliza en algunos sistemas de reconocimiento de texto, si bien nosotros lo descartamos al considerar que cuando hay ruido provoca mucha confusión al resaltar diversos puntos de la imagen como prominentes (seleccionados).

Normalmente los algoritmos de prominencia devuelven un mapa, el cual es utilizado para obtener la correspondiente extracción de regiones mediante binarización. Los tonos oscuros de salida suelen corresponder con las regiones interesantes.

3.9.1. COLOR SALIENCE

LIRE (Lucene Image Retrieval) , librería JAVA creada por ACM implementa un modelo de atención a la visualización (prominencia o salience) basándose en el estimador de F. W. M. Stentiford. Dicho estimador se basa principalmente en que dos píxeles próximos de color similar pertenecen al mismo jeroglífico. Dicho algoritmo fue publicado en 2013 en JAVA por Morgan & Claypool, 2013:

http://www.morganclaypool.com/doi/abs/10.2200/S00468ED1V01Y201301ICR025

Como hemos comentado no es el mejor método para resolver el problema, pero vamos a ver el resultado producido al ejecutar el algoritmo sobre la imagen original.

Ilustración 12: Mapa de prominencia utilizando el modelo de Stentiford

Podemos apreciar en la imagen anterior cómo los métodos basados en píxeles vecinos por el color no producen buenos resultados de cara a nuestro problema. Únicamente serían válidos si los jeroglíficos estuvieran pintados, pero en la mayoría de los casos no es así.

3.9.2. CURVATURE Y STRUCTURAL SALIENCE

El modelo de atención a la visualización basado en la estructura (structural salience) fue propuesto por Ammon Sha'ashua y Shimon Ullmann en 1988 en el MIT (Massachussets Institute of Technology). Es un modelo que se basa en las posibles curvas que pasan por cada píxel de la imagen, y cuáles de ellas son máximas. Por máximas entendemos aquellas curvas que podemos formar de mayor tamaño, independientemente de que existan espacios o zonas donde no hay píxeles. En el trabajo "Automatic Egyptian Hieroglyph Recognition by Retrieving Images as Texts" de la Universidad de Ámsterdam, se planteaba utilizar este método para el reconocimiento, de la misma que se utiliza en algunos sistemas de reconocimiento de texto

Este método, similar al utilizado por la Universidad de Amsterdam en su estudio, ha sido uno de los más estudiados en el trabajo. En un principio la idea pasaba por utilizar este método para obtener los jeroglíficos principales, y a partir de ahí segmentarlos obteniendo las regiones y procesarlos mediante o bien métodos de Hu o mediante una red neuronal de comparación de imágenes.

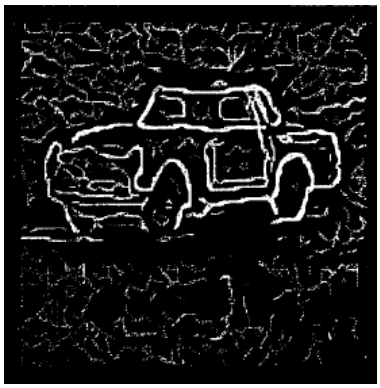


Ilustración 13: Curvature Salience Map después de 30 iteraciones

Para probar el algoritmo se ha utilizado la librería LTI-LIB, desarrollada por el Technical Computer Science (German Lehrstuhl fuer Technische Informatik (LTI)), RWTH Aachen University. Esta librería está escrita en C++, e incluye numerosos métodos válidos para visión artificial o tratamiento de imágenes. Nosotros nos centraremos en la clase *salienceEdges*, que implementa el algoritmo de Ammon Sha'ashua y Shimon Ullmann. Su entrada es una imagen de bordes, que podemos obtener mediante distintos métodos. La clase *cannyEdges* implementa el algoritmo de Canny, y la clase *susanEdges* implementa el algoritmo de Susan, otro potente método de detección de bordes.

El algoritmo utilizado se basa en la siguiente iteración, para cada píxel i:

$$\begin{cases} E_i^{(0)} = \sigma_i \\ E_i^{(n+1)} = \sigma_i + \rho_i \cdot \max\limits_{\rho_j \in \delta(\rho_i)} E_j^{(n)} \cdot f_{i,j} \end{cases}$$

Donde, σ_i son los valores locales de *salience*, $\delta(\rho_i)$ son los posibles vecinos del punto i, ρ_i toma un valor entre 0 y 1 dependiendo de si el píxel está activo o no. Si está activo se aproximará a 1, y si no lo está será menor que 1. $f_{i,j}$ representa una función de acoplamiento que varía en función del ángulo formado por la curva en el punto (α_k), y el incremento entre dos puntos Δs :

$$f_{k,k+1} = e^{-\frac{2\cdot\alpha_k\cdot\tan\left(\frac{\alpha_k}{2}\right)}{\Delta s}}$$

El siguiente código muestra la prueba desarrollada en C++ para probar este método en el reconocimiento de cartuchos. Utilizamos el algoritmo de Susan para la detección de bordes, y posteriormente calculamos el mapa de prominencia (saliency map).

```
// objetos para cargar y grabar ficheros
loadBMP loader;
saveBMP grabador;

// ---- Obtención de bordes binarios --- //
lti::splitImageToRGB rgb;
lti::channel8 R, G, B, edges;
lti::image theimage;

// Cargar imagen
loader.load( "prueba.bmp", theimage );

// obtenemos los canales RGB
rgb.apply(theimage, R, G, B);
lti::susanEdges susanedge;
lti::susanEdges::parameters edgeparams;

// establecemos umbral binarización
edgeparams.threshold = 10;
susanedge.setParameters(edgeparams);

// extraemos imagen binaria de canal rojo
susanedge.apply(R, edges);
// grabamos imagen binaria en archivo
grabador.save("susan_edges.bmp",edges);

// ---- Obtención del salience map --- //
lti::edgeSaliency sal;
lti::channel salmap;
lti::edgeSaliency::parameters salparam;

// Establecemos los parámetros de obtención del salience map
salparam.iterations = 20;
salparam.couplingCenter = 0.2;
salparam.couplingNeighbour = 0.0;
salparam.rho = 0.6;
salparam.initialSigma = 1.0;
salparam.gamma = 0.5;
salparam.curvationTable=lti::edgeSaliency::parameters::biased;
sal.setParameters(salparam);

// Lanzamos el algoritmo Ammon Sha'ashua y Shimon Ullmann
sal.apply(edges, salmap);
// grabamos el archivo de salida
grabador.save("salience_map.bmp",salmap);
```

Ilustración 14: Llamada al algoritmo de Curvature Salience

Los resultados del método son los siguientes:

Ilustración 15: Aplicación del algoritmo de bodes de Susan y de Curvature Salience iterando 20 veces

Podemos apreciar cómo el *salience map* resalta zonas de la imagen de particular interés (prominentes). El resultado es en principio bueno, si bien el método no nos sirve para crear regiones por el método de frontera, ni tampoco nos separa claramente los jeroglíficos. No obstante sería bueno utilizar este método en nuestro trabajo de cara a obtener unos bordes más precisos y eliminar el excesivo ruido. Pero aun así sigue existiendo ruido procedente de roturas de la piedra.

El principal inconveniente de este método y por el que se ha descartado, es que es muy lento. Realizar las 20 iteraciones requieren más de 5 minutos de computación, lo cual puede no ser un inconveniente si el objetivo es dejar a un ordenador trabajando en el problema, pero si lo puede ser si el objetivo es buscar una solución rápida y móvil. Sin embargo, queda claro que este método es apropiado en un futuro, donde posiblemente la computación sea más rápida.

3.10. ALGORITMO DE CONTORNOS DEFORMABLES

El algoritmo de contornos deformables de Chunming Li utiliza distintas variables, como son la continuidad del contorno, la suavidad del contorno y la atracción del borde, para conseguir obtener los contornos de los elementos de una imagen. Es una técnica de segmentación muy interesante que se podría pensar en utilizar para localizar la posición de los jeroglíficos. Sin embargo, mientras los resultados

ofrecían un excelente resultado detectando el borde del cartucho, no se comportaban correctamente detectando los jeroglíficos. El algoritmo se base en minimizar la siguiente ecuación:

$$\min E = \sum_{i=1}^{N} (\alpha_i E_{cont} + \beta_i E_{curv} + \gamma_i E_{imag}) \qquad \alpha_i, \beta_i, \gamma_i \geq 0$$

Donde la continuidad del contorno es:

$$E_{cont} = \left\| p_i - p_{i-1} \right\|^2 \quad \text{con} \quad p_i = (x_i, y_i)$$

La suavidad del contorno es:

$$E_{curv} = \left\| p_{i-1} - 2p_i + p_{i+1} \right\|^2$$

Y la atracción del borde es:

$$E_{imag} = -\left\| \nabla I \right\|^2$$

El algoritmo de los contornos deformables sería:

1. Definir el contorno inicial con un número de puntos N e inicializar $\alpha_i, \beta_i, \gamma_i$.

2. Mientras el número de puntos que se mueven a una nueva localización sea mayor que un determinado umbral T_1 realizar el paso 3.

3. Desde i = 0 hasta N (el punto N es el primero y el último procesado).

   i. Fijar $E_{\min}$ a un valor elevado.

   ii. Para todos los puntos j en un entorno de vecindad del punto p_i de dimensión m x m obtener:

   $$E_j = \alpha_i E_{cont,j} + \beta_i E_{curv,j} + \gamma_i E_{imag,j}$$

   iii. Si $E_j < E_{\min}$ entonces $E_{\min} = E_j$ y $j_{\min} = j$.

   iv. Mover el punto p_i a la localización $j_{\min}$ y contabilizar este movimiento de posición.

4. Determinar las curvaturas para la siguiente iteración:

   i. Desde i = 0 hasta N-1 obtener c_i.

   ii. Si $c_i > c_{i-1}$ y $c_i > c_{i+1}$ (si la curvatura es mayor que la de los vecinos) y $c_i > T_2$ (y la curvatura es mayor que un determinado umbral) y la magnitud del gradiente en $p_i > T_3$ (la fuerza del borde supera un umbral) entonces fijar β_i a 0 (eliminar la contribución de la curvatura en el cómputo de la energía para la siguiente iteración).

$$c_i = \left\| \frac{u_i}{|u_i|} - \frac{u_{i+1}}{|u_{i+1}|} \right\| \qquad \text{Donde,} \quad \begin{cases} u_i = (x_i - x_{i-1}, y_i - y_{i-1}) \\ u_{i+1} = (x_{i+1} - x_i, y_{i+1} - y_i) \end{cases}$$

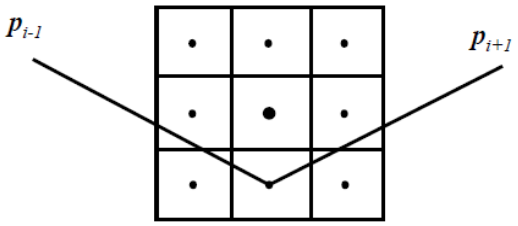


Ilustración 16: Valores de los puntos en el algoritmo de Chunming Li

A continuación vamos a ver los resultados que se producen con diversas imágenes al ejecutar el algoritmo en Matlab. El algoritmo es bastante lento, si bien una solución es reducir la dimensión de la imagen de entrada. En el siguiente ejemplo vemos cómo el algoritmo nos devuelve el borde del cartucho de un jeroglífico. Imaginemos que la imagen de entrada en escala de grises la convertimos a una anchura de 300 píxeles. El resultado de la búsqueda del cartucho en la iteración inicial y en la iteración 350, que es cuando hemos visto que se ha alcanzado un resultado que no varía posteriormente es:

Ilustración 17: Imagen de entrada del algoritmo de los contornos deformables

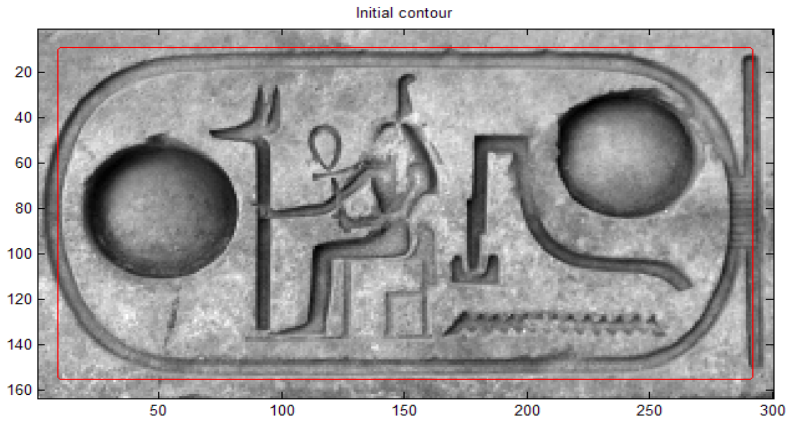


Ilustración 18: Contorno inicial detectado por el algoritmo de contornos deformables

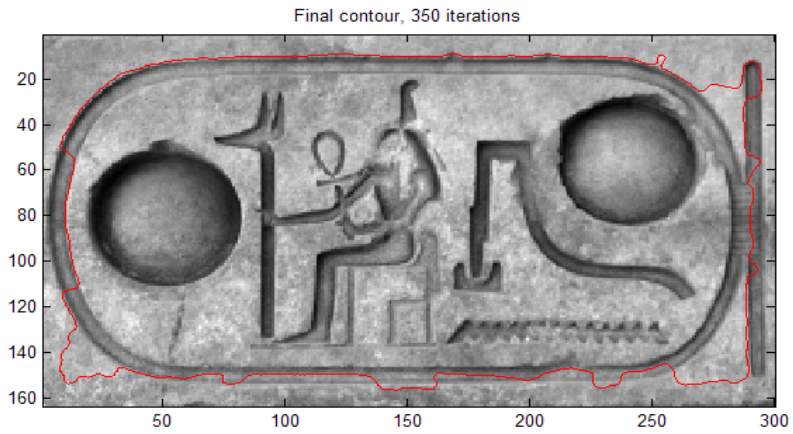
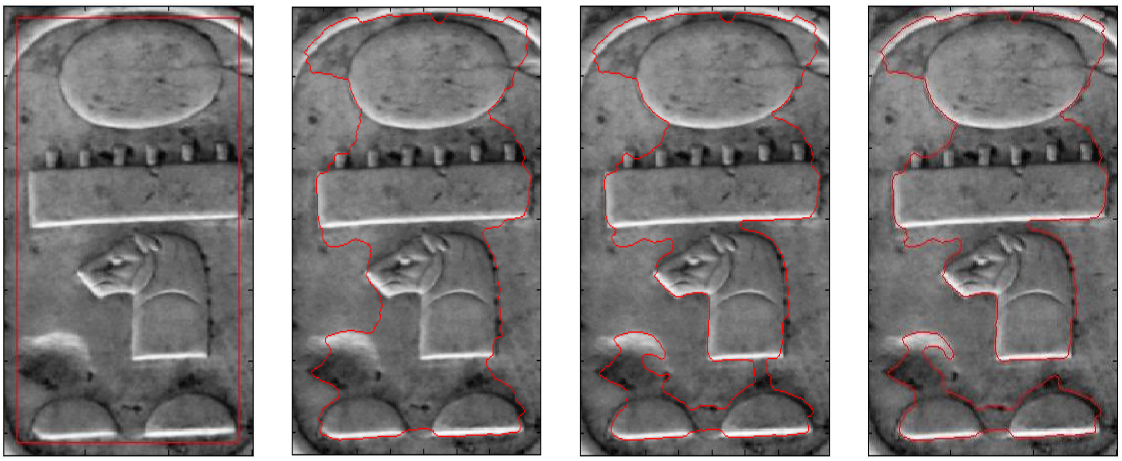


Ilustración 19: Obtención del borde del cartucho después de 350 iteraciones

En los siguientes dos ejemplos mostraremos cómo se comporta el algoritmo en la extracción de los elementos de un cartucho. Como podemos apreciar no es capaz de separar algunos jeroglíficos. Si bien existen técnicas que podrían dividir una región en dos en función de la concentración de la masa de los puntos, no sería aconsejable en este caso ya que algunos jeroglíficos pueden tener formas variadas. Al igual que en el caso anterior, como el algoritmo es lento ha sido necesario reducir el tamaño de la imagen a un ancho de 150 píxeles. Se ha utilizado una imagen en escala de grises.

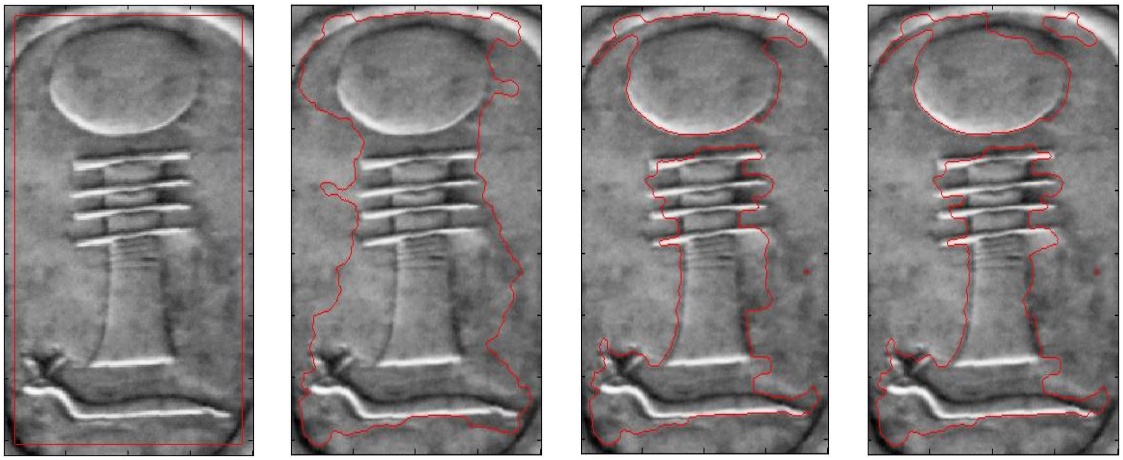


Contorno inicial del algoritmo	Contorno después de 300 iteraciones	Contorno después de 1.000 iteraciones	Contorno después de 10.000 iteraciones

Ilustración 20: Segmentación de jeroglíficos por el algoritmo de contornos deformables de Chunming Li

Como observamos en el primer ejemplo, a partir de un número de iteraciones el resultado varía poco. Entre la iteración 1000 y la 10000 vemos que el jeroglífico de abajo se ha separado, aunque

prácticamente no existen cambios en el resto de contornos. En el siguiente ejemplo vamos a ver los resultados para el mismo número de iteraciones, si bien la imagen tiene en este caso 200 píxeles de ancho.



Contorno inicial del algoritmo | Contorno después de 300 iteraciones | Contorno después de 1.000 iteraciones | Contorno después de 10.000 iteraciones

Ilustración 21: Segundo ejemplo de segmentación por el algoritmo de contornos deformables

En este segundo ejemplo podemos ver cómo los elementos devueltos después de 10.000 iteraciones son dispares. Por un lado arriba aparecen 2 elementos cuando sólo hay un jeroglífico de interés, y abajo aparecen dos jeroglíficos dentro de un mismo elemento. El algoritmo tiende a converger hacia unos contornos concretos.

3.11. MOMENTOS DE HU

Los momentos de una imagen son valores calculados medios sobre las intensidades de los píxeles, o sobre una función aplicada a dichos momentos de cara a obtener una interpretación correcta. Los momentos son útiles para describir regiones obtenidas después de la segmentación de la imagen. Algunos momentos clásicos son el área de una región, o el centroide o punto central de una región.

Los momentos de Hu tienen la particularidad de que se comportan de una forma invariante a la translación, escala o rotación. Esto es muy útil, ya que segmentando elementos o regiones podemos averiguar de cuál se trata independientemente de su escala, posición o rotación.

Los momentos son útiles en reconocimiento óptico de caracteres (OCR), utilizándose tanto momentos de Hu, momentos invariantes afines, o momentos invariantes de Tsirikolias-Mertzios. Todos estos momentos son invariantes a traslación, rotación, y al cambio de escala.

La idea inicial del trabajo pasaba por realizar una segmentación buscando jeroglíficos para posteriormente averiguar de cuál se trataba utilizando momentos de Hu. Sin embargo dicho modo no pudo realizarse correctamente debido a la dificultad de extracción de los distintos jeroglíficos, como se ha visto en el apartado de extracción de regiones por frontera.

Los momentos de Hu de una imagen en grises con píxeles de intensidad $I(x,y)$ son calculados de la siguiente manera, donde i y j representan el orden del cálculo:

$$M_{ij} = \sum_x \sum_y x^i y^j I(x,y)$$

El área de la región coincidiría con M_{00}, mientras que el centroide se definiría por las siguientes expresiones:

$$\bar{X} = \frac{M_{10}}{M_{00}} \qquad \bar{Y} = \frac{M_{01}}{M_{00}}$$

Los momentos centrales se calculan mediante la expresión:

$$\mu_{pq} = \sum_x \sum_y (x - \bar{X})^p (y - \bar{Y})^q f(x,y)$$

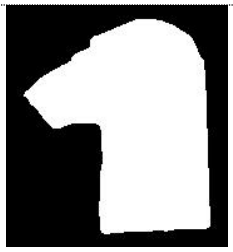
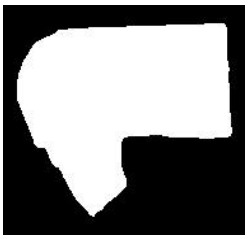
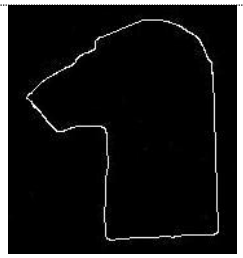
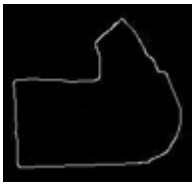
Siendo en el caso de orden p=3, q=3:

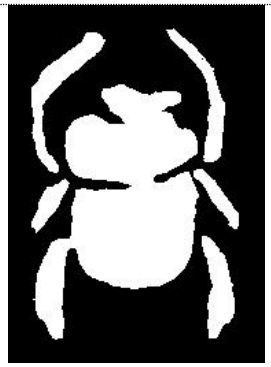
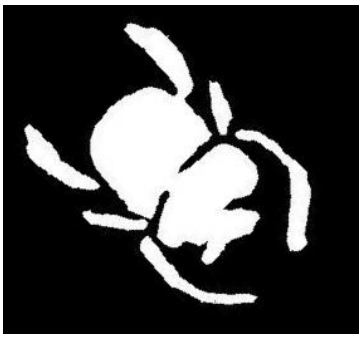
$$\mu_{00} = M_{00}$$
$$\mu_{01} = 0$$
$$\mu_{10} = 0$$
$$\mu_{11} = M_{11} - \bar{X} \cdot M_{01} = M_{11} - \bar{Y} \cdot M_{10}$$
$$\mu_{20} = M_{20} - \bar{X} \cdot M_{10}$$
$$\mu_{02} = M_{02} - \bar{Y} \cdot M_{01}$$
$$\mu_{21} = M_{21} - 2 \cdot \bar{X} \cdot M_{11} - \bar{Y} \cdot M_{20} + 2 \cdot \bar{X}^2 \cdot M_{01}$$
$$\mu_{12} = M_{12} - 2 \cdot \bar{Y} \cdot M_{11} - \bar{X} \cdot M_{02} + 2 \cdot \bar{Y}^2 \cdot M_{10}$$
$$\mu_{30} = M_{30} - 3 \cdot \bar{X} \cdot M_{20} + 2 \cdot \bar{X}^2 \cdot M_{10}$$
$$\mu_{03} = M_{03} - 3 \cdot \bar{Y} \cdot M_{02} + 2 \cdot \bar{Y}^2 \cdot M_{01}$$

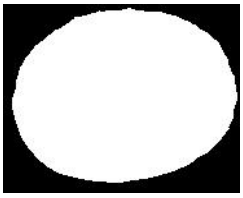
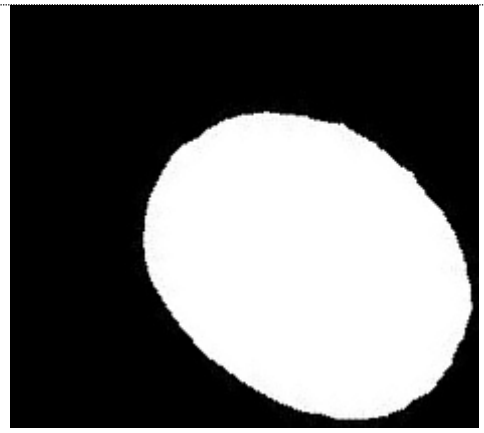
Finalmente los momentos invariantes a traslación rotación y escala se calcularían para el caso anterior mediante las siguientes 7 expresiones:

$$I_1 = \eta_{20} + \eta_{02}$$
$$I_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2$$
$$I_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2$$
$$I_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2$$
$$I_5 = (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})\left[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2\right] + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})\left[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2\right]$$
$$I_6 = (\eta_{20} - \eta_{02})\left[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2\right] + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03})$$
$$I_7 = (3\eta_{21} - 3\eta_{03})(\eta_{30} + \eta_{12})\left[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2\right] - (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})\left[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2\right]$$

Los momentos de Hu proporcionan unos resultados muy buenos si somos capaces de extraer regiones completas que representen a cada jeroglífico, y dicho modelo no cambie demasiado entre cartuchos. Una ventaja es que serían insensibles a rotación, traslación y escala, si bien como vimos en el método de extracción de regiones por frontera es bastante complicado extraer los jeroglíficos íntegramente. Vamos a ver unos ejemplos de cómo se comportarían los momentos de Hu ante jeroglíficos representados mediante regiones binarias.

	
Altura = 190 Anchura = 177 Momento 1 = 0.19025443205219675 Momento 2 = 3.73132380728284 Momento 3 = 13.214012779121056 Momento 4 = 3.086960419220151 Momento 5 = -1.3161645455376652 Momento 6 = -0.0015748022236414146 Momento 7 = -0.9843737755807177	Altura = 177 Anchura = 190 Momento 1 = 0.19025443205219655 Momento 2 = 3.5302616232998134 Momento 3 = 13.21401277912108 Momento 4 = 0.8667090519485426 Momento 5 = -1.3161645455380075 Momento 6 = -0.0015748022236553444 Momento 7 = 0.9843737755811693 Rotación -90º
	
Altura = 190 Anchura = 177 Momento 1 = 8.504429836273193 Momento 2 = 4.389151213256418 Momento 3 = 45778.15834719773 Momento 4 = 16552.50098618956 Momento 5 = 2687850.593410682 Momento 6 = 4847.6734030400585 Momento 7 = -3.714023043681744E7	Altura = 88 Anchura = 95 Momento 1 = 3.316674483156725 Momento 2 = 0.258382280536926 Momento 3 = 1016.1989230547558 Momento 4 = 3.7979737161994382 Momento 5 = -9363.393254757617 Momento 6 = -11.601197520850276 Momento 7 = 8733.826119905749 Rotación 90º / Reducción Escala 50% **(Con contornos no rellenos no se comporta bien)**

	
Altura = 257 Anchura = 233 Momento 1 = 0.2538329143781923 Momento 2 = 3.6514532135038933 Momento 3 = 39.60537007960968 Momento 4 = 6.33365448069493 Momento 5 = 113.15215309662051 Momento 6 = -0.19573101791765557 Momento 7 = -201.21172198496777	Altura = 233 Anchura = 257 Momento 1 = 0.25383291437818434 Momento 2 = 3.3694131207149156 Momento 3 = 39.60537007961322 Momento 4 = 7.308773981198991 Momento 5 = 113.1521530966657 Momento 6 = -0.19573101791777603 Momento 7 = 201.2117219849074 Rotación 90º
	
Altura = 262 Anchura = 187 Momento 1 = 0.251985450979049 Momento 2 = 3.6361411950180287 Momento 3 = 11.321277865616693 Momento 4 = 2.4140597947557203 Momento 5 = 0.16990404864730294 Momento 6 = 0.01132965073154552 Momento 7 = 0.4739257306855415	Altura = 262 Anchura = 283 Momento 1 = 0.24623410283278135 Momento 2 = 3.509408771261847 Momento 3 = 10.238115369591975 Momento 4 = 0.25518050254096963 Momento 5 = 0.08071650994502065 Momento 6 = 0.007374454883846629 Momento 7 = 0.14674873984322828 Rotación 135º

	
Altura = 148 Anchura = 183 Momento 1 = 0.16431153381063782 Momento 2 = 3.603522520862988 Momento 3 = 0.07501823827415635 Momento 4 = 0.0055003821996701865 Momento 5 = -5.2850509170606875E-6 Momento 6 = -2.6100109156224808E-5 Momento 7 = -5.394190282834056E-6	Altura = 211 Anchura = 233 Momento 1 = 0.16426280879403543 Momento 2 = 3.660765104556512 Momento 3 = 0.07363879325221541 Momento 4 = 0.008028309347146082 Momento 5 = -4.597304767246065E-6 Momento 6 = -2.3404967747490313E-5 Momento 7 = 7.936135149357285E-6 Rotación 45º / Traslación (30,30)
	
Altura = 250 Anchura = 112 Momento 1 = 0.4040064022256796 Momento 2 = 3.859905077874854 Momento 3 = 474.41477647538574 Momento 4 = 410.256864684472 Momento 5 = 144289.94995368744 Momento 6 = 117.2468218126589 Momento 7 = 30780.375662307175	Altura = 125 Anchura = 56 Momento 1 = 0.4028960647477308 Momento 2 = 3.860528866402779 Momento 3 = 115.0256430532626 Momento 4 = 98.75441853014384 Momento 5 = 8358.917740218212 Momento 6 = 28.084629716646617 Momento 7 = 1773.790746303395 Rotación 180º / Reducción Escala 50%

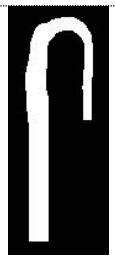
	
Altura = 203 Anchura = 82 Momento 1 = 0.824706582854388 Momento 2 = 3.7789128217377135 Momento 3 = 269.4153865094688 Momento 4 = 125.67761002073489 Momento 5 = 13885.652629600912 Momento 6 = 61.091133655992586 Momento 7 = 3193.962922799748	Altura = 101 Anchura = 82 Momento 1 = 0.5057227231383701 Momento 2 = 3.5345762937850647 Momento 3 = 67.19108132928402 Momento 4 = 13.080839351862913 Momento 5 = -147.82807124470742 Momento 6 = -2.017776063743595 Momento 7 = 58.08231335138312 Achatar mediante Shrink (Y=2, X=1) en ImageJ **(Si se modifica la escala de un solo eje los resultados no son buenos)**

Ilustración 22: Los momentos de Hu de diferentes jeroglíficos

3.12. MODELOS FLEXIBLES DE VISIÓN POR COMPUTADOR

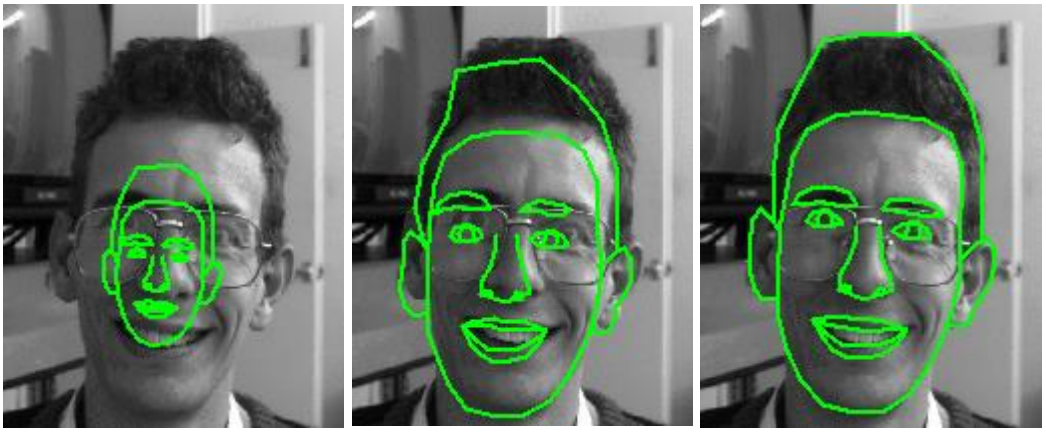
Los modelos flexibles de visión por computador tienen el objetivo de intentar crear un modelo formado por distintos puntos y vectores de objetos sobre un conjunto de imágenes, y buscar dicho modelo en nuevas imágenes con el fin de encontrar el objeto. Los puntos característicos de los modelos se suelen dar de alta manualmente, y el sistema genera el modelo en base a múltiples imágenes.

Estos sistemas se utilizan en distintos reconocimientos de objetos, utilizándose también en la identificación de rasgos faciales. Los procesos de búsqueda iterativamente van buscando el objeto buscado. Algunos de estos modelos son:

- Modelos estadísticos de formas, *Statistical Shape Models (SSM).*
- Modelos activos de formas, *Active Shape Models (ASM).*
- Modelos combinados de apariencia, *Combined Appearance Models (CAM).*
- Modelos activos de apariencia, *Active Appearance Models (AAM).*
- Modelos basados en visualización de apariencia, *View-Based Appearance Models (VAM).*
- Modelos de seguimiento basados en visualización de apariencia, *Tracking with View-Based Apperarance Models (TVAM).*

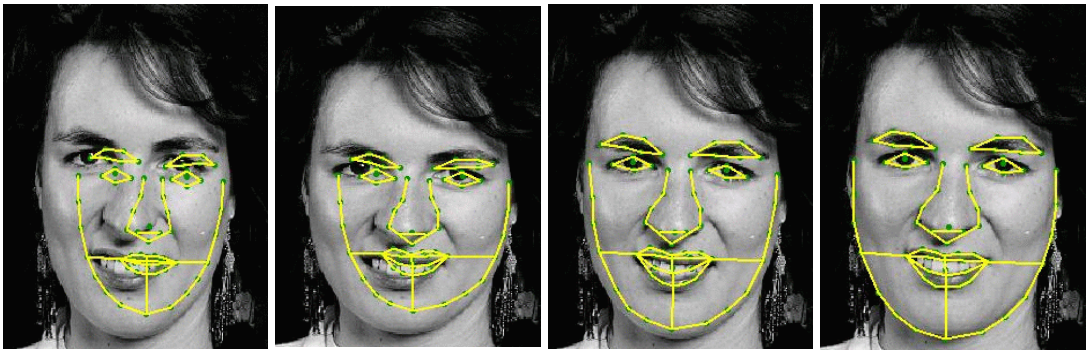
La Universidad de Manchester ofrece distintos programas de prueba y explicaciones sobre los distintos modelos:

http://www.isbe.man.ac.uk/~bim//Models/index.html



Posición inicial | Después de 5 iteraciones | Convergencia

Ilustración 23: Búsqueda de cara en modelo activo de formas (ASM)



Posición inicial | Después de 1 iteración | Después de 2 iteraciones | Convergencia

Ilustración 24: Búsqueda de cara en modelo activo de apariencia (AAM)

Estos modelos son capaces de representar la variabilidad, así que utilizan modelos deformables. Se construyen a partir del análisis estadístico de la estructura interna de un objeto cuyo contorno está marcado por puntos, estos puntos reciben el nombre de *landmarks*.

Como ejemplo vamos a comentar el modelo AAM, *Active Appearance Model*, o modelo de apariencia activo, que es una generalización del modelo ASM, *Active Shape Model* o modelo activo de formas. Utiliza toda la información que se encuentra en la región de la imagen objetivo en vez de sólo los bordes.

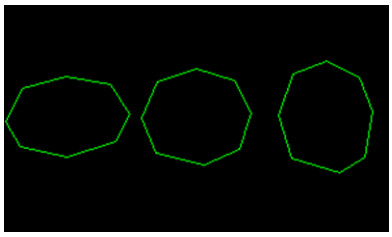
AAM utiliza un modelo estadístico de la forma y la apariencia del nivel de gris de los objetos de interés que puede generalizarse a casi todos los casos. Para ello empareja un modelo con una imagen

encontrando los parámetros que minimizan la diferencia entre dicha imagen y el modelo sintetizado proyectado sobre dicha imagen. Un problema que aparece es el excesivo número de parámetros para resolver el modelo. Existe tanto una fase de entrenamiento como de búsqueda. En la fase de búsqueda se mueve el modelo con el fin de adaptarse a los *landmarks* de la imagen buscada.

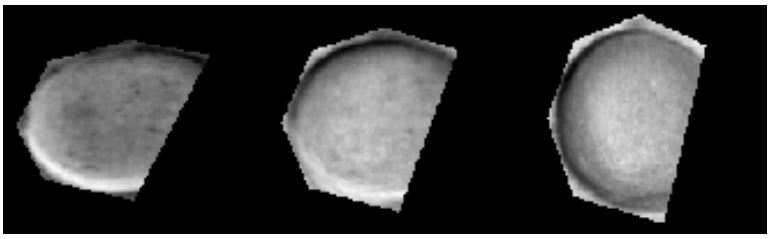
Vamos a ver cómo establecemos los *landmarks* en el caso de la búsqueda del jeroglífico RA dentro de un cartucho. Para crear nuestro modelo definiremos los puntos en 4 imágenes de jeroglíficos diferentes:

Ilustración 25: Estableciendo los *landmarks* del modelo (AAM)

Los valores dados de alta nos permitirán general el modelo de forma AAM, así como el modelo de textura. Los modelos obtenidos serían los siguientes:



Modelo de formas AAM



Modelo de apariencia AAM (Texturas)

Ilustración 26: Modelo AAM del cartucho RA

El algoritmo utilizado en el sistema AAM sería:

- Se define un número determinado de puntos sobre el contorno del objeto que los interesa para determinar la forma.
- Para obtener las texturas se muestrea una imagen usando una función de ajuste previa (*warping function*) como *piece-wise affine*, o *thin-plate*, y se separan las texturas mediante un esquema de triangulación.
- Se busca un espacio normalizado, alineando para ello las formas eliminando la posición, escalado y orientación.
- Se normaliza la imagen para quitar los efectos de la iluminación, buscando un contraste y brillo estándar para todas las imágenes.
- El análisis estadístico para obtener el modelo AAM se hace mediante PCA, utilizando tanto la forma como la textura. Se obtiene un vector para cada cara del conjunto de entrenamiento.
    - Para la forma, la adquisición de los datos es directa ya que los *landmarks* de la forma son los propios datos recogidos manualmente al introducir la imagen de la cara.
    - Para la textura se necesita un método que consiste en coger la información de textura que hay entre los *landmarks.* Para ello se dividen las imágenes en zonas iguales que nos permitan comparar los valores de textura en todo el conjunto de entrenamiento. Cada una de las formas de cada imagen es ajustada a una forma de referencia, que es normalmente la forma media, y luego es muestreada. Con ello se consigue un trozo de textura con el mismo número de puntos que puede ser procesada. Los valores se normalizan y se procesa la textura PCA.

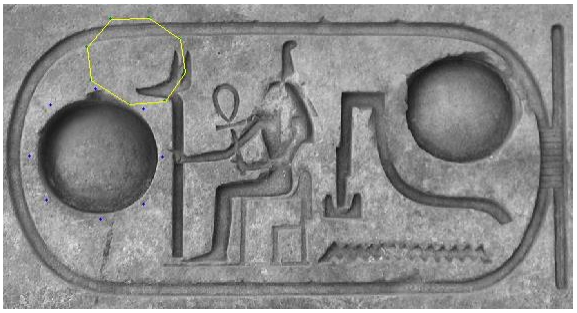
Los métodos PCA (*Principal Components Analysis*) y LDA (*Linear Discriminant Analysis*) son dos métodos ampliamente utilizados en el reconocimiento de objetos y facial. El funcionamiento de PCA y LDA se basa en transformar la imagen [m x n] píxeles en un vector de la forma [m · n]. Dicho vector es muy grande, por lo que se utilizan técnicas de compresión generando auto-vectores eficientes computacionalmente.

El análisis PCA deriva de la Transformada de Karhunen-Loewe (KL). Dada una representación vectorial s-dimensional de cada cara en un conjunto de imágenes de entrenamiento, el análisis PCA busca encontrar un sub-espacio t dimensional (t<s) cuya base de vectores corresponde a las direcciones de máxima varianza del espacio de la imagen original. Este nuevo sub-espacio posee normalmente dimensión inferior (t <<s). La base de vectores del PCA es definida como auto-vectores de la matriz de covarianza. Un auto-vector es un vector cuya dimensión es la misma que las imágenes iniciales y por lo tanto se puede ver como una imagen en el llamado espacio imagen.

Los primeros auto-vectores representan las direcciones de mayor varianza en la región de los objetos. Cada valor de un auto-vector es una coordenada en el espacio imagen, representando la contribución de ese dato a la varianza de la región de objetos en esa dirección. Un valor grande de esa cifra representa una contribución importante. Como las imágenes se muestran en blanco y negro, valores muy próximos al blanco o al negro representan contribuciones significativas de esas magnitudes

Los resultados que se producen en las búsquedas muestran que el algoritmo se comporta bien en entornos locales del objeto, pero no lo hace bien si la búsqueda se hace sobre zonas alejadas del objeto buscado. Sería necesario realizar una búsqueda por el espacio de la imagen. Así por ejemplo, si partimos de la siguiente imagen donde el inicio de búsqueda se muestra en el elemento amarillo, vemos que no es capaz de localizar el jeroglífico:

Elemento buscado lejos de objeto (amarillo)



Elemento encontrado (amarillo)

Ilustración 27: Primera búsqueda mediante AAM

Sin embargo, si posicionamos el elemento a buscar próximo al objeto buscado los resultados son correctos, aunque debido al ruido y diferencias de textura de los jeroglíficos en algunos casos ha producido resultados incorrectos.

Los resultados posicionando el elemento de búsqueda próximo al objeto buscado son los siguientes:

Elemento buscado próximo a objeto (amarillo)

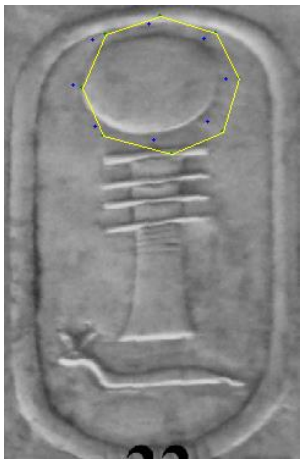
Elemento encontrado (amarillo)

Elemento buscado próximo a objeto (amarillo)

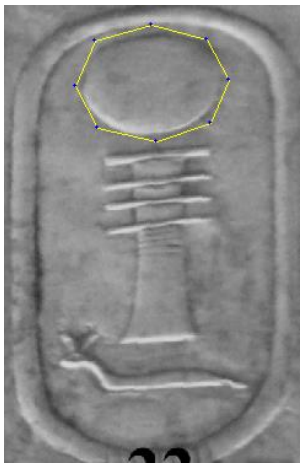
Elemento encontrado (amarillo)

Ilustración 28: Segunda búsqueda mediante AAM

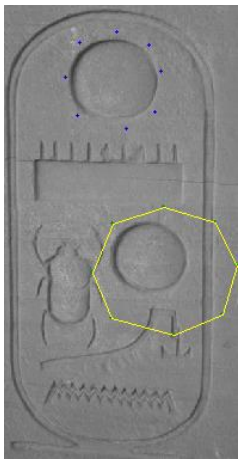
En las siguientes imágenes mostramos las pruebas realizadas sobre otros jeroglíficos, buscando el elemento en el entorno local del objeto buscado:



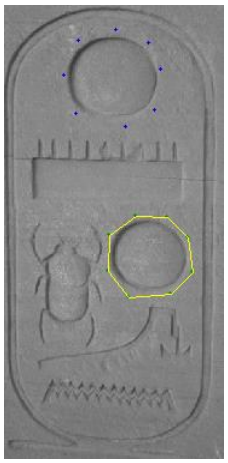
Elemento buscado próximo a objeto (amarillo)



Elemento encontrado (amarillo)



Elemento buscado próximo a objeto (amarillo)



Elemento encontrado (amarillo)

Ilustración 29: Tercera y cuarta búsqueda mediante AAM

El método no es el ideal en el problema de reconocimiento de jeroglíficos, ya que no se comporta bien si el entorno no es local, y además devuelve resultados en zonas incorrectas de la imagen. Ante texturas distintas de diferentes imágenes, o texturas con excesivo ruido no ofrece buenos resultados.

3.13. CÁLCULO DEL ÁNGULO DE UNA REGIÓN

Un procedimiento interesante utilizado en el método de búsqueda de jeroglíficos es el cálculo del ángulo de una región o grupo de píxeles. Dentro de nuestro trabajo aplicaremos el método sobre una ventana de 3X3, buscando el ángulo del píxel central. Este método nos servirá para comparar puntos del jeroglífico y ver si el ángulo es similar entre el punto que buscamos y el que estamos comprobando. Las fórmulas de cálculo del ángulo de una región son las siguientes:

$A =$ número de pixeles de región

$S_x = \sum x_i =$ Sumatorio de posiciones en X con píxeles activos

$S_y = \sum y_i =$ Sumatorio de posiciones en Y con píxeles activos

$S_{xx} = \sum x_i^2 =$ Sumatorio cuadrado de posiciones en X con píxeles activos

$S_{yy} = \sum y_i^2 =$ Sumatorio cuadrado de posiciones en Y con píxeles activos

$S_{xy} = \sum x_i y_i =$ Sumatorio de posiciones en X por posiciones en Y con píxeles activos

$$M_{xx} = S_{xx} - \frac{S_x^2}{A}$$

$$M_{yy} = S_{yy} - \frac{S_y^2}{A}$$

$$M_{xy} = S_{xy} - \frac{S_x \cdot S_y}{A}$$

$$\phi = \tan^{-1}\left[\frac{M_{xx} - M_{yy} + \sqrt{(M_{xx} - M_{yy})^2 + 4 \cdot M_{xy}^2}}{2 \cdot M_{xy}}\right]$$

A modo de ejemplo vamos a ver cómo se calcularía el ángulo sobre la siguiente región.

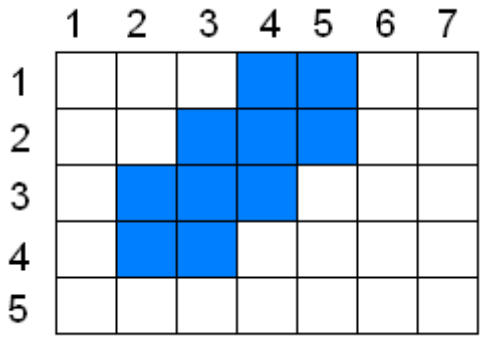


Ilustración 30: Región binaria para calcular orientación

Aplicando las fórmulas de cálculo del ángulo que tiene una región tenemos:

$$A = \text{número de pixeles de región } = 10$$

$$S_x = \sum x_i = 2+2+3+3+3+4+4+4+5+5 = 35$$
$$S_y = \sum y_i = 1+1+2+2+2+3+3+3+4+4 = 25$$
$$S_{xx} = \sum x_i^2 = 2^2+2^2+3^2+3^2+3^2+4^2+4^2+4^2+5^2+5^2 = 133$$
$$S_{yy} = \sum y_i^2 = 1^2+1^2+2^2+2^2+2^2+3^2+3^2+3^2+4^2+4^2 = 73$$
$$S_{xy} = \sum x_i y_i = 2\cdot 3+2\cdot 4+3\cdot 2+3\cdot 3+3\cdot 4+4\cdot 1+4\cdot 2+4\cdot 3+5\cdot 1+5\cdot 2 = 80$$

$$M_{xx} = S_{xx} - \frac{S_x^2}{A} = 133 - \frac{35^2}{10} = 10,5$$
$$M_{yy} = S_{yy} - \frac{S_y^2}{A} = 73 - \frac{25^2}{10} = 10,5$$
$$M_{xy} = S_{xy} - \frac{S_x \cdot S_y}{A} = 80 - \frac{35\cdot 25}{10} = -7,5$$

$$\phi = \tan^{-1}\left[\frac{M_{xx} - M_{yy} + \sqrt{(M_{xx} - M_{yy})^2 + 4\cdot M_{xy}^2}}{2\cdot M_{xy}}\right]$$

$$\phi = \tan^{-1}\left[\frac{10,5 - 10,5 + \sqrt{(10,5-10,5)^2 + 4\cdot(-7,5)^2}}{2\cdot(-7,5)}\right]$$

$$\phi = \tan^{-1}\left[\frac{\sqrt{4\cdot(-7,5)^2}}{2\cdot(-7,5)}\right] = \tan^{-1}\left[\frac{\sqrt{4\cdot(-7,5)^2}}{2\cdot(-7,5)}\right]$$

$$\phi = \tan^{-1}[-1] = -45^o \ \ (-\frac{\pi}{4})$$

Por lo tanto, la región tiene una orientación de -45º dentro de la imagen. Intuitivamente vemos que era una solución que se podía pensar, ya que la región vemos que está inclinada unos 45º, formando una especie de guión o línea corta.

3.14. DISTANCIA DE LEVENSHTEIN

Se llama Distancia de Levenshtein, distancia de edición, o distancia entre palabras, al número mínimo de operaciones requeridas para transformar una cadena de caracteres en otra. Se entiende por operación, bien una inserción, eliminación o la sustitución de un carácter. Esta distancia recibe ese nombre en honor al científico ruso Vladimir Levenshtein, quien estudió este método en 1965. Es útil en programas que determinan cuán similares son dos cadenas de caracteres, como es el caso de los correctores de ortografía.

En este trabajo cada jeroglífico se corresponderá con un carácter de la cadena. Así por ejemplo, la distancia entre RA-MN-PHTY-T-T (Ramsés I), y RA-MN-HPR (Tutmosis III) será 3, ya que haremos lo siguiente:

- RA-MN-PHTY-T-T → RA-MN-PHTY-T (Eliminación de 1 elemento)
- RA-MN-PHTY-T → RA-MN-PHTY (Eliminación de 1 elemento)
- RA-MN-PHTY → RA-MN-HPR (Substitución de PHTY por HPR).

El siguiente código muestra el algoritmo en JAVA, tal cual se ha utilizado en el trabajo. Aunque se ha utilizado este método para comparar la cadena obtenida con las cadenas de la base de conocimiento, otros métodos podrían responder mejor ante situaciones que queramos forzar. Por ejemplo, si detectamos que una cadena reconocida pertenece a un faraón concreto, y la distancia de Levenshtein nos devuelve otro resultado, podríamos optar por usar una red neuronal, y forzarla mediante entrenamiento para que ante determinada entrada devuelva una salida concreta.

```
public class LevenshteinDistance {
   private static int minimum(int a, int b, int c) {
       if(a<=b && a<=c) return a;
       if(b<=a && b<=c) return b;
       return c;
   }

   public static int computeLevenshteinDistance(String str1, String str2) {
       return computeLevenshteinDistance(str1.toCharArray(),
                                         str2.toCharArray());
   }

   private static int computeLevenshteinDistance(char [] str1,
           char [] str2) {
       int [][]distance = new int[str1.length+1][str2.length+1];

       for(int i=0;i<=str1.length;i++) distance[i][0]=i;
       for(int j=0;j<=str2.length;j++) distance[0][j]=j;
       for(int i=1;i<=str1.length;i++)
       {
           for(int j=1;j<=str2.length;j++)
           {
                 distance[i][j]= minimum(distance[i-1][j]+1,
                                         distance[i][j-1]+1,
                                         distance[i-1][j-1]+
                                         ((str1[i-1]==str2[j-1])?0:1));
           }
       }
       return distance[str1.length][str2.length];
   }
}
```

Ilustración 31: Algoritmo de la Distancia de Levenshtein escrito en JAVA

3.15. TRANSFORMADA DE HOUGH

La transformada de Hough es un método útil de búsqueda de rectas, círculos y elipses en el espacio de la imagen. En este trabajo la principal utilidad será la rotación del cartucho a una posición recta.

La transformada de Hough fue propuesta por Paul Hough. Inicialmente se aplicaba a la detección de rectas en una imagen, pero más tarde se extendió para identificar cualquier figura que se pudiera describir con unos cuantos parámetros.

Las rectas se pueden representar con la ecuación $y = m \cdot x + n$. En la transformada de Hough, la idea principal es considerar las características de una recta en término de sus parámetros (m,n), y no como puntos de la imagen $(x_1, y_1),...,(x_n, y_n)$. Basándose en lo anterior, la recta $y = m \cdot x + n$ se puede representar como un punto (m,n) en el espacio de parámetros. Cuando las rectas son verticales estos parámetros no están definidos, por lo que en el algoritmo que implementa la transformada de Hough se utilizan coordenadas polares, denotadas (ρ, θ).

El parámetro ρ representa la distancia entre el origen de coordenadas y el punto (x, y), mientras que θ es el ángulo del vector director de la recta perpendicular a la recta original y que pasa por el origen de coordenadas.

Usando estas coordenadas, la ecuación de una recta se puede escribir de la forma $\rho = x \cdot \cos(\theta) + y \cdot \sin(\theta)$. Es posible asociar a cada recta un par (ρ, θ) que es único si $\theta \in [0, \pi)$ y $\rho \in R,\ \theta \in [0, 2 \cdot \pi)$ y $\rho \geq 0$. El espacio (ρ, θ) se denomina espacio de Hough para el conjunto de rectas en dos dimensiones.

Para un punto arbitrario en la imagen con coordenadas (x_0, y_0), las rectas que pasan por ese punto son los pares (ρ, θ) con $r = x \cdot \cos(\theta) + y \cdot \sin(\theta)$ donde ρ (la distancia entre la línea y el origen) está determinado por θ.

El algoritmo para implementar la transformada de Hough de obtención de rectas sería el siguiente:

```
1: Cargar imagen
2: Detectar los bordes en la imagen (por ejemplo por Canny)
3: Por cada punto en la imagen:

  3.1:   Si el punto (x, y) está en un borde:

    3.1.1:   Por todos los posibles ángulos θ:

      3.1.1.1:     Calcular ρ para el punto (x, y) con un ángulo θ.

      3.1.1.2:     Incrementar la posición (ρ, θ) en el acumulador
4: Buscar las posiciones con los mayores valores en el acumulador
5: Devolver las rectas cuyos valores son los mayores en el acumulador.
```

Ilustración 32: Algoritmo de implementación de la transformada de Hough

Desde el punto de vista de su aplicación en este trabajo de investigación, la transformada de Hough nos permite conocer la orientación de un cartucho de una manera sencilla, y rotarlo para poder tenerlo en posición recta de manera que los podamos reconocer sin necesidad de girar los elementos a buscar.

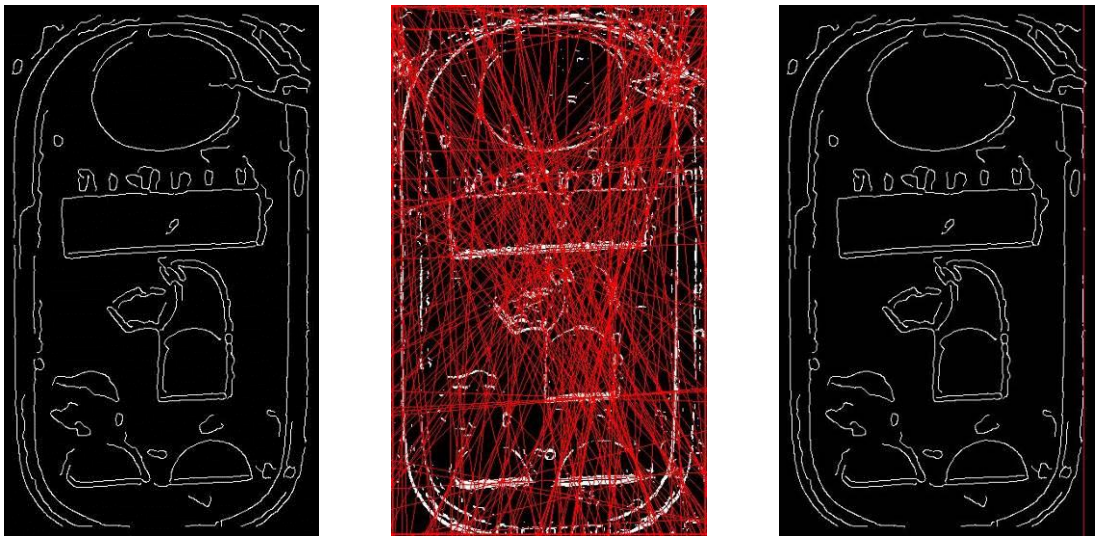


Imagen original (Bordes Canny) | Transformada de Hough para Threshold = 30 (Contador > 30) | Transformada de Hough para Threshold = MAX(Contadores)-1

Ilustración 33: Aplicación de la transformada de Hough para detectar la orientación del cartucho

En las tres imágenes anteriores podemos ver cómo se comporta la transformada de Hough. Si utilizamos un umbral bajo nos devolverá muchas posibles rectas que existen en la imagen. Si cogemos el máximo valor calculado de los contadores y utilizamos dicho valor menos uno como umbral, nos devolverá la recta más probable. En un cartucho esta línea debería coincidir con el borde más alto del cartucho. Podría darse el caso de que se nos devolvieran dos líneas. En dichos casos se podría coger cualquiera.

3.16. REDES NEURONALES

Uno de los mecanismos habituales de reconocimiento de imágenes es el uso de redes neuronales de retro-propagación. El principal objetivo de la red neuronal de reconocimiento de imágenes es clasificar un vector en una clase minimizando el posible error mediante un entrenamiento supervisado.

El algoritmo tiene dos fases. Durante la primera fase, una vez que se ha aplicado un patrón a la entrada de la red como estímulo, este se propaga desde la primera capa a través de las capas superiores de la red, hasta generar una salida. La señal de salida se compara con la salida deseada y se calcula una señal de error para cada una de las salidas.

Durante la segunda fase, las salidas de error se propagan hacia atrás, partiendo de la capa de salida, hacia todas las neuronas de la capa oculta que contribuyen directamente a la salida. Sin embargo las neuronas de la capa oculta solo reciben una fracción de la señal total del error, basándose aproximadamente en la contribución relativa que haya aportado cada neurona a la salida original. Este proceso se repite, capa por capa, hasta que todas las neuronas de la red hayan recibido una señal de error que describa su contribución relativa al error total.

A medida que se entrena la red, las neuronas de las capas intermedias se organizan entre sí de tal modo que las distintas neuronas aprenden a reconocer distintas características del espacio total de entrada. Después del entrenamiento, cuando se les presente un patrón arbitrario de entrada que contenga ruido o que esté incompleto, las neuronas de la capa oculta de la red responderán con una salida activa si la nueva entrada contiene un patrón que se asemeje a aquella característica que las neuronas individuales hayan aprendido a reconocer durante su entrenamiento.

Existe un algoritmo implementado en Java que permite integrar fácilmente una red neuronal en la aplicación (Librerías Neuroph – Java Neural Network Framework). Además incluye una herramienta para poder crear la red neuronal y realizar el entrenamiento, de manera que desde el programa Java sólo sea necesario cargar los datos del entrenamiento previamente guardados. La dirección de las librerías Neuroph es la siguiente:

http://neuroph.sourceforge.net/image_recognition.html

La entrada de la red estará formada por los píxeles de la imagen. Para evitar manejar redes muy grandes las imágenes se pueden escalar. No obstante, es necesario que todas las imágenes tengan una misma resolución (por ejemplo 200 x 300 píxeles). La entrada de la red neuronal será un array unidimensional. Para rellenarlo es necesario coger de fila en fila de la imagen y ponerlas consecutivamente en el array de entrada, que representa las neuronas de la capa de entrada.

Para probar la técnica se ha creado una red neuronal con 400 neuronas (imágenes de 20x20 píxeles), una capa oculta de 74 neuronas y una capa de salida con 37 neuronas. Cada neurona de salida representa un jeroglífico concreto. La red se ha entrenado sobre una base de datos de 37 imágenes, representando cada una a un jeroglífico.

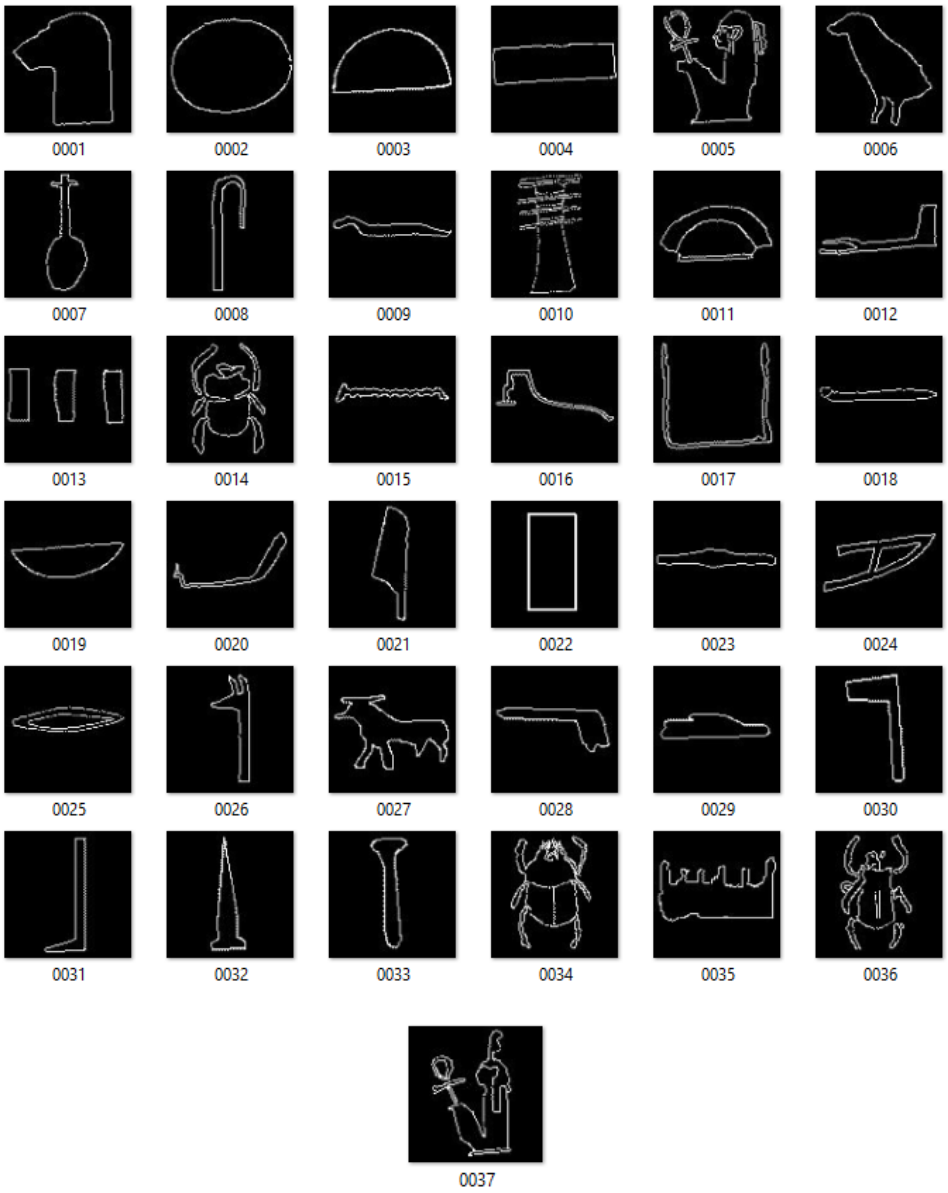


Ilustración 34: Imágenes para el entrenamiento de la red neuronal

Aunque idealmente se hubiera utilizado una capa oculta de 220 neuronas, correspondientes a la suma de elementos de entrada y salida entre dos, por motivos de rendimiento se ha utilizado una capa con menos neuronas. Se ha utilizado una función de activación sigmoidal. En la siguiente imagen vemos la representación de la red neuronal en Neuroph, así como el gráfico representando la reducción de error mediante el entrenamiento iterativo. En cada iteración se corrige el error de la red.

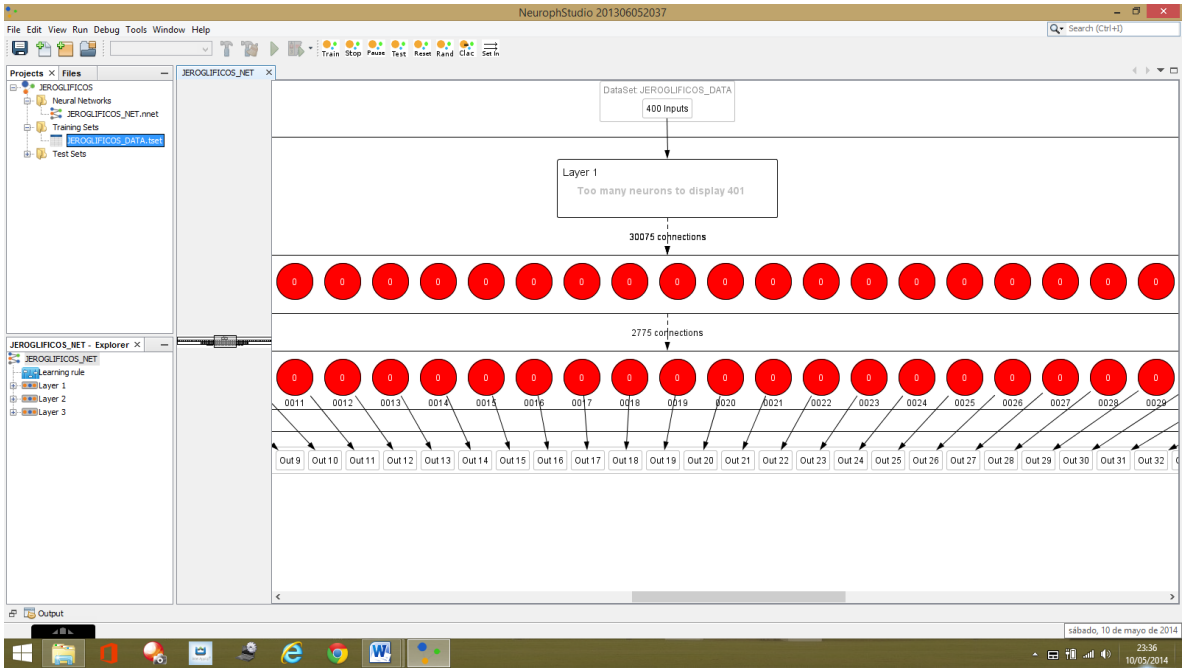


Ilustración 35: Red neuronal creada mediante la herramienta Neuroph

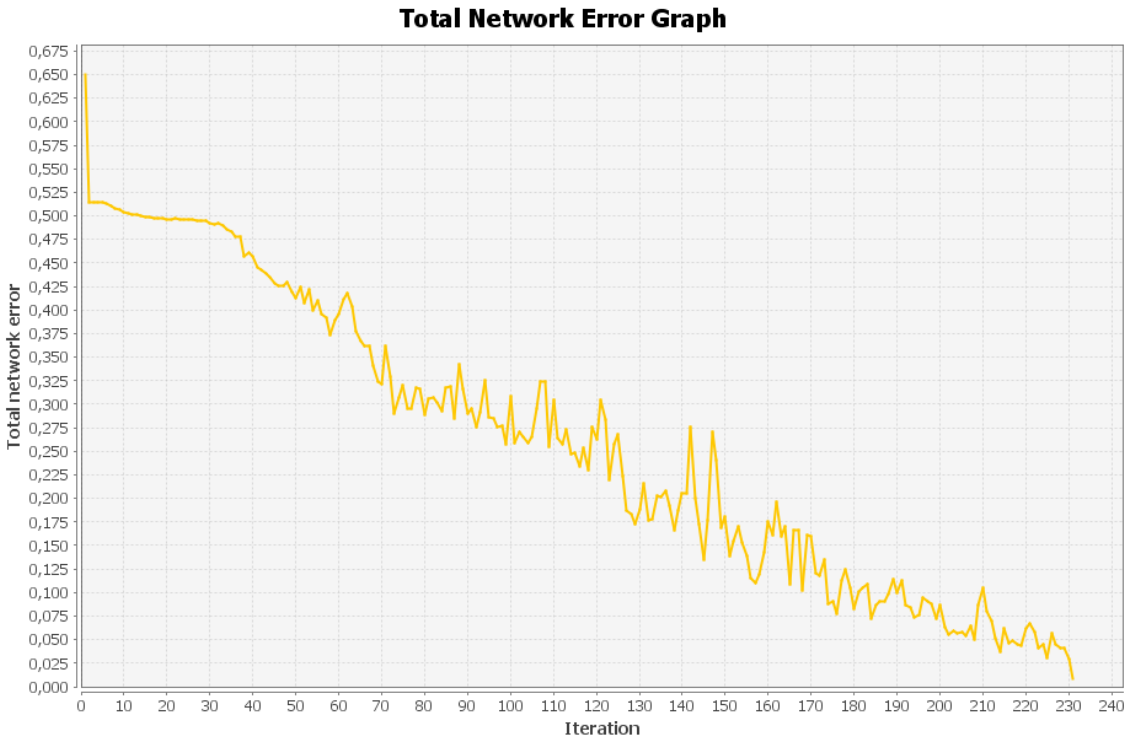


Ilustración 36: Entrenamiento de la red neuronal de búsqueda de jeroglíficos

Las pruebas sin embargo no han arrojado resultados muy buenos. Vemos a continuación distintas pruebas:

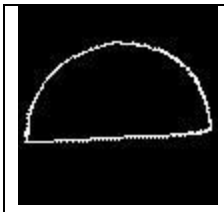
			
0001 : 0	0001 : 0,0014	**0001 : 0,0001**	0001 : 0
0002 : 0	0002 : 0	0002 : 0	0002 : 0
0003 : 0,9695	0003 : 0,0002	0003 : 0,0006	0003 : 0,0019
0004 : 0	0004 : 0	0004 : 0,0005	**0004 : 0,0636**
0005 : 0,0007	0005 : 0,0002	0005 : 0,0001	0005 : 0,0029
0006 : 0	0006 : 0	0006 : 0,0001	0006 : 0
0007 : 0,0001	0007 : 0	0007 : 0,0001	0007 : 0
0008 : 0,0716	0008 : 0,058	**0008 : 0,3113**	0008 : 0,0234
0009 : 0	0009 : 0	0009 : 0	0009 : 0
0010 : 0	0010 : 0,0001	0010 : 0,0002	0010 : 0,0001
0011 : 0	0011 : 0	0011 : 0	0011 : 0,0005
0012 : 0	0012 : 0,0036	0012 : 0,0002	0012 : 0
0013 : 0	0013 : 0	0013 : 0	0013 : 0
0014 : 0,0005	**0014 : 0,9918**	0014 : 0,0026	0014 : 0,0072
0015 : 0	0015 : 0	0015 : 0	0015 : 0
0016 : 0,0022	0016 : 0	0016 : 0,0002	0016 : 0,0127
0017 : 0,0001	0017 : 0	0017 : 0	0017 : 0
0018 : 0	0018 : 0,0001	0018 : 0	0018 : 0,0001
0019 : 0	0019 : 0,0001	0019 : 0	0019 : 0
0020 : 0,0002	0020 : 0	0020 : 0,0004	0020 : 0,0002
0021 : 0,0001	0021 : 0	0021 : 0,0001	0021 : 0,0001
0022 : 0,0002	0022 : 0,0007	0022 : 0,0035	0022 : 0
0023 : 0,0028	0023 : 0	0023 : 0,0001	0023 : 0,0001
0024 : 0	0024 : 0	0024 : 0	0024 : 0,0001
0025 : 0,0001	0025 : 0,0237	0025 : 0,0001	0025 : 0
0026 : 0	0026 : 0,0025	0026 : 0,0003	0026 : 0,0009
0027 : 0,0001	0027 : 0	0027 : 0	0027 : 0,0001
0028 : 0,0001	0028 : 0	0028 : 0	0028 : 0,0007
0029 : 0	0029 : 0,0002	0029 : 0,0018	0029 : 0,019
0030 : 0,0004	0030 : 0	0030 : 0	0030 : 0,0003
0031 : 0	0031 : 0	0031 : 0,0008	0031 : 0,0002
0032 : 0,0002	0032 : 0	0032 : 0	0032 : 0
0033 : 0,0006	0033 : 0	0033 : 0	0033 : 0,0001
0034 : 0	0034 : 0	0034 : 0,0001	0034 : 0
0035 : 0	0035 : 0	0035 : 0	0035 : 0,0007
0036 : 0	0036 : 0,0071	0036 : 0,0002	0036 : 0,0002
0037 : 0,0007	0037 : 0,0001	0037 : 0,0805	**0037 : 0,0013**

Ilustración 37: Test de distintas imágenes sobre la red neuronal

En cada ejemplo vemos en azul el jeroglífico que debería tener mayor probabilidad, al ser el que corresponde a la imagen. Las dos primeras imágenes venían del banco de pruebas utilizado para entrenar la red neuronal, por lo que el sistema las ha detectado correctamente. Las dos imágenes del final corresponden a imágenes reales extraídas del reconocimiento de cartuchos. Podemos ver que la red neuronal no se comporta correctamente, ya que los elementos más probables son los que están en rojo, y sin embargo los que deberían haber salido son los que están en azul.

Algunas conclusiones que se extraen de las pruebas sobre redes neuronales son:

- Para problemas de clasificación de imágenes basta con utilizar una capa oculta e intentar aproximar el número de neuronas de la capa a la media entre neuronas de entrada y neuronas de salida. No obstante, debido al gran número de neuronas de entrada de un problema de reconocimiento de imágenes se podrían ajustar las neuronas de la capa intermedia. Si se establece más de una capa oculta se puede utilizar la regla de la pirámide, intentando dibujar una pirámide entre neuronas de entrada (fila de debajo de la pirámide), y neuronas de arriba (punta de la pirámide). Las neuronas de las capas intermedias se irán reduciendo para mantener esta estructura piramidal.
- El reconocimiento de imágenes binarias mediante redes neuronales se comporta mejor con regiones. Tal como ocurre con los momentos de Hu, la contribución de todos los píxeles de una región produce mejores resultados. Ante bordes, como es este caso, el funcionamiento no es muy bueno. En el problema del reconocimiento de jeroglíficos, como lo que tenemos es el borde de estos, el sistema no funcionará bien.
- Un problema con el reconocimiento de imágenes es el excesivo coste computacional necesario para realizar el entrenamiento. En nuestro problema, para sólo 37 jeroglíficos, no se han podido utilizar las neuronas de la capa intermedia deseadas debido a limitaciones técnicas. Neuroph no admite demasiadas conexiones. Esto ocurre, ya que para imágenes de 20x20, se producen 400 neuronas de entrada, con conexiones con cada una de las neuronas intermedias, y a su vez conexiones con las neuronas de la capa de salida. Esto provoca un problema si por ejemplo se necesitara reconocer más jeroglíficos. Además, cada vez que se introdujera un nuevo jeroglífico sería necesario entrenar de nuevo toda la red.
- Finalmente vamos a comentar cómo se podría abordar el reconocimiento, en el caso de haber decidido el uso de redes neuronales. Como la red neuronal aceptaría imágenes de 100x100, escalando a 20x20, sería necesario ir reduciendo la imagen de entrada, y recorriéndola de arriba abajo para detectar jeroglíficos. Se establecería un umbral para considerar que se había devuelto un jeroglífico correcto. La red neuronal acepta imágenes de 100x100, por lo que introducir imágenes de diferentes tamaños daría lugar a resultados incongruentes. Sería por tanto necesario recorrer la imagen mediante una ventana de 100x100.

3.17. UTILIZACIÓN DE SOA

Este trabajo se enfoca principalmente en el reconocimiento de cartuchos desde dispositivos móviles como pueden ser teléfonos móviles o tabletas. Si bien estos dispositivos han alcanzado una gran potencia, es cierto que los procesos de reconocimiento que se utilizan en este trabajo requieren una capacidad de cálculo muy grande. Por este motivo se plantea la utilización de una arquitectura SOA simple, basada en un servicio web de reconocimiento.

Los dispositivos móviles realizarán una fotografía del cartucho a reconocer, harán algunas operaciones de suavizado o filtrado de la imagen, y la reducirán a un tamaño pequeño (por ejemplo un ancho de 100 píxeles) para transmitirla a un servicio web. Dicho servicio web será el que realmente realice el proceso de reconocimiento devolviendo el nombre del faraón o la dinastía.

El motivo de utilizar SOA es doblemente interesante. Por un lado evita cargar de proceso a los dispositivos móviles, y por otro lado, que además es sumamente importante, aísla el código principal del sistema de reconocimiento de los dispositivos centralizándolo en un punto. Esto permite realizar optimizaciones del código sin necesidad de recargar la aplicación móvil.

Como herramienta de desarrollo se ha seleccionado NetBeans, la cual mantiene una distribución libre y nos permite crear aplicaciones web utilizando servicios web basados en el modelo JAX-WS (Java API for XML Web Services). Se utilizará el servidor de aplicaciones web Apache Tomcat Server, aunque podrían utilizarse otros servidores como Oracle Weblogic, siendo este último comercial.

Dentro de los dispositivos móviles seleccionamos aquellos basados en Android, debido a su amplia extensión. NetBeans permite desarrollar aplicaciones Android.

Vemos a continuación el ejemplo de un servicio web y un cliente desarrollados en NetBeans. El servicio web recibe una imagen en un array de bytes como entrada, y devuelve su anchura y altura mediante un array de enteros. En este caso tanto el servicio web como el cliente se ejecutan desde un PC.

```
package es.uned.WebServidor;

import java.awt.image.BufferedImage;
import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.io.InputStream;
import javax.imageio.ImageIO;
import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.WebParam;

/**
 *
 * Servidor WEB
 */
```

```
@WebService(serviceName = "ParametrosImagen")
public class ParametrosImagen {

    /**
     * Web service operation
     */
    @WebMethod(operationName = "datosImagen")
    public java.lang.String[] datosImagen(@WebParam(name = "imagen")
           byte[] imagen)
        {
        int width = 0;
        int height = 0;
        try {
            // convierte array a BufferedImage para obtener anchura y
            // altura
            InputStream in = new ByteArrayInputStream(imagen);
            BufferedImage img = ImageIO.read(in);
            width = img.getWidth();
            height = img.getHeight();
            }
        catch (IOException ex)
        {
        System.out.println(ex.getMessage());
        }
        // devolvemos los datos como cadenas de texto
        String cadenas[] = new String[3];
        cadenas[1] = String.valueOf(width);
        cadenas[2] = String.valueOf(height);
        return cadenas;
    }
}
```

Ilustración 38: Servicio Web que devuelve la altura y anchura de una imagen

```
package webcliente;

import java.awt.image.BufferedImage;
import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.IOException;
import java.util.List;
import javax.imageio.ImageIO;

/**
 *
 * Cliente WEB
 */
public class WebCliente {

    public static void main(String[] args) {
        try {
            // Abrimos una imagen de prueba y la convertimos a un
            // array de bytes
            File file = new File(openFile.getPath());
            FileInputStream fis = new FileInputStream(file);
            BufferedInputStream inputStream = new BufferedInputStream(fis);
            byte[] imageInByte = new byte[(int) file.length()];
            inputStream.read(imageInByte);
            inputStream.close();

        // LLamamos al servicio web y sacamos por pantalla la altura y
        // anchura
            List<String> salida = datosImagen(imageInByte);
            System.out.println("Anchura: " + salida.get(1));
            System.out.println("Altura: " + salida.get(2));
            }
         catch (IOException ex)
```

```
                {
           System.out.println(ex.getMessage());
              }
        }

        private static java.util.List<java.lang.String> datosImagen(
                byte[] imagen)
            {
            es.uned.webservidor.ParametrosImagen_Service service =
               new es.uned.webservidor.ParametrosImagen_Service();
            es.uned.webservidor.ParametrosImagen port =
               service.getParametrosImagenPort();
            return port.datosImagen(imagen);
        }

    }
```

Ilustración 39: Cliente Web que llama al servicio web anterior

A continuación vemos el mismo cliente del servicio web anterior, pero esta vez desarrollado en Android. Se envía una imagen del dispositivo móvil al servicio web, y éste nos devuelve la anchura y altura de la imagen. Este tipo de cliente es necesario en el trabajo para poder enviar una imagen al servidor web de reconocimiento. Los servicios web en Android se apoyan sobre la tecnología KSOAP2.

```
    private static final String METHOD_NAME = "datosImagen";
    private static final String NAMESPACE = "http://localhost/";
    private static final String SOAP_ACTION = NAMESPACE + METHOD_NAME;
    private static final String URL =
           "http://192.168.0.1:8080/WebServidor/ParametrosImagen?wsdl";

    public void onCreate(Bundle savedInstanceState)
        {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        // cargamos imagen de un fichero
        Bitmap bitmap = BitmapFactory.decodeResource("/sdcard/prueba.jpg");

        // transformamos imagen en array de bytes
        ByteArrayOutputStream baos = new  ByteArrayOutputStream();
        bitmap.compress(Bitmap.CompressFormat.JPEG,100, baos);
        byte [] imageInByte = baos.toByteArray();

        try {
            // Creamos la petición de llamada al servicio web,
            //incluyendo valores de entrada
            SoapObject request = new SoapObject(NAMESPACE, METHOD_NAME);
            request.addProperty("imagen",imageInByte);

            // Creamos el sobre de llamada al servicio web
            SoapSerializationEnvelope envelope =
                    new SoapSerializationEnvelope(SoapEnvelope.VER11);
            envelope.dotNet = true;
            envelope.setOutputSoapObject(request);

            // Modelamos el transporte
            HttpTransportSE androidHttpTransport = new HttpTransportSE(URL);

            // Realizamos la llamada
            androidHttpTransport.call(SOAP_ACTION, envelope);

            // Obtenemos el resultado
            SoapObject result=(SoapObject)envelope.getResponse();
```

```
            String ancho =
                 ((SoapObject)result.getProperty(0)).getProperty(0).toString();
            String altura =
                 ((SoapObject)result.getProperty(1)).getProperty(0).toString();

            // Mostramos el resultado
            ((TextView) findViewById(R.id.campo_ancho)).setText("ANCHO = " +
                        ancho);
            ((TextView) findViewById(R.id.campo_altura)).setText("ALTURA = " +
                        altura);
            }
        catch (Exception E)
            {
            E.printStackTrace();
            ((TextView) findViewById(R.id.campo_avisos)).setText("ERROR:" +
                        E.getClass().getName() + ":" + E.getMessage());
            }
    }
```

Ilustración 40: Llamada al mismo servicio Web desde Android

4. Solución propuesta

4.1. Utilización de la herramienta ImageJ

En el trabajo se ha decidido utilizar las librerías del programa ImageJ para realizar distintas operaciones. Así por ejemplo, la conversión a escala de grises, la aplicación del filtro mediana, la obtención de bordes mediante el algoritmo de Canny, o la binarización basándose en el método IsoData han sido integradas a través de librerías de ImageJ o de plug-ins existentes. El plug-in que implementa el algoritmo de Canny a través del filtrado Canny-Deriche ha sido desarrollado por Thomas Boudier y Joris Meys. Su explicación está en el apartado relativo al algoritmo de Canny.

Uno de los problemas principales a la hora de realizar el reconocimiento consistía en obtener una imagen de bordes con el menor ruido posible y con la suficiente calidad para el reconocimiento.

Después de numerosas pruebas se comprobó que el aplicar un filtro mediana previo a la obtención de bordes reducía considerablemente el ruido. Así mismo la binarización a través del método IsoData era la que mejores resultados producía. A continuación veremos diferentes pruebas realizadas sobre tres cartuchos con un brillo y contraste diferente, y podremos comprobar visualmente cómo estos métodos seleccionados son los que se comportan mejor. Las pruebas se muestran con y sin filtro mediana.

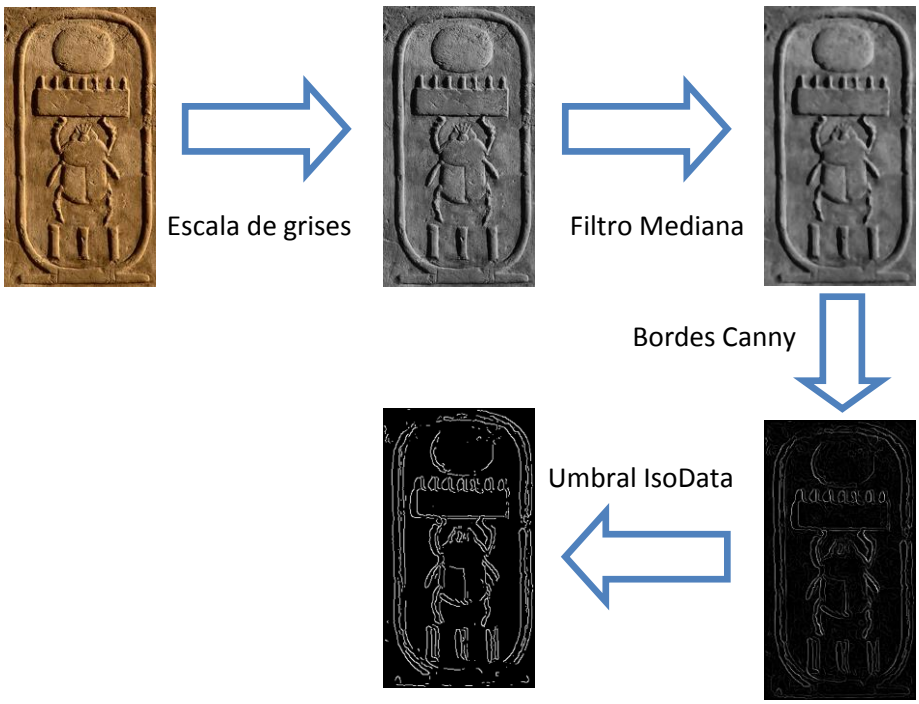


Ilustración 41: Proceso de extracción de bordes

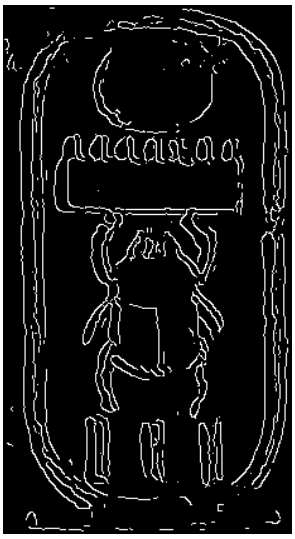

Original

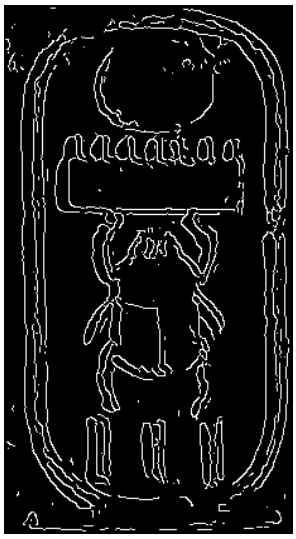

Escala de grises


Filtro mediana


Bordes Canny

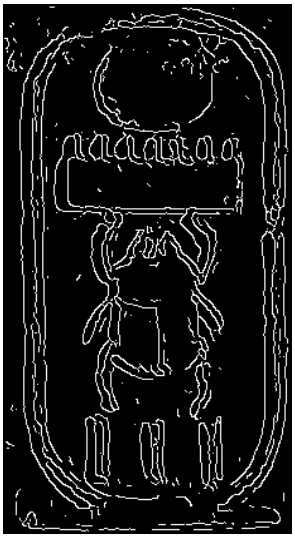

Default (143,88)


IsoData (114,26)


MaxEntropy (99,45)


Moments (133,30)


Otsu (114,26)


RenyiEntropy (84,64)


Yen (78,29)

Original

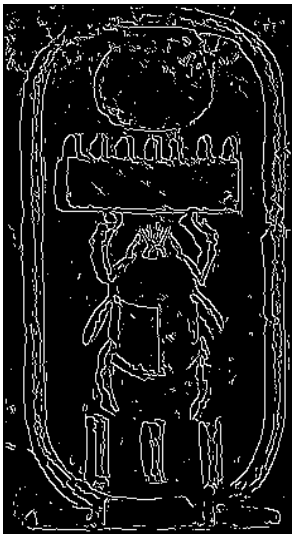
Escala de grises

Bordes Canny

Default (180,39)

IsoData (128,85)

MaxEntropy (171,02)



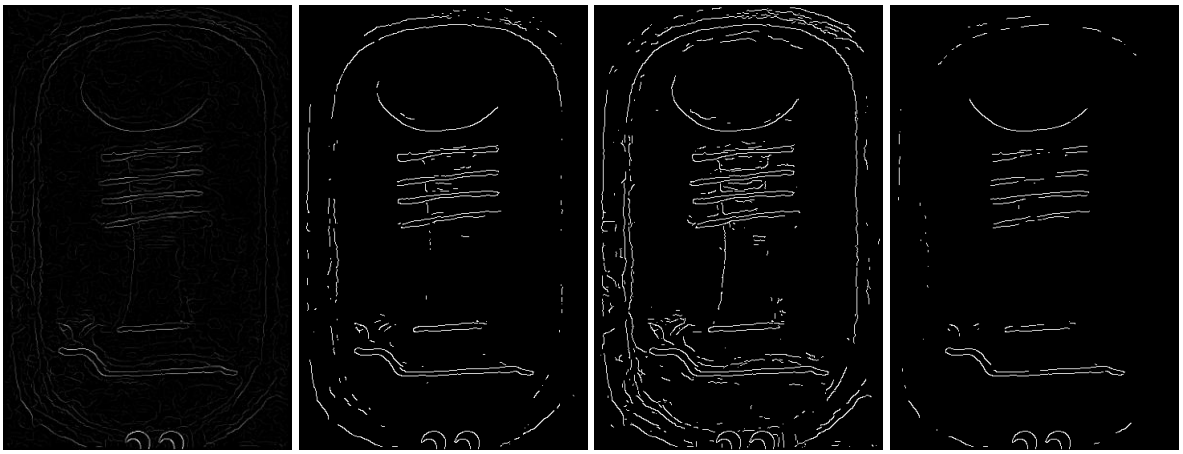
Moments (138,22)

Otsu (131,19)

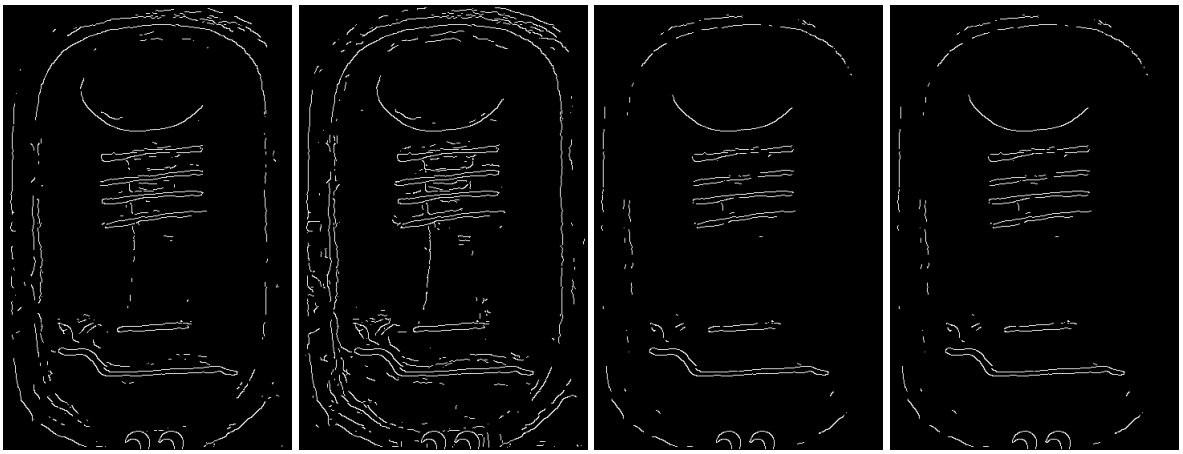
RenyiEntropy (142,91)

Yen (135,88)

Original Escala de grises Filtro mediana

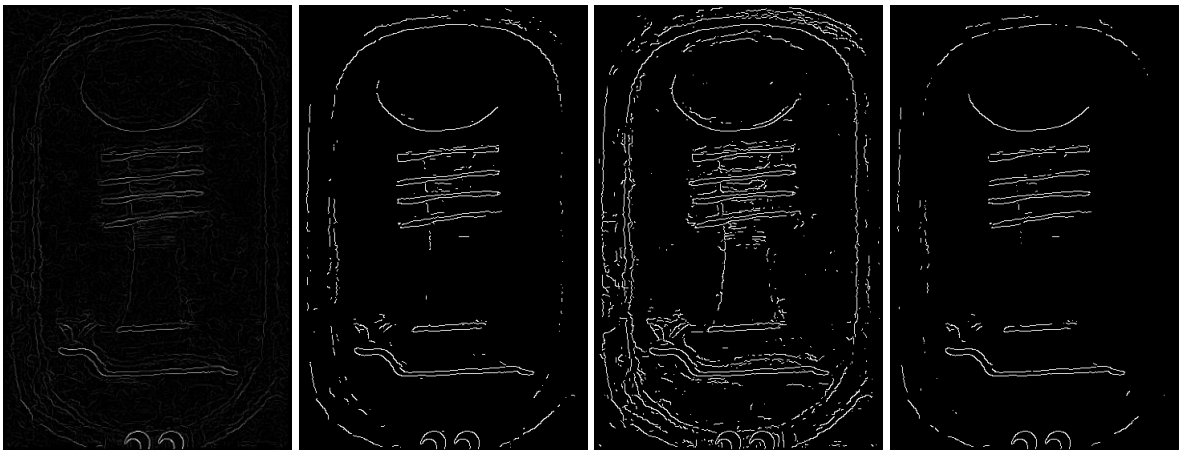


Bordes Canny Default (80,60) IsoData (52,70) MaxEntropy (114,70)

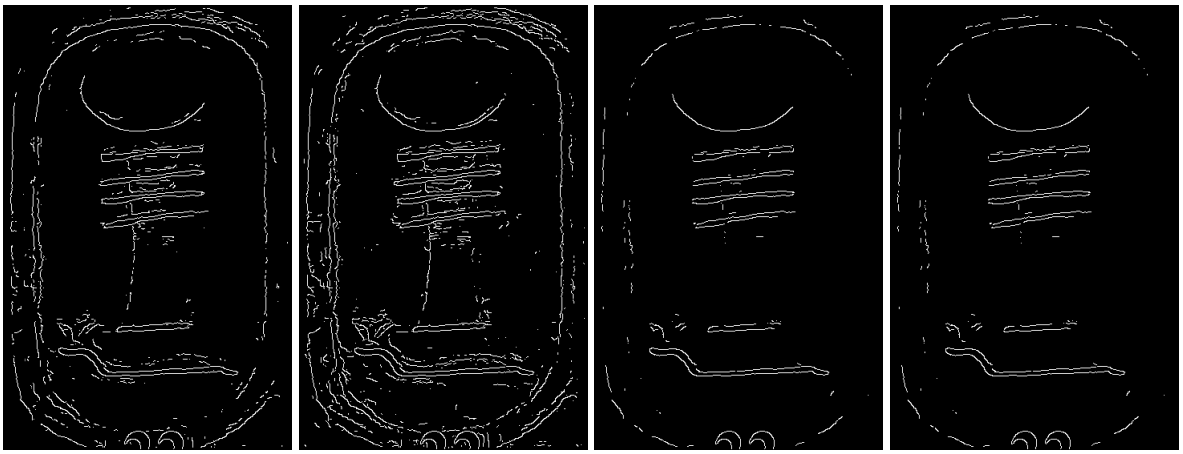


Moments (69,75) Otsu (52,70) RenyiEntropy (97,65) Yen (97,65)

Original Escala de grises

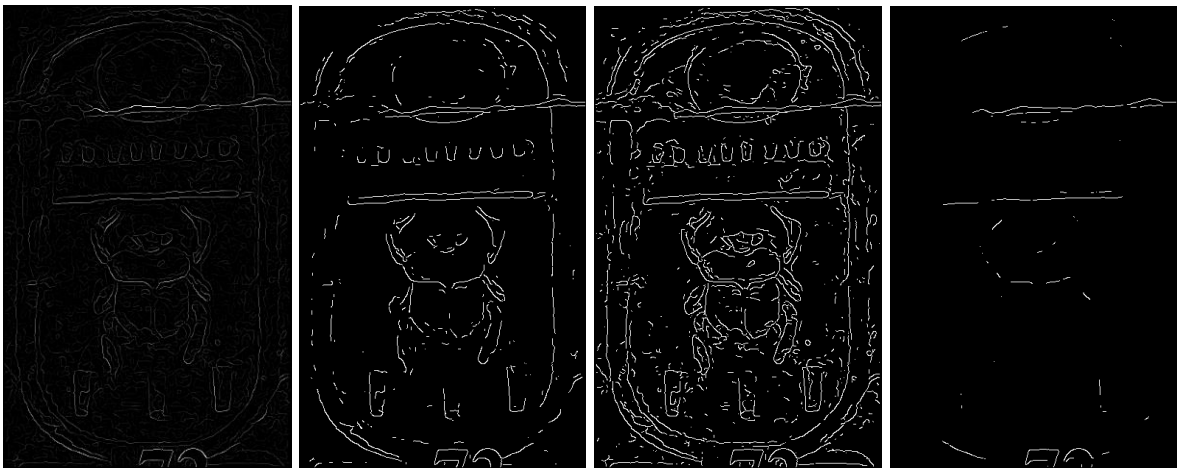


Bordes Canny Default (86,81) IsoData (51,44) MaxEntropy (104,49)



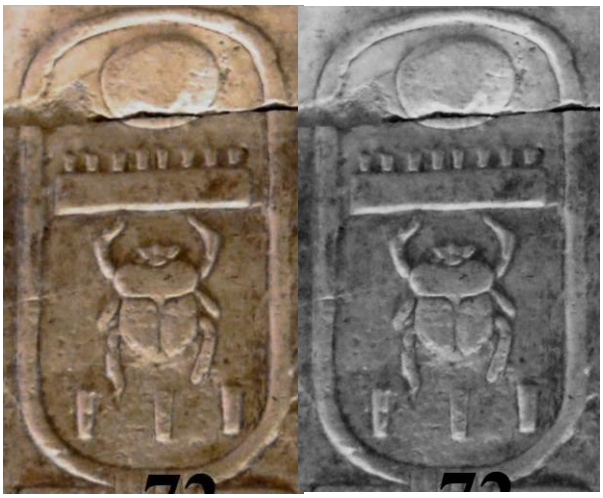
Moments (67,52) Otsu (53,05) RenyiEntropy (104,49) Yen (104,49)

Original Escala de grises Filtro mediana

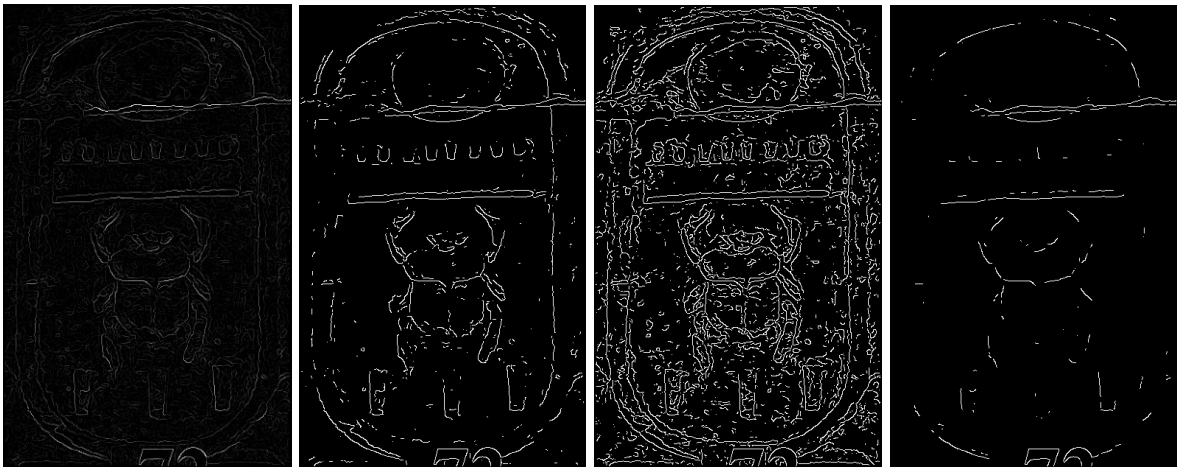


Bordes Canny Default (111,85) IsoData (65,42) MaxEntropy (230,03)

Moments (94,97) Otsu (65,42) RenyiEntropy (230,03) Yen (232,14)



Original Escala de grises



Bordes Canny Default (116,90) IsoData (60,62) MaxEntropy (197)

Moments (95,25) Otsu (62,78) RenyiEntropy (194,84) Yen (194,84)

Ilustración 42: Pruebas de búsqueda de mejor método de extracción de bordes

4.2. BÚSQUEDA POR EL ESPACIO DE LA IMAGEN

La solución de la búsqueda de jeroglíficos en el espacio de la imagen ha sido el método que finalmente se ha utilizado. Se comporta de manera parecida a como lo haría un humano. Si se mira un cartucho tratando de identificar a quién pertenece lo que intentamos es ver qué jeroglíficos conocemos en dicho cartucho. Normalmente trazamos con la mirada un barrido de la imagen tratando de buscar elementos conocidos, de acuerdo a elementos que hemos visto anteriormente.

El método utilizado será similar a nuestra forma de buscar los jeroglíficos, aunque con ciertas diferencias. Para poder implementarlo mediante software debemos utilizar una imagen que podamos tratar de una forma práctica y eficiente. Para ello utilizaremos la imagen binarizada devuelta por el algoritmo de Canny.

La búsqueda de los jeroglíficos se realizará buscando todos los posibles elementos conocidos, almacenados en una base de datos de conocimiento, en el espacio de la imagen. Como los jeroglíficos tienen múltiples tamaños, buscaremos los elementos con distintos tamaños.

Esta solución tiene distintos problemas, aunque afortunadamente existen soluciones:

- Es un método lento. La búsqueda de todos los jeroglíficos por la pantalla requiere de un elevado coste computacional. Existen distintas soluciones para este problema que han permitido ejecutar una búsqueda completa de 40 jeroglíficos en una imagen en menos de 15 segundos en un ordenador Core i3 con 6Gb de memoria. Con 40 jeroglíficos aproximadamente podemos leer todos los posibles cartuchos de Abydos, ya que los jeroglíficos al igual que los caracteres de nuestro alfabeto se repiten. Las soluciones propuestas son:

    - La primera solución consiste en trabajar con imágenes reducidas. En el trabajo se plantea utilizar imágenes de un eje menor limitado a 100 píxeles. De esta manera se acelerará mucho el proceso. Hay que implementar mecanismos de ajuste de tamaño tanto de la imagen de entrada como de los jeroglíficos.

    - La optimización de los procesos de búsqueda mediante la no creación de excesivas matrices, o evitando cálculos repetitivos puede acelerar los resultados.

    - La computación paralela es una importante mejora. La utilización de hilos de ejecución en Java frente a computación iterativa aumenta considerablemente el rendimiento. Además, otra posibilidad sería la ejecución de distintos hilos en distintos servidores. Esto se podría implementar de una forma sencilla mediante servicios web internos.

- Finalmente, y debido a que la finalidad del trabajo es el reconocimiento en dispositivos móviles, se plantea utilizar SOA (Software Oriented Architecture) desviando el proceso de reconocimiento a un servidor de aplicaciones web. Los dispositivos móviles simplemente capturarán la imagen a reconocer, harán pequeños ajustes como filtrado máximo o suavizado, y reducirán la imagen para transmitirla a un servicio web que devolverá la respuesta. Esta solución así mismo permite optimizar el sistema de reconocimiento en un único punto sin necesidad de modificar las aplicaciones descargadas en cada dispositivo móvil.

- Un jeroglífico puede cambiar entre distintos cartuchos. Efectivamente los jeroglíficos, al igual que nuestros caracteres, pueden cambiar de un cartucho a otro. Sin embargo, los escultores solían utilizar moldes similares, por lo que normalmente existe poca diferencia entre unos y otros. Aun así, se propone utilizar un método de búsqueda aproximada basada en un umbral de coincidencia T. El umbral utilizado es T=75%. Este método tiene en cuenta 3 factores:

  - Un píxel del jeroglífico buscado es igual a un píxel de la imagen de entrada si la distancia entre dicho punto y un punto activo en la imagen de entrada es menor que 3. En la siguiente imagen, suponiendo que el jeroglífico buscado es la línea naranja, y la imagen original es la línea azul, la distancia mínima del punto naranja P(3,2) será 2, ya que los píxeles azules P(1,3) y P(2,4) se encuentran a una distancia de 2 y no hay ningún píxel más próximo. Dicho punto sería seleccionado a priori. Sin embargo, en el punto naranja P(6,2) la distancia mínima será 3, ya que los punto azules más próximos serían P(3,5), P(4,5) y P(5,5). Dicho punto quedaría descartado. Este método se puede aplicar mediante una máscara de convolución 1x1, 3x3, 5x5, 7x7,etc sobre la imagen de entrada.

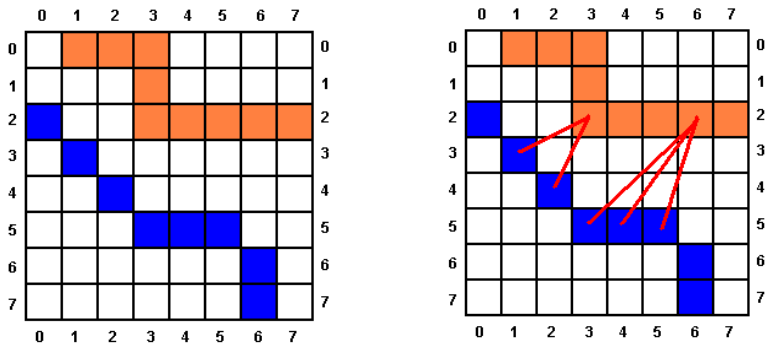


Ilustración 43: Distancia mínima ente un punto buscado y el punto posible

  - Un punto candidato del apartado anterior debe requerir además otra condición. El ángulo que forma dicho punto en una máscara 3x3 debe ser inferior a la expresión (100-T) x (PI/2) / 100. Esto garantiza que dos puntos que pertenecen a ángulos diferentes sean considerados distintos. En la imagen anterior, el punto naranja P(3,2) que a priori era candidato, será descartado. Como podemos ver el ángulo que forma es

totalmente diferente al ángulo que forman los dos candidatos P(1,3) y P(2,4), que son una línea recta. El método de cálculo de ángulos se explica en la siguiente sección.

- Una vez que se ha evaluado completamente la coincidencia de un elemento en la imagen, diremos que dicho elemento es un posible jeroglífico si el número total de píxeles considerados candidatos es superior al umbral T establecido del total de píxeles del elemento. Es decir, si un jeroglífico tiene 100 puntos activos, y hemos encontrado una figura en la imagen de entrada que comparte 80 puntos candidatos, si el umbral T=75, entonces lo consideraremos jeroglífico posible.

- Un jeroglífico posible diremos que es jeroglífico si el número de líneas continuas que lo forman es menor a otro umbral R. Este método final sirve para descartar jeroglíficos posibles en zonas con mucho ruido. En dichas zonas puede que se supere el umbral T fijado anteriormente de píxeles coincidentes. Sin embargo, comprobando que existen muchas líneas discontinuas lo descartaremos. El umbral utilizado es R=30.

- En la lista de cada tipo de jeroglíficos encontrados, si hay más de 5 posibles elementos de un determinado jeroglíficos, nos quedaremos con los que mayor probabilidad tienen. Para eso ordenaremos la lista de cada tipo de jeroglífico por su probabilidad.

- Una vez localizados todos los jeroglíficos comprobaremos si algún jeroglífico se encuentra dentro de otro mayor. Esto puede ocurrir si por ejemplo cierta parte de un jeroglífico coincide con un jeroglífico entero. En algunos jeroglíficos podemos encontrar también partes que corresponden a otros jeroglíficos. La solución simplemente consiste en rechazar el jeroglífico más pequeño. En estos casos nos quedaremos con el mayor, al considerar que si localizamos un jeroglífico mayor es más probable que sea correcto. Esto ocurrirá también cuando se encuentre una parte de un jeroglífico más pequeño en otro mayor. Consideraremos el 30% del área compartida como umbral para considerar si un jeroglífico se solapa con otro.

- Igualmente rechazaremos jeroglíficos similares localizados consecutivamente. Al establecer una localización basada en un umbral, es posible que en píxeles consecutivos localicemos el mismo jeroglífico. En estos casos nos quedaremos sólo con uno.

El método anterior devuelve distintos jeroglíficos localizados, y es posible que devuelva algún jeroglífico en zonas donde no debería encontrar, ya que puede confundirse con zonas donde falta parte del cartucho, o hay excesivo ruido, aunque esto no sería lo normal. Leyendo la lista de jeroglíficos de arriba abajo y de izquierda a derecha obtendremos una cadena de códigos.

El algoritmo de la distancia de Levenshtein nos devuelve la distancia mínima entre dos cadenas para conseguir que una cadena se transforme en otra. Dicho algoritmo se utiliza con la cadena de códigos

obtenida comparándola con las cadenas de códigos de cada cartucho de cada faraón. El cartucho con la menor distancia de Levenshtein será la salida del método. Dicho método se comenta más adelante.

Otra cuestión que surge es la posibilidad de confundir el cartucho en sí con jeroglíficos. Para evitar este problema y centrar el cartucho antes de realizar la búsqueda, es necesario localizar los límites del cartucho para poder excluirlo. Esto se consigue localizando las cuatro curvas que forman el propio cartucho. Al igual que con los jeroglíficos, podemos utilizar cuatro falsos-jeroglíficos que nos devuelvan las posiciones del cartucho.

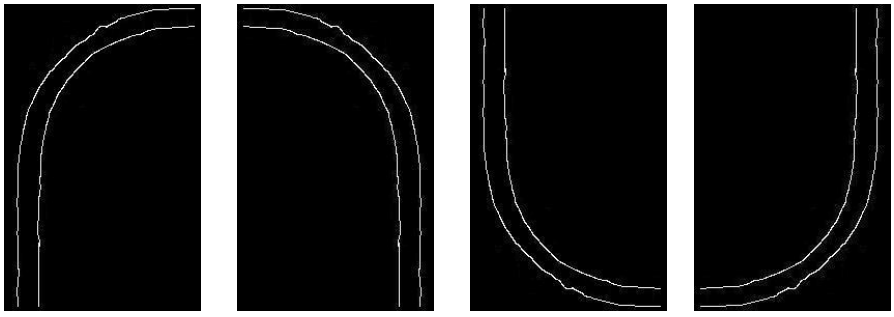


Ilustración 44: Elementos de búsqueda para localizar el cartucho

A continuación vamos a ver cómo obtenemos los contornos y centramos un cartucho en base a la localización de los bordes. A partir de la imagen inicial obtenemos la imagen en escala de grises. Aplicamos un filtro de mediana para eliminar ruido manteniendo contornos, y un filtro de máximos reforzando contornos. A partir de la imagen resultante obtenemos los bordes mediante el algoritmo de Canny. Binarizamos la imagen de bordes de Canny y realizamos la búsqueda de contornos. La imagen consideramos que se encuentra no inclinada. Si la imagen estuviera inclinada sería necesario localizar los contornos girándolos, y una vez localizados rotar la imagen. En las siguientes imágenes vemos que los bordes resultantes se encuentran en verde.

Ilustración 45: Obtención del borde del cartucho para un umbral T= 75 %

Podemos observar en la imagen cómo un borde ha sido localizado en un jeroglífico. Esto ha ocurrido ya que dicho jeroglífico tiene la forma de un borde en uno de sus laterales. Dicho error no afectará al proceso ya que una vez localizados los bordes nos quedaremos con el valor menor y mayor en

el eje X, y el menor y mayor en el eje Y. Dentro del método eliminaremos la parte de la imagen fuera de los siguientes límites:

minimo_x = minimo_x + 10 % (maximo_x-minimo_x)

maximo_x = maximo_x – 10 % (maximo_x-minimo_x)

minimo_y = minimo_y + 3 % (maximo_y-minimo_y)

maximo_y = maximo_y – 3 % (maximo_y-minimo_y)

Los valores anteriores son apropiados en los cartuchos de la lista de Abydos. Sin embargo podría ser necesario establecer otro criterio en otros cartuchos. Un método alternativo sería dibujar a partir de los valores menores y mayores un cartucho en negro eliminando los valores activos de la imagen.

En cualquier caso, una vez obtenido el cartucho, y centrada la imagen a buscar lanzaremos la búsqueda de los jeroglíficos. Como comentamos anteriormente la ejecución en paralelo es un método óptimo de búsqueda. Partimos, al igual que en la búsqueda de contornos, de la imagen binarizada de los bordes de Canny eliminando la parte relativa a bordes que hemos comentado. Podemos ver el resultado en la siguiente imagen.

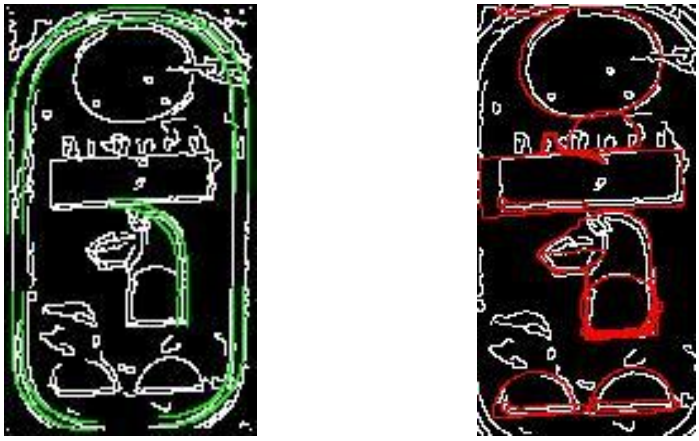


Ilustración 46: Localización de los jeroglíficos con umbral T = 75%

Podemos ver cómo en la imagen de la derecha se han eliminado los bordes del cartucho, y se han localizado los distintos jeroglíficos. Aun así vemos que se han producido algunas detecciones erróneas. No obstante la solución no es la definitiva. Faltaría aplicar el método por el que jeroglíficos mayores absorben a otros cuando son más pequeños y están integrados, aunque sólo sea una parte, en el mayor.

La búsqueda de jeroglíficos como se ha comentado antes se realizaría de izquierda a derecha y de arriba a abajo.

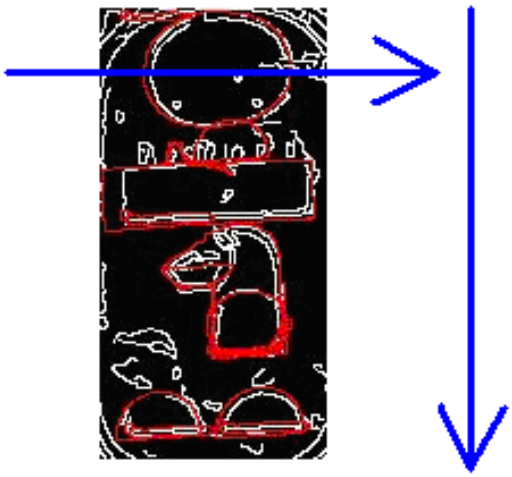


Ilustración 47: Búsqueda de jeroglíficos

El resultado obtenido en el ejemplo sería una cadena de la forma:

1. RA (Circunferencia)
2. MN (El rectángulo)
3. PHTY (La cabeza del León)
4. T (Semi-círculo)
5. T (Semi-círculo)

La cadena se representaría como RA-MN-PHTY-T-T.

Entre el jeroglífico RA y el jeroglífico MN podemos apreciar cómo se ha localizado otro semi-círculo T. Durante el proceso dicho semi-círculo quedará absorbido tanto por RA como por MN ya que ambos son jeroglíficos mayores, y el semi-círculo levemente está dentro de ellos.

A partir de la cadena resultante y utilizando el algoritmo de la distancia de Levenshtein se obtendrá que se trata de Ramsés I. En este caso la distancia de Levenshtein será 0 ya que coincide totalmente con la cadena.

Este método posee diversas ventajas respecto a otros métodos, incluyendo por supuesto que es efectivo. El gran inconveniente es sin duda el tiempo de computación, que como vimos puede ser mejorado. Además está claro que la traducción de textos jeroglíficos egipcios no requiere tampoco de gran velocidad, al ser una actividad poco frecuente en tiempo pequeño. Las ventajas son:

- Es un método efectivo. Se busca realmente lo que se quiere encontrar.
- Es extensible. Se puede abordar el reconocimiento de texto genérico a partir de este método.
- Es más manejable que otros métodos, ya que podemos modificar el algoritmo adaptándolo según creamos necesario. Otros métodos como el reconocimiento por métodos de atención a la

visualización son más difíciles de adaptar, ya que requieren de procesos iterativos ligados a fórmulas matemáticas donde es difícil modificar algo.

- Ante todo es un método válido en otros campos. Podría ser utilizado en el reconocimiento de otro tipo de objetos de leves variaciones. A modo de ejemplo podríamos hablar de los propios jeroglíficos mayas, que tampoco presentan grandes variaciones. Sin embargo, no sería apropiado en caracteres con grandes variaciones, como puede ser el Chino Mandarín.

4.3. CARTUCHOS GIRADOS

Un problema existente es el que hace referencia a qué ocurre si la foto que se realiza de un cartucho está girada. El método de búsqueda de jeroglíficos expuesto en el apartado anterior es insensible al tamaño de los jeroglíficos, pero no lo es al giro de estos.

La solución consiste en girar antes el jeroglífico hasta una posición recta que permita realizar la búsqueda como se ha expuesto en el punto anterior.

Imaginemos que queremos realizar la búsqueda en la imagen siguiente, rotada 20º:

Ilustración 48: Cartucho girado

La técnica utilizada para rotar el cartucho consiste en la aplicación de la transformada de Hough. La transformada de Hough para rectas, como se ha comentado anteriormente, nos permitiría obtener las rectas más probables dentro del jeroglífico. Si utilizamos el máximo contador de rectas del algoritmo que implementa la transformada de Hough, podríamos conocer cuál o cuáles son las rectas principales, que en un cartucho egipcio deberían coincidir con los bordes altos del cartucho.

Por ejemplo, en la imagen anterior, dicha operación devolvería la recta más larga que tiene el cartucho, que en este caso es su borde izquierdo.

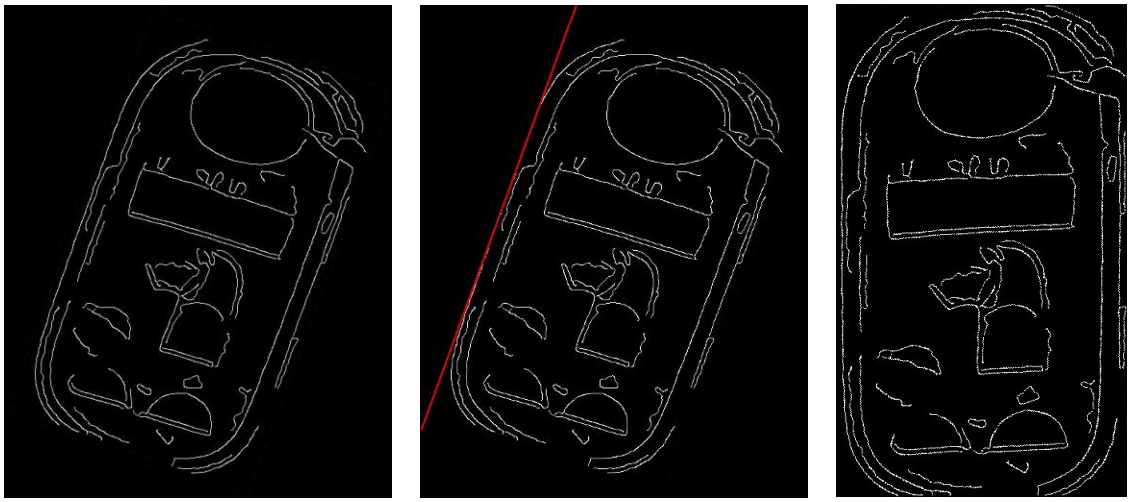


Ilustración 49: Transformada de Hough con umbral máximo y rotación del cartucho

Rotando el cartucho en la dirección contraria a la recta devuelta por el algoritmo lo pondríamos en posición recta. Para calcular cuánto debemos girar el cartucho obtenemos la ecuación de la recta de la forma:

$$\frac{x-x_1}{x_2-x_1}=\frac{y-y_1}{y_2-y_1}\text{, o bien: } y=(x-x_1)\cdot\frac{y_2-y_1}{x_2-x_1}+y_1$$

Teniendo los puntos (x_1,y_1) y (x_2,y_2) los siguientes valores:

$$\begin{cases}x_1=\rho\cdot\cos(\theta)-1000\cdot\text{sen}(\theta)\\ y_1=\rho\cdot\sin(\theta)+1000\cdot\cos(\theta)\end{cases}$$
$$\begin{cases}x_2=\rho\cdot\cos(\theta)+1000\cdot\text{sen}(\theta)\\ y_2=\rho\cdot\sin(\theta)-1000\cdot\cos(\theta)\end{cases}$$

Tanto ρ como θ son devueltos por el algoritmo que implementa la transformada de Hough. La ecuación $f(x)=m\cdot x+b$, nos indica el valor de la pendiente m. A partir de la pendiente podemos obtener fácilmente el ángulo de la recta con el eje X mediante la expresión $\alpha=\arctan(m)$.

El valor de m es $m=\frac{y_2-y_1}{x_2-x_1}$, por lo que el valor del ángulo sería $\alpha=\arctan\left(\frac{y_2-y_1}{x_2-x_1}\right)$.

Para calcular cuánto deberíamos rotar la imagen deberíamos aplicar la siguiente fórmula:

$$\lambda_{imagen}=\alpha-\frac{\pi}{2}$$

Esto quiere decir, que como el ángulo de la recta del ejemplo anterior sería $\alpha=70^o$, habría que rotar la imagen $\lambda_{imagen}=\alpha-\frac{\pi}{2}=70^o-90^o=-20^o$.

Otra forma analizada de rotación de la imagen consistiría, al igual que en la búsqueda de elementos por el espacio de la imagen, en buscar un elemento por la imagen de bordes del algoritmo de Canny.

El elemento buscado será un elemento común en los cartuchos egipcios. En nuestro caso hemos seleccionado el elemento del borde superior izquierdo. Se contará con que el cartucho puede estar girado como máximo 30º. Si estuviera girado más podría tratarse de un cartucho horizontal, y no vertical, por lo que al rotarlo estaríamos rotando 90º los jeroglíficos.

El elemento buscado lo intentaremos localizar en 7 rotaciones diferentes: -30º, -20º, -10º, 0º, 10º, 20º y 30º. Para cada rotación se haría una búsqueda con distintos tamaños. Una forma de acelerar el proceso de búsqueda es la utilización de distintos hilos de ejecución Java.

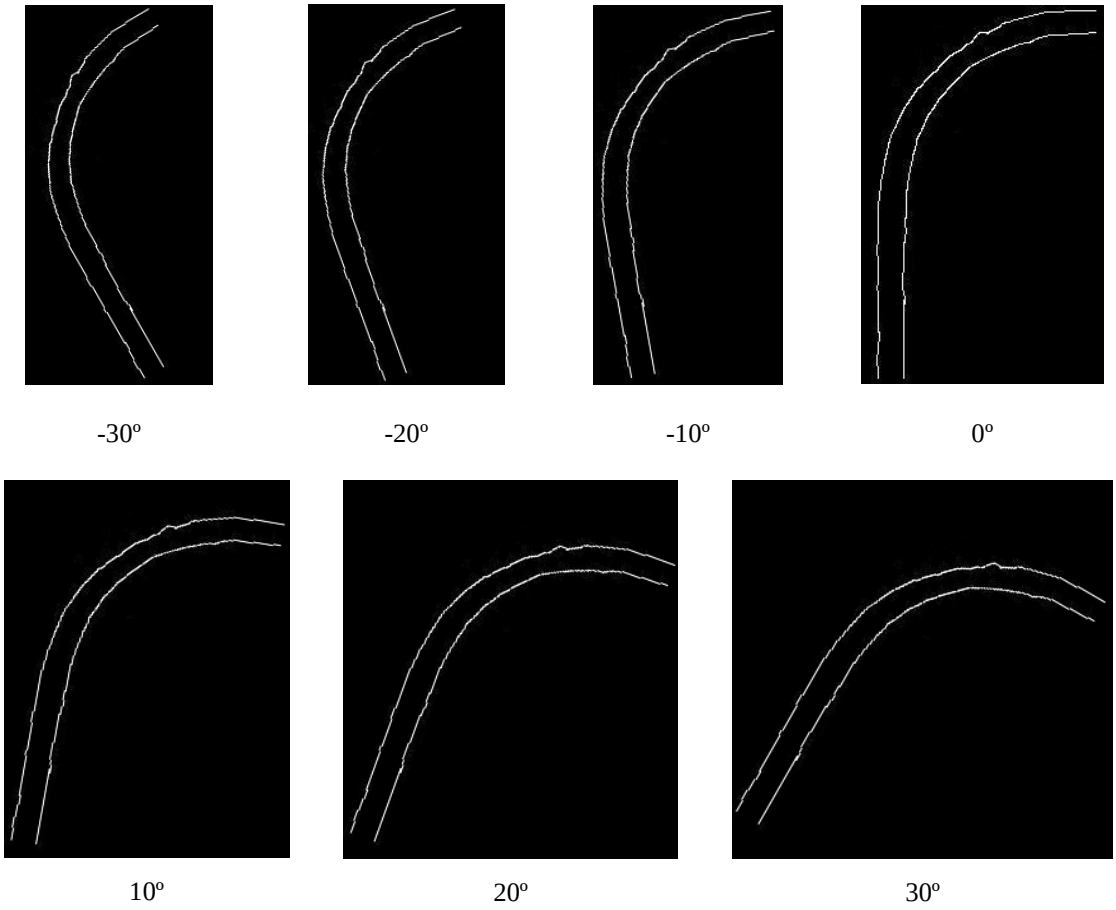


Ilustración 50: Elemento buscado rotado 60 grados

Al realizar la búsqueda sobre la imagen de bordes de Canny, obtendremos en qué posición se encuentra el elemento. Con el fin de no obtener resultados negativos se puede utilizar un umbral del 90º de aceptación de elementos encontrados. En la imagen girada que hemos mostrado anteriormente se obtendrían los siguientes elementos encontrados:

Ilustración 51: Búsqueda de elemento en imagen rotada

Se ha devuelto que se ha encontrado el elemento rotado 20º. Una vez que sabemos qué rotación ha sido necesaria sobre los elementos para localizarlos en la imagen, sabremos también cómo debemos rotar la imagen para ponerla recta. Como el elemento buscado se ha encontrado con una rotación de 20º, será necesario rotar la imagen -20º, que es justo el inverso. Para cada posible ángulo habría que rotar la imagen de la siguiente manera:

- Elemento encontrado rotado -30º: Rotamos la imagen de entrada 30º.
- Elemento encontrado rotado -20º: Rotamos la imagen de entrada 20º.
- Elemento encontrado rotado -10º: Rotamos la imagen de entrada 10º.
- Elemento encontrado rotado 0º: No rotamos la imagen de entrada.
- Elemento encontrado rotado 10º: Rotamos la imagen de entrada -10º.
- Elemento encontrado rotado 20º: Rotamos la imagen de entrada -20º.
- Elemento encontrado rotado 30º: Rotamos la imagen de entrada -30º.

Si tuviéramos más ángulos de búsqueda, sería necesario rotar la imagen de la misma forma. A partir de la imagen rotada, y haciendo la búsqueda explicada en el punto anterior, obtendremos que se trata del cartucho de Ramsés I.

Ilustración 52: Búsqueda final una vez rotada -20 grados la imagen de entrada (bordes de Canny)

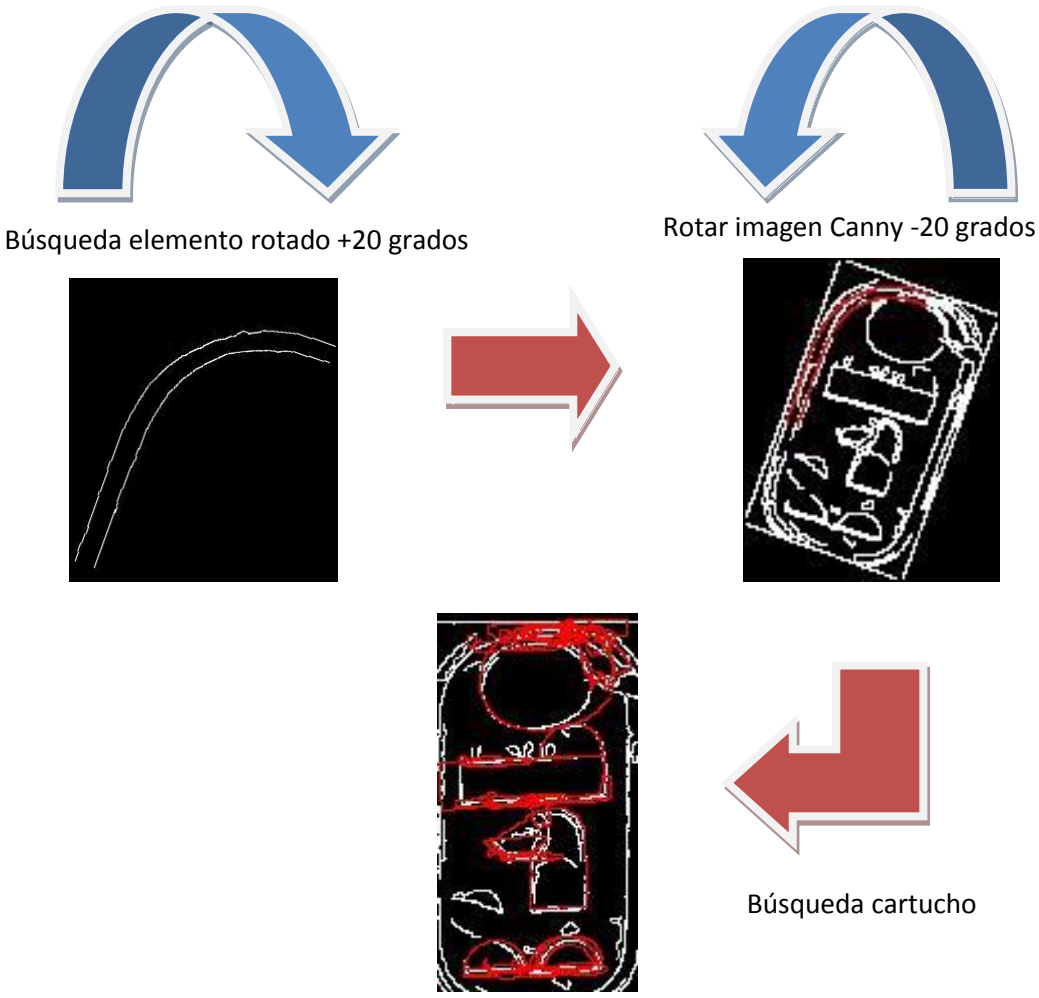


Ilustración 53: Esquema del proceso de rotación

El método de la transformada de Hough ofrece mejores resultados, puesto que el método de encontrar el borde superior izquierdo del cartucho puede devolver resultados en puntos incorrectos, como pueden ser lados de jeroglíficos. Se podrían buscar también distintas esquinas del jeroglífico, pero aun así ofrece mejores resultados la transformada de Hough y además es un proceso mucho más rápido.

4.4. Base de datos de conocimiento

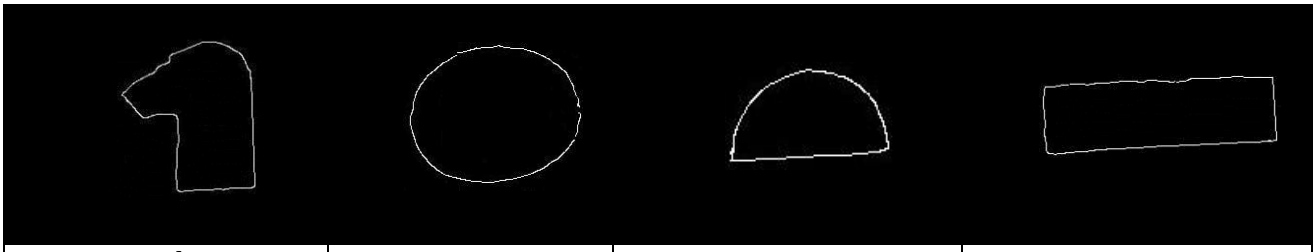
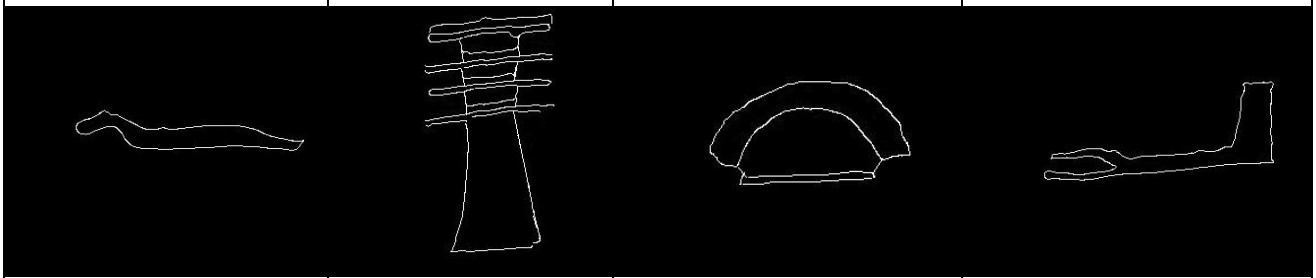
Por base de datos de conocimiento entendemos un almacén de datos donde se encuentran los distintos jeroglíficos a buscar. Se hicieron distintas pruebas con distintas bases de conocimiento. La primera idea pensada consistía en buscar directamente los cartuchos en una base de datos de cartuchos, y no de jeroglíficos. El problema de este método era que su eficiencia sería escasa al encontrarnos con cartuchos nuevos. Dentro del cartucho de un faraón los jeroglíficos podían cambiar levemente de

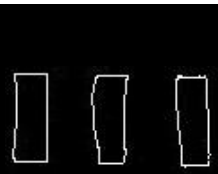
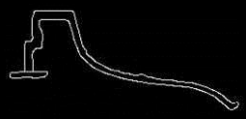
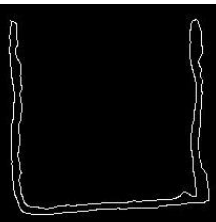
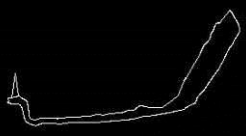
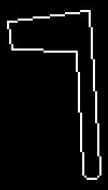
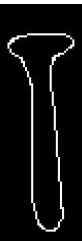
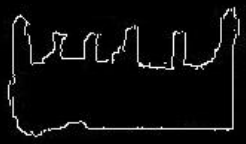
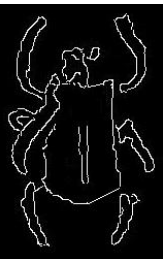
posición, e incluso ser más grandes o más pequeños. Además tampoco era una idea apropiada para reconocimiento de más texto.

La segunda idea barajada consistía en obtener regiones correspondientes a jeroglíficos mediante el algoritmo de regiones por frontera, y utilizar los momentos invariantes de Hu para obtener los jeroglíficos mediante una base de datos de jeroglíficos con sus momentos de Hu. Como comentamos en el algoritmo de regiones por frontera el método fue descartado por los resultados negativos que se producían.

El método de búsqueda de jeroglíficos por el espacio de la imagen fue el que produjo mejores resultados. Se extrajeron los diversos jeroglíficos utilizados en la Lista Real de Abydos. Muchos jeroglíficos son repetidos entre distintos cartuchos. El sistema no obstante abre la posibilidad a meter dos o más veces un mismo jeroglífico en la base de datos de conocimiento si se ha visto que está escrito de manera muy diferente en distintos textos o cartuchos. Para obtener los jeroglíficos se binarizó la imagen de bordes producida por el algoritmo de Canny, y se extrajeron de esta los elementos necesarios.

La base de datos creada para este método, que fue elegido finalmente, es la siguiente. Se ha intentado respetar la fonética real del jeroglífico egipcio. De esta manera un egiptólogo podrá reconocer fácilmente los distintos jeroglíficos utilizados.

			
phty	ra	t	mn
			
maat	w	nfr	s
			
f	djed (dd)	h	a

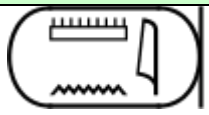
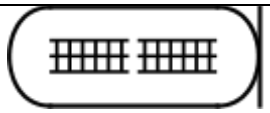
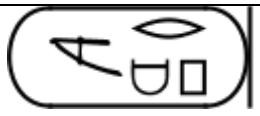
			
u	hpr	n	stp
			
ka	kh	nb	Dsr
			
i	p	S	mr
			
r	wsr	ba	mt
			
d	ntr	b	da
			
wad	hpr	mn	hpr

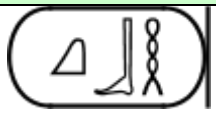
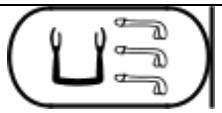
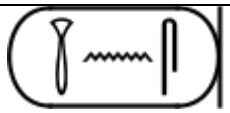
maat			

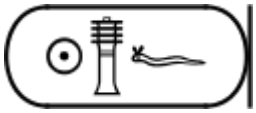
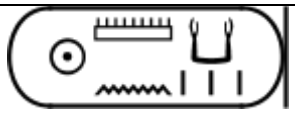
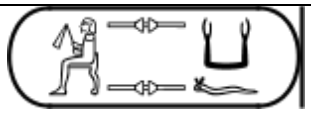
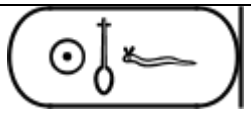
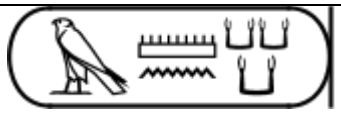
Ilustración 54: Base de conocimiento de jeroglíficos utilizada

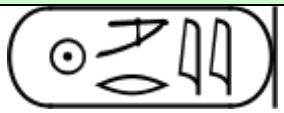
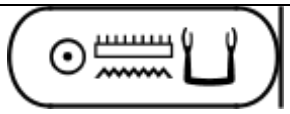
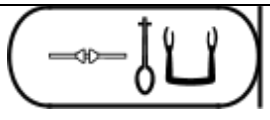
Como podemos ver, es posible añadir algunos jeroglíficos del mismo carácter más de una vez. Esto es necesario si existen cambios significativos del mismo jeroglífico entre diferentes cartuchos. El jeroglífico Jeper (*hpr*), que representa al escarabajo sagrado, aparece repetido tres veces, el jeroglífico *maat* aparece dos veces, y el jeroglífico *mn* aparece dos veces.

Además de la base de datos de jeroglíficos es necesario contar con la base de datos de los nombres reales. Cada faraón tiene un nombre real compuesto por distintos jeroglíficos. A continuación vemos los 76 faraones que aparecen en la lista real de Abydos, incluyendo su transliteración.

Número	Cartucho del Faraón	Transliteración (códigos jeroglíficos)	Transliteración fonética	Nombre del Faraón	Dinastía
1		Y5:N35-M17	Meni m-n-i	Narmer	I
2		X1:X1-M17	Teti t-t-i	Aha	I
3		M17-X1-U33	Atet i-t-t	Djer	I
4		M17-X1-G1	Ita i-t-ꜥ	Wadj	I
5		N24-N24	Sepati sp$_3$ti	Den	I
6		U7-D21:N42*Q3	Meribiap mri-bi$_3$-p	Anedjib	I
7		A23D	Semsu smsw	Semerkhet	I

Número	Cartucho del Faraón	Transliteración (códigos jeroglíficos)	Transliteración fonética	Nombre del Faraón	Dinastía
8		N29-D58-V28	Qe-be-h ḳ-b-ḥ	Qaa	I
9		D58-U28-G43-P11	Bedjau bḏ$_3$w	Hetepsekhemwy	II
10		D28-D52:D52:D52	Kakau k$_3$-k$_3$w	Nebra	II
11		W10A*E11-R8:N35	Ba-netjer b$_3$-nṯr	Ninetjer	II
12		M13-N35-S29	Wadjnes w$_3$ḏ-n-s	Wadjnes	II
13		S29-N35:D46-M17	Senedj s-n-d-i	Senedj	II
14		U28-U28-D1-M17-M17	Djadjay ḏ$_3$-ḏ$_3$-y	Khasekhemwy	II
15		V30-D28-Z1	Nebka nb-k$_3$	Nebka	III
16		//-D45-V17	Djesersa ḏsr-s$_3$	Djeser	III
17		X1:X1-M17	Teti t-t-i	Sekhemkhet	III
18		O34:I10-S29	Sedjes sḏs	Hudjefa II	III
19		N5:V30-D28	Neferkara nfr-k$_3$-rʿ	Neferkara	III
20		S29-F35-G43	Sneferu s-nfr-w	Sneferu	IV
21		Aa1:I9-G43	Khufu ḫ-f-w	Khufu	IV

Número	Cartucho del Faraón	Transliteración (códigos jeroglíficos)	Transliteración fonética	Nombre del Faraón	Dinastía
22		N5-R11-I9	Djedefra (Radjedef) ḏd-f-rˁ (rˁ-ḏd-f)	Djedefra	IV
23		N5-N28-D36:I9	Khafra ḫˁ-f-rˁ	Khafra	IV
24		N5-Y5:N35-D28:Z2	Menkaura mn-k$_3$w-rˁ	Menkaura	IV
25		A51-O34:O34-D28:I9	Shepseskaf šps-s-k$_3$-f	Shepseskaf	IV
26		F12-S29-D28-I9	Userkaf wsr-k$_3$-f	Userkaf	V
27		N5-D61-G43	Sahura s$_3$ḥw-rˁ	Sahura	V
28		D28-D28-M17	Kakai k$_3$-k$_3$-i	Neferirkara I Kakai	V
29		N5-F35-I9	Neferefra nfr-f-rˁ	Neferefra Isi	V
30		N5-N35-F12*S29:D21	Niuserra ni-wsr-rˁ	Niuserra Ini	V
31		G5-Y5:N35-D28*D28:D28	Menkauhor mn-k$_3$w-ḥr	Menkauhor Kaiu	V
32		N5-R11-D28	Djedkara ḏd-k$_3$-rˁ	Djedkara Isesi	V
33		E34:N35-M17-S29	Unas wnis	Unas	V
34		X1:X1-M17	Teti tti	Teti II	VI
35		N5-F12*S29-D28	Userkara wsr-k$_3$-rˁ	Userkara	VI

Número	Cartucho del Faraón	Transliteración (códigos jeroglíficos)	Transliteración fonética	Nombre del Faraón	Dinastía
36		N5-U7:D21-M17-M17	Meryra mry-rˁ	Pepi I	VI
37		N5:U7-D21:N35	Merenra mr-n-rˁ	Nemtiemsaf I	VI
38		N5-F35-D28	Neferkara nfr-k$_3$-rˁ	Pepi II	VI
39		N5-U7:N35-G42-G17-V16:I9	Merenra Samsaf mr-n-rˁ s$_3$-m-s$_3$-f	Nemtiemsaf II	VI
40		N5-R8-D28	Netjerikara nṯr.i-k$_3$-rˁ	Netjerikara	VIII
41		N5-Y5:N35-D28	Menkara mn-k$_3$-rˁ	Menkara	VIII
42		N5-F35-D28	Neferkara nfr-k$_3$-rˁ	Neferkara II	VIII
43		N5-F35-D28-V30-D58-M17-M17	Neferkara Neby nfr-k$_3$-rˁ nb-ii	Neferkara Neby	VIII
44		N5-R11-D28-U4-A33	Djedkara Shemai ḏd-k$_3$-rˁ-šm$_3$i	Djedkara Shemai	VIII
45		N5-F35-D28-Aa1:N35:D46-G43-D56	Neferkara Khendu nfr-k$_3$-rˁ ḫndw	Neferkara Khendu	VIII
46		G5-U7:D21:N35	Merenhor mr-n-ḥr	Merenhor	VIII
47		O34-F35-D28	Seneferka s-nfr-k$_3$	Neferkamin I	VIII
48		N5-N35-D28	Nikara ni-k$_3$-rˁ	Nikara	VIII
49		N5-F35-D28:X1-D21:D21:E23	Neferkara Tereru nfr-k$_3$-rˁ tr-r-rw	Neferkara Tereru	VIII

Número	Cartucho del Faraón	Transliteración (códigos jeroglíficos)	Transliteración fonética	Nombre del Faraón	Dinastía
50		G5-F35-D28	Neferkahor nfr-k$_3$-ḥr	Neferkahor	VIII
51		N5-F35-D28-Q3:Q3-S29-N35:D58*M17*M17	Neferkara Pepiseneb nfr-k$_3$-rˁ pi-pi-s-n-bi	Neferkara Pepiseneb	VIII
52		O34:F35*D28-D36:N35-W24*G43	Neferkamin Anu s-nfr-k$_3$-ˁ-n-nw-w	Neferkamin Anu	VIII
53		N5-A28-D28:Z2	Qakara ḳ$_3$-k$_3$w-rˁ	Qakara Ibi	VIII
54		N5-F35-D28:Z2	Neferkaura nfr-k$_3$w-rˁ	Neferkaura	VIII
55		G5-F35-D28:Z2	Neferkauhor nfr-k$_3$w-hr	Neferkauhor	VIII
56		N5-F35-D4-D28	Neferirkara nfr-ir-k$_3$-rˁ	Neferirkara II	VIII
57		N5:V30-P8	Neb-hapet-ra nb-ḥ$_3$pt-rˁ	Mentuhetep II	XI
58		N5-S29-S34-D28	Sankhkara s-ˁnḫ-k$_3$-rˁ	Mentuhetep III	XI
59		N5-S29-R4:X1*Q3-F34	Sehetepibra s-ḥtp-ib-rˁ	Amenemhat I	XII
60		N5-L1-D28	Kheperkara ḫpr-k$_3$-rˁ	Senusret I	XII
61		N5:S12-D28*D28:D28	Nebukaura nbw-k$_3$w-rˁ	Amenemhat II	XII
62		N5-N28-L1	Khakheperra ḫˁi-ḫpr-rˁ	Senusret II	XII
63		N5-N28-D28:D28*D28	Khakaura ḫˁ-k$_3$w-rˁ	Senusret III	XII

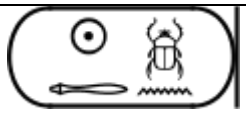
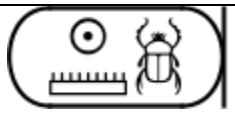
Número	Cartucho del Faraón	Transliteración (códigos jeroglíficos)	Transliteración fonética	Nombre del Faraón	Dinastía
64		N5:N35-U4-D36:X1	Nimaatra ni-m$_3$ʿt-rʿ	Amenemhat III	XII
65		N5-U4:D36-P8-G43	Maatkherura m$_3$ʿt-ḫrw-rʿ	Amenemhat IV	XII
66		N5-V30-F9:X1*X1	Nebpehtira nb-pḥwt-rʿ	Ahmose I	XVIII
67		N5-D45-D28	Djeserkara ḏsr-k$_3$-rʿ	Amenhetep I	XVIII
68		N5:I29-L1-D28	Aakheperkara ʿ$_3$-ḫpr-k$_3$-rʿ	Tuthmosis I	XVIII
69		N5:O29-L1:N35	Aakheperenra ʿ$_3$-ḫpr-n-rʿ	Tuthmosis II	XVIII
70		N5:Y5-L1	Menkheperra mn-ḫpr-rʿ	Tuthmosis III	XVIII
71		N5:O29-L1-Z2	Aakheperura ʿ$_3$-ḫprw-rʿ	Amenhetep II	XVIII
72		N6-Y5-L1-Z2	Menkheperura mn-ḫprw-rʿ	Tuthmosis IV	XVIII
73		N5-C10-V30	Nebmaatra nb-m$_3$ʿt-rʿ	Amenhetep III	XVIII
74		N5:D45-L1:Z2-U21*N5:N35	Djeserkheperura Setepenra ḏsr-ḫprw stp-n-rʿ	Horemheb	XVIII
75		N5:Y5-F9:X1*X1	Menpehtyra mn-pḥti-rʿ	Ramesses I	XIX
76		N5-C10A-Y5	Menmaatra mn-m$_3$ʿt-rʿ	Seti I	XIX

Ilustración 55: Base de conocimiento de faraones. Lista Real de Abydos

5. Análisis y diseño del sistema

5.1. VISIÓN GENERAL

La visión general del sistema muestra una aplicación en la que un usuario a través de un dispositivo móvil o aplicación web puede tomar fotografías y realizar la búsqueda de cartuchos sobre dichas fotografías. El caso de uso de búsqueda de cartuchos del dispositivo móvil hace uso de otro caso de uso externo alojado en el servidor de aplicaciones. Dicha comunicación se realiza mediante servicios web.

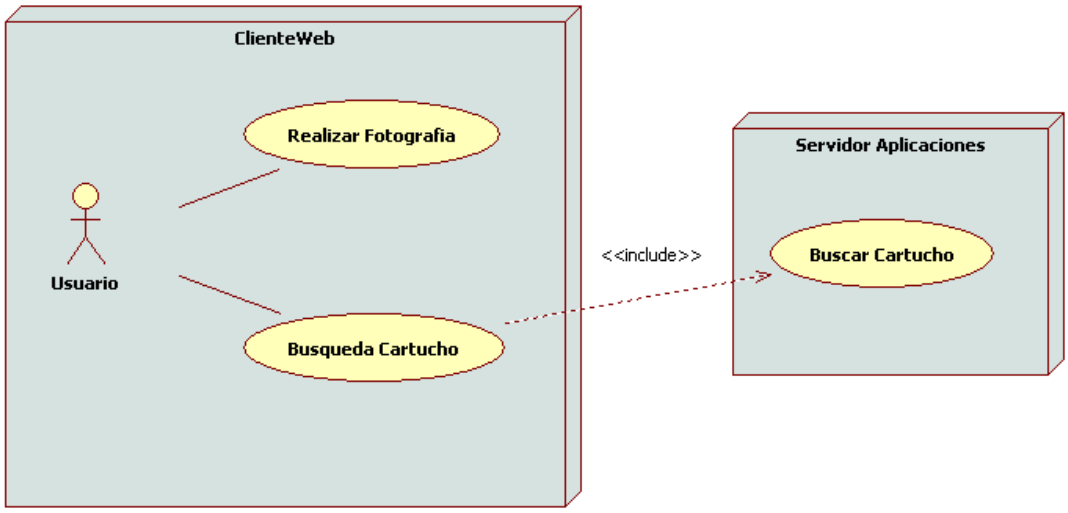


Ilustración 56: Visión general del sistema

5.2. ARQUITECTURA DE CLASES DEL SISTEMA

A continuación vamos a ver cómo es la arquitectura de clases del sistema. Dentro de dicho esquema es importante identificar las clases relativas a la comunicación mediante servicio web. La clase *LocalizacionCartucho* es el verdadero motor de la aplicación, relacionado estrechamente con las clases de *LocalizacionImagen*, utilizada para buscar elementos en la imagen de bordes de Canny.

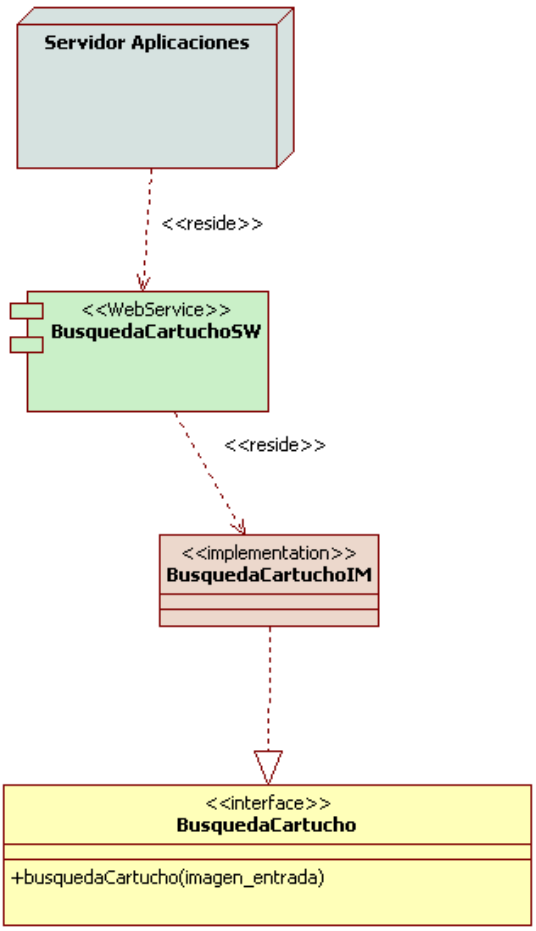


Ilustración 57: Arquitectura del servicio web de búsqueda de cartuchos

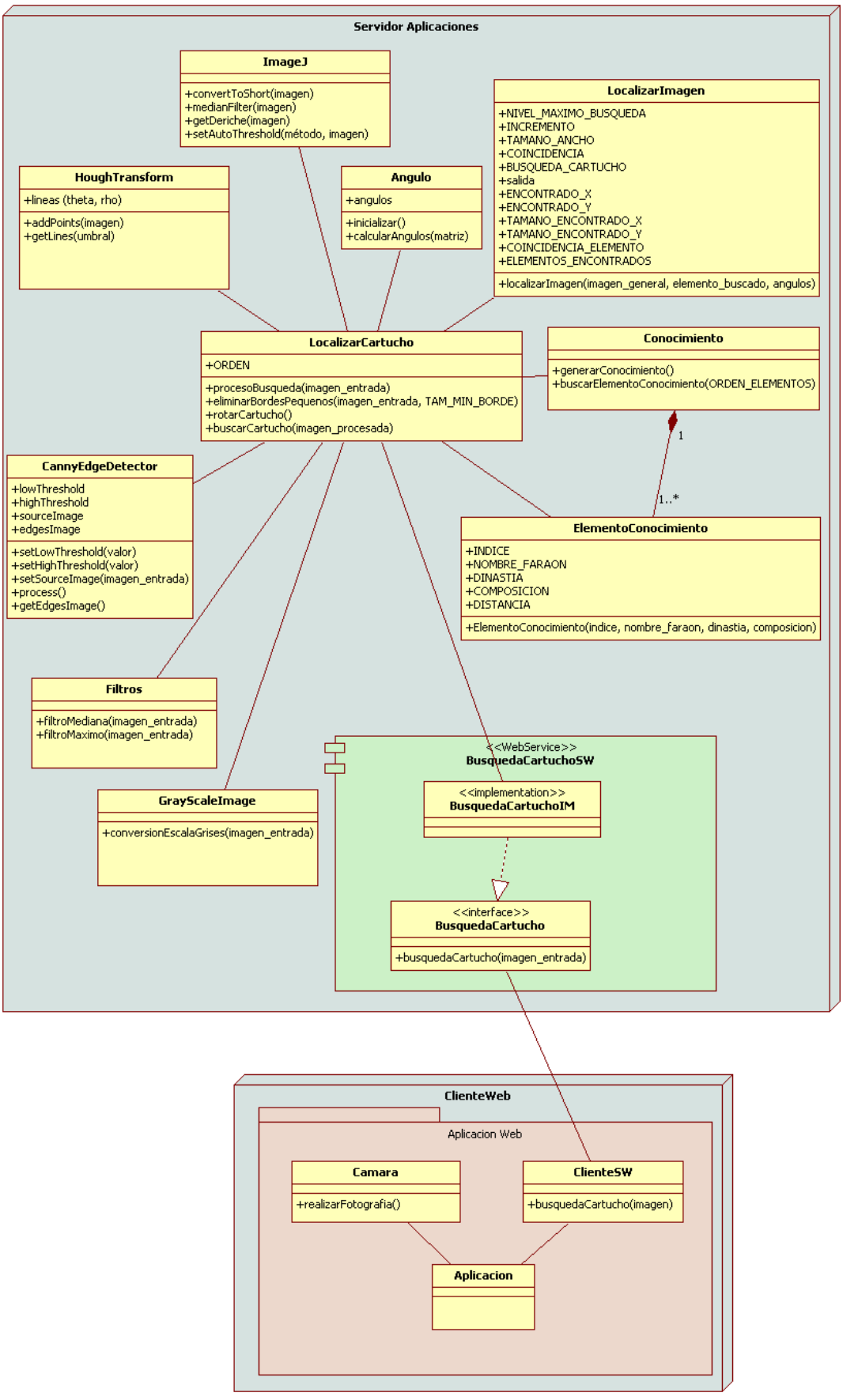


Ilustración 58: Diagrama de clases general

5.3. FUNCIONAMIENTO DEL SISTEMA

El diagrama de secuencia que comentaremos a continuación es especialmente importante, ya que muestra cómo funciona completamente el proceso desde el punto de vista de un programa secuencial e iterativo. Podemos ver cómo se relacionan los distintos objetos creados en cada clase con el fin de realizar la búsqueda del cartucho.

El proceso de funcionamiento sería el siguiente:

- El usuario realiza una fotografía, *:Camara.realizarFotografia();*
- El usuario lanza el proceso de búsqueda del cartucho llamando a un servicio web del servidor, *:ClienteSW.busquedaCartucho();*
- El servicio web llama a su vez al método principal de reconocimiento, *:BusquedaCartucho.busquedaCartucho();*, quien a su vez llama al proceso de búsqueda del cartucho *:LocalizarCartucho.procesoBusqueda();*
- El método de reconocimiento del cartucho realiza las siguientes operaciones:
    - Utilizando las clases de ImageJ se convierte la imagen a escala de grises, *:ImageJ.convertToShort()*, y se aplica un filtro mediana a la imagen, *:ImageJ.medianFilter()*.
    - Utilizando el plug-in de aplicación del algoritmo de Canny de Thomas Boudier y Joris Meys, *:ImageJ.getDeriche()*, se obtienen los bordes, y se aplica una umbralización binaria IsoData, *:ImageJ.setAutoThreshold*, que como vimos es la que ofrece mejores resultados.
    - Se reducen los bordes de tamaño inferior a un umbral predefinido mediante *:LocalizarCartucho.eliminarBordesPequenos();*
    - Para obtener la rotación del cartucho se utiliza la transformada de Hough. Esta se comporta mejor con el algoritmo de reconocimiento de bordes de Tom Gibara, lo que nos lleva a realizar el proceso anterior mediante este algoritmo:
        - Convierte a escala de grises la imagen, *:GreyScaleImage.conversionEscalaGrises();*, aplica un filtro máximo en la imagen, *:Filtros.filtroMaximo();*, aplica un filtro mediana en la imagen, *:Filtros.filtroMediana();*, y aplica el algoritmo de Canny para extraer los bordes. Para ello establece la imagen, *:CannyEdgeDetector.setSourceImage();* , realiza el proceso de búsqueda de bordes, *:CannyEdgeDetector.process();* , y finalmente extrae la imagen de bordes de salida, *:CannyEdgeDetector.getEdgesImage();*
        - Se aplica un método de eliminación de bordes pequeños, necesario para eliminar ruido leve, *: LocalizarCartucho.eliminarBordesPequenos();*

- Para obtener la línea recta más probable se utiliza la transformada de Hough mediante *:HoughTransform.addPoints()* y *:HoughTransform.getLines()*. Este último método nos devuelve a partir de un umbral máximo la recta más probable, que coincide con el lado del cartucho.
- A partir de la recta más probable se calcula el ángulo de rotación y se rota la imagen mediante *:LocalizarCartucho.rotarCartucho()*. La imagen que se rota es la obtenida mediante el algoritmo de Canny implementado por Thomas Boudier y Joris Meys.

- Se hace un pre-cálculo de operaciones trigonométricas para almacenarlas en una matriz. Esta técnica acelera estas operaciones ya que las búsquedas sobre una tabla indexada son más rápidas que su cálculo. *:Angulo.calcularAngulos();*
- A partir del cartucho rotado localizamos cada uno de los cuatro extremos del cartucho. Esto sirve para encuadrar el cartucho. Para localizar cada extremo utilizamos el método *:LocalizarImagen.localizarImagen().* Una vez localizados los cuatro extremos del cartucho procedemos a extraerlo del resto de la imagen.
- El siguiente paso consiste en realizar la búsqueda de cada jeroglífico en el espacio de la imagen resultante. Esto se realiza lanzando distintos hilos de ejecución, uno por cada jeroglífico, del método *:LocalizarImagen.localizarImagen().* Este método devuelve los 5 jeroglíficos más probables de cada tipo, siempre que superen un umbral de coincidencia.
- Los jeroglíficos grandes absorben a los que se encuentran dentro de ellos. Recorriendo la imagen de arriba abajo, y de izquierda a derecha se crea una lista de jeroglíficos encontrados.
- Se genera una base de datos de conocimiento, mediante *:Conocimiento.generarConocimiento();*
  - Para cada elemento de conocimiento, o faraón, se llamará al método *:ElementoConocimiento.elementoConocimiento();* Cada elemento de conocimiento tendrá los siguientes atributos:
    - INDICE del elemento de conocimiento.
    - NOMBRE_FARAON al que hace referencia el elemento.
    - DINASTIA a la que pertenece el faraón.
    - COMPOSICION[] es la lista de elementos. Por ejemplo: RA-MN-PHTY-T-T.
  - Se localiza el elemento de la base de datos de conocimiento más próximo a los jeroglíficos detectados. Para realizar dicha búsqueda se utiliza el algoritmo de la distancia de Levenshtein, eligiendo aquel cuya distancia es menor. El método utilizado es *:Conocimiento.buscarElementoConocimiento();*

- Finalmente se devuelve el resultado a través de la salida del servicio web al cliente, indicando el nombre del faraón y la dinastía (que sirve para la datación).

A continuación podemos ver el diagrama de secuencia correspondiente.

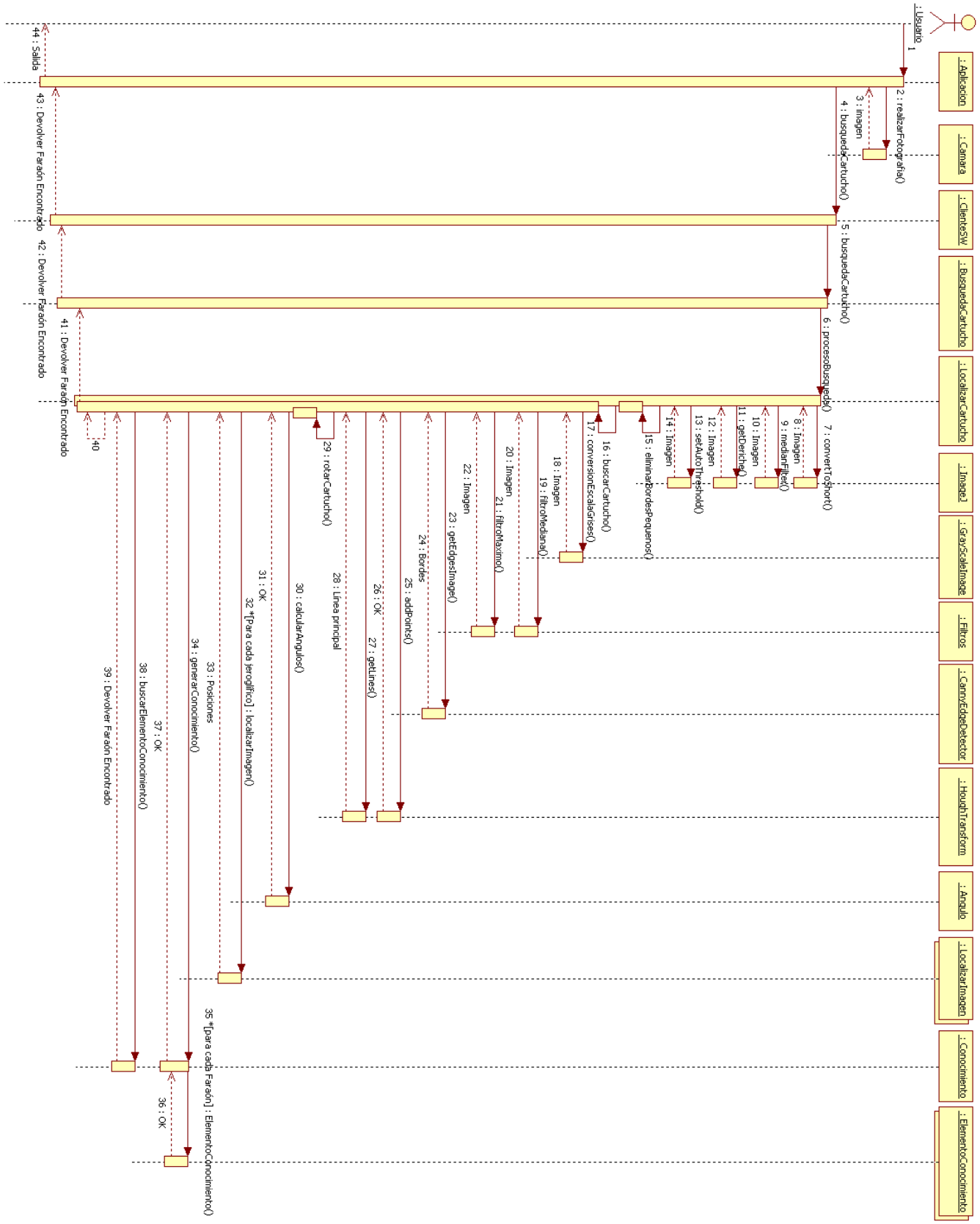


Ilustración 59: Diagrama de secuencia de la búsqueda de un cartucho

5.4. APLICACIÓN WEB

El servicio web desarrollado puede ser consumido por distintos clientes web. Basta con llamar al servicio enviando una imagen con un cartucho, y el servicio responderá indicando de quién se trata. El cliente web puede ser una aplicación móvil de Android, una aplicación que implemente una llamada al servicio web, o bien una página web de llamada. En este trabajo se ha implementado una pequeña aplicación cliente que nos permite seleccionar una imagen y enviarla al servicio web. Básicamente hay un cuadro de texto donde se devuelven los resultados, y un botón que nos abre un cuadro de selección de un fichero.

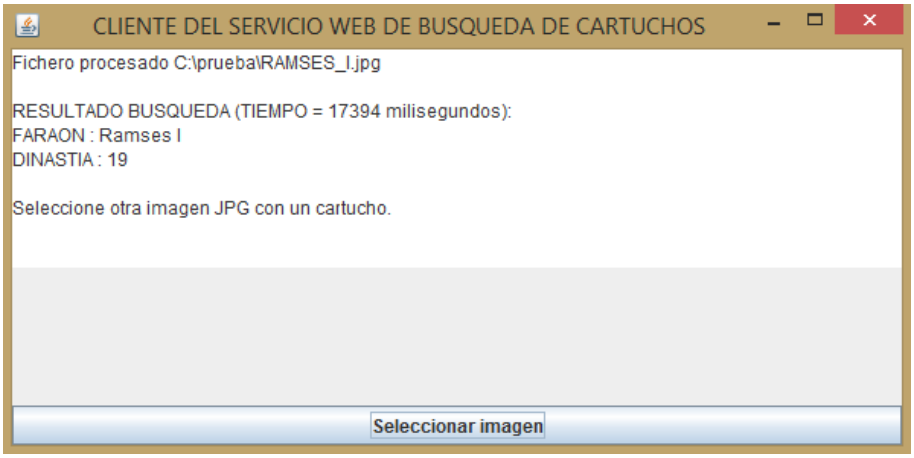


Ilustración 60: Pantalla de la aplicación web que consume el servicio web de búsqueda

La utilización de un servicio web de reconocimiento abre grandes posibilidades, ya que no es necesario descargar el código de reconocimiento ante un cambio del algoritmo. El usuario descargará un cliente concreto y no tendrá que actualizarlo en ningún momento. El servicio web podrá ser modificado de forma transparente al usuario, aumentando por ejemplo la base de datos de jeroglíficos o cartuchos.

6. Pruebas

6.1. Faraón Teti

Teti fue el primer faraón de la dinastía VI de Egipto del Imperio Antiguo. Reinó del año 2322 a.C. hasta el año 2312 a. C.

Ilustración 61: Faraón Teti

El trabajo devuelve los siguientes resultados (ante una coincidencia mínima del 75% de los jeroglíficos)

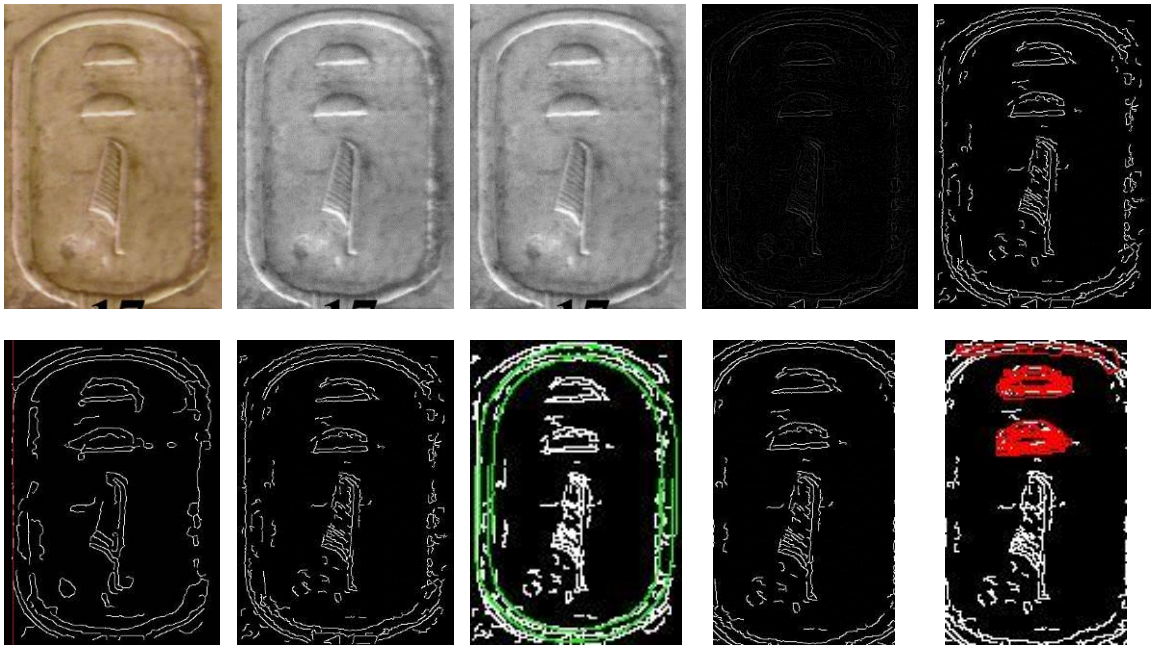


Ilustración 62: Secuencia de reconocimiento del cartucho de Teti

Los pasos de la secuencia de reconocimiento serían los siguientes:

- Imagen original.
- Imagen en escala de grises.
- Imagen después de aplicar Filtro Máximo y Filtro Mediana.
- Bordes por el algoritmo de Canny.
- Eliminación de bordes inferiores al 10% del ancho de la imagen.
- Aplicación de la transformada de Hough para rotar cartucho.
- Rotación del cartucho (si fuera necesaria).
- Detección de bordes para extraer cartucho.
- Selección de la parte interna de la imagen sin cartucho.
- Detección de posibles jeroglíficos. Muchos quedarán absorbidos por los mayores o más probables.

La pantalla de la aplicación devolvió el siguiente resultado, que como podemos ver devuelve correctamente el resultado, a pesar de no localizar un jeroglífico:

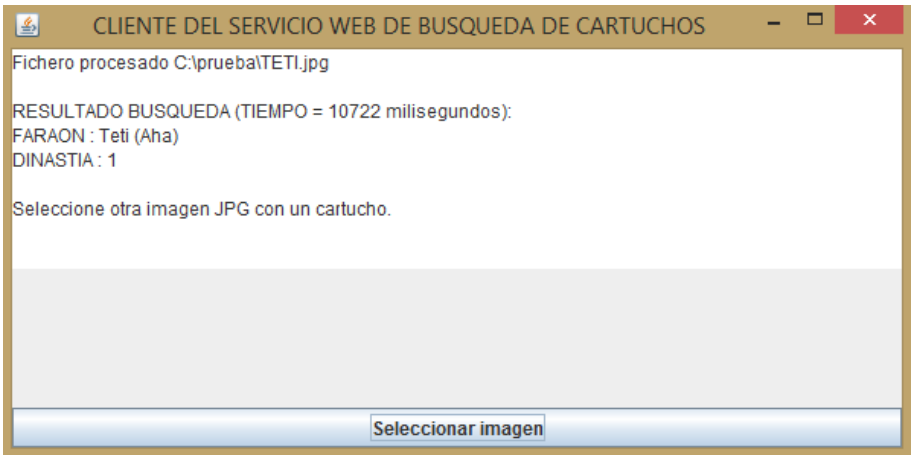


Ilustración 63: Resultado de la detección de Teti

6.2. FARAÓN RAMSÉS I

Men-pehty-Ra Ra-mesesu, o Ramsés I, fue el fundador de la XIX dinastía, perteneciente al Imperio Nuevo de Egipto. Reinó entre los años 1295 a. C. y 1294 a. C. Fue el padre de Seti I, conocido faraón bíblico.

Ilustración 64: Faraón Ramsés I

El trabajo devuelve los siguientes resultados (ante una coincidencia mínima del 75% de los jeroglíficos). Los pasos son los mismos que se han explicado en el faraón Teti.

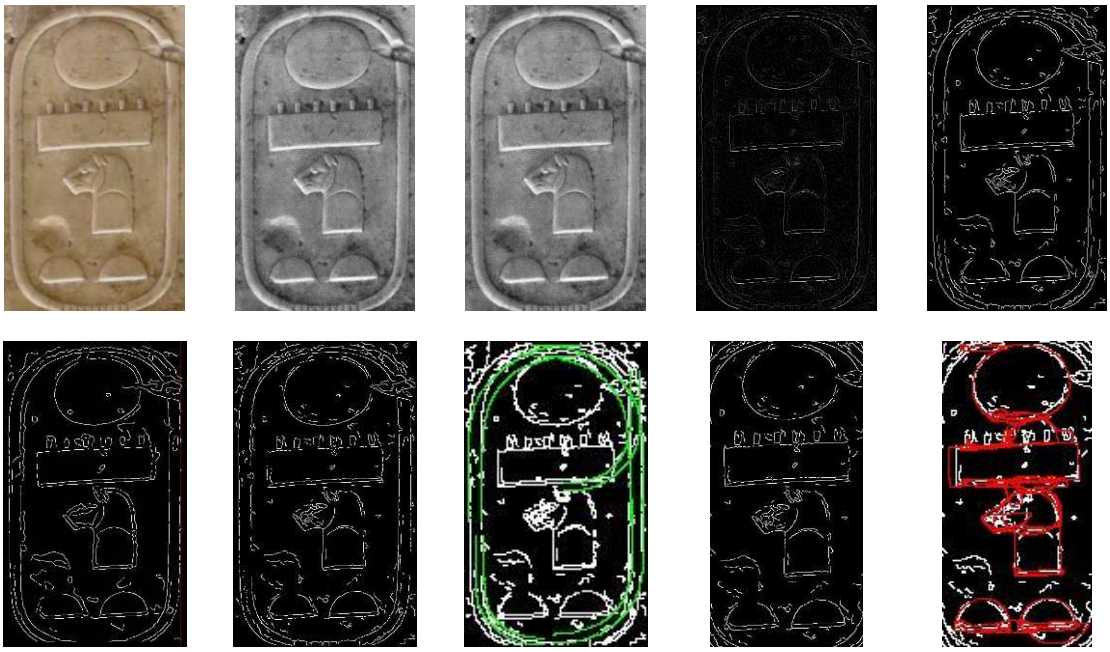


Ilustración 65: Secuencia de reconocimiento del cartucho de Ramsés I

La pantalla de la aplicación devolvió el siguiente resultado, que como podemos ver es correcto:

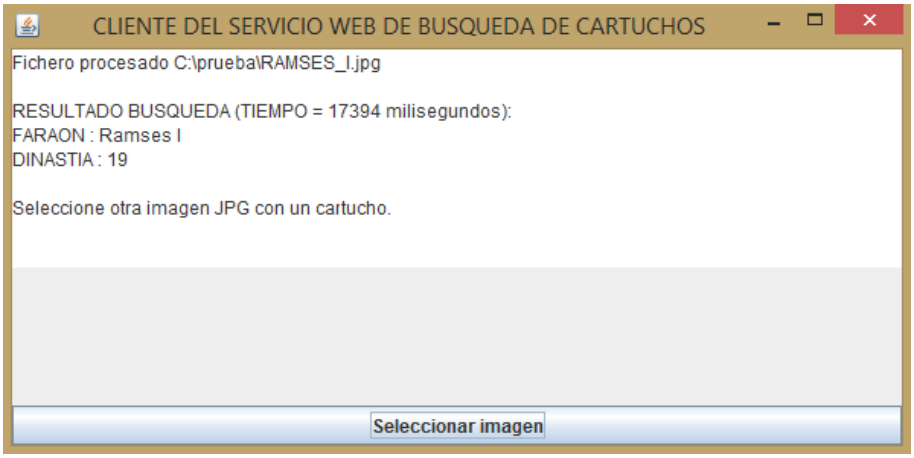


Ilustración 66: Resultado de la detección de Ramsés I

Podemos ver a continuación como el comportamiento ante el cartucho girado es también correcto:

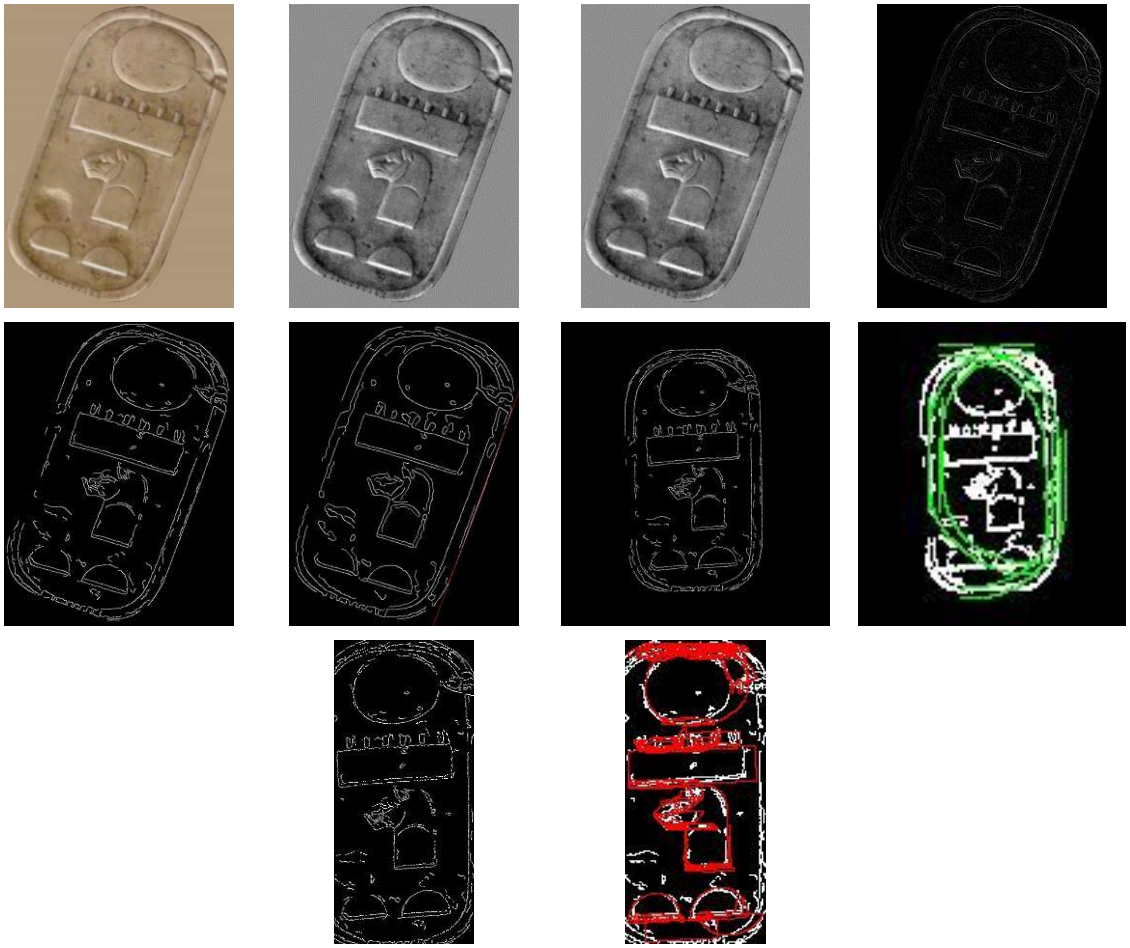


Ilustración 67: Proceso de reconocimiento del cartucho girado de Ramsés I

La pantalla de la aplicación devolvió el siguiente resultado:

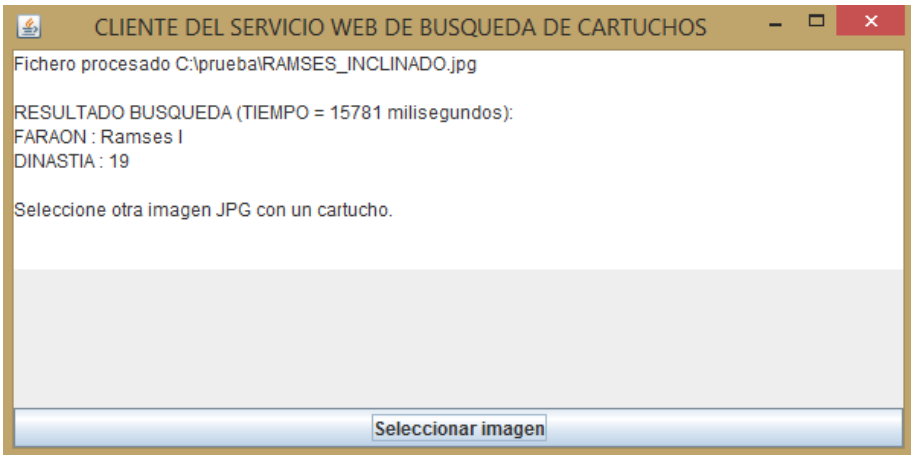


Ilustración 68: Resultado de la detección de Ramsés I inclinado

6.3. Faraón Tutmosis IV

Men-jeper-u-ra, o Tutmosis IV, es el octavo faraón de la dinastía XVIII de Egipto. Su reinado fue en torno a los años 1400 a.C. y 1390 a.C. Para la realización de la prueba se ha utilizado una imagen procedente de un relieve egipcio distinta a las de la lista de Abydos. Se ha añadido un jeroglífico Jeper (escarabajo) a la base de datos, ya que existían notables diferencias con el jeroglífico de Abydos.

Ilustración 69: Faraón Tutmosis IV

El trabajo devuelve los siguientes resultados. Los pasos son los mismos que se han explicado en el faraón Teti.

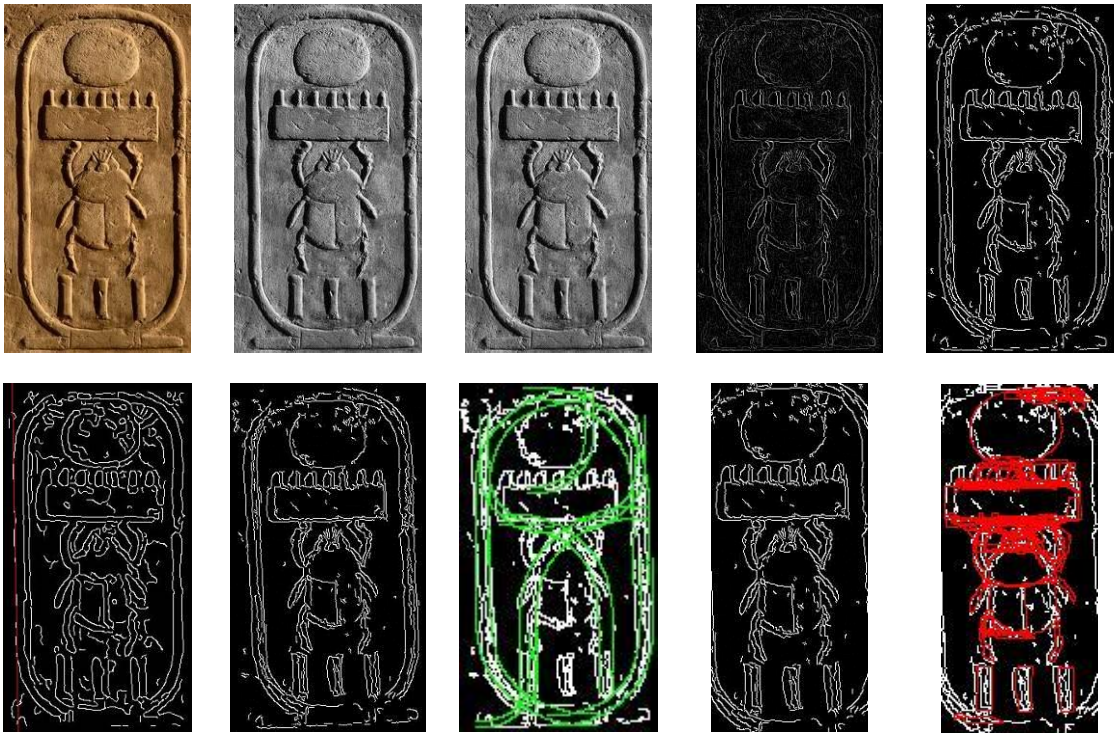


Ilustración 70: Secuencia de reconocimiento del cartucho de Tutmosis IV

La pantalla de la aplicación devolvió el siguiente resultado, que como podemos ver es correcto:

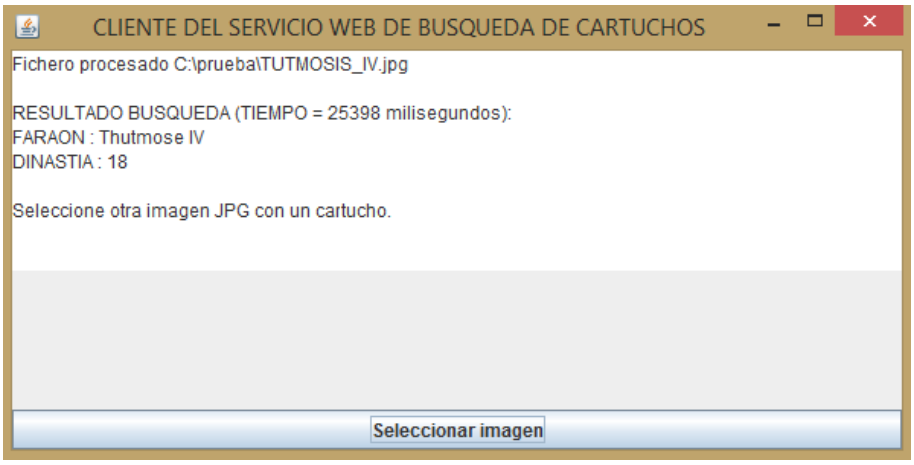


Ilustración 71: Resultado de la detección de Tutmosis IV

En el siguiente ejemplo vamos a ver un cartucho de Tutmosis IV en el que no funcionó bien el reconocimiento. En algunos casos es difícil obtener una solución correcta. El motivo es que el cartucho está partido en dos, y esa rotura le hace pensar al detector que el jeroglífico superior es en realidad otro. Sin embargo, aumentando el umbral de reconocimiento de jeroglíficos del 75% al 77% es capaz de reconocer el jeroglífico, aunque en este caso el último jeroglífico no lo detecta correctamente y confunde el cartucho con otro muy parecido, el de Tutmosis III.

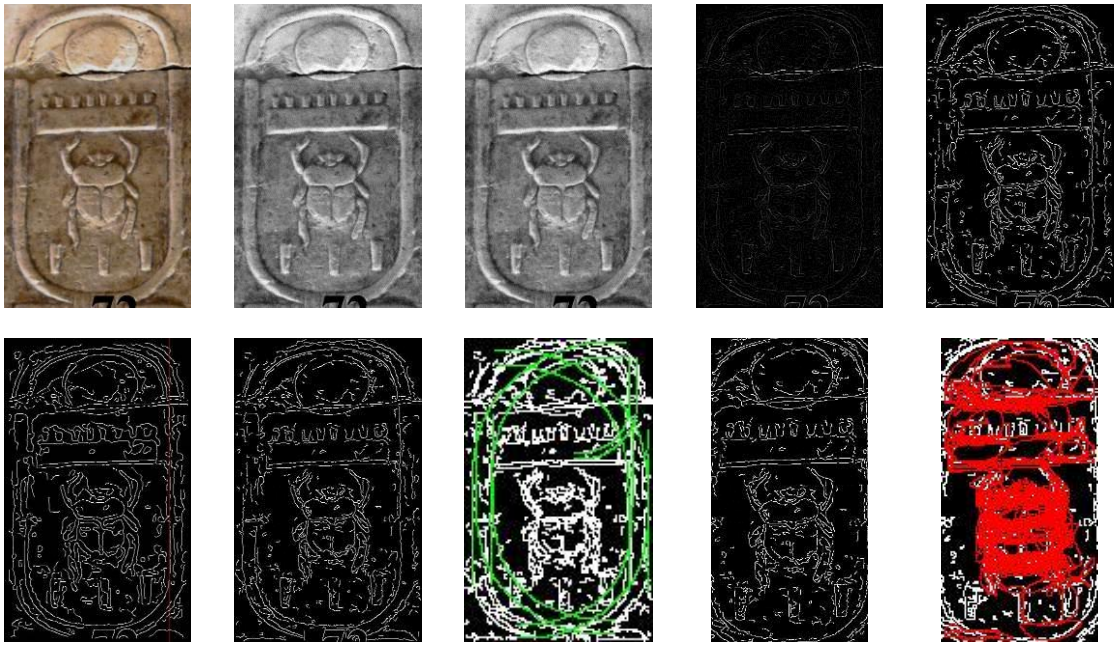


Ilustración 72: Reconocimiento incorrecto de cartucho de Tutmosis IV (devuelve Neferkara). Umbral 75%

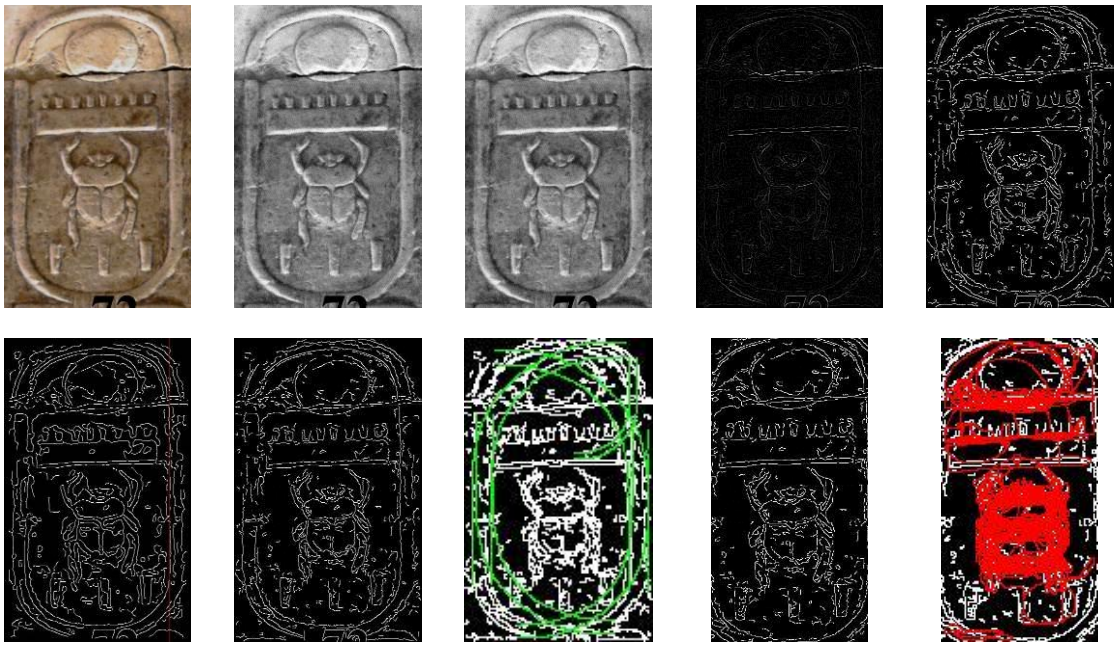


Ilustración 73: Reconocimiento incorrecto de cartucho de Tutmosis IV (devuelve cartucho muy parecido de Tutmosis III). Umbral 77%

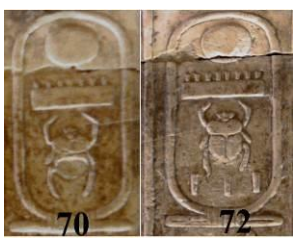


Ilustración 74: Cartucho de Tutmosis III y Tutmosis IV en Lista de Abydos

6.4. FARAÓN TUTMOSIS III

Tutmosis III es el sexto faraón de la dinastía XVIII de Egipto. Gobernó entre los años 1479 a.C. y 1425 a.C. Se le ha denominado "el Napoleón de Egipto", debido a su carácter militar y sus grandes conquistas. Fue uno de los monarcas más importantes y poderosos de los tres mil años de civilización faraónica. En el transcurso de su reinado el imperio egipcio alcanzó su máxima extensión territorial.

A diferencia de otras pruebas realizadas de reconocimiento, se ha elegido un jeroglífico procedente de un escarabajo. Dichos escarabajos podían ser utilizados en la envoltura de las momias o bien como elementos de propaganda política.

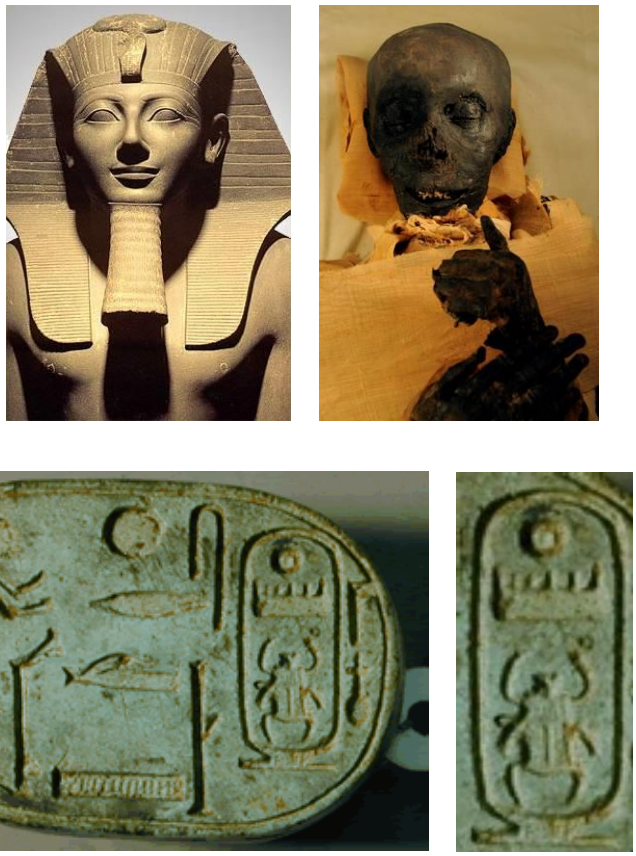


Ilustración 75: Faraón Tutmosis III y escarabajo para reconocimiento

Durante las pruebas se detectó que los jeroglíficos MN y JEPER presentaban diferencias significativas con la lista real de Abydos. Tras incorporarlos a la base de datos de reconocimiento, y reduciendo el umbral de reconocimiento de jeroglíficos al 60% (lo normal es 75%) el reconocimiento fue correcto. Mostramos a continuación los resultados obtenidos. Los pasos son los mismos que se han explicado en el faraón Teti.

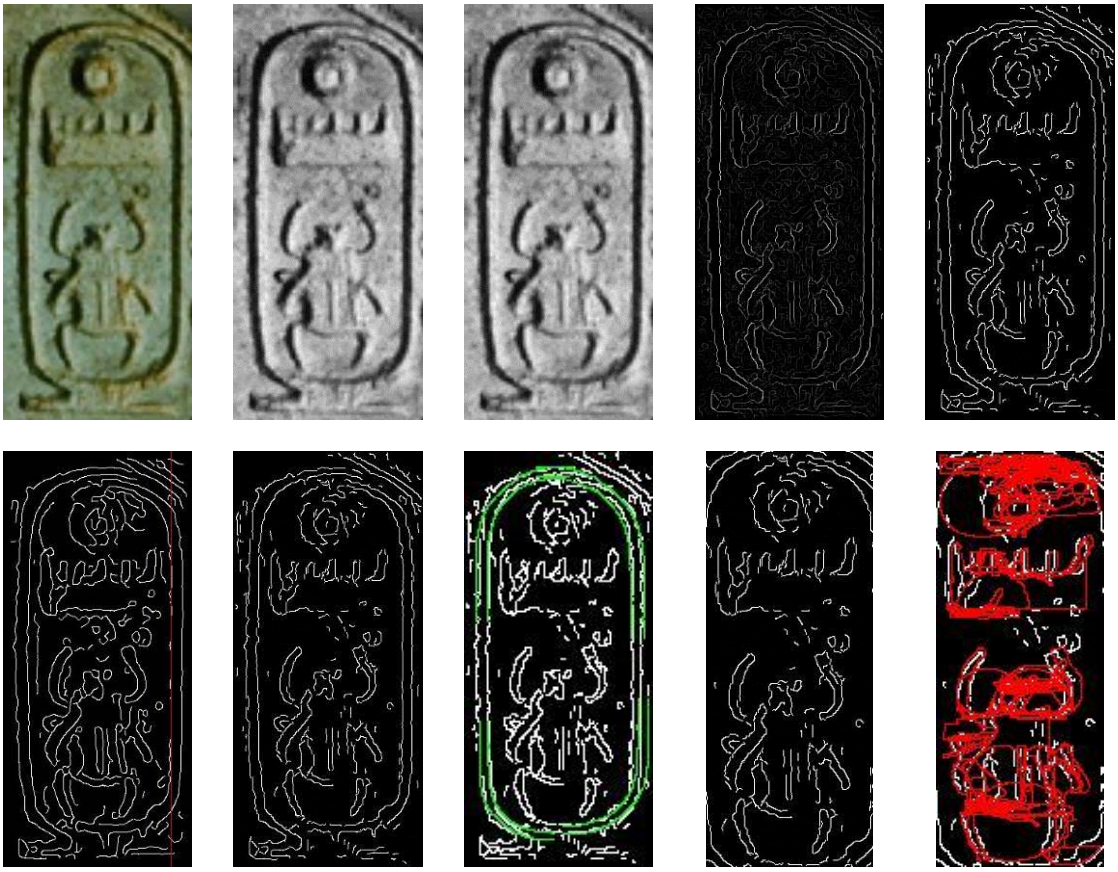


Ilustración 76: Secuencia de reconocimiento del cartucho de Tutmosis III

La pantalla de la aplicación devolvió el siguiente resultado, que como podemos ver es correcto ante una coincidencia de jeroglíficos del 60% (normalmente es el 75%):

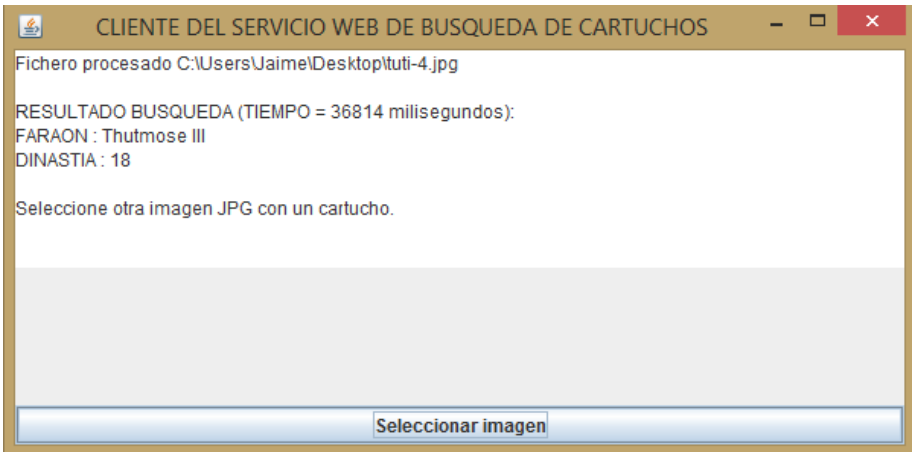


Ilustración 77: Resultado de la detección de Tutmosis III

6.5. FARAÓN DJEDFRA

Dyedefra fue el tercer faraón de la dinastía IV de Egipto. Reinó aproximadamente entre el año 2566 a.C. y el 2557 a.C.

Ilustración 78: Faraón Dyedefra

El trabajo devuelve los siguientes resultados (ante una coincidencia mínima del 75% de los jeroglíficos). Los pasos son los mismos que se han explicado en el faraón Teti.

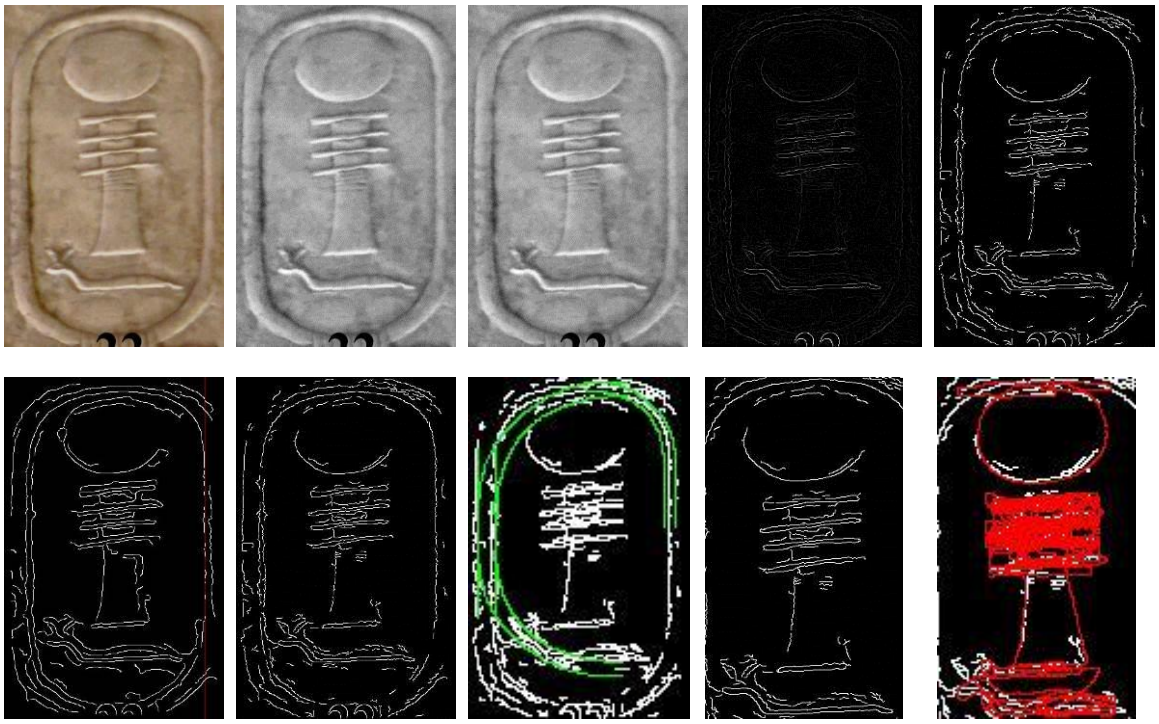


Ilustración 79: Secuencia de reconocimiento del cartucho de Dyedefra

La pantalla de la aplicación devolvió el siguiente resultado, que como podemos ver es correcto:

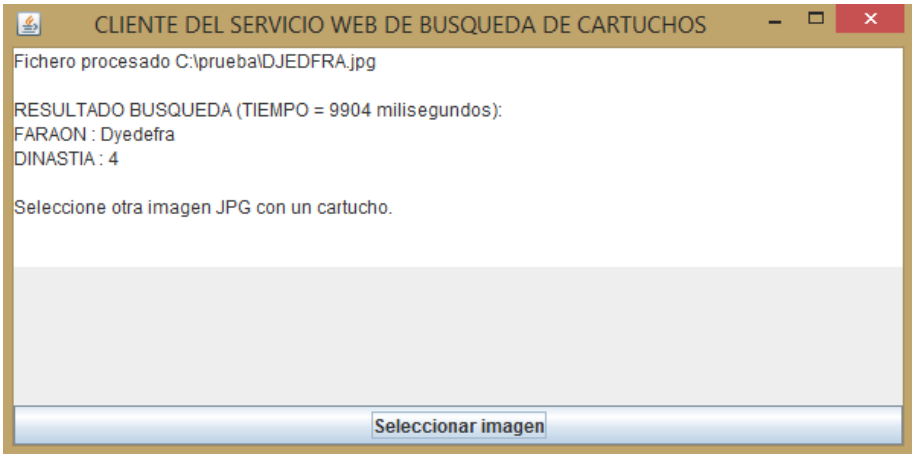


Ilustración 80: Resultado de la detección de Dyedefra

6.6. FARAÓN SANAJT

Sanajt o Nebka (2682 a.C. a 2665 a.C.) fue el primer faraón de la dinastía III de Egipto1 Con él comienza el periodo denominado por los modernos historiadores Imperio Antiguo de Egipto. El nombre Sanajt significa 'el fuerte protector'. También es conocido como Nebka (su nombre de trono).

Ilustración 81: Faraón Sanajt (Neb-Ka)

El trabajo devuelve los siguientes resultados (ante una coincidencia mínima del 75% de los jeroglíficos). Los pasos son los mismos que se han explicado en el faraón Teti.

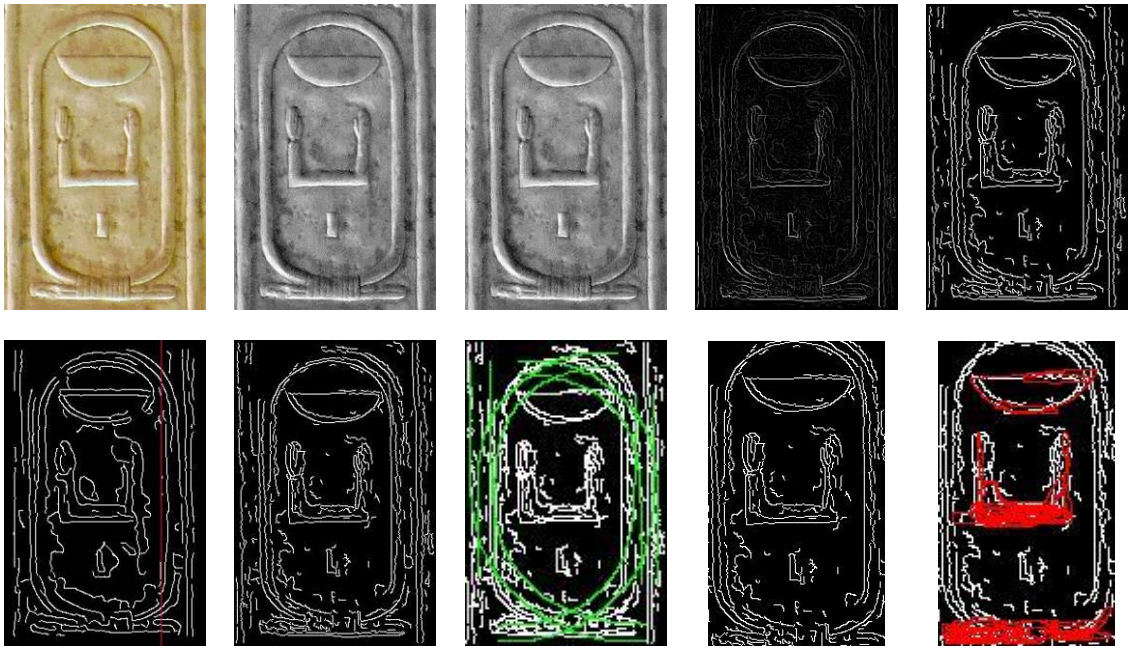


Ilustración 82: Secuencia de reconocimiento del cartucho de Sanajt (Neb-Ka)

La pantalla de la aplicación devolvió el siguiente resultado, que como podemos ver es correcto:

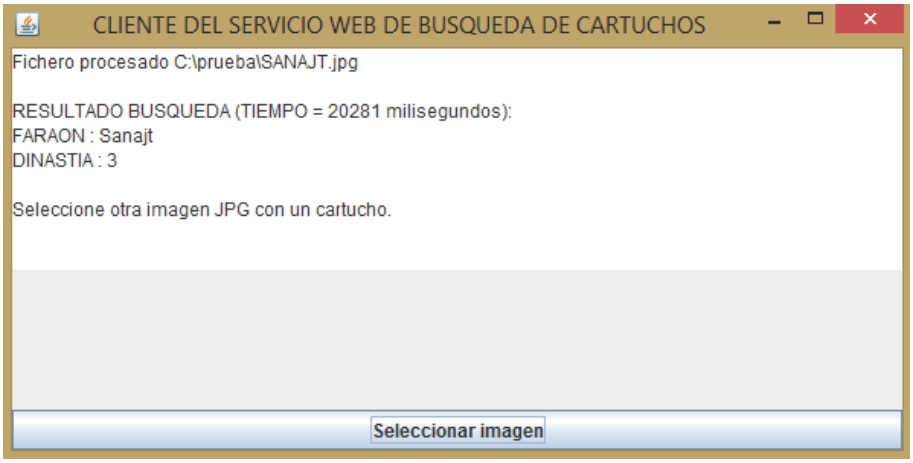


Ilustración 83: Resultado de la detección de Sanajt (Neb-Ka)

6.7. FARAÓN AMENOFIS III

Amenofis III fue un importante faraón de la dinastía XVIII de Egipto que gobernó entre los años 1390 a.C. y 1353 a.C. 4 Sucedió a su padre, Tutmosis IV, y fue el padre del conocido faraón Amenofis IV, también llamado Akenatón. Akhenatón, el faraón hereje, es considerado el padre del monoteísmo mundial, ya que rompió el culto a todas las divinidades egipcias para centrarlo en un único Dios, Atón.

Ilustración 84: Faraón Amenofis III en el Museo Británico, y su templo (coloso Memnón)

El trabajo devuelve los siguientes resultados (ante una coincidencia mínima del 75% de los jeroglíficos). Los pasos son los mismos que se han explicado en el faraón Teti.

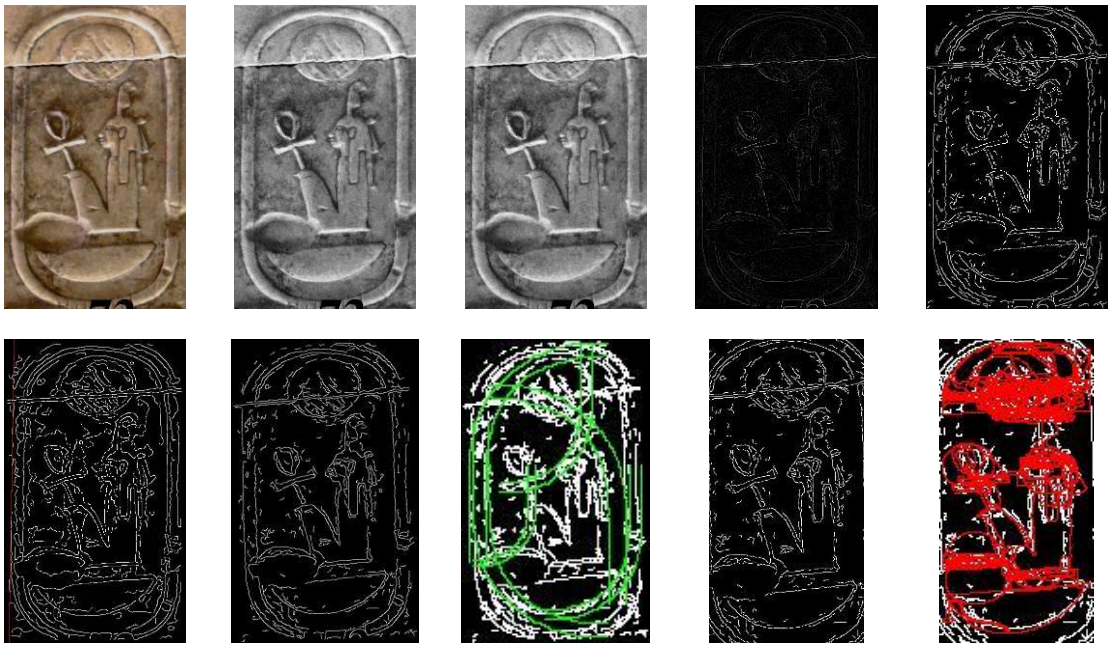


Ilustración 85: Secuencia de reconocimiento del cartucho de Amenofis III

La pantalla de la aplicación devolvió el siguiente resultado, que como podemos ver es correcto:

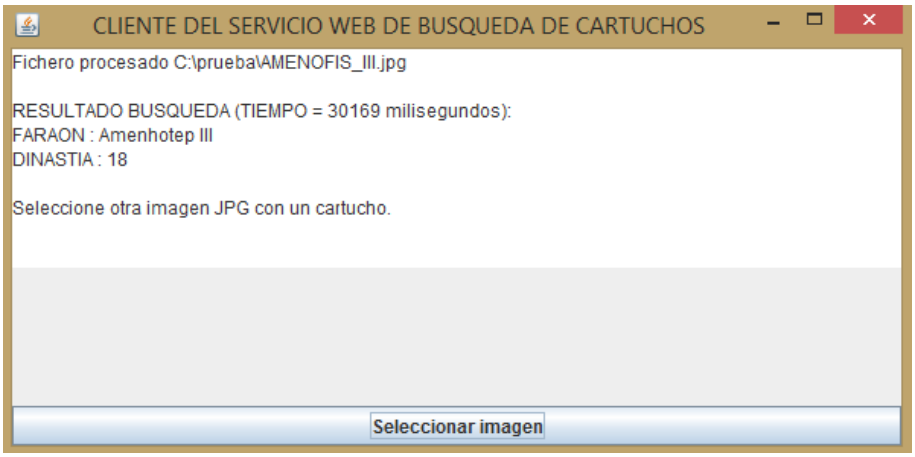


Ilustración 86: Resultado de la detección de Amenofis III

6.8. OTRAS APLICACIONES PROBADAS

El algoritmo de búsqueda fue probado con relativo éxito en otro proyecto de búsqueda de marcas de cantero, si bien dicha búsqueda no debía realizarse sobre imágenes muy grandes con elementos pequeños. Ante la búsqueda de las 4 marcas siguientes:

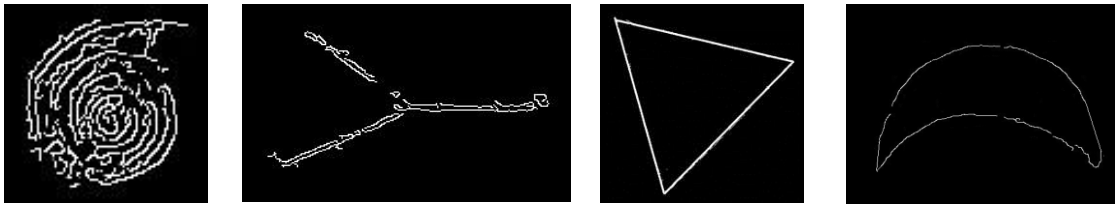
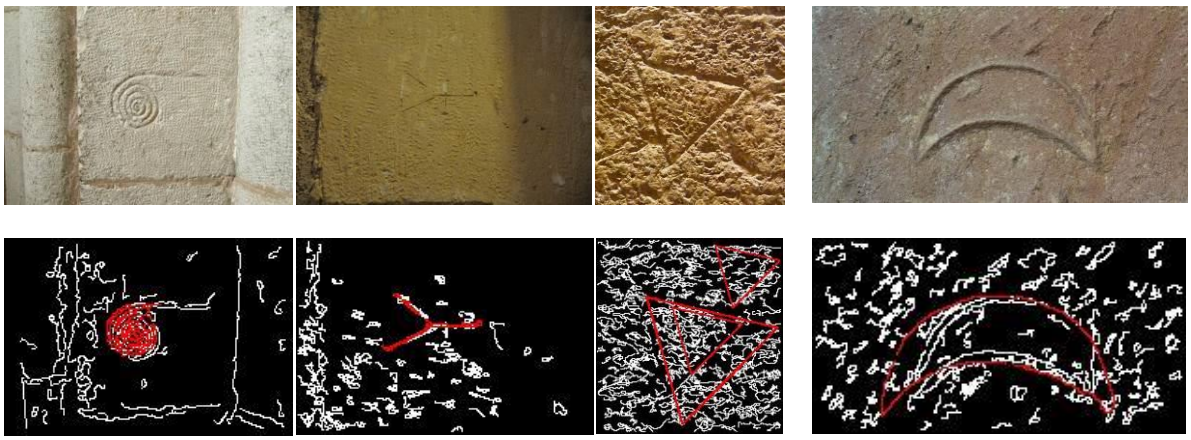


Ilustración 87: Marcas de cantero a buscar

Ante un umbral de búsqueda del 70%, el algoritmo devolvió los siguientes resultados sobre las siguientes imágenes:



Marca espiral | Marca con triángulo y cruz | Marca triangular | Marca con forma de luna

Ilustración 88: Búsqueda de marcas de cantero sobre imágenes

Podemos ver que en la imagen de los triángulos, ante el excesivo ruido devolvió más de un triángulo. En el resto de imágenes el reconocimiento fue correcto.

Conclusiones

Este trabajo de investigación recoge un estudio de distintos métodos y técnicas destinadas al reconocimiento de jeroglíficos egipcios. Se ha desarrollado un nuevo método de detección de objetos en una imagen que tiene en cuenta distintos factores como son distancias entre puntos, los ángulos de un borde en cada punto, y la estructura que estos bordes poseen.

El trabajo une dos áreas de conocimiento diferentes como son la Egiptología y la Visión Artificial con la idea de ofrecer un programa con una posible utilidad dentro del reconocimiento de nombres de faraones. Con ciertas variaciones del algoritmo, y con un aumento de la base de datos, se podría extender el algoritmo para utilizarlo en el reconocimiento de textos de carácter general.

El algoritmo ha tenido que ser revisado y optimizado múltiples veces con el fin de ir perfeccionando el reconocimiento. El sistema gestiona una base de datos de conocimiento inicial que puede ser fácilmente ampliable añadiendo nuevos jeroglíficos y cartuchos. Si un jeroglífico cambia significativamente entre dos cartuchos diferentes será necesario incluirlo varias veces en la base de datos.

Las pruebas arrojan el buen comportamiento ante cartuchos conocidos, que no se encuentren rotos, y que posean jeroglíficos con relativa similitud a los de la base de datos. Además el algoritmo es rápido al trabajar con imágenes de un tamaño reducido.

Uno de los posibles puntos a mejorar sería la utilización de algún algoritmo distinto a la distancia de Levenshtein para el reconocimiento del jeroglífico completo. El motivo de esto es que algunos jeroglíficos concretos nos indican que representan a un reducido grupo de cartuchos. Si encontramos dichos jeroglíficos con suficiente probabilidad, podríamos asumir que se trata de alguno de dichos cartuchos con lo que evitaríamos posibles errores de identificación si por ejemplo el algoritmo de Levenshtein lo rechaza al considerar que eliminándolo se llega a otro nombre más próximo. Una idea sería la utilización de redes neuronales que asignaran pesos diferentes a cada jeroglífico.

Ha sido un trabajo de investigación complejo y lleno de dificultades. Cada vez que se ajustaba el algoritmo se detectaban desviaciones en el reconocimiento de algunos cartuchos que obligaban a modificar de nuevo el algoritmo hacia una versión más perfeccionada. Además, ciertas modificaciones obligaban a estudiar nuevas funcionalidades del algoritmo. Por ejemplo cuando se vio la necesidad de poder mantener distintas variantes del mismo jeroglífico dentro de la base de datos. No obstante, sin duda alguna, ha sido uno de los trabajos más gratificantes que he desarrollado, ya que se une la afición personal por el mundo antiguo con la vocación personal de la Ingeniería Informática.

Referencias

Abydos description des fouilles, Vol 1. Mariette, Auguste. (Paris, 1869) Plancha 43.

Automatic Egyptian Hieroglyph Recognition by Retrieving Images as Texts. Morris Franken and Jan C. van Gemert, Intelligent Systems Lab Amsterdam (ISLA), University of Amsterdam, 2013.

Distance Regularized Level Set Evolution and its Application to Image Segmentation. C. Li, C. Xu, C. Gui, and M. D. Fox, IEEE Trans. Image Processing, vol. 19 (12), pp. 3243-3254, 2010

Egyptian Grammar. Being an Introduction to the Study of Hieroglyphs. Alan Gardiner. 3rd Ed., Rev. London: Oxford University Press, 1957

Extracting Salient Curves from Images: An Analysis of the Saliency Network. T. D. Alter Ronen Basri. Massachusetts Institute of Technology (MIT), Artificial Intelligence Laboratory, A.I.Memo Nº1550, 1995.

Finding Objects of Interest in Images using Saliency and Superpixels. Tesis Nº4908 Radhakrishna Achanta, École polytechnique fédérale de Lausanne, Faculté Informatique et Communications, Suisse, Laboratoire de communications audiovisuelles. 2011.

How to read Egyptian Hieroglyphs. A step-by-step guide to teach yourself. Mark Collier y Bill Manley. British Museum Press, 1998.

ImageJ User Guide IJ 1.46r. Tiago Ferreira, Wayne Rasband. 2012

Object Reading: Text Recognition for Object Recognition. Sezer Karaoglu, Jan C. van Gemert, and Theo Gevers. Intelligent Systems Lab Amsterdam (ISLA), University of Amsterdam, 2012.

Structural Saliency: The Detection of Globally Salient Structures Using a Locally Connected Network. Shimon Ullman, Amnon Sha'ashua. Massachusetts Institute of Technology (MIT), Artificial Intelligence Laboratory, A.I.Memo Nº1061, 1988.

Visión por computador. Imágenes digitales y aplicaciones. 2ª Edición. Gonzalo Pajares Martinsanz / Jesús M. de la Cruz García. Editorial Ra-Ma. 2007. ISBN 978-84-7897-831-1

Referencias Web

Distintos métodos de Saliency Detection. Huchuan Lu's publication(Selected). IIAU-Lab (Intelligent Image Analysis and Understanding Lab). Prof. Huchuan Lu, Dalian University of Technology (China).
http://ice.dlut.edu.cn/lu/publications.html

Efficient Salient Region Detection with Soft Image Abstraction. Ming-Ming Cheng. University of Oxford, Parks Road, Oxford OX1 3PJ
http://mmcheng.net/code-data/

Image Processing and Analysis in Java

http://rsbweb.nih.gov/ij/

ImageJ Documentation Wiki

http://imagejdocu.tudor.lu/

Java Neural Network Framework

http://neuroph.sourceforge.net/

Modelos Flexibles de Visión por Computador (AAM). Universidad de Manchester

http://www.isbe.man.ac.uk/~bim//Models/index.html

Saliency Map Algorithm : MATLAB Source Code. Itti, Koch, Niebur algorithm
http://www.vision.caltech.edu/~harel/share/gbvs.php

Structural Salience: Extracting salient curves from images. Department of Computer Science and Applied Mathematics (Weizmann Institute of Science)
http://www.wisdom.weizmann.ac.il/~vision/SaliencyNetwork/

Structural Saliency: The Detection of Globally Salient Structures Using a Locally Connected Network. Shimon Ullman, Amnon Sha'ashua

http://robotics.stanford.edu/~shashua/