ESCUELA POLITÉCNICA SUPERIOR DE GANDÍA
INGENIERÍA TÉCNICA DE TELECOMUNICACIONES

UNIVERSIDAD POLITECNICA DE VALENCIA

Proyecto final de carrera

Reconocimiento de objetos mediante WebCam en tiempo real

Autor: Steve Rossius
Director: Ignacio Bosch Roig

Índice

Introducción

El presente Proyecto Final de Carrera (PFC) se ha realizado dentro del Grupo de Tratamiento de Señal (GTS) del Departamento de Comunicaciones de la Universidad Politécnica de Valencia, y tiene como finalidad la implementación de un sistema en tiempo real de captura, detección y seguimiento de objetos mediante *WebCam*.

El GTS forma parte de uno de los grupos de investigación del Instituto de Telecomunicaciones y Aplicaciones Multimedia (*iTEAM*). El *iTEAM* es un centro de investigación integrado en la Ciudad Politécnica de la Innovación, el nuevo parque científico de la Universidad Politécnica de Valencia(UPV), donde se desarrollan actividades de Investigación, Desarrollo e innovación (I+D+i) dentro del área de las Tecnologías de la Información y las Comunicaciones. El instituto está formado por 9 grupos de investigación reconocidos por la UPV, entre los cuáles se encuentra el Grupo de Tratamiento de Señal. Éste se dedica al tratamiento de señales bi o multidimensionales y particularmente de imágenes. Entre sus áreas de trabajo en I+D+i cabe destacar:

- Tratamiento de señales ultrasónicas
- Tratamiento de señales infrarrojas
- Tratamiento de señales láser-ultrasonidos para control de calidad sin contacto
- Restauración de imágenes y películas antiguas
- Análisis digital para la detección, caracterización y clasificación de objetos en imágenes
- Tratamiento digital de la información multimedia
- Recuperación e indexación de información multimedia
- Técnicas de compresión de video
- Marcas de agua de señales para la autenticación e identificación de señales

Este proyecto final de carrera se ha dividido en varios apartados con el fin de organizar y entender mejor las estrechas relaciones que hay entre las distintas temáticas expuestas.

El primer apartado introduce el estado de arte del reconocimiento de objetos, exponiendo diversas metodologías y soluciones implementadas.

En el segundo apartado se expondrán diversas técnicas del tratamiento digital de imágenes como introducción para su futuro uso en el desarrollo de la aplicación. Además se introduce conceptos básicos y fundamentales para

un entendimiento de la complejidad del reconocimiento llevado a cabo en este proyecto.

En el tercer apartado, se da una explicación detallada de cada una de las fases de las que consta la aplicación, sus funciones básicas y opciones de configuración.

En el cuarto apartado se procede a describir el proceso de la creación de una interfaz gráfica vinculada a la aplicación.

En el quinto apartado se exponen las conclusiones a las que se ha llegado así como futuras líneas de investigación y profundización para ampliar el código.

Objetivos

En este proyecto final de carrera se pretende dar una descripción detallada de un reconocedor de objetos en tiempo real. El objetivo principal de este trabajo es el desarrollo de una aplicación capaz de diferenciar y reconocer objetos en movimiento. Los objetos estudiados en este proyecto son todos formas cerradas de complejidades bajas. La aplicación desarrollada se puede dividir en tres etapas cíclicas: La adquisición, el procesado, y la interpretación.

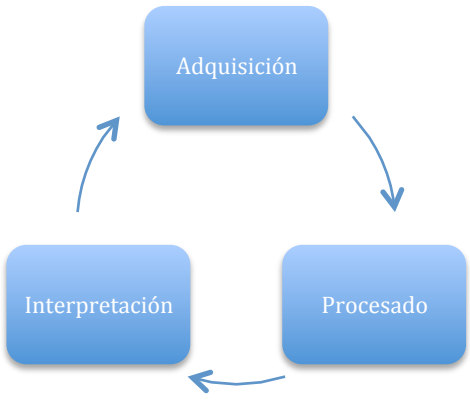


fig. 1 Esquema general de la aplicación

En la primera fase, la adquisición, se generan las imágenes con las que va trabajar el algoritmo y la captura de dichas imágenes ocurre mediante un dispositivo óptico de video con capacidad de ser configurado y manipulado desde un ordenador. En especifico, para este proyecto, se ha utilizado una *WebCam* con una resolución de 0.3 megapíxeles.

El algoritmo de la segunda fase, el procesado de lo objetos, se programara en un entorno de *MatLab* posterior o igual a la versión R2009A. Se realizará un algoritmo de segmentación capaz de diferenciar el fondo y los objetos y otro algoritmo de extracción de características capaz de encontrar en la imagen todos aquellos elementos que pueden ser de interés para el reconocimiento de los objetos contenidos en la imagen.

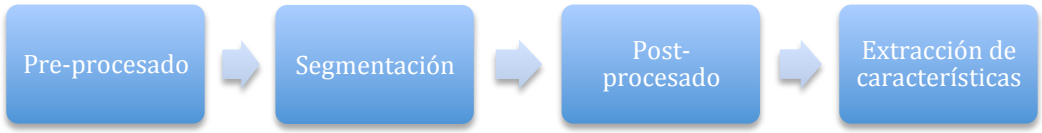


Fig.2 Esquema del procesado

La tercera fase descrita como la interpretación se basa en la comparación de los objetos encontrados en la imagen con aquellos objetos existentes en una base de datos que se actualizará conforme se procesan las imágenes. El seguimiento de estos objetos ocurrirá únicamente en planos paralelos al de la imagen, es decir no se estudiarán distorsiones visuales debidos a la perspectiva.

Los objetivos secundarios se describen como:

-El estudio de la captura en tiempo real mediante el Image acquisition toolbox de *MatLab*
-Estudio de la generación de una interfaz gráfica para la manipulación de la aplicación

1. La inteligencia artificial

El avance tecnológico que se ha producido en las ultimas décadas en el campo de la ingeniería informática, ha creado multitud de nuevas ramas científicas y campos de investigación. La tecnología, y en especial los ordenadores, están cada vez mas ligados e integrados en nuestra vida cotidiana. La mayoría de los dispositivos móviles poseen hoy en día mas capacidad computativa que cualquier ordenador de hace una década. La capacidad de disminuir cada vez mas los transistores que forman los procesadores, esta llegando a limites impuestos por la propia naturaleza de la electricidad.

Sin duda alguna, uno de los campos mas interesantes y revolucionarios es la robótica y su estrecha relación con la inteligencia artificial. Ya no hace falta dominar un lenguaje computacional para interactuar con los sistemas inteligentes. Los sistemas de inteligencia artificial ya son capaces de entender y procesar nuestras ordenes empleando el lenguaje natural. Estamos acostumbrados a ver equipos inteligentes completamente autónomos, en entornos estrictamente industriales o por lo menos altamente controlados como fabricas de coches. La aplicación de estos sistemas en entornos más dinámicos como nuestros hogares o oficinas, necesita de una enorme cantidad de sensores que permita al sistema inteligente percibir el entorno con toda su complejidad para interactuar con él.

Igual que en los humanos, la entrada de datos más importante es la visión. En un sistema inteligente, normalmente se trata de una cámara compuesta por multitud de lentes y sensores específicos para cada sistema y campo de aplicación. Una vez realizados todos los ajustes mecánicos, la visión artificial permite al sistema detectar la mayoría de los objetos para decidir como reaccionar y actuar frente a ellos. Este proceso esta dotado de una inmensa complejidad, debido a la infinidad de situaciones u objetos a los que el sistema se puede enfrentar.

1.1 Visión artificial

La visión artificial engloba a cualquier proceso óptico mediante el cual un sistema inteligente es capaz de extraer información de un entorno para su interpretación mediante el uso de la computadora. En un principio los sistemas de visión artificial estaban estrechamente basados en la visión humana, pero debido a la falta de entendimiento de los procesos que tienen lugar en el cerebro a la hora de interpretar los datos del sistema visual, estos sistemas artificiales resultaron básicamente imprácticos. El cerebro

humano, en combinación con el sistema visual del que dispone nuestro cuerpo, permite un reconocimiento y una interpretación de su entorno mucho mas flexible y adaptable a cambios que cualquier sistema de visión artificial. Sin embargo, los sistemas de visión artificial son capaces de procesar cantidades de datos mucho mayores y en menos tiempo si se trata de tareas repetitivas. La infalibilidad de la precisión matemática, permite un estudio y análisis mucho mas detallado y extenso por parte de un sistema basado en la visión artificial. Esta característica permite la creación de sistemas muy diferentes en cuanto a componentes y capacidades, según la tarea a implementar.

El comienzo de cualquier proceso de reconocimiento, es la cuantificación digital del entorno. El sistema ha de ser capaz de procesar la información obtenida y aplicar los algoritmos adecuados para su procesamiento y correcta interpretación. Ésta resultaría imposible sin un especificado pre-procesado de la imagen, debido a que los patrones a reconocer pasan a ser infinitos. El sistema ha de ser capaz de identificar al objeto independientemente de su iluminación, color o posición dentro de la imagen, y por lo tanto ha de conocer a todas las infinitas posibles variaciones que el objeto puede presentar.

La aplicación de modelos jerárquicos que imita el proceso visual del cerebro humano, ayuda a los sistemas inteligentes a crear sus bases de datos. En este método, primero se buscan los rasgos mas pequeños del objeto para después buscar las posibles interconexiones que puede haber entre estos pequeños elementos. Cada pieza es almacenada una única vez en memoria y repitiendo estructuras y uniones se llega a formar finalmente el objeto. Puede haber objetos diferentes que compartan ciertas estructuras y rasgos, y solo haría falta recurrir al catálogo de piezas que se ha generado para reconstruir estos objetos e identificarlos. De esta manera se pretende reducir el impacto que tiene una base de datos completa en la memoria de un sistema de visión inteligente.

Muchas veces no toda la información necesaria para completar una tarea esta contenida en una imagen. El ser humano es capaz de extraer más información de una imagen que la que realmente esta disponible. Esto ocurre gracias al aprendizaje de patrones, referencias e información de contexto que un sistema artificial no puede conocer. Por lo tanto, para crear un sistema inteligente robusto, es necesario que tenga la capacidad de aprender.

El aprendizaje supervisado no es más que el proceso mediante el cual se entrena un sistema con un conjunto de datos discretos para conseguir la mejor aproximación de una función multivariable. Esta aproximación se ha realizado mediante multitud de métodos, tanto estadísticos como

probabilísticos. Las redes neuronales y modelos Bayesianos si cumplen haciendo estas aproximaciones, pero con la limitación de que el conjunto de datos discretos sea limitado.

Otro gran campo de aplicación de sistemas de visión artificial, es la mejora y restauración de imágenes mediante una multitud de técnicas que estudia el tratamiento digital de imágenes. Veremos algunas de las técnicas mas utilizadas en apartados posteriores.

1.2 El Reconocimiento de objetos

Un procesado de la imagen captada por un sistema de visión artificial, es fundamental para un reconocimiento correcto y eficaz. Al procesar la imagen se elimina la mayor parte de información irrelevante para reconocer el objeto, y se extraen únicamente aquellos parámetros que permiten una identificación e interpretación inconfundible del objeto. La selección de estos parámetros es una de las tareas mas criticas en el reconocimiento. El tipo de procesamiento empleado también depende del propósito del sistema y del tipo de objetos que se deseen extraer. El preparamiento para la interpretación no es el mismo para un sistema cuya finalidad es detectar astros en el espacio, que para uno empleado en la detección de matriculas de coches. En cada caso hay que estudiar el orden y la tipología de los filtros empleados, para obtener la mejor interpretación por el sistema posible.

El propio proceso de reconocimiento es la interpretación de todos los parámetros seleccionados en el procesado. La interpretación, ocurre mediante la comparación de los modelos extraídos de la actual imagen con aquellos obtenidos mediante aprendizaje supervisado o cualquier otro proceso. El estudio de todas las posibilidades de identificar el objeto sin ambigüedades mediante la comparación, puede ser un proceso costoso, teniendo en cuenta que la imagen solo es una representación en dos dimensiones de un objeto real dotado de tres. Cambios en la perspectiva que se producen en movimientos en direcciones no paralelas al sistema de visión, presentan una gran problemática. Es imprescindible en estos casos, que el sistema disponga de una representación 3D del objeto en su base de datos, o por lo menos un conjunto de diferentes vistas discretas que permitan una identificación. Este método, conocido como RBV (reconocimiento basado en vista), también esta inspirado en la manera de interpretar del cerebro humano.

2. El Tratamiento digital de imágenes

El gran avance tecnológico en el campo de la adquisición digital de la imagen, y su profunda y cada vez mas estrecha relación con ordenadores, nos permiten un estudio cada vez mas detallado y especificado de las características de la imagen digital. Hoy en día se encuentra en casi cualquier hogar una cámara, un escáner o un ordenador. Paradójicamente, aunque los equipos estén cada vez mas sofisticados y complejos, su manejo a nivel de usuario se ha adaptado cada vez mas a una ejecución casi automática y de lo más simple. Esto es debido a que un entendimiento y un estudio de la tecnología de la imagen digital, solo es accesible mediante un amplio conocimiento de fundamentos básicos.

El tratamiento digital de imágenes estudia el conjunto de procesos y técnicas que nos permiten manipular la imagen de tal manera que el resultado nos permita descubrir o resaltar cierta información contenida en ella. Como herramienta básica para llevar a cabo todos estos procesos se necesita de un ordenador capaz de maniobrar con imágenes de dos o más dimensiones.

En este apartado introduciremos algunos conceptos básicos y estudiaremos algunas técnicas del tratamiento digital de imagen fundamentales para comprender la aplicación implementada en este proyecto.

2.1 Conceptos básicos de la imagen digital

Para entender correctamente el funcionamiento y el manejo de una aplicación de imagen digital, se precisa de unos conocimientos previos de algunos conceptos que aparecerán frecuentemente tratados en este proyecto.

2.1.1 Tipos de imagen digital

La clasificación de las imágenes se puede estudiar de numerosas formas y múltiples criterios. Por ahora solo vamos a centrarnos en la manera en las que están descritas en el ordenador. La adquisición de una imagen digital mediante un sistema compuesto por sensores ópticos y digitalizadores, es el primer paso en cualquier procesado digital. Las imágenes digitales se pueden clasificar en dos grandes grupos: Imágenes vectoriales y imágenes basadas en mapas de bits.

Una imagen vectorial se compone de contornos y rellenos descritos por fórmulas matemáticas. Estas precisas ecuaciones vectoriales describen la imagen perfectamente y son escalables sin perdida alguna de su calidad. Cuanto mas compleja es una imagen, mas complejo son las ecuaciones que la definen, lo que conlleva a un mayor tiempo de computo y mayor peso en memoria. Por esta razón, tienen su aplicación principal en la representación de caracteres, gráficas y estructuras geométricas como logotipos, ya que todos ellos tienen líneas y formas claramente diferenciables y clasificables.

Las imágenes de mapas de bits, están descritos mediante una concatenación de pequeños cuadraditos de igual tamaño, llamados pixel, que recorre toda la imagen. En cada pixel se guarda la información de color que presenta la imagen en este punto, dando resultado a una malla que ocupa toda la imagen. Las regiones curvas de un mapa de bits son estructuras dentadas debido a esta topología.

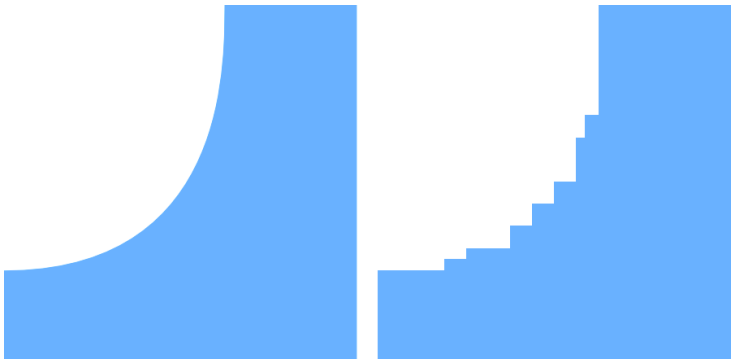


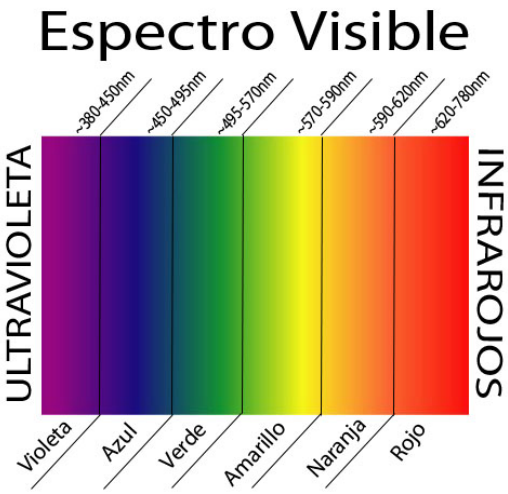
Fig.3 Comparación de una región curva descrita mediante una imagen vectorial con una descrita por un mapa de bits

Las imágenes de mapas de bits están orientados a imágenes que presentan gran diversidad en su gama de color y tonalidad, pero pierden mucha nitidez a la hora de realizar escalados sobre la imagen. Configurando diversos parámetros como la cantidad de pixeles, el tamaño de cada uno y el rango de colores que pueden tomar, se puede conseguir una representación más o menos fiel de la imagen real.

Debido a que la aplicación implementada en este proyecto, pretende estudiar un reconocimiento de objetos tomadas mediante *WebCam* en un entorno con grandes saltos de color y el escalado digital no es importante, utilizaremos imágenes en mapas de bits.

2.1.2 El color de un objeto

Para poder percibir el color de un cuerpo, se necesita luz. El color es una interpretación individual y subjetiva de las ondas electromagnéticas que refleja el cuerpo iluminado. De la multitud de longitudes de onda dentro del espectro electromagnético de la luz solar, el ojo humano solo es capaz de detectar aquellas comprendidas entre 380 y 780nm, zona llamada espectro visible. Dentro de este espectro visible se pueden clasificar los 6 colores espectrales como franjas de color, incluyendo en sus limites todas las demás tonalidades.



La percepción de distintas longitudes de onda por los conos y bastones en la retina humana, no es la única base para nuestro sistema óptico. El cerebro y sobretodo la memoria, tienen un papel importante en la identificación del color. Algunos objetos con un color típico, son percibidos con su color almacenado en la memoria. El color rojo de un tomate nos parece mucho más intenso de lo que en realidad es, y esto es debido a que en nuestra memoria existe un prototipo de color rojo que se interpola con nuestra percepción.

También la interpretación y clasificación personal de todos los colores visibles, depende de factores sociales, demográficos y lingüísticos. Así por ejemplo, en el griego antiguo no se diferenciaba entre el color de la miel y entre el del césped, debido a que la hierba era marrón la mayor parte del año.

El estudio científico evade esta problemática, estableciendo para cada color un único punto de coordenadas dentro de cada espacio de color. El color en estos sistemas es cuantitativo, es decir, una combinación de varios números que permiten un análisis objetivo y estandarizado. A continuación estudiaremos el espacio de color más conocido y utilizado, y algunos modelos alternativos y complementarios.

2.1.2.1 El espacio RGB y su almacenamiento

Uno de los espacios de color más utilizado es el RGB. Es un sistema basado en mapas de bits y la capacidad de los conos en el ojo humano de

distinguir claramente entre rojo, verde y azul. Es un sistema de color aditivo, es decir la suma de todos los colores genera el blanco.

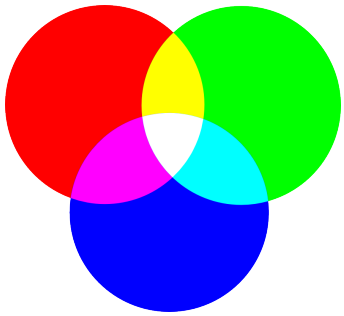


Fig.4 Generación de colores en el espacio RGB

Una imagen en color RGB basa en su funcionamiento en la matriz bidimensional de una imagen en blanco y negro. Esta matriz esta compuesta por los valores de luminancia de cada pixel que compone la imagen. Trabajando con 8 bits se tiene un total de 256 valores posibles para cada pixel, que se traducen en diferentes tonalidades de grises. De esta manera se cubre toda la escala de grises, desde el valor 0, correspondiente al negro, hasta el valor máximo 255, que corresponde al blanco.

En el espacio RGB partimos de la misma idea, solo que ahora tenemos una matriz tridimensional compuesta por 3 planos correspondientes a los tres colores primarios: rojo, verde y azul. Cada plano de color es como una imagen en blanco y negro, únicamente que el valor máximo 255 ahora es el color primario puro y por lo tanto todos los valores entre 0 y 255 son tonalidades de este color. La combinación de los tres planos de colores primarios, permite la representación de todos los demás colores. De esta manera por ejemplo, la suma del rojo puro (255 0 0) con el verde puro (0 255 0), genera el amarillo puro (255 255 0).

La ausencia de color, es decir un valor de 0 en todos los planos de color, genera el negro puro, mientras que la suma de los tres colores primarios puros, es decir un valor de 255 en todos los planos de color, genera el blanco puro. La escala de grises en este nuevo espacio tridimensional es la recta diagonal de la matriz que une el vértice del color blanco puro (0 0 0) con el vértice del negro puro (255 255 255). A lo largo de esta recta, los tres planos de color coinciden en su valor y cada punto de la recta es equidistante hacia los ejes de cada dimensión.

Cada uno de los tres planos de color trabaja a su vez con 8 bits, es decir que por plano de color seguimos teniendo 256 diferentes valores, pero debido a la tridimensionalidad de una imagen en color, obtenemos un total de 16777216 posibles variables.

2.1.2.2 El espacio CMYK

Opuestamente a la idea aditiva del modelo RGB para crear los colores, esta el espacio CMY. En este espacio, la suma de todos los colores da como resultado el negro, y la ausencia de color genera el blanco.

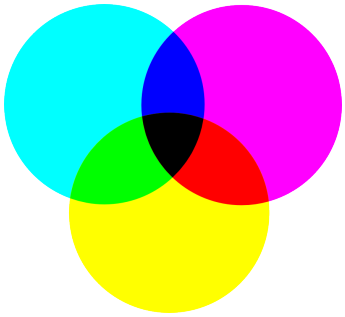


Fig.5 Generación de colores en el espacio CMY

El espacio CMY esta compuesto por los colores complementarios a los tres primarios, es decir: cian, magenta y amarillo. Es el espacio de color utilizado en imprenta, aunque en este ámbito, el negro no es generado como la suma de todos los colores. En la práctica , el negro así producido contiene impurezas y tiende más a clasificarse como un marrón oscuro. Por esta razón se añade un cartucho especial de color negro, originando la familiar denominación CMYK.

2.1.2.3 Modelos circulares de color: HSV y HSL

Es más cercano a la idea humana del color , pensar sobre términos de matiz y saturación a la hora de definir un color que hacerlo por sus propiedades sustractivas y aditivas. De esta idea nacieron los modelos HSV (*Hue , Saturation, Value*) y HSL (*Hue, Saturation, Lightness*) como una transformación no lineal del espacio RGB.

En el modelo HSV, la matiz se representa en una región circular, cuyo centro es el blanco. La distancia del centro hacia el color en la región circular define la saturación. Cada color primario esta separado del otro 120˚ y los colores mixtos están definidos entre los espacios resultantes. Una tercera dimensión es añadida para definir el negro, generando así un cono en el espacio, cuya altura viene dada por el brillo o valor.

El modelo HSL sigue esta idea, solo que ahora su representación en el espacio es un doble cono, en cuyos vértices opuestos se localizan el negro y el blanco. El centro del doble cono seria por lo tanto un tono gris, dado que el eje que une sus vértices es la luminancia.

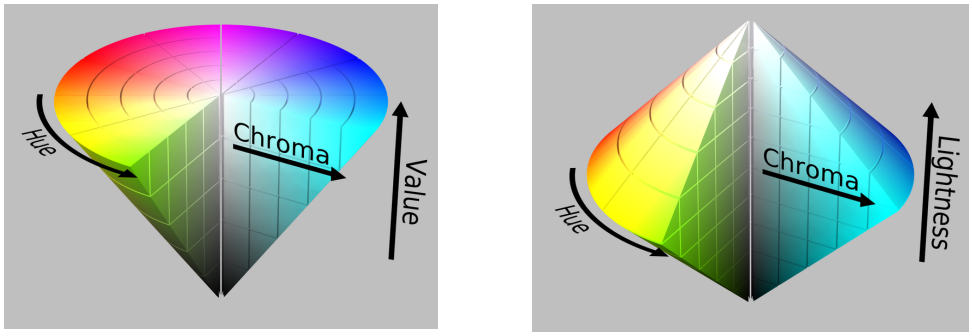


Fig.6 Espacios de color HSV y HSL

2.1.2.4 Espacio de color L*a*b

El espacio de color L*a*b esta basado en la percepción de color del ojo humano. Este modelo de referencia de color fue diseñado para ser independiente del dispositivo en el cual se emplea, ya sea por pantalla o impresora. Crea colores inalterables, persistentes y mas resistentes frente a errores separando la luminosidad "L" de los dos canales cromáticos "a" y "b". En el espacio L*a*b es posible referenciar mas colores que en CMYK y RGB, incluso colores imaginarios que no tienen presencia en la naturaleza.

2.2 Operaciones básicas del tratamiento digital

Si una imagen digital es manipulada mediante operaciones del tratamiento digital, se pueden obtener dos resultados distintos: o bien el proceso ha creado una nueva imagen conteniendo información extraída de la original, o de la otra manera, el resultado es meramente información numérica con datos de interés.

En el desarrollo de esta aplicación, van a influir ambos tipos de tratamiento digital. Por un lado habrá que modificar la imagen para generar nuevas (pre-procesado) y por el otro lado extraeremos de estas nuevas imágenes información numérica para su interpretación.

A continuación veremos algunas de las operaciones básicas en las que se basa este proyecto.

2.2.1 Pre-procesado de la imagen

El primer paso tras la adquisición de la imagen es pre-procesarla. La etapa del pre-procesado es la aplicación de técnicas y herramientas para realizar

una transformación de la imagen con el fin de mejorarla para procesos posteriores. Estos procesos abarcan desde el recorte de la imagen en zonas de interés, hasta técnicas de suavizados para reducir ruidos.

El primer paso en el pre-procesamiento es asegurarnos de que la imagen contiene a las regiones de interés de la mejor manera posible. Para ello podemos aplicar técnicas de recorte, giro y escaldo, o cualquier otra técnica que nos permita delimitar físicamente la zona a estudiar.

La aplicación de filtros de eliminación de ruido también podría describirse como técnica de pre-procesamiento. El ruido se define como cualquier elemento en las imágenes que no son interesantes para la computación que se pretende llevar a cabo. Existen numerosos filtros que aplican transformaciones pixel a pixel, que no solo dependen de la tonalidad del pixel a tratar, sino también de las de sus vecinos. La aplicación de estos filtros ocurre mediante máscaras, matrices cuadradas con un numero de filas impar. Algunos de los filtros más utilizados son: filtros gaussianos, filtros de mediana, filtros de suavizados (paso bajo y paso alto) y filtros SUSAN (*Smallest Univale Segment Assimilating Nucleus*).

También procesos que modifican el contraste pueden aplicarse en el pre-procesamiento. La ampliación del contraste es una técnica que modifica la función de transferencia de tal manera que su pendiente sea distinta de uno. De normal se aumenta la pendiente en zonas con pixeles de tonalidad intermedia entre oscuro y brillante, y se baja en aquellas zonas con tonalidades cerca de negro y blanco.

La binarización de la imagen también es una técnica de ampliación de contraste. Es un caso extremo, en que la función de transferencia para los tonos oscuros y claros es cero y para las tonalidades intermedias es $\pi/2$. La imagen resultado se compone únicamente de tonalidades blanco y negro, con lo cual se puede simplificar enormemente tares de segmentación o detección de bordes.

La función de transferencia puede ser modificada de numerosas formas tales como el clipping , slice, logarítmicos, operaciones aritméticas y lógicas, todos ellos con resultados muy distintos sobre la imagen.

2.2.2 Segmentación de la imagen

En cualquier proceso de reconocimiento de imágenes, el paso fundamental es diferenciar claramente todos los elementos que componen la imagen. Esto es segmentar los diferentes objetos del fondo. Para ello se pueden aplicar multitud de procesos y técnicas distintas, dependiendo del

tipo de imagen y el resultado deseado. Normalmente la segmentación de una imagen monocromática se basa en las características de las tonalidades de gris, tales como la discontinuidad y la similitud. La discontinuidad busca líneas, bordes o puntos en función a cambios abruptos en la tonalidad de gris, mientras que la similitud establece regiones basándose en las relaciones espaciales o cromáticas que puede haber entre los pixeles que la forman.

2.2.2.1 El Histograma: Thresholding

El histograma es una representación gráfica de la cantidad de pixeles que hay de cada tonalidad de color de la imagen. En una imagen en tonos de gris se obtiene un diagrama de barras, donde la superficie de cada barra es proporcional a la frecuencia de aparición de cada tonalidad de gris. En una imagen de ocho bits obtenemos un total de 256 distintos valores, donde la primera posición representa al negro y la ultima al blanco.

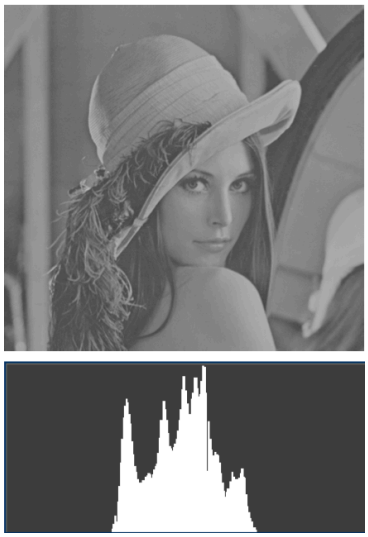


Fig.7 Histograma de una imagen en blanco y negro

El histograma de color, muchas veces es representado como la unión de los histogramas de cada componente de color de la imagen RGB. Esta representación no es del todo acertada, ya que no se tiene en cuenta la tonalidad de color en concreto, sino cada uno de sus tres componentes. El verdadero histograma de color es una representación tridimensional de todas las tonalidades de color que aparecen en la imagen. El histograma por lo tanto ahora esta compuesto por cubitos que dan información acerca de la frecuencia de aparición de cada tonalidad.

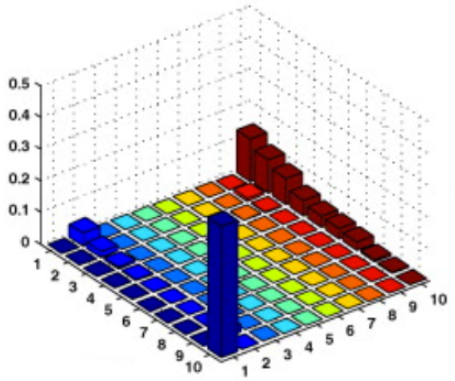


Fig.8 Representación 3D de un histograma de color

Dado un histograma perteneciente a una imagen cuyo color de fondo es claramente distinto a la tonalidad del objeto, podemos identificar fondo y objeto en dos regiones dominantes. Para diferenciar fondo y objeto resulta obvio seleccionar una frontera que divida ambas regiones mediante un umbral. Todo valor superior o inferior a cierto umbral pertenecerá o bien al objeto, o bien al fondo. Sea g(x, y) la imagen monocromática y f((x, y) el histograma, podemos separar el fondo de los objetos aplicando un umbral de decisión T.

$$g(x,y) = \begin{cases} 1 \rightarrow si\ f(x,y) > T \\ 0 \rightarrow si\ f(x,y) \leq T \end{cases}$$

No hay convención estándar sobre que umbral separa mejor las diferentes tonalidades, ya que en cada imagen, el conjunto de tonalidades es muy distinto. Por ello hay que realizar un estudio independiente del histograma en cada nueva imagen para fijar el umbral optimo. Para diferenciar varios objetos en una misma imagen, basta con fijar varios umbrales entre los distintos lóbulos que puede presentar el histograma. En la práctica, la segmentación por umbralización multinivel del histograma, es una técnica que tiende a fallar. Esto estriba de la dificultad que se presenta en aislar múltiples regiones debido a los pixeles situados en el área de transición entre objetos. Una posible solución es la eliminación de los bordes antes de calcular el histograma.

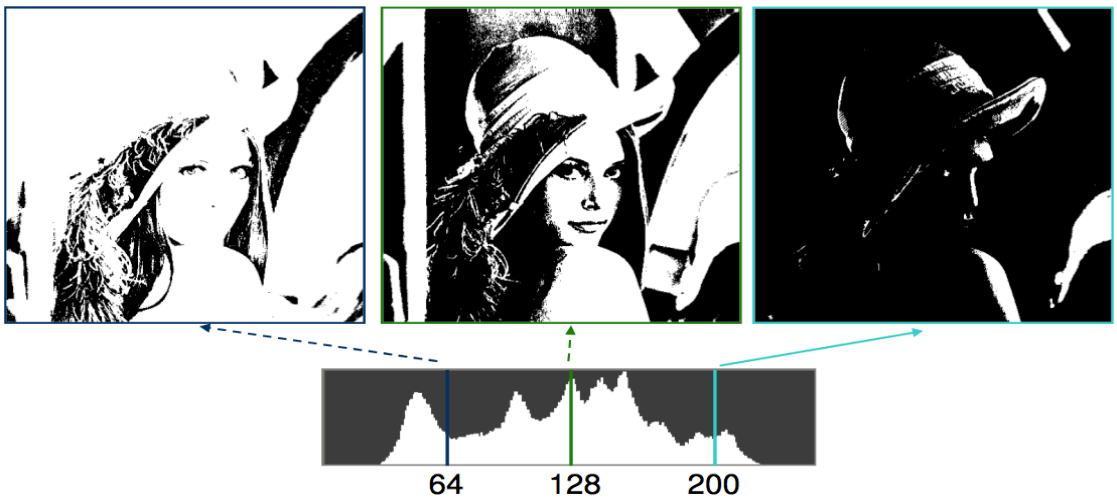


Fig.9 El resultado de fijar tres umbrales distintos, son tres imágenes con tonalidades también distintas.

2.2.2.2 Segmentación mediante detección de bordes

Los bordes de una imagen digital se pueden definir como transiciones entre dos niveles de gris significativamente distintos. Para ello se emplean diferentes aproximaciones discretas de la primera (la gradiente) y la segunda (laplaciana) derivada de los niveles de gris. Cuanto mayor sean las variaciones existentes entre dos regiones fronterizas en la imagen, mayor es el valor de la primera derivada. El vector gradiente apunta a la dirección donde la variación es máxima y su módulo es proporcional con el valor de esta variación. La derivada parcial del eje "x" e "y" e puede aproximar por la diferencia de pixeles adyacentes de la misma fila o columna, respectivamente.

$$\frac{\partial f(x,y)}{\partial x} \approx \nabla_x f(x,y) = f(x,y) - f(x-1,y)$$
$$\frac{\partial f(x,y)}{\partial y} \approx \nabla_y f(x,y) = f(x,y) - f(x,y-1)$$

Las gradientes de cada fila o columna se obtienen a partir de una convolución con una máscara predefinida. Estas máscaras suelen estar implementadas mediante matrices de igual número impar de filas que de columnas, de tal manera que exista un pixel central. Una máscara puede estar rellena de cualquier conjunto de números reales, según la finalidad de la aplicación, cumpliendo con que la suma total de los valores que componen la máscara sea nula. Existen multitud de máscaras predefinidas, entre las cuales destacamos los operadores de *Roberts*, *Prewitt*, *Sobel* y *Frei-Chen*.

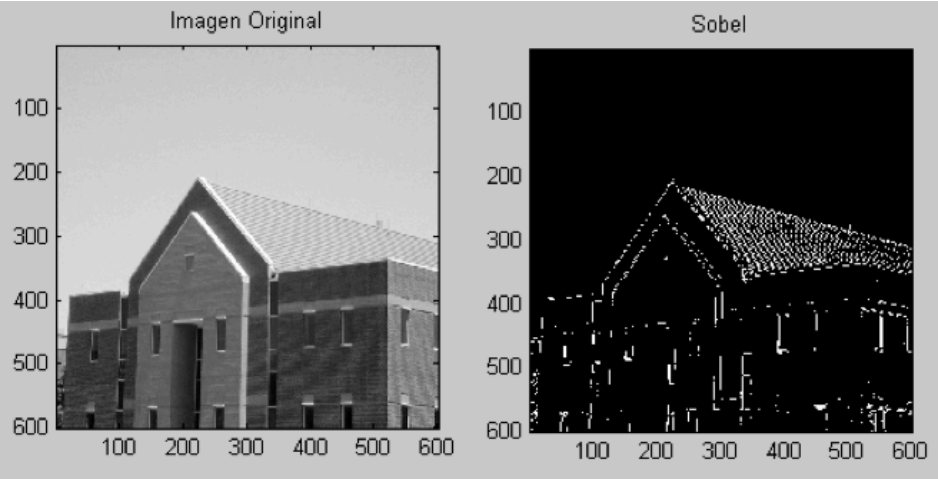


Fig.10 Imagen resultante de aplicar una máscara de *Sobel*, tanto en sentido horizontal como vertical.

La segunda derivada, definida como *Laplaciana*, es pocas veces utilizado como detector de bordes. Al ser de segundo orden es muy sensible frente a ruido por lo que usualmente juega un papel secundario en la detección de bordes.

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

La función *Laplaciana* responde a cambios en la intensidad, y por lo tanto puede dar información acerca de si un pixel esta situado en un lado oscuro o brillante de un borde.

2.2.2.3 Segmentación basada en la conectividad

Otra técnica de segmentación por similitud, es aquella basada en la conectividad espacial que pueden presentar los pixeles en una imagen. Dos pixeles son vecinos si tienen en común por lo menos una frontera o una esquina, según el tipo de vecindad. Se les llama vecinos directos a aquellos que comparten sus fronteras, y vecinos indirectos si comparten esquina. Un conjunto de pixeles cumple el criterio de vecindad si cumplen entre ellos con un criterio de similitud establecido. Recorriendo la imagen pixel por pixel, de izquierda a derecha y de arriba hacia abajo, se etiqueta cada conjunto de pixeles vecinos con un identificador distinto.

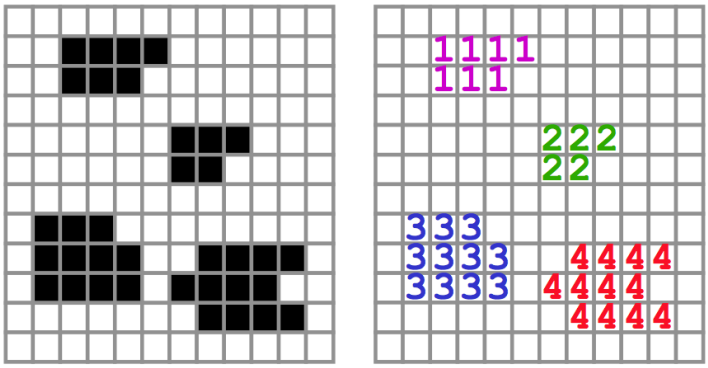


Fig.11 Ejemplo de una segmentación basada en la conectividad

Este tipo de conectividad puede resultar de distintas maneras: cuatro conectividad, ocho conectividad e incluso seis conectividad. Un pixel x_0 con coordenadas (x, y) tiene dos vecinos horizontales (x_1, x_3) y dos verticales (x_2, x_4) y sus coordenadas están dadas por:

$$(x+1, y), (x-1, y), (x, y+1), (x, y-1)$$

Este conjunto de pixeles se conoce como cuatro conectividad con el pixel central x_0 en (x, y). De la misma manera se dejan definir el conjunto de las diagonales como:

$$(x+1, y+1), (x-1, y+1), (x-1, y-1), (x+1, y-1)$$

El conjunto compuesto por los pixeles que forman la cuatro conectividad y las cuatro diagonales se le conoce como la ocho conectividad respecto al pixel central "p" en (x, y). Cualquier variación de estos conjuntos de pixeles que se tienen en cuenta para formar la vecindad es admisible como criterio de segmentación.

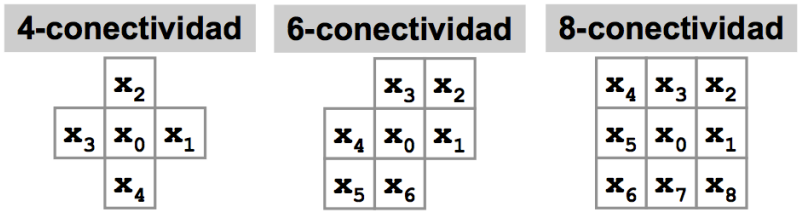


Fig.12 Diferentes combinaciones de vecindad entorno a x_0

2.2.3 Procesamiento morfológico

En la mayoría de los casos la segmentación no da resultados exactos acerca de las contornos de los objetos o regiones segmentadas. Suelen

aparecer zonas con pixeles mal clasificados que generan bordes imprecisos y discontinuos. Para mejorar los desperfectos en estas zonas criticas es preciso realizar un post-procesado de la imagen que realce la forma y la geometría de los objetos.

Para extraer de los objetos sus estructuras geométricas se precisa de otro conjunto de forma y tamaño conocido, denominado elemento estructurante. Tanto el tamaño como la forma de los elementos estructurantes son seleccionados atendiendo a la finalidad de su aplicación y las formas que se desean extraer.

Fig.13 Algunos elementos estructurantes típicos

2.2.3.1 Dilatación y erosión

La transformación morfológica conocida como erosión es el resultado de comprobar si el elemento estructurante esta completamente contenido en un conjunto de pixeles. Si algún pixel por vecindad no cumple con la limitación impuesta por el elemento estructurante, se pone a cero el pixel central. La consecuencia es una disminución del área del objeto sobre el cual se ha aplicado, así como la desaparición de todos los elementos más pequeños que el elemento estructurante. La aplicación repetitiva de la transformación de erosión conlleva a la desaparición total de todos los objetos de la imagen.

La transformación dual a la erosión es la dilatación. La dilatación es el resultado de comprobar que el elemento estructurante este al menos contenido en un pixel de un conjunto de pixeles. Si se cumple lo anterior, se pone a uno el pixel central. Con ello se consigue una expansión del objeto en su región fronteriza y un cierre de todas las discontinuidades menores que el elemento estructurante. La aplicación repetitiva de la dilatación conlleva a que los objetos lleguen a expandirse tanto que ocupan toda la imagen.

La dilatación y la erosión no son procesos inversos. El orden de aplicación altera el resultado final.

Fig.14 Ejemplo de proceso de dilatación y posterior erosión utilizando una cruz como elemento estructurante.

2.2.3.2 Apertura y cierre

Combinando erosión y dilatación en ordenes distintos creamos herramientas de realce conocidas como apertura y cierre. El erosionado permite eliminar pequeños objetos, pero también reducimos el tamaño de todos los demás objetos en la imagen. Al aplicar una dilatación, volvemos a expandir los objetos a su tamaño inicial menos aquellos pixeles previamente eliminados por la erosión. Este proceso recibe el nombre de apertura. Esta operación no preserva estrictamente la geometría de los objetos, ya que erosionando y dilatando se suaviza los bordes.

La combinación inversa, es decir realizar primero una dilatación y luego una erosión recibe el nombre de cierre. La dilatación permite cerrar agujeros y grietas, y la expansión de los pixeles es parcialmente invertida tras aplicar una erosión. De nuevo, se altera la geometría de los objetos y se suaviza sus bordes, pero se consigue rellenar las fisuras.

3. El funcionamiento principal del programa

3.1 Pre-procesado de la imagen

El programa es capaz de diferenciar objetos enteros sobre fondos no uniformes. Para poder reconocer correctamente los objetos que se colocan sobre el fondo, es preciso realizar un pre-procesado de la imagen. El proceso de la clasificación del fondo y la extracción de la información de los objetos, se explicarán a continuación.

El fondo, llamado *Background*, esta formada por una imagen temporalmente invariante, es decir su composición a nivel de pixel no cambiará durante todo el proceso del reconocimiento. El sistema identificará esta imagen al principio de su ejecución como la media de la sucesión de las tres primeras imágenes que el sistema lee, y la almacenará en una variable.

Una vez identificada la imagen fondo, puede empezar el proceso de reconocimiento. Se colocan diversos objetos desconocidos sobre el fondo conocido creando una nueva imagen. La diferencia entre el fondo y esta nueva imagen, esta compuesta únicamente por los objetos y un fondo uniforme y nulo. Esta imagen la llamaremos *foreground*. (fig.)

Fig.15 La diferencia entre *Background* y la imagen compuesta por el mismo fondo con objetos, es el *Foreground*.

Debido a la tridimensionalidad de una imagen en color, el proceso de comparación y extracción de la diferencia, es mas costoso que en una imagen monocromática. Para ahorrar tiempo de ejecución y optimizar el código para una respuesta en tiempo real, es conveniente convertir tanto la imagen fondo como la imagen con los objetos a escala de grises. La diferencia de ambas imágenes es un *foreground* en escala de grises. Multiplicando elemento a elemento el *foreground* monocromático por la

imagen fondo con los objetos en color, obtenemos el *foreground* con toda la información cromática. (fig.) Esta operación se realiza con cada nueva imagen que le enviemos al programa.

Pequeñas irregularidades debidas a cambios de iluminación o por la propia compresión, pueden ser eliminados aplicando un filtrado de mediana al *foreground* monocromático. Este filtrado elimina ruido conocido como "sal y pimienta" y suele ser mas efectivo que una convolución ya que preserva mejor los bordes.

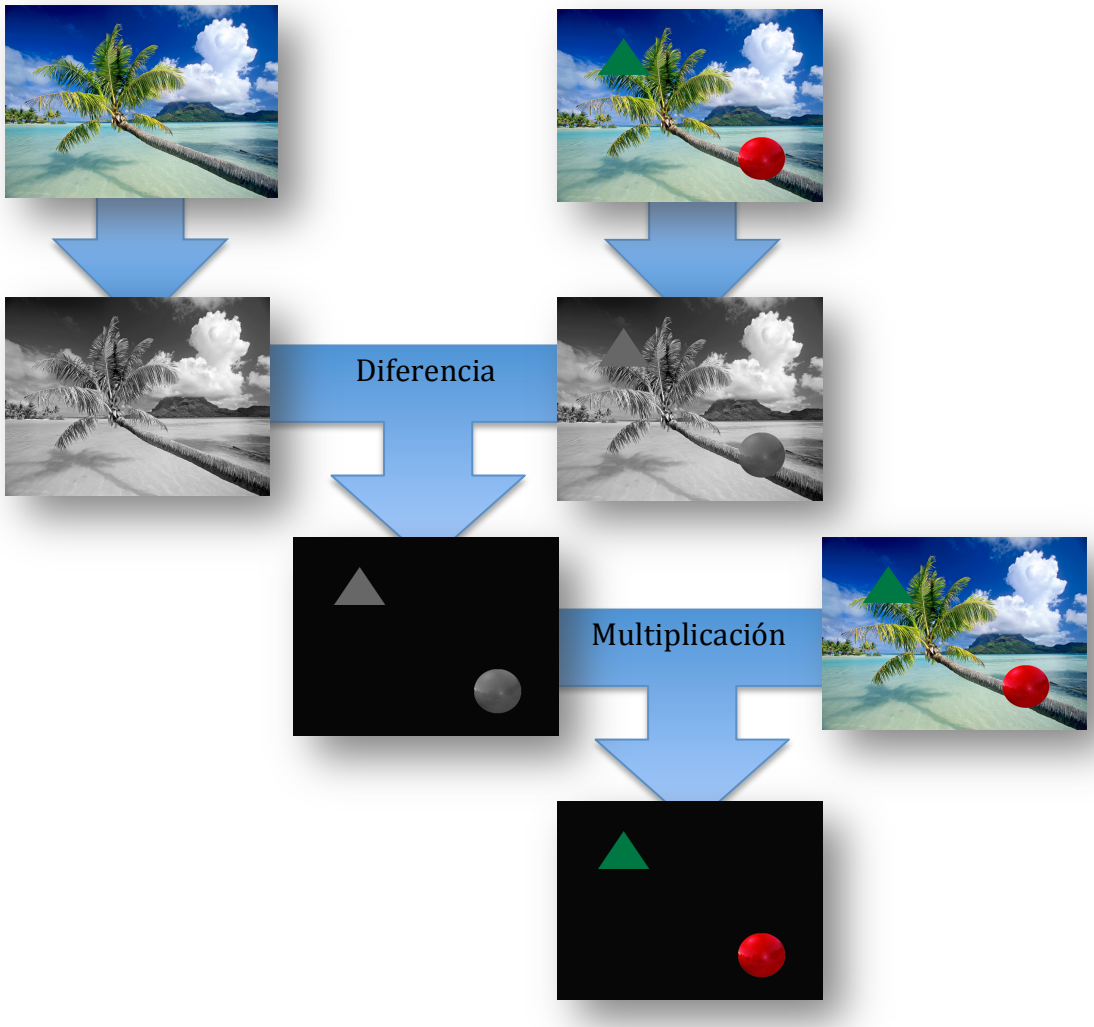


Fig.16 Proceso completo para la extracción de los objetos

3.2 Extracción de características

El siguiente paso en el reconocimiento, es la extracción de características de los objetos contenidos en el *foreground*. La selección de las características es clave en la identificación exacta de los objetos. Este programa extrae de cada objeto los siguientes parámetros:

- Identificador de objeto
- Área y perímetro
- Rectángulo mas pequeño que puede contener el objeto
- Información de color RGB
- Distancia de movimiento

Todos estos elementos son almacenados en este orden en un único vector por objeto. El conjunto de todos los vectores es guardado en una matriz con igual número de filas que objetos encontrados. En el programa, esta Matriz es denominada "S" y únicamente almacena la información de los objetos de la imagen actual.

La función encargada de rellenar la matriz "S" se llama *objectstat.m*. Dicha función tiene como parámetros de entrada el *foreground* monocromático y el *foreground* con la información de color. La Matriz "S" tendrá tantas filas como objetos encuentre la función *bwlabel.m* en el *foreground* monocromático. La función *bwlabel.m* (incluida en la librería de *MatLab*), genera una matriz asignándole valores no nulos enteros a cada objeto y ceros a pixeles pertenecientes al fondo.

3.2.1 El identificador de objeto

El primer elemento en el vector que caracteriza al objeto, es su identificador. El número identifica al objeto según el orden de aparición en sentido ascendente. El primer objeto encontrado llevará el identificador uno, el segundo objeto distinto el dos y así sucesivamente. El identificador es modificado posteriormente por la función *matrixfind.m* para dar solución a los siguientes problemas:

El número asignado al objeto debe de ser invariante respecto al movimiento que puede haber entre imágenes. Aunque el objeto se mueva, el programa tiene que identificarlo con el mismo número. Debido a que MATLAB recorre las imágenes en sentido vertical, es decir fila por fila, en la mayoría de los casos no se identifican correctamente los objetos cuando estos se mueven entre imágenes. Además el programa ha de re-identificar correctamente los objetos aunque estos hayan desaparecido temporalmente.

De la misma manera, si un mismo objeto aparece repetido varias veces en la misma imagen, el programa ha de ser capaz de identificarlo con el mismo número. Por defecto MATLAB asigna un nuevo número a cualquier objeto nuevo encontrado. Para solucionar los casos anteriores, es preciso reajustar el valor del identificador de objeto en la función *matrixfind.m*.

3.2.2 La relación entre área y perímetro: Compilancia

Mediante la función *regionprops.m* que incluye la librería de *MatLab*, podemos averiguar fácilmente el área y el perímetro del objeto encontrado en la imagen. Para darle al programa mayor robustez frente a alejamientos y acercamientos en planos paralelos a los de la imagen, se calcula la relación entre el área y el perímetro.

$$C_i = 4\pi * A_i / P_i^2$$

Cuanto mas cera este el resultado de uno, más cercano al circulo será la forma del objeto. Este valor es el segundo elemento almacenado en el vector que caracteriza al objeto.

3.2.3 El rectángulo mas pequeño que contiene el objeto

De nuevo recurrimos a la función *regionprops.m* de la librería de MATLAB, pero esta vez preguntando por el llamado "*Bounding Box*", el rectángulo más pequeño que puede contener todo el objeto. El valor devuelto es un vector de vectores, siendo el primer vector la posición del pixel superior izquierdo del rectángulo y el segundo vector la longitud total del rectángulo en cada dimensión. El rectángulo obtenido no es utilizado directamente para la clasificación e identificación de objetos. El programa recurre al rectángulo para calcular el color medio y para posicionar el texto identificativo de cada objeto.

3.2.4 Información de color RGB

Un dato relativamente discriminativo y decisivo en el reconocimiento, es el color que presentan los objetos. La problemática surge a la hora de decidir que color es el que identifica nuestro objeto. Debido a la gran divergencia que pueden presentar colores tomados de forma aleatoria de entre todos los pixeles que forman el objeto, el reconocimiento correcto y repetitivo del objeto también presenta una tasa de éxito aleatoria.

Una manera mucho más directa de identificar el color de un objeto es calculando el promedio de todos los pixeles que lo forman, es decir, calcular el color medio (fig.). Este proceso lo realiza la función *mediacolor.m*.

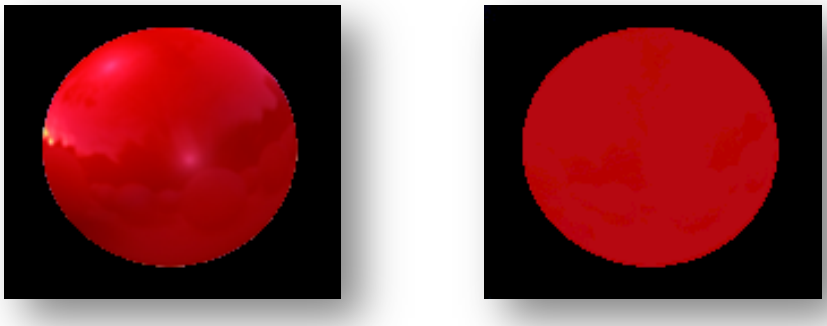


Fig.17 Imagen del objeto antes y después de calcular su color medio.

Esta función recorre toda la matriz tridimensional que forma el objeto, realizando un sumatorio en cada dimensión de todos los valores cromáticos y dividiéndolo entre el total de pixeles no negros. El resultado es un vector que contiene el color medio según sus tres componentes RGB. Sea "S" la superficie del objeto, entonces:

$$\bar{m}_{(S)} = \frac{\sum_{(i,j)\in S} x_{(i,j)}}{|S|}$$

3.2.5 La distancia mínima

Otra característica que nos ayuda a la hora de clasificar es la distancia *Euclídea* que existe entre dos puntos del objeto en dos instantes sucesivos. La distancia *Euclídea* entre dos puntos P_1 y P_2 con coordenadas (x_1, y_1) y (x_2, y_2) respectivamente, se define como:

$$d_E(P_1, P_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Esta distancia es medida a partir del punto central del objeto en cada instante que obtenemos de nuevo llamando a la función *regionprops.m* de la librería de MATLAB. El llamado *Centroid* del objeto, consta de dos valores: una coordenada "x" y otra coordenada "y", que definen el centro en un espacio cartesiano.

Con este dato, un objeto podría ser identificado como el mismo, si la distancia *Euclídea* que une su centro con el centro del mismo objeto en el instante anterior, es la mínima entre todas las distancias posibles.

3.3 El proceso de interpretación

Una vez obtenido la matriz "S" de la imagen actual, que contiene todos los vectores con todos los parámetros anteriormente explicados, se pasa a la fase del reconocimiento de objetos. El programa compara los valores contenidos en cada vector de "S" con todos los vectores contenidos en una matriz maestra, llamada "M". Este proceso tiene lugar en la función *matrixfind.m*.

En la matriz maestra se almacenan todos los objetos encontrados de todos los instantes de tiempo posibles. Cada objeto en "M" es único, y por lo tanto su identificador de objeto también lo es. Los objetos en la matriz maestra se ordenan según su aparición temporal en la imagen, es decir, el primer objeto encontrado llevará el identificador uno, el segundo llevará el dos y así sucesivamente. Aunque la matriz "M" podría cargarse ya iniciada como una base de datos que contiene definidos ya diversos objetos, el programa la inicializa con los valores encontrados en la primera imagen con objetos.

El siguiente paso es simple: el programa compara el objeto actual con todos los objetos que contiene la matriz maestra. Si el objeto se encuentra en la matriz maestra, se actualizan los parámetros en "M" y se corrige el identificador de objeto en "S" si hace falta. Y si en el caso contrario el objeto no se encuentra en la matriz maestra, se añade a "M" el vector de "S" de dicho objeto, actualizando el identificador de objeto en "M".

El proceso de comparación y evaluación de la matriz maestra, esta basado en n sistema de puntos. Por cada característica que tienen en común el objeto de la matriz "S" y el objeto al que se le compara de la matriz "M", el programa asigna puntos. Dado a que tenemos en total siete características (Compilancia, color R, color G, color B, la distancia *Euclídea,* área y perímetro), la puntuación máxima es de siete. El objeto va ser identificado correctamente si alcanza la puntuación máxima.

Con cuatro o menos puntos el objeto es identificado como uno nuevo y es añadido de inmediato a la matriz maestra, actualizándose su identificador de objeto para ocupar el último puesto en la lista de objetos. En el caso de que el objeto coincida con más de cuatro puntos con un objeto de "M", se actualizará el número del identificador de objeto del vector en "S" con el de la matriz maestra. Si además se reconoce el objeto obteniendo más

de cinco puntos, se actualiza los parámetros de color, compilancia, área y perímetro, calculando la media entre lo valores de “S” y de “M”. El valor del centro del objeto también es actualizado con la nueva posición del objeto en la imagen y guardado en la matriz maestra.

Al final del proceso, obtenemos la matriz “S” con todos los objetos ordenados y clasificados, y una matriz maestra actualizada que incluye todos los objetos tanto conocidos como nuevos.

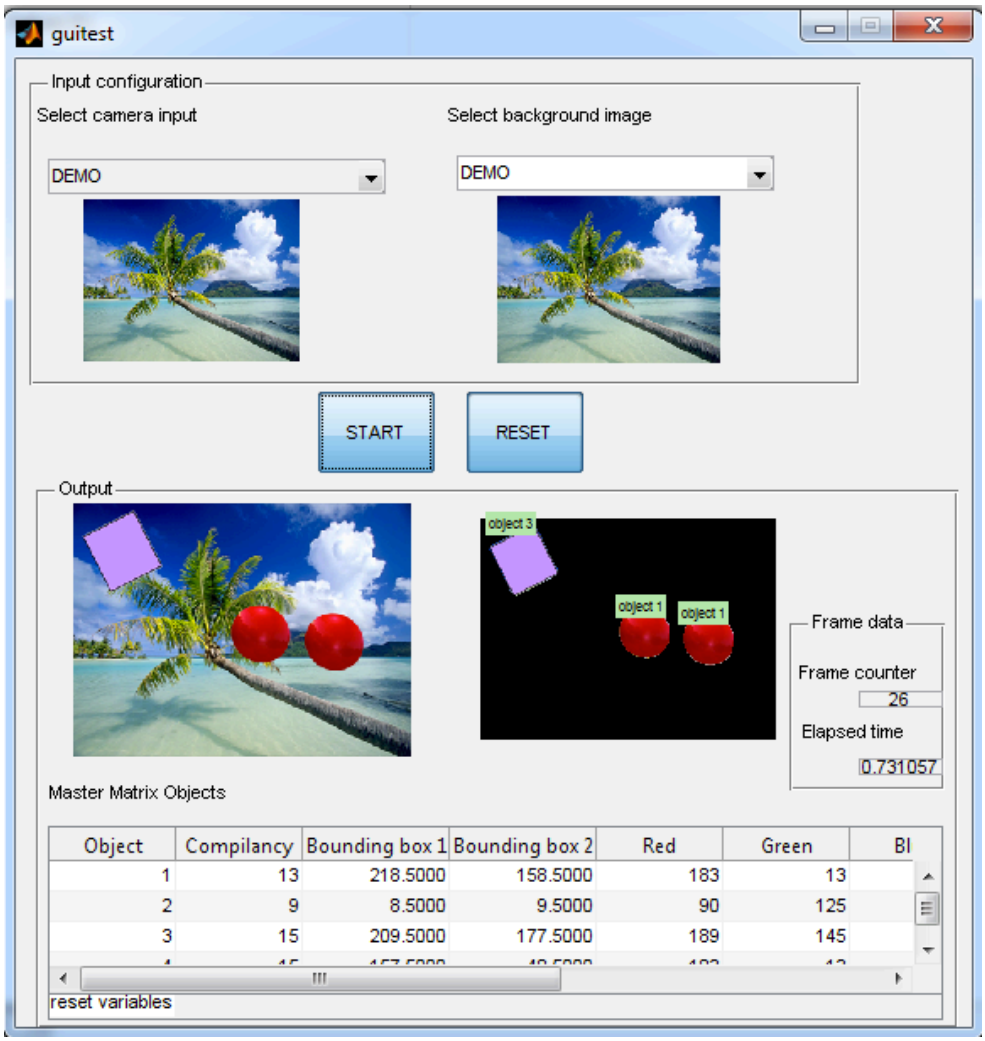
3.4 Muestra de resultados y realimentación del sistema

Una vez obtenido el vector “S”, el sistema muestra la imagen sin procesar de los objetos añadiendo a cada objeto una etiqueta extraída del identificador correspondiente. La matriz maestra que se ha obtenido al final de interpretar la actual imagen, es la matriz maestra en la que se basa la interpretación de objetos de la siguiente imagen.

4. La interfaz gráfica

Todas las anteriores funciones descritas se implementaran en un entorno gráfico con el fin de crear una interfaz que ermita una fácil y cómoda ejecución de cada uno de las diferentes fases desarrolladas en el apartado anterior. Además se recopilan y muestran algunos datos que permitan entender mejor el procesamiento de la interpretación en dicha interfaz gráfica. Toda la interfaz se generará con el entorno de creación de interfaces que incluye *MatLab*.

Detrás de cada elemento interactivo que forman la interfaz gráfica (GUI - Graphical User Interface), esta su correspondiente función llamada *callback*. En esta función se escribe el código que se tendrá que ejecutar al interactuar con el elemento. Tanto el contenido de los menús *popup*, como los textos estáticos y lo títulos de los paneles, son configurados previamente en *MatLab*. Una excepción es el menú *popup* de la selección de camera, ya que su contenido puede variar según el terminal en el que se ejecuta el programa. Para su creación se utiliza el comando de *uicontrol*, que permite la creación de menús desde el propio programa.



La GUI constará de cuatro gráficas, que se irán activando según les llegue la información. La primera muestra el input de los imágenes con objetos a reconocer. Si se ha elegido la opción de imagen desde archivo, se mostrará la imagen seleccionada, y si el input seleccionado corresponde con una cámara se mostrará una vista previa del dispositivo. La segunda gráfica corresponde con el campo de selección de fondo. De nuevo se puede elegir entre una imagen desde archivo, o tomar una imagen desde la cámara seleccionada como dispositivo de entrada. La selección de archivo en la configuración de la imagen de entrada y fondo solo trabaja con imágenes TIF, ya que no presentan perdidas en su codificación. Además solo se puede trabajar en un modo, es decir si el input seleccionado es de archivo, el fondo también lo ha de ser. Si se elige trabajar con una cámara, el fondo ha de ser tomado como foto con el mismo dispositivo. La tercera gráfica muestra la imagen del dispositivo de entrada junto con las etiquetas de los objetos que se han encontrado en ella. La cuarta gráfica muestra la salida de la función *removebackground.m*, es decir los objetos sin el fondo.

Además se ha incluido un panel llamado *frame* data que recopila los datos del número de la imagen actual y el tiempo que tarda el programa en procesar e interpretar cada imagen.

La matriz maestra, que contiene a todos los objetos que se vayan encontrando durante la interpretación, se muestra debajo de las gráficas. Se ha incluido un pequeño campo de texto llamado log, que informa acerca de algunos errores de manejo y ejecución.

Los botones principales están compuestos por un botón de START que inicia el proceso de reconocimiento, con las inicializaciones configuradas previamente, y un botón de RESET que reinicia los valores de la matriz maestra y de la configuración de la *WebCam*.

Se ha incluido un modo de demostración, que consta de un conjunto de 27 imágenes que se cargarán automáticamente al seleccionar este modo. Durante la demostración se verán tres objetos distintos sobre un fondo no uniforme, y se verán algunas capacidades de reconocimiento del programa como traslaciones de objetos en el espacio, giros de objetos, fusión de objetos para formar nuevos y la aparición repetitiva del mismo objeto.

5. Conclusiones y futuras líneas de investigación

5.1 Conclusiones

Como hemos visto en el desarrollo de este PFC, el procesamiento y la interpretación correcta de los objetos contenidos en una imagen es una tarea compleja y computacionalmente cara. Cualquier reconocimiento de objetos requiere una capacidad de computo muy elevada debido a que las imágenes a procesar son matrices tridimensionales grandes. Si bien el programa desarrollado no presenta mayores dificultadas en interpretar objetos contenidos en imágenes estáticas cargadas desde un archivo, la implementación en tiempo real con imágenes obtenidas desde una *WebCam* presenta numerosas dificultades.

Entre ellas están las distorsiones y ruidos introducidos por un entorno no controlado que dificultan considerablemente el reconocimiento. Especialmente durante el desarrollo del proyecto en su fase de experimentación en tiempo real, la introducción de pequeñas sombras aleatorias que modifican el color del objeto y la oclusión parcial de los objetos que modifican su forma, han demostrado la debilidad del sistema frente a estas alteraciones.

En un entorno descontrolado, como lo presenta el mundo real, hace falta un algoritmo de procesado muy robusto y sensible que permita separar el fondo del objeto según las condiciones de las escena de la cual se captan. El procesamiento de las imágenes en el descrito proyecto es un proceso genérico, que no varia independientemente de las características del entorno en el que se lanza el reconocimiento. Esto conlleva numerosas desventajas y una gran inflexibilidad a la hora de separar el fondo de la imagen. De nuevo se puede observar que ampliando el código para un procesado sensible y adaptable a cada entorno conlleva a más tiempo de ejecución por imagen, alejándonos de un resultado en tiempo real.

También la clasificación de los objetos, y sobretodo el parámetro clasificador que escogía la relación entre la compilancia, color y cercanía para interpretar el objeto, son fases de especial delicadeza en el reconocimiento. Se pueden dar casos en los que objetos distintos en forma pero del mismo color y relativamente cercanos entre ellos, sean interpretados con la misma etiqueta. Una solución a estos casos extremos es la inclusión de mas parámetros que describen el objeto. Ello aumenta el vector de características del objeto, lo que conlleva una mayor matriz maestra y un mayor tiempo de ejecución del código de comparación por imagen. Debido a ello se han mantenido los parámetros en los básicos para intentar alcanzar un reconocimiento en tiempo real.

La actualización de la matriz maestra con los nuevos datos del objeto encontrado, se puede considerar un proceso de clasificación por aprendizaje. El programa es capaz de relacionar como mismos objetos a aquellos que presentan ligeras variaciones en forma y color, siempre y cuando estos no superen el umbral establecido para la interpretación. Otra manera de actualizar la matriz maestra hubiera sido incluir cualquier variación del mismo objeto como un vector nuevo, pero asignándole la misma etiqueta. De esta manera se generaría una base de datos con una amplia variedad de posibles apariciones del mismo objeto. Debido a que la búsqueda de elementos en una matriz es costoso, se ha descartado dicha posibilidad, y se ha implementado un código que actualiza el color y la forma haciendo la media respecto a la imagen anterior. De esta manera se asegura que la matriz maestra tenga el menor tamaño posible para acelerar el proceso de comparación.

Como conclusión se puede definir la realización de un sistema en tiempo real como un balance entre error de interpretación admisible y el tiempo de computo por imagen limite para alcanzar el tiempo real, dependiente de las características de computación del ordenador.

5.2 Futuras líneas de investigación

Uno de los puntos más interesantes para futuras líneas de investigación, es la separación de objeto y fondo en un sistema dinámico como lo puede ser la visión artificial de un robot móvil. En este proyecto el fondo es invariable en el tiempo y permanece inmóvil durante todo el proceso de reconocimiento. La complejidad del desarrollo de un sistema que permita separar los objetos del fondo cuando éste también varia con el tiempo, resulta de gran interés.

En la fase de procesamiento se pueden estudiar numerosas mejoras. Para solucionar la problemática dada por la oclusión parcial e incluso completa que pueden presentar algunos de los objetos en la imagen, se podría implementar un filtro de *Kalman*. Este filtro predice la posición del objeto basándose en los parámetros de su posición en la imagen anterior a la actual. Estos parámetros pueden incluir valores de velocidad, dirección y aceleración, que permiten estimar la posición aproximada del objeto aunque éste se encuentre oculto.

También la aplicación de códigos de detección de esquinas como el SIFT, podrían resultar muy eficaces en el reconocimiento de objetos parcialmente ocultados. Este código filtra las esquinas mas importantes del

objeto, estableciendo una malla de conexiones entre las distintas esquinas encontradas. De esta manera, aunque el objeto cambie su apariencia debido a cambios en la perspectiva, el programa es capaz de identificarlo hasta cierto punto.

Como citado con anterioridad en el desarrollo de esta memoria, se podría cargar la matriz maestra con diversos objetos ya etiquetados, en vez de generarla y actualizarla desde cero conforme avanza el código. De esta manera se podría llegar a optimizar el tiempo que tarda el programa en reconocer muestras muy similares como idénticas. También la tolerancia del programa frente a cada parámetro característico del objeto, se podría implementar de tal forma, que su valor sea dinámico y se vaya ajustando a la desviación típica que puede presentar la muestra.

Claro esta, que todos estos procesos de ampliación han de respetar el tiempo de computo limite admisible para percibir el sistema como en tiempo real.

Para entender los procesos que se necesitan para realizar un reconocimiento, el IMAGE TOOLBOX integrado de *MatLab* es una herramienta perfecta, pero para lograr mejores resultados hace falta una compilación del código en otros lenguajes que permitan la creación de ejecutables directos. Una optimización del uso de la memoria y un mayor control sobre todos los procesos del código solo es permitida en lenguajes de programación de alto nivel como lo puede ser C++. La adaptación del código y la creación de una interfaz gráfica fuera del entorno limitado de *MatLab* resulta inevitable para futuras investigaciones.

Bibliografía

[1] Sergios Theodoridis, Konstantinos Koutraoumbas, “An Introduction to pattern recognition, a MatLab approach”, Academic Press is an imprint of Elsevier 2010

[2] Frank Y. Shih, “Image processing and pattern recognition: Fundamentals & techniques” IEEE Press 2010

[3] Welch & Bishop, “An introduction to the Kalman Filter” UNC-Chapel Hill 2006

[4] Dorin Comaniciu, Peter Meer, “ Mean Shift: A robust approach toward feature space analysis”, IEEE Transactions on pattern analysis and machine intelligence Vol. 24 No. 5 2002

[5] www.mathworks.es

Apéndice

A continuación se incluye todo el código implementado de todas las funciones que componen el programa.

```
function varargout = guitest(varargin)
% GUITEST M-file for guitest.fig
%      GUITEST, by itself, creates a new GUITEST or raises the
existing
%      singleton*.
%
%      H = GUITEST returns the handle to a new GUITEST or the handle
to
%      the existing singleton*.
%
%      GUITEST('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in GUITEST.M with the given input
arguments.
%
%      GUITEST('Property','Value',...) creates a new GUITEST or raises
the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before guitest_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to guitest_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows
only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help guitest

% Last Modified by GUIDE v2.5 13-Dec-2012 12:47:14

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @guitest_OpeningFcn, ...
                   'gui_OutputFcn',  @guitest_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
```

```
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before guitest is made visible.
function guitest_OpeningFcn(hObject, eventdata, handles, varargin)
handles = guidata(hObject);

handles.sframe=0;
handles.M=zeros(1,11);
handles.video=0;
a=imaqhwinfo;
camera_name = char(a.InstalledAdaptors);
camera_info = imaqhwinfo(camera_name);
[A,B]=size(camera_info.DeviceIDs);

for k = 1 : B
        C=camera_info.DeviceInfo(1,k).DeviceName;
        dropdownListStrings{k+2}=sprintf(C, num2str(k));
end
dropdownListStrings{1}=['Image from file'];
dropdownListStrings{2}=['DEMO'];
uicontrol('Style', 'popup',...
          'Position', [20 455 200 60],...
          'String', dropdownListStrings,...
          'Callback' , @popupmenu1_Callback );


handles.log = uitable('Position', [20 5 530 20]);
handles.t = uitable('Position', [20 20 530 100]);
set(handles.t, 'ColumnName', {'Object', 'Compilancy', 'Bounding box
1', ...
                              'Bounding box 2', 'Red', 'Green',
'Blue', ...
                              'Centroid 1','Centroid 2'});
set(handles.t, 'RowName', []);
set(handles.log, 'RowName', []);
set(handles.log, 'ColumnName',[]);

% Choose default command line output for guitest
handles.output = hObject;
% Update handles structure
guidata(hObject, handles);
end
end


function varargout = guitest_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
end

%select image input
function popupmenu1_Callback(hObject, eventdata, handles)
handles = guidata(hObject);
val = get(hObject,'Value');

%from file
if val == 1
```

```
    [FileName,PathName] = uigetfile('*.tif','Select image');
    Imwork=imread(FileName);
    handles.Imwork = imresize(Imwork, [240 320]);
    imshow(handles.Imwork,'Parent', handles.axes4);
    handles.mode=0;
%demo mode
elseif val == 2
    handles.imagelist = dir('*.tif');
    handles.N = numel(handles.imagelist);
    handles.Imback = imread(handles.imagelist(1).name);
    handles.Imback = imresize(handles.Imback, [240 320]);
    handles.Imwork=imread(handles.imagelist(1).name);
    handles.Imwork = imresize(handles.Imwork, [240 320]);
    imshow(handles.Imwork,'Parent', handles.axes4);
    handles.mode=2;
    handles.modeb=2;
%camara 1
elseif val == 3
    handles.video = videoinput('winvideo', 1);
    set(handles.video, 'FramesPerTrigger', Inf);
    set(handles.video, 'ReturnedColorspace', 'rgb');
    handles.video.FrameGrabInterval = 5;
    vidRes = get(handles.video, 'VideoResolution');
    nBands = get(handles.video, 'NumberOfBands');
    hImage = image(zeros(vidRes(2),vidRes(1),
nBands),'Parent',handles.axes4);
    start(handles.video);
    preview(handles.video, hImage);
    handles.mode=1;
%camara 2
elseif val == 4
    handles.video = videoinput('winvideo', 2);
    set(handles.video, 'FramesPerTrigger', Inf);
    set(handles.video, 'ReturnedColorspace', 'rgb');
    handles.video.FrameGrabInterval = 5;
    vidRes = get(handles.video, 'VideoResolution');
    nBands = get(handles.video, 'NumberOfBands');
    hImage = image(zeros(vidRes(2),vidRes(1),
nBands),'Parent',handles.axes4);
    start(handles.video);
    preview(handles.video, hImage);
    handles.mode=1;
end

% Update handles structure
guidata(hObject, handles);
end


function popupmenu1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end

%select background
function popupmenu2_Callback(hObject, eventdata, handles)
handles = guidata(hObject);
```

```
val = get(hObject,'Value');
if val == 1
    [FileName,PathName] = uigetfile('*.tif','Select background
image');
    Imback=imread(FileName);
    handles.Imback = imresize(Imback, [240 320]);
    handles.modeb=0;
elseif val == 2
    handles.imagelist = dir('*.tif');
    handles.N = numel(handles.imagelist);
    handles.Imback = imread(handles.imagelist(1).name);
    handles.Imback = imresize(handles.Imback, [240 320]);
    handles.Imwork=imread(handles.imagelist(1).name);
    handles.Imwork = imresize(handles.Imwork, [240 320]);
    imshow(handles.Imwork,'Parent', handles.axes4);
    handles.mode=2;
    handles.modeb=2;
elseif val == 3
    preview(handles.video);
    Imback = getsnapshot(handles.video);
    handles.Imback = imresize(Imback, [240 320]);
    handles.modeb=1;
end

imshow(handles.Imback,'Parent', handles.axes2);

% Update handles structure
guidata(hObject, handles);
end


function popupmenu2_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end

%START
function togglebutton1_Callback(hObject, eventdata, handles)
handles = guidata(hObject);


if handles.mode == 0 || handles.modeb == 0
    tic
    Imwork=handles.Imwork;
    Imback=handles.Imback;
    im=rgb2gray(Imwork);
    bg=rgb2gray(handles.Imback);
    bgim=imabsdiff(bg,im);
    dif=sum(sum(bgim));
    if dif > 0
        [nobg,fore]=removebackground(Imwork,Imback);
        imshow(fore,'Parent',handles.axes3);
        S=objectstat(nobg,fore);
        if sum(handles.M)==0
            handles.M=S;
        else
           [S,handles.M]=matrixfind2(S,handles.M);
        end
```

```
        imshow(Imwork,'Parent',handles.axes6);
        objecttext(S)
        set(handles.t, 'Data', handles.M);
    else
        set(handles.log,'Data',{'error: no object found'});
    end

    %time control display
    v=toc;
    uicontrol('position',[500,150,50,10],'style','edit','string',v);
    v=0;

    %frame counter

uicontrol('position',[500,190,50,10],'style','edit','string',handles.s
frame);
    handles.sframe=handles.sframe+1;


elseif handles.mode==1 || handles.modeb==1
    %closepreview(handles.video);
    bg=rgb2gray(handles.Imback);
    for i=0:60

    tic
    Imwork=getsnapshot(handles.video);
    Imwork = imresize(Imwork, [240 320]);

    im=rgb2gray(Imwork);
    bgim=imabsdiff(bg,im);
    dif=sum(sum(bgim));

    %si la diferencia es mayor que 150k interpretamos
    if dif > 150000
        [nobg,fore]=removebackground(Imwork,handles.Imback);
        imshow(fore,'Parent',handles.axes3);

        S=objectstat(nobg,fore);

        %si la matriz maestra es nula, inicializamos con S
        if sum(handles.M)==0
            handles.M=S;
        else
        [S,handles.M]=matrixfind2(S,handles.M);
        end
        imshow(Imwork,'Parent',handles.axes6);
        objecttext(S)
    %si no es mayor que 150k, error no hay objeto
    else
        fore=zeros(240,320);
        set(handles.log, 'Data', {'error: no object found'});
        imshow(fore,'Parent',handles.axes3);
    end

    %Show master matrix in data box t
    set(handles.t, 'Data', handles.M);

    %time control display
    v=toc;
```

```
    uicontrol('position',[500,150,50,10],'style','edit','string',v);
    v=0;

    %frame counter
    uicontrol('position',[500,190,50,10],'style','edit','string',i);


    end
    stop(handles.video);
    flushdata(handles.video);
elseif handles.mode==2 || handles.modeb==2
    Imback=handles.Imback;
    bg=rgb2gray(Imback);
    for k = 2: handles.N
        tic
        Imwork=imread(handles.imagelist(k).name);
        Imwork = imresize(Imwork, [240 320]);
        im=rgb2gray(Imwork);
        bgim=imabsdiff(bg,im);
        dif=sum(sum(bgim));
        if dif > 0
            [nobg,fore]=removebackground(Imwork,Imback);
            imshow(fore,'Parent',handles.axes3);
            S=objectstat(nobg,fore);
            if sum(handles.M)==0
                handles.M=S;
            else
            [S,handles.M]=matrixfind2(S,handles.M);
            end
            imshow(Imwork,'Parent',handles.axes6);
            objecttext(S)
            set(handles.t, 'Data', handles.M);
        else
            set(handles.log,'Data',{'error: no object found'});
        end

        %time control display
        v=toc;

        uicontrol('position',[500,150,50,10],'style','edit','string',v
        );
        v=0;

        %frame counter

        uicontrol('position',[500,190,50,10],'style','edit','string',h
        andles.sframe);
        handles.sframe=handles.sframe+1;
        pause(0.5);
    end

else
    set(handles.log, 'Data', {'error: input & backround missmatch'});
end
% Update handles structure
guidata(hObject, handles);
end

% --- Executes on button press in togglebutton2.
function togglebutton2_Callback(hObject, eventdata, handles)
```

```
handles = guidata(hObject);

handles.sframe=0;
handles.M=zeros(1,11);
set(handles.log, 'Data', {'reset variables'});
set(handles.t, 'Data', handles.M);
if handles.video ~= 0
    stop(handles.video)
    flushdata(handles.video)
end

% Update handles structure
guidata(hObject, handles);
end


function S=objectstat(nobg,fore)
labeled = bwlabel(nobg,8);
stats =
regionprops(labeled,'Area','Perimeter','BoundingBox','Centroid');
[N,W] = size(stats);
S=zeros(N, 11);

for i=1:N
 S(i,1)=i;
 a = stats(i).Area;
 p = stats(i).Perimeter;
 c = (a/p);
 S(i,2)=c;
 S(i,10)=a;
 S(i,11)=p;

 bb=stats(i).BoundingBox;
 S(i,3)=bb(1);
 S(i,4)=bb(2);
 subImage = imcrop(fore, bb);

 m=round(mediacolor(subImage));
 S(i,5)=m(1);
 S(i,6)=m(2);
 S(i,7)=m(3);

 cc=stats(i).Centroid;
 S(i,8)=cc(1);
 S(i,9)=cc(2);
end


function [x2,M]=matrixfind2(x2,M)
aux=x2; %copiamos x2 en una matriz auxiliar
[N,W] = size(M);
[N2,W2]=size(aux);

for i=1:N2     %recorremos todos los obetjos de la nueva imagen
    ymax=0;
    dmin=10000;
    for j=1:N  %buscamos a cada objeto en la matriz master
        y=[find(abs(M(j,2)-aux(i,2))<=2) find(abs(M(j,10)-
aux(i,10))<=30) find(abs(M(j,11)-aux(i,11))<=10) find(abs(M(j,5)-
aux(i,5))<=10) find(abs(M(j,6)-aux(i,6) )<=10) find(abs(M(j,7)-
```

```
aux(i,7))<=10)];
        y=sum(y);  %cada vez q una caracteristica del nuevo objeto
coincide con uno de la matriz master, se activa un elemento de "y"
        x = [aux(i,8),aux(i,9);M(j,8),M(j,9)];
        d = pdist(x,'euclidean');
        if d<dmin
            dmin=d;
            y=y+1;
        end
        if y>ymax   %averiguamos la y maxima y la guardamos en y2
            ymax=y;
            z=j;  %guardamos la posicion de la "y" maxima "j" en otra
variable "z"
        end
    end
    if  ymax==7              %si la y maxima ha sido un 4: 100% de
coincidencia y asi sucesivamente (k)
        x2(i,1)=M(z,1);    %sustituimos el indice de objeto x2(i,1)
por el dado en la matriz master
        M(z,2)=round((x2(i,2)+M(z,2))/2);
        M(z,5)=round((x2(i,5)+M(z,5))/2);
        M(z,6)=round((x2(i,6)+M(z,6))/2);
        M(z,7)=round((x2(i,7)+M(z,7))/2);
        M(z,8)=x2(i,8);
        M(z,9)=x2(i,9);
        M(z,10)=round((x2(i,10)+M(z,10))/2);
        M(z,11)=round((x2(i,11)+M(z,11))/2);
        k=100;
    elseif  ymax==6
        x2(i,1)=M(z,1);
        M(z,2)=round((x2(i,2)+M(z,2))/2);
        M(z,5)=round((x2(i,5)+M(z,5))/2);
        M(z,6)=round((x2(i,6)+M(z,6))/2);
        M(z,7)=round((x2(i,7)+M(z,7))/2);
        M(z,8)=x2(i,8);
        M(z,9)=x2(i,9);
        M(z,10)=round((x2(i,10)+M(z,10))/2);
        M(z,11)=round((x2(i,11)+M(z,11))/2);
        k=80;
    elseif  ymax==5
        x2(i,1)=M(z,1);
        M(z,2)=round((x2(i,2)+M(z,2))/2);
        M(z,5)=round((x2(i,5)+M(z,5))/2);
        M(z,6)=round((x2(i,6)+M(z,6))/2);
        M(z,7)=round((x2(i,7)+M(z,7))/2);
        k=60;
    elseif  ymax <= 4
        [N,W] = size(M);
        x2(i,1)=N+1;      %si no lo encontramo (y==0) a?adimos una
nueva fila con ese objeto a M

        M=[M;x2(i,:)];
        k=0;
    end
end


function [nobg,fore]=removebackground(Imwork,Imback)
bg=rgb2gray(Imback);
im=rgb2gray(Imwork);
```

```
nobg = ((im > bg+5) | (im < bg-5));

nobg = medfilt2(nobg, [9 9]);
nobg=bwareaopen(nobg,600,8);
nobg=imfill(nobg,'holes');

nobg = uint8(nobg);

fore(:,:,1)=Imwork(:,:,1).*nobg;
fore(:,:,2)=Imwork(:,:,2).*nobg;
fore(:,:,3)=Imwork(:,:,3).*nobg;


function m=mediacolor(subImage)
o=[0,0,0];
m=[0,0,0];
[x,y,z]=size(subImage);
for i=1:x
    for j=1:y
        for k=1:z
            if subImage(i,j,k)>0
                o(k)=o(k)+1;      %por cada dimension el numero de
elmentos distintos de 0
            end
            p=sum(sum(subImage)); %La suma de todos los puntos en una
dimension k
            m(k)=p(:,:,k)/o(k);   %la media
        end
    end
end


function objecttext(S)

[N,W] = size(S);
for i=1:N
text(S(i,3),S(i,4),['object',num2str(S(i,1))],'FontSize',6,'Back
groundColor',[.7 .9 .7]);
end
```