ETIKETAGGER

(HERRAMIENTA PARA EL ETIQUETAJE DE SECUENCIAS DE IMÁGENES)

Memoria del proyecto de final de carrera correspondiente a los estudios de Ingeniería Superior en Informática presentado por César Cuadros Cortina y dirigido por Ricardo Juan Toledo Morales.

Bellaterra, Septiembre de 2009

El firmante, Ricardo Juan Toledo Morales, profesor del Departamento de Ciencias de la Computación de la Universidad Autónoma de Barcelona

CERTIFICA:

Que la presente memoria ha sido realizada bajo su dirección por César Cuadros Cortina

Bellaterra, Septiembre de 2009

Firmado:

A mis padres, por su increíble paciencia.

A Arturo, por sus sabios consejos e inagotables ganas de ayudar.

A Enric M.A., por la parte de él que quedó en mi.

Índice general

Índice de figuras

Capítulo 1

Introducción

El mundo de la Visión por Computador se ha ido desarrollando a pasos agigantados, se han creado proyectos de todo tipo con la idea de hacer un mundo al alcance de todos, facilitar aquello que nos es difícil o imposible conseguir, e incluso avanzar en un campo de aplicaciones que señalan un cambio notable en la era de la tecnología. Más concretamente, el campo de Reconocimiento de Objetos tiene una gran variedad de aplicaciones que están evolucionando de un modo muy cercano a nosotros, como el caso de robots capaces de desenvolverse en un entorno, localizar su posición, moverse a través de él y realizar acciones determinadas. Por esta razón, toda iniciativa o cooperación que se pueda aportar a este campo es acogida con los brazos abiertos, siendo este mi caso, una herramienta para el etiquetaje automático de objetos en secuencias de imágenes, diseñada principalmente para ser aplicada como complemento en un proyecto anterior: *Reconocimiento de objetos en panoramas y su aplicación a la localización de robots* [14].

1.1. Motivaciones

La principal razón que me ha motivado a realizar este proyecto es la oportunidad de ayudar en la evolución de nuevas tecnologías, el poder formar parte de algo nuevo, realizado en gran parte por estudiantes, capaz de mejorar el progreso de cara al futuro. Por otro lado, el interés que despertaron en mi las asignaturas relacionadas con la inteligencia artificial es uno de los factores de motivación influyentes para elegir desarrollar esta herramienta, así como el hecho de poder demostrarme a mi mismo las posibles aplicaciones prácticas de mis estudios. Realizar un buen proyecto también es una meta a alcanzar para la cual es necesario adquirir una actitud y responsabilidad de trabajo adecuadas, buenas costumbres en la programación y optimización de soluciones, y capacidad de trabajar en equipo, ya que mi herramienta es un complemento para proyectos de otros estudiantes. Considero que estos son factores imprescindibles y que se han de conocer mínimamente al acabar la carrera, para así poder afrontarse al mundo laboral con profesionalidad y personalidad como ingeniero informático.

1.2. Objetivos

El objetivo principal es crear una herramienta para la evaluación de los resultados del reconocimiento de objetos en imágenes. Dicha herramienta ha de servir para crear el *Ground Truth* para los datasets facilitados por el usuario, obteniendo de este modo una relación de objetos así como de sus características visuales y su posición concreta en una imagen. Más específicamente se pretende realizar un interfaz de usuario con las funcionalidades propias de un editor gráfico, y conseguir así un método para etiquetar de manera eficiente, rápida y fiable. El objetivo principal se materializa a partir de 3 hitos (ver figura 1.1):

1.2.1. Realizar interfaz gráfico

El interfaz gráfico será la aplicación sobre la cual el usuario trabajará directamente, desde él se cargará el dataset para ser etiquetado pudiendo navegar por todas sus imágenes. Tendrá las características básicas de un editor de imágenes orientado al etiquetaje, como dibujar rectángulos para delimitar objetos en la fotografía (*Bounding Boxes o BBoxes* de ahora en adelante), modificar sus dimensiones una vez ya dibujados, borrarlos, realizar zoom en las imágenes, relacionar los objetos de cada imagen con sus propiedades visuales y deshacer/rehacer acciones.

1.2.2. Conseguir automatización

A veces tendremos la necesidad de etiquetar un dataset con secuencias de imágenes compuesto por los *frames* que forman un vídeo, y por lo tanto, imágenes consecutivas que serán muy parecidas, con un pequeño desplazamiento en el espacio. La automatización tiene como finalidad conseguir que una vez se hayan establecido las *Bounding Boxes* de los objetos de interés de una imagen, éstas se mantengan dibujadas en las imágenes siguientes, es decir que se haga un seguimiento o *tracking* de los objetos y se dibujen las *Bounding Boxes* automáticamente para los objetos que ya fueron etiquetados en imágenes anteriores.

1.2.3. Validación de resultados

La validación es una funcionalidad extra del programa mediante la cual podremos realizar una evaluación de los resultados procedentes del método de reconocimiento de objetos que se está probando en el robot del departamento de robótica. El proceso de validación consiste en comprobar cuáles de las cajas que constan en el *Ground Truth* están también en los resultados que se están evaluando, lo cual forma el conjunto de *True Positive*. Por otra parte las cajas que no se encuentran entre los resultados forman el conjunto de los *False negative*. Por último, los resultados que no concuerdan con el *Ground Truth* conforman el conjunto de los *False Positive*. El criterio que dicta si dos cajas son equivalentes es, en primera

instancia, que ambas cajas se correspondan con el mismo objeto. Además el área de la intersección entre las cajas debe ser mayor que el 50 % del área total.

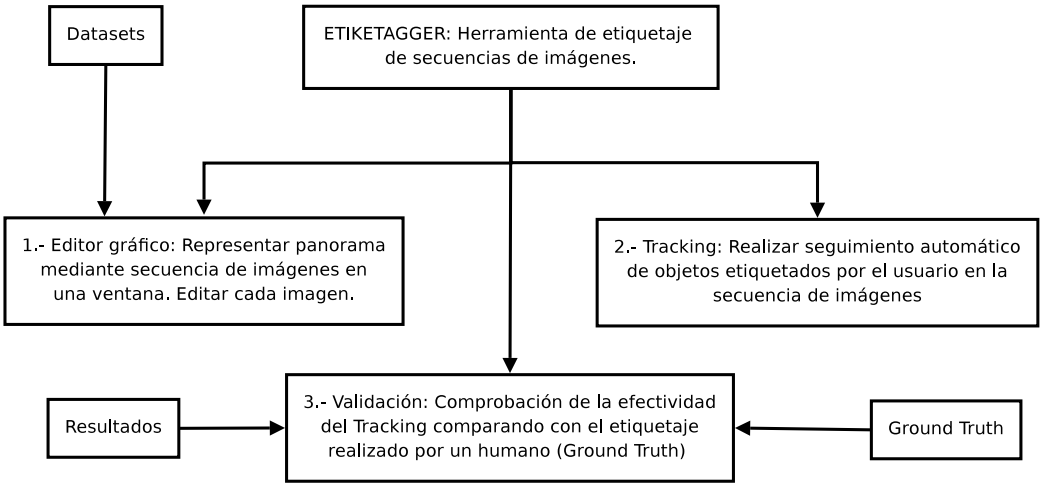

Figura 1.1: Esquema de los objetivos.

1.3. Organización de la memoria

En la Introducción se sitúa al lector en el ámbito sobre el cual se va a desarrollar la memoria, hablando tando de las raíces y el sentido del proyecto como de otros proyectos relacionados que especifican la razón de ser de *Etiketagger*. Se explican las motivaciones que han conducido a aceptar la responsabilidad de implementar esta herramienta, desde un punto de vista personal y profesional. Para terminar con la introducción se presentan los objetivos que deberían ser alcanzados según las previsiones realizadas por el autor y el director de proyecto.

El segundo apartado, Estado del Arte, resume cómo era el punto de partida desde el cual se debe planificar la solución a los objetivos, es en esta sección donde se explica la necesidad de crear la herramienta de etiquetaje a partir de los elementos previos conseguidos hasta la actualidad y de las ausencias funcionales que presentaban.

En la Propuesta de la Solución se plantean las ideas que hemos adoptado para alcanzar los objetivos de un modo eficaz y dentro de los plazos señalados, los

métodos bajo los cuales se han aplicado esas ideas y los recursos necesarios que han intervenido para lograrlo.

El capítulo 4 se puede entender como el corazón de la memoria, ya que contiene toda la información correspondiente al *'Cómo..?'*, relacionado con el desarrollo del proyecto. En este capítulo hemos definido cuáles son los conocimientos indispensables que se deben tener para que sea viable la implementación de la herramienta y el significado de los términos teóricos más importantes que se utilizan durante todo el documento. También se analiza cómo enfrentarse a los requisitos del proyecto, generando una idea abstracta de los componentes que se deben utilizar desde un punto de vista de ingeniería del software y dividiendo en fases correspondientes a los grupos de ideas que se han ido adoptando. Seguidamente se especifica el diseño que hemos elegido para cada parte en la que se divide el proyecto, explicando su causa-efecto y mostrando tablas y diagramas con la forma que adapta, con tal de poder hacer un seguimiento visual de cada módulo. En la siguiente sección explicamos qué significan y cómo se han implementado todas y cada una de las clases abstractas y diferentes funciones que componen el programa, así como las relaciones y dependencias que existen entre ellas. Para finalizar el capítulo presentamos una serie de pruebas, experimentos y evaluaciones de los resultados que han hecho posible el entendimiento e implementación de ciertas funciones, comportamiento de las mismas, depuración de errores y eficacia del programa.

En el último capítulo se resumen las conclusiones a las que hemos llegado, explicando claramente qué objetivos han sido alcanzados con éxito, posbiles mejoras de la herramienta, ampliaciones y aplicaciones diversas si se fusionan con otras herramientas existentes.

La memoria se ha realizado con LaTEX [6],[7],[18].

Capítulo 2

Estado del Arte

Como situación inicial se dispone del software y hardware necesario capaz de moverse por el entorno y capturar los datasets de imágenes para experimentos de reconocimiento de objetos(figura 2.1), así como de obtener los *datasets* relacionados a los objetos de interés en dichas imágenes, los cuales proporcionan un conjunto de propiedades asociadas a cada objeto.

Figura 2.1: Robot móvil utilizado para la toma de imágenes.

El punto de partida en el proyecto Etiketagger es una primera versión de un programa dedicado a tal efecto: un software básico, realizado en C++, capaz de enmarcar objetos mediante Bounding Boxes en una imagen y relacionarlo con sus características correspondientes (ver figura 2.2). Sin embargo este software no

es suficientemente óptimo, sobretodo en lo que respecta al tiempo necesario para subsanar los errores cometidos en el proceso de etiquetaje de las secuencias de imágenes, y añadiendo el hecho de que las Bounding Boxes son estáticas.

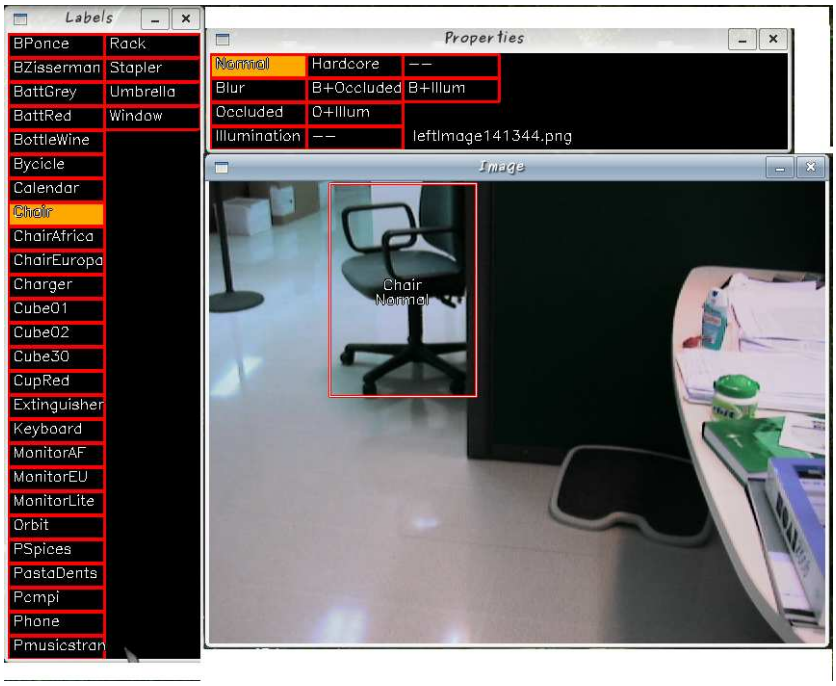

Figura 2.2: Situación inicial y punto de partida de Etiketagger (versión en C++).

Por otro lado, se ha decidido fusionar la validación de resultados de los experimentos de reconocimiento de objetos con el propio programa. Anteriormente era necesario disponer de diferentes programas para realizar el proceso de validación. Por una parte se cargaban los resultados de un experimento y se validaban con un *script* en MATLAB, obteniendo así mismo una imagen con los objetos encontrados y/o no encontrados para realizar las presentaciones. Luego estas imágenes eran retocadas y embellecidas para una presentación decente, por lo tanto se debía pasar por un programa de edición como el Gimp y así obtener la comparativa.

Con tal de solucionar los problemas que conlleva la versión existente hasta ahora, se ha decidido implementar un nuevo interfaz gráfico amigable, fácil de utilizar y con una gestión optimizada como editor gráfico, etiquetador de objetos,

y predictor de los mismos de manera que sea tolerante a errores, permitiendo una corrección sin desaprovechar el tiempo ni tener que repetir el proceso una y otra vez.

Capítulo 3

Propuesta de Solución

3.1. Descripción de la propuesta

En la implementación del software propuesto se ha decido utilizar el lenguaje de programación Python [1] por su expresividad [2], gestión de memoria y amigabilidad de las interfaces con las imágenes y los botones mediante widgets (wxPython [4]). También tiene la ventaja de disponer de una buena *API*(Aplication Program Interface) [3] de consulta, y poder importar librerías de terceros de un modo sencillo y obtener así una mayor funcionalidad.

3.1.1. Metodología

La metodología de la solución se basa, a grandes rasgos, en crear una interfaz de usuario como editor gráfico y trabajar sobre una sóla imagen hasta obtener unos resultados de edición gráfica aceptables, seguidamente ampliar la funcionalidad para una lista de imágenes. Una vez conseguido este punto se pasa a la automatización. Con ella se pretende reducir el esfuerzo del usuario localizando objetos en la imagen, implementando para ello un mecanismo de predicción (explicado con detalle más adelante). Finalmente, se procede a una fase de validación con tal de concluir si los resultados obtenidos cumplen con los requisitos necesarios.

3.2. Planificación y recursos

3.2.1. Planificación

Como punto de partida, se realizó una planificación en el informe previo correspondiente al siguiente diagrama de Gantt, representado en la figura 3.1:

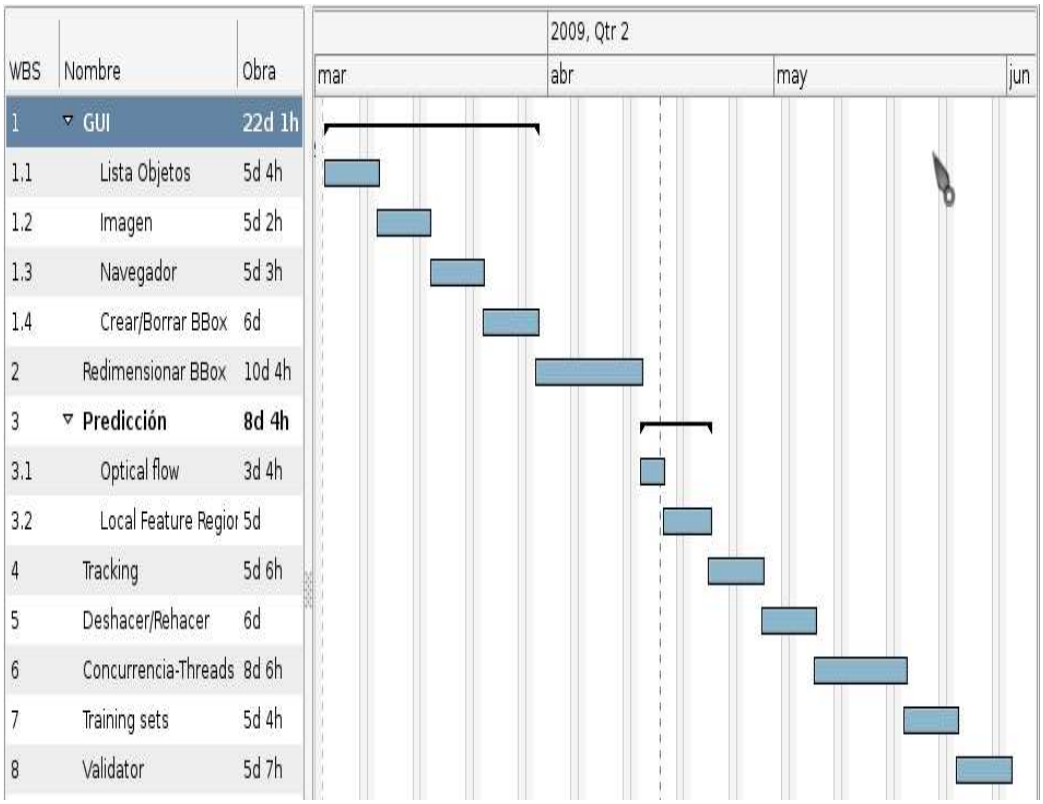

Figura 3.1: Diagrama inicial de planificación.

Sin embargo, en la realidad, el porceso de desarrollo del interfaz gráfico se alargó más de lo planificado, debido a la gran cantidad de detalles y algunas reestructuraciones del proyecto. Por otro lado, la implementación de la automatización del seguimiento de objetos fue más sencilla de lo planificado, ya que se disponia de gran cantidad de ejemplos de uso de las librerías de OpenCV [5], [12] en programas de funcionalidad parecida, como aplicaciones de detección y reconocimiento de caras [12], con muchos ejemplos de código adaptado a Python [9],

[13]. En definitiva, el tiempo de desarrollo real del proyecto se representa en la figura 3.2 mediante otro diagrama de Gantt :

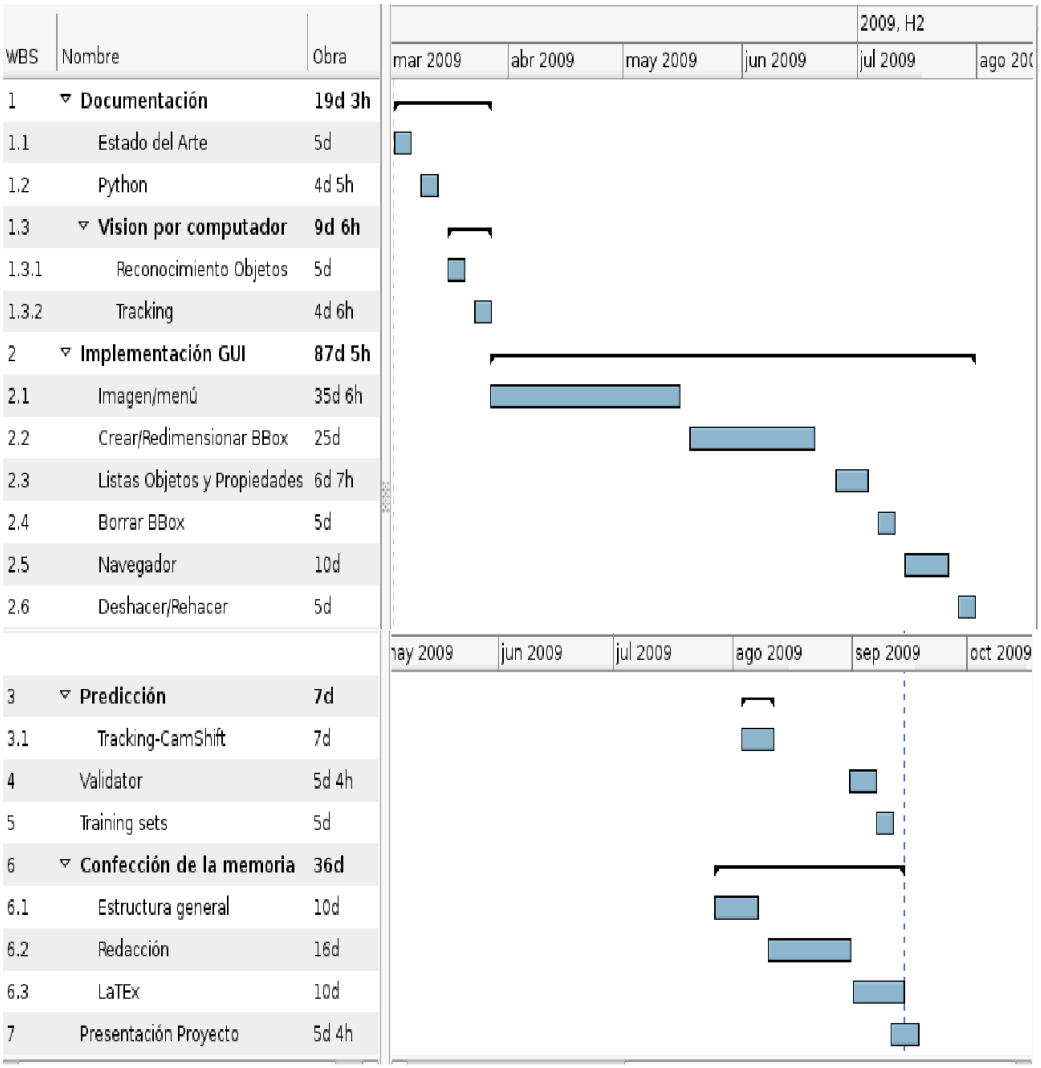

Figura 3.2: Diagrama correspondiente al desarrollo real del proyecto.

El proyecto Etiketagger se ha desarrollado en un total de 7 meses, desde Marzo hasta Septiembre del 2009. Consta de 3 hitos principales, entre los cuales el más extenso y que ha ocupado gran parte del tiempo de desarrollo es el editor gráfico. Posteriormente se pasa a desarrollar el software de predicción y validación de resultados:

- Repaso del Estado del Arte, documentación sobre Python, reconocimiento de obtejos y proyectos anteriores: Marzo 2009
- Inicio de desarrollo de software: Abril 2009

  Hito 1: Desarrollo del interfaz y editor gráfico: Abril, Mayo, Junio, Julio y Agosto 2009. Se crea el interfaz de usuario y el editor gráfico con todas sus opciones.

  Hito 2: Automatización/predicción de *BBoxes*: primera quincena de Agosto 2009. Se desarrolla el software necesario, mediante librerías de OpenCV, para realizar la predicción en la localización de objetos para secuencias de imágenes correspondientes a un panorama.

  Hito 3: Validación de resultados: Agosto y Septiembre 2009. Incorporar los scripts de validación de los resultados de los experimentos de reconocimiento de objetos.
- Final desarrollo de sofware: Septiembre 2009
- Realización de la Memoria: Julio, Agosto, Septiembre del 2009 .
- Realización de la presentación: Septiembre 2009

3.2.2. Recursos

En esta sección se detallan los recursos humanos y recursos técnicos, es decir, las personas que han intervenido y aportado ideas, y las herramientas y sofware utilizado para realizar el proyecto de manera óptima, tanto entornos de desarrollo como intérpretes, librerías externas, y datasets proporcionados para realizar pruebas y tests.

Es importante mencionar el apoyo y la orientación del director del proyecto Ricardo Toledo, imprescindible a la hora de presentar un trabajo digno de un ingeniero gracias a su supervisión, así como la insistencia en la necesidad de ir trabajando día a día. También es necesario mencionar el apoyo de Arturo Ribes, quien realizó el proyecto para el cual ha sido diseñada esta herramienta, ha sabido

orientarme en todo momento con sabios consejos y me ha insistido en utilizar las buenas prácticas de programación. Esta orientación también ha sido imprescindible para crear un programa que se acople perfectamente a los experimentos que se realizan en el departamento de robótica del IIIA-CSIC. Así como Arnau Ramisa, del mismo departamento, quien también ha supervisado la evolución del proyecto y ha aportado ideas para conseguir una herramienta apta y una buena memoria. Y por último, pero no por ello menos importante, yo mismo, César Cuadros, desarrollador directo del proyecto.

El software necesario para el desarrollo del proyecto es el IDE Eclipse con el módulo Pydev, para desarrollo en python, un intérprete Python 2.6.2 (release26-maint, Apr 19 2009) e Ipython 0.9.1 (consola interactiva de python). Otro editor utilizado es el Kwrite, con menores prestaciones pero apto para practicar y realizar pequeños programas de pruebas. Por otro lado se han utilizado librerías de wxWidgets (librería wx para python) con tal de implementar el interfaz gráfico y una librería de OpenCV (Open source Computer Vision), para poder procesar la información visual de las imágenes y obtener resultados en acorde a los objetivos propuestos, esta librería es *Ctypes_opencv* [11].

También se dispone de datasets facilitados por el departamento de Robótica del IIIA-CSIC, para comenzar a hacer pruebas sobre imágenes procedentes del entorno en el que se utilizará mayormente el software que se propone en este proyecto, y con las propiedades adecuadas. Como software de apoyo, en el que basarse para implementar un editor gráfico, se ha centrado la atención en el GIMP, editor gráfico de código abierto con una funcionalidad adecuada para dar ideas de sobre el estilo, diseño y *front-end* final que ofrecer al usuario.

Hablar de la viabilidad económica puede resultar supérfluo, ya que no se existe ninguna dependencia en este sentido. Sin embargo, habiendo analizado desde el punto de vista temporal y técnico como hemos hecho, podemos decir que la resuloción del proyecto es completamente viable.

Capítulo 4

Desarrollo Del Proyecto

4.1. Fundamentos teóricos

- Se requieren conocimientos avanzados sobre programación Funcional y Orientada a Objetos, así como bases de Geometría vectorial y matemáticas, para poder jugar con las coordenadas de la pantalla y transformarlas según convenga.

- *Bounding Box*: Término cuyo significado se puede entender por su porpio nombre *caja delimitadora*. En esta herramienta las Bounding Boxes definen el área de la imagen en la cual se encuentra un objeto. Además de la posición y el tamaño de la caja, nos interesa asociarle un identificador de objeto y las propiedades visuales con las que se encuentra en la imagen, como borroso, ocluido, mal iluminado, etc...

- *Ground Truth*: Representa la realidad tal y como es, en nuestro caso el ground truth sería la localización exacta y sin errores de todos aquellos objetos que se están etiquetando, para ello se crea un fichero GT (Ground Truth) que contiene, por cada línea y separados por espacios, los siguientes parámetros:

  <nombre imagen><min x><min y><max x><max y><id><propiedades>, donde:

nombre imagen: es el nombre que recibe la imagen en la cual se han editado Bounding Boxes;

min x: es la coordenada del eje x de la posición inicial de la Bounding Box en la ventana;

min y: es la coordenada del eje y de la posición inicial de la Bounding Box en la ventana;

max x: coordenada final del eje x;

max y: coordenada final del eje y;

id: representa el identificador del objeto que encuadra la Bounding Box, el cual tiene asociado una etiqueta;

propiedades: identificador de las propiedades visuales de ese objeto en determinada imagen (borroso, mal iluminado, *hardcore*);

En la figura 4.1 podemos ver un ejemplo de este tipo de ficheros.

```
despacho3 550 25 604 147 0 0
despacho3 32 372 89 425 0 0
despacho2 19 17 161 46 0 0
despacho2 475 411 601 460 0 0
despacho1 51 124 252 308 0 0
despacho1 513 113 580 352 0 0
```

Figura 4.1: Ejemplo de fichero GT.

- Visión por computador: Algoritmos CamShift() y MeanShift().

  El algoritmo que vamos a utilizar para la detección y seguimiento de la posición de un objeto es el Cam-Shift (Continuously Adaptative Mean SHIFT) [10], [16], [17]. Este método está basado en uno desarrollado previamente, denominado Mean-Shift [15] en el que se desplaza una ventana en la dirección del gradiente dentro de una imagen de probabilidad, hasta alcanzar un

máximo, tal y como se indica en la figura 4.2. De esta forma si se consigue una imagen de la probabilidad de encontrar el objeto deseado se puede buscar la de dicho objeto. A continuación está explicado el método en más profundidad.

Veamos el desarrollo del algoritmo Mean-Shift, que sigue los pasos mostrados a continuación:

1. Consideramos la selección de la ventana donde se encuentra el objeto como primer paso, aunque realmente es el segundo o último paso de Cam-Shift.
2. Se calcula el centro de masas de la ventana dentro de la imagen de probabilidad.
3. Se centra y escala la ventana en el centro de masas obtenido en el paso previo.
4. Se vuelve al paso 2 mientras no se cumpla el criterio de convergencia seleccionado.

El algoritmo de Cam-Shift sigue esta serie de pasos:

1. Se selecciona una Región de Interés (ROI) dentro de la imagen donde se espera encontrar el objeto (BBox)
2. Se selecciona una localización inicial donde colocar la ventana empleada para Mean-Shift. Esta ventana se emplea para calcular la distribución de probabilidad empleada en la creación de la imagen de probabilidad, si no se dispone previamente de ella.
3. Se calcula la distribución de probabilidad de la imagen.
4. Se itera el algoritmo Mean-Shift hasta encontrar el centroide de la imagen de probabilidad y se almacena la posición del centroide y el momento de orden cero.
5. Para el siguiente fotograma se centra la ventana en la posición del centroide y se actualiza su tamaño en función del momento de orden cero.

Se vuelve al punto 3.

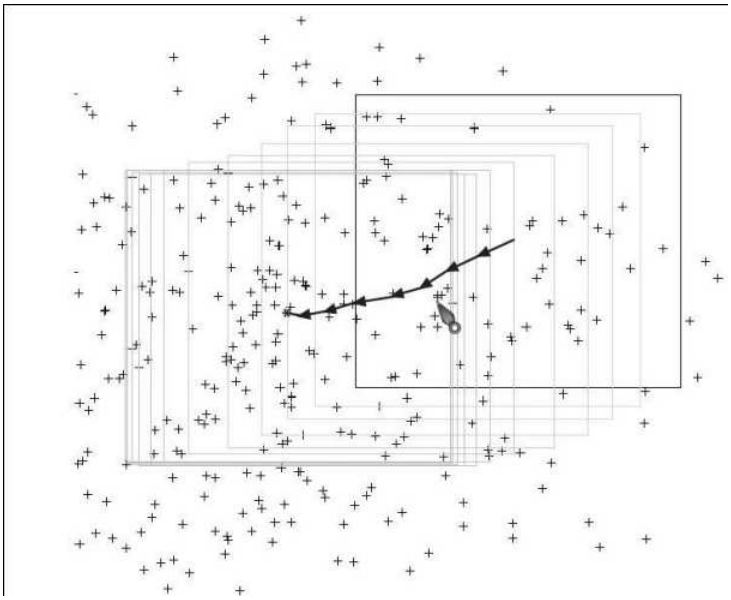

Figura 4.2: Mean-Shift - Desplazamiento de la ventana hacia el centro de masas, hasta cumplir el criterio de convergencia.

Imagen de probabilidad. Para calcular la imagen de probabilidad se puede emplear cualquier método que asocie a un valor de píxel una probabilidad. El método comunmente empleado es una retroproyección de histograma. Para su cálculo se suele emplear un histograma obtenido del canal *Hue* del modelo HSV [16], [17] dentro de la ventana seleccionada (figura 4.3). El histograma obtenido se escala de forma que el mayor valor tenga probabilidad 1 y a cada píxel de la imagen real se le asocia el valor del histograma que corresponda a su color. El cálculo del histograma se puede tener almacenado o calcular en la propia secuencia.

Centro de masas. Como ya hemos visto, tenemos que calcular el centro de masas de una ventana, para ello se emplean los momentos. En el cálculo de los momentos se emplea la intensidad de la imagen de probabilidad en el punto (x, y) denominada I(x, y):

1. Cálculo del momento de nivel cero:

$$M_{00} = \sum_x \sum_y I(x, y)$$

2. Cálculo de los momentos de primer nivel:

$$M_{10} = \sum_x \sum_y xI(x, y)$$
$$M_{01} = \sum_x \sum_y yI(x, y)$$

3. Cálculo del centro de masas:

$$x_c = \frac{M_{10}}{M_{00}}; y_c = \frac{M_{01}}{M_{00}}$$

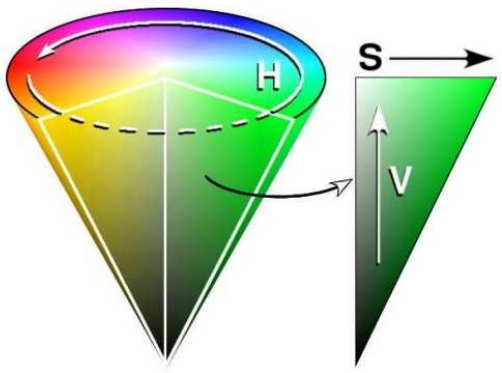

Figura 4.3: Modelo de color HSV - H:Canal de Color. S:Canal de Saturación. V:Canal de Brillo.

Convergencia. Finalmente se necesita tener un criterio de convergencia para detener el algoritmo Mean-Shift. Este puede estar basado en un número máximo de iteraciones, en una distancia máxima entre centros de masas consecutivos, etc. Además, en el caso de que el momento tenga un valor 0 el algoritmo también debe detenerse puesto que ningún píxel tiene valor, lo que indica que el objeto se ha perdido.

Explicación del algoritmo MeanShift:

Para las tareas de visión de bajo nivel es necesario que éstas sean controladas eficientemente por un número reducido de parámetros y que puedan representar confiablemente la entrada de datos, logrando de esta manera una fácil interconexión con las tareas de alto nivel.

Para ver más claro un espacio de características vamos a tomar el ejemplo del mapeo de una imagen a su correspondiente espacio tridimensional de características L*u*v* como se muestra en la figura 4.4.

Como se puede observar, se tiene una relación clara entre los colores dominantes de cada sección de objeto en la imagen y los clusters que se obtienen en el espacio de características. Hay que recordar que el espacio L*u*v* fue diseñado para tener una correspondencia más uniforme entre distancias geométricas y distancias preceptuales para colores que están bajo la misma referencia de iluminación.

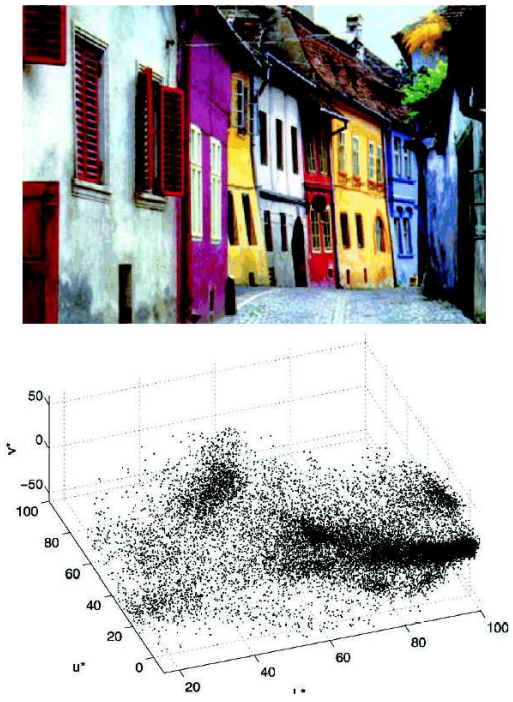

Figura 4.4: Mapeo y espacio tridimensional L*u*v de una imagen a color.

El método que se utiliza es uno perteneciente a hacer detección de clusters por un método no paramétrico basado en el estimador de la densidad. Lo que se hace es estimar la función de densidad de probabilidad (p.d.f. probability density function) implícita en la representación de la imagen en el espacio de características utilizado. Hecho lo anterior se sabe que las regiones con alta densidad en el espacio de características corresponden a máximos locales en la p.d.f., es decir las modas de la densidad desconocida.

Para detectar estas modas se utiliza la técnica Mean-Shift, un método para realizar tracking en tiempo real de objetos no rígidos basados en características como color y/o textura, donde las distribuciones estadísticas identifican el objeto de interés. El método es adecuado para una amplia variedad de objetos con diferentes colores y/o patrones de textura. Las iteraciones de Mean-Shift son empleadas para encontrar el objetivo candidato que es el más parecido al modelo.

4.2. Análisis y abstracciones

Como ya sabemos, este proyecto aporta una herramienta para facilitar la tarea de preparar los datos necesarios para el entrenamiento de un detector de objetos que utilizará un robot, de modo que sea capaz de orientarse y moverse en un espacio determinado; como aprendizaje y entrenamiento se entiende una comparación y validación humana de qué objetos es capaz de seguir, cuáles no y cómo de bien lo hace. Analizando estos puntos se decide dividir el proyecto en diferentes fases centradas en cada uno de ellos.

Generalmente, el proceso de etiquetaje involucra una lista de imágenes, una lista con nombres de objetos y otra con las propiedades de dichas imágenes. Para ello es imprescindible disponer de un interfaz capaz de relacionar los objetos, su localización en la imagen y propiedades de ésta, por ello la primera fase es implementar un editor gráfico con las siguientes funcionalidades:

- cargar de un directorio y mostrar la lista de objetos (nombres),

- cargar de un directorio una lista de texto con los nombre de las imágenes que componen la secuencia a etiquetar,
- visualizar cada imagen en una ventana donde se puedan dibujar cajas delimitadoras (Bounding Boxes) para encuadrar objetos y mostrar las propiedades de éste en otra ventana,
- cada objeto localizado por el usuario o por el programa debe relacionarse con su nombre (etiqueta), coordenadas de la imagen y propiedades visuales.

Una vez dispongamos de un editor gráfico en condiciones se pasará a implementar el proceso de automatización para conseguir que el programa haga una predicción de la localización de los objetos enmarcados en una imagen sobre las siguientes imágenes de la lista y dibujar un nueva BBox lo más encuadrada posible en el objeto correspondiente. Los errores se deben minimizar ya sea por modificación de parámetros o por interacción del usuario. Con esta fase se pretende reducir el tiempo que emplea el usuario en el etiquetaje, puesto que un alto porcentaje de objetos presentes en una imagen serán facilmente localizados por un algoritmo de tracking, evitando que el usuario tenga que encuadrarlos o requiriendo una pequeña modificación en la BBox predicha. Para lograr este proceso de automatización en una lista de imágenes se decide utilizar librerías de OpenCV, donde encontramos las funciones y algoritmos adecuados para el reconocimiento de objetos y su seguimiento.

Para finalizar se debe añadir una funcionalidad extra para poder realizar el proceso de validación, que se implementará como un modo de funcionamiento secundario del programa llamado modo de Validación, será necesario permitir cargar dos ficheros, uno de ellos el Ground Truth deseado, y otro con los resultados procedentes de las imágenes capturadas por el robot.

4.3. Diseño de la solución

A continuación se definen las diferentes fases fruto del análisis del proyecto, explicando la funcionalidad que se requiere en cada fase, el diseño de las diferen-

tes capas de abstracciones y el porqué de cada una de ellas.

4.3.1. Editor gráfico. (Montaje y modo de funcionamiento)

Al iniciar el programa se crea el entorno necesario para el uso de Etiketagger, que consiste en 3 ventanas, una para la edición de la imágenes, la segunda para escoger las etiquetas de los objetos y una tercera que contiene información sobre las propiedades visuales de cada imagen. Podemos observar una captura con el lanzamiento inicial del programa en la figura 4.5.

i) Ventana de imágenes:

Se trata de la ventana sobre la cual se sucederán la mayoría de las acciones para las cuales se ha diseñado esta herramienta. En ella mostraremos las imágenes que el usuario decida etiquetar, las cuales se podrán editar creando, modificando o eliminando las Bounding Boxes que se deseen, ya sean hechas por el usuario o predichas por el programa. La ventana principal de imágenes, llamada *ETIKETAGGER: Main Image*, se crea como primera instancia al lanzar el programa, una vez creada esta ventana se procede a crear las ventanas secundarias comentadas (listas de objetos y propiedades visuales). Al tratarse de un editor gráfico se ha dividido el montaje en dos capas principales: creación del entorno(marco, menús y botones) y manejador de eventos interactivos.

1. ENTORNO. La creación del entorno es simple, se trata de diseñar un Menú principal con opciones desplegables, con la idea de conseguir un manejo intuitivo del programa. Desde el Menú se podrá acceder a las opciones siguientes:

   File

   - *Load image list*, abre un diálogo parar cargar una lista de imágenes archivada en cualquier directorio elegido por el usuario.

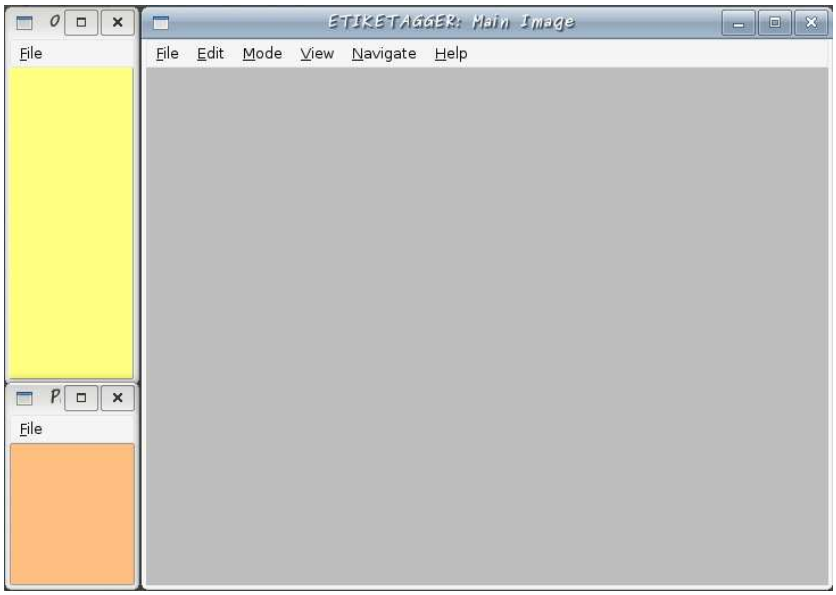

Figura 4.5: Lanzamiento de Etiketagger (las 3 ventanas).

- *Load/Save ground truth*, abre un diálogo para cargar y salvar un archivo con datos relativos a las imágenes (Ground Truth), desde o en la ruta indicada por el usuario.
- *Save image*, abre diálogo para salvar la imagen actual con todas sus modificaciones.
- *Exit*, salir del programa, cerrando todas las aplicaciones relativas a Etiketagger.

Edit

- *Undo/Redo*, deshacer o rehacer una acción recientemente realizada.

Navigate

- *Back/Next*, navegar por las imágenes correspondientes a la lista cargada por el usuario. A partir de la imagen en la que se encuentre el editor se puede navegar a la siguiente o a la anterior de forma cíclica.

View

- *BBoxes*, mostrar ventana de multiselección para mostrar opciones de visualización de resultados.
- *Refresh*, actualizar la vista de la imagen, dibujando todas las BBoxes en su posición correcta.

Help

- *Help*, mostrar ayuda, instrucciones del programa.
- *Version*, muestra una ventana con la versión, año y autor de Etiketagger.

2. MANEJADOR DE EVENTOS.

   Es el elemento principal de Etiketagger, se encarga de hacer posible la edición gráfica deseada a través de la interacción del usuario. Cada uno de los botones del Menú, o las acciones realizadas por el usuario a través del mouse o el teclado son capturadas mediante esta entidad, la cual se encarga de procesarlas según las relaciones **Evento-Función** mostradas en la siguiente tabla:

Evento	*Función*
Mantener presionada la tecla SHIFT	El editor gráfico cambia a modo DIBUJO
La tecla SHIFT está libre	El editor gráfico pasa a modo EDICIÓN
Mantener presionada la tecla D	El editor gráfico cambia a modo DELETE
Clic con el botón izquierdo del mouse	Recoge la acción y actúa segun el modo en el que se encuentre el editor: - MODO DIBUJO: se comienza a dibujar una BBox. - MODO EDICIÓN: se recoge la posición, y si coincide con una BBox se actua según la zona implicada (redimensionar BBox, mover) - MODO DELETE: se elimina la BBox sobre la cual se encuentra el mouse.
El mouse se mueve	Se recoge la posición del mouse, y si está en alguna posición relevante, como encima o en el interior de un BBox se tiene en cuenta, dibujando las líneas del rectángulo correspondiente por donde se mueva el mouse.
Soltar botón izquierdo del mouse	Termina la acción que se realizaba con el botón presionado.
Rueda del mouse	Según el sentido en el que gire se hace ZOOM IN o ZOOM OUT centrando la imagen en la posición del mouse.
Presionar la telca U	Equivalente a Undo, deshace la última acción realizada.
Presionar la telca R	Equivalente a Redo, rehace la última acción deshecha.
Presionar teclas A/Z, S/X, D/C	Modifica valor de parámetros que usa el algoritmo Cam-Shift sobre el ajuste para la localización automática de BBoxes: - A/Z: incrementa/decrementa el valor Vmin - S/X: incrementa/decrementa el valor Vmax - D/C: incrementa/decrementa el valor Smin

El manejador de eventos clasifica las acciones en 3 grupos, cambio de modo gráfico (Dibujo, Edición, Delete), eventos del mouse (movimiento, clic, rueda) y eventos de teclas (deshacer, rehacer, navegar, parámetros). El flujo de funcionamiento se puede entender observando la figura 4.6.

ii) Ventana de etiquetas

Esta ventana permite lanzar un diálogo para cargar un archivo de texto que contenga una lista de objetos, facilitada por el usuario. La funcionalidad es poder elegir el nombre del objeto para etiquetarlo en la imagen, imprimiéndolo en el interior de la BBox en cuanto se crea.

iii) Ventana de propiedades visuales

De modo análogo a la ventana de etiqetas se puede cargar un archivo de texto donde se muestran las propiedades visuales de una imagen, las cuales también son facilitadas por el usuario, estas propiedades tienen una funcionalidad de clasificación.

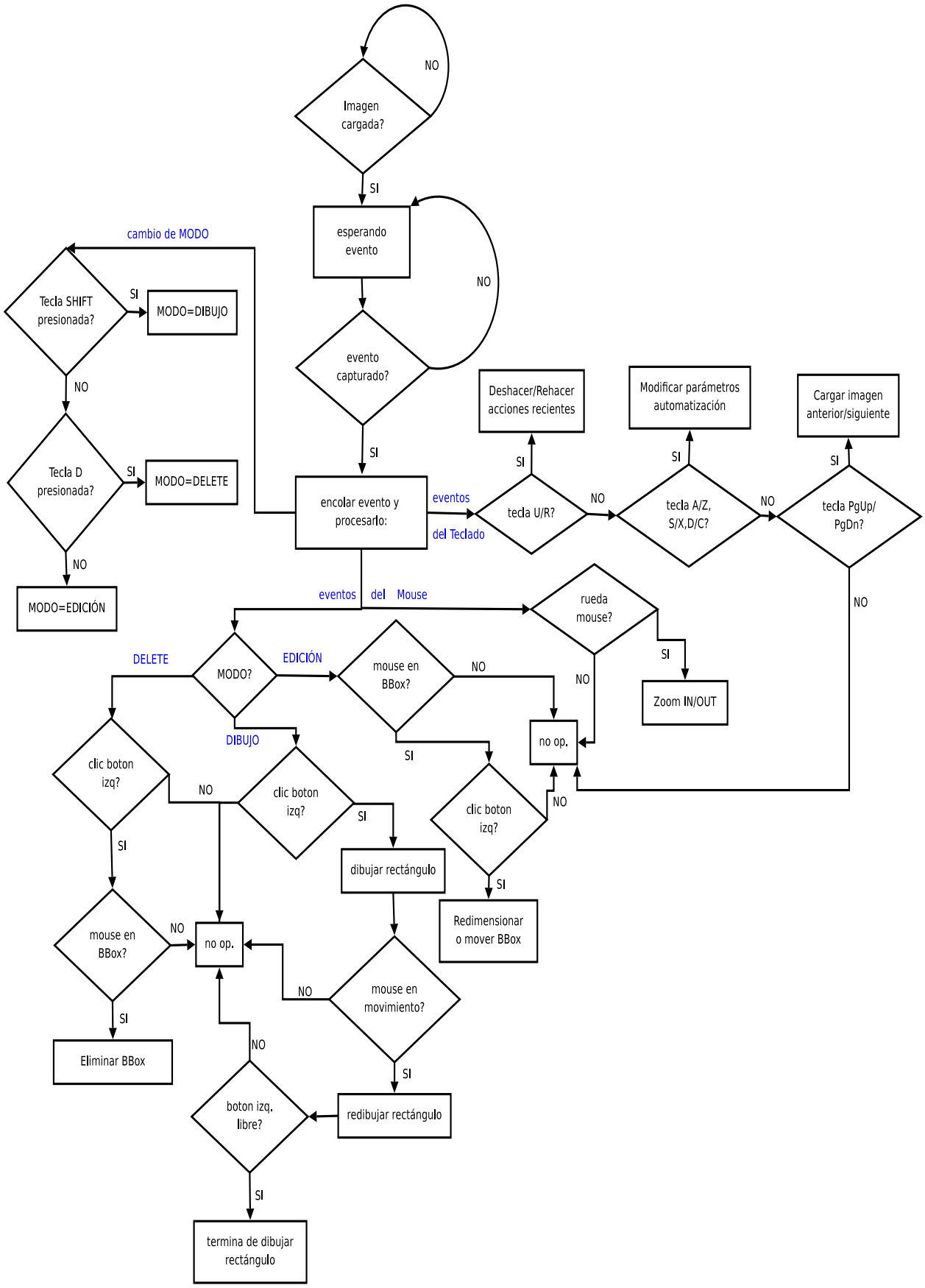

Figura 4.6: Diagrama de flujo con manejador de eventos.

4.3.2. Localización automática de objetos

Este proceso de seguimiento, conocido como *tracking*, realiza la función de buscar las características de cada Bounding Box de una imagen en la siguiente imagen de la lista, creando una nueva Bounding Box en ella de manera que encuadre el mismo objeto respecto la imagen anterior. Este método se va repitiendo para cada objeto nuevo que el usuario etiqueta a medida que aparecen, obteniendo como resultado la mejor aproximación posible para etiquetar de nuevo el mismo objeto a medida que va apareciendo en los fotogramas de la secuencia de imágenes. El estudio de cada imagen se realiza mediante el algoritmo CamShift, explicado en el apartado 4.1 (Fundamentos teóricos), el cual se encarga de ir parametrizando la imagen sobre una zona de interés, hasta llegar a un punto en que las propiedades visuales coinciden con la región de interés de la imagen anterior. El efecto de seguimiento de un objeto en varios frames quedaría del modo que se muestra en la figura 4.7.

Figura 4.7: Tracking de un objeto en una secuencia de imágenes.

De este modo se consigue automatizar el proceso de etiquetaje de cada objeto en todo el panorama que componen los diferentes frames de un vídeo, ahorrando al usuario el tiempo que conlleva tener que dibujar y editar cada una de las Bounding Boxes de cada objeto que aparece en las decenas o cientos de imágenes que lo compongan. De esta manera, habrá muchas ocasiones en que la predicción hecha

por el algoritmo de tracking sea suficientemente buena como para que no requiera que el usuario modifique la BBox.

4.3.3. Validación

En la fase de validación se permite evaluar los resultados obtenidos en el reconocimiento de objetos del robot, como verdaderos positivos (TP), falsos positivos (FP) y falsos negativos (FN). Para realizar la validación de resultados se ha de ejecutar Etiketagger en otro modo de funcionamiento (modo de Validación), mediante el cual tan sólo se observan datos que definen los resultados obtenidos en comparación con cierto Ground Truth creado previamente por el usuario. En este modo de funcionamiento no se permite editar ninguna imagen ni cambiar ninguna de sus propiedades, tan sólo es posible visualizar resultados y navegar por la lista de imágenes, podemos ver un ejemplo de comparativa entre Ground truth y resultados en las figuras 4.8 y 4.9, donde se muestran las BBoxes de tipo TP , tipo FP, FN y todas las BBoxes correspondientes a los ficheros GT y Results (mostradas siempre en verde y en rojo respectivamente).

Figura 4.8: Resultados obtenidos en el modo de Validación.Se muestran los grupos TP y FP.

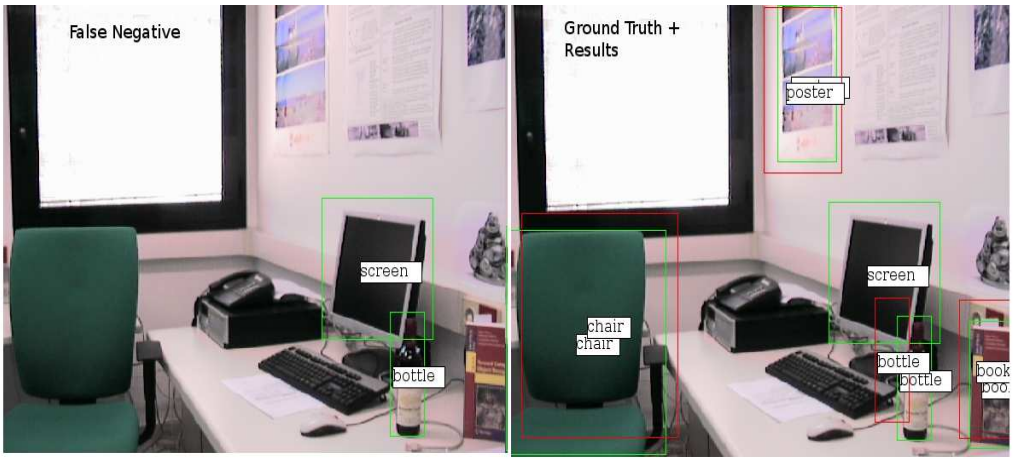

Figura 4.9: Resultados obtenidos en el modo de Validación.Se muestran los grupos FN y GT + Results.

4.4. Implementación

El procedimiento de implementación del proyecto *Etiketagger* queda dividido en 3 bloques principales, un bloque físico, donde es explica el proceso de creación y funcionamiento de las ventanas del navegador (editor gráfico, y listas de objetos y propiedades), y otros dos más abstractos, que conforman las características de las Bounding Boxes y el seguimiento o *tracking* de los objetos en las secuencias de imágenes. El método de implementación es completamente modular, se ha dividido en las clases mencionadas separando todas las funciones tanto como ha sido posible según su significado, de este modo es mucho más sencillo repasar el mecanismo del programa a medida que se ha va desarrollando y del mismo modo se facilita la evolución e inserción de nuevas funcionalidades sin apenas coste de refactorización de todo el proyecto.

4.4.1. Ventanas del programa

1. Ventana de imágenes/editor gráfico.

   Es la ventana principal y sobre la cual se trabajará más, por ello contiene el lanzamiento del programa y la funcionalidad en sí. Su estructura es la siguiente:

   Inicialmente se definen las variables globales, fijando sus valores a los convenidos, así como los valores ASCII que tienen las teclas que interactúan con el programa o los difenrentes modos de funcionamiento del programa, de esta manera podremos modificar estos valores fácilmente en un futuro y por otro lado nos aseguramos que el programa se iniciará en un modo concreto. También definimos los nombres de las carpetas para las rutas relativas de donde se podrán cargar las imágenes según su extensión, tal y como se requiere bajo las especificaciones hacia las cuales va dirigida esta herramienta de etiquetaje.

   Clase ImageFrame

   Es la clase del objeto que representa la ventana principal, es decir, el editor gráfico, junto todos sus complementos y respectivas funcionalidades. Se compone de:

   - Constructor, define cómo será el esqueleto de esta ventana y en qué orden se irán creando los diferentes componentes que la forman. En el constructor encontramos la inicialización de variables y la definición de los componentes del menú.
   - Inicialización de variables, para todas aquellas variables que necesitan establecer un valor al inicio ya se que pueden utilizar para diferenciar si se ha accedido a algún modo, activado algún evento o modificado opciones de menú.
   - Definición de los componentes del menú principal. Toda ventana debe tener un menú a través del cuál comunicarse con el programa. El orden

en el que se crean los botones o desplegables del menú se ha creado siguiendo un criterio que haga que el uso del programa sea intuitivo. El método de creación del menú se puede obsevar en la figura 4.10, tanto la barra de menú, sus componentes y los subcomponentes de éstos se crean mediante wxWidgets para Python (wx.python).

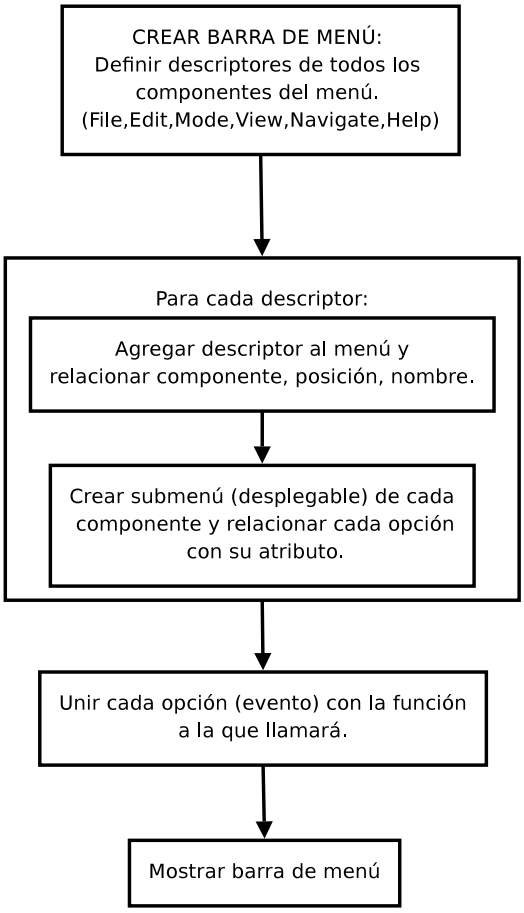

Figura 4.10: Flujo de la creación de la barra de menú.

Así conseguimos una estructura del tipo mostrado en la figura 4.11.

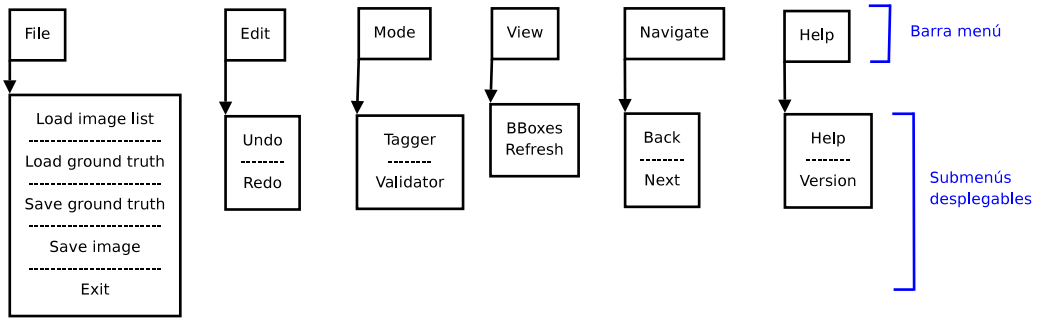

Figura 4.11: Diseño de la barra de menú y las opciones correspondientes.

Obteniendo una ventana con un menú similar al de la figura 4.12.

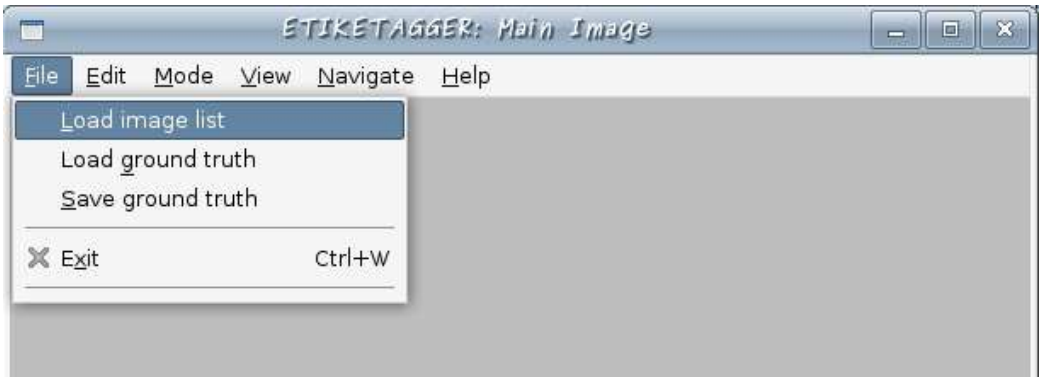

Figura 4.12: Ejemplo de barra de menú.

- Funciones: Es necesario diferenciar entre los dos modos de funcionamiento del programa, modo TAGGER y modo VALIDATOR:

MODO TAGGER (Etiquetador)

taggerMode(): Esta función se llama desde el menú, seleccionando Mode>Tagger. Su significado es el cambio de modo de programa a etiquetador, este es el modo básico de funcionamiento, para el cual está diseñada la herramienta. El programa siempre empieza en este modo por defecto.

validationMode(): Cambia el modo de programa a validador seleccionando Mode>Validator. Cumple la función de validar los resultados obtenidos por el programa, comparando el etiquetaje de objetos manual (realizado por un humano) con la predicción automática que ofrece la herramienta .Para ello es necesario cargar un fichero Ground Truth y un fichero de Resultados, por eso, como se puede observar en el diagrama inferior(figura 4.13), al cambiar el modo a Validator, automáticamente se abren los dos diálogos consecutivos respectivamente para cargar los ficheros deseados.También nos avisa en caso de no tener una lista de objetos cargada, y abre el diálogo correspondiente en caso necesario.

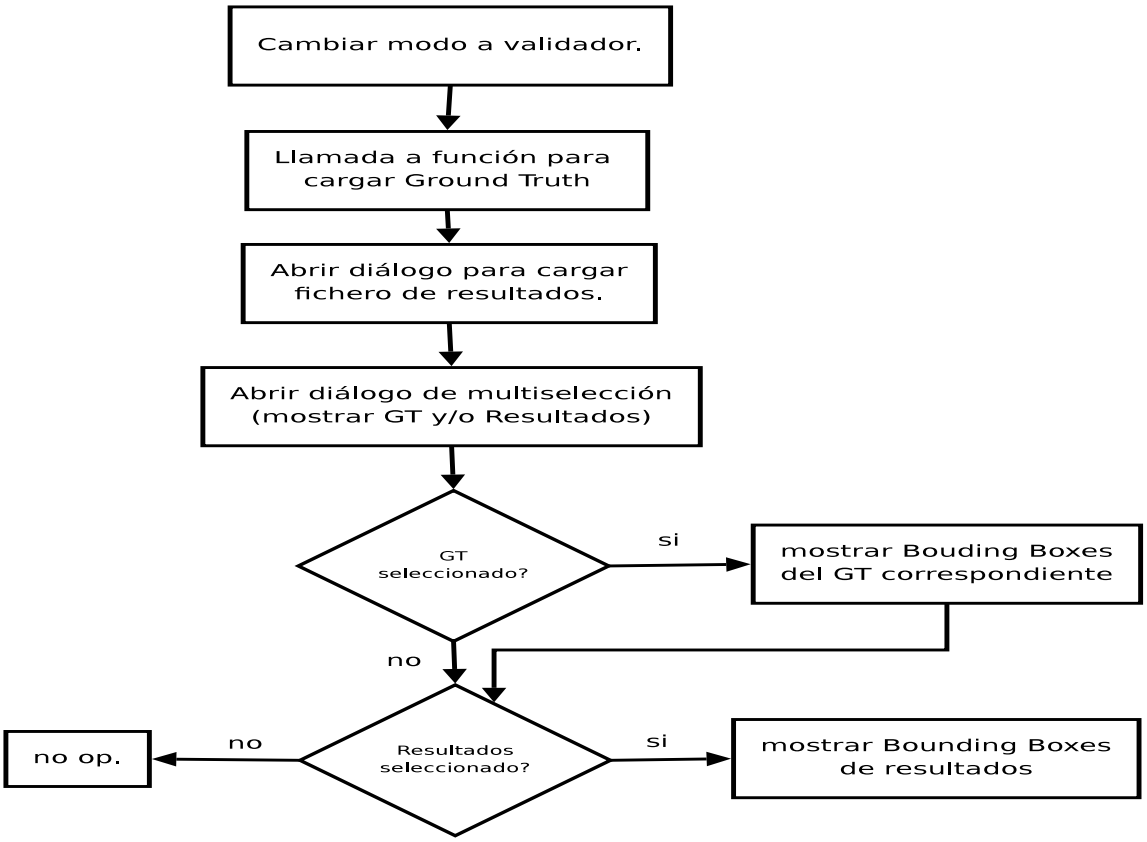

Figura 4.13: Flujo del cambio de modo.

getpicture(): Esta función devuelve el nombre de la imagen que corresponde mostrar en la ventana de imágenes. La imagen se puede seleccionar de dos formas diferentes, la primera es cuando se selecciona una nueva lista de imágenes, por tanto abre el diálogo para obtener la ruta y nombre de la primera imagen de la lista, el segundo modo es cuando se navega por la lista de imágenes, obteniendo el nombre de la siguiente o anterior imagen de la lista de forma cíclica (última imagen = anterior de primera imagen y viceversa) dependiendo del valor de la variable de navegación (*Navigate*) controlada por las funciones *navigateBack()* y *navigateNext()*. Podemos observar el flujo de esta función en la figura 4.14:

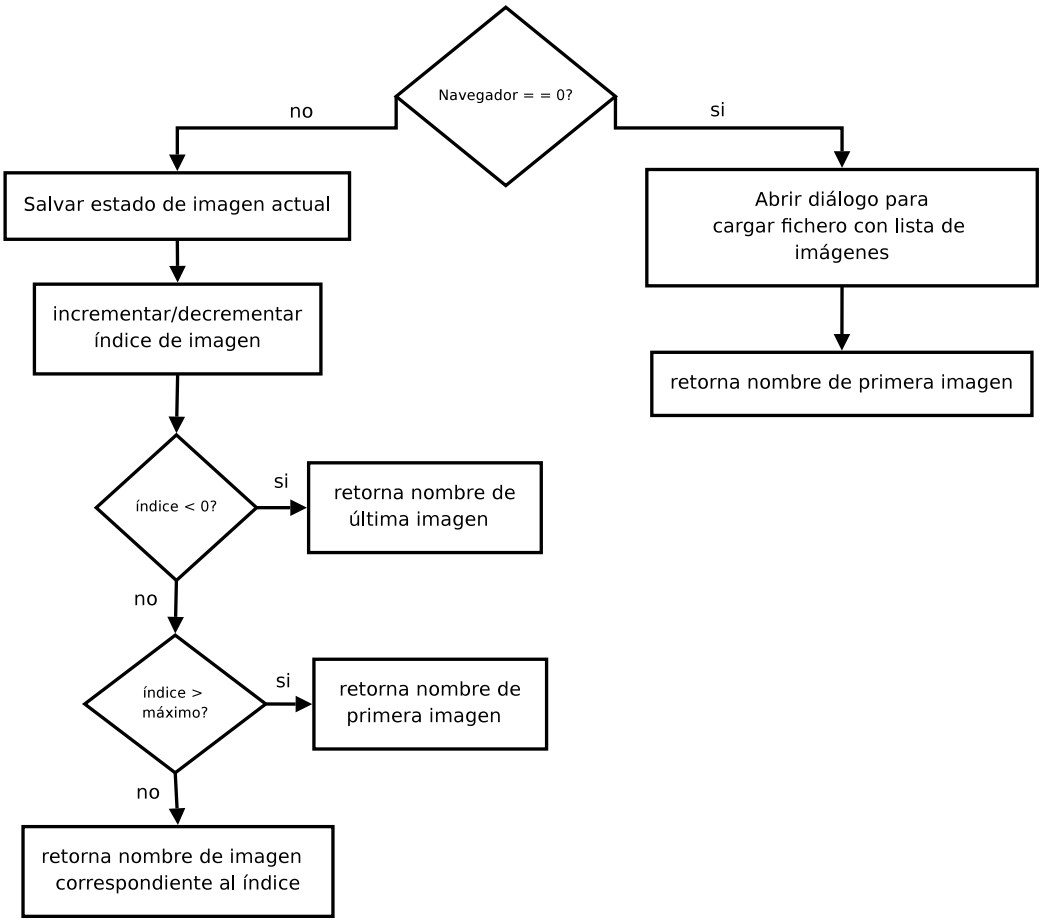

Figura 4.14: Diagrama que muestra cómo se obtiene la imagen a mostrar.

loadImageList(): Función que atiende el evento de cargar lista de imágenes cuando el usuario slecciona File>Load image list. Resetea valores antiguos para que no se mezclen con los de la nueva lista de imágenes (estados y lista de Bounding Boxes). Llama a la función *getpicture()* para obtener nombre de una imagen, y seguidamente llama a *gettingpicture()* para dibujar los detalles de medidas de imagen y escala.

gettingpicture(): Es una función muy sencilla de control. En ella se llama a *resolveImgPath()*, la cual nos devuelve la ruta del nombre

de la imagen. Si la imagen fue editada (creando o modificando Bounding Boxes) se carga su estado para mostrar las modificaciones que se realizaron anteriormente. Se encaja la imagen en la ventana. Veamos el funcionamiento en la figura 4.15.

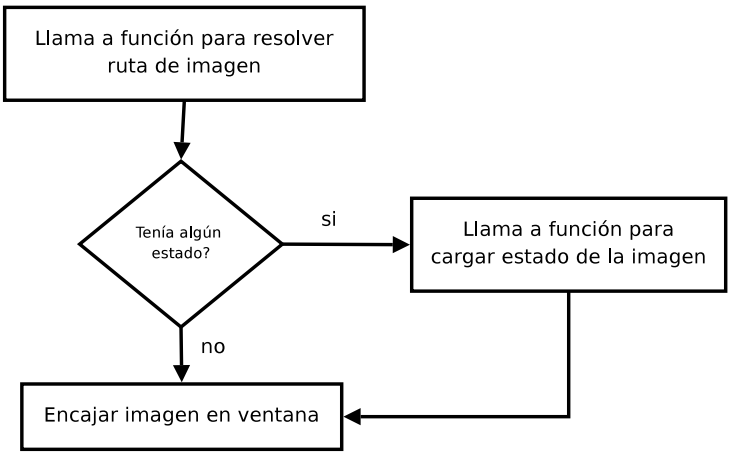

Figura 4.15: Obtenemos la ruta de la imagen y se dibuja el estado si lo tenía.

resolveImgPath(): Construye la ruta de la imagen a partir de su nombre, de los directorios existentes y de la extensión. Verifica existencia y ruta de una imagen, en caso de que no se encuentre la imagen se muestra un mensaje de error por pantalla para notificar al usuario.

loadGroundTruth(): Abre diálogo para cargar un fichero de texto. Una vez cargado, resetea estados de la imagen anterior (si la había), lee línea a línea el fichero para guardar en la variable de estados las nuevas Bounding Boxes de todas la imágenes que contiene, añadiéndolas a una lista de Bounding Boxes, que se usará para dibujarlas en la imagen; en caso de que una imagen no posea ningún estado, lo crea con una lista vacía de BBoxes. Finalmente llama a la función que carga estos estados. En la figura 4.16 se puede hacer un seguimiento del método para cargar un fichero Ground Truth.

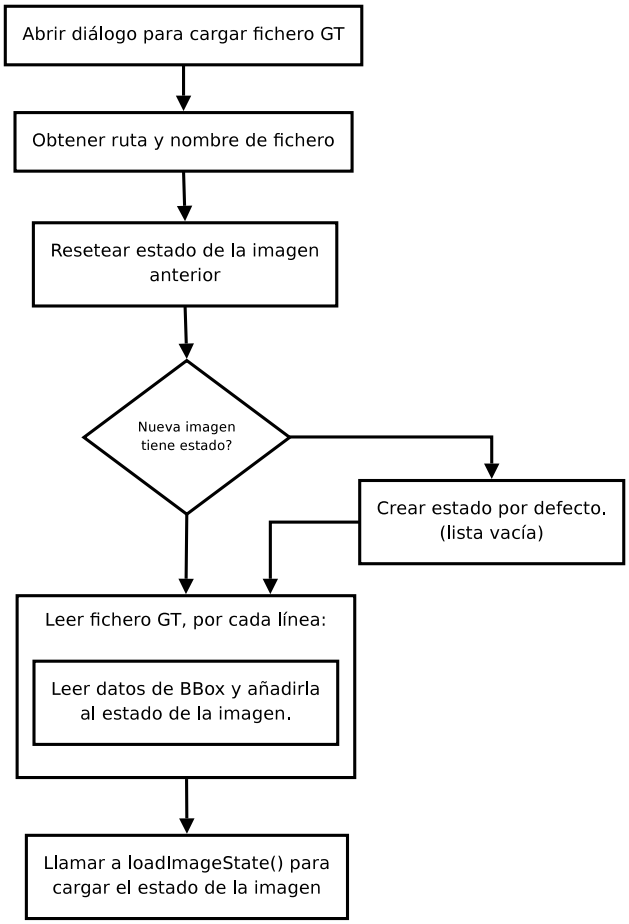

Figura 4.16: Diagrama de flujo que representa cómo se carga un fichero GT.

saveGroundTruth(): Salva el estado de la imagen actual, recupera una lista con los estados de cada imagen e inicializa lista donde se guardarán los datos del GT. Para cada BBox de cada imagen toma los datos de la Bounding Box (coordenadas inicio y fin, id y propiedades) y va agregando a la lista. Toma el nombre de la imagen junto los elementos de la lista, creando líneas de texto por cada imagen y separando los elementos por un espacio. Seguidamente abre un diálogo para seleccionar la ruta donde guardar el fichero GT y escribe todas la líneas creadas en dicho fichero. Veamos la figura 4.17:

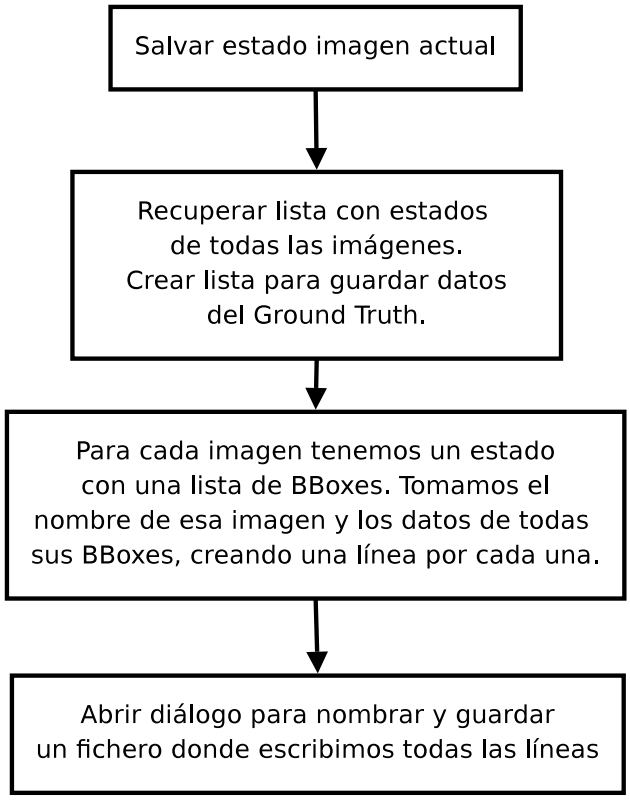

Figura 4.17: Diagrama de flujo que representa cómo se guarda un fichero GT.

saveImage(): Función encargada de abrir un diálogo para guardar la imagen actual con todas las modificaciones que se hayan realizado, al igual que todas las BBoxes que se muestren por pantalla. Es una función ideada para ir guardando todas las imágenes con los resultados obtenidos sobre el los conjunto de *true positive*, *flase positive* y *false negative*.

navigateBack(): Esta función se llama cuando el usuario selecciona Navigate>Back, o bien pulsando el botón "PgUp". Es muy simple, simplemente cambia el valor de la variable de control *Navigate* a -1, la cual se restará del índice cuando seguidamente se llama a la función *getpicture()* para obtner el nombre de la imagen, explicado con anterioridad. Finalmente se llama a *gettingpicture()* para arreglar los

detalles de medidas y barras de desplazamiento. Observar figura 4.14.

navigateNext(): Funciona de modo análogo a *navigateBack()*, donde *Navigate* toma el valo +1, para activar esta opción el usuario debe seleccionar Navigate>Next o con la tecla "PgDn". Esta función tiene la peculiaridad que al navergar hacia la siguiente imagen, nos interesa mantener las Bounding Boxes creadas en la imagen actual, para ello se guarda la ruta de la imagen y se llama a la función predictBBoxes(), que explicamos a continuación, pasándole esta ruta como parámetro. Observar figura 4.14.

predictBBoxes():Función para predecir la localización de cada BBox de una imagen en la siguiente imagen. Esta función prepara los valores e imágenes y lanza el proceso de predicción explicado en el apartado '4.1-Fundamentos teóricos'. Para ello se recibe como parámetro la ruta de la primera imagen, la cual contiene las Bounding Boxes creadas por el usuario (guardadas en la variable de estado de la imagen), seguidamente se obtienen el estado de la primera imagen y la ruta de la imagen siguiente, definimos parámetros visuales y llamamos a la función *Tracker()*, que se encargará, a grandes rasgos, de establecer los parámetros mencionados; después pasamos a crear el proceso de seguimiento de todos los objetos encuadrados por el usuario (*createTracker()*). Para cada Bounding Box de la imagen se crea una región de interés (donde se encuentra el objeto a encontrar en la siguiente imagen) y se llama a la función *startTracking()*, encargada de localizar el objeto en sí mismo, obteniendo una nueva región que se aplicará a la siguiente imagen mediante la función *track()*, que retorna las nuevas coordenadas donde debe ser dibujada la Bounding Box predicha. Para que la función *track()* trabaje correctamente se necesita asegurar que la región de interés que utilizará esté dentro de la imagen, por eso se controlan las coordenadas y se encajan dentro de los límites de la imagen si es necesario. así la guardamos en la lista del estado de la nueva imagen. Las funciones *Tracker()*, *createTracker()*, *startTra-*

cking() y *track()* se explicarán con detalle más adelante. El diagrama de flujo que representa la predicción de Bounding Boxes se puede observar en la figura 4.18.

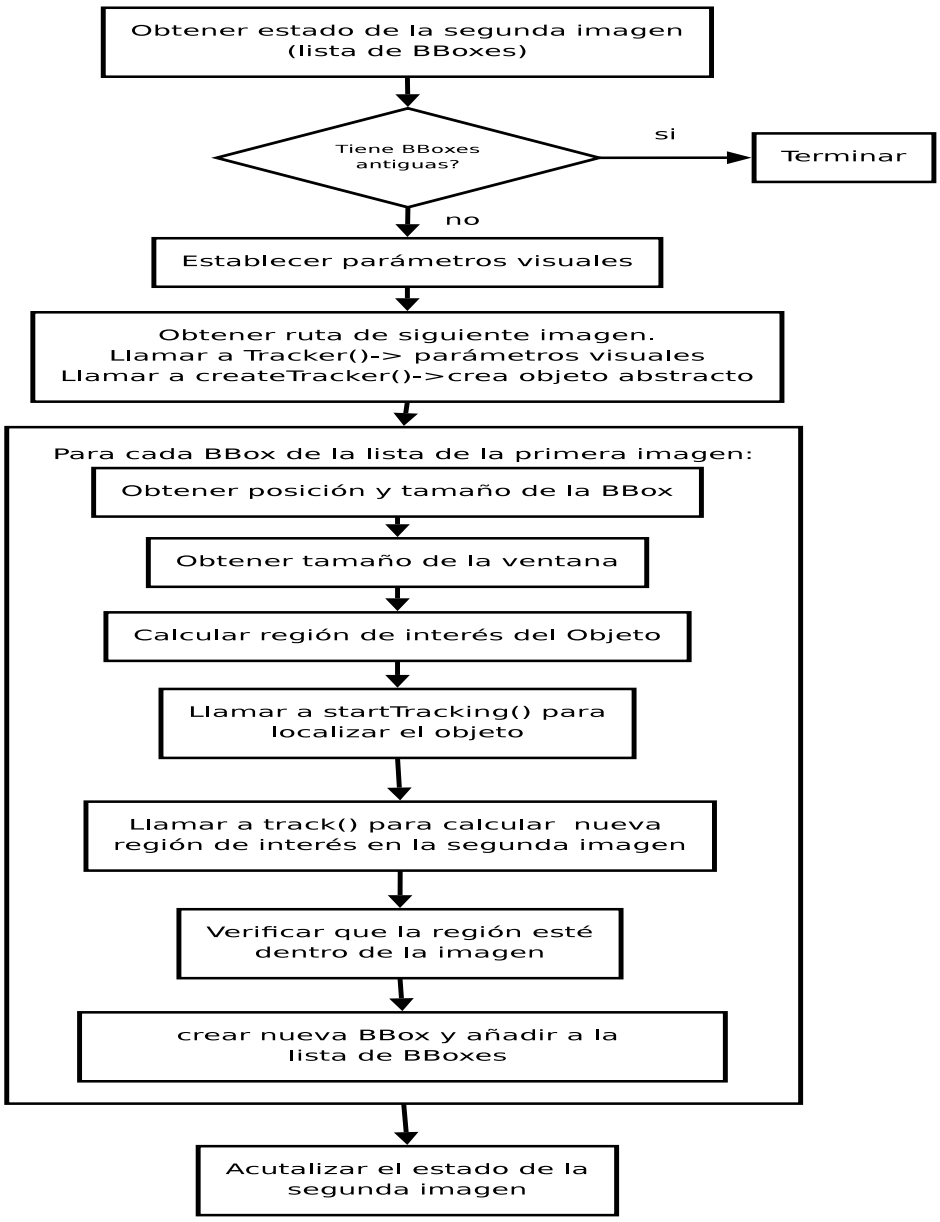

Figura 4.18: Diagrama de la función que implementa de forma general el proceso de seguimiento de objetos.

OnLeftDown(): En esta función se procesa la captura del evento de clic con el botón izquierdo del mouse sobre una imagen, obtiene las coordenadas actuales del mouse en cuanto esta acción se produce. Si estamos en modo de EDICION se comprueba que al hacer clic estamos encima de una Bounding Box, en tal caso tomamos la posicion inicial de la BBox para operar con ella. Si estamos en modo DIBUJO se procede a dibujar las líneas que componen la Bounding Box mediante la llamada a la función *createBoundingBox()*, y si estamos en modo DELETE, se elimina la Bounding Box de la lista de esa imagen. Observemos el modo en el que opera fijándonos en la figura 4.19.

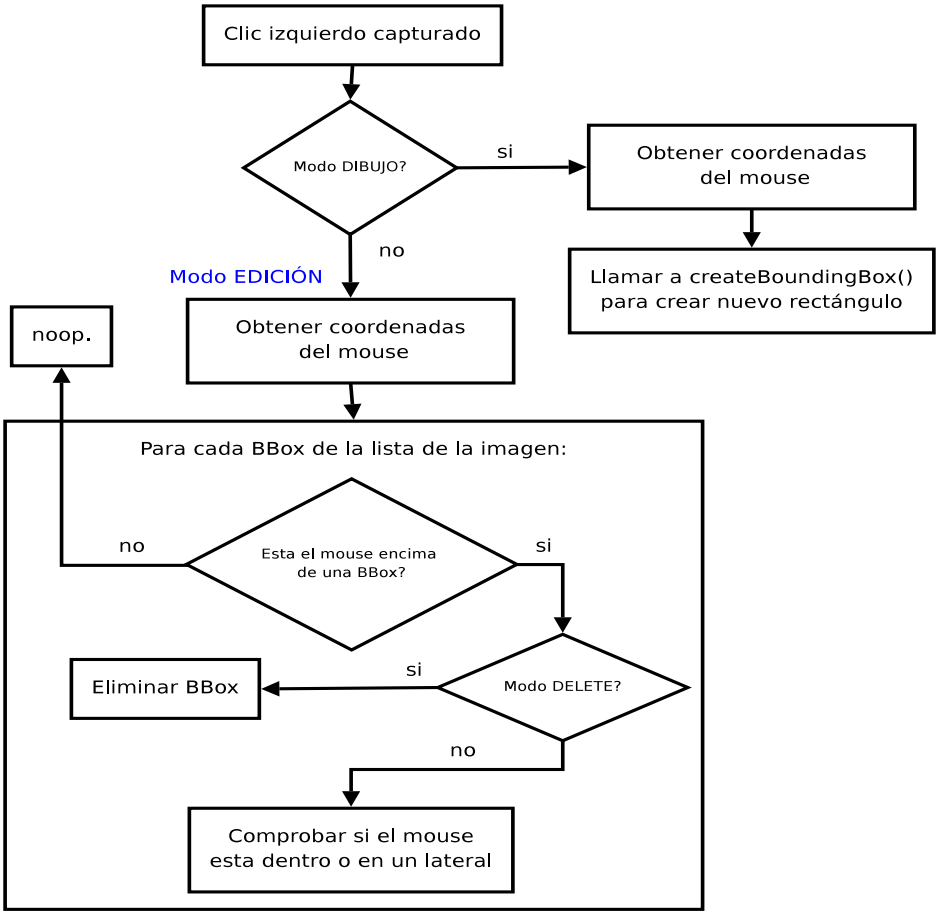

Figura 4.19: Podemos observar cómo se dividen los eventos de clic izquierdo según el modo de editor.

printBBoxes(): Es una pequeña función de test y control utilizada sólo para verificar las coordenadas de las Bounding Boxes de una lista concreta. Carece de importancia una vez comprobado el correcto funcionamiento del programa.

OnMoving(): Encargada de procesar el evento de movimiento del ratón, cuando este se mueve tomamos posición real del mouse en la ventana real, y actuamos según el modo en el que estemos; si el modo es DIBUJO, comprobamos que el botón izquierdo esté presionado, en tal caso se está dibujando una nueva Bounding Box, por eso llamamos al procedimiento *Draw()*, para dibujar la caja, y vamos actualizando las medidas de la misma a medida que se crea. Sin embargo, si se está en modo EDICIÓN, y el raton se encuetra encima de una Bounding Box, comprobamos que el botón izquierdo esté presionado y pasamos a mover la Bounding Box si se acontece. Del mismo modo comprobamos si el raton está en alguno de los laterales o esquinas de la caja para cambiar las dimensiones de ésta, modificando el estilo del cursor para remarcar la zona en la que se encuentra en todo momento. Veamos la figura 4.20.

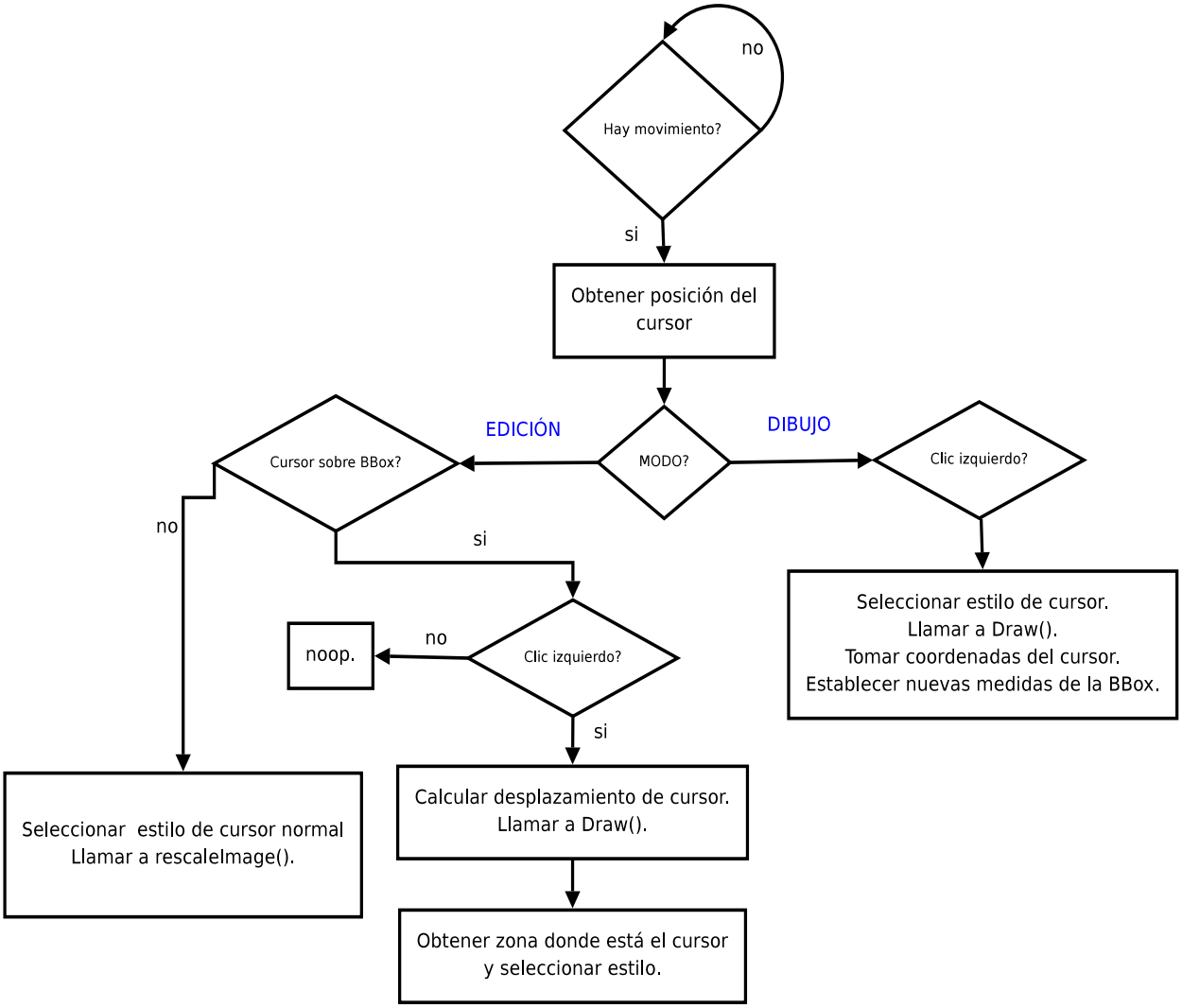

Figura 4.20: Acciones a realizar según el movimiento del cursor.

OnLeftUp(): En esta función se centra la atención en el evento de soltar el botón izquierdo del ratón, cuyo significado es el de dejar de dibujar o editar, según el modo del editor gráfico. Al liberar el botón en modo DIBUJO se usan las coordenadas finales del mouse para crear la Bounding Box con las dimensiones exactas hasta que el usuario decición dejar de dibujar. El método que usa python para dibujar tiene en cuenta el sentido hacia el cual se hizo el rectángulo,

utilizando anchuras y alturas negativas si el rectángulo se construyó en el sentido contrario respecto los ejes de abcisas y ordenadas, por esta razón y con tal de no trabajar con coordenadas negativas se llama a la función *reparar()*, explicada más adelante, la cual transforma las coordenadas a valores positivos sin modificar las dimensiones ni la posición originales de la Bounding Box. Seguidamente se vuelve a poner el modo del editor por defecto (EDICIÓN) y se actualiza el historial de la imagen (nuevo estado con una Bounding Box más en la lista). En el caso de que se estuviese modificando una Bounding Box existente (modo EDICIÓN), coprobamos que se haya trabajado sobre una Boundign Box de la lista, actualizamos el historial con las últimas coordenadas modificadas por el usuario, reseteamos las variables de control que diferencian si se quería mover o redimensionar el rectángulo y en caso en que la Bounding Box se haya modificado con éxito establecemos el modo en edición para evitar ambigüedades. Finalmente llamamos a la funcion *rescaleImage()* para dibujar la Bounding Box en su posición/dimensiones finales. Podemos ver el diagrma de flujo correspondiente a esta función en la figura 4.21.

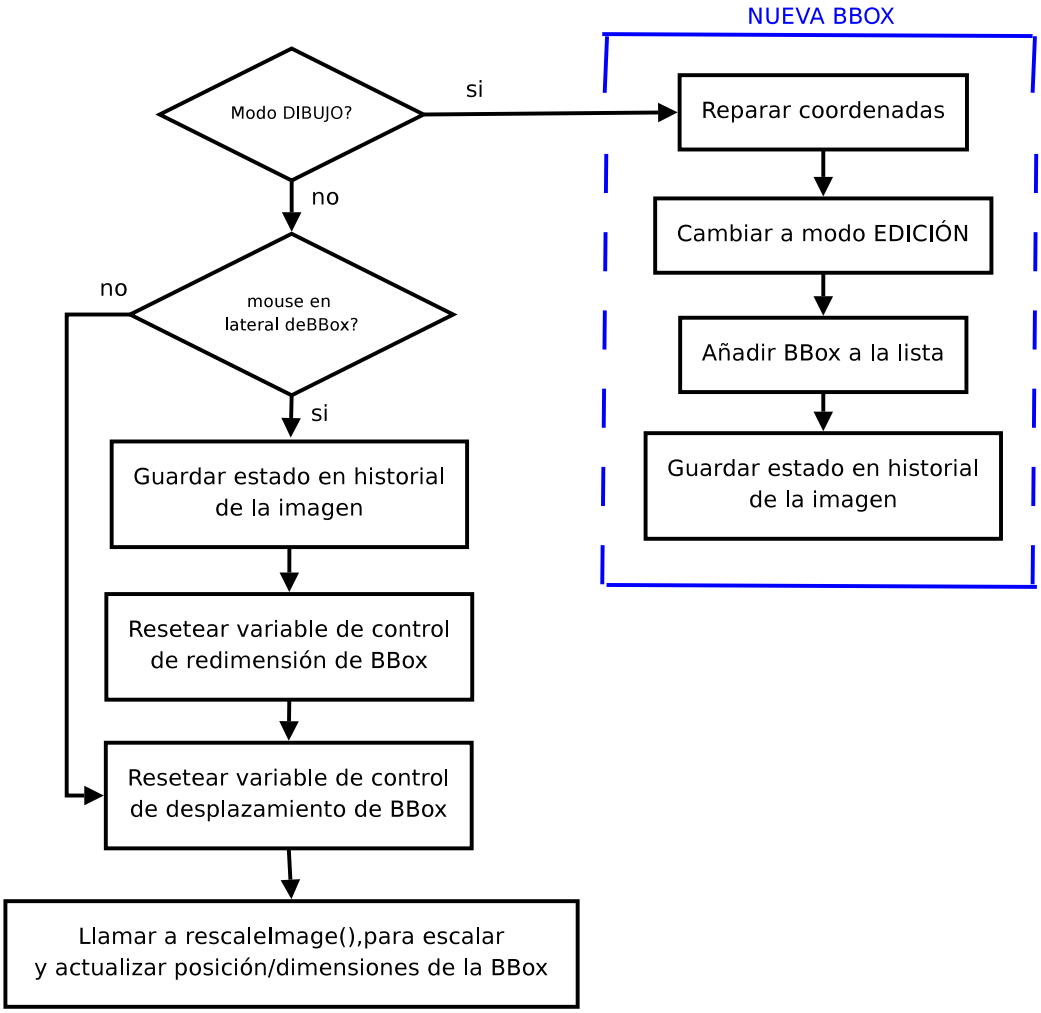

Figura 4.21: Operaciones que se realizan al soltar el botón izquierdo del mouse.

OnKeyDown(): Su función es controlar el evento de presionar una tecla, la cuales cambian entre los modos EDICIÓN, DIBUJO y DELETE, navegan por la imágenes, o deshace/rehace una acción. Cuando se presiona una tecla, se captura el evento y se procede según la tecla presionada y el modo del editor gráfico en el que se está. Para entender mejor el evento de presionar una tecla podemos seguir el diagrama de flujo de la figura 4.22.

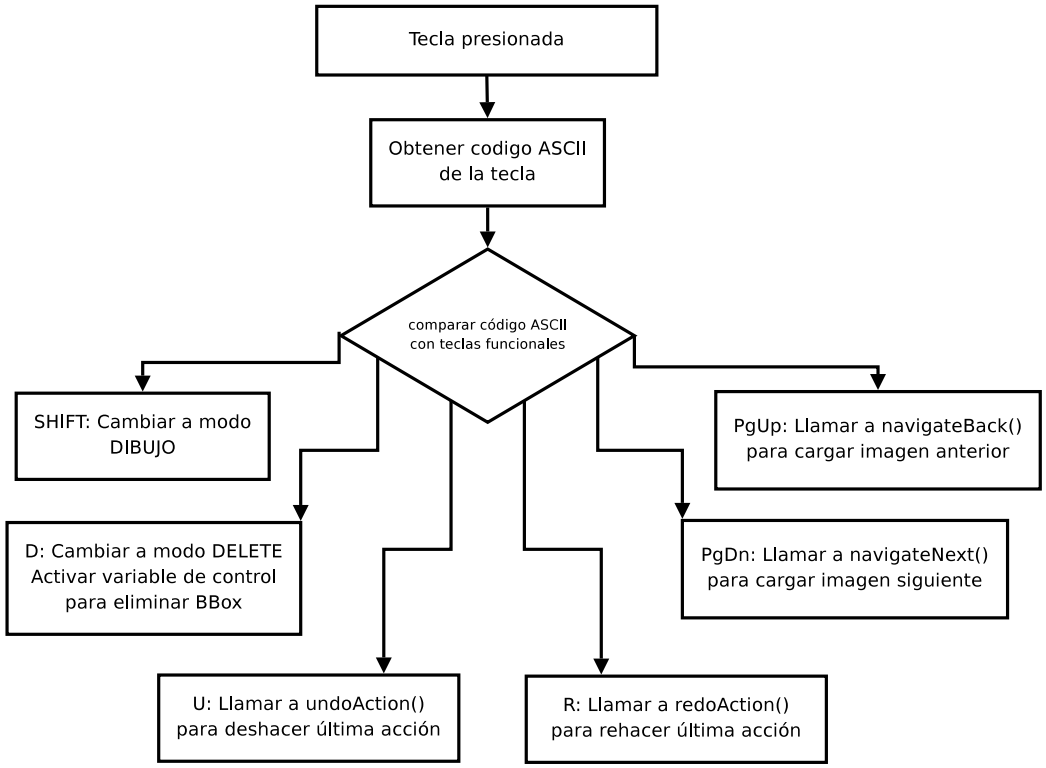

Figura 4.22: Funcionamiento correspondiente al presionar una tecla.

OnKeyUp(): Esta función controla el evento de soltar la tecla SHIFT o D, las cuales cambian de modo a EDICIÓN, ya que este es el modo por defecto del editor gráfico. En el caso de modo DELETE,que es una especialización del modo EDICIÓN, tan sólo se resetea la variable de control que se utiliza para eliminar una Bounding Box.

selectCursor(): Esta función se encarga de modificar el estilo del puntero del mouse, es una función orientada sólamente al diseño, con tal de conseguir una usabilidad y amigabilidad entre editor y usuario. Su control es muy sencillo, se llama a esta función cuando el ratón se mueve por la imagen, y se le pasa un parámetro que corresponde a la zona donde se encuentra el puntero. Según sea esta zona (lateral, esquina, interior de Bounding Box, o bien zona exterior) se modifica el estilo del puntero mediante los widgets disponibles para python.

checkBox(): Es una función de testeo, pruebas y control utilizada al principio de la construcción del editor gráfico. No tiene funcionalidad ninguna una vez el editor funcionaba correctamente. Su interés era simplemente para entender y trabajar con la orientación de los rectángulos cuando eran creados, y ayudó a implementar la función *reparar()*, perteneciente a la clase BBox, que se explica más adelante.

swPaint(): Es un procedimiento básico que llama a dos funciones. Esta función se creó para poder separar el proceso de dibujar la primera imagen sin modificaciones y con el proceso de redibujarla con sus Bounding Boxes correspondientes. Las funciones llamadas son *paintImage()* y *DrawBBoxes()*. La primera, como su nombre indica es para imprimir una imagen en pantalla, la segunda se encarga de dibujar todas las Bounding Boxes que hay en una lista, a continuación se estudian con más detalle.

paintImage(): La imagen ya convertida en objeto de tipo *bitmap* se monta y se dibuja en esta función, la cual llama al método de wxPython encargado de dibujar un *bitmap* en un área determinada.

Draw(): Este es un procemiento con la misión de llamar a la función encargada de pintar la Bounding Box actual (*Draw()* de la clase BBox), si está en modo EDICIÓN llama a las funciones que obtiene las coordenadas necesarias para modificar los atributos del objeto BBox, siempre teniendo en cuenta la conversión entre coordenadas reales y virtuales según la posición desplazada por las barras de desplazamiento (scrollbars) y al hacer zoom. En este modo se diferencia entre *panning* (mover el rectángulo sin modificar sus dimensiones) y *redimension* (redimensionar rectángulo). Si estamos en modo DIBUJO las coordenadas son las correspondientes a la posición del cursor, ya calculadas con anterioridad(ver figura 4.23). Cabe mencionar que antes de realizar alguna de estas acciones se refresca la imagen llamando a *paintImage()* para trabajar con los datos actualizados, tanto para el programa como para el usuario.

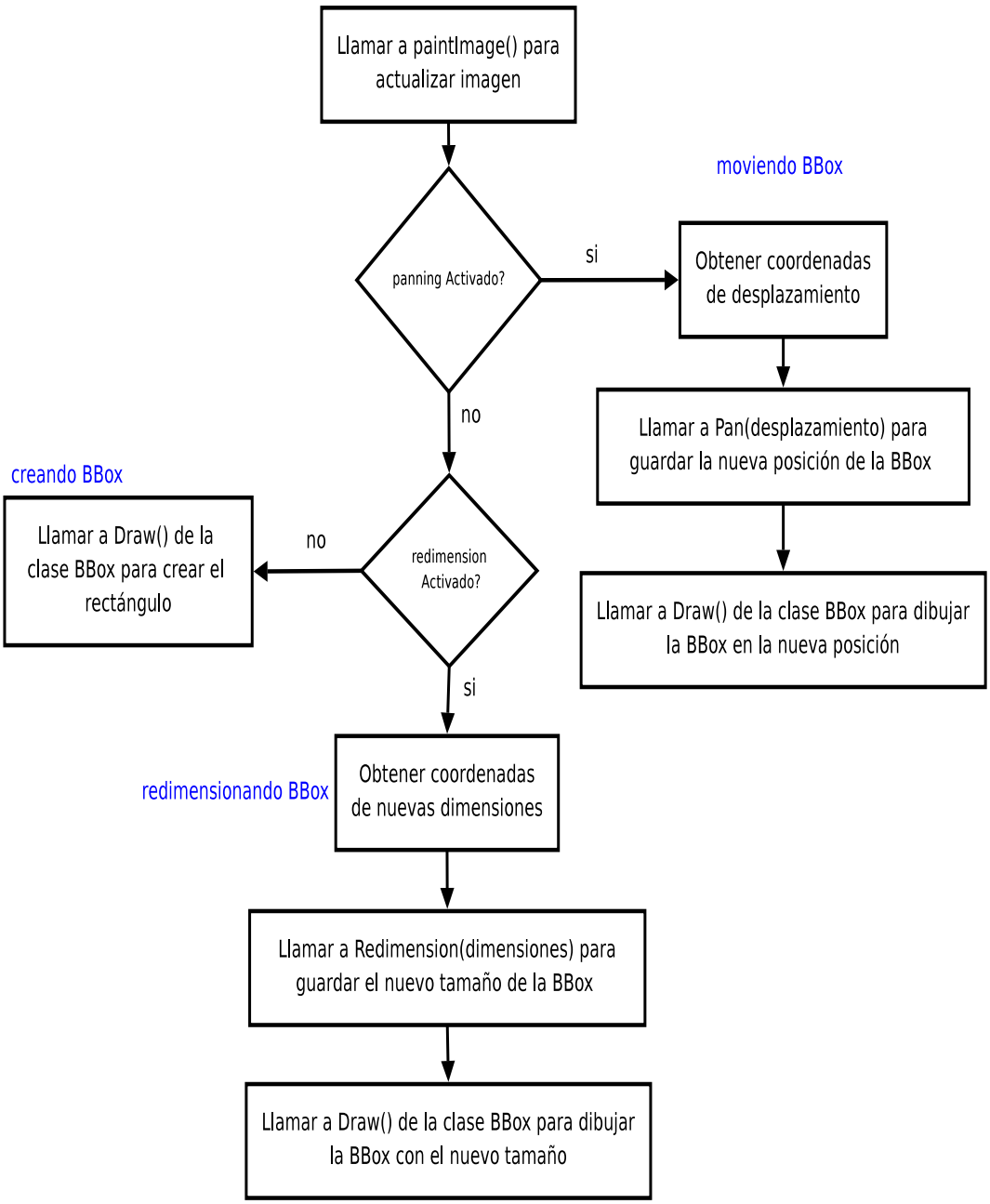

Figura 4.23: Método para pintar un rectángulo en la imagen.

deleteBbox(): Función que llama al método *remove()* de la clase BBox, encargado de eliminar una Bounding Box de la lista de todas

las Bounding Boxes de la imagen.

createBoundingBox(): Al dibujar una nueva BBox se llama a esta función, la cual obtiene la anchura, altura, id y nombre que se seleccionó de la lista de objetos, para crear un objeto BBox con estas propiedades, que se guardará en la lista de Bounding Boxes de la imagen. Una vez creada se llama a la función *DrawBBoxes()* para que actualice la imagen.

DrawBBoxes(): En esta función se preparan las propiedades del rectángulo (anchura de líneas, color, fondo transparente) y seguidamente recorremos la lista de Bounding Boxes, llamando a la función *Draw()* de la clase BBox por cada uno de los elementos de la lista, encargada de dibujar el rectángulo con las propiedades establecidas.Se diferencia entre el modo del programa, en caso de estar en modo VALIDATOR consulta las opciones que haya seleccionado el usuario para mostrar las BBoxes correspondientes al tipo de resultado que se desea visualizar (TP, FP, FN, Ground Truth o Results).

refreshBBoxes(): Función auxiliar que se llama cuando el usuario selecciona View>Refresh. Llama a la función *DrawBBoxes()* para refrescar la imagen y dibujar las Bounding Boxes existentes.

fitImageToWindow(): Esta es una sencilla función de encajar la imagen en las dimensiones de la pantalla. Así la ajustamos cuando el usuario modifica las dimensiones de la ventana principal (maximizar, minimizar o personalizar tamaño).

setEstadoActual(): Cuando navegamos sobre una lista de imágenes sobre la cual se ha trabajado es posible que se encuentren fallos o se quieran realizar modificaciones en una de las imágenes de la lista, para conservar las Bounding Boxes que se dibujaron sobre ella y las acciones que se relizaron, se dispone de una variable llamada estado, la cual contiene la lista y el historial de acciones de cada imagen, así como las listas de BBoxes que corresponden a los verdaderos positivos, falsos positivos, falsos negativos, fichero de ground truhth y

fichero de resultados cargados en el modo de validación. Así se hace posible escribir el estado de una imagen en las variables adecuadas, que se utilizan para representar las BBoxes en la imagen.

getEstadoActual(): De modo inverso a la función *setEstadoActual()*, esta función escribe las variables que contienen la lista de Bunding Boxes y el historial en los componenetes correspondientes del estado de la imagen sobre la cual se trabaja y lo retorna, de este modo podremos consultar el estado de esta imagen en un futuro.

saveImageState(): El estado de cada imagen se guarda en un diccionario que contiene nombres de imágenes, cada uno asociado a su estado. Por eso, cuando se crea un nuevo estado (o se modifica) en una imagen, este se debe guardar de algún modo. Es en esta función donde se realiza esta acción, llamando a la función *getEstadoActual()* que acabamos de explicar. La función *saveImageState()* se llama justo antes de navegar a otra imagen.

loadImageState(): Esta función es llamada justo despues de haber navegado a una nueva imagen o bien al cargar un fichero Ground Truth. En este caso, se pretende cargar el estado de una imagen sobre la cual se trabajó, para ello debemos obtener el estado de la imagen, por eso se llama a la función getEstado(), una vez tenemos el estado se le pasa a la función *setEstadoActual()*, para actualizar variables de imagen.

getEstado(): Función que se llama al navegar hacia una imagen, si la imagen ya tenía estado, busca el nombre de la misma en la lista de estados, en caso contrario crea un estado nuevo con parámetros vacíos (lista de Bounding Boxes e historial). Finalmente retorna el estado de la imagen.

Para comprender mejor el sentido las funciones relacionadas con el estado de la imagen veamos la figura 4.24, donde se muestra la correlación entre las funciones *setEstadoActual()*, *getEstadoActual()*, *saveImageState()*, *loadImageState()* y *getEstado()*:

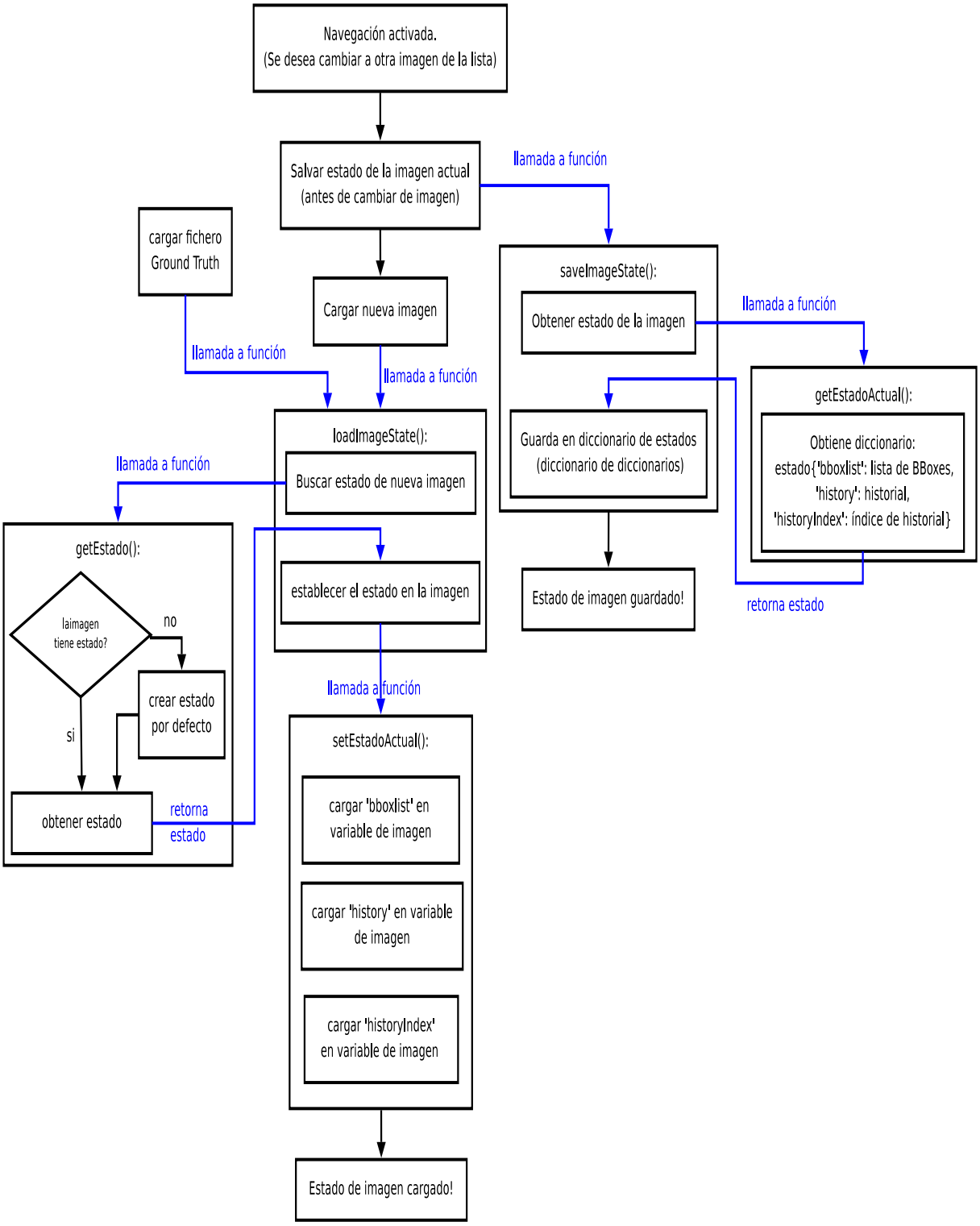

Figura 4.24: Diagrama que engloba las funciones relacionadas con el estado de una imagen.

rescaleImage(): Función designada para reescalar la imagen cuando se hace zoom in/out. Creamos un factor de zoom según se haya ampliado o reducido la imagen, lo aplicamos sobre ella tomando las coordenadas reales del mouse para hacer zoom desde esa posición, respetando los márgenes y ajustando las scrollbars. Incluye los eventos de clic, rueda y movimiento del mouse. La acción explícita que controla el factor de escala es el uso de la rueda del mouse, que es cuando se llama a la función *changeScale()* que pasamos a explicar a continuación.

changeScale(): Al capturar el movimiento de la rueda del mouse se llama a esta función, esta rotación será negativa o positiva dependiendo del sentido en el que se rota la rueda, de esta manera se diferencia el zoom in (ampliación de la imagen) del zoom out (reducción de la imagen), incrementando o decrementando un 5 % el tamaño de la imagen respectivamente.

OnFrameResize(): Es la función a la que se llama cuando se recoge el evento de redimensionar el tamaño de la ventana, en ella se obtiene el tamaño final, se establece dicho tamaño y se llama a la función *fitImageToWindow()* para encajarla correctamente.

historyUpdate(): Diseñada para actualizar la lista del historial de una imagen; se eliminan los estados no válidos, se añade la nueva lista de Bounding Boxes y se incrementa el índice del historial para apuntar a la última acción.

undoAction(): Comprobamos que el historial contenga alguna acción, en tal caso decrementamos el índice del historial, así apuntará a la acción anterior que contiene la antigua lista de Bounding Boxes, cargamos esta lista en la variable de la imagen y pintamos.

redoAction(): En este caso se comprueba que haya alguna acción por delante de la que está apuntando el índice del historial, si esta condición se cumple cargamos la lista de Bounding Boxes correspondiente del mismo modo que la función anterior y actualizamos dibujando

los rectángulos en la imagen.

close(): En esta función se procesa el evento de cerrar la ventana principal, provocando el cierre del resto de ventanas (lista de objetos y lista de propiedades)

version_message(): Lanza una ventana con información sobre el autor y la versión del programa. Es una función meramente informativa.

MODO VALIDATOR (Validador)

loadResults(): Función que abre un diálogo para cargar un fichero de resultados, que puede tener el mismo formato que un fichero Ground Truth, y contiene las BBoxes resultado de realizar un reconocimiento de objetos por el robot, es decir un dataset.

selectBBoxes(): Esta función se llama cuando el usuario seleccion la opción de menú View>BBoxes, lanza un diálogo de multiselección para que el usuario escoja el tipo de resultados que desea visualizar, se pueden seleccionar por separado o bien todos al mismo tiempo, según convenga. La función guarda las opciones seleccionadas, activando los flags correspodientes para dibujar los resultados por pantalla, estos flags indexan las listas de Bounding Boxes para representar los Verdaderos Positivos (TP), Falsos Positivos (FP), Falsos Negativos (FN), el Ground Truth o los resultados. En caso de que se produzca un error se lanzan mensajes por pantalla para informar al usuario.

validateResults(): Una vez se han cargado los ficheros de Ground Truth y deresultados en memoria, se realiza una comparativa entre las BBoxes que representan un mismo objeto. Para cada BBox del GT se calcula el solapamiento que existe entre esta y todas las que existan en el fichero de resultados para el mismo objeto mediante la llamada a la funcion overlaps() de la clase BBox; si el área de solapamiento es mayor que el 50 % del área total de ambas cajas se toma esa muestra del

fichero de resultados como un acierto (TP). En caso de no encontrar ninguna BBox en el fichero de resultados que corresponda al objeto que etiqueta la BBox del Ground Truth, se iterpreta a ésta como un falso negativo (FN), ya que no reconoce un objeto donde sí lo debería hacer; finalmente, en caso de quedar BBoxes en los resultados que no se han solapado lo suficiente se interpretan como falsos pisitivos, puesto que ha localizado un objeto donde no está. Todas las BBoxes se van añadiendo a las listas correspondientes para ser representadas cuando el usuario desee visualizarlas. Esta clasificación de resultados la podemos observar en la figura 4.25.

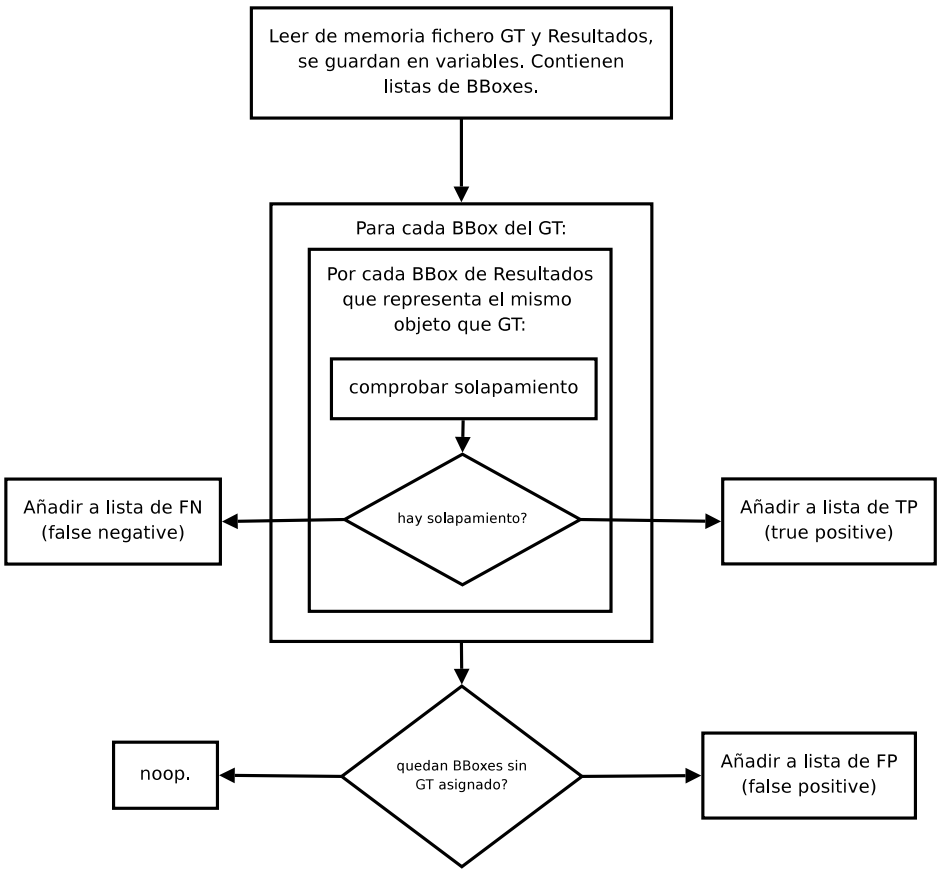

Figura 4.25: Método mediante el cual se clasifican los resultados.

2. Ventana de etiquetas de objetos.

Clase Objects

Clase para mostar la ventana con lista de objetos. Está formada por:

- Constructor, donde se especifica el nombre de la ventana, la localización donde se lanzará y dimensiones exactas. Esta clase es mucho más sencilla que la anterior, también contiene un pequeño menú para cargar la lista de objetos con la que se desee trabajar, el cual se contruye del mismo modo explicado en la clase ImageFrame. Contiene el manejador de eventos para cambiar el tamaño de la ventana.
- Funciones:

  getList(): Función que muestra el diálogo para seleccionar una lista de objetos de un directorio. Segmentamos la lista en una linea por cada etiqueta. Mostramos todas las etiquetas de la lista en la ventana.

  getListMenu(): Función de acceso externo a *getList()*.

  OnResize(): Esta función controla el evento de cambiar de tamaño la ventana, llama al método necesario para modificar las dimensiones de la lista si se cambian las dimensiones de la ventana. En esta función se captura el evento de hacer clic sobre una etiqueta.

  OnLeftDown(): Se llama a esta función desde *OnResize()*, cuando el clic del ratón es capturado, obtiene el nombre y el id del la etiqueta sobre la cual se ha hecho clic y guarda los datos para trabajar con ellos según convenga.

  GetSelectedItem(): Retorna el identificador de la etiqueta actual sobre la que se ha hecho clic.

  GetSelectedLabel(): Retorna el nombre de la etiqueta actual sobre la que se ha hecho clic.

  idToLabel(): En esta función nos encargamos de retornar el nombre de la etiqueta correspondiente a un identificador que le pasamos.

objectClose(): Función que cierra la ventana de objetos.

3. Ventana de propiedades visuales.

Clase Properties

Clase para mostar la ventana con lista de propiedades. Funciona exactamente del mismo modo que la clase Objects, y se compone de:

- Constructor, especifa nombre, posición y dimensiones de la ventana, crea el menú para cargar la lista y maneja el evento de redimensionar la ventana.
- Funciones:

  setProperties(): Abre el diálogo necesario para cargar la lista de propiedades visuales correspondientes a una imagen o secuencia.

  getPropList(): Función de acceso externo a setProperties(), devuelve la lista de propiedades cargada en esta función.

  OnResize(): Esta función controla el evento de cambiar de tamaño la ventana, llama al método necesario para modificar las dimensiones de la lista si se cambian las dimensiones de la ventana. En esta función se captura el evento de hacer clic sobre una etiqueta.

  OnLeftDown(): Se llama a esta función desde OnResize(), cuando el clic del ratón es capturado, obtiene el nombre y el id de la propiedad sobre la cual se ha hecho clic y guarda los datos para trabajar con ellos según convenga.

  GetSelectedItem(): Retorna el identificador de la propiedad actual sobre la que se ha hecho clic.

  GetSelectedLabel(): Retorna el nombre de la propiedad actual sobre la que se ha hecho clic.

  propClose(): Cierra la ventana de propiedades.

4.4.2. Abstracciones

Clase BBox

Clase para mostar la localización los objetos. Se encarga de dibujar la Bounding Box encuadrando el objeto en una región de la imagen a partir de unas coordenadas concretas. Esta clase está formada por:

- Constructor, define los atributos necesarios que debe tener una Bounding Box para que pueda ser representada en una imagen. Estos atributos son:

  id: identificador, es el índice del objeto en la lista de etiquetas;

  name: nombre de la etiqueta u objeto que encuadra la BBox;

  pos: posicion, coordenas bidimensionales donde se encuentra la esquina inicial del rectángulo;

  size: tamaño que posee la caja, representado como una tupla de las dimensiones ancho y alto;

  properties: propiedades visuales del objeto en la imagen;

  En el constructor se inicializan las variables necesarias correspondientes a las propiedades mencionadas, así como el resto de variables que se requieren para trabajar con ellas, como tipo, modo de editor grafico, lista de Bounding Boxes y estilo de cursor.

- Funciones:

  Draw(): Esta función es la encargada de dibujar el perímetro del rectangulo que forma la Bounding Box, para ello se define un color de línea por defecto, el azul, que se cambiará según el modo del editor gráfico (rojo en EDICIÓN) o la posición del cursor (verde si está encima). A continuación se pasa a dibujar la BBox según las coordenas que se hayan capturado como hemos explicado en la clase ImageFrame. Como acabamos de decir, según la posición del cursor se pinta la línea correspondiente en verde, o se dibuja una zona extra para resaltar que el cursor está sobre una esquina o en el interior de una BBox, cada zona está asociada a una función, y para escojer la

zona se hace mediante un diccionario que contiene llamadas a las funciones que explicaremos a continuación.

dibujaA(),dibujaB(),dibujaC(),dibujaD(): Funciones para dibujar un pequeño cuadrado sobre la esquina correspondiente, esta función será llamada cuando el mouse pasa por encima de una esquina, por eso mismo se aprovecha para cambiar el estilo de mouse y así comunicar visualmente al usuario que se encuentra en una esquina de una Bounding Box. Veamos la figura 4.26 para entender el mecanismo visual explicado.

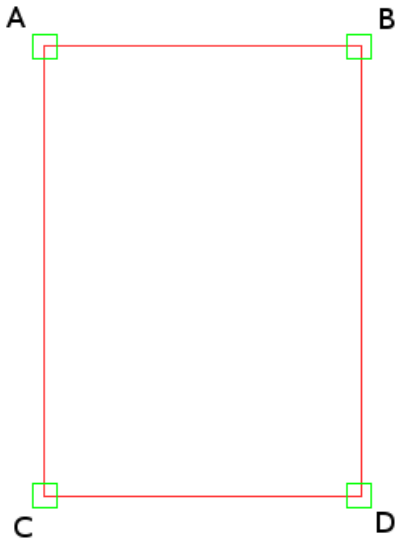

Figura 4.26: Figura que muestra cómo se remarca la zona de las esquinas.

dibujaAB(),dibujaCD(),dibujaAC(),dibujaBD(): En este caso, las funciones lo que hacen es pintar de color verde el lateral que se encuentra entre las esquinas indicadas en el nombre de cada una, señalando que el cursor está en un lateral y sólo se podrá aumentar o disminuir el tamaño de la Bounding Box a partir de ese lateral. En la figura 4.27 se muestra el resultado de aplicar la función dibujaAC(), es decir se pinta el lateral comprendido entre la esquina A y la C. El resto de las funciones son equivalentes.

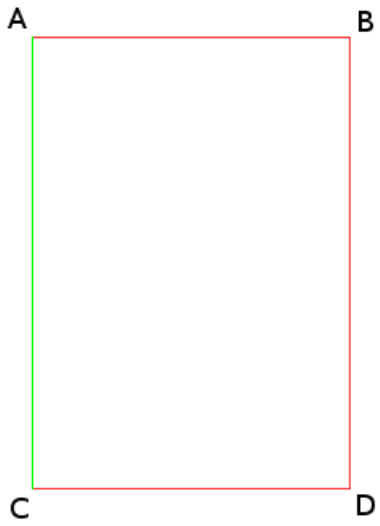

Figura 4.27: Figura que muestra cómo se remarcan los laterales.

dibujaPan(): Es una función para pintar un rectángulo de color verde en el interior de la Bounding Box, con la idea de indicar que el cursor está dentro de los límites de la misma y se puede hacer panning de ésta, es decir, mover la BBox por la imagen sin modificar sus dimensiones. Podemos ver cómo sería el resultado de llamar a esta función si observamos la figura 4.28.

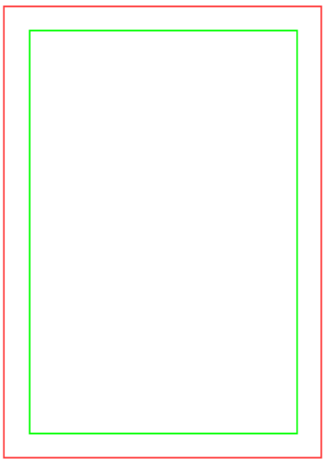

Figura 4.28: Figura que muestra cómo se remarca el interior de la BBox.

mouseOverBbox(): Función que recibe como parámetros la posición del ratón y la escala actual con tal de comprobar si el mouse esta encima de alguna de las BBoxes de la lista. Se compara esta posición con la región donde se encuentran cada una de la BBoxes de la lista de la imagen actual, teniendo en cuenta un margen de acción sobre el cual se activan los efectos visuales. En caso de que el cursor se encuentre sobre alguna de las zonas de interés se devuelve el identificador de esa zona, y se activan las variables que permiten la redimensión o el panning de la BBox.

overlaps(): Esta función recibe una BBox del fichero de resultados como parámetro, en ella analizamos el solapamiento que existe entre esta BBox y la que representa al mismo objeto en el fichero de Ground truth. Tomamos las posiciones máximas y mínimas de ambas cajas y calculamos el área de solapamiento, en caso de que exista, se dividen las áreas obteniendo un ratio que representa el porcentaje de intersección, si este es igual o mayor que el 50 % del área que representa la unión de ambas superficies, entondes se retorna un valor lógico True, que indica un acierto, represen-

tado como verdadero positivo. Si es menor del 50 %, o bien si no existe solapamiento devolverá un False, clasificando como falso pisitivo.

SetSize(): Actualiza los valores de la variable que establecen un nuevo tamaño de la BBox.

GetSize(): Retorna los valores actuales correspondientes a las dimensiones de la BBox (anchura y altura).

SetPos(): Actualiza los valores de la variable que establecen una nueva posición inicial de la BBox.

GetPos(): Retorna los valores actuales correspondientes a la posición inicial de la BBox.

SetModo(): Establece el modo del editor gráfico (DIBUJO, EDICIÓN).

GetModo(): Retorna el modo del editor gráfico (DIBUJO, EDICIÓN).

GetZone(): Retorna el identificador de la zona sobre la que se encuentra el cursor del mouse.

GetCornerType(): Función auxiliar que retorna el tipo de la esquina, usada inicialmente para verificar la orientación de la BBox creada. Esta función es obsoleta y ha sido sustituída por la función reparar().

GetPropId(): Retorna el identificador de la propiedad que tiene el objeto que encuadra la BBox.

GetObjectId(): Retorna el identificador del nombre del objeto que encuadra la BBox.

GetLabel(): Retorna el nombre del objeto que encuadra la BBox.

IsPanning(): Retorna el valor de la variable que indica si el cursor está en el interior de la BBox, la cual controla la acción de panning.

Pan(): Esta función implementa los cálculos necesarios para mover la BBox entera por la pantalla(panning). Recibe un offset y devuelve la nueva posición en la que se debe dibujar la BBox despues de moverla.

Redimension(): Función para cambiar las dimensiones de una BBox. Recibe posicion final, comprueba comprueba sobre qué zona se desea redimensionar (laterale o esquinas) y devuelve las nuevas dimensiones para dibujarla despues de redimensionar.

reparar(): El sentido en el que se crea una BBox puede dar un resultado con dimensiones negativas, dependiendo hacia dónde se mueve el ratón cuando se está dibujando, respecto al origen de coordenadas. Trabajar con valores negativos complicaba la implementación más de lo necesario, por eso hemos decidido crear esta función que convierte estos valores en absolutos sin modificar las dimensiones, este hecho se entiende como reparar una BBox.

SetBoxLabel(): Es la función encargada de imprimir la etiqueta con el nombre del objeto que encuadra la BBox, sobre la imagen y dentro del rectángulo, para ello obtiene el nombre del objeto, las coordenadas de la BBox y calcula una distancia relativa para imprimir la etiqueta aproximadamente en el centro del rectángulo.

Clase Tracker

Esta clase contiene los procedimientos encargados del seguimiento de objetos en imágenes correlativas. Es el cerebro de la automatización de la herramienta, implementada con las librerías de OpenCV, la clase está formada por:

- Constructor, el cual inicializa los parámetros Hue y los que van a componer las barras de los histogramas de la región de interés, realiza las conversiones de imagen necesarias, las máscaras de píxels y probabilidad de haber encontrado el centro de masas.

- Funciones:

  createTracker(): Crea las imágenes para el modelo de color HSV, para el canal Hue, para la máscara de píxels, los histogramas y el centro de masas.

releaseTracker(): Libera los recursos ocupados por la función anterior, así se agiliza el mecanismo y se reduce el coste.

startTracking(): Crea el tracker y calcula valores necesarios para realizar un histograma y poder decicir hacia dónde se desplazará la región de interés.

track(): Es la función que se encarga de realizar el seguimiento del objeto en la región de interés de una imagen sobre la siguiente imagen. Se desarrolla mediante el algoritmo Camshift.

updateHueImage(): Se ajustan los parámetros visuales del canal Hue. Estos parámetros se han determinado mediante un test sobre un conjunto de imágenes, y quedan fijos.

4.5. Test y pruebas

El proceso de automatización en el seguimiento de objetos (*tracking*) se ha tenido que testear hasta llegar a una solución aceptable, modificando los parámetros de color, saturación y brillo que se utilizan para crear los histogramas, hasta llegar a un compromiso entre ellos que garantizase un equilibrio para las diferentes imágenes. Para probar el proceso de *tracking* se ha creado un pequeño test, llamado *TestTracker()*, encargado de analizar imágenes de 2 en 2 localizando el mismo objeto en ambas, de manera que se podían modificar los valores máximos y mínimos para el componente Hue [16],[12] del modelo HSV de la imagen, y recalcular posteriormente el histograma, con la idea de encontrar una combinación en que la Bounding Box encuadrase el objeto del modo más preciso. La modificación de estos parámetros (vmin, vmax y smin)se realizan mediante el teclado y son manejados por el handler de eventos siguiendo el diagrama de la figura 4.29, repitiendo el proceso para un conjunto de pares de imágenes hasta verificar que el mecanismo de predicción era válido para incluirlo en la herramienta.

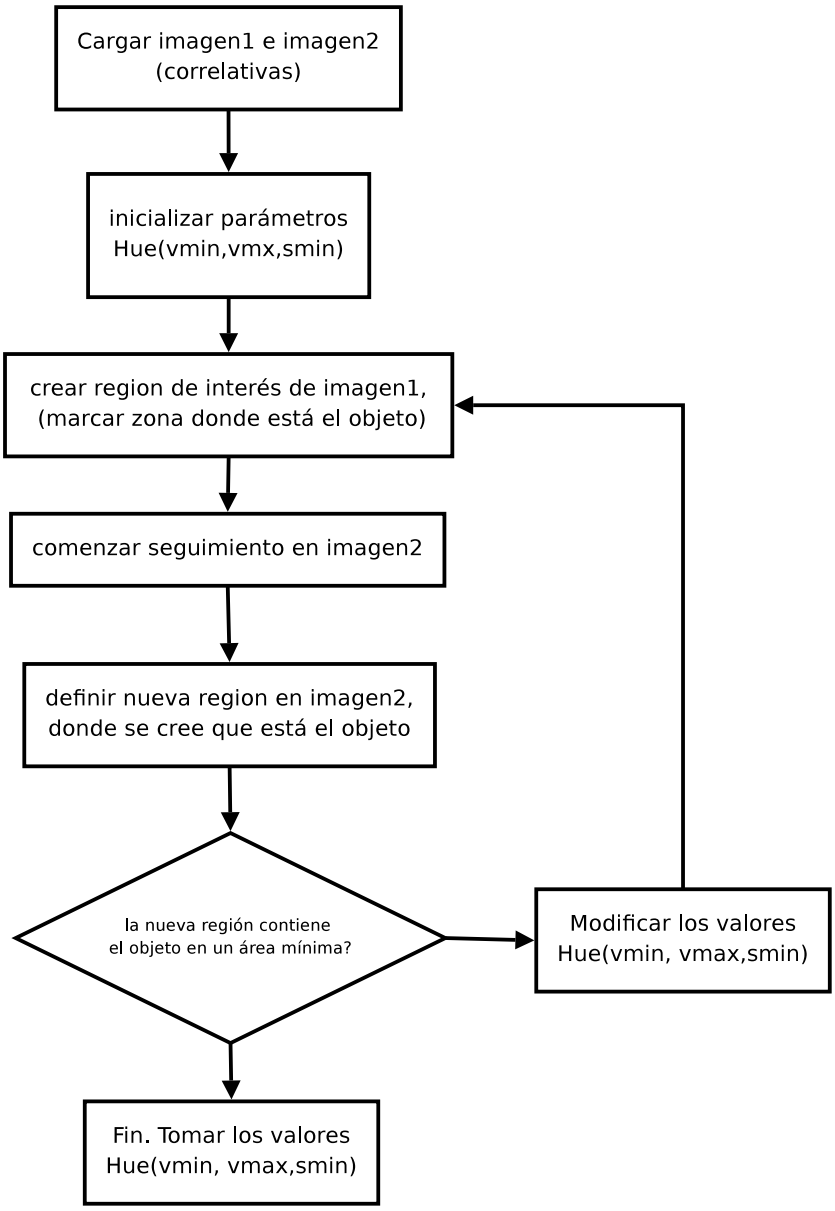

Figura 4.29: Diagrama de testeo de parámetros Hue para encontrar valores adecuados hasta lograr funcionamiento correcto.

Capítulo 5

Conclusiones

En este trabajo se ha estudiado la posibilidad de optimizar el proceso de etiquetaje de imágenes para el entrenamiento en el reconocimiento de objetos de un robot. Este proceso consiste en encuadrar cada objeto de interés que hay en una secuencia de imágenes o panorama, etiquetarlo con su nombre y guardar la posición donde se encuentra en cada imagen. Posteriormente se compara el etiquetaje realizado por el usuario, conocido como *Ground Truth*, con el etiquetaje que realiza el robot y se evaluan los resultados.

Con la idea de reducir el tiempo de etiquetaje, se ha diseñado una herramienta para etiquetar una secuencia de imágenes de forma que los mismos objetos en imágenes correlativas se puedan etiquetar automáticamente con un mínimo esfuerzo por parte del usuario, logrando minimizar el tiempo para evaluar el aprendizaje de reconocimiento de objetos en un robot.

El trabajo desarrollado ha alcanzado todos los objetivos propuestos en la planificación, se ha implementado un interfaz gráfico capaz de editar imágenes para encuadrar objetos de interés etiquetándolos visualmente. Todas las operaciones que se realizan sobre una serie de imágenes se puede guardar en ficheros para ser recuperado, modificado y continuado cuando se requiera. El programa se acopla perfectamente a todos los requisitos necesarios para importar secuencias de imágenes y ficheros Ground Truth previamente creados, con la idea de agilizar el trabajo hecho hasta ahora por el departamento de Robótica del IIIA-CSIC, en el

campo de Reconocimiento de Objetos en Panoramas y su aplicación en la localización de robots.

También se ha implementado un proceso de seguimiento y predicción de objetos para un etiquetaje automático, con el propósito de agilizar el proceso de creación del Ground Truth, es evidente que para un gran número de imágenes en una secuencia de vídeo, el mecanismo de predicción de objetos ahorrará una cantidad de tiempo nada despreciable, con lo que el proceso de validación de los resultados del entrenamiento para el robot encargado de reconocer objetos se verá optimizado notablemente.

Sin embargo, durante la construcción de la herramienta, hemos visto que se pueden incluir gran cantidad de detalles y ampliaciones en el editor gráfico, como una caja de herramientas que generalice el uso del programa para otras aplicaciones, opciones de configuración para grosor y color de las BBoxes, mejoras en el algoritmo de predicción e incluso la posibilidad de integrar un sintetizador de vídeo que muestre los resultados obtenidos, de modo que se pueda apreciar contínuamente la evolución del trabajo que se va realizando.

5.1. Posibles aplicaciones futuras.

Hay varias aplicaciones hacia las cuales se podría orientar esta herramienta:

- Etiquetaje de imágenes para multitud de aplicaciones, así como el etiquetaje de monumentos, fenómenos de la naturaleza, fauna y flora, o incluso personas para álbumes de fotos digitales, panoramas o depósitos digitales de vídeos y fotografías.

- También es factible fusionarlo con otros proyectos, como el 1034, *Estudi d'interfícies multitàctils i multiusuàri* [19], realizado este año en la UAB por Antoni Gurguí Valverde, creando un método de etiquetaje multitáctil, lo cual reduciría aún más el tiempo necesario para tal efecto.

Bibliografía

[1] Programmig Python; Mark Lutz - O'Reilly, 2nd Edition

[2] Python para todos; Raúl González Duque - Creative Commons Attibution Share Alike 2.0

[3] Python/C API Reference Manual, Release 2.6
http://docs.python.org/c-api/

[4] wxPython 2.8.9.2, Package wx (Python's widgets)
http://www.wxpython.org/docs/api/

[5] Learning OpenCV, Computer Vision with the OpenCV Library; GaryBradski & Adrian Kaebler - O'Reilly

[6] The Not So Short Introduction to LaTEX 2E, Or LaTEX 2E in 90 minutes; by Tobias Oetiker, Hubert Partl, Irene Hyna and Elisabeth Schlegl, Version 3.16, 25 September, 2000
http://www.ccd.uab.es/ jaume/latex/lshort.pdf

[7] Hypertext Help with LaTEX
http://www.ccd.uab.es/ sergi/latex2e-html/ltx-2.html

[8] Open Source Computer Vision Library - Reference Manual, Intel
http://www.cs.unc.edu/Research/stc/FAQs/OpenCV/OpenCVReferenceManual.pdf

[9] OpenCV 1.1 Python Reference, Motion Analysis and Object Tracking Reference

http://opencv.willowgarage.com/documentation/python/
motion_analysis_and_object_tracking_reference.html

[10] camshift_wrapper.c - by Robin Hewitt, 2007
http://www.cognotics.com/opencv/downloads/camshift_wrapper

[11] ctypes-opencv, A Python wrapper for OpenCV using ctypes
http://code.google.com/p/ctypes-opencv/

[12] Face detection using OpenCV
http://opencv.willowgarage.com/wiki/FaceDetection

[13] OpenCV and Python
http://www.depthfirstsearch.net/blog/2008/09/22/opencv-and-python/comment-page-1/

[14] Reconocimiento de Objetos en Panoramas (y su aplicación a la localización de robots); Arturo Ribes Sanz, Proyecto Final de Carrera 2008, Escola Técnica Superior de Enginyeria(UAB)

[15] Reporte e Investigación del método Mean Shift, descripción del mismo y aplicaciones a Filtrado preservando bordes, Segmentación y Tracking de objetos no rígidos en tiempo Real - Alonso Ramírez Manzanares. CIMAT
http://www.cimat.mx/ alram/VC/MSAA.htm

[16] CAMSHIFT Tracker Design Experiments with Intel OpenCV and SAI - Alexandre R.J.François
http://pollux.usc.edu/ afrancoi/pdf/camshift-tr.pdf

[17] CAMSHIFT Experiments - Alexandre R.J.François
http://mfsm.sourceforge.net/MFSM_0.7/tutorials/Camshift.html

[18] Introducción a Latex: figuras y tablas - Antonio Gabriel López, Sergio Alonso y Carlos Porcel
http://hallsi.ugr.es/cursoLatex/cursoLaTeX_3_1.pdf

[19] Estudi d'interfícies multitàctils i multiusuàri basada en visió per computador; Antoni Gurguí Valverde, Enginyeria Informàtica. Projecte Final de Carrera 2009, Escola Técnica Superior en Enginyeria (UAB)

Firmado:

Bellaterra, Septiembre de 2009

Resumen

La adaptación del reconocimiento de objetos sobre la robótica móvil requiere un enfoque y nuevas aplicaciones que optimicen el entrenamiento de los robots para obtener resultados satisfactorios. Es conocido que el proceso de entrenamiento es largo y tedioso, donde la intervención humana es absolutamente necesaria para supervisar el comportamiento del robot y la dirección hacia los objetivos. Es por esta razón que se ha desarrollado una herramienta que reduce notablemente el esfuerzo humano que se debe hacer para esta supervisión, automatizando el proceso necesario para obtener una evaluación de resultados, y minimizando el tiempo que se malgasta debido a errores humanos o falta de infraestructuras.

Resum

La adaptació del reconeixement d'objectes envers la robótica mòbil requereix un enfocament i noves aplicacions que optimitzin l'entrenament dels robots per obtenir resultats satisfactoris. Es coneix que el procès d'entrenament és llarg i dur, on la intervenció humana és absolutament necessària per tal de supervisar el comportament del robot i la direcció cap els objectius. És per aquesta raó que s'ha desenvolupat una eina que redueix considerablement l'esforç humà que s'ha de fer per aquesta supervisió, automatitzant el procès necessari per a obtenir una evaluació de resultats, i minimitzant el temps que es malgasta degut a errors humasn o falta d'infrastructures.

Abstract

Adaptation of Object recognition to mobile robotics needs to change the point of view, as new applications to optimize the robot's training to achieve satisfactory results. It is known that trainning process is long, hard, and there is an absolutely necessary factor to supervise robot's behaviour and the direcction to the milestones: human intervention. This is the reason for the development of this tool, in order to reduce human effort to realize this supervision, automating the process to get a results evaluation, minimizing wasted time because of human errors or lack of infrastructures.