UNIVERSIDAD DE ZARAGOZA

CENTRO POLITÉCNICO SUPERIOR

PROGRAMA OFICIAL DE POSGRADO EN INGENIERÍA DE TELECOMUNICACIÓN

MÁSTER EN INGENIERÍA ELECTRÓNICA

TRABAJO FIN DE MÁSTER

Implementación de algoritmos de detección de personas en imágenes utilizando métodos de planificación y estimación en el desarrollo software

Autor: Raúl Igual Catalán

Director: Carlos Tomás Medrano Sánchez

Codirectora: Inmaculada Plaza García

4 de Septiembre de 2010

Curso 2009/2010

TRABAJO FIN DE MÁSTER

Implementación de algoritmos de detección de personas en imágenes utilizando métodos de planificación y estimación en el desarrollo software

RESUMEN

En este Trabajo Fin de Máster se ha implementado un algoritmo de detección de personas en imágenes, aplicando conceptos de calidad a la gestión y al desarrollo del proyecto. Se ha elegido la norma UNE 166001 sobre Requisitos de Proyectos de Investigación para trabajar de manera ordenada y metódica, evitando obviar aspectos importantes. Entre otras actuaciones, se realiza una estructuración del proyecto en fases con una temporalización asociada, se desglosa cada una de las fases en las tareas que la componen, se estima el presupuesto del proyecto, etc.., en definitiva, se trata de aplicar los principios de la calidad a la gestión del trabajo fin de máster.

Para implementar algoritmos de visión por computador con calidad, se deberán considerar aquellos **modelos y estándares que sean de aplicación**. De este modo, se ha realizado una revisión normativa sobre modelos que apliquen tanto al desarrollo del producto software como al propio producto software, seleccionando razonadamente los más idóneos:

- Como modelo que regirá el **desarrollo** del producto, se ha tomado el **Proceso de Software Personal** (PSP), que permite mejorar la eficiencia del desarrollador a nivel individual.
- Como modelo aplicable al propio **producto** software se ha seleccionado la normativa **ISO 9126**, ya que puede ser adaptada a la evaluación del programa desarrollado.

Una vez se dispone de los modelos o estándares de referencia, ya es posible diseñar el algoritmo. Previamente se ha realizado una **búsqueda sobre métodos de detección de personas**, y a la vista de esos resultados, se ha optado por detectar la cabeza, por ser la parte más significativa del cuerpo. Para ello **se empleará el Modelo Omega** que representa la cabeza a través de una figura geométrica con la forma de la letra Ω.

Siguiendo este modelo se ha implementado un **algoritmo de detección** en el **lenguaje de programación C**, utilizado como soporte la **librería OpenCV**, necesaria para el manejo de imágenes en C. En una primera fase, la detección se realiza sobre imágenes donde únicamente aparece una cabeza aislada. En fases posteriores la detección se ha realizado sobre imágenes de personas que atraviesan un pasillo, con el consiguiente aumento en la complejidad del algoritmo. Éste se estructura en diversas operaciones: generación del modelo omega ideal, detección de fondo, obtención de la transformada de distancia, detección de bordes, dilatación, cálculo de gradientes, medición del ajuste final a través de una función matemática, suavizado de la función, cálculo de máximos, etc.

Una vez el algoritmo ha sido implementado, se **ha evaluado** a través de la aplicación del modelo normativo ISO 9126. También se han **analizado los resultados** del algoritmo; su efectividad realizando la detección, así como la **velocidad de ejecución**, que se encuentra muy próxima al tiempo real. Se ha valorado el **cumplimiento de lo planificado**, la aplicación del PSP, y se han establecido **actuaciones futuras** derivadas de este proyecto.

0. Tabla de contenidos

Anexos

0.1 Índice de figuras

Figuras de los Anexos

0.2 Índice de Tablas

Tablas de los Anexos

1. Introducción

1.1 Alcance del proyecto

Este proyecto fin de Máster se enmarca en **dos ámbitos** bien diferenciados dentro del campo de conocimiento de la Ingeniería Electrónica: la **Visión por Computador** y la **Gestión de la Calidad**.

El proyecto se basa en implementar un algoritmo que recibe una imagen de entrada, y debe ser capaz de detectar a las personas presentes en la misma, con una tasa de errores aceptable. Además, se propone que esa implementación se realice "con calidad", es decir, dando repuesta a los requisitos definidos y a los objetivos marcados, pero buscando el apoyo de los modelos o estándares referidos al software y a la gestión de proyectos, para mejorar nuestro trabajo.

1.2 Objetivos del proyecto

A continuación se enuncian de modo general, los objetivos que se pretenden alcanzar con la realización de este trabajo. Para ello se van a clasificar en los dos ámbitos que abarca el proyecto:

Desde el ámbito de la visión por computador:

- Realizar el estado del arte sobre las técnicas existentes para la detección de personas.
- **Diseñar, desarrollar y evaluar un algoritmo de detección de personas**, que sea susceptible de ser utilizado, en un futuro, para realizar seguimiento de personas. Para ello se **detectará la cabeza** de las personas por ser la parte más significativa del cuerpo.
- Realizar un **análisis de tiempos** del algoritmo implementado, valorando la viabilidad de que esa detección se pueda realizar **en tiempo real**.

Desde el ámbito de la gestión de la calidad:

- **Gestionar el proyecto** siguiendo lo dispuesto en la normativa **UNE 166001** que se refiere a los requisitos de los proyectos de investigación. Entre otros aspectos, en esta normativa se expone que la gestión debe abarcar los siguientes campos: revisión normativa sobre la materia, descripción de la innovación que supone, memoria, planificación, presupuesto, control de la documentación y plan de explotación de resultados.
- Realizar el **estado del arte sobre los modelos** y estándares normativos que sean **aplicables al software**, en todos sus ámbitos: su concepción, su desarrollo, su evaluación final...
- Seleccionar los modelos de calidad de software que mejor se adapten a las características del proyecto, y **aplicarlos a la implementación del algoritmo** seleccionado para este proyecto.

Como objetivo global del proyecto se pretende que **los objetivos** enunciados para ambos ámbitos se desarrollen paralelamente de manera **interrelacionada**.

2. Gestión del proyecto

La gestión de este proyecto se va a realizar siguiendo la norma UNE 166001, donde se especifica, entre otros aspectos [UNE-166001], que los proyectos de investigación se deben estructurar en una serie de fases, definiendo el alcance de cada una de ellas.

2.1 Estructura general

Este proyecto se ha estructurado en seis fases fundamentales: **planificación, organización, ejecución, control, análisis de resultados y presentación de resultados o cierre**. Las diferentes fases se relacionan y conectan entre sí, de la forma representada en la figura 1.

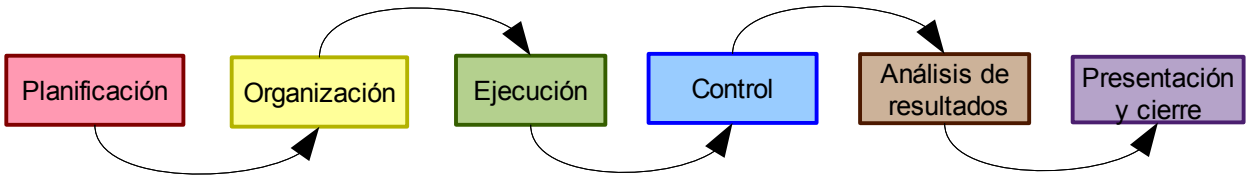


Figura 1. *Fases del proyecto*

El orden de ejecución de las fases es el indicado en la figura 1, aunque durante el transcurso del proyecto puede haber retro-alimentación volviendo a fases anteriores en caso de ser necesario. La mayor interacción se produce entre las fases de ejecución y control, donde a medida que se desarrollan los algoritmos, se establece un protocolo de control paralelo, para detectar a tiempo las posibles deficiencias. En el Anexo I se incluye un esquema desglosado del proyecto, donde se detalla la relación entre las fases.

Esta estructura se caracteriza por su dinamismo, ya que está abierta a la introducción de modificaciones en función de la evolución del proyecto.

2.2 Planificación específica

A continuación, se describe brevemente el contenido de cada una de las fases. En el Anexo II se incluye una descripción mucho más detallada, enumerando las diferentes tareas que abarca cada una de ellas.

Planificación: La fase de planificación es, junto con la de ejecución, una de las fases clave del proyecto. En ella se estructura el proyecto en fases, se realiza el estado del arte tanto en el ámbito de visión por computador, como de los modelos de calidad aplicables al software. Como resultado de este estudio previo, se concretarán los algoritmos que se van a implementar, así como los diferentes procesos para lograr esa implementación. También se establecerán unas directrices sobre cuáles deben ser los resultados obtenidos al finalizar la programación y se seleccionarán los modelos que serán de aplicación durante el desarrollo del proyecto, adaptándolos al software de visión seleccionado.

Organización: Durante esta fase se dispondrá de todo lo necesario para poder acometer el proyecto con garantías. Esto incluye la adquisición de recursos y su puesta a punto. También se establecerá un plan de formación en diversos ámbitos: el matemático, el ámbito de la programación y el ámbito de la calidad.

Ejecución: Esta es la fase crucial del proyecto. En ella se desarrollará el algoritmo de detección de personas siguiendo el modelo de programación seleccionado. El desarrollo del algoritmo incluye su comprensión, su codificación, su depuración y su optimización. Además, se deberán registrar los resultados obtenidos durante la ejecución y los parámetros que influyen en ellos. También se deberá seguir lo dispuesto en la planificación, realizando anotaciones en los registros correspondientes.

Control: Esta fase está íntimamente relacionada con la fase de ejecución. Durante su desarrollo se comprobará que los programas se han implementado y optimizado satisfactoriamente, y para ello se utilizará el modelo de calidad de software seleccionado (ver apartado 2.4.2).

Paralelamente se realizará el seguimiento sobre toda la estructura del proyecto mediante el control de la planificación (cuantificar en qué grado se está cumpliendo lo planificado).

Análisis de resultados, presentación y cierre: En esta fase se recapitula todo lo acontecido durante el proyecto, dejando constancia escrita a través de la redacción de una memoria. Además se documentarán y organizarán todos los algoritmos, para poder ser consultados en cualquier momento. De igual modo, se registrarán y organizarán la búsquedas realizadas en el ámbito de la calidad, así como los resultados de la planificación, que servirán de guía a la hora de abordar nuevos proyectos.

Se extraerán conclusiones basadas en la experiencia adquirida durante el proyecto, que servirán de referencia a la hora de acometer futuros trabajos. Además, los resultados serán difundidos en público para que pueda valorarse la calidad del trabajo realizado.

2.3 Temporalización del proyecto

Tal y como recoge la norma UNE 166001, en todo proyecto es necesario planificar la duración de cada una de sus fases. En este caso, estimamos que la duración total del proyecto es de **103 días hábiles**, entendiendo por día hábil aquel que tiene una carga de trabajo equivalente a **5 horas**.

En la tabla 1 y la figura 2 se muestra la temporalización general a nivel de fase, y en el Anexo II se incluye la temporalización detallada de las tareas que componen cada una de las fases.

Fase	Tiempo (días)	Porcentaje %
Planificación	24	23,3
Organización	9	8,7
Ejecución	44	42,7
Control	7	6,8
Análisis de resultados presentación y cierre	19	18,4

Tabla 1 y Figura 2. *Temporalización general*

A la vista de la figura 2, podemos deducir que la fase de ejecución, es la que ocupa la mayor parte del proyecto, seguida por la fase de planificación. El resto de fases tienen una duración más limitada.

2.4 Modelos aplicables al proyecto

Este proyecto se va a desarrollar siguiendo los modelos y estándares propuestos por los organismos relevantes en materia de calidad. Durante la fase de planificación se realizó el estado del arte de los modelos que serían aplicables a este proyecto. En el Anexo III puede consultarse el listado de todos los que fueron localizados, clasificados en diferentes categorías.

A la vista de esa clasificación, es posible **dividir los modelos en dos grandes familias**:

- Aquellos que aplican al **desarrollo del producto** software.
- Los que se centran en el **propio producto** software.

Debido a la gran diversidad de modelos existentes no es posible analizarlos y aplicarlos en su totalidad, por lo que después de estudiar varios de ellos, se han seleccionado dos modelos: uno para aplicarlo al desarrollo del producto, y otro para el propio producto. Comenzamos analizando el primero de ellos.

2.4.1 Modelos aplicables al proceso de programación

A la hora de seleccionar el mejor modelo, es necesario conocer el modo de trabajar del programador. En este caso, el proyecto es de carácter individual, y no se espera la intervención a corto plazo de otros programadores que operen sobre el mismo código. Por ello el modelo más idóneo para regir el proceso de programación será aquel que prime la eficiencia personal del programador.

El **Proceso de Software Personal (PSP)** es un modelo que se ajusta perfectamente a esta descripción, por lo que su elección está más que justificada, ya que:

- Permite sistematizar el proceso de programación, mediante el empleo de registros, la realización de comparativas...
- Es un modelo que potencia el aprendizaje gradual y personal.
- Propicia que el programador aprenda de sus propios errores y se habitúe a tomar anotaciones frecuentes durante el proceso de programación, lo que contribuye a su profesionalización.

Descripción del Proceso de Software Personal

El PSP está extensamente descrito en [Humphrey, 01], [Plaza, 10], y es una herramienta que el programador puede emplear a nivel individual durante el desarrollo del código.

El PSP contempla la creación de una tabla de registros para anotar los errores que se cometan durante el proceso de programación. Esta **tabla de registros** está compuesta por varios campos, que deben cumplimentarse para cada uno de los errores:

- Fase del desarrollo del programa en la que los errores fueron introducidos y eliminados. El PSP distingue entre las fases de planificación, diseño, codificación, compilación, depuración y cierre.
- El tiempo empleado en detectar y corregir cada uno de los errores.
- Una clasificación de los errores en los siguientes tipos: documentación (comentarios, mensajes), sintaxis (ortografía, puntuación), construcción de paquetes (librerías, versiones), asignación (declaraciones, alcance), interfaz (llamadas), comprobación (mensajes error), datos (estructura, contenido), función (lógica, punteros), software (memoria) y entorno (pruebas, depurador).

Así, el encabezado de la tabla de registros es el siguiente:

Número	Introducido	Eliminado	Tipo	Tiempo de Corrección

Tabla 2. *Campos de la tabla de registros*

En el Anexo X se incluye esta tabla de registros cumplimentada para diversos errores cometidos.

El PSP recomienda utilizar **indicadores** que permitan estimar si el proceso de programación se está realizando correctamente. En este proyecto se van a utilizar los siguientes:

- Minutos/LOC : Tiempo, en minutos, empleado en escribir una línea de código.

- Errores/KLOC: Número de errores cometidos por cada mil líneas de código.
- V/F: La relación entre el tiempo empleado en revisar el código antes de probarlo, frente al tiempo empleado en probar ese código y corregir los errores no detectados durante la revisión.
- Rendimiento: Cociente entre el número de errores eliminados antes de compilar y el número total de errores introducidos antes de compilar.

En el apartado 6.6.1 se exponen los resultados obtenidos al aplicar el PSP, analizando el uso de estos indicadores.

2.4.2 Modelos aplicables al producto software

Para el producto software se ha seleccionado, de entre todos los modelos enumerados en el Anexo III, la norma ISO 9126 ya que proporciona una visión integral del producto desde dos perspectivas bien diferenciadas: el desarrollador y el usuario. Pero además de esto:

- Realiza una descripción de las características a considerar en el producto software.
- Contempla de manera homogénea el producto software desde su creación hasta su utilización final.
- Es un modelo de gran utilidad para evaluar el producto software, y por tanto, para llevar a cabo lo descrito en la fase de control de este proyecto (apartado 2.2).
- Es relativamente sencillo de implementar.
- Esta norma es la base del proyecto SQuare, una prometedora normativa software que todavía no se encuentra totalmente desarrollada.

Descripción de la norma ISO 9126

En cuanto a la estructura del modelo ISO 9126, cabe destacar que se divide en cuatro partes:

- ISO 9126-1: Calidad del producto software. Este documento describe el modelo de calidad software. Distingue entre calidad interna/externa y calidad en uso.
- ISO 9126-2: Métricas para calidad externa.
- ISO 9126-3: Métricas para calidad interna.
- ISO 9126-4: Métricas para calidad en uso.

Para utilizar con corrección el modelo debe estar perfectamente definida la diferencia entre los **distintos tipos de calidad que define la norma**. Así, la **calidad interna** 'es la totalidad de características del producto desde una perspectiva interna. Se usan para especificar la calidad de productos intermedios'. La **calidad externa**, sin embargo, contempla las características del producto desde una perspectiva externa, incluyendo requisitos derivados de las necesidades de calidad de los usuarios. Por su parte, la **calidad en uso** 'es la visión de calidad que tiene el usuario del producto software cuando lo usa en un entorno y

contexto específico'.

Sintetizando, la calidad interna concibe al producto software desde una perspectiva de desarrollo interno, mientras que en la calidad externa contempla al producto finalizado pero desde el punto de vista del desarrollador. La calidad en uso lo hace desde el punto de vista de los diferentes casos de uso por parte del usuario del producto.

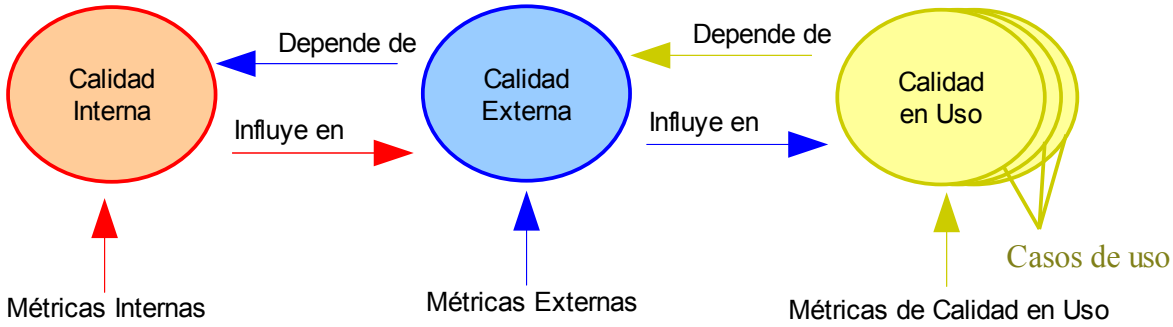


Figura 3. *Modelo de Calidad ISO 9126*

Este modelo se utilizará para evaluar el producto software en la fase de control del proyecto, y en este caso resulta más útil contemplarlo desde la perspectiva de **calidad interna y externa**, ya que el resultado de este proyecto es un algoritmo para ser utilizado por los desarrolladores y no por un usuario final. **Principalmente, nos centraremos en la calidad externa**, por estar más cercana tanto a los aspectos internos como a los aspectos de uso (ver figura 3).

El modelo de calidad interna/externa propuesto por la norma ISO 9126, está compuesto por características y subcaracterísticas. Así, las características que propone este modelo para el producto software son seis: funcionalidad, fiabilidad, usabilidad, eficiencia, mantenibilidad y portabilidad. En este proyecto no entraremos al nivel de subcaracterísticas por sobrepasar los objetivos temporales definidos. No obstante, las características contienen a las subcaracterísticas, con lo que implícitamente sí las estamos considerando.

Los documentos ISO 9126-2 y ISO 9126-3 serán de gran utilidad para este proyecto puesto que muestran la manera de medir el modelo de calidad interna y externa, respectivamente. No obstante hay que tener en cuenta que estas métricas deberán ser adaptadas al software en cuestión (en este caso se trata de software de visión por computador) y el tipo de evaluación que se quiera realizar (evaluación más funcional o más centrada en la optimización del software).

En el Anexo IV se incluye un modelo de calidad, con sus métricas asociadas que puede servir de referencia para evaluar este producto software, tanto antes como después de realizar el proceso de optimización del código. No obstante, cabe remarcar que los evaluadores del software pueden ampliar o adaptar el modelo, para que se centre más en los aspectos que deseen evaluar, aunque siempre se tendrá que tomar como marco de trabajo la norma ISO 9126.

3. Estado del arte sobre métodos de detección de personas

En este apartado presentamos el estado del arte relativo a la parte de Visión por Computador de este proyecto. En el apartado 8 (bibliografía) se incluye la información de todas las referencias aquí citadas.

La detección y seguimiento de personas es una tarea crucial para llevar a cabo numerosas aplicaciones de visión por computador. Si este tipo de problemas se resolvieran con soluciones fiables y en tiempo real, podrían llegar a tener aplicación en varias áreas como la videovigilancia, el análisis de eventos deportivos, el reconocimiento de actividades, etc. Sin embargo, este campo de trabajo presenta una gran complejidad, especialmente si las oclusiones entre las personas son muy frecuentes, como ocurre en las escenas con alta masificación.

En este proyecto **pretendemos implementar un módulo de detección de personas, que en un futuro pueda integrarse en aplicaciones de seguimiento.**

Métodos de detección de personas

Para realizar la detección existen numerosas posibilidades. En [Zhao, 08], la detección se realiza en tres pasos. En primer lugar, se seleccionan los candidatos a cabezas a partir de los picos que presenta el borde del *foreground* (aquella parte de la imagen que no corresponde al fondo). En segundo lugar, se buscan aquellos puntos de la región de *foreground* que mejor encajen con un modelo de cabeza y hombros considerado como ideal (modelo omega).

En [Li, 09] la detección se basa en un clasificador rápido de Viola-Jones [Viola, 01], seguido de un histograma de gradientes (HOG) basado en un clasificador de características AdaBoost [Duda, 01], que se utiliza para detectar el conjunto cabeza-hombros en zonas de entrada predefinidas. Las cabezas detectadas se someten a un procedimiento de seguimiento (tracking) utilizando un algoritmo basado en el filtro de partículas [isard, 98]. En [Garcia, 05] se vuelve a utilizar el modelo omega y se describe en detalle el procedimiento de ajuste. Además se utiliza información de color para eliminar algunos de los candidatos a cabeza.

En [Wu, 09] la detección se realiza sobre el cuerpo completo. Se define una jerarquía de las partes del cuerpo: en primer lugar el cuerpo completo, luego los hombros-cabeza, piernas... Se entrena un detector de cada una de las partes del cuerpo utilizando una versión del algoritmo de Adaboost [Duda, 01]. El clasificador se basa en la respuesta de algunas características de borde (que describen las formas de borde típicas de una silueta humana), seleccionadas para optimizarlo.

En [Birchfield, 98] realizan la detección utilizando un doble modelo elíptico: de gradiente de intensidad y de color. Mientras que en [Yang, 02] se realiza una recopilación de diferentes trabajos sobre detección de cabezas de personas:

- Uno de ellos propone el uso de una red neuronal supervisada, basando su aprendizaje en el modelo de color de la cara. Las cabezas candidatas se someten a un proceso de corrección de la iluminación, y de ecualización. La imagen ecualizada se aplica a la red neuronal.

- También se propone un modelo jerárquico de tres niveles basado en la degradación gradual de la imagen de entrada. La imagen más degradada se utiliza para localizar los primeros candidatos siguiendo un modelo de color. Las imágenes menos degradas se utilizan para confirmar la detección a partir de los rasgos faciales.
- El último de ellos utiliza un modelo un tanto distinto. Divide la cara en diferentes regiones y mide la diferencia de brillo entre esas regiones. La detección es independiente de las condiciones de iluminación de la imagen de entrada, ya que el modelo expresa esa diferencia de modo relativo.

[Eshel, 08] presenta un sistema de detección constituido por varias cámaras. Sobre las imágenes recogidas por ellas, se aplica una homografía, detectando todos los puntos situados a una determinada altura (candidatos a cabezas de las personas).

Selección del modelo

En este proyecto **la detección de personas se realizará utilizando el modelo de hombros-cabeza (modelo omega).**

Este modelo ya ha sido empleado previamente para implementar algoritmos de seguimiento [Zhao, 08]. Por ello esperamos que **su carga computacional no sea muy elevada**, con lo que resultaría factible realizar aplicaciones en tiempo real.

El modelo **es muy intuitivo** ya que los conceptos que maneja no son muy complejos. Además, de implementarse con éxito, permitirá **configurar un módulo de detección de personas en el grupo CVLab**, contribuyendo de esta manera a incrementar el “know-how” que posee el grupo.

4. Estructura del algoritmo

Como ya se comentó en la introducción, la detección de personas se realizará mediante la detección de sus cabezas. Para ello se utilizará un algoritmo que medirá el grado de ajuste de la cabeza que se quiere detectar, con respecto a un modelo de cabeza considerado como ideal. Por tanto, es necesario definir dos aspectos básicos:

- El modelo de cabeza ideal.
- La función que determine el grado de ajuste.

4.1 Modelo de cabeza ideal

El modelo utilizado para realizar la detección de las cabezas se conoce como “modelo omega”, denominado así por su forma (Ω), y en nuestro caso va a estar constituido por los siguientes elementos:

- Un punto central del modelo (figura 4).
- Treinta puntos de borde (figura 4).
- Un vector gradiente unidad asociado a cada uno de los treinta puntos de borde (figura 5).

Todos los elementos del modelo están definidos a través de sus coordenadas *(x,y)*. El punto central se toma como origen de coordenadas y los puntos de borde estarán referidos a él.

Los vectores gradiente tienen su origen en su correspondiente punto de borde del modelo, siendo vectores unitarios.

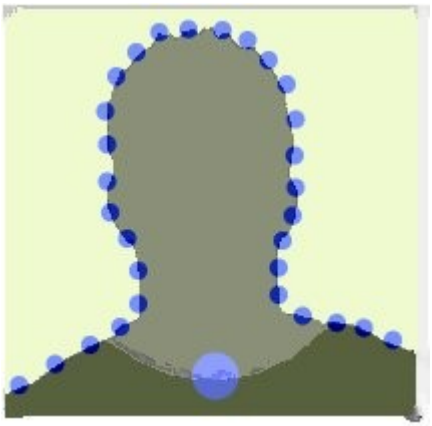


Figura 4. *Punto central y puntos de borde del modelo omega*

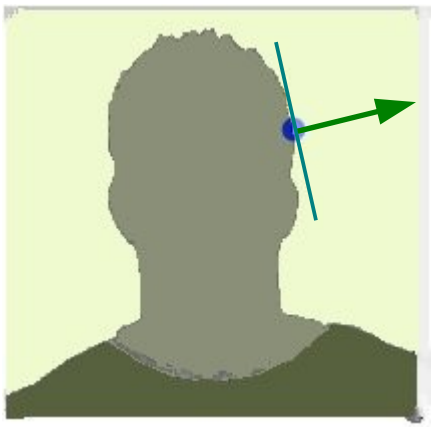


Figura 5. *Vector gradiente en uno de los puntos de borde*

Para determinar los puntos del modelo omega, y los vectores gradiente unidad se han tomado una serie de cabezas de muestra. En esas cabezas se han determinado los puntos de borde de manera manual siguiendo el contorno de la cabeza.

Posteriormente se simetrizaron esos puntos de borde respecto al eje de la cabeza (figura 6),

ajustándolos después con Splines, de manera que al final del proceso se obtuvo el modelo omega definitivo, constituido por los puntos ajustados y su gradiente asociado.

4.2 Función de evaluación del modelo omega

Para medir el grado de ajuste de una supuesta cabeza presente en una imagen de entrada, frente al modelo de cabeza ideal (modelo omega) centrado en el punto *(x,y)* de la imagen de entrada, se ha seguido lo expuesto en [Zhao,08], que pasa por computar la siguiente función:

$$S(x,y)=(1/k)\sum_{i=1}^{k} e^{-\lambda D(\vec{u}_i)}\left|(\vec{v}_i * \vec{O}(\vec{C}(\vec{u}_i)))\right| \quad (1)$$

La salida de esta función (*S(x,y)*) es un valor numérico comprendido entre 0 y 1. Cuanto más próximo a 1 se encuentre, mayor será el grado de ajuste, y cuanto más próximo a 0, el grado de ajuste será menor.

El significado de cada uno de los términos que componen la función es el siguiente (figura 7):

- *k*: Número de puntos de borde del modelo omega. En este caso 30.
- *i*: índice del sumatorio que representa a cada uno de los 30 puntos de borde del modelo omega, es decir, que el sumatorio está compuesto de 30 términos suma.
- $D(\vec{u}_i)$: Distancia entre el punto de borde *i* del modelo omega y el punto de borde de la imagen de entrada más cercano a *i*.
- λ : Parámetro que expresa el grado de sensibilidad de la función a variaciones de $D(\vec{u}_i)$. Cuanto más grande es lambda, más sensible es la función a cambios en $D(\vec{u}_i)$.
- $\vec{v}_i$: Vector gradiente unidad correspondiente el punto de borde *i* del modelo omega.
- $\vec{C}(\vec{u}_i)$: Punto de borde más cercano a *i*.
- $\vec{O}(\vec{C}(\vec{u}_i))$: Vector gradiente unidad correspondiente al punto de borde de la imagen de entrada más próximo al punto *i* del modelo omega.

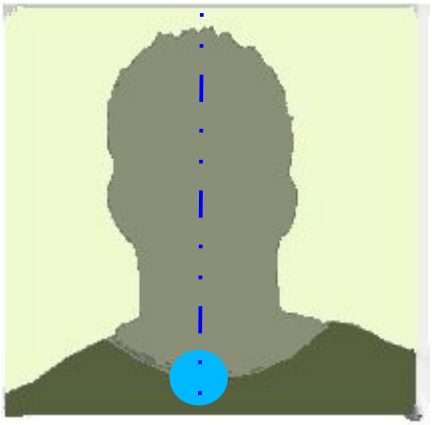


Figura 6. *Eje de simetría*

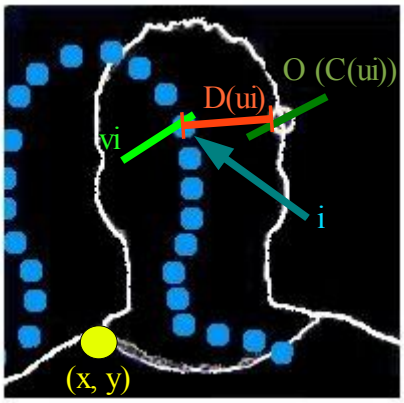


Figura 7. *Representación gráfica de los términos de S(x,y)*

La filosofía de esta función está basada en la medición del grado de ajuste de los dos elementos fundamentales del modelo omega:

- Los 30 puntos que lo componen.
- El vector gradiente unidad asociado a cada uno de esos puntos.

Así, para cada uno de esos 30 puntos del modelo omega se computa la expresión superior contenida dentro del sumatorio. Vamos a analizarla parte por parte:

- $e^{-\lambda D(\vec{u}_i)}$: mide el grado de ajuste del punto *i* del modelo con uno de los puntos de borde de la hipotética cabeza. De tal manera que el grado de ajuste será mayor cuanto más próximos estén los puntos, dicho de otro modo, cuanto menor sea la distancia entre ellos ($D(\vec{u}_i)$). El índice de esta exponencial siempre será negativo, por lo que el valor de la exponencial será mayor cuanto menor sea el valor de $D(\vec{u}_i)$, es decir, cuanto más próximos estén los puntos. λ es un valor constante.
- $\left|(\vec{v}_i * \vec{O}(\vec{C}(\vec{u}_i)))\right|$: valor absoluto del producto escalar del vector gradiente del modelo omega correspondiente al punto *i*, con el vector gradiente del punto de borde de la imagen más cercano a *i*. Mide el grado de ajuste de los dos gradientes, el ideal y el tomado de la imagen de entrada. El valor absoluto del producto escalar es mayor cuanto más parecidas son las direcciones de los vectores que intervienen en la operación, siendo máximo cuando esos vectores son coincidentes u opuestos, y mínimo cuando sean ortogonales. Por tanto, cuanto más parecidas sean las direcciones de los vectores gradientes, mayor valor tomará el producto escalar.

4.3 Pseudocódigo del algoritmo

En este apartado se describe el algoritmo asociado al procedimiento de detección de cabezas expuesto en el apartado anterior. En líneas generales se estructura según lo representado en la figura 8. En el Anexo V puede encontrarse un esquema mucho más detallado.

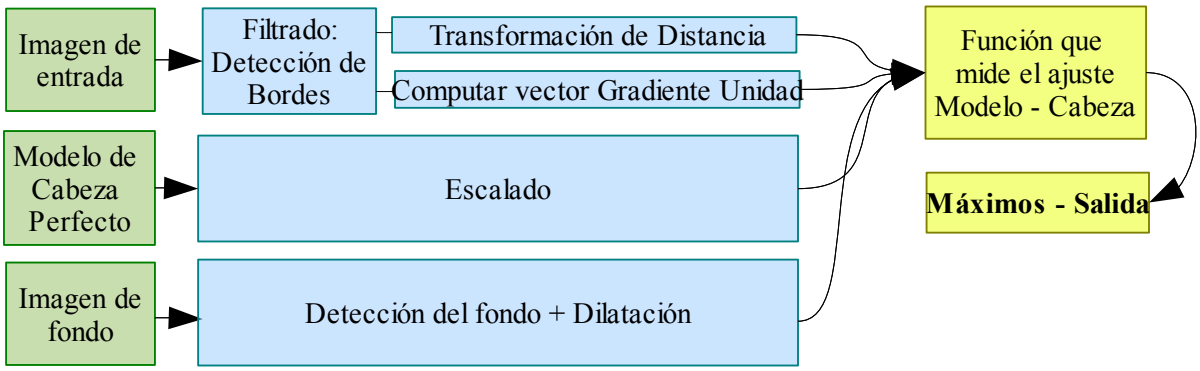


Figura 8. *Estructura general del algoritmo*

Así, en un principio, se inicializa el algoritmo, recibiendo como entradas la imagen sobre la que se va a realizar la detección de cabezas, la imagen de fondo, y el modelo de cabeza perfecto. Sobre esa primera imagen se aplica una detección de borde, que es la base para calcular el gradiente en cada punto del borde, y realizar una transformada de distancia para cada píxel de la imagen.

El modelo de cabeza ideal de entrada se escala para el píxel *(x,y)*, y junto con el gradiente, la transformada de distancia, y la imagen con el fondo detectado, se utilizan en la evaluación de la función *S(x,y)*, que posteriormente será suavizada para limitar el número de máximos locales. De esa función suavizada se extraerán los valores máximos, que se corresponden con las cabezas detectadas, siendo éstas la salida del algoritmo. Cada una de estas acciones se detallan a continuación.

Inicio del programa

El primer paso consiste en declarar e inicializar todas las variables y constantes que serán de utilidad a lo largo de las diferentes funciones que componen el programa.

- La estructura del algoritmo es la siguiente:

– Declaración e inicialización de: las variables para contener las imágenes, la función *S(x,y)*, los vectores para el cálculo del gradiente...

– Se cargan las imágenes que se utilizarán en el programa: imagen de entrada e imagen de fondo (figuras 9 y 10).

– Desde un fichero se obtienen los valores del modelo omega ideal (figura 11):

  – El valor de las coordenadas *(x,y)* de los 30 puntos del modelo.

  – El valor de las componentes *(x,y)* del gradiente vertical y horizontal de cada uno de los 30

puntos del modelo.

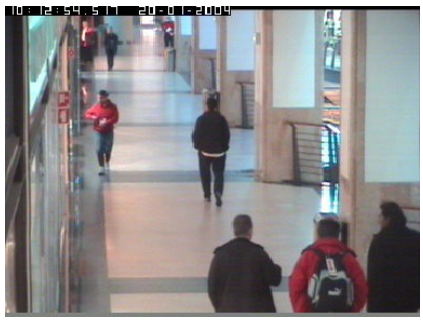


***Figura 9**. Imagen de entrada*

***Figura 10.** Imagen de fondo*

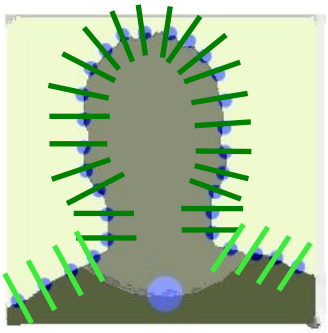


***Figura 11.** Modelo omega*

Filtrado

Mediante la aplicación del filtrado se persigue un doble objetivo: obtener una imagen binaria de salida donde se detecten los bordes presentes en la imagen de entrada, y el gradiente asociado a cada uno de esos puntos de borde.

- La estructura del algoritmo es la siguiente:

Para realizar el filtrado se utiliza la función Canny de Opencv que internamente calcula el gradiente. Para que esta información se pueda extraer al exterior de la función, previamente se realiza un acondicionamiento de las variables que contendrán el gradiente vertical y horizontal, de tal manera que se adapten al modo en que el filtro de Canny calcula ese gradiente.

A continuación, tomando la imagen de entrada (figura 9):

- Se realizará un filtrado de Canny, obteniendo como resultado una imagen binaria con la detección de bordes (figura 12).
- Se tomará el gradiente en cada punto del borde (figura 13), utilizando para ello el gradiente calculado por el propio filtro de Canny durante la realización del filtrado.
- Posteriormente los gradientes vertical y horizontal serán normalizados, de tal manera que el vector gradiente resultado sea unitario.

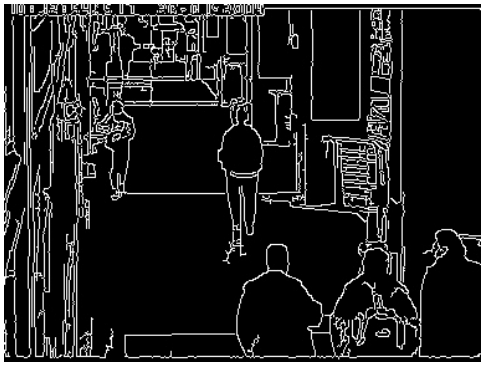


Figura 12. *Detección de borde sobre la imagen de entrada*

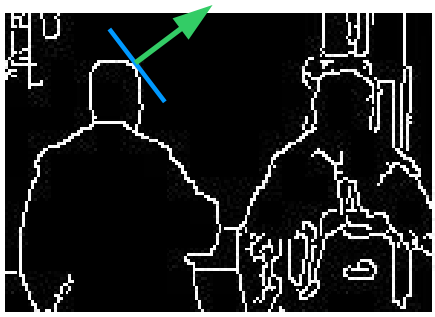


Figura 13. *Vector gradiente en uno de los puntos de borde*

Transformada de distancia

Al realizar esta operación, para cada uno de los píxeles de la imagen de entrada, se obtiene la distancia al punto de borde más cercano de esa imagen. También se obtiene el valor de las coordenadas de ese punto de borde.

- La estructura del algoritmo es la siguiente:

Para cada uno de los píxeles de la imagen que no pertenezcan al borde detectado con Canny, se realizará lo siguiente hasta que se detecte un punto de borde:

- Se recorrerá una corona cuadrada que rodea a ese píxel (representado en color amarillo en la figura 14) en busca de un punto de borde, partiendo desde la corona más cercana (compuesta por 8 píxeles) y aumentando progresivamente el tamaño de la corona (24, 48...), hasta que se detecte como mínimo un píxel de esa corona que sea borde. Para ello se realizará lo siguiente:
  - Tomar las coordenadas extremas en el eje vertical y recorrerlas para todos los valores de la coordenada horizontal.
  - Tomar las coordenadas extremas del eje horizontal y recorrerlas para todos los valores de la coordenada vertical.
  - Únicamente recorrer aquellos puntos que se encuentren dentro de la imagen.
- En caso de que en una misma corona se detecten varios píxeles de borde (figura 15), se guardarán las coordenadas de todos ellos, y se calculará la distancia entre el píxel central (del que deseamos obtener la transformada de distancia) y cada uno de esos puntos de borde guardados (figura 15).
- Finalmente se tomará la distancia mínima y las coordenadas del punto de borde que hacen esa distancia mínima (figura 16). La distancia mínima y las coordenadas constituyen la transformada de distancia del píxel central.

Para los puntos de borde detectados en Canny, se tomará como distancia mínima el valor cero, y como punto de borde más cercano, él mismo.

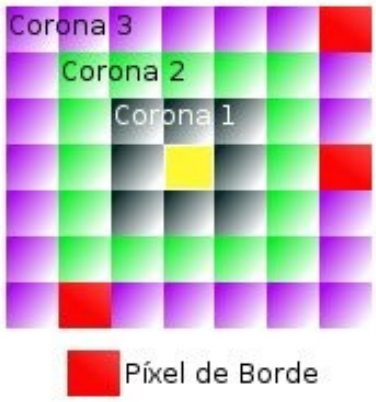


Figura 14. *Representación del Píxel central (amarillo), corona 1 (gris), corona 2 (verde), corona 3 (morado) y tres píxeles de borde (rojo)*

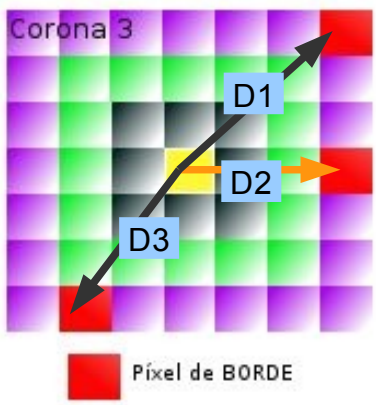


Figura 15. *Distancia del píxel central a cada uno de los píxeles de borde*

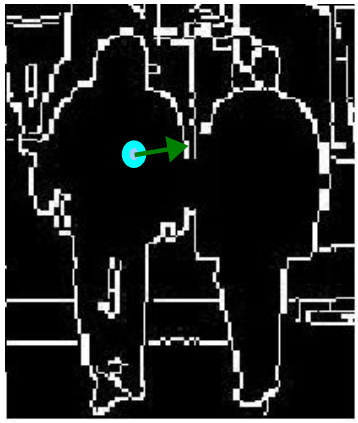


Figura 16. *Distancia desde un píxel de la imagen (azul) al punto de borde más cercano*

Detección del fondo

Sobre la imagen de entrada (figura 9) se realiza la sustracción de la imagen de fondo (figura 10), y una umbralización posterior de la imagen resultado. De esta forma se obtiene una imagen binaria donde se diferencian los píxeles que pertenecen al fondo de los que no.

Esta imagen se somete posteriormente a un proceso de dilatación. El resultado se muestra en la figura 18.

Función de ajuste *S(x,y)*

Hasta ahora se han realizado varias operaciones, cálculo de gradientes, transformadas de distancia... Todas ellas se utilizarán para el cálculo de la función de ajuste S(x,y), que ya fue descrita en el apartado 4.2

- La estructura del algoritmo es la siguiente:

En primer lugar se desplaza el modelo omega genérico al punto *(x,y)* donde se evaluará, para ello debe escalarse y trasladarse. Así los pasos a realizar son los siguientes:

– En primer lugar se escala el modelo omega a la posición *(x,y)*, aplicando la siguiente expresión: 2.377e-04*y*y + 3.249e-02 + 6.287, obtenida del ajuste experimental del tamaño del modelo en función de *y*. El origen de coordenadas en OpenCv es el borde superior izquierdo, con lo que cuando menor sea la coordenada vertical, el modelo omega tendrá un tamaño más pequeño (figura 17).

– Posteriormente, se centra el modelo omega escalado en el punto *(x,y)*, para ello se suma a las componentes horizontales de ese modelo la coordenada *x*, y a las componentes verticales la

coordenada *y*.

– Ya tenemos el nuevo modelo escalado y centrado en *(x,y)*, compuesto entre otros, por 30 puntos que definen el borde de la cabeza. El siguiente paso consiste en aceptar o rechazar cada uno de esos 30 puntos, para ello:

  – Se comprueba que cada punto se encuentre dentro de los límites de la imagen

  – Se comprueba que cada punto se encuentre dentro del *foreground*.

  – En caso de cumplir estos dos requisitos, el punto se toma como válido y pasa a computarse, en caso contrario, no (figura 19) . De tal manera que puede darse el caso de que existan modelos omega con menos de 30 puntos.

Figura 17. *Modelo omega escalado para tres posiciones distintas*

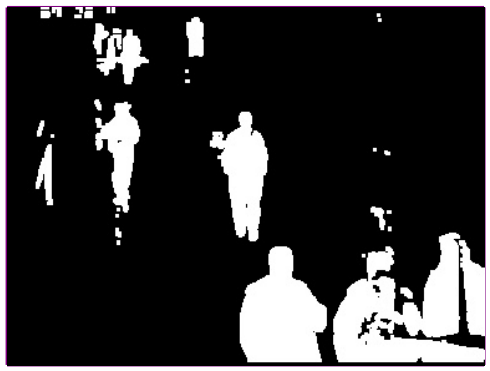
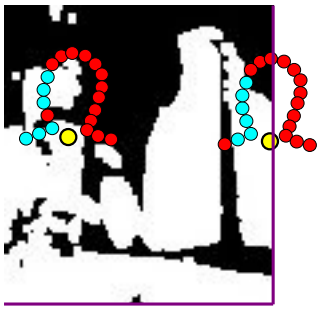


Figura 18. *Imagen de detección del fondo: foreground (blanco) y background (negro)*



Amarillo: *Centroide del modelo*
Azul: *Puntos del modelo válidos*
Rojo: *Puntos del modelo no válidos*

Figura 19. *Modelo omega sobre imagen de detección del fondo*

Una vez ya se tiene el nuevo modelo validado es posible comenzar a computar la función *S(x,y)*, que recordemos tenía la siguiente expresión:

$$S(x,y)=(1/k)\sum_{i=1}^{k} e^{-\lambda D(\vec{u}_i)}\left|(\vec{v}_i * \vec{O}(\vec{C}(\vec{u}_i)))\right| \quad (1)$$

– En primer lugar se computarán cada uno de los términos (30 como máximo) de los que está compuesto el sumatorio. Para ello se realizan las siguientes operaciones:

  – $e^{-\lambda D(\vec{u}_i)}$:Para calcular la exponencial se tomará la distancia entre el punto *i* del nuevo modelo omega y el punto de borde más cercano que ya fue calculado cuando se realizó la transformada de distancia. Está representado en la figura 20.

  – $\left|(\vec{v}_i * \vec{O}(\vec{C}(\vec{u}_i)))\right|$:Para realizar este producto escalar se tomará el vector gradiente correspondiente al punto *i* del modelo omega (es siempre el mismo, independientemente del escalado y de la traslación del modelo en *(x,y)*), y el vector gradiente del punto de borde más cercano al punto *i* del modelo omega (calculado en la función de Canny) (figura 21).

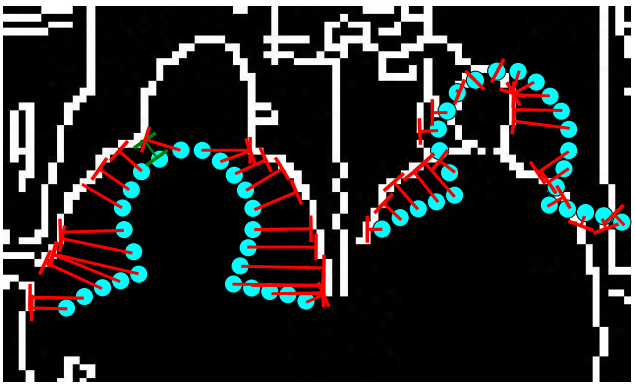


Figura 20: *Distancia de cada punto de borde del modelo centrado en (x,y), al punto de borde más cercano de la imagen*

Figura 21. *Gradiente del modelo ideal (amarillo) y gradiente calculado (rojo)*

Suavización de la función de ajuste *S(x,y)*

A continuación, se procederá a suavizar la función S(x,y). Para ello, se realizará la media aritmética del valor de S(x,y) en el punto (x,y), con el valor de S(x,y) en sus puntos vecinos, pudiendo tomar los 8 píxeles vecinos (la corona cuadrada más próxima a (x,y)), o los 24 píxeles vecinos (las dos coronas cuadradas más próximas a (x,y))

Búsqueda de máximos en la función

Una vez se tiene la nueva función S(x,y) suavizada, comienza la búsqueda de los valores de (x,y) que hacen esa función máxima, y que en principio se corresponderán con aquellos píxeles de la imagen que se asemejan más a la forma de una cabeza.

- La estructura del algoritmo es la siguiente:

Se compara el valor de *S(x,y)* en el punto de *foreground (x,y)*, con el valor que toma esa función en los 8 puntos vecinos, y en caso de ser el mayor de todos ellos, el punto *(x,y)* será tomado como máximo local.

Sin embargo, para que un máximo local sea considerado como cabeza, el valor de *S(x,y)* deberá superar un determinado umbral, y haber sido calculado considerando un número mínimo de puntos del modelo omega como válidos (figura 19). Es decir, que los máximos locales están sujetos a una serie de parámetros descritos con más extensión en el apartado 5.1.

Aquellos máximos considerados como cabezas, se dibujarán sobre la imagen de entrada para su visualización, y se almacenarán en un fichero de texto, constituyendo ambos la salida del algoritmo.

4.4 Lenguaje de programación

A la hora de seleccionar un lenguaje de programación para implementar un algoritmo de visión por computador, existen fundamentalmente **dos alternativas** bien diferenciadas: utilizar un **lenguaje de cálculo numérico** (como Matlab, Octave, Scipy...) o un **lenguaje de bajo nivel** (C, C++, Fortran....). En el Anexo VI se realiza una comparativa de las ventajas e inconvenientes de ambas opciones.

En este caso **se ha seleccionado el lenguaje C**, porque su **velocidad de ejecución** es mucho mayor que la de los lenguajes de cálculo numérico, y uno de los objetivos de este proyecto es conseguir una ejecución en tiempo real. Por el contrario, programar algoritmos que operen con imágenes en C, tiene un **grado de dificultad elevado**; nuestra experiencia demuestra que la extensión de un programa en C es de 2 a 3 veces superior a la extensión de ese mismo programa en un lenguaje de cálculo numérico. Para tratar las imágenes se utilizará la **librería Opencv** [Igual, 08], [Bradski, 08], que aunque sigue siendo compleja, facilita en gran medida la realización de algunas operaciones básicas como filtrados, transformaciones afines...

En el Anexo VII se incluye una **descripción de los recursos de programación** más empleados durante la realización de este proyecto.

5. Análisis de resultados del algoritmo

En este apartado se van a analizar los resultados obtenidos al ejecutar el algoritmo: el grado de efectividad en la detección de cabezas y la velocidad con que se realiza esa detección. Aunque previamente resulta imprescindible conocer que factores o parámetros van a influir en el resultado final.

5.1 Parámetros que intervienen en el algoritmo

Durante la realización de las operaciones de detección de personas intervienen una serie de acciones intermedias como detecciones de borde, suavización de resultados, umbralizaciones... Estas operaciones están condicionadas por una serie de parámetros que, dependiendo del valor que tomen, modifican los resultados proporcionados por el algoritmo.

Uno de los retos de este proyecto consiste en **encontrar el equilibrio justo de esos parámetros** para obtener los mejores resultados posibles. Los parámetros implicados en este proceso son los siguientes:

- Umbral: Determina el valor mínimo que debe tomar la función *S(x,y)*, para que el punto *(x,y)* sea considerado como candidato a centroide de una cabeza. Cuanto mayor sea el valor del parámetro umbral, menor número de puntos candidatos a cabeza obtendremos. Aunque el número de errores cometido (falsos positivos) también será menor. Si disminuye el valor del umbral, se obtienen muchos más puntos candidatos a cabeza, y también aumenta el número de errores cometidos.
- Corona: Indica el número de píxeles que se tomarán, alrededor de un píxel central *(x,y)*, para suavizar la función *S(x,y)*. Este parámetro puede tomar los valores 8 o 24. En el primer caso, se suaviza la función tomando los 8 píxeles vecinos, que se corresponden a la corona más cercana al píxel *(x,y)*. En el segundo se toman 24 píxeles, es decir, las dos coronas más cercanas que rodean al píxel central.
- Lambda: Parámetro implicado en el cálculo de *S(x,y)*. Valores típicos de lambda son 0.05, 0.1, 0.15 ó 0.2. (Explicado con más detalle en el apartado 4.2)
- Ditate1 y Dilate2: Tras la detección del *background*, se realiza una dilatación. Estos parámetros indican las dimensiones del elemento estructural que se utiliza. Toman unos valores típicos de 3, 5 ó 7.
- Nº puntos del modelo: Este parámetro determina el número mínimo de puntos de borde del modelo omega centrado en *(x,y)* que se deben evaluar, para que el punto de la imagen *(x,y)* pueda ser considerado como candidato a centroide de una cabeza (recordemos que sólo se evalúan aquellos puntos que están dentro del *foreground*).

5.2 Configuración de los parámetros

Como ya se comentó en el apartado anterior, el cálculo de los valores óptimos de los parámetros resulta de vital importancia para que la salida del algoritmo proporcione unos resultados aceptables.

Se **han realizado varias simulaciones**, combinando diferentes valores de esos parámetros, para tratar de encontrar la configuración óptima.

En función de esos resultados, es posible clasificar los **parámetros** en dos grupos:

- Aquellos que **sólo obtienen un resultado aceptable si toman un valor determinado**. Dentro de este grupo se encuentran los siguientes parámetros:
  - Nº de puntos del modelo: El valor para el que siempre se obtienen los mejores resultados es 20. Por encima de ese valor, bastantes cabezas se quedan por detectar, y si el valor es inferior aparecen muchos falsos positivos.
  - Dilate 1 y Dilate 2: La detección de los bordes es óptima cuando el elemento empleado para la dilatación es un cuadrado de dimensión 3x3.
  - Umbral: Fijado en principio a 0.6. El parámetro umbral es un parámetro especial, puesto que como veremos en el siguiente apartado, a la hora de representar la salida, se va a realizar un barrido de los diferentes valores que puede adoptar la variable umbral, entre 0.6 y 0.9. Anotando el número de detecciones realizadas para cada uno de esos valores intermedios.
- Aquellos que **presentan un buen comportamiento tomando diferentes valores**:
  - Lambda: 0.1 y 0.2
  - Corona: 8 y 24

Estos parámetros constituirán las entradas que los usuarios configurarán inicialmente en el fichero que se pasa al algoritmo, como se expone en el siguiente apartado.

5.3 Entrada y salida

Las entradas del algoritmo se configurarán a través de un fichero de texto con los valores de todos los parámetros: lambda, corona, dilate1, dilate2, umbral y nº puntos del modelo. Además también se incluirá como entrada un código numérico que identifica a la imagen sobre la que se va a realizar la detección.

En lo referente a las salidas del algoritmo, se informará de todas las cabezas detectadas a través de un doble vía:

- Por un lado se mostrará por pantalla una imagen de salida, similar a la imagen de entrada, donde se remarcarán todos aquellos puntos que el algoritmo identifica como potenciales centroides de cabezas (figura 22). Esta es la salida de tipo visual.
- Además, se generará un fichero de texto (figura 23) donde se imprimirán los resultados numéricos obtenidos al ejecutar el algoritmo. Este fichero de texto estará compuesto por tres columnas.

- La primera columna contiene el valor de *S(x,y)*, en el punto *(x,y)* detectado como centroide de una cabeza.
- La segunda columna representa el valor de la coordenada *y* para la que se ha detectado el centroide.
- La tercera columna contiene el valor de la escala que se ha aplicado al modelo de cabeza ideal, para realizar el cálculo de *S(x,y)*.
- Es conveniente que el propio usuario genere una cuarta columna, donde indique si el máximo detectado es efectivamente una cabeza, es decir, el punto es un verdadero positivo, o por el contrario el máximo detectado no se corresponde con una cabeza, es un falso positivo.

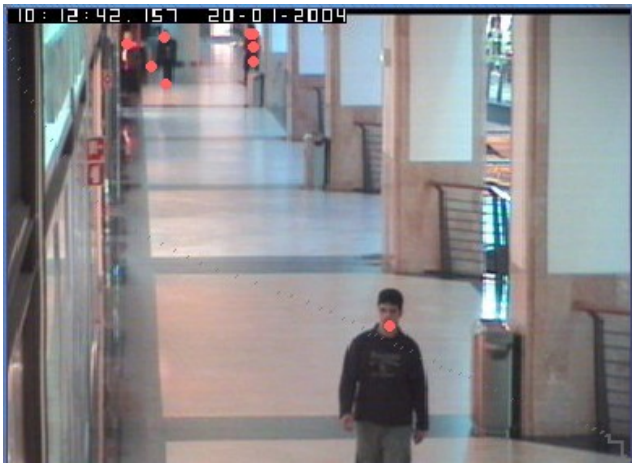


Figura 22. *Representación gráfica de la salida del algoritmo*

```
0,777084 21 7,074033
0,780601 28 7,383027
0,608304 34 7,666426
0,731002 84 10,693891
0,640448 49 8,449828
0,760835 32 7,570058
0,703292 37 7,814546
0,746772 40 7,966946
0,712012 58 8,971230
0,626089 37 7,814546
0,666082 43 8,123627
```

Figura 23. *Aspecto del fichero de salida*

La impresión de los datos en un fichero de texto resulta muy útil para tratarlos con posterioridad. No hay que olvidar que el programa genera una determinada cantidad de máximos para cada una de las simulaciones, y que se pueden realizar tantas simulaciones como combinaciones posibles de los parámetros de entrada existan.

Con lo que la cantidad de datos generados a la salida es muy abultada, y se hace necesario tratarlos para poder representarlos convenientemente y extraer conclusiones que sean útiles para el desarrollo del algoritmo.

5.4 Análisis del funcionamiento del algoritmo

Para analizar correctamente los resultados, es necesario representarlos previamente, de manera que resulten fáciles de interpretar. En esta ocasión los resultados se van a representar a través de **la curva ROC** [Wu, 05], que nos permitirá deducir cual es la configuración de los parámetros de entrada que mejores valores de salida aporta. En cada uno de los ejes de la gráfica se representará un indicador distinto:

- En el **eje vertical** de la gráfica se representarán el número de **verdaderos positivos (TP)**, puntos que realmente son cabezas, y son detectadas correctamente.
- En el eje horizontal el número de **falsos positivos (FP)**, puntos que no son cabeza, y son detectados como tales.

Se representarán **tantas curvas ROC como simulaciones** se realicen, en este caso 4. Cada una de las simulaciones tiene una configuración de parámetros distinta, la elección de estos parámetros se ha realizado según lo dispuesto en el apartado anterior.

	Dilate1	Dilate2	Nº puntos	Corona	Lambda
Simulación 1	3	3	20	8	0.1
Simulación 2	3	3	20	8	0.2
Simulación 3	3	3	20	24	0.1
Simulación 4	3	3	20	24	0.2

Tabla 3. *Configuración de las simulaciones*

Para cada simulación se han evaluado cinco imágenes de entrada: Imagen_1, Imagen_2, Imagen_3, Imagen_4, Imagen_5 (localizables en [CAVIAR,10]). Obteniéndose los datos representados en la figura 24. Cada uno de los puntos de esa gráfica expresa el número de TP y FP, para un determinado valor del parámetro umbral, tomando éste siete valores discretos distintos [0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9].

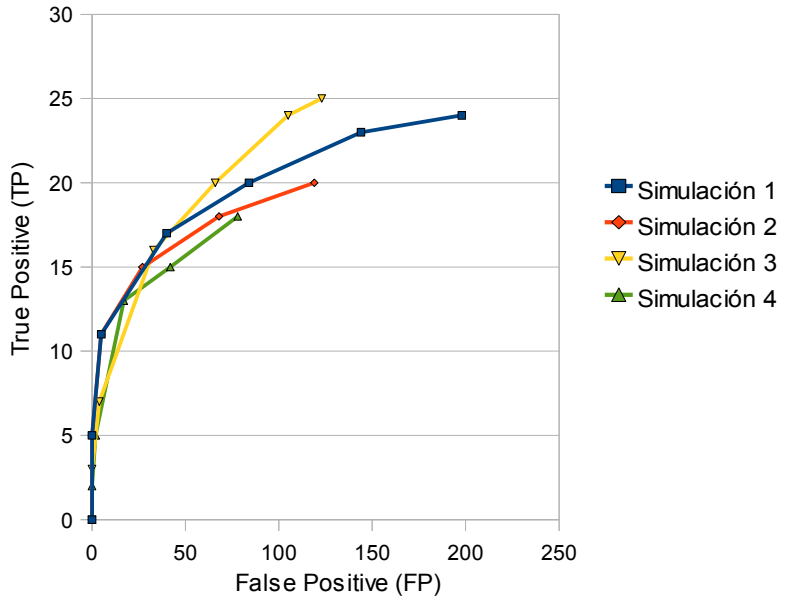


Figura 24. *Representación de la curva ROC para cada una de las simulaciones*

Así, a la vista de la gráfica podemos extraer una serie de conclusiones:

- La gráfica ideal es aquella que tiene una pendiente de subida inicial con valor infinito (línea paralela al eje *y*), y una vez está arriba la pendiente es cero (línea paralela al eje *x*).
- Las simulaciones representadas en la figura 24, aunque lejos de ser ideales, sí siguen esta tendencia.
- Existe un total de 32 cabezas entre todas las imágenes analizadas, aunque algunas de ellas no han sido detectadas por encontrarse excesivamente lejos de la cámara.
- Observamos que los parámetros de la **simulación 3 son los mejores**, porque permiten obtener el mayor número de verdaderos positivos, con el menor número de falsos positivos, estando claramente por encima del resto de simulaciones.

5.5 Análisis de tiempos

El lenguaje de programación C se seleccionó para implementar el algoritmo porque permite desarrollar altas velocidades de ejecución. Por este motivo se ha realizado un análisis de tiempos utilizando la **herramienta que nos proporciona el entorno de programación** (descrita en el Anexo VIII).

En la figura 25 se representa el número de frames por segundo que es capaz de computar el programa. El análisis se ha realizado para cada una de las cinco imágenes de prueba. Para cada imagen se han realizado cuatro simulaciones, correspondientes a las cuatro configuraciones de los parámetros de entrada expuestas en la tabla 3. En la figura 25 **se representa el valor medio (frames/sg) de esas cuatro simulaciones para cada una de las imágenes.**

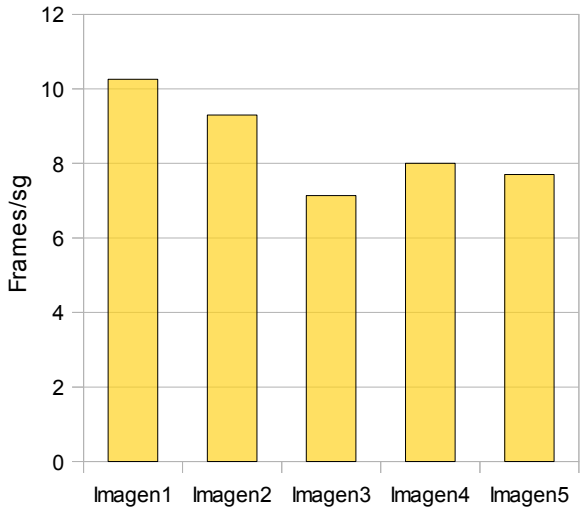


Figura 25. *Velocidad de ejecución para cada una de las imágenes*

A la vista de estos resultados, es posible extraer una serie de conclusiones:

- Los algoritmos se encuentran **muy próximos a una ejecución en tiempo real** (fijada en torno a 10 frames/sg), con lo que desde el punto de vista temporal, el resultado es muy positivo.
- La velocidad de ejecución de las dos primeras imágenes es mayor. Esto se debe a que el número de personas presentes en esas imágenes es menor, y por tanto, el volumen de cálculos también. Por contra, las tres últimas imágenes son más complejas, contienen a más personas, y su evaluación requiere un mayor gasto temporal.

En el Anexo IX se incluye un análisis de tiempos más detallado, donde se desglosa la **velocidad de ejecución de cada una de las funciones que componen el algoritmo**.

6. Análisis de la gestión del proyecto

En este apartado se va a analizar el ajuste a la planificación temporal, los costes y recursos empleados, a quién pertenecen los resultados y cómo se van a proteger y a explotar. También se valorará el proceso de desarrollo del algoritmo a través del PSP, así como el propio algoritmo mediante la aplicación del modelo ISO 9126.

6.1 Análisis de la planificación temporal

En la siguiente tabla se muestra la planificación para las diferentes fases, contrastando las fechas de inicio y de fin, previstas y reales. Remarcar, que la dedicación durante las fechas expuestas no ha sido exclusiva, y que la carga horaria de cada una de las fases se corresponde con lo reflejado en el apartado 2.3, desglosado en el Anexo II.

Seguimiento de la Planificación Temporal					
Fases	**Fecha inicio prevista**	**Fecha inicio real**	**Fecha fin prevista**	**Fecha fin real**	**Desviación fecha fin**
Planificación	01/12/09	01/12/09	01/03/10	14/03/10	14 días
Organización	01/03/10	14/03/10	01/04/10	08/04/10	8 días
Ejecución	01/04/10	08/04/10	05/04/10	10/08/10	36 días
Implementación del algoritmo	20/04/10	03/05/10	10/05/10	29/05/10	19 días
Depuración del algoritmo	10/05/10	29/05/10	30/05/10	26/06/10	26 días
Control de la implementación	30/05/10	26/06/10	10/06/10	30/06/10	20 días
Optimización del algoritmo	10/06/10	30/06/10	30/06/10	02/08/10	33 días
Control de la optimización	30/06/10	02/08/10	05/07/10	06/08/10	32 días
Análisis de resultados, presentación y cierre	05/07/10	08/08/10	31/07/10	30/08/10	30 días
PROYECTO TOTAL	01/12/09	01/12/09	31/07/10	30/08/10	30 días

Tabla 4. *Seguimiento de la planificación temporal*

A la vista de los resultados podemos extraer una serie de conclusiones:

- La planificación, en líneas generales, se ha **ajustado a la realidad**, aunque con una **desviación de entorno a un mes** entre lo planificado y lo realizado.
- Aquellas fases que abarcan los procesos de programación, depuración y optimización son mucho más difíciles de planificar, puesto que dependen enormemente de los propios resultados que

ofrezca el algoritmo. En ellas se han dado las mayores desviaciones.

- La duración de los procesos de control ha sido menor de la prevista.
- Una buena planificación del proyecto puede **facilitar el desarrollo del mismo**, ya que marca objetivos concretos a realizar en un periodo temporal definido, con lo que se favorece el trabajo continuado.
- La **experiencia adquirida servirá para ajustar mejor** la planificación temporal en futuros proyectos.

6.2 Costes y recursos empleados

Al ser un proyecto software no se requiere de equipos demasiado específicos. Los recursos necesarios para llevar a cabo el proyecto son los siguientes (precios en euros):

Recurso	**Cantidad**	**Precio unidad**	**Precio total**
Ordenador de Sobremesa	1	550	550
Pantalla Plana para Ordenador	1	130	130
Teclado	1	20	20
Ratón	1	7	7
Norma UNE 166001	1	14,06	14,06
Norma ISO 9126 - 1	1	29,33	29,33
Norma ISO 9126 - 2	1	29,33	29,33
Norma ISO 9126 - 3	1	29,33	29,33
Norma ISO 9126 - 4	1	29,33	29,33
Material Fungible	1	400	400
Gastos de personal: Ingeniero Técnico de Telecomunicaciones			6000
Total			7238,38
IVA(18,00%)			1302,91
TOTAL			**8541,29**

Tabla 5. *Presupuesto*

Si también se realizara la adquisición de las imágenes, debería presupuestarse una cámara. Por ejemplo, una cámara IP típica tiene un coste aproximado de 400 euros.

6.3 Propiedad de los resultados

Los resultados de este proyecto de I+D+i pertenecen a los grupos de investigación de la Universidad de Zaragoza involucrados en su desarrollo. Estos grupos son:

- **EduQTech** (Education Quality Technology): Grupo interuniversitario de I+D+i cuyo principal objetivo es la aplicación de los conceptos de calidad a las actividades de investigación, desarrollo e innovación y a la docencia.
- **CvLab** (Computer Vision Laboratory): CV Lab es un grupo de investigación del I3A cuyo trabajo se centra en la investigación de técnicas computacionales para visión artificial en los campos de videovigilancia, biometría y procesado de imágenes médicas.

Como se ha comentado, estos grupos pertenecen a la **Universidad de Zaragoza**, por lo que la propiedad intelectual de este proyecto pertenece a la propia universidad.

Figura 26. *Logotipos de EduQTech y CvLab*

6.4 Protección de los resultados

La protección de la propiedad de los resultados se realizará mediante una licencia para software libre. La licencia a utilizar todavía no se ha concretado, y se seleccionará en función de los resultados obtenidos al concluir el proyecto, aunque actualmente se barajan las siguientes posibilidades:

- Para el software:
  - Licencia Pública General de GNU (GPL): http://www.gnu.org/licenses/licenses.es.html
  - Common Public Licence (CPL): http://www.opensource.org/licenses/cpl1.0.php
- Para el contenido:
  - GNU Free Documentation License (GFDL): http://www.gnu.org/licenses/fdl.html
  - Creative Commos (CC): http://es.creativecommons.org/

Como se puede observar, la licencia para el software es distinta de la licencia para el resto de la documentación.

6.5 Plan de explotación

Como plan de explotación propiamente dicho se prevee que este proyecto se utilice para:

– Será la base de un futura tesis doctoral codirigida entre los dos grupos anteriormente citados. Concretamente esta tesis versará sobre el seguimiento de varias personas presentes en una misma imagen. Este seguimiento se realizará en tiempo real, a partir de la información visual de las personas, proporcionada por una cámara.

6.6 Análisis del Software

6.6.1 Análisis del proceso de programación

Como ya se enunció en el apartado 2.4.1, el desarrollo del software se rige por el Proceso de Software Personal (PSP).

En el Anexo X puede encontrarse una clasificación (siguiendo lo dispuesto en el apartado 2.4.1) de los **principales errores cometidos**. En este punto, vamos a analizar los resultados de ese proceso para los diferentes tipos de errores a través de la figura 27. Para cada **tipo de error** se representa el **número de veces que acaeció** en relación con el número total de errores (expresado en tanto por ciento), así como el de **tiempo empleado en su resolución** respecto del tiempo total empleado para solventar todos los errores (expresado en tanto por ciento).

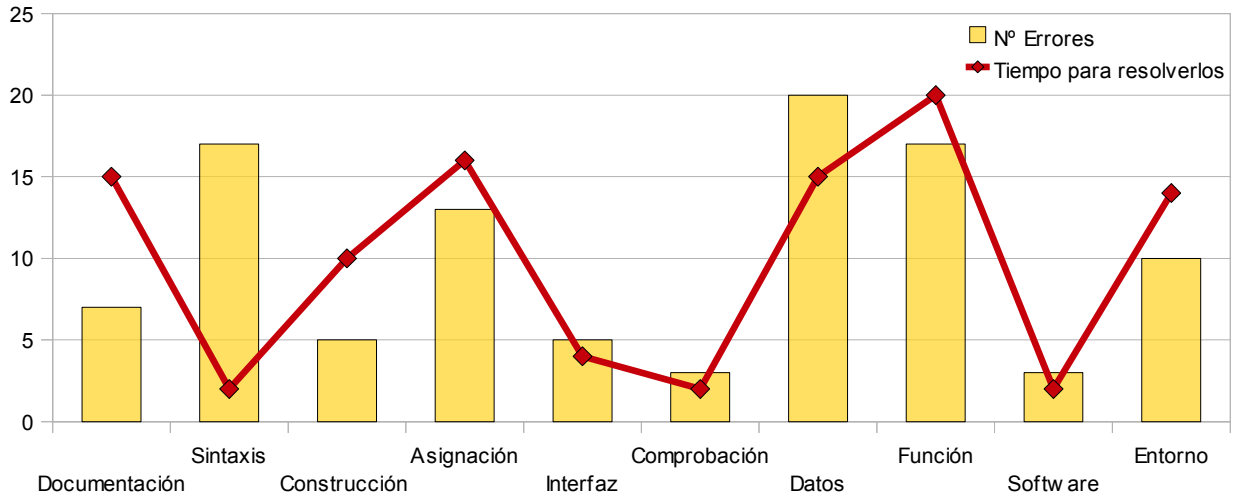


Figura 27. *Análisis de los tipos de errores*

A la vista de la gráfica podemos extraer una serie de conclusiones:

– Los errores más frecuentes son los de tipo sintáctico y los relativos al tratamiento de datos o funciones, mientras que los errores de construcción, de comprobación, o de software son poco frecuentes.

– No existe una relación directa entre el número de errores de un determinado tipo y el tiempo empleado para solventarlos. Así, los errores de documentación, de construcción, y entorno no son muy numerosos, sin embargo su resolución lleva aparejado un gasto temporal elevado. Por contra,

los errores de tipo sintáctico, que son abundantes, se solventan en un periodo de tiempo corto.

También resultaría interesante conocer la **relación entre el tiempo medio empleado para resolver un error y la fase en la que fue cometido**. En la figura 28 se representa esa relación para las fases de diseño y codificación (por ser las más características).

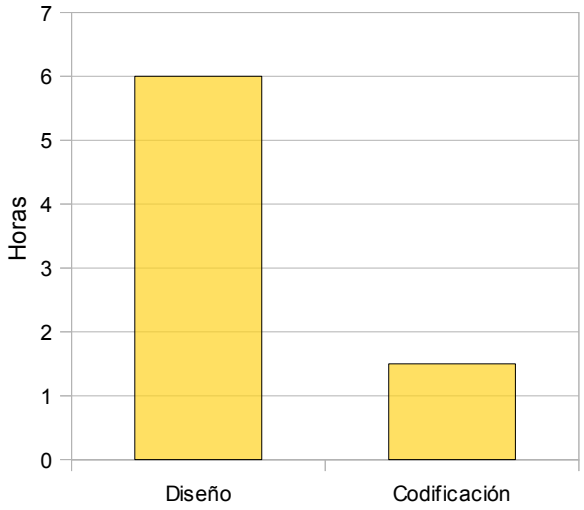


Figura 28. *Relación entre la fase en la que se introdujo un error y el tiempo medio empleado en resolverlo*

Así, a la vista de esa tabla resumen, se pueden extraer las siguientes conclusiones:

– Los **errores introducidos en la fase de diseño** del algoritmo llevan asociado un tiempo de corrección elevadísimo, por lo que podemos concluir que tienen una **gran trascendencia**. Este tipo de errores afectan a la propia estructura del programa, con lo que su corrección pasa por modificarla, incrementando en gran medida el tiempo empleado. Por contra, los errores cometidos en la fase de codificación suelen ser de tipo sintáctico, de asignación... en definitiva, son errores más fáciles de solventar.

Durante el desarrollo del proyecto se han anotado diversos tiempos y tasas de errores que nos van a permitir **medir el grado en que nuestro proceso de programación es correcto**, a través de los indicadores de la tabla 6, que ya fueron expuestos en el apartado 2.4.1. El análisis de los indicadores se va a realizar para dos momentos temporales distintos: el primero de ellos se corresponde con el inicio de la programación (PSP_v1), y el segundo con la fase final del proceso (PSP_v2).

	Minutos / LOC	Errores / KLOC	V/F	Rendimiento
PSP_v1	1,8	0,3	0,9	30,00%
PSP_v2	1,2	0,21	1,8	60,00%

Tabla 6. *Resumen de los indicadores del PSP*

Podemos considerar como valores aceptables de estos indicadores para programadores profesionales, los siguientes: V/F superior a 2, y el rendimiento superior al 50% [Humphrey, 01].

Como puede observarse, al inicio del proyecto, el valor de estos indicadores era deficiente, por lo que se **tomaron acciones encaminadas a mejorar el proceso de programación**: dedicando más tiempo a la revisión del código, realizando un diseño más detallado de los algoritmos previo a la codificación... Como resultado de ese plan de actuación, se consiguió mejorar la eficiencia de la programación, de tal manera que en la última versión del algoritmo desarrollada (correspondiente al PSP_v2), se ha conseguido mejorar la forma de trabajar, si bien todavía no se han alcanzado unos niveles profesionales.

6.6.2 Análisis del producto software

Como ya se comentó en el apartado 2.4.2, el producto software resultado de este proyecto se **evaluará siguiendo el modelo de la norma ISO 9126** sobre calidad de software.

Así, se ha adaptado el modelo de calidad ISO 9126 para aplicarlo a software de visión por computador, tal y como que puede consultarse en el Anexo IV. La **aplicación de este modelo al algoritmo de detección de cabezas** puede consultarse en el Anexo XI.

Tomando como referencia esa aplicación, se ha realizado una evaluación final del producto software resultante, siguiendo lo dispuesto en el Anexo XII (gestión de la evaluación).

Cada una de las características que contempla el producto software: funcionalidad, eficiencia.., ha sido calificada con una puntuación entre 0 y 1, en función de la calificación aportada por los indicadores de los que se compone cada una de ellas, y que están recogidos en el citado Anexo XI. Además a cada una de las características del producto se le asigna un "peso", que indica **la relevancia de esa característica** concreta. En este caso los pesos vienen expresados en tanto por ciento.

Los resultados de la evaluación son los siguientes:

Evaluación del Modelo de Calidad Interna - Externa						
	Funcionalidad	Fiabilidad	Usabilidad	Eficiencia	Mantenibilidad	Portabilidad
Valoración	0.78	1	0.94	1	0.49	0.67
Peso (%)	30,00%	15,00%	15,00%	20,00%	10,00%	10,00%

Tabla 7.*Valoración y ponderación de las características del modelo ISO 9126*

A la vista de esto podemos concluir que:

- La evaluación de las características funcionalidad, portabilidad y mantenibilidad es aceptable aunque podría mejorarse. Analizando más en detalle cada una de ellas podemos deducir los aspectos que son mejorables:
  - Funcionalidad: La evaluación de esta característica resulta muy costosa de mejorar, puesto que está relacionada con la detección correcta de cabezas. Esta cuestión no está resuelta, y a corto

plazo no se esperan resultados en este campo, puesto que reviste una alta complejidad. El problema estriba en que el algoritmo de detección toma como cabezas correctas una cantidad significativa de puntos que no lo son.

  - Portabilidad: La característica portabilidad adquiere mayor importancia cuanto más definitiva es la versión. Recordemos que el software evaluado es un prototipo y no una versión final, con lo que la portabilidad no es una característica muy relevante para el grado de desarrollo actual del producto.
  - Mantenibilidad: El mantenimiento del algoritmo se llevará a cabo por los desarrolladores del mismo, con lo que las exigencias de mantenibilidad no son tan elevadas como si se realizara por programadores externos. Aún así, sería conveniente introducir mejoras en este aspecto que favorezcan la claridad en la programación y la auto-interpretación del código.

- Las características de fiabilidad, usabilidad y eficiencia obtienen una buena valoración, lo cual es un signo muy positivo puesto que constata que la programación del algoritmo es fiable y se ha documentado de manera correcta.
- La característica funcionalidad es la más relevante de todas, ya que el principal objetivo del algoritmo es que sea capaz de realizar la detección de personas, por lo que la funcionalidad está muy presente. Debe ser en esta característica, donde inicialmente se concentren los mayores esfuerzos para mejorar el algoritmo.
- La característica eficiencia también tiene un peso elevado. Recordemos que la elección del lenguaje de programación C se produjo por la necesidad de ejecutar el algoritmo en tiempo real, es decir, de manera eficiente.
- Las características de mantenibiliadad y portabilidad tienen un papel secundario, ya que el algoritmo se encuentra en una fase preliminar. A medida que la investigación evolucione, su importancia se incrementará.

7. Conclusiones y trabajo futuro

Con la realización de este proyecto se ha logrado adquirir una **visión integradora del conocimiento**, puesto que dos materias en principio alejadas como la visión por computador y la gestión de la calidad han interaccionado de manera estrecha. Entre ellas se ha establecido una **cooperación necesaria**: el progreso en una de ellas, ha supuesto avances significativos en la otra.

El ámbito de la visión por computador ha versado en torno a la detección de personas, un problema muy complejo, que actualmente todavía no ha sido resuelto. En este proyecto hemos implementado un modo de hacerlo, con unos resultados que podemos considerar positivos aunque mejorables. Para tratar de mejorar estos resultados se plantean varias modificaciones al algoritmo: la inclusión de otras características para disminuir el número de falsos positivos, el escalado del modelo omega una sola vez al comienzo del programa para todos los puntos de la imagen... Pero aún así la detección de personas nunca sería completamente fiable, debido a la gran complejidad del problema, en el que sólo se pueden producir progresos si se avanza paso a paso.

Precisamente, con este proyecto se han asentado los cimientos para, en un futuro cercano, **incluir el módulo de detección dentro de un algoritmo de seguimiento de personas** (tracking).

También nos hemos adentrado en el campo de la calidad, valorando esta incursión de manera muy positiva. La aplicación de modelos y estándares proporciona una guía muy útil para **trabajar de modo eficiente**, optimizando el tiempo empleado y evaluando objetivamente los resultados. Los conocimientos adquiridos serán la base para abordar futuros proyectos, mejorando el trabajo de programación desarrollado a través de la aplicación del PSP. Así mismo, aplicar modelos de calidad como el ISO 9126 ayuda a considerar características y subcaracterísticas que habitualmente son olvidadas por los desarrolladores pero que redundan en una mejora de la calidad del producto. En la futura tesis doctoral se contempla la extensión del modelo de evaluación utilizado, considerando subcaracterísticas y la calidad desde el punto de vista del usuario.

En definitiva, este proyecto ha sido ante todo integrador, pero no es más que una pequeña porción de un trabajo mucho más amplio, puesto que **la detección de personas es un campo todavía por mejorar**; supone un gran reto pero también presenta grandes oportunidades.

8. Bibliografía

[UNE-166001] Norma UNE 166001:2006, Gestión de la I+D+i. Requisitos de un proyecto de I+D+i. Publicado: Asociación Española de Normalización y Certificación (AENOR)

[ISO-9126-1] Norma ISO 9126-1:2001, Product quality – Part1: Quality Model. Publisher: International Organitation for Standarization

[ISO-9126-2] Norma ISO 9126-2:2003, Product quality – Part2: External metrics. Publisher: International Organitation for Standarization

[ISO-9126-3] Norma ISO 9126-3:2003, Product quality – Part3: Internal metrics. Publisher: International Organitation for Standarization

[ISO-9126-4] Norma ISO 9126-4:2004, Product quality – Part4: Quality in use metrics. Publisher: International Organitation for Standarization

[Humphrey, 01] Watts S. Humphrey, "Introducción al Proceso de Software Personal". Editorial: Addison Wesley. Año 2001. ISBN:84-7829-052-4.

[Plaza, 10] I. Plaza García, C.T. Medrano Sánchez, A. B. Posa Gómez, "Calidad en actividades de I+D+i. Aplicación en el sector TIC". Editorial: RC Libros. Año 2010. ISBN: 978-84-037769-3-0

[Kernighan, 00] D.W. Kernighan, D.M. Ritchie, "The C programming lenguage". Editorial: Prentice Hall, ISBN 0-13-110362-8

[Igual, 08] R. Igual, C.T. Medrano, "Tutorial de OpenCV". http://docencia-eupt.unizar.es/ctmedra/tutorial_opencv.pdf. Última actualización: 2008

[Bradski, 08] G. Bradski, A. Kaehler, "Learning Opencv. Computer Vision with the OpenCV Library", Año: 2008. Publisher: O`Reilly Media. ISBN: 978-0-596-51613-0

[Wu, 05] Bo Wu, Ram Nevatia, "Detection fo Multiple, Partially Occluded Humans in a Single Image by Bayesian Combination of Edgelet Part Detectors", Proceedings of the IEEE Interntional Conference on Computer Vision. Volumen: 1. Páginas: 90-97

[CAVIAR, 10] CAVIAR: Context Aware Vision using Image-based Active Recognition. Dirección: http://homepages.inf.ed.ac.uk/rbf/CAVIAR/

[Zhao, 08] T. Zhao and R. Nevatia and B. Wu, "Segmentation and tracking of multiple humans in crowded environments". Año : 2008. Revista: IEEE Transactions on Pattern Analysis and Machine Intelligence. Número 30. Volumen 7.

[Li, 09] Li, Min and Zhang, Zhaoxiang and Huang, Kaiqi and Tan, Tieniu, "Rapid and robust human detection and tracking based on omega-shape features". Año 2009. ICIP'09: Proceedings of the 16th IEEE international conference on Image processing. Páginas: 2517—2520. Localización: Cairo, Egypt. Publisher : IEEE Press. Address : Piscataway, NJ, USA. ISBN: 978-1-4244-5653-6.

[García, 05] Garcia, Justin and Lobo, Niels da Vitoria and Shah, Mubarak and Feinstein, Jason, "Automatic

Detection of Heads in Colored Images". Año: 2005. CRV '05: Proceedings of the 2nd Canadian conference on Computer and Robot Vision. Páginas: 276-281, doi: http://dx.doi.org/10.1109/CRV.2005.21. Publisher : IEEE Computer Society, address : Washington, DC, USA. ISBN : 0-7695-2319-6.

[Wu, 09] Wu, Bo and Nevatia, Ram. "Detection and Segmentation of Multiple, Partially Occluded Objects by Grouping, Merging, Assigning Part Detection Responses". Revista: Int. J. Comput. Vision. Volumen : 82. Número: 2. Año: 2009. ISSN : 0920-5691. Páginas : 185—204. doi : http://dx.doi.org/10.1007/s11263-008-0194-9. Publisher : Kluwer Academic Publishers. Address : Hingham, MA, USA.

[Isard, 98] Michael Isard and Andrew Blake, CONDENSATION - conditional density propagation for visual tracking". Revista: International Journal of Computer Vision. Año: 1998. Volumen: 29. Páginas: 5-28.

[Duda, 01] Duda, Richard O. and Hart, Peter E. and Stork, David G, "Pattern Classification". Año: 2001. Publisher : John Wiley & Sons.

[Eshel, 08] Ran Eshel and Yael Moses, "Homography Based Multiple Camera Detection and Tracking of People in a Dense Crowd". Año: 2008.

[Yang, 02] Ming-Hsuan Yang, David J.Kriegman, Narendra Aguja, "Detecting faces in Images: A survey". Año: 2002. Publisher: IEEE Transactions on Pattern Analisys and Machine Intelligence. Volumen: 24. Número: 1.

[Birchfield, 98] Stan Birchfield, "Elliptical Head Tracking Using Intensity Gradients and Color Histograms". Año : 1998. Publisher : IEEE Conference on Computer Vision and Pattern Recognition