UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE INFORMÁTICA

Sistemas Informáticos 2014/2015

IDENTIFICACIÓN Y CLASIFICACIÓN DE MARIPOSAS CON TÉCNICAS DE VISIÓN COMPUTARIZADA

Tristán Adolfo Nicolás Puppo

Profesor director: Carlos Gregorio Rodríguez

Declaración de conformidad

El alumno:

Tristán Adolfo Nicolás Puppo aquí firmante, autoriza a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado.

Madrid, 1 de Junio de 2015

Tristán Adolfo Nicolás Puppo

RESUMEN

Estando a las puertas de finalizar los estudios y tras un Erasmus muy fructífero, la necesidad de buscar un tema para desarrollar un proyecto de fin de carrera me llevó a un Trabajo Académicamente Dirigido (TAD) comenzado por una alumna de la Licenciatura en Matemáticas, inconcluso, sobre visión computarizada con aplicaciones a la biología, en el campo de las mariposas. Dicho proyecto carecía de un desarrollo a fondo de la interfaz gráfica, y de otros objetivos que creemos haber completado, gracias a nuevo material disponible, como son el haber podido procesar un mayor número de muestras, conseguir resultados más reales y automatizar casi al 100% el proceso de interpretación de las muestras.

La visión computarizada o también conocida como visión artificial es un nuevo campo de investigación cuyo objetivo principal es que el ordenador entienda lo que ve a través de la programación; en nuestro caso le enseñamos a interpretar imágenes de mariposas. Esta nueva disciplina tiene muchos posibles campos de aplicación a parte de la biología, como por ejemplo en medicina, video-juegos, robótica, procesos industriales, etc. y por tanto muchos nuevos desafíos y problemas por resolver.

Era fundamental conocer el campo de la visión artificial, y más concretamente, trabajar con técnicas y procedimientos básicos sobre procesamiento de imágenes para poder plantear y resolver el objetivo del proyecto: crear una base de datos de mariposas inteligente, que al introducir una nueva muestra, fuese capaz de reconocer, a través de técnicas de visión computarizada, si había mariposas parecidas o iguales a la misma.

Palabras clave: visión computarizada, inteligencia artificial, tratamiento de imágenes, interpretar, OpenCV

ABSTRACT

After a productive Erasmus study period abroad and upon my return to Madrid to complete my degree, for my Final Project I chose to develop a TAD (Trabajo Academicamente Dirigido) which had been initiated by a student in the Mathmatics Degree Program, but remained unfinished. This TAD involved a computerized version with applications in the field of biology, specifically, in the study of butterflies. It lacked an in-depth graphic interface and several other objectives were also incomplete as well. The Project presented here has completed these aspects thanks to the availability of new material and strategies such as having been able to process a greater number of samples, obtain more real results and automate almost 100% of the sample interpretation process.

Computer vision, also known as artificial vision, is a new field of research in which computers are programmed to process, analyze and understand images. In our case the computer is taught to interpret images of butterflies. This new field has many possible applications besides those of biology such as in medicine, video games, robotics, industrial processes, etc. Thus new challenges and have arisen and must be solved.

The objective of the Project was to create an intelligent data base of butterflies which, when faced with a new sample, would be capable of recognizing through computer vision technology whether similar or same butterflies were already included in the data base. To fulfill this objective it was essential to become familiar with the field of artificial vision, and specifically, with basic techniques and procedures of image processing.

Keywords: computer vision, artificial intelligence, image processing, interpreting, OpenCV

ÍNDICE

1. Introducción

1.1 Motivación del proyecto

Sobre una base de datos con imágenes de mariposas deseamos encontrar un método que nos facilite la tarea de ver si una nueva mariposa la tenemos ya clasificada o, por el contrario, todavía no se encuentra dentro de la colección. Para ello, necesitamos discriminar las imágenes de diferentes formas y así seleccionar las más parecidas a la nueva, para que de esta manera, sea más fácil tomar la decisión de si está clasificada o no, ya que tendremos que fijarnos en un número muy reducido de ejemplares en vez de tener que mirar todas una por una.

1.2 Objetivo del proyecto

El objetivo del trabajo es crear un programa que, utilizando la imagen de la nueva mariposa, vaya comparando esta con cada una de las que tenemos ya guardadas en nuestra base de datos y que decida, según diferentes criterios, si son suficientemente parecidas y es seleccionada, o no lo son y es descartada, obteniendo como resultado el conjunto de todas las seleccionadas, es decir, las más semejantes a la que deseamos añadir.

Posteriormente, tras realizar un estudio de las muestras podremos recorrer todas las mariposas presentes en las colecciones, con la posibilidad de estudiar y comparar cada una de ellas.

1.3 Muestras

Las imágenes que vamos a utilizar en el trabajo están sacadas de la página web de Proyecto Mariposa (proyectomariposa.net), que es un proyecto cuyo objetivo es crear una base de datos de la biodiversidad de las mariposas diurnas de Colombia y testar la utilidad del código de barras genético. Estas muestras ya estaban tomadas antes de comenzar nuestro estudio y están pensadas para el tratamiento humano y no automático, por ello, para facilitar este cambio, de todas las imágenes que están disponibles vamos a trabajar solo con las que tienen debajo una escala métrica (véase Figura 1.1).

Figura 1.1: Escala métrica

Las muestras tienen, por tanto, un ejemplar de mariposa en la parte superior que está encuadrado por un color-check a la izquierda, que nos indicará la temperatura del color, y debajo por la escala métrica de 0 a 3 cm, que nos indicará el tamaño. (Véase Figura 1.2).

Figura 1.2: Imagen de una de las muestras.

1.4 Solución

Los principales pasos que hay que llevar a cabo son:

Preprocesar las imágenes, reescalando todas las muestras, para que los resultados de los análisis posteriores sean más reales.

Encontrar la silueta de la mariposa para poder aislarla del fondo, llegando a obtener una 'máscara', en la que los puntos del cuerpo de la mariposa sean blancos y el resto negros.

Comparar las muestras, utilizando información sobre el color y algunas propiedades características de la forma. Para que los datos que analizamos sean solo de la mariposa utilizaremos la máscara (Véase Figura 1.3).

Figura 1.3: Imagen de una de las muestras.

1.5 Organización de la memoria

Tras el resumen previo, junto con la motivación y los objetivos del proyecto expuestos previamente, se describe a continuación brevemente la organización de la memoria.

En el documento se pueden encontrar una parte principal donde se explica el desarrollo del proyecto y varios apéndices.

La parte principal consta de varios capítulos:

- Capítulo 2: Fase de análisis, donde se detalla el diseño y la arquitectura general del desarrollo del software, así como los requisitos del sistema.

- Capítulo 3: Fase de diseño, donde se explican las tecnologías utilizadas para desarrollar el proyecto, así como diagrama de clases y casos de uso.

- Capítulo 4: Fase de implementación. Primera toma de contacto con las muestras y preprocesamiento. Se explican los pasos seguidos para encontrar en la imagen la escala métrica. Encontrar y medir esta escala en la imagen, es esencial para poder hacer un reescalado de las imágenes y así una comparación real de tamaños y formas. También, la comparación por color depende de este reescalado, pues para caracterizar el color de una mariposa se usan histogramas que miden las cantidades de los colores desde forma absoluta. Este capítulo explica en detalle cómo funciona la comparación de imágenes de mariposas por forma y color.

- Capítulo 5: Aplicación final e interfaz. Se comenta cómo juntando los distintos programas que teníamos se consigue la aplicación final con explicación gráfica de cada botón.

- Capítulo 6: Fase de Pruebas. Se exponen un par de ejemplos y los resultados obtenidos para una mariposa.

También hay tres apéndices.

- Apéndice A: Se explican los principales conceptos sobre el tratamiento de imágenes que son necesarios para entender y poder seguir el documento. A lo largo del texto se hacen múltiples referencias a este apéndice para que se pueda entender el funcionamiento de los métodos que se utilizan. Se divide principalmente en tres partes:
    1. Filtrado de imágenes.
    2. Análisis de imágenes.
    3. Contornos.

- Apéndice B: El resultado final del trabajo es una aplicación escrita en el lenguaje de programación Python y por ese motivo en este apéndice se puede encontrar parte del código desarrollado.

1.6 Nomenclatura y notación

Antes de continuar vamos a explicar la notación que va a ser utilizada.

- Vamos a nombrar muchas funciones de la librería OpenCV. Esta es una librería de visión computarizada destinada al tratamiento de imágenes y visión por ordenador en tiempo real (ver A). Estas funciones ya están implementadas. Para nombrarlas utilizaremos la letra MingLiU, por ejemplo, `cv2.Canny`. Algunas de ellas son muy importantes y se utilizan muchas veces a lo largo del proyecto por ello es importante entender bien su

funcionamiento. De estas se pueden encontrar explicaciones más detalladas y algunos ejemplos de utilización en el apéndice A. Tras nombrarlas indicaremos en que parte del apéndice se pueden encontrar.

- Por ultimo decir que en algunos puntos hay referencias a páginas web donde se puede ampliar información.

2. Fase de análisis

Este capítulo tiene por objeto la explicación y documentación de las fases típicas de todo proyecto de desarrollo de software, que son las aplicadas en nuestro caso:

- **Fase de requisitos**, en la cual se identificarán los requisitos del sistema
- **Fase de diseño,** en esta fase se detallarán los aspectos más importantes de la arquitectura del sistema
- **Fase de implementación,** en la que se traduce el diseño a un lenguaje de programación (en nuestro caso Python), generación de código e implementación de funciones como tareas principales.
- **Fase de pruebas,** como en todo proyecto, se realizan diversas pruebas software para comprobar si se obtiene el resultado esperado.
- **Fase de mantenimiento,** orientado a mantener la operatividad del software, ya que al utilizarlo como usuario final puede ser que no cumpla con todas las expectativas previstas.

Todas las acciones y tareas que se incluyen y se realizan sobre cada una de estas fases, es lo que se denomina como el ciclo de vida de un proyecto software.

Existen varios modelos para explicar el ciclo de vida de un proyecto de estas características. Uno de los más comunes es el modelo en cascada con retroalimentación entre fases, figura 2.1. Si al terminar una etapa los resultados no son los esperados, podemos volver a la fase anterior. Este modelo es muy flexible y realista. Además, promueve una metodología de trabajo efectiva basada

en los siguientes principios: definir antes de diseñar, diseñar antes de implementar.

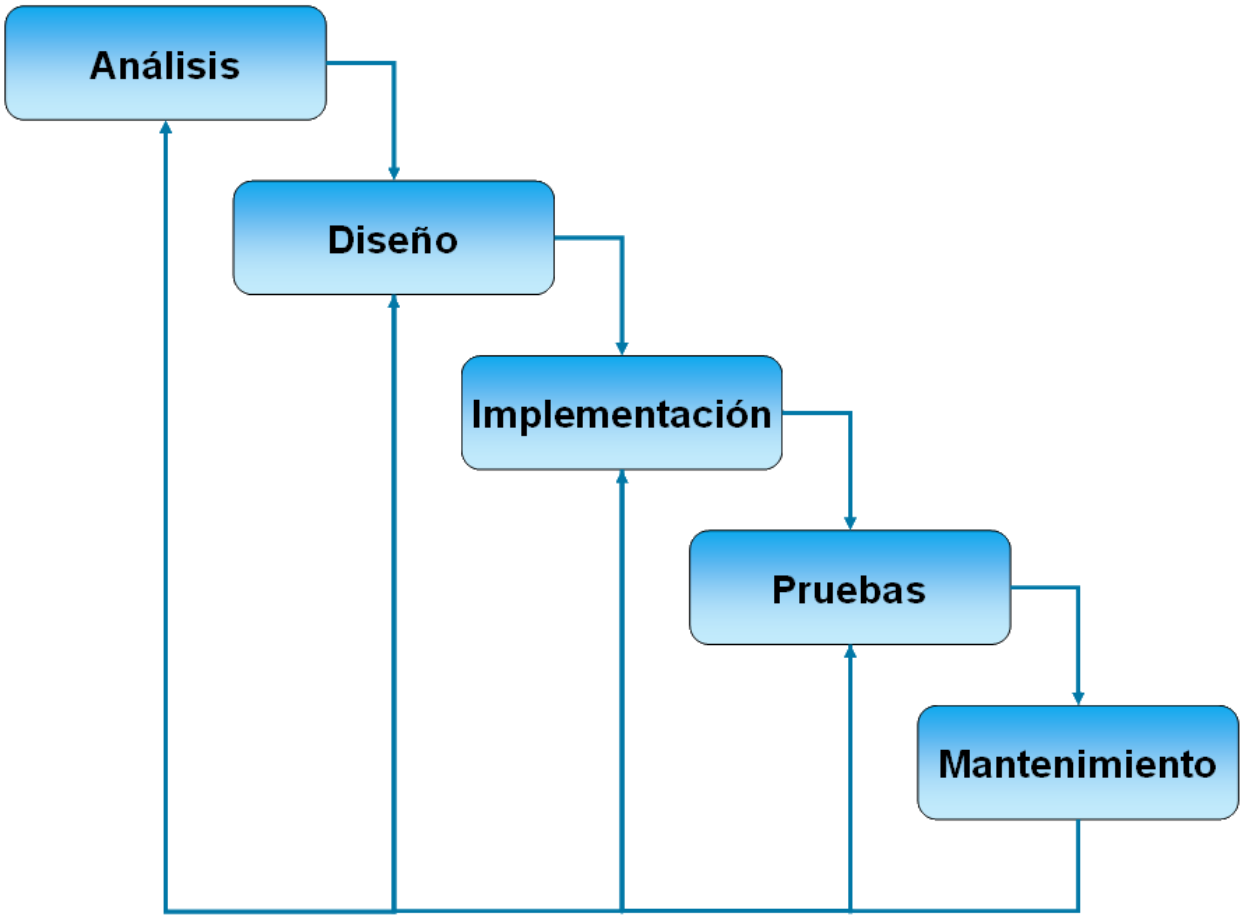

Figura 2.1 Modelo en cascada del desarrollo software con retroalimentación

Los requisitos que se explican a continuación están relacionados directamente con las propiedades que debe tener el funcionamiento de la aplicación software.

Facilidad de uso de la aplicación

Pretendemos que nuestra aplicación sea lo más fácil posible de utilizar prescindiendo de interfaces complicadas y recargadas de botones. Mientras menos botones más facilidad de uso. Además, éstos serán muy intuitivos desde un punto de vista práctico.

Cálculos complejos transparentes al usuario

Pretendemos que la aplicación sea lo más visual posible, por lo que los cálculos o tratamientos de la imagen junto con sus algoritmos deben quedar lo más posible ocultos al usuario. Al usuario final del sistema, sólo le interesa que el programa reconozca si hay mariposas parecidas o no. El cómo se llega al reconocimiento de las mariposas, debe quedar oculto porque no le interesará, además de que complicaría la interfaz y el uso del programa.

Rapidez de reconocimiento

Los algoritmos a usar deben ser lo menos complejos posible, ese es un objetivo importante; lograr una aplicación rápida y ágil en el reconocimiento, para que pueda usar en la medida de lo posible en sistemas en tiempo real con una alta carga de trabajo. El tratamiento de imágenes puede ser a veces muy costoso, por lo que hay que optimizar bien los algoritmos.

Robustez del algoritmo

Este es el gran caballo de batalla de todos los reconocedores de imágenes, en la medida de lo posible el algoritmo debe ser robusto ante posibles deficiencias en la fotografía original, debidas en gran parte a los problemas derivados de la

captura de las imágenes en entornos de exterior. Sin embargo hay que decir que no todas las posibles dificultades son salvables, una muy baja luminosidad o un alto ruido, entre otras cosas, pueden hacer que hasta el mejor reconocedor falle o incluso no funcione.

Modularidad

Es recomendable que el programa esté dividido en módulos más o menos independientes de manera que cada uno de ellos realice tareas específicas y concretas. De esta forma si se necesita mejorar o corregir un aspecto del programa sólo tendríamos que modificar aquellos módulos implicados sin necesidad de intervenir en todo el programa. De esta manera se logra también un mejor mantenimiento del sistema.

Usabilidad

Se trata de que la aplicación pueda ser utilizada por todo tipo de usuarios, desde el usuario experto en la tecnología hasta el usuario más novel.

Bajo coste computacional y de almacenamiento

Está claro que nos interesan unos algoritmos de coste computacional bajo, tanto espacial como temporalmente, ya que al proporcionarle una determinada funcionalidad, en la práctica interesa que los algoritmos se computen lo más rápido posible ocupando el mínimo especio en memoria para poder ejecutar la aplicación de forma efectiva.

3. Fase de diseño

Después de la fase de análisis y determinación de los distintos requisitos que debería tener el sistema a desarrollar, pasamos a la fase de diseño. En esta fase se realiza una propuesta de cómo se deben procesar las muestras, que lógicamente vendrá guiada por la fase de análisis anteriormente expuesta.

La estrategia de la solución de diseño se centra en la creación de un programa capaz de procesar la nueva muestra para obtener todos los datos necesarios para una posterior comparación con el resto de muestras ya presentes en la base de datos.

Se pretende que la solución sea lo más fácil posible de usar, por lo que se intenta automatizar y hacer lo más simple posible las operaciones a realizar por la aplicación, para alcanzar la funcionalidad y solución deseadas.

3.1 Vista Lógica

En lo que se refiere a la vista lógica, la aplicación, presenta dos subsistemas recogidos en un único sistema que proporciona la funcionalidad de los algoritmos para el tratamiento de imágenes digitales.

Por un lado tenemos la "***carga de imagen***", que será una imagen que cargaremos directamente al sistema, ya que como se ha informado anteriormente, la aplicación software no se encarga de la extracción de la foto como tal. Por otro lado, aparece el "***procesamiento de las muestras***", el cual se encargará de reescalar

las mariposas y aislarlas para obtener una máscara que nos facilitará el estudio posterior. Por último, aparece la ***Comparación con la base de datos***; una vez aislada podemos buscar mariposas similares por color y/o forma en la base de datos. En la figura 3.1 se muestra el esquema general de la vista lógica.

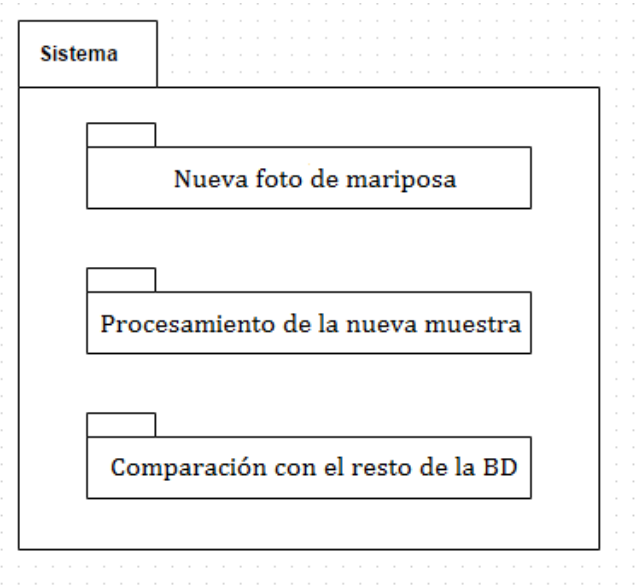

Figura 3.1 Vista lógica

3.2 Diagrama de clases

El diagrama de clases es el diagrama principal para el diseño estático, presentando y describiendo las clases y objetos del sistema con sus relaciones estructurales. En la definición de clases se muestran los atributos y métodos incluidos en la clase correspondiente.

En la figura 3.2 se presenta el modelado de las clases que nos permitirá lanzar la aplicación. Contiene la clase principal *menu.py*, siendo la clase que abarca todas las funcionalidades necesarias para manejar la base de datos (*database.py*) que contiene colecciones (*collection.py*) de mariposas (*butterfly.py*). Además la clase menú utilizará métodos de las clases *get_histogram.py, resize.py y build_mask.py*; por lo que tendremos tres relaciones de uso.

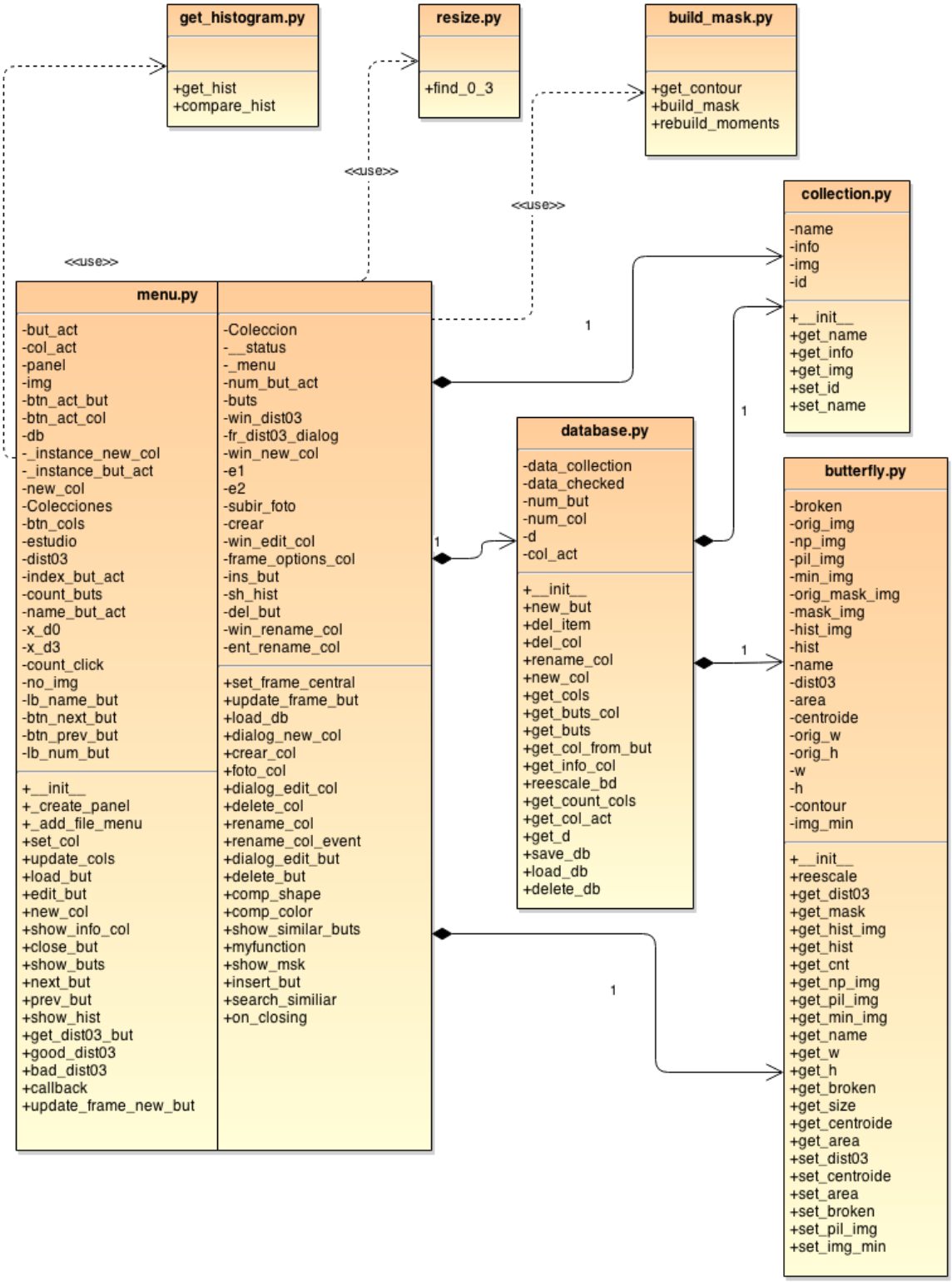

Figura 3.2 Diagrama de clases.

3.3 Casos de uso

Como punto final de la fase de diseño se encuentra el diagrama de casos de uso, el cual describe detalladamente el funcionamiento de la aplicación visto tanto desde la perspectiva del usuario como la del sistema, que se identifican como dos actores presentes en éste.

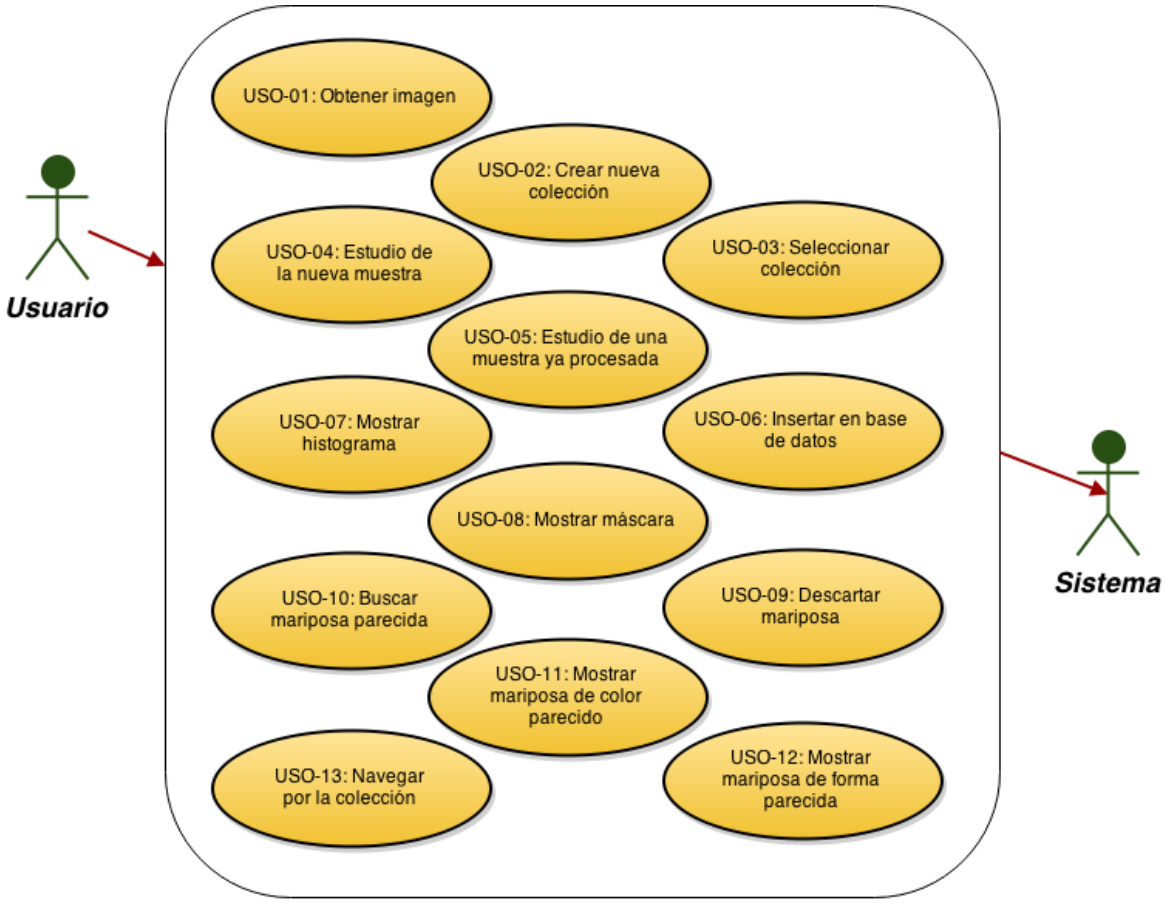

Figura 3.3 Diagrama casos de uso.

CASOS DE USO:

USO - 01: Obtener imagen

Descripción: paso previo al procesamiento, consiste en cargar la imagen de la mariposa.

1. El Usuario abre la Interfaz y pulsa sobre el botón “New Butterfly…”.
2. El Usuario deberá buscar la nueva imagen.
3. El Sistema crea un nuevo objeto mariposa, y calcula su histograma, máscara, contorno e imagen pequeña para mostrar al usuario.
4. El Sistema muestra en el panel central la imagen.
5. Paso al caso de uso siguiente: “Crear nueva colección (USO-02)” o “Seleccionar colección (USO-03)”.
6. El caso de uso termina.

USO - 02: Crear nueva colección

Descripción: las mariposas se clasifican en colecciones y se debe crear una nueva en caso de que no haya ninguna o se quiera añadir una más al sistema.

1. El Usuario pulsa el botón “New Collection…”.
2. El Sistema a través de un cuadro solicita nombre, descripción y foto opcional de la nueva colección.
3. El Usuario pulsa el botón “Crear”
4. El Sistema añade la nueva colección a la Base de datos. Esta colección está vacía.
5. Paso al caso de uso siguiente: “Estudio de la nueva muestra (USO-04)”.
6. El caso de uso termina.

USO - 03: Seleccionar colección

Descripción: Si en la base de datos ya hay colecciones, éstas se muestran apiladas en el panel inferior.

1. El Usuario pulsa cualquiera de las colecciones disponibles.
2. El Sistema actualiza el panel central con la primera mariposa de la colección pulsada. En caso de no tener ninguna mariposa, no muestra ninguna. Si en el panel central hay una mariposa pendiente de estudio, no la descarta a menos que el Usuario cargue una nueva sin haber procesado la anterior.
3. Paso al caso de uso siguiente: "Estudio de la nueva muestra (USO-04)" o "Estudio de una muestra ya procesada (USO-05)"
4. El caso de uso termina.

USO - 04: Estudio de la nueva muestra

Descripción: a través de un menú el sistema brinda una serie de opciones para tratar la nueva imagen:

1. El Usuario pulsa encima de la mariposa que está pendiente de estudio.
2. El Sistema muestra un cuadro con 4 opciones: "Insertar en base de datos (USO-06)", "Mostrar histograma (USO-07)", "Mostrar máscara (USO-08)" y "Descartar mariposa (USO-09)".
3. Paso al caso de uso siguiente: Cualquiera de los cuatro casos anteriores.
4. El caso de uso termina.

USO - 05 Estudio de una muestra ya procesada

Descripción: en cualquier momento podemos también estudiar una mariposa ya presente en la Base de Datos. También tiene su propio menú con opciones disponibles para el estudio de la mariposa actual.

1. El Usuario pulsa encima de la mariposa que ya está procesada.
2. El Sistema muestra un cuadro con 6 opciones: "Buscar mariposas parecidas (USO-10)", "Mostrar mariposas de color parecido (USO-11)", "Mostrar mariposas de forma parecida (USO- 12)", "Mostrar histograma (USO-07)", "Mostrar máscara (USO-08)" y "Descartar mariposa (USO-09)"
3. Paso al caso de uso siguiente: Cualquiera de los seis casos anteriores.
4. El caso de uso termina.

USO - 06: Insertar en base de datos

Descripción: aquí es donde se ponen en marcha los algoritmos más importantes de la aplicación tal como el reconocimiento de la escala métrica en la imagen y el reescalado de toda la base de datos. Cada vez que se añade una nueva mariposa, la Base de Datos se vuelve a reescalar y se recalculan máscaras, perímetros, y contornos.

1. El Usuario pulsa encima del botón "Insertar en base de datos".
2. El Sistema pregunta al Usuario si está seguro de introducir la nueva mariposa en la colección actual.
3. El Usuario pulsa "Sí", para proceder o "No" para abortar.
4. En caso de que el Usuario no aborte el Sistema busca en la imagen la escala métrica y la muestra al Usuario en una ventana aparte.

5. El Usuario supervisa si el algoritmo ha conseguido encontrarla. En caso contrario el Usuario deberá especificar a través de dos clicks el tamaño de la escala métrica, para poder reescalar bien dicha mariposa.
6. El Sistema buscará coincidencias por color y forma para mostrar al Usuario mariposas parecidas, si las hubiese, advirtiéndole con un mensaje de alerta.
7. El Sistema introduce la nueva mariposa en la colección preseleccionada y reescala toda la base de datos.
8. Paso al caso de uso siguiente: "Navegar por la colección (USO-13)", Obtener nueva mariposa (USO-01)", "Crear nueva colección (USO-02)" o "Seleccionar colección (USO-03)".
9. El caso de uso termina.

USO - 07: Mostrar histograma

Descripción: toda mariposa tiene un histograma como atributo que se calcula al principio.

1. El Usuario pulsa encima del botón "Mostrar histograma".
2. El Sistema muestra en una nueva ventana los tres niveles de color (BGR), en un color distinto cada uno y en una gráfica normalizada del 0 al 1.
3. El Usuario puede tener dicho histograma abierto siempre que quiera.
4. El caso de uso termina.

USO - 08: Mostrar máscara

Descripción: toda mariposa tiene una máscara como atributo que se calcula al principio y se va recalculando a medida que se reescala la Base de Datos.

1. El Usuario pulsa encima del botón “Mostrar máscara”.
2. El Sistema en una nueva ventana muestra la máscara en tamaño real.
3. El Usuario puede tener la máscara abierta siempre que quiera.
4. El caso de uso termina.

USO - 09: Descartar mariposa

Descripción: esté o no esté ya procesada, podemos descartar una mariposa en cualquier momento.

1. El Usuario pulsa encima del botón “Descartar mariposa”.
2. El Sistema pregunta al Usuario si está seguro.
3. En caso de que el Usuario pulse “Sí”, se procede a la eliminación del espécimen, en caso contrario, todo queda igual.
4. El caso de uso termina.

USO - 10 Buscar mariposas parecidas

Descripción: este método busca en toda la Base de Datos coincidencias tanto por forma como por color con la mariposa actual.

1. El Usuario pulsa encima del botón “Buscar mariposas parecidas”.
2. El Sistema aplica los dos algoritmos de búsqueda consecutivamente y en una nueva ventana muestra los resultados en caso de que los haya. Si no, advierte que no ha habido éxito en la búsqueda.

3. El Usuario puede pulsar encima de cualquiera de las mariposas del resultado para que pase a ser la mariposa actual y poderla estudiar, o simplemente cerrar la ventana de resultados.
4. El caso de uso termina.

USO - 11 Mostrar mariposas de color parecido

Descripción: este método busca en toda la Base de Datos coincidencias por color con la mariposa actual.

1. El Usuario pulsa encima del botón “Mostrar mariposas de color parecido”.
2. El Sistema aplica el algoritmo de búsqueda y en una nueva ventana muestra los resultados en caso de que los haya. Si no, advierte que no ha habido éxito en la búsqueda.
3. El Usuario puede pulsar encima de cualquiera de las mariposas del resultado para que pase a ser la mariposa actual y poderla estudiar, o simplemente cerrar la ventana de resultados.
4. El caso de uso termina.

USO - 12 Mostrar mariposas de forma parecida

Descripción: este método busca en toda la Base de Datos coincidencias por forma con la mariposa actual.

1. El Usuario pulsa encima del botón “Mostrar mariposas de forma parecida”.
2. El Sistema aplica el algoritmo de búsqueda y en una nueva ventana muestra los resultados en caso de que los haya. Si no, advierte que no ha habido éxito en la búsqueda.

3. El Usuario puede pulsar encima de cualquiera de las mariposas del resultado para que pase a ser la mariposa actual y poderla estudiar, o simplemente cerrar la ventana de resultados.
4. El caso de uso termina.

USO - 13 Navegar por la colección

Descripción: en cualquier momento se puede navegar hacia delante y hacia atrás entre las mariposas de una colección.

1. El Usuario pulsa encima de los botones ◄ o ► para avanzar o retroceder dentro de la colección.
2. El Sistema simplemente actualiza el panel central con la nueva mariposa que toque. De la última pasará a la primera.
3. El caso de uso termina

4. Fase de Implementación

4.1 Descripción

En esta fase se trata todo lo relacionado con la implementación de la aplicación, y se verán las técnicas que hemos usado para la detección de la escala métrica, el reescalado así como el aislamiento de la mariposa y el cálculo de contornos. También veremos cómo funcionan los algoritmos para buscar coincidencias dentro de la base de datos por color y forma. Todo este proceso de implementación se adecúa a todo lo desarrollado en la fase de análisis y en la fase de diseño.

4.2 Procesando las muestras

Las imágenes obtenidas de la base de datos de Proyecto Mariposa están a distinta escala, por tanto, es necesario llevar a cabo un proceso previo de reescalado para hacer que todas queden igual y no obtener datos falsos en las posteriores comparaciones.

4.2.1 Encontrando la escala métrica

La primera estrategia que probamos para intentar encontrar la posición de la escala métrica, fue guardando una imagen de una escala métrica aislada y probar con `cv2.matchTemplate`. Lo que hace este método es, dado un patrón de tamaño `h x w`, va desplazándolo sobre la imagen de tamaño `H x W` y comparándolos pixel a pixel.

Aplicándolo sobre nuestras imágenes observamos que funcionaba muy bien, solo en las imágenes en la que la escala métrica era parecida, pero como hay varios tipos de escalas métricas en nuestra base de datos decidimos buscar otro método.

Aplicando varios métodos de detección de bordes y esquinas dimos con el método `cv2.cornerHarris()` también de la librería OpenCV. La lógica a seguir era encontrar la esquina situada en la parte más inferior y a la izquierda para localizar el cero y en la parte más inferior y a la derecha para localizar el 3. Así pues ya tenemos la distancia en pixeles de la escala métrica.(B.1) (Véase Figura 4.1)

Figura 4.1: Ejemplo. A la izquierda se ven todas las esquinas detectadas y a la derecha las dos que nos interesan.

Este método resultó ser eficaz, pero en ocasiones falla. Por eso, la interfaz nos da la opción de seleccionarlo manualmente, como veremos más tarde en el capítulo 6.2.

4.2.2 Reescalando la imagen

Una vez encontradas las escalas métricas en cada imagen, hay que llevar a cabo el reescalado. Hacer este reescalado consiste en encontrar la escala métrica en cada imagen y hacer que ésta tenga el mismo tamaño en cada una de ellas (véase Figura 4.2), así, 3cm de los que marcan las reglas de las imágenes va a equivaler al mismo número de píxeles en cada una de ellas.

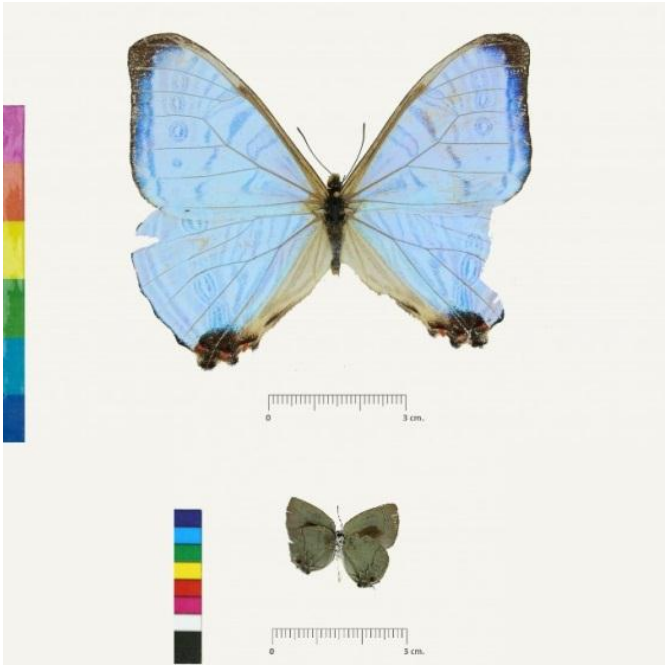

Figura 4.2: Las escalas métricas tienen el mismo tamaño; las dos mariposas están reescaladas.

Para intentar perder la menor información posible sobre las imágenes, principalmente del color, el nuevo valor de alto y ancho elegido es la media de los altos y anchos de cada imagen. Dicho proceso se realiza cada vez que se añade una nueva mariposa. Por ello, cada objeto mariposa dispone de una imagen original, que es el que se reescala y así también evitamos perder calidad en cada reescalado.

4.2.3 Aislando la mariposa.

Una vez que pudimos conseguir tener todas las imágenes guardadas a la misma escala, había que encontrar una máscara o contorno de la mariposa para tenerla aislada, es decir, sin influencia de los valores del fondo ni de ningún otro elemento sobre los datos que se obtuvieran.

Esta función se usa normalmente para obtener una imagen binaria (sólo pixeles blancos y negros) a partir de una imagen en escala de grises. Al aplicar la función `cv2.threshold()` en una imagen, obtenemos la mariposa, el color-check y la escala métrica (Véase Figura 4.3) perfectamente aislados.

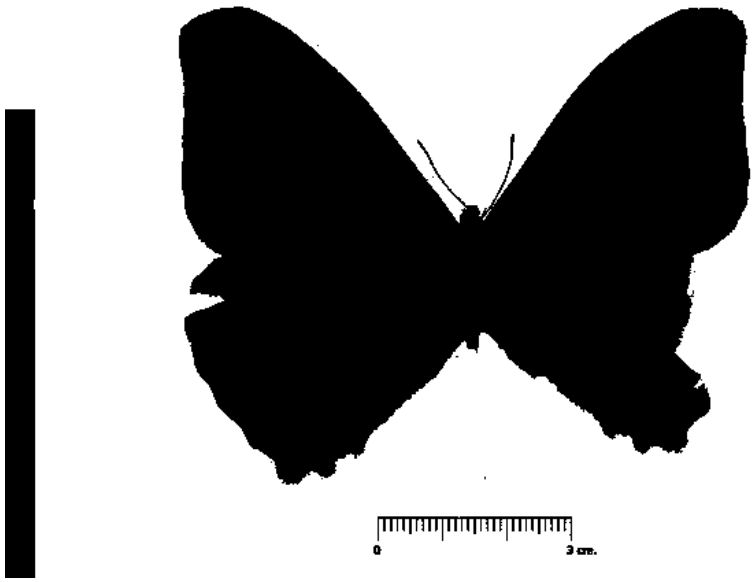

Figura 4.3: Imagen al aplicarle Thresholding.

4.2.4 Seleccionando el contorno

El objetivo era simplemente eliminar el color-check y la escala métrica de la imagen obtenida anteriormente.

Usando la función cv2.findContours() conseguimos una serie de contornos que bordean los tres elementos de nuestra imagen. La dificultad era saber cuál de ellos era el que nos interesaba.

La función cv2.arcLenght() nos daba la longitud de cada contorno. El más grande lo eliminamos directamente pues era el recuadro de la imagen. El siguiente contorno más grande tenía que ser el de la mariposa, si además al calcular su centroide coincidía que estaba en el centro de la imagen.

En cuanto tenemos el contorno acertado solo queda rellenarlo e invertir los colores para que nos quede la máscara de nuestra mariposa. Figura 4.4. Véase también B.2.

Figura 4.4: Máscara, tras aplicar thresholding e inversión de colores.

Uniendo todo, se obtiene el programa build_mask.py que hace lo que deseábamos:

- Conseguir una máscara limpia de cada mariposa.

4.3 Comparando las imágenes

Recordemos que el objetivo era que por cada nueva imagen el programa nos muestre las más parecidas a ésta utilizando criterios basados en el color y la forma de los ejemplares de las muestras. Por tanto, una vez que ya tenemos todas las imágenes a la misma escala y hemos encontrado la máscara de cada una de ellas, podemos empezar entonces a realizar dichas comparaciones.

4.3.1 Por color

Las comparaciones por color las hemos hecho utilizando histogramas (ver A.2.1) (véase Figura 4.5). Lo que hicimos fue utilizar la máscara para restringir la zona de la imagen sobre la cual se calcula el histograma. Después normalizarla (hacer que todos los valores varíen entre 0 y 1). Al haberlo normalizado se seleccionan mariposas con los mismos colores independientemente de si son más grandes o más pequeñas, ya que de momento solo importaba el color (dos mariposas blancas serían seleccionadas como similares independientemente de la forma o tamaño. En el estudio de la forma ocurre lo contrario, por este motivo es conveniente juntar los dos tipos de análisis).

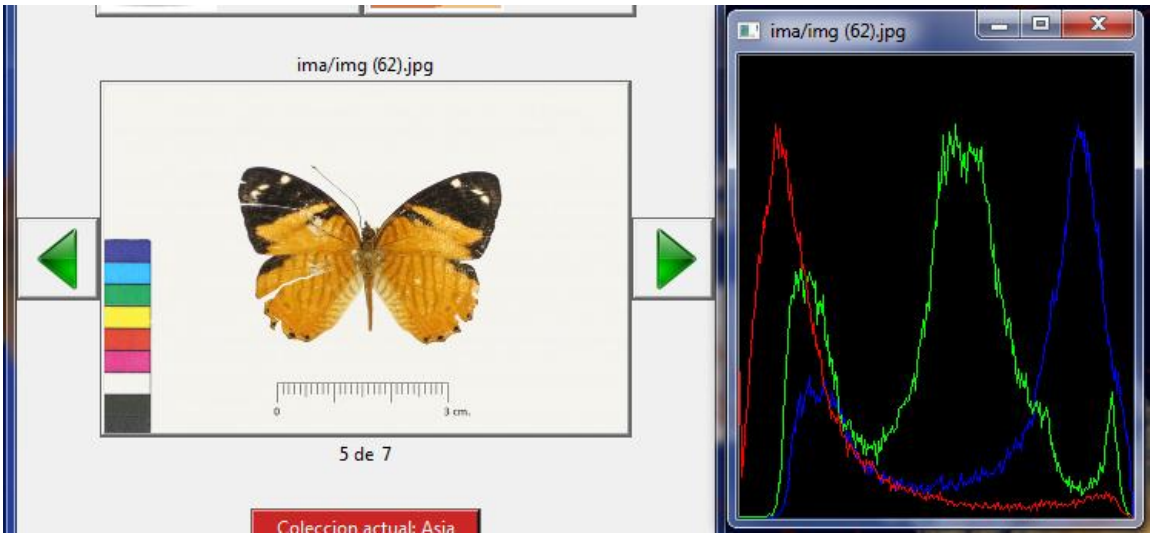

Figura 4.5: Muestra con histograma sobre cada capa.

Cada vez que introducimos una nueva mariposa, calculamos automáticamente su histograma para que a la hora de compararlas por color, estén todos calculados. Esto lo llevamos a cabo utilizando cv2.compareHist() (ver A.2.1).

Para ello lo que hicimos fue dividir la imagen en tres capas (azul, verde, roja) y comparar los histogramas capa por capa. De cada comparación se obtiene un valor que va a indicar cuán parecidas son las imágenes en esa capa, de forma que, si son lo suficientemente parecidas en las tres capas, la imagen es seleccionada y si hay una capa en la que no se parece, se descarta. Esto tiene que ser así porque, en caso contrario, diríamos que una imagen amarilla (con el máximo de rojo y de verde) es parecida a una blanca, ya que también tiene el máximo de rojo y verde y solo se diferenciaría en la capa azul.

Para las comparaciones que se realizan con cv2.compareHist podemos elegir varios métodos (ver A.2.1). En nuestro caso, elegimos el de chi cuadrado (CV COMP CHISQR), es un poco más lento que otros métodos pero un poco más eficaz.

Con el método de la chi-cuadrado, cuanto más próximo es el valor a 0 más parecidas son las imágenes.

El programa que realiza todo esto es `get_histogram.py` (ver B.3) y tiene dos pasos principales:

- Calcular los histogramas de cada imagen utilizando `cv2.calcHist()`.
- Comparar una por una cada imagen con la que deseamos introducir utilizando los histogramas de cada una de ellas.

4.3.2 Por forma

Para las comparaciones básicamente, usamos principalmente los contornos de la mariposa, el área y perímetro (Momentos de Hu) de la máscara de la mariposa.

El método que utilizamos fue basarnos en la función `cv2.matchShapes()` (ver A.3.2). Lo que hace esta función es comparar dos contornos, es decir, dados dos contornos cerrados y la norma que queremos que utilice, devuelve un valor que representa cuán parecidos son. Cuanto más próximo a 0 sea el valor, más parecidos son los contornos. Además utilizamos otras propiedades que son el área y el perímetro del contorno.

Estos valores son muy fáciles de obtener usando las funciones `cv2.contourArea()`, `cv2.arcLength()` que calculan el área y perímetro del contorno.

Juntando ambos métodos, el programa detecta perfectamente mariposas de forma parecida y mariposas de color parecido. (Véase Figura 4.6) Cuando añadimos una mariposa, simplemente aplicamos los dos filtros y si los dos se cumplen, tendremos una coincidencia de mariposa parecida o repetida.

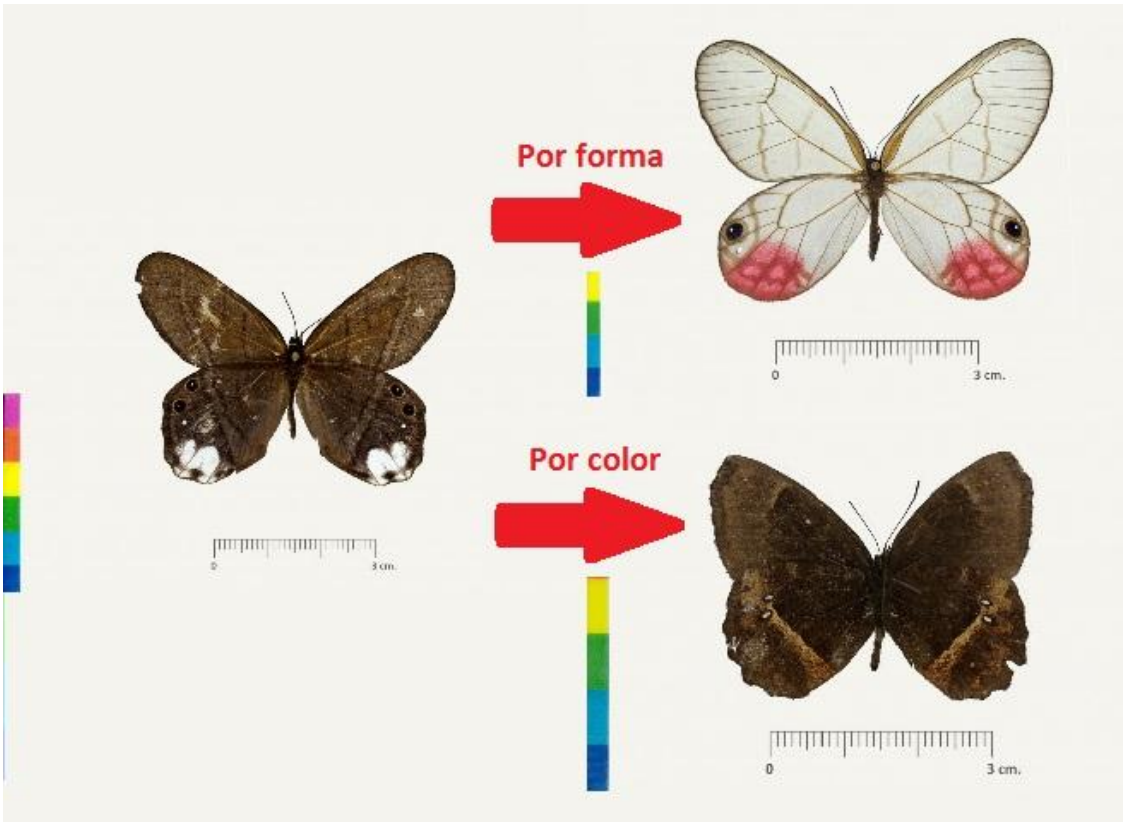

Figura 4.6: Resultado al aplicar los dos métodos a la mariposa de la izquierda.

4.4 Lenguaje de Programación: Python

El hecho de que sea un lenguaje multiplataforma y que a la hora de tratar imágenes sea uno de los lenguajes más utilizados, fue decisivo a la hora de elegirlo como lenguaje de desarrollo. A continuación enumeramos una serie de ventajas:

- **Orientado a objetos**

Facilita mucho el uso de técnicas como la modularidad, abstracción, polimorfismo y encapsulamiento, las cuales resultan muy útiles para los objetivos.

- **Facilidad de aprendizaje**

Python es un lenguaje muy sencillo de aprender y muy manejable para enfrentarse a nuevos retos. Su gran comunidad de programadores a nivel mundial permite ampliar rápidamente conocimientos en el caso de ser necesario ya que hacen que resulte fácil encontrar información y ayuda.

- **Entornos de desarrollo**

En nuestro caso hemos elegido Eclipse por su facilidad a la hora de diseñar interfaces a la vez que resulta ser de amplia difusión.

- **Librerías**

Existe una gran variedad de librerías que facilitan el tratamiento de imágenes y el trabajo de desarrollo. Las que hemos utilizado a lo largo del trabajo han sido OpenCV y PIL para el tratamiento de las imágenes y TKinter para el desarrollo de la interfaz. El poder trabajar con estas librerías facilita enormemente el trabajo de implementación.

4.5 Herramientas de trabajo

El uso de un lenguaje como Python hace que todo lo que rodea el desarrollo de una aplicación basada en él sea relativamente sencillo. Como hemos visto en el apartado anterior, se decidió elegir Eclipse como entorno de desarrollo. Me pareció conveniente y necesario la utilización de un sistema de control de versiones software, comúnmente conocido como GitHub. Aunque podía haberse realizado de forma manual, disponer de herramientas que facilitan esta gestión supone una gran comodidad a la hora de trabajar de manera independiente ya que la sincronización hace que el avance sea ininterrumpido y dinámico. Además permitía a mi Tutor estar al tanto de mis avances, así como en caso de errores no perder trabajo innecesariamente.

El desarrollo documental se ha realizado bajo el control de Microsoft Word 2010. Para la realización de las distintas figuras así como diagramas de clases o casos de uso, utilizamos una aplicación de Google Drive llamada Draw.io (2015).

4.6 Problemas y resolución

Este proyecto resultó para mí un campo totalmente nuevo, especialmente en el tratamiento de imágenes. También me ha aportado el aprender un nuevo lenguaje de programación.

Gracias a tutoriales de OpenCV y de Python que se pueden encontrar en internet, fui avanzando mis conocimientos hasta conseguir una base para afrontar el proyecto.

El desarrollo de una aplicación para tal fin ha supuesto un importante trabajo de investigación, líneas y posibilidades. Esto ha hecho que en ocasiones, el trabajo no se finalizara de forma satisfactoria, haciendo que o bien se desecharan o bien se buscaran soluciones alternativas al respecto. Por ejemplo, partes del programa deberían haber funcionado totalmente automáticas, pero hemos tenido que introducir soluciones en las que el usuario interviene y supervisa que el algoritmo haya trabajado bien.

5. Aplicación Final

Al llegar aquí ya teníamos creados los métodos que seleccionan, dentro de una colección de imágenes de mariposas, las más parecidas a otra dada, fijándonos en el color y la forma. Ahora faltaba unir los métodos y crear una interfaz intuitiva para gestionar la base de datos.

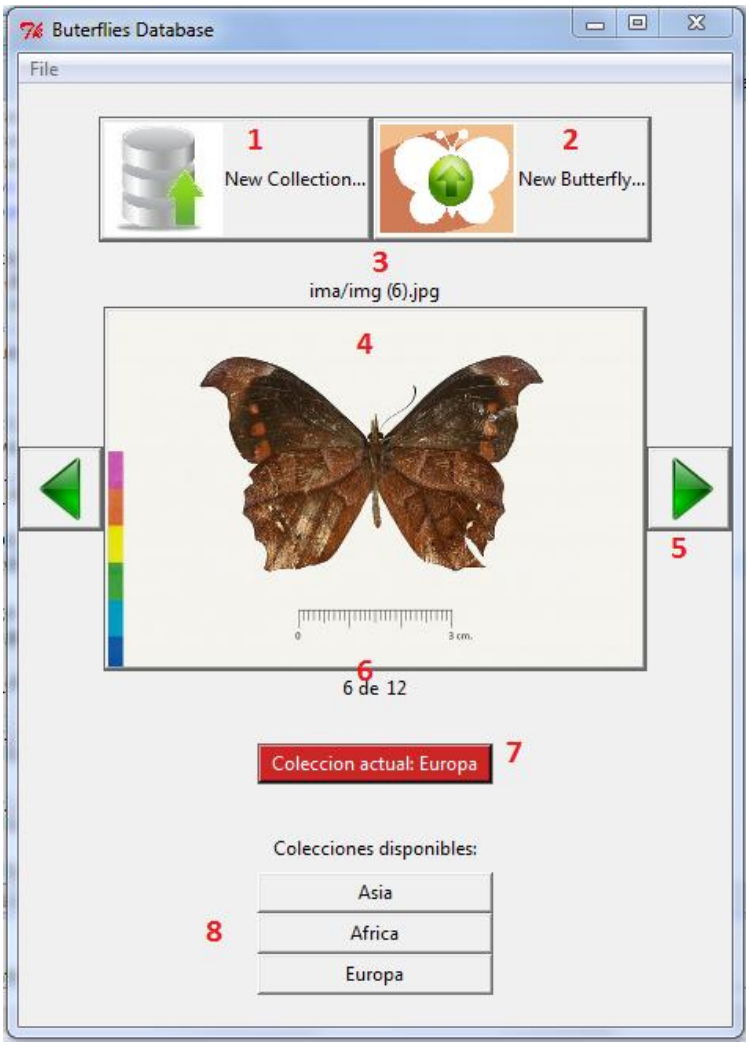

Figura 5.1: Ventana principal de nuestra aplicación.

5.1 Interfaz

Como hemos dicho antes, para el desarrollo de la GUI decidimos utilizar Tkinter, que es un binding de la biblioteca gráfica Tcl/Tk para Python.

La interfaz nos facilita movernos de colección en colección y de mariposa en mariposa. Cada mariposa y cada colección tienen sus menús propios que se muestran al pulsar encima.

1. Botón nueva colección: Al crear una nueva colección deberemos especificar nombre, descripción y una imagen opcional.

2. Botón nueva mariposa: Deberemos buscar la imagen de la nueva mariposa. Se cargará en el panel central para estudiarla.

3. Nombre de la mariposa actual

4. Mariposa actual. Puede estar pendiente de estudio o estar metida en la base de datos. Al pulsar encima se desplegará el menú de mariposa correspondiente (Véase Figura 5.2)

5. Botones para recorrer las mariposas de cada colección hacia delante o hacia detrás.

6. Número de mariposa dentro de la colección.

7. Colección actual: Al pulsar encima se despliega menú de edición para la colección.

8. Colecciones disponibles. Pulsando encima se seleccionan para que pasen a ser la actual y se muestran automáticamente las mariposas de dicha colección.

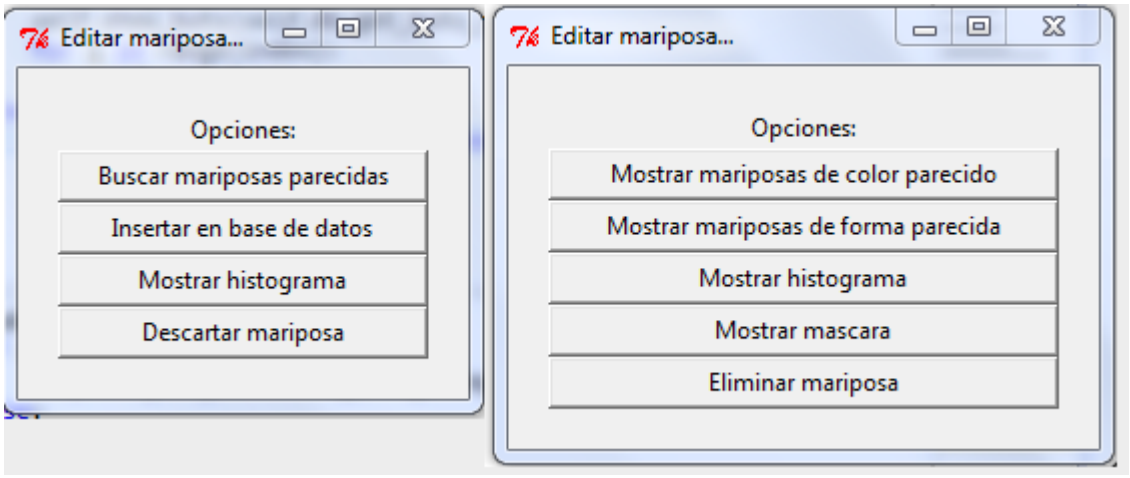

Figura 5.2: Opciones disponibles para una mariposa que no pertenece a la base de datos (dialogo de la izquierda) y para una que ya pertenece a la base de datos (derecha).

6. Fase de pruebas

El objetivo de la realización de pruebas respecto al desarrollo de la aplicación es cerciorarnos y comprobar el correcto funcionamiento del sistema, así como garantizar que cumple con todos los requisitos propuestos en la fase de análisis.

Se han realizado una serie de pruebas que se enumeran a continuación:

1. Carga de nueva mariposa.
2. Introducción en Base de Datos.
3. Mostrar mariposas de forma parecida.

6.1 Prueba 1: Carga de imágenes

El objetivo de esta primera prueba es verificar el correcto funcionamiento de la aplicación al abrir una determinada imagen.

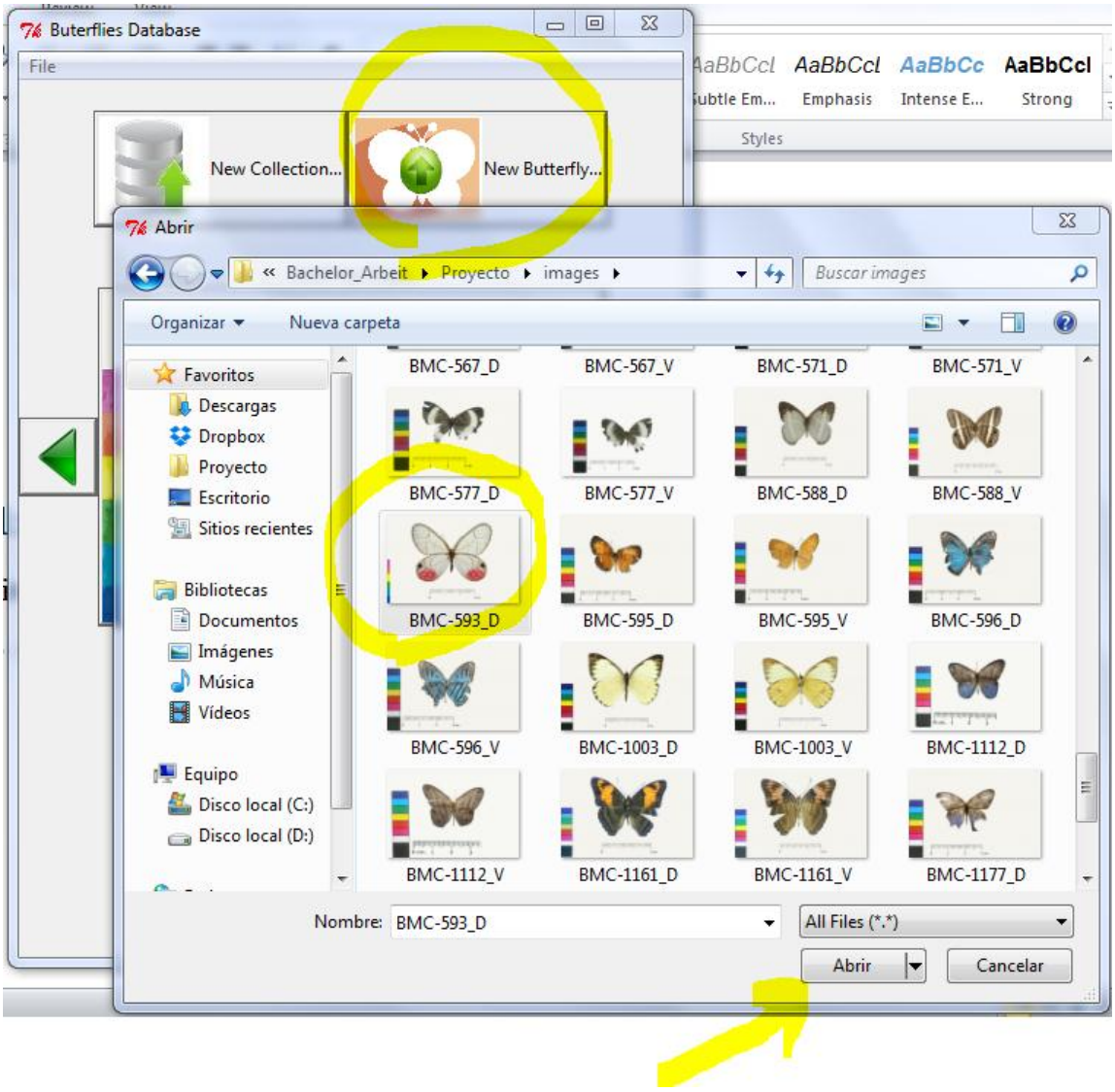

Figura 6.1: Carga de una mariposa.

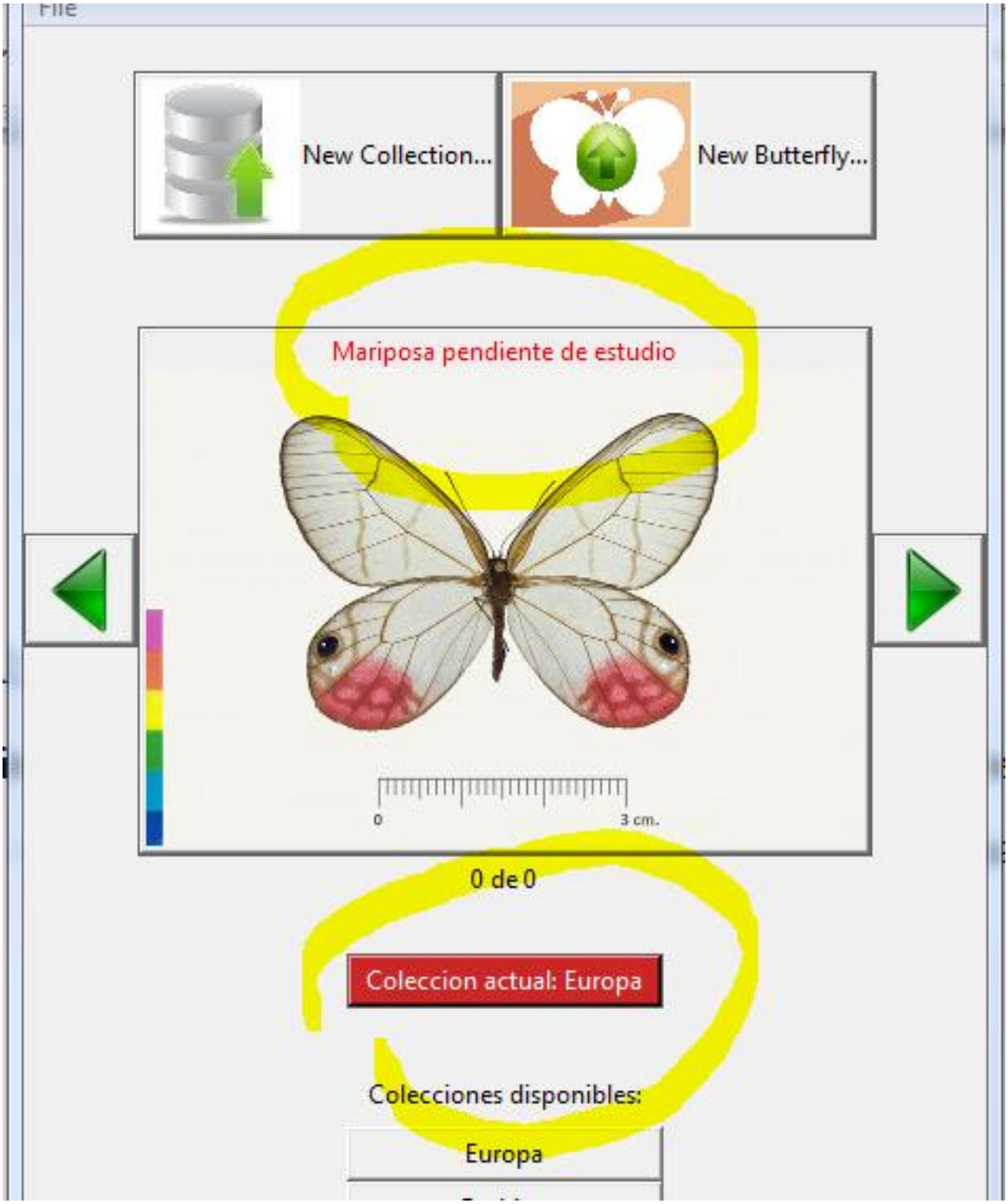

Figura 6.2: La mariposa todavía no está en la base de datos, pero aun y así ya tenemos la colección seleccionada. No hace falta crear una nueva, a no ser que queramos.

6.2 Prueba 2: Introducción en Base de Datos.

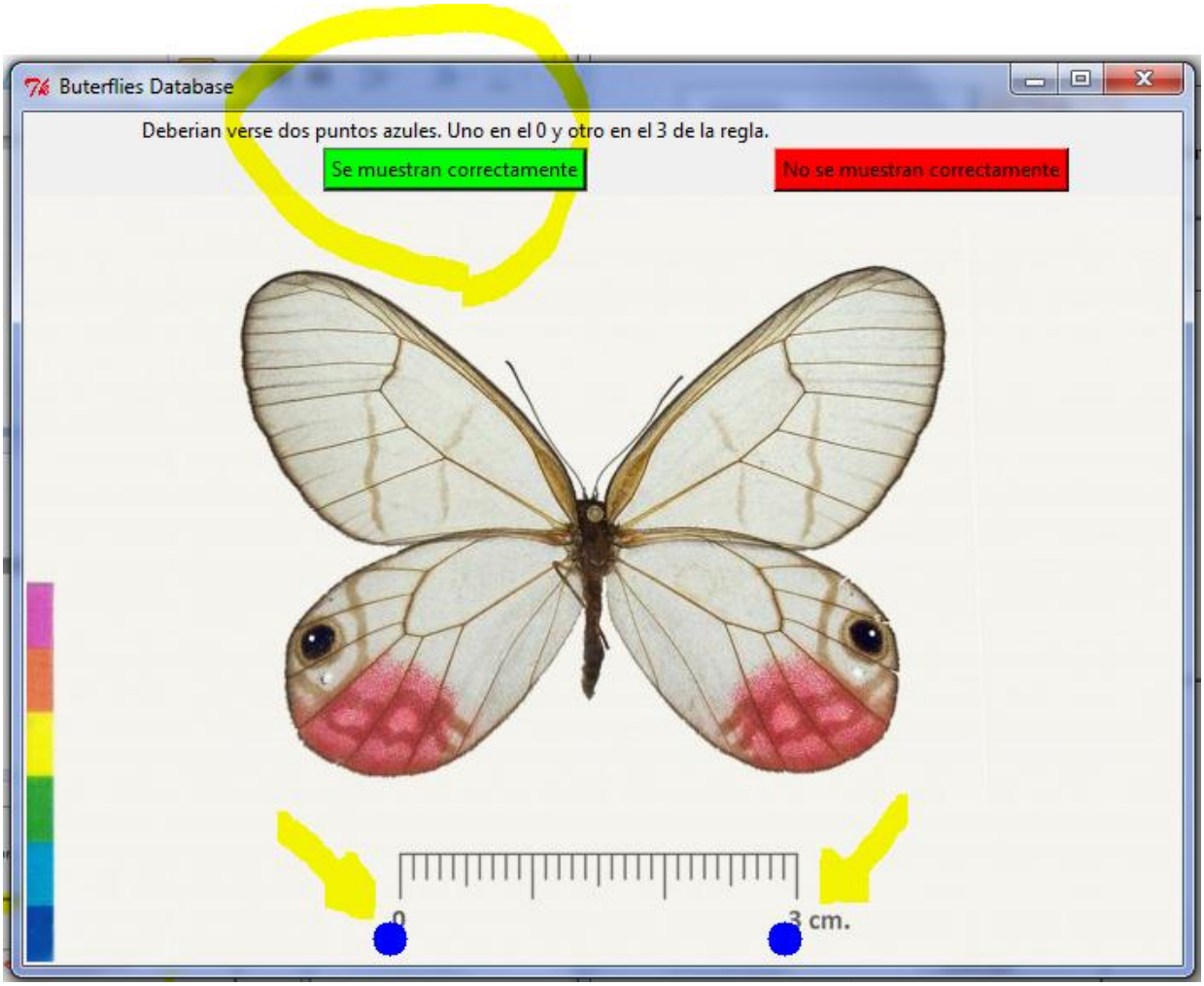

Figura 6.3: Ventana donde se muestra el resultado del algoritmo de búsqueda de la escala métrica.

El sistema calcula la localización del 0 y del 3 en la imagen. En este caso ha sido satisfactoria, por lo que pulsamos el botón verde.

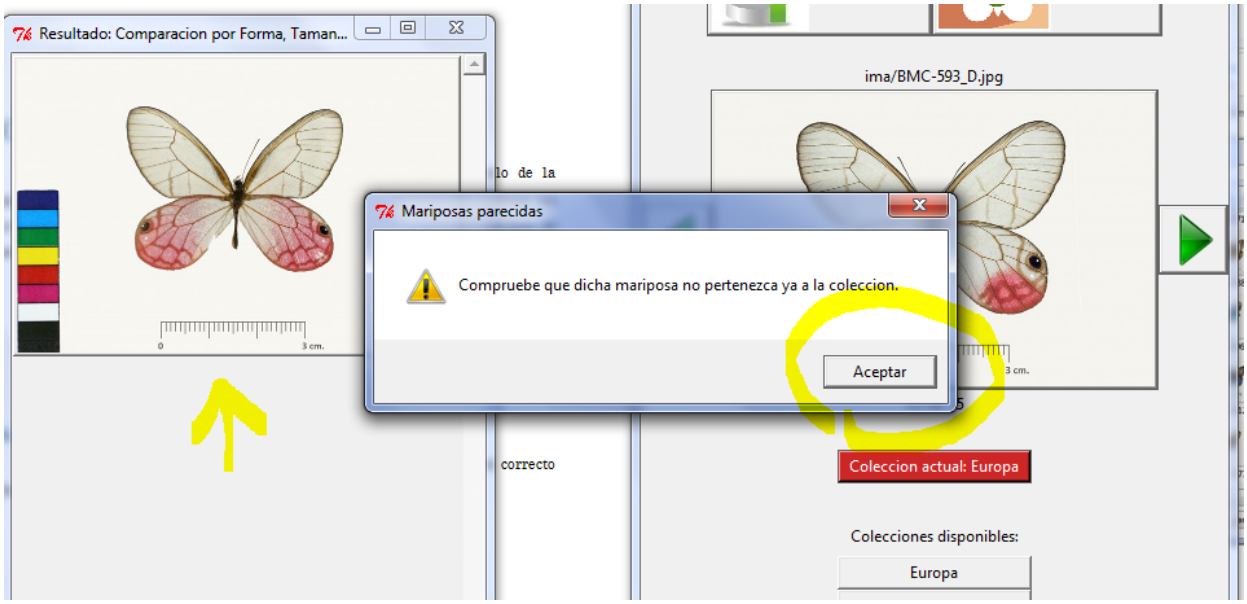

Figura 6.4: El sistema al meter la nueva mariposa en la base de datos, ha descubierto una parecida. En este caso son muy parecidas pero no es la misma. Aceptamos y cerramos la ventana de resultados que está a la izquierda.

6.3 Prueba 3: Mostrar mariposas de forma parecida

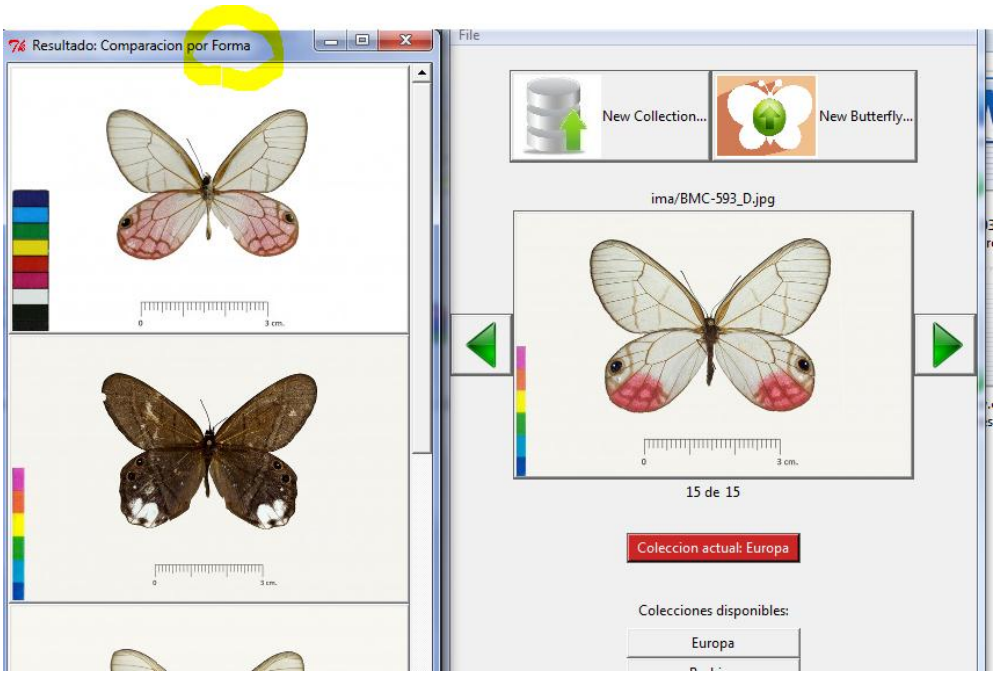

Figura 6.5: Se nos abre otra ventana de resultados pero esta vez con todas las mariposas que tienen forma parecida. Además las ordena por orden de similitud. Si buscásemos por color, la mariposa marrón obviamente no saldría, pero la primera mariposa sí.

7. Conclusiones

El proyecto realizado ha seguido el ciclo de vida típico de cualquier aplicación que se puede desarrollar a nivel educativo o empresarial, intentando poner en práctica técnicas de ingeniería del software y afianzando distintos conceptos aprendidos durante la carrera tales como programación orientada a objetos o documentación utilizando diagramas de clases.

En cuanto al diseño de la interfaz gráfica se ha conseguido una estructura eficaz, robusta y capaz de proporcionar al usuario un punto de vista amigable y eficiente mediante el uso fácil de la misma.

La implementación de los distintos algoritmos, tales como el reconocimiento de patrones o el estudio de los momentos de una imagen, ha permitido la adaptación de nuevos conceptos y técnicas utilizados en el tratamiento de imágenes y aplicados a la visión computarizada en imágenes digitales, abriendo todo un campo nuevo de posibilidades en el que todavía queda mucho por hacer e investigar.

Con respecto a la fiabilidad en el reconocimiento de la escala métrica, ésta se detecta en aproximadamente un 65% de las veces en las mariposas. Se debería buscar una solución mejor, pero el problema está en que hay demasiados tipos de escalas métricas. En cambio a la hora de calcular las máscaras, éstas las detecta en un 100% de los casos.

Como conclusión final quiero destacar la satisfacción de haber aprendido y experimentado con una nueva rama de la informática y el no descartar en un futuro profundizar más en ello.

8. Bibliografía

- Arlow, J., y Neustadt, I. (2006). "UML 2". ANAYA, 2006.
- Tutorial OpenCV-Python: www.opencv-python-tutroals.readthedocs.org/
- Documentación OpenCV: www.docs.opencv.org/
- Tutorial Python: www.codecademy.com/es/tracks/python/
- Draw.io (2015). www.draw.io/
- Dropbox (2015) www.dropbox.com/
- Gmail (2015) https://www.gmail.com/
- Laganière, R. (2011) OpenCV 2 Computer Vision Application Programming Cookbook, Packt Publishing, Olton, UK.

Apéndice A. Conceptos generales del tratamiento de imágenes

OpenCV (Open source Computer Vision library) es una librería de Visión Computarizada, destinada al tratamiento de imágenes y, principalmente, a la visión por computador en tiempo real. Contiene gran cantidad de funciones que abarcan una amplia gama de áreas en el proceso de visión, como reconocimiento de objetos, calibración de cámaras o visión robótica. Se puede utilizar con varios lenguajes de programación C, C++ o, como en nuestro caso, Python.

En la mayoría de las funciones de la librería que vamos a utilizar, se encuentra una gran variedad de cálculos, fórmulas y, en definitiva, métodos matemáticos. Por ello, en este apéndice, se explican conceptos generales del tratamiento de imágenes utilizando las principales funciones de OpenCV que van a ser necesarios para desarrollar el proyecto, centrándonos principalmente en entender su funcionamiento y tratar de explicar que hay detrás.

Antes de empezar, es conveniente explicar que un ordenador "ve" una imagen como una matriz, donde en cada posición hay tres números, el valor del color rojo, verde y azul (cada elemento de la matriz representa un píxel). Otra forma de verlo es como si la imagen se separara en tres matrices o "capas", una la de los valores del color rojo, otra verde y otra azul.

En imágenes en escala de grises solo habría una matriz donde cada número representa el valor del gris en ese píxel (véase Figura A.1).

A.1. Filtrado de imágenes

Los filtros que vamos a usar, dependiendo de su función es, una vez dada la imagen y un núcleo, es desplazarse píxel a píxel sobre la imagen, comparando los valores de la posición (i; j) con respecto a todos los del núcleo, obteniendo así uno nuevo. El valor que tenía el píxel (i; j) se cambia por el nuevo. Dependiendo del método que utilicemos, el núcleo tendrá tamaños diferentes, pudiendo llegar a ser de tamaño 1 1, por lo que el método de comparación también será distinto.

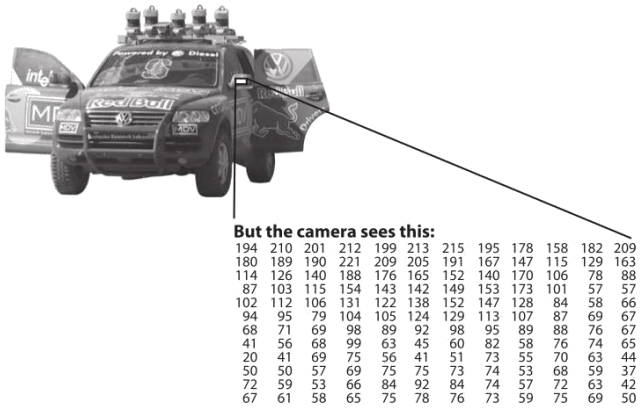

Figura A.1: El ordenador ve la imagen como una matriz de números. Si la imagen fuera a color, en lugar de una matriz serían tres.

A.1.1. Threshold

Estos filtros tienen muchas aplicaciones en el procesamiento de imágenes y además su funcionamiento es muy sencillo.

Dada una imagen y un límite L, se recorre toda la imagen comparando cada píxel con L y tomando una decisión sobre ese píxel dependiendo del tipo de threshold que hayamos elegido.

Hay muchos tipos diferentes de thresholds: THRESH BINARY, THRESH TRUNC, THRESH TOZERO, etc. (véase Figura A.2) y algunos un poco más complejos como cv2.adaptiveThreshold que funciona muy bien cuando hay mucha luz o cambios de iluminación dentro de la misma imagen. En este último lo que cambia principalmente es que en vez de fijarnos únicamente en un pixel para hacer la comparación, se utilizan otros métodos, como por ejemplo la media.

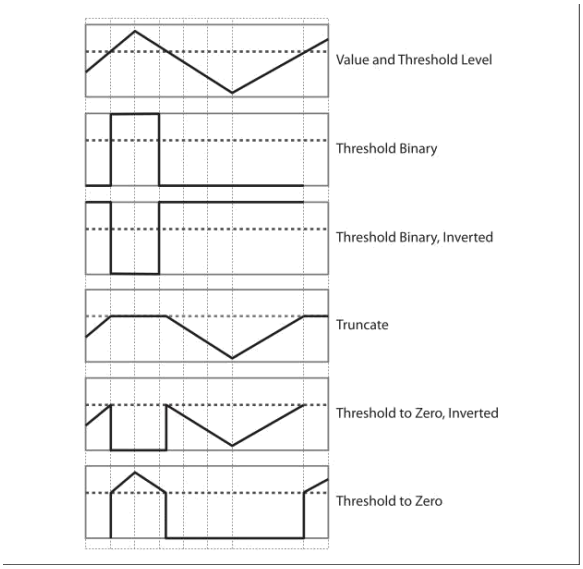

Figura A.2: Diferentes tipos de thresholds.

A.2. Análisis de imágenes

Hay muchas y muy variadas funciones y métodos que se utilizan para el análisis de imágenes, pero aquí nos vamos a centrar en los relacionados con el color y en concreto en los histogramas y espacios de color.

A.2.1. Histogramas

Los histogramas son una herramienta muy utilizada en Computer Vision debido a su gran variedad de aplicaciones como por ejemplo, detectar cambios de escena, puntos de interés o el reconocimiento de objetos.

Como ya hemos dicho, un histograma puede tener muchos usos pero aquí vamos a utilizarlo para contabilizar la cantidad de píxeles de cada color que tiene una imagen.

Dada la imagen y el tamaño del histograma (en este caso 256) se va a crear un array en el cual la posición 0 va a representar la cantidad de píxeles con valor 0 que hay en la imagen, la 1, la cantidad de píxeles con valor 1, y así sucesivamente hasta el 255. Estos datos se pueden representar en forma de gráfico (véase Figura A.3).

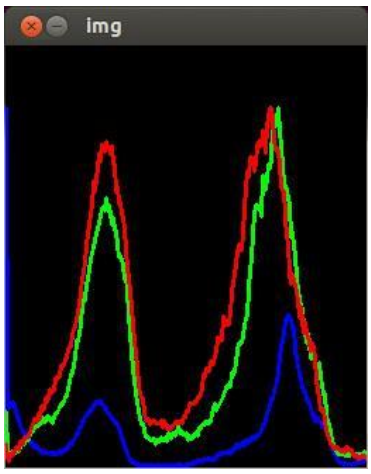

Figura A.3: Histograma.

También se puede utilizar una máscara para calcular el histograma únicamente sobre los píxeles de la máscara con valores distintos de cero.

A.3. Contornos

Muchos de los algoritmos que se utilizan para la detección de bordes están basados en el cálculo del gradiente, es decir, la dirección de máxima variación (cuanto más grande sea el gradiente más probabilidad hay de que sea un elemento del contorno). Algunos de estos métodos son `cv2.Sobel`, `cv2.Laplacian` o `cv2.Canny`. Después se puede aplicar `cv2.findContours` sobre una de las imágenes conseguidas con estos métodos para obtener los contornos de diferentes formas (solo los exteriores, en forma de árbol, etc.).

En esta sección se explica con más detalle la función `Canny`, aunque apenas lo hemos utilizado a lo largo del proyecto.

Una vez que se tienen calculados los contornos se pueden utilizar para muchas aplicaciones, combinándolos con otros métodos, como máscaras o por sí solos. Utilizando por ejemplo las funciones `cv2.contourArea` o `cv2.arcLength` es muy fácil calcular el área y el perímetro con respecto a los contornos. Aquí nos vamos a centrar en explicar cómo comparar dos contornos.

A.3.1. Canny

Lo que hace esta función es aplicar el algoritmo Canny con los límites superior e inferior dados por el usuario. Es decir, primero aplica un filtro gaussiano para reducir el ruido y después busca los bordes en cuatro direcciones: vertical, horizontal y en las diagonales. Una vez hecho esto, va fijándose píxel por píxel. Si el píxel tiene un gradiente mayor que el límite superior se toma como elemento del borde, si está por debajo del límite inferior se descarta y si está entre los dos límites se toma como borde solo sí está conectado a otro píxel que está por encima del límite superior.

A.3.2. Comparar contornos

Cuando ya se tienen calculados los contornos, una de las operaciones que se puede llevar a cabo con ellos es compararlos. Para ello se puede utilizar la función `cv2.matchShapes`.

Esta función lo que hace es sobre dos contornos y una métrica calcula un valor (utilizando los momentos invariantes de Hu) que indica cuán parecidos son. Las métricas son las de la Fig A.4.

Value of method	cvMatchShapes() return value
CV_CONTOURS_MATCH_I1	$I_1(A,B)=\sum_{i=1}^{7}\left\lvert\frac{1}{m_i^A}-\frac{1}{m_i^B}\right\rvert$
CV_CONTOURS_MATCH_I2	$I_2(A,B)=\sum_{i=1}^{7}\left\lvert m_i^A-m_i^B\right\rvert$
CV_CONTOURS_MATCH_I3	$I_3(A,B)=\sum_{i=1}^{7}\left\lvert\frac{m_i^A-m_i^B}{m_i^A}\right\rvert$

In the table, m_i^A and m_i^B are defined as:

$$m_i^A = \text{sign}(h_i^A)\cdot\log\left|h_i^A\right|$$
$$m_i^B = \text{sign}(h_i^B)\cdot\log\left|h_i^B\right|$$

where h_i^A and h_i^B are the Hu moments of A and B, respectively.

Figura A.4: Posibles métricas en la función cv2.matchShapes.

Además de comparar dos contornos con la función `cv2.matchShapes` se pueden utilizar otras propiedades de los contornos muy sencillas de calcular como por ejemplo, el área, el perímetro, la proporción entre base y altura del rectángulo de mínima área que lo contiene, etc. (http://www.cis.hut.fi/research/IA/paper/publications/bmvc97/bmvc97.html)

Apéndice B. Fragmentos de código

B.1. resize.py

```
import cv2
import sys
import numpy as np
#Le llega una imagen y busca el 0 y el 3 de la regla que escala las imágenes y devuelve la distancia en pixeles que los separan
def find_0_3(img_o):
    img = img_o
    h,w = img.shape[:2]
    mask = np.zeros((h,w,3), np.uint8)
    v0 = [h,w]
    v3 = [0,0]

    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    gray = np.float32(gray)
    #Detecta "perfectamente" el zero y el tres
    dst = cv2.cornerHarris(gray,15,21,0.22)

    #el resultado se dilate para el marcado de las esquinas
    dst = cv2.dilate(dst,None)

    # Threshold para un valor óptimo, puede variar dependiendo de la imagen.
```

```
mask[dst>0.005*dst.max()]=[0,0,255]

#Como sabemos que la métrica está en la mitad de abajo de todas la imágenes acotamos para reducir el coste en tiempo
for x in range(w):
    for y in range(h/2,h):
        px = mask[y,x]
        #Si el pixel es rojo mejoramos el valor para seguir bajando
        if px[0] == 0 and px[1] == 0 and px[2] == 255:
            if v3[0] < y or v3[1] < y:
                v3 = [x,y]
            if v0[0] > x or v0[1] < y:
                v0 = [x,y]
cv2.circle(img,(v3[0],v3[1]), 10, (0,0,255), -1)
cv2.circle(img,(v0[0],v0[1]), 10, (0,0,255), -1)
sol = v3[0]-v0[0]

img = img_o
return sol,img
```

B.2. build_mask.py

```
import cv2
import numpy as np

def get_contour(contours,w,h):

    max1 = 0
    i = 0
    centroide = False
    area = 0
    for c in contours:
        if cv2.arcLength(c, True) > max1:
            max1 = cv2.arcLength(c, True)
            cnt = contours[i]
        i = i + 1

    i = 0
    max2 = 0
    for c in contours:
        #Sacamos el centroide para ver si es contorno de mariposa
        centro = False
        M = cv2.moments(c)
        if M['m00'] != 0:
            cx = int(M['m10']/M['m00'])
            cy = int(M['m01']/M['m00'])
```

```
                if cy < 280 and cy > 180 and cx < 400 and cx > 300:
                    centro = True
            if cv2.arcLength(c, True) > max2 and cv2.arcLength(c, True) != max1
and centro:
                max2 = cv2.arcLength(c, True)
                cnt = contours[i]
                centroide = (cx,cy)
                area = cv2.contourArea(cnt)
            i = i + 1
            centro = False

        M = cv2.moments(cnt)
        cx = int(M['m10']/M['m00'])
        cy = int(M['m01']/M['m00'])

        return cnt,centroide,area

    def build_mask(img):
        t1 = 230
        t2 = 255
        t3 = 0
        m1 = 20
        m2 = 255

        #Carga de imágenes
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```
        h,w = img.shape[:2]
        ret,thresh = cv2.threshold(gray,t1,t2,t3)

        #Seleccionamos el contorno que nos interesa. El más grande es el recuadro de la foto. Así que probablemente el 2º más grande sea el de la mariposa.
        contours, hierarchy = cv2.findContours(thresh,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
        cnt,centroide,area = get_contour(contours,w,h)

        cv2.drawContours(image=gray,contours=[cnt], contourIdx=0, color=(0,255,0),thickness=-1)



        ret, mask = cv2.threshold(gray, m1, m2, cv2.THRESH_BINARY)
        mask_inv = cv2.bitwise_not(mask)
        blur = cv2.bilateralFilter(mask,10,150,150)
        return cnt,mask_inv,centroide,area

    def rebuild_moments(img):
        h,w = img.shape[:2]
        contours, hierarchy = cv2.findContours(img,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
        i = 0
        max1 = 0
        for c in contours:
            if cv2.arcLength(c, True) > max1:
```

```
            max1 = cv2.arcLength(c, True)
            cnt = contours[i]
        i = i + 1

    area = cv2.contourArea(cnt)
    return cnt,área
```

B.3. get_histogram.py

```
import cv2
import numpy as np

def get_hist(img,msk):
    h = np.zeros((300,256,3))
    b,g,r = cv2.split(img)
    bins = np.arange(256).reshape(256,1)
    color = [(255,0,0),(0,255,0),(0,0,255)]
    hist_item = []

    for item,col in zip([b,g,r],color):
        hist_item = cv2.calcHist([item],[0],msk,[256],[0,255])
        cv2.normalize(hist_item,hist_item,0,255,cv2.NORM_MINMAX)
        hist=np.int32(np.around(hist_item))
        pts = np.column_stack((bins,hist))
        cv2.polylines(h,[pts],False,col)
```

```
    h=np.flipud(h)
    return h, hist_item

#Cuanto más cercano a 0 más parecido tendrán
def compare_hist(img1,msk1,img2,msk2):

    b,g,r = cv2.split(img1)

    histB = cv2.calcHist([b],[0],msk1,[256],[0,255])
    cv2.normalize(histB,histB,0,1,cv2.NORM_MINMAX)
    histG = cv2.calcHist([g],[0],msk1,[256],[0,255])
    cv2.normalize(histG,histG,0,1,cv2.NORM_MINMAX)
    histR = cv2.calcHist([r],[0],msk1,[256],[0,255])
    cv2.normalize(histR,histR,0,1,cv2.NORM_MINMAX)

    #Cálculos de los histogramas de la segunda imagen
    b,g,r = cv2.split(img2)

    histB1 = cv2.calcHist([b],[0],msk2,[256],[0,255])
    cv2.normalize(histB1,histB1,0,1,cv2.NORM_MINMAX)
    histG1 = cv2.calcHist([g],[0],msk2,[256],[0,255])
    cv2.normalize(histG1,histG1,0,1,cv2.NORM_MINMAX)
    histR1 = cv2.calcHist([r],[0],msk2,[256],[0,255])
    cv2.normalize(histR1,histR1,0,1,cv2.NORM_MINMAX)
```

```
v1 = cv2.compareHist(histB, histB1, cv2.cv.CV_COMP_CHISQR)
v2 = cv2.compareHist(histG, histG1, cv2.cv.CV_COMP_CHISQR)
v3 = cv2.compareHist(histR, histR1, cv2.cv.CV_COMP_CHISQR)

return v1, v2, v3
```