ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA

DE TELECOMUNICACIÓN

INGENIERÍA DE TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA

DESARROLLO SOFTWARE PARA MEDICIÓN AUTOMÁTICA DE ALTURAS A TRAVÉS DE UNA CÁMARA NO CALIBRADA

AUTOR: LUIS ALBERTO GARCÍA MORENO

TUTOR: SIAMAK KHATIBI

CO-TUTOR: FELIPE ALONSO ATIENZA

Curso académico 2009/2010

"See, feel and think"

Siamak Khatibi, July 2010

Agradecimientos

La culminación de este proyecto cierra una etapa muy importante en mi vida. Una etapa un tanto difícil pero al mismo tiempo muy enriquecedora e inolvidable. De esta fase no sólo finalizo con mayores conocimientos acerca de las telecomunicaciones, sino que esta etapa me ha servido tanto para enriquecerme personalmente, como para centrar los objetivos y prioridades que de ahora en adelante dispondré a lo largo de mi vida. Por tanto, a todas aquellas personas que me han ayudado en los momentos difíciles y me han animado a continuar con este periodo tan crucial en mi futuro, quisiera agradecerles su apoyo y comprensión.

En primer lugar, quisiera agradecer con especial emoción a toda mi familia, especialmente a mis padres y hermano, por el apoyo recibido durante todos estos años. Ellos me han ayudado a superar los momentos más duros que se me han presentado durante estos años, han reconocido mi esfuerzo e incluso, por qué no decirlo, me han dirigido por el camino correcto a través de su comprensión y cariño.

En segundo lugar, agradecer a mis tutores el esfuerzo, dedicación y paciencia que me han dedicado a lo largo de este proyecto. Como tutor en mi universidad de estancia durante la realización del proyecto en Karlskrona (Suecia), quiero agradecerle a Siamak Khatibi, Doctor en *Electrical Engineering*, por el apoyo y guía a lo largo de la realización del proyecto, aportándome ideas y soluciones en momentos claves del trabajo. También agradecerle a mi tutor en España, Felipe Alonso Atienza, por facilitarme los trámites para la presentación del proyecto en español y ayudarme con esta memoria.

También agradecer a todas las instituciones que han hecho posible mi estancia durante un año en Suecia, como estudiante Erasmus, haciendo posible que disfrutara de la mejor etapa de mi vida.

Por último, agradecerle a todos mis compañeros, pero por encima de todo amigos, que he tenido a lo largo de esta travesía. Tanto a los que he tenido aquí en Madrid, en especial a mi eterno compañero Rafael, como a los integrantes de mi año Erasmus, los cuales gracias a las charlas en las salas comunes de nuestra residencia me han hecho solventar más de un escollo que se me presentó durante la realización del proyecto, teniendo mención especial a mi amigo Roberto. A todos ellos, GRACIAS.

Resumen

Proyecto Fin de Carrera realizado en la universidad Blekinge Tekniska Högskola, situada en Karlskrona (Suecia), durante el curso académico 2009/2010 en virtud de una beca de intercambio del programa Erasmus. El proyecto ha sido realizado en el Departamento de Ingeniería, bajo la supervisión del Profesor Siamak Khatibi, Doctor en *Electrical Engineering* por *Chalmers University of Technology* de Göteborg(Suecia).

La estimación de la altura es un dato muy útil cuando nos referimos a la identificación de personas. También es bien sabido que el uso del vídeo en sistemas con cámaras de videovigilancia está creciendo día tras día y con ello, la necesidad de pistas antropomórficas que ayuden a la lucha en contra de la delincuencia. La detección y medida de las personas es un aspecto básico en el campo de la seguridad automatizada.

Ya existen varios estudios acerca de la detección de altura humana. Estos sistemas son métodos manuales simplemente basados en métodos manuales basados en fotografías estáticas, pero en todos ellos, una supervisión humana es necesaria con el objeto de realizar una medida en la fotografía. En cuanto a los métodos de detección de objetos en movimiento, hay alguno basado en estadísticos, en antropometría e incluso en la manera de andar de las personas. También hay algunos métodos basados en el conocimiento de los parámetros intrínsecos de la cámara, al igual que basados en cámaras estereoscópicas, pero en este caso, una etapa de calibración es siempre necesaria y siempre necesita de alguien con los conocimientos suficientes para calibrarlo.

El presente proyecto presenta un nuevo método de estimación de alturas de las personas que atraviesen un determinado escenario a partir de la secuencia de imágenes registradas por una cámara cualquiera en un lugar fijo. El método propuesto combina ideas de geometría proyectiva con algoritmos de seguimiento de personas, ofreciendo mayor simplicidad en relación a los algoritmos propuestos hasta la fecha y posibilitando la utilización de casi cualquier tipo de cámaras, como podrían ser las cámaras web.

Índice general

1 Introducción

En este capítulo se presenta una visión general del problema a tratar, así como el propósito del proyecto y la estructura del mismo.

1.1 Estado del arte

La estimación de la altura es un dato muy útil cuando nos referimos a la identificación de personas. También es bien sabido que el uso del vídeo en sistemas con cámaras de videovigilancia está creciendo día tras día y con ello, la necesidad de pistas antropomórficas que ayuden a la lucha en contra de la delincuencia. La detección y medida de las personas es un aspecto básico en el campo de la seguridad automatizada. Hoy en día, la implementación de sistemas de seguridad está muy extendida, pero en todas ellas existe una gran limitación, ya que siempre debe haber alguien encargado de controlar las pantallas. Sin embargo, este sistema es capaz de detectar el intruso y almacenar su altura sin necesidad de la intervención humana, lo que agiliza de una manera crucial el método. De este modo nos acercamos al paradigma de la seguridad totalmente automatizada.

Por otra parte, la estimación de la altura de una persona podría ser muy útil en las investigaciones antropomórficas debido al hecho de que hace posible su medida sin necesidad de distorsionar la muestra. En otras palabras, la gente que pase en frente de las cámaras previamente fijadas, por ejemplo, en una calle muy concurrida de gente dentro de una ciudad, no se van a percatar de que están siendo medidos por este nuevo sistema, por lo que las medidas tomadas serán de tal manera que se tomarán las que realmente son, ya que no intentarán cambiar su estatura, hecho que inconscientemente se realiza cuando se tiene la certeza de que se está siendo medido.

Ya existen varios estudios acerca de la detección de altura humana. Estos sistemas son métodos manuales simplemente basados en métodos manuales basados en fotografías estáticas, pero en todos ellos, una supervisión humana es necesaria con el objeto de realizar

una medida en la fotografía. En cuanto a los métodos de detección de objetos en movimiento, hay alguno basado en estadísticos, en antropometría e incluso en la manera de andar de las personas. También hay algunos métodos basados en el conocimiento de los parámetros intrínsecos de la cámara, al igual que basados en cámaras estereoscópicas, pero en este caso, una etapa de calibración es siempre necesaria y siempre necesita de alguien con los conocimientos suficientes para calibrarlo. Este proyecto está basado en relaciones invariantes dentro de la geometría proyectiva, los cuales nos permiten medir sin hacer ninguna calibración en la cámara o en los métodos estadísticos.

Uno de los puntos que debe quedar claramente definido es el cual el propósito de éste trabajo no es el de conseguir el mejor detector de personas, ya que ya existen una gran cantidad de métodos los cuales son capaces de detectar a cualquier persona pasando por delante de la cámara con una precisión sombrosa. Pero esta parte será abordada en un capítulo final, el cual explicará de manera profunda y en detalle la manera en la que el sistema puede ser mejorado y en qué campos es posible su mejora.

En resumen, este nuevo método combina ideas de geometría proyectiva con simples algoritmos de seguimiento de personas. De esta manera, se está ofreciendo mayor simplicidad y la posibilidad de usar casi cualquier tipo de cámaras, como podrían ser las cámaras web. De hecho, la mayoría de los algoritmos empleados no son nuevos. Pero lo que no está hecho todavía es la idea de combinar todos ellos y hacerlos ejecutar de una manera autónoma y automática.

Este proyecto, por tanto, está simplemente buscando un nuevo sistema que pudiera ser clasificado como “plug and play”. En otras palabras, se está buscando implementar un sistema que simplemente posicionándolo en el escenario correcto, pudiera ser capaz de configurarse por sí mismo sin la necesidad de alguien con conocimientos previos acerca de esta tecnología, por lo que resulte fácil de usar para todo el mundo.

1.2 Motivación

Las imágenes o secuencias de imágenes contienen una cantidad enorme de información geométrica acerca de la escena representada. La motivación de este proyecto será el extraer esta información de una manera cuantificable y precisa.

La fotogrametría es el proceso de obtener medidas desde imágenes. Las medidas sobre las imágenes están siendo usadas para proporcionar información útil acerca del curso de los eventos y el tamaño tanto de objetos como de personas, por ejemplo, en la escena de un crimen. Una de las actividades forenses más comúnmente usadas que requieren de la fotogrametría es el análisis de la altura de las personas. Esta estimación de alturas, tiene muchas y variadas aplicaciones, como pudieran ser las dedicadas a biometría o el seguimiento de personas. Para el primer caso, la estatura puede ser usada para el rastreo de una misma persona en un sistema de videovigilancia, y de esta manera mantener control sobre su posición en todo momento. En cuanto al segundo caso, esta aplicación puede ser usada para diferenciar distintas personas a partir de un grupo dado de individuos en una misma escena.

La estatura, de esta manera, podría llegar a ser una característica de información muy útil en el futuro. En el caso de que se pudiera disponer con dos o más imágenes, la estimación de la estatura mediante la correspondencia entre ambas fotografías es muy cara, computacionalmente hablando, y existen ciertas ambigüedades en las medidas, por lo que hasta el momento no representa ninguna eficiencia. En caso de que simplemente se disponga de una imagen, la medición de la estatura debe ser realizada a partir de esta imagen. Para ello, la única acción que debe ser llevada a cabo por una persona será la de tomar una altura real del escenario hacia donde la cámara está dirigida. Éste hecho, será crucial en el método utilizado.

De esta manera, el objetivo del proyecto será la de tratar con una situación en la que la cámara tiene una posición fija y los sujetos tienen cierto movimiento en el escenario al que la cámara apunta. Esta cámara será previamente fijada en un escenario en concreto, el cual se explicará con detalle a lo largo de esta memoria.

1.3 Alcance del trabajo

El objetivo de este proyecto es el de conseguir una medición de la estatura de los sujetos que pasen por delante de una cámara fijada con antelación en un lugar concreto de un escenario. El propósito no es simplemente medir la altura de la persona en un simple fotograma, ya que eso se ha realizado hace ya mucho tiempo. Esto se hacía tomando un fotograma de un vídeo y manualmente seleccionando la altura de referencia y la altura a ser medida. Gracias a la geometría proyectiva, esta altura podrá ser calculada fácilmente. Las nuevas opciones que se presentan aquí son la posibilidad de situar una cámara en un escenario en concreto y una vez fijada allí, será posible detectar automáticamente en qué clase de escenario se está trabajando y una vez facilitada una altura real, automáticamente se podrán detectar a las personas y mostrar sus alturas por pantalla.

En éste trabajo, no se va a hacer un estudio de qué posibilidades se pueden alcanzar una vez conseguidos los resultados, ya que una gran cantidad de oportunidades diferentes y variadas se abren ante esta nueva aplicación. Teniendo en cuenta que ya se disponen de cámaras prácticamente en todos los lugares y que para la realización de esta aplicación no es necesaria una cámara de gran calidad, esta aplicación puede ser aplicada en tantas aplicaciones como imaginemos. Sabiendo además, que las personas a ser estudiadas no necesitan pararse en frente de la cámara y que incluso no necesitan darse cuenta de que están siendo medidos, un estudio acerca de la altura de la población en una ciudad o incluso en un país puede realizarse. Pero esto es simplemente un simple ejemplo acerca de todas las nuevas posibilidades que esta técnica aporta.

El método que va a ser explicado a lo largo de esta memoria, intentará explicar qué pasos se deben seguir para estimar las alturas que se están tratando de hallar. El principal objetivo, no es el de conseguir la medida más precisa en todo momento, ya que una gran cantidad de medidas podrán ser realizadas en los vídeos y alguna medida puede incluir cierto error. Eso sí, al final del vídeo se dispondrá de una idea clara de la altura real del sujeto.

Debido al estudio de una gran cantidad de campos distintos a lo largo de este proyecto, no se trata de profundizar a fondo en ellos, sino en usar simplemente soluciones

sencillas a todos ellos, consiguiendo en general un resultado satisfactorio. Si deseamos mejorar nuestra solución, simplemente nos bastaría con profundizar en cada paso y e intentar perfeccionarlo con soluciones óptimas. De esta manera, puede hacerse más robusto y preciso en comparación a la solución que se presenta.

1.4 Esquema de la memoria

Esta memoria, ha sido dividida en varios capítulos, dependiendo del objetivo de cada uno de ellos. A continuación se desglosa una pequeña descripción de los elementos que se pueden encontrar en cada capítulo.

- **Capítulo 1: Introducción.**

  En este capítulo se presenta la motivación de este proyecto, así como los objetivos planteados. También se incluye un breve resumen de los contenidos de la memoria.

- **Capítulo 2: Principios teóricos**

  En este capítulo, se explicarán los fundamentos en los cuales explicarán en qué se basa la estimación de altura en este proyecto. También serán enumerados todos los elementos necesarios para su implementación.

- **Capítulo 3: Materiales y métodos**

  En este capítulo se extiende la parte principal y más importante de la memoria. Aquí, la estrategia seguida para solventar el problema propuesto es mostrada. También en esta sección se dividirá el proyecto en dos campos principales: la encargada de analizar el escenario y la parte encargada de detectar a las personas.

- **Capítulo 4: Desarrollo software**

  En este capítulo se explican tanto los diferentes softwares empleados, como un resumen de todas las funciones creadas para la realización del proyecto.

- **Capítulo 5: Resultados**

  En este capítulo se presenta un análisis de los resultados obtenidos.

- **Capítulo 6: Conclusiones y futuras líneas de investigación**

  En este capítulo se van a presentar las posibles líneas de investigación que se pueden seguir en un futuro.

- **Apéndice 1: Instalación de OpenCV con Microsoft Visual C++ 2008**

  En este apéndice, se presenta una pequeña guía para la instalación de OpenCV.

- **Apéndice 2: Cómo configurar Microsoft Visual C++ para llamar al motor MATLAB©.**

  En este segundo apéndice, se va a explicar el modo en el que se puede utilizar el motor MATLAB© desde C++.

2 Principios teóricos

2.1 Introducción a la geometría proyectiva

La geometría proyectiva es una rama de las matemáticas la cual estudia la incidencia entre los puntos y las líneas sin tener en cuenta las medidas que se pudieran realizar. Por lo que, es justo decir, que la geometría proyectiva es una geometría no-métrica.

Gérard Desargues está considerado el primero en introducirse en este campo de las matemáticas, desde que desarrolló, en el siglo XVII, la base matemática para los métodos de perspectiva usados por los artistas del Renacimiento. En el siglo XIX, la geometría proyectiva fue definitivamente incluida como otro campo más a estudio en el ámbito de las matemáticas, gracias al desarrollo de un modelo analítico. Esta evolución terminó a principios del siglo XX, desde que Einstein demostró que, a gran escala, el universo podría ser interpretado con su "nueva geometría" mejor que con la geometría euclídea. Sin embargo, en 1908 Hermann Minkowski descubrió que la teoría de Einstein de especial relatividad era de hecho debido a la existencia de una dimensión de cuatro dimensiones espacio-tiempo y que podría ser analizada usando geometría proyectiva.

Gracias al uso de la geometría proyectiva, es posible calcular longitudes reales directamente desde medidas en la imagen como podemos ver en la Figura 1. En esta figura se representa la proyección de la altura Y, desde el punto de visión C, en un plano situado entre el punto de visión y la altura real. A partir de esta proyección se realizarán las medidas que se explicarán más adelante.

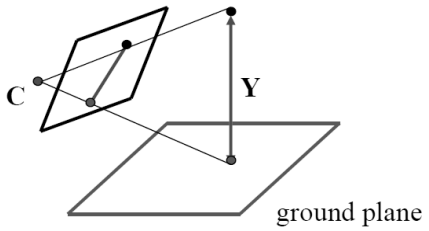


Figura 1: Cómputo de distancias midiendo en la imagen

2.2 Puntos de fuga y línea de horizonte

De acuerdo con los principios de esta disciplina, todas las líneas paralelas irán a parar al mismo punto, llamado punto de fuga, el cual está considerado estar situado en el infinito. Esta convergencia está representada en la Figura 2, en la cual se pueden ver un par de puntos de fuga, V_1 y$V_{2,}$ uno para cada grupo de líneas paralelas.

Además, como podemos ver en la Figura 1, la unión de los puntos de fuga dará lugar a la línea de horizonte, también llamada línea de fuga.

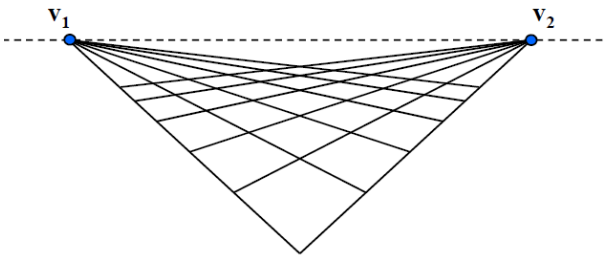


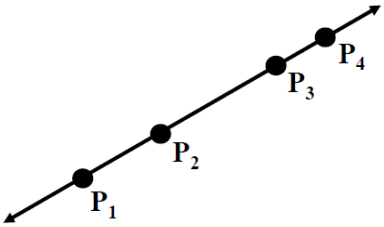
Figura 2: Puntos de fuga provenientes de los distintos grupos de líneas paralelas

Cómo se verá más adelante, la línea de horizonte es muy útil. La propiedad principal de línea de horizonte es que está situada exactamente a la misma altura que está situado el observador. En otras palabras, nos está proporcionando la altura exacta de, en este caso, la cámara. De esta manera, sin usar métodos explícitos de calibración, la geometría proyectiva nos está proporcionando algunos parámetros externos.

2.3 Cross ratio o cociente cruzado

El denominado *cross-ratio* o cociente cruzado, es una constante numérica de una cuádrupla de cuatro puntos en una línea y de cuatro puntos de líneas concurrentes en un plano.

Esta relación, juega un papel clave en la geometría proyectiva, ya que es la única constante que se mantiene invariante, relacionando cuatro puntos de una línea proyectiva.
Así, si tenemos cuatro puntos pertenecientes a la misma línea, la relación entre ellos es siempre la misma, independientemente del orden de los puntos, como queda representado en la Figura 3.



$$\frac{\|P_3 - P_1\| \, \|P_4 - P_2\|}{\|P_3 - P_2\| \, \|P_4 - P_1\|} = CR$$

Figura 3: Relación de cuatro puntos en una línea

Este resultado también es válido cuando se está trabajando en el plano imagen, ya que también es invariante. Esta es la razón por la que el c*ross-ratio* representa un papel tan fundamental en la estimación de la altura a través de imágenes fijas.

2.4 Estimación de la altura

Teniendo en cuenta los resultados anteriores, la estimación de alturas en una imagen se produce de la siguiente manera:

En la Figura 4, se representa la proyección de la realidad en el plano imagen. En dicha figura, H es la altura de las personas que queremos calcular y R es la longitud real del objeto de referencia.Según la fórmula de la imagen que representa el *cross-ratio*, las únicas distancias que necesitamos para medir son las que e encuentran en el plano de la imagen, tales como |t-b| ó |r-b|.

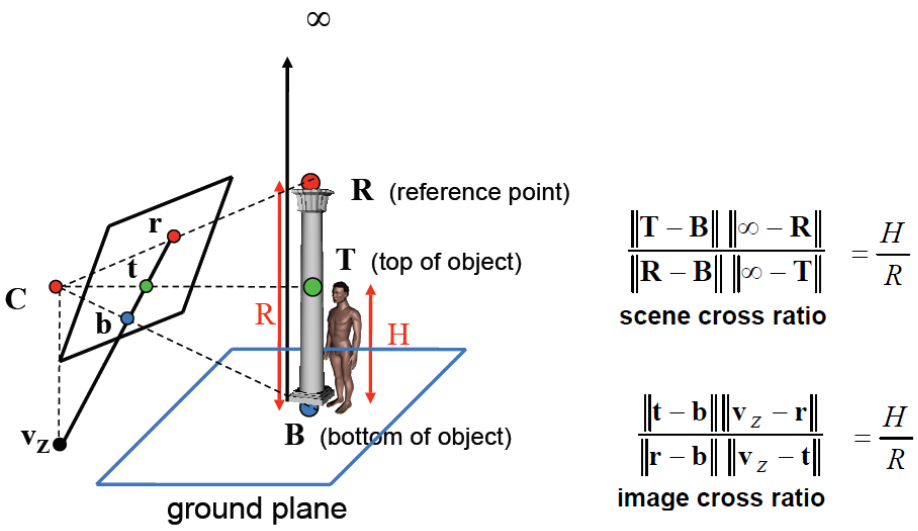


Figura 4. Midiendo alturas en el plano imagen usando el *cross-ratio*

Es importante tener en cuenta que si el punto de fuga vertical Vz se encuentra muy alejado, la relación puede aproximarse como: |Vz-r|/|Vz-t|~1, quedando la fórmula reducida a:

$$\frac{\|t-b\|}{\|r-b\|}=\frac{H}{R} \qquad (1)$$

El procedimiento a seguir para realizar la proyección y la consiguiente medida, es el ilustrado en la Figura 5:

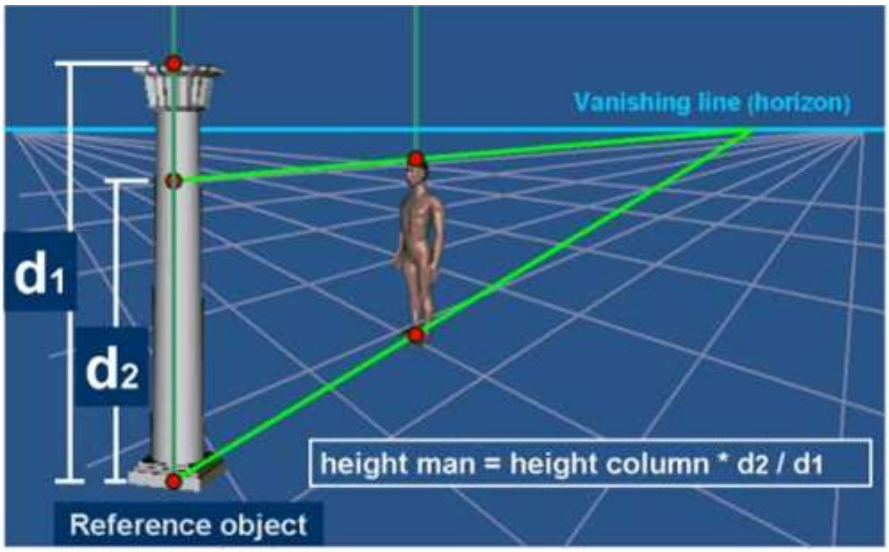


Figura 5. Procedimiento para calcular las alturas en la que el punto de fuga vertical está en el infinito

Los pasos, descritos uno a uno son los siguientes:

- Se traza una línea desde la base del objeto de referencia hasta la línea de horizonte, a través de la base del objeto que quiere ser medido.
- Se traza otra línea desde la intersección, entre la línea de horizonte y la línea anteriormente trazada, uniendo este punto con la parte superior del objeto a ser medido.
- El corte de la línea previa con el objeto de referencia proporcionará el punto que determinará el segmento d_2.
- El último paso es simplemente calcular la formula mostrada en la figura 5, en la cual d_1 y d_2 son distancias medidas en píxeles.

Es importante también mencionar que el objeto a ser medido tiene que encontrarse en el mismo plano que el objeto de referencia. De otro modo, sería necesario medir la distancia entre planos y el procedimiento no es tan simple. Referencias a estos otros métodos, pueden encontrase en [1] y [2].

3 Materiales y métodos

3.1 Introducción

En este capítulo se resumen y explican todos los diferentes métodos y estrategias que han sido usados durante el desarrollo de este proyecto para solventar los diferentes problemas que se presentan a lo largo del desarrollo del proyecto. Serán explicados a fondo y paso a paso, de manera que puedan ser seguidos por posibles nuevos investigadores que pudieran estar interesados en el uso de esta técnica o en su profundización.

La principal característica de este proyecto es que se divide en dos grandes sub-áreas de trabajo. La primera, será la que se encargue de analizar el escenario, una vez la cámara haya sido situada en un lugar apropiado. La segunda, detectará a toda persona que esté pasando por delante de la cámara, dentro del campo visual de esta, y con ello, calcular simplemente con una sencilla relación la altura de la persona, para finalmente que sea mostrado en el vídeo. Las distintas funciones van a realizarse en MATLAB© o en C++, dependiendo del objetivo de cada función, y teniendo en cuenta las características de cada lenguaje, se utilizará uno u otro lenguaje, el que mayores ventajas presente en cada momento.

El unir ambos motores y saber cómo usarlos de manera conjunta, será tema de desarrollo durante un capítulo entero, el cual se encontrará al final de la memoria. Además, a modo de apéndice se ha añadido la manera de configurar ambos programas, explicando también de manera fácil y a modo de tutorial cómo instalar y dónde conseguir todas las librerías necesarias y como se han de hacer correctamente las llamadas de un programa a otro. En la página siguiente, en la Figura 6, se muestra un flujograma acerca del software implementado, simplemente con el objetivo de que el lector se haga una ligera idea acerca de los diferentes objetivos que se intentarán encontrar a lo largo de este proyecto y sabiendo qué programa será el utilizado para resolverlo.

Como se puede comprobar, el siguiente flujograma divide cada tarea simple en un rectángulo independiente. Se puede tratar de entender como un pseudocódigo, en el que principalmente se destaca el orden de cada función y el programa que lo realiza.

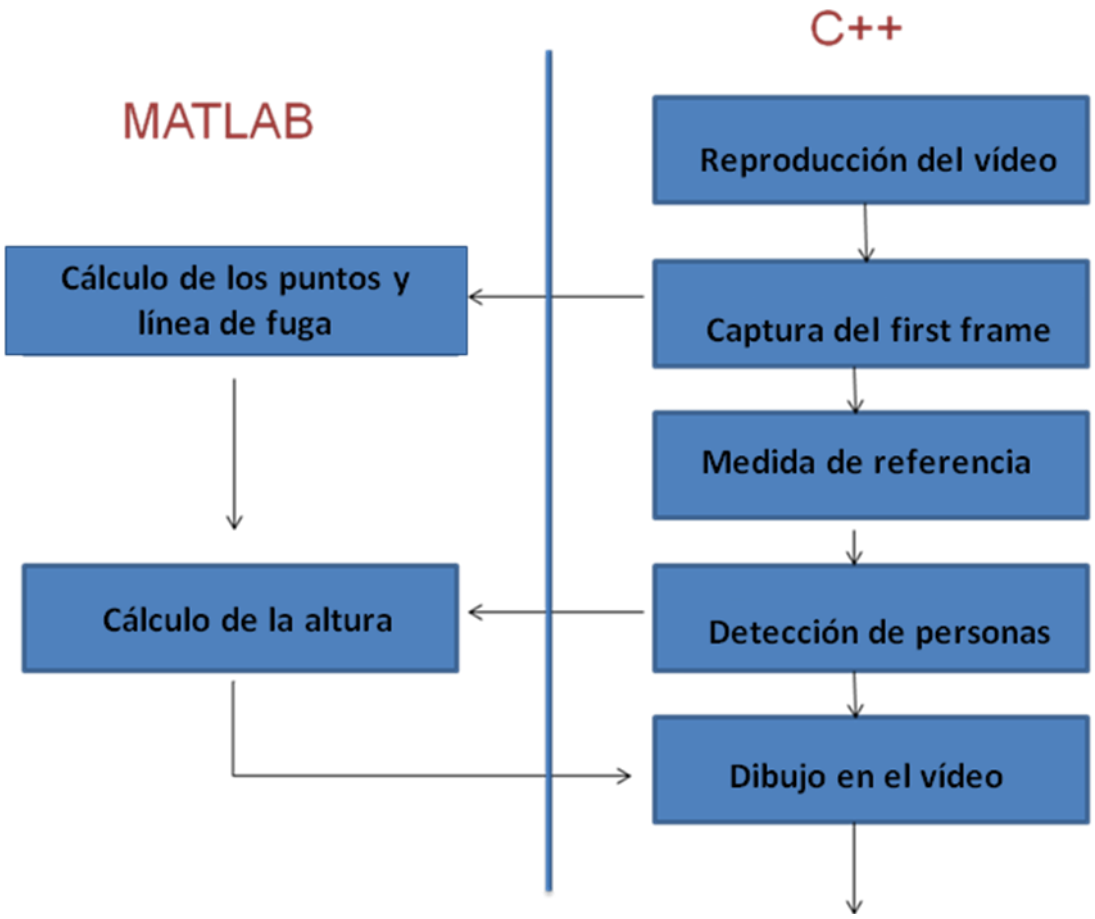


Figura 6. Flujograma general

3.2 Análisis del escenario

3.2.1 Introducción

En esta sección se van a explicar las razones por las cuales es necesario un análisis previo del escenario donde se van a realizar las medidas. En primer lugar, se debe tener en cuenta que la técnica utilizada para el cálculo de alturas está basada en geometría proyectiva. De este modo, el encontrar los puntos y líneas de fuga es un punto crucial en el análisis del escenario. A partir de estos datos, seremos capaces de dibujar diferentes líneas y encontrar los puntos de cruce desde los cuales las diferentes relaciones serán encontradas, y entonces será posible la medida a realizar. Sin estas líneas y puntos de fuga, no será posible aplicar los siguientes pasos, que serán descritos a continuación.

Con el fin de detectar la línea de horizonte, también llamada línea de fuga, lo primero que se debe calcular son los puntos de fuga. Esto debe ser así, ya que mediante la unión de ambos puntos de fuga, se podrá trazar la línea de horizonte.

Cada punto de fuga está definido por un punto imaginario empleado a la hora de representar perspectiva, en el cual todas las líneas paralelas de un mismo plano aparentemente convergen. En el modelo descrito, se ha aclarado que los modelos soportados son los correspondientes a escenarios con perspectiva de dos puntos. Por ello, se deben encontrar dos puntos de fuga distintos. Para conseguir esto, hemos aplicado detección de bordes al escenario. Así, somos capaces de dibujar todas las líneas correspondientes al escenario que van a ser usados como modelo.

Debido a las limitaciones características de este software, se deben usar escenarios, tan “limpios” como sea posible. Se emplea el término “limpio”, cuando nos referimos a lugares donde no hay objetos situados en planos que no sean perpendiculares entre sí, evitando de esta manera, que se detecten líneas las cuales no sean útiles para nuestro estudio. Todos los detalles acerca del uso y principios de las diferentes maneras de conseguir la detección de bordes, serán descritos en un capítulo aparte.

Una vez evitados, en la medida de lo posible, los objetos que puedan introducir líneas que no sean útiles para nuestro propósito, se usarán los objetos situados en los muros, como

pudieran ser las puertas, ventanas y todas las líneas paralelas, como el mismo suelo con el fin de conseguir líneas útiles para nuestro propósito. Entonces, una vez habiendo obtenido una imagen binaria, compuesta simplemente por todos los bordes del escenario, tendremos que usar diferentes técnicas para averiguar los puntos de fuga. Esto se podrá conseguir gracias a las diferentes líneas rectas que se encuentren en la imagen binaria. El uso de estas técnicas también será descrito en los siguientes capítulos y las razones por las que se han escogido unas específicas en vez de otras entre varias opciones distintas.

Tras haber encontrado las líneas rectas que forman parte del escenario, tendremos que encontrar donde van a cruzarse todas ellas. Esto no es una tarea fácil, ya que todas las líneas encontradas no van a ser paralelas entre sí, y las que lo sean, no serán detectadas con el ángulo perfecto. Todo lo referente a la detección de los puntos de fuga, también será explicada en un capítulo aparte. Una vez superada esta fase, teniendo todos los puntos de fuga, seremos capaces de encontrar la línea de horizonte, simplemente gracias a su unión. El resultado entonces será representado en la imagen.

A partir de ahora, una vez clara una visión general de todos los pasos a seguir, los siguientes capítulos van a explicar a fondo cada uno de ellos y las razones por las cuales cada método ha sido elegido más apropiado dependiendo de las condiciones encontradas en las diferentes imágenes.

3.2.2 Escenarios analizados

Debido a la innovación en este proyecto, y de tratarse simplemente de una introducción a nuevos investigadores, van a ser aceptados dos tipos distintos de escenarios por nuestro software de demostración. Estos tipos de escenario, son los llamados de perspectiva cónica de dos puntos. En este tipo de entornos, un grupo de líneas paralelas en un mismo plano va a representar un punto de fuga, y el otro punto se verá representado por otro conjunto de líneas paralelas correspondientes a otro plano, perpendicular al anterior.

Por ejemplo, si miramos a un edificio desde una esquina, un muro parecerá dirigirse hacia un punto de fuga, mientras que el otro muro lo hará hacia el otro punto. Estos puntos aparecerán uno a cada lado del centro de la imagen, y será el punto donde parecen dirigirse los distintos grupos de líneas. Para el correcto funcionamiento del algoritmo, se deben encontrar dos puntos de fuga distintos. Es por ello que el escenario hacia donde la cámara está dirigida debe amoldarse a un modelo previamente definido, y que se acoja a dos modelos distintos.

El primero de estos modelos, es aquel en el que la cámara esté enfrentada a una esquina, de manera que se vean dos muros perpendiculares a la vez. De esta manera, los dos puntos de fuga diferentes estarán situados a ambos lados, siguiendo las líneas detectadas. La razón por la que tenemos que encontrar una esquina es debido a la necesidad de encontrar líneas situadas en planos perpendiculares.

La siguiente figura muestra un modelo de las clases de escenario que pueden ser usados y aceptados para nuestro propósito. Tras este modelo, se muestran dos ejemplos de escenarios en los que esta técnica puede funcionar.

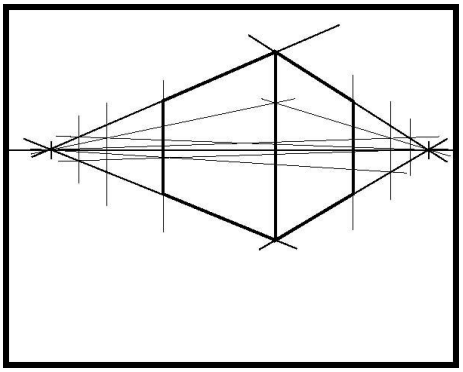


Figura 7. Modelo: Primera clase de escenario aceptado

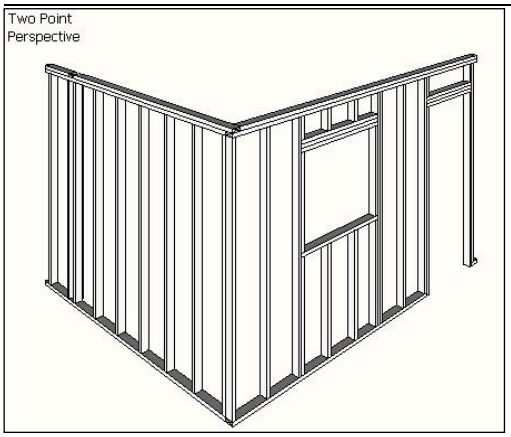


Figura 8.Ejemplo: Primer tipo de escenario

Figura9.Ejemplo: Primer tipo de escenario

El segundo tipo escenario que puede ser usado como modelo para el software implementado, es aquel que esté dirigido hacia un rincón, como queda representado en la figura 10. Esto quiere decir, que las líneas de cada pared, que formen cada punto de fuga en el infinito, irán a parar al otro lado de la imagen. De esta manera, las líneas provenientes de la parte derecha de la imagen, formarán el punto de fuga de la izquierda, y viceversa.

Las figuras 11 y 12 muestran dos ejemplos reales de posibles escenarios correspondientes al tipo de modelo descrito.

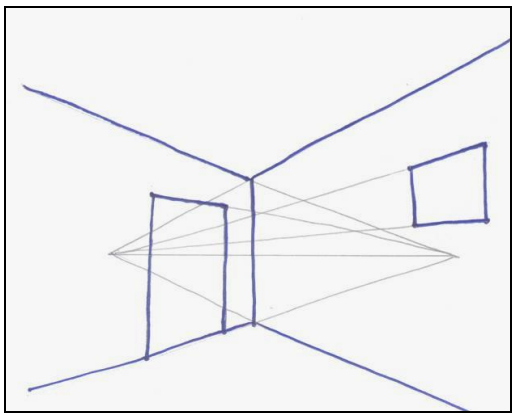


Figura 10. Modelo: Segundo tipo de escenario.

Figura 11. Ejemplo 1: Segundo tipo de escenario

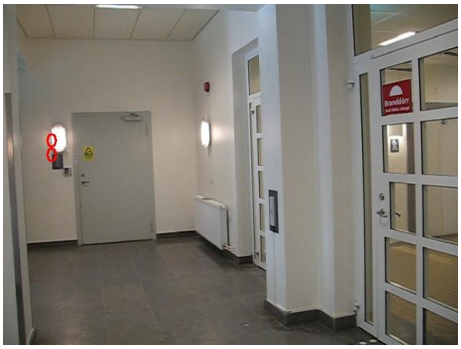


Figura12.Ejemplo 2: Segundo tipo de escenario

3.2.3 Detección de bordes

La detección de bordes es una herramienta fundamental en el procesamiento de imágenes y en el campo de visión artificial. Particularmente en aquellas áreas de detección de características y extracción de las mismas, en donde la meta es detectar cambios bruscos en el brillo de las imágenes.

El caso ideal en este caso, será aplicar una detección de bordes a una imagen con el fin de conseguir un conjunto de puntos conectados que pueda indicar los límites de los objetos y los límites de las diferentes superficies. Esto ayudará a descubrir la orientación de la superficie. De esta manera, la imagen original con gran cantidad de información, se verá reemplazada por una imagen binaria, simplemente mostrando los bordes de la imagen. De esta manera, la cantidad de datos que serán procesados se verá reducida enormemente y los datos menos relevantes serán filtrados y desechados. Si tras la etapa de detección de bordes, se consigue un gran resultado, la siguiente tarea de interpretar la información obtenida en la imagen se verá simplificada. El problema reside en que no siempre se consigue un buen resultado tras la detección de bordes, los bordes extraídos de imágenes que no son claras, son a menudo afectadas por la fragmentación, lo que significa que ciertos bordes pueden verse desconectados, perdiéndose segmentos de los bordes, al igual que bordes erróneos que nos se corresponden con zonas de interés de la imagen.

La detección de bordes es uno de los pasos fundamentales en el procesamiento de imágenes, por ello, durante los últimos años han sido hechas importantes investigaciones en

este campo. A pesar de ello, todavía hay varias investigaciones en marcha intentando mejorar en este terreno, ya que los resultados obtenidos están todavía lejos de ser precisos e infalibles.

Tipos de detección de bordes

Existen muchos métodos para la detección de bordes, pero la mayoría pueden ser separados en dos grandes grupos, basados en búsqueda, llamados técnicamente *search-based* y en cruces por cero, denominados *zero-crossing*.

Los métodos denominados "search-based", detectan bordes primero calculando una medida de la intensidad, normalmente una expresión derivable de primer orden, como pudiera ser la magnitud del gradiente y entonces, basado en esto, la búsqueda de la derivada local máxima de la magnitud del gradiente, usando un cálculo estimado de la orientación local del borde. Normalmente es usada la dirección del gradiente.

Por otra parte, los métodos llamados *zero-crossing* buscan cruces por cero en una expresión de segundo grado, calculado desde la imagen original, con el fin de encontrar los bordes, normalmente los cruces por cero del Lapalciano o los cruces por cero de una expresión diferencial no-lineal.

Los métodos de detección de bordes que han sido publicados principalmente varían unos respecto a otros por el tipo de filtro de suavizado empleado y la manera en la que las medidas de intensidad son calculadas. Muchos detectores de bordes están basados en el cálculo de gradientes en las imágenes, por lo que difieren en el tipo de filtro usado para calcular el gradiente. El método usado para emplear la detección de bordes en este proyecto ha sido el detector de bordes llamado *Canny*.

Método *Canny* de detección de bordes

El operador *Canny* para detección de bordes fue desarrollado por John F. *Canny*, científico y matemático Americano, en 1986 y usa un algoritmo de varias etapas para detectar una amplia gama de bordes en imágenes. Este algoritmo, es famoso por ser el detector de bordes óptimo, hasta el momento. Las intenciones de *Canny*, fueron las de mejorar las técnicas ya existentes en la época en la que comenzó su trabajo. Finalmente, tuvo un gran éxito al conseguir las metas que se propuso, todas sus ideas y método se pueden encontrar en su informe [3]. En este informe, sigue una lista de criterios para conseguir mejorar los

métodos de la detección de bordes. El primero y más obvio es buscar una baja tasa de error, en cuanto a la detección de bordes falsos. Este punto será crucial para el propósito buscado, ya que lo que debemos intentar ante todo es eliminar bordes que no existan en la imagen original.

El Segundo aspecto en el que se centró *Canny*, fue en el de conseguir que aquellos puntos que han sido detectados como bordes, estén bien localizados. En otras palabras, la distancia entre el borde real y el pixel clasificado como borde, sea mínima. Este punto también va a ser una parte crucial en nuestro trabajo, ya que se necesita obtener el punto de fuga más preciso posible, y este punto es donde crucen todas las líneas.

La última característica que hace *Canny* superior a otros métodos para este trabajo es que es el único capaz de simplemente detectar una línea para cada borde. Esto ofreció una mejora ya que todos los métodos anteriores no eran lo suficientemente sólido para eliminar completamente la posibilidad de múltiples detecciones de borde para un solo borde real.

El algoritmo de detección de bordes *Canny*

Este algoritmo se ejecuta en cinco pasos consecutivos, los cuales se detallan a continuación:

- Suavizado.: Hacer la imagen más borrosa con el objetivo de eliminar el ruido.
- Encontrar gradientes: Los bordes deben ser situados en lugares donde los gradientes de las imágenes sean máximos.
- Evitar la eliminación de máximos: Simplemente debemos preocuparnos acerca de máximos locales, tratando de localizar los más significativos.
- Doble límite: Los posibles bordes son determinados por el uso de un doble límite.
- Rastreo de bordes mediante el uso de histéresis: Los bordes definitivos que finalmente serán mostrados vendrán determinados por estar contenidos o fuertemente conectados con un borde ya detectado.

Cada paso está explicado en detalle en las siguientes secciones:

- **Suavizado**

Se asume, que toda imagen que haya sido tomada desde cualquier tipo de cámara va a contener ruido. Para tratar de evitar este ruido, que pueda ser la causa de bordes defectuosamente detectados, el ruido debe reducirse. De esta manera, la imagen que queremos estudiar primero se hace mas borrosa, usando un filtro gausiano. Este filtro gausiano está diseñado con una desviación típica de σ=1.4, con ello, el núcleo del kernel usado, responde a la ecuación (1).

$$B = \frac{1}{159} \cdot \begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} \qquad (1)$$

- **Búsqueda de gradientes**

El algoritmo *Canny* está basado en cambios repentinos en la intensidad de los píxeles, en lugares donde la imagen sufre mayores cambios. Estas áreas son encontradas mediante el uso de gradientes en la imagen. El gradiente en cada pixel de la imagen suavizada está determinada por la aplicación del llamado operador-Sobel, por lo que el primer paso que hay que aplicar es calcular el gradiente en las direcciones x e y, mediante la aplicación de los núcleos, también llamados kernels, mostrados en las Ecuaciónes (2) y (3).

$$K_{GX} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad (2)$$

$$K_{GY} = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \quad (3)$$

A la magnitud del gradiente, se le denominará intensidad del borde a partir de este punto.

Además, usando la ley de Pitágoras, es la manera en la que se va a determinar la distancia Euclídea. Se calculará usando la siguiente expresión (4):

$$|G| = \sqrt{G_x^2 + G_Y^2} \quad (4)$$

Donde G_x y $G_{y,}$ son los gradientes en las direcciones x e y respectivamente.

A veces se puede simplificar aplicando la distancia de Manhattan, usando entonces la expresión (5) para su cálculo.

$$|G| = |G_X| + |G_y| \quad (5)$$

Usando esta expresión, se podrá reducir la carga computacional al cálculo de la distancia.

Una imagen mostrando la magnitud del gradiente en ocasiones expresa los bordes con bastante claridad. Sin embargo, los bordes en la mayoría de ocasiones son bastante anchos, por lo que no podemos indicar exactamente donde se encuentra exactamente este borde. La posibilidad de cómo determinar esto, es explicada en la siguiente sección. No obstante, lo que previamente se ha de calcular es la dirección de los bordes y ser almacenada de la manera en la que se muestra en la ecuación (6):

$$\theta = arctg\left(\frac{|G_y|}{|G_x|}\right) \quad (6)$$

- **Evitar la eliminación de máximos locales.**

  La principal meta de esta sección es el cambio desde bordes "borrosos" en la imagen de magnitudes de gradientes, a bordes claramente definidos. Este paso se

realiza básicamente manteniendo todos los máximos en la imagen del gradiente, y una vez conseguido esto, eliminando todo lo demás.

El siguiente método es aplicado a cada píxel en la imagen de mostrando el gradiente:

1- En primer lugar se redondea la dirección del gradiente previamente almacenado, denotado por θ. Al valor más cercano a 45º, correspondiente al haber usado una vecindad de ocho elementos.

2- Se realiza una comparación entre la magnitud del borde del pixel actual y la magnitud del pixel tanto en la dirección negativa como positiva. Por ejemplo, si la dirección del gradiente es θ =90º (esto significa norte), compara este valor con los pixeles que se encuentran tanto al norte como al sur.

3- Si la magnitud del borde del píxel que está siendo medido es mayor que en ambas direcciones a las que se ha comparado, se conserva el valor de la magnitud del borde. En caso contrario, se elimina el valor.

En la Figura 13, se muestra un simple ejemplo de cómo conservar todos los máximos. De esta manera, son comparados con los píxeles situados encima y debajo de ellos. Así, los pixeles que son declarados como máximos usando esta comparación, son marcados con bordes blancos. El resto de píxeles por tanto, quedan suprimidos

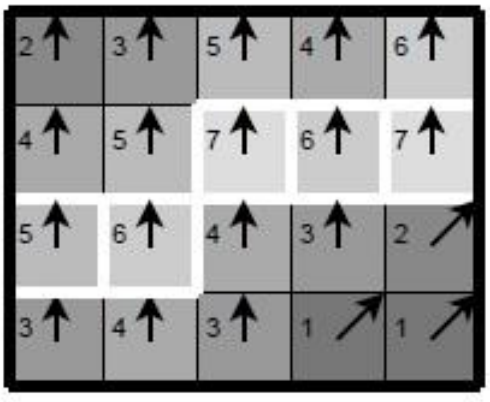


Figura 13: Ilustración de cómo conservar los máximos locales

Las magnitudes de los bordes están indicadas por números y colores, mientras que la dirección del gradiente es mostrada mediante flechas. Los píxeles finalmente marcados como bordes, están representados con sus bordes blancos

- **Doble umbral**

  Lo que aún se mantiene tras mantener los máximos locales, son los píxeles etiquetados con su magnitud, pixel a pixel. Muchos de estos píxeles probablemente formen parte de verdaderos bordes en la imagen principal, pero algunos otros pueden ser causados por ruidos o variaciones de color, por ejemplo cuando se están intentando analizar superficies rugosas.

  La forma más sencilla de elegir entre ellos sería la de usar un umbral, de manera que solo bordes que estén por encima de un cierto valor van a ser conservados como bordes. Sin embargo, en el método que nos ocupa, el denominado *Canny*, se utiliza un doble umbral. Los pixeles que sean parte de un borde y sean mayores del umbral más alto serán clasificados como *fuertes*. Por el contrario, aquellos pixeles que sean menores que el menor umbral serán eliminados. Aquellos pixeles cuyo valor se encuentre entre ambos umbrales serán denominados como *débiles*.

  Aplicando esta clasificación, conseguimos dos tipos diferentes de pixeles. Clasificados como débiles y Fuertes. El resto de pixeles habrán sido suprimidos.

- **Seguimiento de los bordes por histéresis.**

Aquellos bordes que hayan sido clasificados como *fuertes*, son considerados como bordes verdaderos, y pueden ser inmediatamente incluidos en la imagen final de bordes, Aquellos otros que han sido clasificados como *débiles* simplemente son incluidos en la imagen final si están conectados por vecindad con bordes *fuertes*.

Es lógico pensar que el ruido y otras pequeñas variaciones no es muy probable que se encuentren en un borde *fuerte*, si el ajuste del doble umbral ha sido correctamente ajustado.

De esta manera, los bordes Fuertes simplemente serán el resultado de bordes verdaderos en la imagen original. Sin embargo, los bordes *débiles* pueden ser debidos tanto a bordes verdaderos como a variaciones en el ruido o el color. Los *débiles*, estarán distribuidos

independientemente a lo largo de toda la imagen, por lo que solamente una pequeña cantidad estará localizada adyacente a los bordes fuertes. Aquellos bordes débiles que han sido detectados debido a bordes verdaderos, son mucho más probables de estar conectados a bordes fuertes. En el resultado final, el seguimiento de los bordes puede ser implementado usando el análisis denominado BLOB (Binary Large OBject). Usando esta técnica, los píxeles considerados como bordes están divididos en BLOBs, usando una vecindad de 8 elementos conectados alrededor de cada pixel. Cada BLOB que contenga un borde fuerte es entonces guardado, aquellos que no lo contengan, son entonces eliminados.

En resumen, el resultado final será una imagen binaria. Los pixeles en blanco corresponderán a bordes detectados y los negros serán parte de la imagen pero no considerado como borde.

3.2.4 Detección de líneas

A lo largo de toda esta sección se va a explicar la manera en la que se detectan las líneas rectas en la imagen original.. A partir de la imagen binaria, mostrando los bordes del escenario, tendremos que encontrar la técnica más apropiada capaz de detectar aquellas líneas con las cuales, sus puntos de cruce puedan mostrar los puntos de fuga.

Con el fin de descubrir las diferentes líneas que aparezcan en la imagen, la técnica elegida para este propósito ha sido usar la transformada de Hough. Esta técnica será capaz de detectar líneas en la imagen y podría también proporcionar el ángulo de cada línea, por lo que podría ser muy útil clasificar cada línea dependiendo de su ángulo.

En la siguiente sección, van a ser descritas las bases de este método.

- **Transformada de Hough**

  La transformada de Hough es una técnica usada para clasificar las diferentes características de una imagen, y puede ser usada tanto en visión artificial como en análisis de imágenes o procesamiento de imágenes. La principal finalidad de esta técnica es encontrar casos imperfectos de objetos dentro del mismo tipo de formas, mediante un

procedimiento de votos. Este procedimiento es hecho dentro de un espacio paramétrico nuevo, desde donde los candidatos del objeto son encontrados como máximos locales en un espacio acumulativo que ha sido explícitamente construido por este algoritmo, de manera que se puede calcular la transformada de Hough.

La transformada de Hough, en un principio fue usada para identificar las líneas en la imagen, pero más tarde también pudo ser usada para identificar posiciones de cualquier tipo de formas, pero las principales formas para las cuales es usado, son para encontrar círculos y elipses.

En el análisis automático de imágenes digitales, en ocasiones surgen dificultades cuando se intentan detectar formas sencillas, como líneas rectas, círculos y elipses, surge un problema. Muchas veces, el detector de bordes puede ser usado como un paso previo al procesamiento de la imagen para así, conseguir los píxeles de las imágenes, los cuales están situados en los bordes, en el espacio de la imagen. Es posible también, que debido a imperfecciones en los datos de la imagen o imperfecciones en el detector de bordes, se pierdan algunos píxeles o puntos en la zona deseada. También, son posibles desviaciones en el espacio entre los puntos originales de los bordes y los puntos con ruido los cuales son parte de la línea detectada por el detector de bordes.

Por estas razones, a veces no es trivial agrupar todas las características extraídas de los bordes en un conjunto de líneas, círculos o elipses.
El propósito de la transformada de Hough, es manejar este problema haciendo posible clasificar todos los puntos de los bordes en candidatos, a través de un proceso de votación sobre un conjunto de objetos desde una imagen parametrizada.

El caso más simple de la transformada de Hough, es usar una transformación lineal para detectar líneas rectas. En el espacio de la imagen, las líneas rectas pueden ser descritas como y=mx + b, y pueden ser gráficamente representadas para cada par de puntos de la imagen (x, y). Con la transformada de Hough, la principal idea es considerar las características de la línea no como puntos de la imagen, sino en términos paramétricos. En este caso, el parámetro que se refiere a la pendiente *m*, y el que se refiere al corte con el eje de ordenadas *b*.

Por esta razón, la línea recta y = mx + b puede ser representada como un punto (b, m) en el nuevo espacio creado. Sin embargo, las líneas verticales hace incrementar hacia

valores muy altos tanto los valores de m como de b. Por ello, debido a que supone demasiada carga computacional, es mucho mejor parametrizar las líneas mediante la transformada de Hough, que emplearán otros dos parámetros, denominados ρ and θ. El parámetro ρ, representa la distancia entre la línea y el origen, mientras que θ, es el ángulo del vector con respecto al origen.

Usando este tipo de parametrización, la ecuación puede ser escrita como representa la ecuación (7):

$$y = \left(\frac{\cos\theta}{sen\theta}\right)x + \left(\frac{r}{sen\theta}\right) \qquad (7)$$

También puede ser expresado como r= cosθ + ysenθ

En la figura (14), se representan todos los parámetros mencionados anteriormente.

De esta manera, es posible asociar cada línea de la imagen, a un par de coordenadas que representen un punto, el cual es único si θ ϵ [0,π), y r ϵ R, ó si θ ϵ [0, 2π) y r ≥ 0.

El plano compuesto por (r, θ), en ocasiones es denominado espacio de Hough.

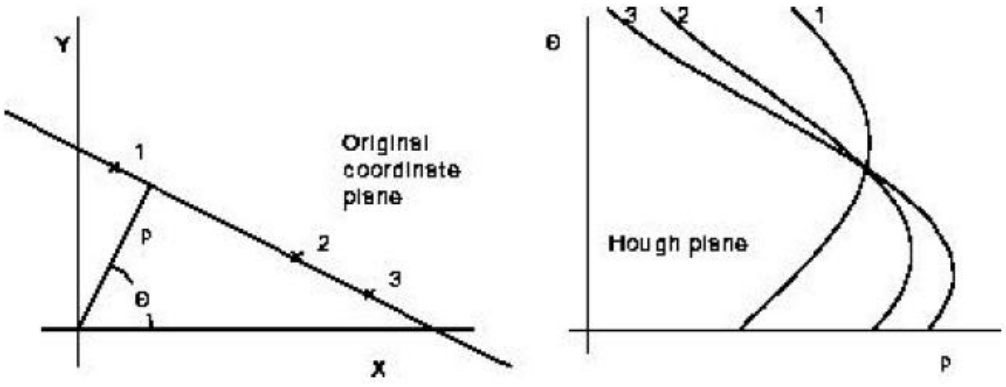


Figura 14: Transformada de Hough. Representación gráfica.

Implementación

La implementación de la transformada de Hough, usa un *array* llamado acumulador, de manera que las posibles líneas puedan ser detectadas. Las dimensiones del acumulador

es el número de incógnitas del problema de la transformada de Hough. Por ejemplo, el problema lineal de la transformada de Hough tiene dos parámetros m y b. Las dos dimensiones de la matriz acumulativa se corresponderán a valores cuantificados de m y b. Para cada píxel y sus vecinos, el algoritmo de la transformada de Hough decide si hay suficiente evidencia como para considerarlo borde en cada pixel. Si es así, calculará los parámetros de esta línea y entonces buscará la celda acumulativa donde corresponden a los parámetros. Mediante la búsqueda del máximo local en el espacio acumulativo, las líneas más probables son seleccionadas y su geometría interpretada.

La manera más simple de encontrar estos máximos, es aplicando algún tipo de umbral, pero diferentes técnicas pueden proporcionar mejores resultados bajo diferentes circunstancias – determinando tanto que líneas ha sido encontradas, como el número de ellas. Las líneas devueltas no proporcionan información acerca de la longitud, a veces es necesario encontrar qué partes de la imagen coinciden con qué líneas. Además de esto, debido a los errores cometidos en la etapa de detección de bordes, aparecerán errores en el espacio acumulativo, los cuales no permitan descubrir de forma trivial los puntos más apropiados y, de esta manera, las líneas adecuadas.

3.2.5 Elección del punto de cruce más apropiado

A lo largo de éste capítulo va a ser explicado en detalle el método usado para elegir los mejores puntos de fuga. Tras haber detectado todas las líneas de los bordes que aparecen en el escenario, debemos encontrar los puntos de fuga. Para conseguirlos, un nuevo método ha sido implementado para hacerlo posible.

En primer lugar, como ya fue explicado en capítulos anteriores, el escenario hacia donde la cámara está apuntando debe ser uno de los cuales se describen y explican en su capítulo correspondiente. Este tipo de escenarios, son los denominados con perspectiva de dos puntos, y no es necesario elegir cuál de ellos es el que va a ser usado antes de ejecutar el algoritmo.

La fotografía original que se usará a modo de plantilla, en la cual todas las explicaciones y aclaraciones van a ser mostradas, es la siguiente.

Figura15: Fotografía original

Cuando se comenzó el estudio acerca de la detección aparecieron una gran cantidad de dificultades de los puntos de fuga en cada imagen. Todos ellos se solventaron aplicando diferentes restricciones, llamados filtros.

El primer filtro usado, es el que nos permita simplemente representar las líneas detectadas que pudieran ser útiles para el posterior cálculo de los puntos de fuga. Se filtrarán todas esas líneas simplemente mostrando las que pertenezcan al rango comprendido entre -60 grados y 60 grados. Usando esto, simplemente serán dibujadas y estudiadas todas las líneas excepto las consideradas como verticales, las cuales no son útiles para este propósito.

La figura 16, muestra el resultado de aplicar la detección de bordes y simplemente serán dibujadas las líneas deseadas sobre la imagen original.

Figura16: Bordes detectados sobre la imagen original.

Los principales problemas que aparecieron a la hora de detectar los puntos de fuga, se explican a continuación:

- **Puntos de cruce considerados como ruido:** Debido que el paso de detección de bordes no es perfecto, en ocasiones este algoritmo detecta líneas las cuales no son bordes, y también en ocasiones puede haber objetos en el escenario los cuales no sean paralelos a los planos que están siendo usados, como pudieran ser paredes y puertas. Debido a esto, simplemente tenemos que preocuparnos y centrar el estudio en las nubes de puntos que pudieran aparecer. Usando el modelo original, todos los puntos de cruce se muestran en la figura (17).

  Redondeados en círculos azules se muestran los puntos que no serán útiles para este estudio, ya que son considerados como ruidosos.

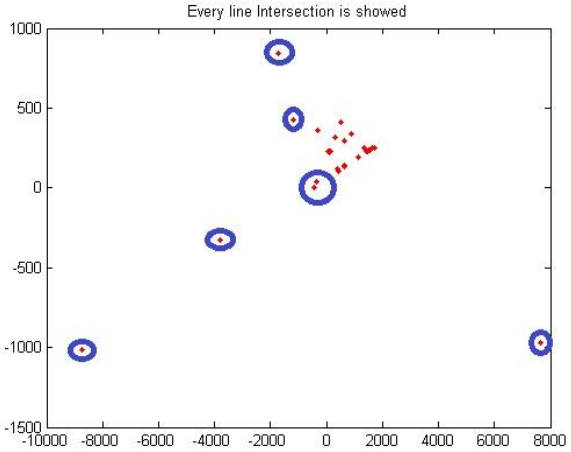


Figura 17: Los puntos inútiles son marcados

- **Puntos de cruce inútiles:** Tras detectar todas las líneas detectadas como bordes que aparezcan en la imagen, y tras haber eliminado los considerados como ruido, todos los puntos de cruce entre todas las líneas rectas van a ser calculados. Para conseguir el objetivo marcado, no todas las intersecciones son útiles, por lo que lo primero que hay que hacer es discriminar entre qué puntos van a ser válidos para este propósito, y qué otros simplemente son puntos de cruce de los que no nos tenemos que preocupar.

  En la imagen, simplemente los puntos útiles están marcados dentro de un círculo azul, lo que significa que cualquier otro punto que no se encuentre redondeado, no va a ser útil para el cálculo de los puntos de fuga. Por ello, van a ser descartados.

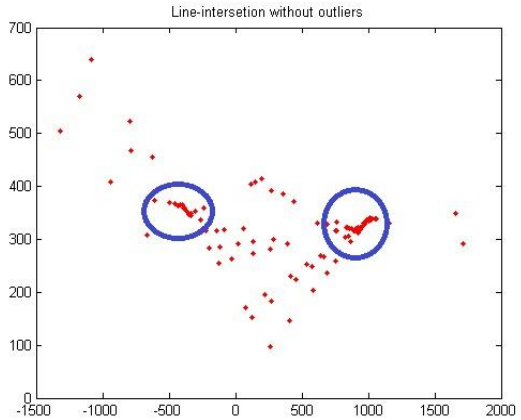


Figura 18: Los puntos útiles son señalados.

- **Dos grupos distintos:** Como ya fue explicado en su capítulo correspondiente, se deben encontrar dos puntos de fuga distintos para conseguir la línea de horizonte. Debido a esto, las diferentes líneas que apuntan a su correspondiente punto de fuga deben ser separados de manera que se puedan dividir los puntos de cruce y entonces, entre estos dos grupos, calcular el punto más apropiado que pueda ser considerado punto de fuga.

  Una vez todos los puntos hayan sido clasificados en dos grupos distintos, también se deben clasificar entre puntos considerados como *inliers* (válidos) o *outliers* (no válidos).

  Aquellos puntos considerados como *inliers*, serán aquellos que se tomen en cuenta a la hora de calcular los diferentes puntos de fuga.

  En la siguiente imagen, todos los puntos han sido ya separados en dos grupos distintos, cada uno de ellos preocupándose acerca de cada punto de fuga. Dentro de cada grupo, los *inliers* son aquellos que están representados en negro.

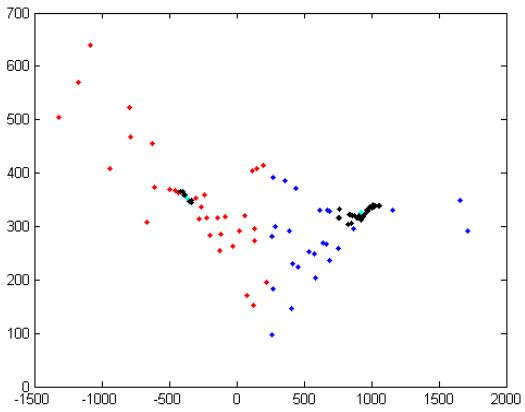


Figura 29: Los puntos han sido separados en dos grupos.

Para solventar todos estos problemas, se ha implementado el siguiente algoritmo:

En un principio, tras haber detectado todas las líneas en el paso de detección de bordes, se dibujan todas las ellas. Usando la función "Hough lines" de MATLAB©, la mayoría de las líneas son detectadas. Esta función también proporciona más información acerca de las líneas detectadas. Esta información consiste en:

- Primer y último punto que componente la línea detectada.

- Ángulo de la línea.
- Longitud de la línea.

Teniendo estos atributos, diferentes métodos se han empleado para solventar cada uno de los problemas que han sido propuestos. Los métodos usados para resolver todos estos problemas, están detallados debajo:

1. **Simplemente una línea por cada línea.**

   En ocasiones, cuando las líneas son detectadas, en vez de encontrar una línea continua y larga en longitud, es dividida en diferentes segmentos de esta línea, más pequeños que la recta original. Para conseguir la mínima cantidad de puntos de cruce, simplemente se va a tener en cuenta un segmento de esta línea. Todas las líneas que tengan el mismo ángulo, van a ser consideradas como partes de la misma línea. Entonces, de entre todas ellas, la línea situada a mayor distancia del medio de la imagen va a ser seleccionada como la representante de esta línea. El por qué esta va a ser elegida, se explicará en el siguiente punto.

2. **Dos grupos distintos:**

   En primer lugar, todas las líneas están divididas en dos grupos. Esto será realizado con el propósito de separar las líneas que apuntan a los diferentes puntos de fuga. Esta separación se ha hecho dependiendo del ángulo que cada línea tiene y en qué posición de la imagen está colocada. Sabiendo, que la mayoría de las cámaras van a estar situadas en lugares donde la línea de horizonte está en torno al centro de la imagen, lo primero que se va a realizar es dividir las líneas que se encuentran por encima de la mitad de la imagen, en la primera mitad, y líneas que se encuentras por debajo, en la segunda mitad.

   El por qué se ha elegido la línea que se encuentra más lejana con respecto al medio de la imagen, es debido a que si elegimos esta línea, será más fácil clasificarlas, ya que habrá líneas que comiencen en una mitad y terminen en la otra. Por lo que, si elegimos las líneas situadas mas lejos de la mitad, incrementaremos el número de líneas correctamente detectadas.

Tras esto, dependiendo en el ángulo de la línea, se verán incluidas en un grupo o en otro.

El criterio seguido para clasificar las diferentes líneas en dos grupos distintos para una vez separados se calcule el punto de fuga, es el siguiente:

Mitad en la que están situadas	Ángulo de la línea	Grupo correspondiente
Primera mitad	Mayor que 0	A
Primera mitad	Menor que 0	B
Segunda mitad	Mayor que 0	B
Segunda mitad	Menor que 0	A

Tabla1: Criterio de clasificación para las distintas líneas

3. Puntos de cruce inútiles

Aquellos puntos que no han sido útiles para nuestra meta, como pueden ser puntos que aparezcan debido a líneas erróneas u objetos los cuales no sean paralelos a los planos elegidos pero estén situados en el escenario y detectados por el detector de bordes van a ser tratados de manera distinta, con el fin de poder separarlos.

Debido a que la etapa de detección de bordes no es perfecta, no todas las líneas paralelas situadas en el mismo plano van a tener la misma dirección. En otras palabras, en el modelo ideal, todas las líneas paralelas van a cruzarse en el mismo punto de fuga, sin error. Sin embargo, al usar el detector de bordes y el detector de líneas rectas, la dirección exacta de la línea nunca va a ser detectada, por lo que el punto de cruce entre todas ellas va a ser el mismo. En vez de esto, lo que vamos a tener es una nube de puntos, resultado de los puntos de cruce entre todas las líneas que son paralelas en el mundo real, pero que se van a cruzar cerca las unas de otras. Sabiendo esto, el punto de fuga correcto va a estar situado dentro de una nube de puntos de cruce, de donde la mejor posición será escogida.

Para resolver este problema, un Nuevo algoritmo ha sido implementado para descubrir las diferentes nubes de puntos presentes en el espacio, donde todos los puntos de cruce han sido representados. Este algoritmo se ha basado en los principios de RANSAC. Debido a la complejidad del algoritmo, una nueva sección es necesaria para explicar paso a paso la estrategia usada para resolver estos problemas.

3.2.6 RANSAC

El algoritmo RANSAC (RANdom Sample And Consensus) fue presentado en 1981 como un método iterativo para estimar parámetros de un modelo, de donde desde todas las muestras disponibles, existe cierta cantidad de datos erróneos que puedan perturbar los datos de entrada.

Está basado en la idea de que todos los datos de entrada pueden ser clasificados como "*outlier*" si no cumple el modelo clasificado como "correcto" formado por los parámetros verdaderos dentro de un intervalo de error, definido por la máxima desviación posible. Los "*outliers*" pueden provenir, por ejemplo, de hipótesis erróneas acerca de la interpretación de los datos o por medidas incorrectas. El porcentaje de *outliers* que RANSAC puede soportar puede ser mayor que la mitad de datos previamente dados.

El algoritmo RANSAC se forma esencialmente siguiendo dos etapas simples, las cuales son repetidas hasta que una cierta condición, previamente fijada, es cumplida. A pesar de las muchas modificaciones hechas sobre el algoritmo, los dos pasos principales son los que se comentan a continuación:

- Hipótesis

  Un modelo es ajustado con algunos hipotéticos *inliers*, por ejemplo, eligiendo aleatoriamente desde un grupo de datos, un mínimo grupo de puntos que pueden ser tratados como *inliers*.

- Test

Tras haber fijado el modelo hipotético, RANSAC comprueba que elementos de todo el conjunto de entrada es consistente con el modelo fijado previamente. Si se ajusta con el modelo previamente fijado, se considera como un *inlier* hipotético.

El modelo, es entonces clasificado como un buen modelo dependiendo de cuantos puntos han sido clasificados como *inliers* hipotéticos.

El modelo, es estimado de nuevo a partir de todos los considerados *inliers*, haciendo un nuevo modelo. Tras haber cambiado el modelo, se evalúa estimando el error de los *inliers* relativos al modelo.

Este procedimiento es realizado un número de veces fijado de antemano, cada uno produciendo modelos válidos con su correspondiente medida de error o modelos inválidos debido a que muy pocos puntos han sido considerados como *inliers* hipotéticos en la etapa de test. Tras haber clasificado cada modelo válido, simplemente lo mantendremos si el último modelo ha conseguido un error menor que el que fue guardado previamente.

RANSAC por tanto, ofrece diferentes ventajas y desventajas. La principal ventaja es que puede encontrar buenas estimaciones con respecto a los parámetros del modelo, incluso cuando se disponen de un número significante de *outliers*, presentes en un conjunto de datos. Una desventaja es que, si se habla acerca del tiempo que requiere para su cálculo, no existe una barrera en el algoritmo. Si se fija un tiempo máximo o un número de iteraciones previamente, quizá la mejor solución no es alcanzada, quizá ni tan siquiera una buena solución es alcanzada.

Siguiendo estos principios, muchos algoritmos RANSAC han sido inventados para resolver el problema que se dispone es este trabajo. El problema reside, en que todos ellos se refieren a modelos de líneas rectas, un modelo que no es el adecuado para este proyecto. En este caso, lo que se está buscando es una distribución de puntos, resultado de dibujar todos los puntos de cruce entre todas las líneas detectadas en el escenario. En vez de una línea recta, lo que se busca es un modelo que pueda buscar *inliers* y *outliers* a partir de una distribución de puntos, e intentar encontrar nubes de puntos, desde los cuales el centro de gravedad se puede encontrar. Este punto será el considerado como punto de fuga.

En la siguiente sección, el algoritmo implementado va a ser descrito y analizado para mostrar cuales han sido las reglas seguidas y el criterio para clasificar o un cierto punto entre *inlier* y *outlier*.

3.2.7 Algoritmo implementado basado en RANSAC

En primer lugar, se deben analizar los datos que se dispone y también fijar el objetivo. Una vez fijados esto dos aspectos, se explicará la metodología usada e implementada en MATLAB©, para conseguir el resultado esperado.

Datos disponibles

Los datos que van a ser recibidos, a la hora de usar RANSAC, son todos aquellos puntos de cruce provenientes de todas las intersecciones entre líneas, tras haberlas encontrado en la fase de detección de bordes. Así, los datos de entrada van a ser un conjunto de puntos distribuidos a lo largo de todo el espacio, pero habrá una nube de puntos donde el punto de fuga deseado va a estar situado.

Propósito

El propósito será, intentar clasificar los puntos de cruce entre todas las líneas rectas en dos grupos diferentes, *inliers* y *outliers*. La mayoría de los *inliers* deberían ser correctos, pero también algún *outlier* incorrecto puede ser aceptado
Los *inliers* deberían ser puntos de cruce donde cada dos líneas rectas se cruzan. Esto dará lugar a una nube de puntos, de donde el punto de fuga será situado.

Algoritmo

Este algoritmo ha sido implementado en MATLAB©, y simplemente necesita un conjunto de puntos de entrada para su funcionamiento. Como ya fue explicado con anterioridad, este algoritmo tratará de buscar la nube de puntos desde donde el punto de fuga pueda ser calculado.

Este algoritmo seguirá los siguientes pasos:

1. En primer lugar, se seleccionará cada punto del conjunto de entrada, calculando para cada uno de ellos la distancia entre este punto y cada uno de los restantes respecto a su posición.

2. Tras ello, se contabilizará el número de puntos que se encuentren más cercanos que un cierto valor, almacenando este valor. Esto será llamado el número de vecinos que cada punto tiene.

3. Finalmente, algunos puntos serán considerados como *inliers* si el número de vecinos considerado es mayor que un cierto valor previamente fijado.

Estos pasos serán dados una vez, y si la nube de puntos ha sido encontrada, esta nube será la que está siendo buscada. En caso de que al final del algoritmo, no se hayan encontrado *inliers*, la distancia desde cada punto hasta donde se considera vecino o no cada punto, va a ser incrementada un cierto valor. Tras este cambio, el algoritmo se ejecutará de nuevo y si se ha encontrado algún *inlier*, se dejará de ejecutar y los *inlier* encontrados serán la nube de puntos buscada.

En caso de que esta distancia de vecindad, alcance un cierto valor previamente fijado, no seguirá creciendo más. Lo que se realizará será decrementar el número de vecinos necesitados para considerarlos *inliers*. Esto es debido a que a veces no hay suficientes puntos de cruce en total, como consideramos anteriormente, y esto es por lo que el número de vecinos tiene que ser decrementado.

Tras ejecutar este algoritmo, todos los *inliers* serán encontrados y la nube de puntos será formada. Entonces, una vez hallada la nube de puntos, simplemente calculando el centro de gravedad entre todos los puntos, el punto de fuga será encontrado.

La siguiente imagen muestra el resultado tras haber ejecutado este algoritmo. Éste fue ejecutado dos veces, una para cada grupo de puntos. Los dos grupos diferentes están representados en rojo y azul. Entre ellos, algunos *inliers* fueron encontrados y representados en negro.

El centro de gravedad final, y el considerado punto de fuga, ha sido representado en cyan.

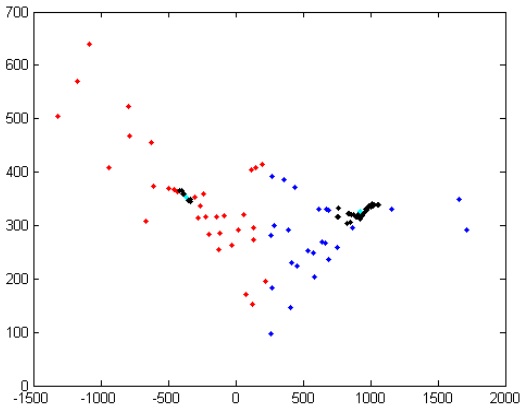


Figure 20: Resultado de ejecutar el algoritmo RANSAC

3.2.8 Estimación de los puntos de fuga

La línea de fuga, también llamada línea de horizonte ya que es exactamente lo que se está representando en la imagen. Esta línea siempre está situada a la altura de los ojos del que realiza la fotografía, por lo que puede aportar una idea acerca de la altura en la que la cámara está situada y desde dónde el observador está mirando.

Como ya fue explicada en la sección de conocimientos previos, esta línea tendrá una importancia crucial cuando se quiera medir la altura de un objeto. Las relaciones calculadas que hacen uso de esta línea serán parte de las bases de este algoritmo. Simplemente una ligera variación en la dirección de esta línea, modificaría todas las medidas, y su precisión se vería severamente reducida.

Si se quiere representar la línea de fuga, lo único que se debe hacer es unir ambos puntos de fuga que hayan sido previamente hallados y la línea resultante será la línea de horizonte que se buscaba.

La siguiente figura, la número 21, representa la línea de horizonte a la que se ha hecho mención. Las líneas azules representan los bordes que han sido detectados y todas ellas van a cruzarse teóricamente en el mismo punto. Entonces, calculando ambos puntos de fuga, los cuales no pueden ser representados ya que se encuentran fuera de los límites de la imagen, y uniéndolos se conseguirá la línea de horizonte.
En esta figura, se representa en verde y refleja perfectamente la altura de la cámara en el momento en que se tomó la fotografía.

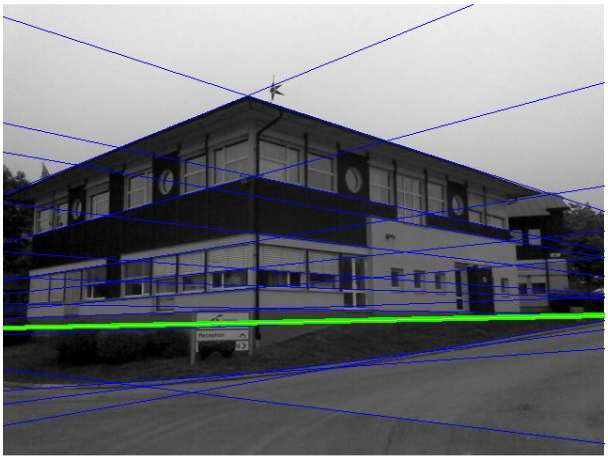


Figura21: Representación de la línea de fuga.

Basadas en esta línea, todas las proyecciones pueden ser hechas y el cálculo de la altura de las personas puede ser realizado automáticamente.

3.3 Detección de personas

3.3.1 Introducción

La detección de personas es, al igual que la geometría proyectiva, una de las partes principales en este proyecto. Se ha señalado en varias ocasiones que la única condición requerida en esta técnica es saber una altura de referencia en el escenario. Pero también hay que destacar que también se requiere saber en qué lugar del escenario se encuentra la persona a la que se dispone medir en todo momento. Una vez que la persona sea detectada y capturada por la cámara, se dispondrá de una longitud en pixeles la cual puede ser usada para calcular la verdadera altura de esa persona. En conclusión, es necesario detectar a la persona mientras está pasando a través del escenario. Por ello, en primer lugar, será importante y necesario hacer una breve introducción acerca de la detección de personas en el ámbito de la visión artificial, y saber por qué es tan importante hoy en día.

El rastreo de personas está basado en el rastreo del vídeo, el cual puede ser definido como el proceso de localizar un objeto en movimiento (o múltiples objetos) a lo largo del tiempo usando una cámara [4]. El propósito del rastreo en vídeo es asociar ciertos objetos en frames consecutivos del vídeo. Estos objetos serán encontrados en la imagen y sus coordenadas estarán disponibles para cualquier uso que pudiera ser considerado útil para posibles nuevas aplicaciones que pudieran ser hechas basadas en este seguimiento.

La estructura general de este seguidor, está básicamente basado en las siguientes fases. Más pasos pueden ser añadidos pero para el propósito de este proyecto, con la simple aplicación de estos pasos será suficiente:

- Segmentación del movimiento: está considerado como un intento de detectar regiones en la imagen que se correspondan a objetos en movimiento. Estas regiones son realmente útiles ya que mucho tiempo computacional puede ser ahorrado debido a que solamente se centra el estudio en una parte en concreto de la imagen, en la cual el

movimiento se está produciendo. La estrategia que se va a seguir es el uso de la eliminación del fondo, la cual será presentada más adelante.

- Seguimiento: es importante distinguir el seguimiento con respecto a la segmentación. Cuando hablamos de seguimiento, no estamos intentando identificar los objetos en movimiento, sino identificar el mismo objeto, llamado modelo, en los diferentes frames disponibles.

En este proyecto, el sistema de seguimiento está básicamente basado en la segmentación del movimiento. Esto es debido al hecho de que no estamos realmente interesados en calcular trayectorias.

3.3.2 Diagramas de flujo

La siguiente figura es un simple diagrama acerca del sistema de seguimiento usado, en el cual se muestran tanto sus datos de entrada como de salida.

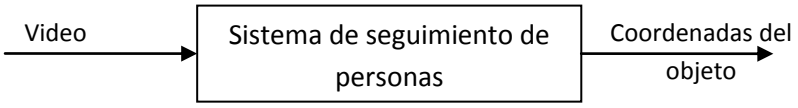


Figura 22.Esquema del sistema de detección de personas

Este es un sencillo diagrama de flujos acerca del seguimiento de personas, mostrando todos los pasos que cumple para hallar las coordenadas finales que serán devueltas.

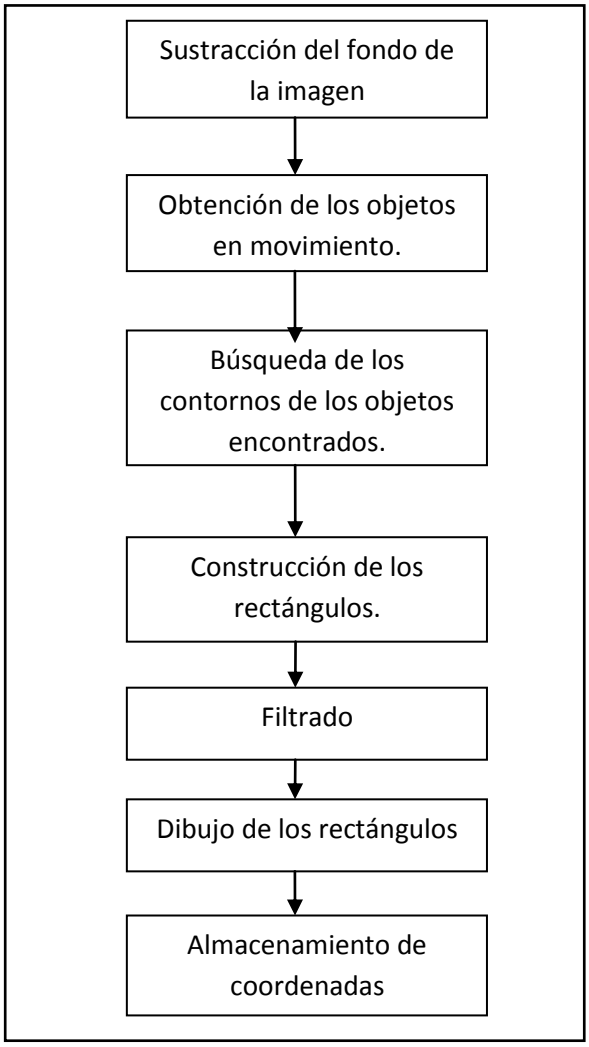


Figura 13. Sistema de seguimiento de personas

3.3.3 Sustracción del fondo. Método del cálculo de la media sobre el fondo.

La sustracción del fondo de la imagen es una técnica capaz de diferenciar objetos respecto de un fondo el cual contiene ambos para poder permitir el análisis y hacer un seguimiento de ellos o detectar situaciones predefinidas o sospechosas. Hay varias técnicas para implementar la sustracción del fondo, pero es recomendado antes que nada, presentar los métodos existentes de implementación y mencionar cuales son los diferentes problemas que se pueden presentar en este campo.

El principal objetivo de este software será que dada una secuencia de imágenes tomadas desde una cámara fija en una cierta posición, tratar de detectar todos los objetos que se encuentren en primer plano. Básicamente, la manera usada para detectar estos objetos que no forman parte del fondo se hará restando el frame actual que va a ser objeto de estudio con una imagen, la cual represente un fondo estático.

Hay al menos tres maneras de estimar el fondo de una imagen [5]:

1. *Diferencia entre frames:* El fondo estimado que será restado más tarde será simplemente el frame anterior. Este método no es muy útil para el propósito buscado, ya que simplemente es capaz de detectar movimiento en las imágenes, como pudiera ser una pierna o un brazo en movimiento. Pero el propósito es detectar la forma completa de la persona. Además, se requiere que sea independiente a condiciones particulares como la velocidad del objeto y la frecuencia de muestreo de la cámara. Estas son las principales razones por las que no ha sido posible aplicar esta simple técnica al software.

2. *Media de los n-frames previos:* Usando esta técnica, el fondo va a ser calculado como una media de entre un cierto número de frames correspondientes al fondo, cuando no se produce movimiento en los frames, ni hay objetos o personas pasando por la escena que va a ser considerado como fondo perturbando la escena. Este método es bastante rápido, pero se necesitan algunos requerimientos de memoria para guardar los primeros frames.

3. *Media continua*: la fase de cálculo del fondo es hecho en cada frame, actualizando la imagen de fondo usando una tasa de aprendizaje, α. Este método no necesita requerimientos de memoria.

$$acc(x, y) = \alpha * imagen(x, y) + (1 - \alpha) * acc(x, y)$$

Donde *acc(x, y)* es la media acumulada de las imágenes, que posteriormente será usada como modelo

En este proyecto, este último método ha sido el elegido para su uso. Ha sido elegido ya que es rápido y aporta muy buena precisión. La estimación del fondo simplemente es realizada durante los primeros diez frames, no a lo largo de todo el vídeo. Esta estrategia podría ser considerada como una mezcla entre los últimos dos métodos. De esta manera se consigue velocidad, precisión y solamente es realizado en una ocasión, al principio, y durante diez frames.

En vez de sucesivas restas entre cada frame y su predecesor, la actualización del fondo ha sido usado para restar en cada iteración el frame actual con la imagen actual del fondo. Este método genera varias mejoras en comparación con la anterior implementación, la más importante es que usando este método es posible conseguir figuras sólidas y no simplemente siluetas como las que se disponían antes. También permite seguir detectando una persona aunque no esté en movimiento durante un tiempo hasta que forme parte del fondo.

La única desventaja que presenta, es que este método introduce algo de ruido, pero las ventajas ofrecidas aún así hacen mejor su uso que el método anterior, ya que el ruido introducido es prácticamente inapreciable.

En el siguiente esquema, se muestra el método empleado, para la estimación del fondo del escenario. Se muestran los frames descartados y los empleados para la estimación del fondo a lo largo de los frames.

Las razones por las que se descartan algunos frames, será detallada más adelante.

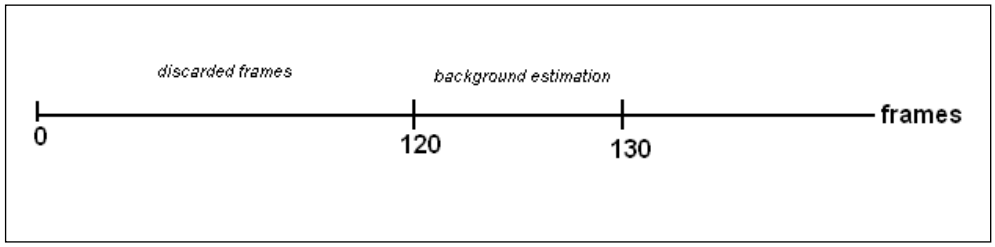


Figura 24: Esquema de la estrategia empleada

3.3.4 Obtención de los objetos en movimiento

Esta es una técnica sencilla la cual consiste en rellenar la silueta completa del objeto en movimiento a partir de los segmentos detectados, usando los procesos morfológicos clásicos, como pueden ser la dilatación y la erosión.

Estos segmentos detectados son los provenientes de restar el fondo y el frame actual, y estos segmentos, en ocasiones no están conectados unas con otros. Debido a este hecho, la forma de este objeto no es uniforme y la mayoría de las partes están desconectadas., es por ello por lo que es necesario procesar cada imagen, para disponer de un solo segmento uniforme y conectado.

La operación que se debe realizar si queremos conectar todos estos puntos es la llamada operación de cerradura. Con esta operación morfológica, los límites desconectados van a conseguir cerrarse y los huecos dentro de la silueta van a desaparecer. Los operadores morfológicos usados para hacerlo posible son la erosión y la dilatación. Para hacerlo posible, dos funciones de OpenCV han sido usadas, las cuales son cvDilate y cvErode. Su uso y finalidad van a ser explicadas en sección llamada “Desarrollo software”.

3.3.5 Búsqueda de contornos y rectángulos envolventes

Un contorno es un conjunto de puntos que representa, de una manera u otra, una curva en la imagen [6]. Esta operación será útil para este trabajo, ya que nos permitirá trabajar con la silueta humana como un objeto independiente y bien definido. Para esta tarea, se usarán algunas funciones de OpenCV, como puede ser cvFindContours, el cual calcula los contornos a partir de imágenes binarias.

Tras haber obtenido los contornos de los objetos en movimiento, el software procesará cada silueta y conseguirá un rectángulo que sea capaz de envolver a cada uno de ellos. Esta operación también está soportada por la librería de OpenCV, por lo que simplemente habrá que escoger las funciones adecuadas en cada momento, sin la necesidad de centrarse en implementarlas por uno mismo.

Con ello, a partir de ahora, lo que disponemos son varios rectángulos los cuales ya se ajustan a la silueta del cuerpo humano. De ahora en adelante, se prestará atención tanto a los diversos problemas que pueden aparecer debido a diferentes tipos de ruido o vibraciones, como a los filtros usados para solventar cada uno de estos problemas. Estos problemas aparecerán debido a la detección de contornos, los cuales no se correspondan con una silueta humana, y si lo es, puede que no esté detectando al individuo completamente en el escenario, ya que puede que tanto los pies como la cabeza queden fuera del ángulo de visión de la cámara.

3.3.6 Filtrado

Durante la etapa de sustracción del fondo, cuando el modelo del escenario se quiere crear con el fin de restarlo de cada imagen para conseguir el objeto en primer plano, pueden aparecer muchos tipos de contratiempos correspondientes al ruido. A continuación se describirán y se explicarán brevemente los más importantes:

- **Cambios debido al movimiento:** Estos cambios aparecen debidos a la cámara. En las capturas de vídeo realizadas para hacer pruebas, aparecen ciertas vibraciones durante los primeros frames de las capturas, debido a la acción de pulsar el botón. Estas vibraciones aparecen durante los primeros dos o tres segundos, ya que es debido a una acción mecánica. También pueden aparecer algunas vibraciones si la cámara no está perfectamente fijada a la pared, o en el lugar donde esté situada.

- **Cambios en la iluminación**: La diferente luz que pueda provenir desde las ventanas o luces que produzcan sombras pueden ser apreciados por la cámara como un tipo de movimiento debido simplemente a los cambios en la iluminación. Si hay un cambio en la luz presente en el escenario, por ejemplo la presencia de nubes, podrían hacer considerar al algoritmo como que la ventana es un objeto en movimiento. Además, a veces el suelo puede reflejar mucho la luz y puede actuar como un espejo, generando problemas en la detección de la silueta de la persona.

La manera elegida de solucionar este problema de vibraciones, ha sido aplicar un tiempo de *offset*, unos pocos segundos, para descartar estos frames, los cuales son considerados como erróneos. En este caso, un *offset* de ciento veinte frames ha sido el elegido. La figura (24), anteriormente presentada indica un esquema de la medida tomada.

Para solventar los problemas de ruido debidos a los cambios en la iluminación, el filtro creado está basado en las proporciones humanas de la silueta humana. De acuerdo al hombre de Vitrubio creado por Leonardo Da Vinci [7], el cual asegura que las proporciones del cuerpo humano son de 3/8, en relación al alto y el ancho de una persona normal. Con el fin de dar cierta flexibilidad al software, se ha empleado un rango de proporciones aceptables, las cuales van desde 1/3 hasta 3/8. El valor de estos límites se ha basado en la experiencia sobre el terreno, usando muestras de vídeo y analizando cada una de ellas.

Además, en estas proporciones se han ha tenido en cuenta la posibilidad de que el sujeto se encuentre de perfil, y no siempre de frente.

Este tipo de ruido aparece durante todo el tiempo, no simplemente en la fase de estimación del fondo. Por ello, el algoritmo para comprobar las proporciones entre el alto y el ancho es usado en todos los frames y el rectángulo que se ajusta a la forma humana simplemente será dibujado cuando estas proporciones se cumplan.

Figure 25. Rectángulo rojo ajustándose a la forma humana.

Sin embargo, este filtro no es tan preciso como lo necesitamos. A veces algunos rectángulos que se ajustan a la condición de proporciones son detectados, pero no son el resultado de una forma humana. En ocasiones, ciertas ventanas o puertas casualmente pueden cumplir estas proporciones. Pero, el objetivo de este algoritmo no es el de encontrar objetos en movimiento, sino personas en movimiento.

También se debe saber, que en ocasiones las ventanas detectadas no están en movimiento, sino que debido a los cambios de luz o sombras, pueden actuar como tales.

Figura 26. Rectángulo detectado indeseado.

En la figura 26, hay un claro ejemplo acerca de los rectángulos que no se ajustan a la relación impuesta, por lo que serán descartados.

Con el propósito de solucionar el problema de las figuras cuadradas, se ha implementado y usado otro algoritmo en este proyecto. Se aplica tras el comentado previamente, y está basado en el número de pixeles en movimiento que hay dentro de un rectángulo detectado. El algoritmo consiste en contar el número de pixeles en blanco que hay en una ROI (Region Of Interest) de una imagen.

Una Región de interés, es un área rectangular dentro de una imagen, para segmentar objetos para futuros procesamientos. Usando esta ROI, mucho tiempo de computación puede

ser ahorrado, ya que simplemente se analiza una parte de la imagen, no la imagen al completo, la cual requeriría un análisis de una cantidad de datos muy elevada. En la figura 27, se muestra un ejemplo de la región denominada ROI:

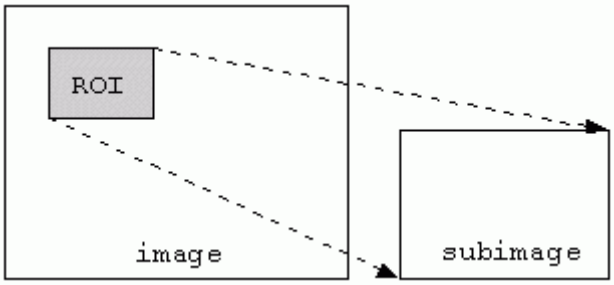


Figura 27: Región de interés

ROI tiene una fantástica importancia práctica, ya que en muchas situaciones acelera las operaciones de visión artificial, permitiendo simplemente el análisis de pequeñas sub-regiones de la imagen.

En la figura 28, sin embargo se representan con pixeles blancos los objetos en movimiento en la imagen actual. Esta imagen es conseguida tras haber hecho la diferencia entre el modelo de fondo creado y el frame actual. En el software de demostración, lo encontraremos en la venta llamada “Final”.

Figura 28: Ventana “*Final*”, mostrando el movimiento de los objetos en blanco

El siguiente problema que se nos presenta, consiste en evitar posibles formas que puedan ajustarse al rango de proporciones que se ha limitado en un principio. Una buena manera de distinguir un rectángulo no deseado de otro, el cual se ajusta a las proporciones previamente

mencionadas, es tratando de modelar la forma humana con algún tipo de figura geométrica básica.

En este caso, se ha implementado un algoritmo el cual dibuja una elipse el cual se ajusta perfectamente al interior del rectángulo pintado.

Una vez dibujada la elipse, se contabilizarán los pixeles en blanco (pixeles en movimiento). Si el rectángulo detectado se ajusta al cuerpo de un humano, la gran mayoría de los pixeles se encontrarán dentro de esta elipse

Esta técnica ha sido llevada a cabo siguiendo los siguientes pasos:

- Una vez creado el rectángulo, se define éste como la ROI y la cantidad de pixeles en blanco de su interior son contados.
- Entonces se dibuja la elipse que se ajuste perfectamente al rectángulo.
- Al no ser posible definir la ROI como una elipse, esta elipse se dibujará totalmente en negro, cambiando los pixeles blancos de su interior por pixeles en negro. Contando los pixeles en blanco restantes, conseguiremos saber los que se encuentran fuera de la elipse.
- Sabiendo los píxeles totales, y los que se encuentran fuera de la elipse podremos calcular qué porcentaje de pixeles están dentro de la elipse del total.
- Ahora, simplemente aplicando un umbral para que la zona sea aceptada como persona detectada, se podrán evitar objetos como puertas, ventanas, etc.…

  En este caso se ha elegido el 95 por ciento de los pixeles.

En la figura 29, se pueden apreciar los distintos pasos mencionados anteriormente:

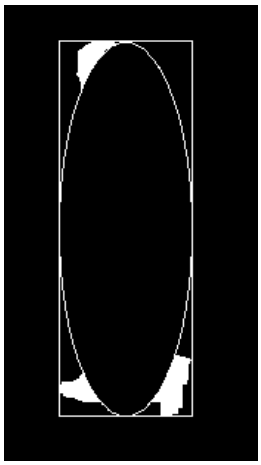


Figura 29 Elipse ajustándose al rectángulo.

Es de necesaria mención, que estos operadores, los cuales definen regiones de interés y el contar píxeles en blanco son realizados directamente con funciones previamente creadas en OpenCV, las cuales van a ser presentadas más adelante.

Además de los filtros mencionados, se han empleado otros filtros o condiciones para descartar rectángulos erróneos. Estos filtros no son tan complejos como los anteriormente presentados, pero estos filtros "simples" también tienen una importancia crucial en el proyecto.

El primero se encargará del área del rectángulo detectado. Éste es empleado para descartar pequeños rectángulos que puedan surgir debido a vibraciones. En la figura 30, se puede observar un ejemplo de este tipo de ruido.

Otra condición que debe ser cumplida es que el objeto que quiere ser medido, en este caso las personas, tienen que estar en un rango de alturas de entre 160 y 210 cm, si no cumplen este punto, será considerado como defectuoso y erróneo, por lo que será descartado.

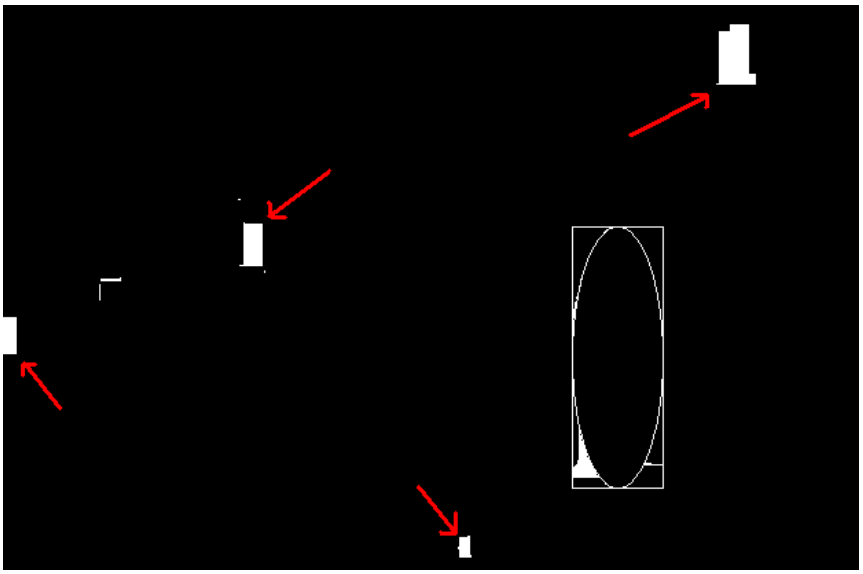


Figura 30. Pequeños rectángulos debido a las vibraciones

3.3.7 Formación de rectángulos y almacenamiento de coordenadas.

Una vez los rectángulos detectados en cada frame hayan sido filtrados y cumplan las condiciones, estos son pintados en el vídeo y las coordenadas guardadas. Los únicos puntos necesitados para medir la altura en pixeles del rectángulo son tanto el que se encuentra en el punto más alto como el que se encuentra más bajo. Tras ello, estos datos serán introducidos en el algoritmo encargado de la medida de alturas, y se procederá al cálculo de la altura de la persona.

En la figura 31, se muestra un esquema el cual a grandes rasgos representa la interconexión entre ambos sistemas:

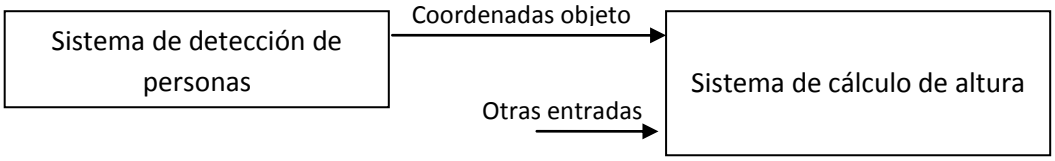


Figura 31. Interconexión entre ambos sistemas

3.3.8 Limitaciones

Este sistema de detección tiene algunas limitaciones. En la siguiente sección todas ellas van a ser descritas y las razones por las que aparecen serán explicadas.

La principal restricción que aparece, es aquella por la cual la persona a la que se quiere detectar, debe de estar situada en el escenario de tal manera que todo su cuerpo sea

visible para la cámara. Es necesario saber las coordenadas tanto del punto más alto (la cabeza) como el punto más bajo (los pies) para tener la longitud en pixeles de la persona en la imagen. No es correcto por tanto, si la cabeza o los pies no están en la imagen, por lo que estos frames son descartados.

También, cabe recordar que como ya se detalló en la sección de principios teóricos, se requiere que la persona esté en el mismo plano que el objeto medido como referencia. En otras palabras, la parte más baja del objeto de referencia tiene que estar en el mismo "suelo" que la persona que está pasando por delante de la cámara.

En ocasiones, debido al efecto de espejo en el suelo y a la diferencia de colores entre el suelo y los zapatos de la persona, el software puede no detectar bien la parte baja de la silueta humana. Este es un aspecto el cual se podría mejorar, intentado borrar las sombras de las personas para conseguir mejores resultados. Esto no ha sido realizado en este proyecto ya que no era una de las metas. No obstante esta parte puede ser mejorada en trabajos futuros.

En cuanto a las vibraciones, aparecen como otro punto débil ya que introducen gran cantidad de ruido a la hora de hacer la diferencia entre las imágenes. Sin embargo, este punto no es tan crítico, ya que si pensamos en sistemas de circuito cerrado de vídeo, no van a aparecer ya que las cámaras están inmóviles y bien fijadas a sus puntos de apoyo.

Como ya se ha mencionado con anterioridad, para tratar de descartar valores erróneos en la medida debido a valores extremos, el software está limitado a medir alturas estándar.

Desde 160 a 210 cm. Pero si este rango se quiere cambiar para aumentarlo, es bastante sencilla su modificación.

Tampoco se ha de olvidar que la entrada de la altura de referencia se hace a partir de clics del ratón, realizado a mano y por simple apreciación visual, por lo que también se introducirá cierto ruido. Para tratar de reducir esto en la medida de lo posible, se requiere al usuario de la aplicación que seleccione la altura de referencia dos veces, para posteriormente tomar el punto medio de ellos.

4 Desarrollo software

A lo largo de este capítulo se va a realizar un estudio pormenorizado del desarrollo del software. En primer lugar, los lenguajes de programación que han sido elegidos serán descritos. En segundo lugar todas las funciones creadas para el funcionamiento del algoritmo serán explicadas. Finalmente, un esquema global será mostrado para conseguir una visión general de la estructura del software.

4.1 Lenguajes de programación

A lo largo de esta sección, se van a presentar los lenguajes de programación y algunas librerías usadas, junto con las razones por las que se han elegido unas frente a otras.

El programa principal que se ha implementado ha sido en C++, pero para hacer operaciones específicas, el motor MATLAB© ha sido usado para diferentes propósitos. El cómo hacerlos funcionar de manera conjunta y llamar a uno desde el otro, será explicado con detalle al final de este capítulo. Junto a este capítulo, se añadirá un apéndice donde todas las instrucciones acerca de cómo configurar ambos programas. Antes de entrar en detalle a los lenguajes usados, se va a realizar una breve explicación del por qué de cada lenguaje elegido:

- C++: Es un lenguaje de programación óptimo, ya que dispone de alta velocidad a la hora de tratar con altas cifras. Sin embargo, no es sencillo a la hora de producir efectos visuales, como gráficas o dibujar líneas, por lo que otro programa debe ser elegido para realizar estas tareas.

- MATLAB©: Es un lenguaje basado en scripts, el cual la conversión a objetos es realizada en tiempo de ejecución y es por ello por lo que puede presentar en ocasiones algo de lentitud. Por otra parte, es excelente a la hora de representar gráficas y prototipos. También dispone de paquetes especiales para el tratamiento de matrices y vectores de una forma sencilla e intuitiva.

Sabiendo todas estas características, queda demostrado que el uso de MATLAB© y C++ de manera conjunta puede resultar una forma muy útil de desarrollar el software.

En las secciones siguientes, cada lenguaje de programación va a ser presentado y descrito.

4.2 C++

C++ es un lenguaje de programación diseñado en los ochenta por Bjarne Stroustrup, de los laboratorios Bell de AT&T. Él intentó mejorar el mejor lenguaje de programación que estaba disponible en aquella época, este lenguaje era C.

C apareció en los setenta, y fue diseñado por Dennis Ritchie, con el propósito de programar en sistemas operativos UNIX.

Tras varios años, otro programador llamado Bjarne Stroustup, introdujo el ahora denominado C++. Su objetivo era extender el exitoso lenguaje de programación llamado C, para ser capaz de usarlo manejando objetos y clases, facilidades las cuales pueden ser encontradas en otros lenguajes pero que C, no era capaz de soportar todavía. Para hacer esto posible, C fue rediseñado aumentando sus posibilidades pero manteniendo sus características más importantes, permitiendo al programador tener bajo control todo lo que está realizando, debido a esto C es uno de los lenguajes de programación más rápidos. Por estas razones, C++ en el campo de lenguajes de programación orientados a objetos, es considerado un lenguaje híbrido.

En referencia a las clases y objetos, por el momento simplemente es importante saber que es un sistema que intenta acercar los lenguajes de programación a la comprensión humana, basándose en la construcción de objetos, con sus propias propiedades y agrupados en clases.

Una de las particularidades de C++, es la posibilidad de llamar a dos funciones distintas por el mismo nombre. La única manera entonces que el compilador tendrá para decidir qué función está siendo llamada es mediante el número de argumentos de entrada.

Hoy en día, debido al éxito y extensión de C++, existe un estándar llamado ISO C++, en el cual las compañías de compiladores más importantes y modernas se han unido para mejorar este lenguaje.

Los elementos necesarios para usar este lenguaje son:

- Un equipo trabajando bajo un cierto sistema operativo.
- Un compilador C++:
    - Si estamos usando Windows: deberíamos usar *MingW*.
    - En entornos UNIX, se debe usar *g++*.
- Cualquier editor de texto, o si es posible, usar un entorno de desarrollo (IDE) como:
    - Para Windows:
        - *Notepad (*No recomendado*)*
        - *Notepad++ editor*
        - *DevCpp (*también incluye *MingW)*
        - *Code:Blocks*
    - Para UNIX:
        - *Kate*
        - *KDevelop*
        - *Code: Blocks.*
        - *SciTE.*

También es necesario mencionar, que para C++ existe una gran cantidad de librerías, cada una de ellas dedicada a diferentes campos, dependiendo de su finalidad.
La librería que ha sido usada para este proyecto ha sido la llamada OpenCV.

OpenCV es una librería libre acerca de visión artificial, originalmente desarrollada por Intel. Desde la primera versión que apareció a principios de 1999, ha sido usada para infinidad de aplicaciones. Desde sistemas de seguridad con detección de movimiento hasta aplicaciones que requieren reconocimiento de objetos. Esto es debido a que su publicación ha sido producida bajo licencia BSD, que permite su libre uso para propósitos tanto comerciales como de investigación.

Las primeras versiones no fueron definitivas, simplemente versiones beta se hicieron públicas entre 2000 y 2005. La primera versión definitiva apareció en 2006. Entonces, recibió el apoyo de Willow Garage, un laboratorio de investigación, el cual provee mas desarrollo e investigación activa en ello. Una nueva versión fue sacada entonces al mercado, denominada versión 1.1.

La segunda versión que apareció más importante apareció en Octubre de 2009. OpenCV2 incluye cambios para el uso de este con C++. OpenCV es un software multiplataforma, existiendo versiones para LINUX, Mac OS y Windows. Contiene más de 500 funciones que engloban una gran cantidad de distintas áreas, acerca de la visión artificial, como puede ser el reconocimiento de objetos (reconocimiento facial), calibración de cámaras, visión estéreo y visión artificial.

El objetivo de este proyecto es intentar aportar una herramienta de fácil uso y alta eficiencia. Esto ha sido posible programándolo sobre C y C++ de una forma optimizada, aprovechándose de las ventajas que nos ofrecen los procesadores multi-núcleo. OpenCV, también puede usar las instrucciones primitivas y específicas integradas en los procesadores Intel. OpenCV intenta ayudar a la gente a construir sofisticadas aplicaciones de visión artificial de una manera rápida y sencilla.

La libertad de OpenCV ha sido estructurada de tal manera que se pueden crear productos comerciales usando tantas funciones como sean necesarias, sin la obligación de que el producto creado sea de libres derechos, por lo que no es necesario reportar parte de los beneficios que pueda producir este producto a los creadores de las librerías.

4.3 MATLAB©:

MATLAB© es un software matemático que ofrece su propio Entorno de Desarrollo (IDE), incluso con su propio lenguaje de programación, llamado M. Está disponible para UNIX, Windows y plataformas Apple Mac OS X.

Fue creado en los setenta por Cleve Moler, jefe del departamento de informática en la universidad de Nuevo Méjico. Intentó diseñar un nuevo lenguaje capaz de usar LINPACK y EISPACK sin la necesidad de saber Fortran. Este software fue rápidamente expandido a otras universidades y fue muy bien recibido por la comunidad de matemáticas aplicadas.

En los ochenta, Jack Litte, ingeniero industrial, se unió a Moler, ya que se dio cuenta del tremendo potencial comercial que tenía el producto. Ambos implementaron MATLAB© en C, y fundaron Mathworks en 1984.

MATLAB© entonces se convirtió en un lenguaje para computación técnica y alto rendimiento. Integra visualizaciones, cálculos complejos y un entorno fácil de usar donde los problemas y soluciones son expresados en una notación matemática familiar y sencilla. Los usos típicos para los que MATLAB© es usado son:

- Matemáticas y computación de datos.
- Algoritmo de desarrollo.
- Modelado, simulación y creación de prototipos.
- Análisis de datos, exploración y visualización de los mismos.
- Gráficas científicas.

El elemento básico de MATLAB© es una matriz, la cual no necesita ser inicializada o dimensionada al principio de un programa. Esto dota de la posibilidad de solventar una gran cantidad de problemas computacionales, tomando especial atención a los problemas en la formulación de las matrices y vectores.

En 2005, MATLAB© era usado por más de un millón de usuarios, divididos en campos tanto educativos como académicos. También es usado como una de las herramientas más potentes para los investigadores.

MATLAB© también ofrece una Application Program Interface, la comúnmente denominada API. Esta librería permite escribir programas en Fortran y C que puedan interactuar con MATLAB©. Incluye la posibilidad de llamar rutinas desde MATLAB©, para usar MATLAB© como un motor de cálculo desde fuera del propio MATLAB©, también puede ser usado para escribir ficheros .MAT.

La librería está originalmente escrita en C, pero existen envolventes para lenguajes como Ruby, C##, Phyton y JAVA, con el fin de llegar al máximo de audiencia posible.

4.4 Funciones generadas

A lo largo de todo este capítulo, todas las funciones que han sido generadas para el desarrollo del software van a ser descritas y expuestas.

Como se ha podido apreciar, durante toda la memoria presentada hasta el momento, en muchas ocasiones varias funciones han sido mencionadas debido a su utilidad y la importancia que tienen en este proyecto a la hora de realizar diferentes cálculos. Todos los datos de entrada necesitados para cada función y los datos resultantes van a ser detalladas a continuación.

Esto será útil para futuros trabajos en los cuales alguien pudiera necesitar alguna de las funciones creadas para este proyecto y le pueda permitir el ahorro de mucho tiempo a algún otro investigador que esté intentando hacer algún tipo procesamiento de imágenes o incluso para intentar mejorar el trabajo que se ha realizado para intentar superar los resultados obtenidos.

Todas las funciones usadas en C++, provenientes de la librería OpenCV simplemente van a ser mencionadas. Si se requiere más información acerca de estas funciones, se pueden encontrar en el siguiente libro [6]

En un principio, todas las funciones referentes al procesamiento de imágenes y el intento de búsqueda de los puntos de fuga van a ser descritos. Todos ellos han sido implementados en MATLAB©.

Las más importantes son las detalladas a continuación:

- Nombre: "funcCalcVpoints_MOD"

  Descripción

  Dentro de esta función y llamando a algunas otras que serán descritas más adelante, esta función va a ser capaz de encontrar de una imagen correspondiente al primer frame la línea de horizonte, a través de los puntos de fuga calculados.

  Datos de entrada:

  - Cadena de caracteres de la ruta: Proveniente del primer frame previamente almacenado.

  Datos de salida:

  - Una línea, situada en el plano z=1, que representará la línea de horizonte.

- Nombre: "my_RANSAC"

  Descripción

  Es llamada dentro de la función "funcCalcVpoints_MOD". A partir de un grupo de puntos, que serán los correspondientes a todos los puntos de cruce de un mismo punto de fuga, a partir de un radio dado y un umbral de elementos, este algoritmo va a ser capaz de clasificar todos los datos entre *inliers* y *outliers*. Además, de entre los *inliers* calculará el punto de fuga deseado. Los principios teóricos de esta función han sido explicados en su sección correspondiente.

  Datos de entrada

- Conjunto de puntos de cruce correspondientes a un punto de fuga.
- Distancia considerada como vecindad para tratar el resto de puntos de cruce como vecinos o no.
- Mínimo número de vecinos para considerar un punto de cruce como *inlier*.

Datos de salida:

- Un conjunto de puntos considerados como *inliers*.
- El índice de cada punto, clasificándolo con 1 si es *inlier* o con 0 si no ha sido considerado *inlier*.
- Centroide final calculado entre todos los puntos considerados como *inliers*.

- Nombre: "dist"

Descripción

Situada dentro de "my_RANSAC", será capaz de calcular la distancia en píxeles entre dos puntos de cruce.

Datos de entrada:

- Coordenadas de los dos puntos de cruce a estudiar.

Datos de salida:

- La distancia euclídea entre los dos puntos, simplemente siguiendo la relación:

$$\sqrt{(x-y)}^2 = |x-y|$$

Nombre: “funcCalcHeight”

Descripción

Usando la anteriormente explicada geometría proyectiva, este algoritmo va a ser capaz, a partir de las coordinadas de altura de referencia y las del objeto a medir, calcular la altura detectada.

Datos de entrada:

- Línea de fuga la cual será usada para calcular los puntos de cruce.
- Puntos superior e inferior del objeto detectado a partir del detector de objetos y movimiento implementado en C. Estas coordenadas estarán representadas en coordenadas pixel. Las coordenadas X e Y son necesarias.
- Puntos superior e inferior del objeto de referencia situado en el escenario. Estas coordinadas serán introducidas manualmente por el usuario, a través de clics del ratón sobre la imagen y seleccionando el objeto. También será necesario introducir la distancia de referencia, la cual fue previamente medida en el escenario real.

Datos de salida:

- Altura del objeto detectado que más tarde será mostrado por pantalla y dibujado sobre la persona en movimiento.

Una vez que todas las funciones MATLAB© han sido explicadas, es el momento de mostrar que funciones de OpenCV han sido usadas durante este trabajo.

Simplemente un breve explicación del propósito de ellas será mencionado, para poder entender el funcionamiento global del software. Como fue indicado con anterioridad, si se desea saber más acerca de las funciones usadas, el libro de referencia puede usarse con este fin.

A continuación se muestran las funciones más importantes usadas de OpenCV:

- cvNamedWindow: Con esta instrucción C++ abre una ventana y muestra una imagen por pantalla.
- cvMoveWindow: Esta funcuón permite mover una ventana eligiendo las coordenadas de la esquina superior izquierda de la ventana.
- cvCaptureFromFile: Usando esta, inicializamos la estructura CvCapture para leer los datos provenientes del fichero de video.
- cvCreateVideoWriter: Crea una estructura para la escritura en un fichero de video.
- cvQueryFrame: Cada vez que es llamado, un frame es tomado desde el fichero de entrada.
- cvCloneImage: Como su propio nombre indica, sipmlemente copia una imagen a partir de un fichero fuente a un fichero de destino.
- cvSaveImage: Almacena en el disco duro la imagen seleccionada.
- cvInitFont: A través de esta función, se podrá seleccionar la fuente a usar en la pantalla.
- cvPutText: Usando esta función , seremos capaces de escribir texto en la imagen deseada.
- cvShowImage: Usada para mostrar una imagen dentro de la ventana deseada.
- cvSetMouseCallback: Fución que registra todos los eventos provenientes del ratón.
- cvRunningAvg: Empleada para realizar una media en tiempo real de las imágenes para mas adelante eliminar el fondo del escenario.
- cvAbsDiff: Diferencia entre dos imágenes en valor absoluto.
- cvCvtColor: Función empleada para convertir de un espacio de color a otro. En este caso, para cambiar de RGB a escala de grises.

- cvThreshold: Usando este umbral, se podrán descartar pixeles por debajo o por encima de un cierto umbral. En este caso solo se descartarán valores por debajo de un cierto umbral.

- cvDilate: Función encargada de realizar dilatación sobre la imagen binaria.

- cvErode: Función encargada de realizar dilatación sobre la imagen binaria. Combinando dilatación y erosión seremos capaces de aplicar una apertura.

- cvFindContours: Esta función permite unir los pixeles situados en los contornos de la imagen.

- cvBoundingRect: Devolverá un rectángulo que rodee el contorno detectado.

- cvSetImageROI: Usado para seleccionar la región de interés (ROI) en la imagen, esta ROI es usada simplemente para analizar simplemente esta parte de la imagen y ahorrar mucho tiempo computacional.

- cvCountNonZero: Función empleada para contar el numero de píxeles que no son cero, dentro de la ROI.

- cvResetImageROI: Aplicando esto, la ROI será de nuevo la imagen al completo. –

- cvEllipse: Usada para definir una elipse en la imagen.

- cvRectangle: Usada para crear un rectángulo en la imagen.

- cvWaitKey: Esta función hará que el progrma espere durante un cierto numero de milisegundos hasta que le usuario presione una tecla.
- cvReleaseImage: Libera la parte de memoria usada tras haber usado la imagen.

- cvDestroyWindow: Tras haber usado una ventana, siempre es recomendable destruirla para liberar los recursos tomados del sistema operativo.

Existe otro tipo de sentencias que deben ser explicadas en detalle, ya que no son triviales o apmliamente usadas. Estas instrucciones son las correspondientes al control del motor MATLAB© desde C++, y las más importantes que ahn sido usadas son las sigueintes:

- En un principio, una librería se debe añadir, incluyendo en el código "#include<engine.h>".
- Una vez incluida, mediante la instrucción "Engine *ep;", se crea un puntero hacia el motor MATLAB©.
- engEvalString(Engine* ptr, string cmd) : Con esta función, usando el puntero al motor creado anteriormente, podemos introducir cualquier comando que queramos que se evalúe en el motor MATLAB©.
- engGetVariable(Engine *ep, const char *name):Toma desde el motor apuntado, el valor de la variable que está almacenada en un tipo mxArray
- mxGetPr(pm): A partir de los datos almacenados en una estructura mxArray, se transforma a una variable del entrono C++.
- engCLose(ep): Usando esta, se permite a C++ abandonar la sesión del motor MATLAB©.

5 Resultados

5.1 Introducción

En esta sección, se van a analizar los resultados obtenidos para comprobar el grado de precisión de los algoritmos desarrollados. Sabiendo que este proyecto no está orientado a expertos programadores y debido a que el principal propósito de este proyecto no es el de conseguir la mejor precisión en los algoritmos tanto de detección de personas como en el cálculo de los puntos de fuga, sino el de combinar varios campos de conocimiento para el desarrollo de una aplicación útil e independiente. Teniendo en cuenta estos aspectos acerca del alcance del proyecto presentado, es justo reconocer que, como se verá más adelante, los resultados obtenidos son bastante satisfactorios.

A partir del software creado, una nueva línea de trabajo ha sido creada con posibilidades de crecimiento y mejora. Si algún grupo de investigación desea estar interesado en profundizar sobre el trabajo realizado, las posibilidades comentadas en la sección de líneas futuras de trabajo pueden ser útiles como guía de trabajo.

El resultado de este proyecto es un software robusto, sencillo y de rápida ejecución, capaz de ser implementado en cualquier plataforma. No obstante, es posible su mejora, por ejemplo haciendo un exhaustivo trabajo de depuración orientado al tiempo de ejecución del programa.

Aunque el código a simple vista parezca como código estructurado, en vez de un lenguaje orientado a objetos, este hecho proporciona mayor velocidad al algoritmo, además de permitir también, el uso de librerías OpenCV, ya que sólo están disponibles para C++.

5.2 Análisis de los resultados

Con el fin de poder analizar los resultados obtenidos una vez finalizado el algoritmo, se van a analizar dos vídeos a modo de prueba, asimismo todas sus características serán comentadas.

A partir de cada vídeo, se procederá a un análisis de los datos que ha proporcionado a la salida el algoritmo, que como ya se ha indicado con anterioridad en esta memoria simplemente son mostrados por pantalla, aunque con una simple modificación en el algoritmo van a poder ser utilizadas estas medidas para cualquier otro fin.

Para cada vídeo se van a analizar el número de frames en los que se ha detectado a la persona y por tanto se ha procedido a realizar la medición. De esta manera, se presentarán los datos recogidos en un histograma para poder ver una distribución de las medidas realizadas. Todas las medidas que se van a analizar son en centímetros.

En ambas ocasiones el sujeto que se encuentra en el escenario es el mismo, y cuya estatura es de 185 centímetros. Hay que tener en cuenta que el sujeto no se detiene delante de la cámara sino que se encuentra andando en el escenario. De esta manera, nunca se va a producir la misma medida, ya que la altura de las personas se verá afectada por el cambio de la posición de las piernas al andar. Una vez explicados todos los elementos que pueden alterar nuestra medida, es posible comenzar el análisis de los vídeos seleccionados.

En el primer vídeo, que tiene una duración de 20 segundos, se han realizado 57 medidas. Hay que tener en cuenta que los primeros tres segundos siempre se suprimen para evitar fallos debidos a las vibraciones y que no todos los frames son adecuados para la medición, ya que el sujeto no aparece por completo en el escenario.

A continuación se muestran dos fotogramas del vídeo a modo de muestra, para saber el tipo de escenarios y sujetos que están siendo analizados:

Figura 32: Fotograma del primer escenario

Figura 33: Sujeto detectado en el escenario

El siguiente histograma muestra la distribución de las medidas:

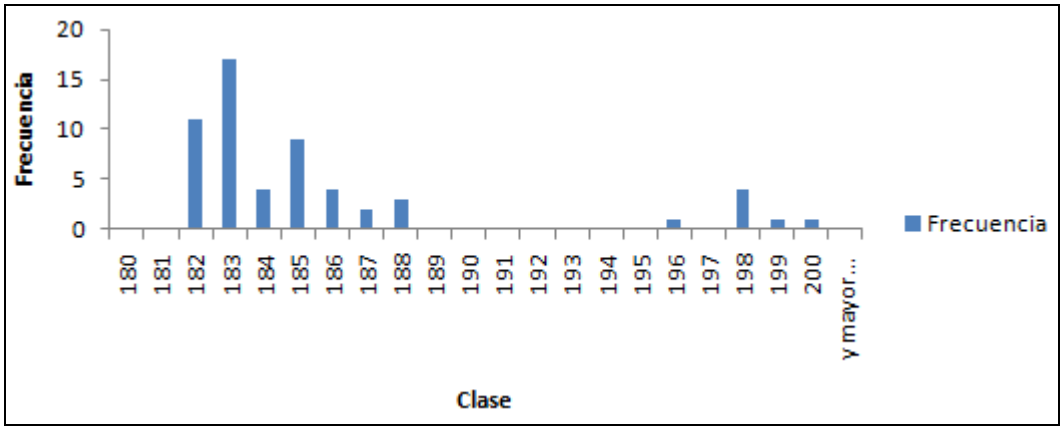


Figura 34: Histograma correspondiente a las medidas realizadas en el primer vídeo

Como se puede apreciar la mayoría de las medidas se sitúan en torno a la medida real, que como ya se indicó es de 185 centímetros. De hecho, en torno a un error de ±3 centímetros se encuentran el 87,7% de las medidas. Este error de ±3 centímetros corresponde a un error del 1,6 %. En cuanto a la medida con mayor error detectado, corresponde al 8%.

En cuanto al segundo vídeo, de una duración de 17 segundos se han realizado 103 medidas. En este caso, a pesar de ser más corto que el vídeo anterior se han realizado más medidas ya que el sujeto se ha encontrado un número mayor de veces totalmente visible en el escenario.

A continuación, al igual que se hizo en el caso anterior, se muestran dos fotogramas como muestra del vídeo analizado:

Figura 35: Fotograma del segundo escenario

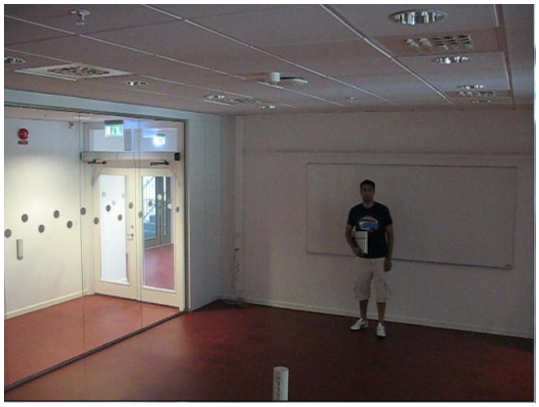


Figura 36: Sujeto detectado en el escenario

El siguiente histograma muestra la distribución de las medidas en el segundo vídeo detectado:

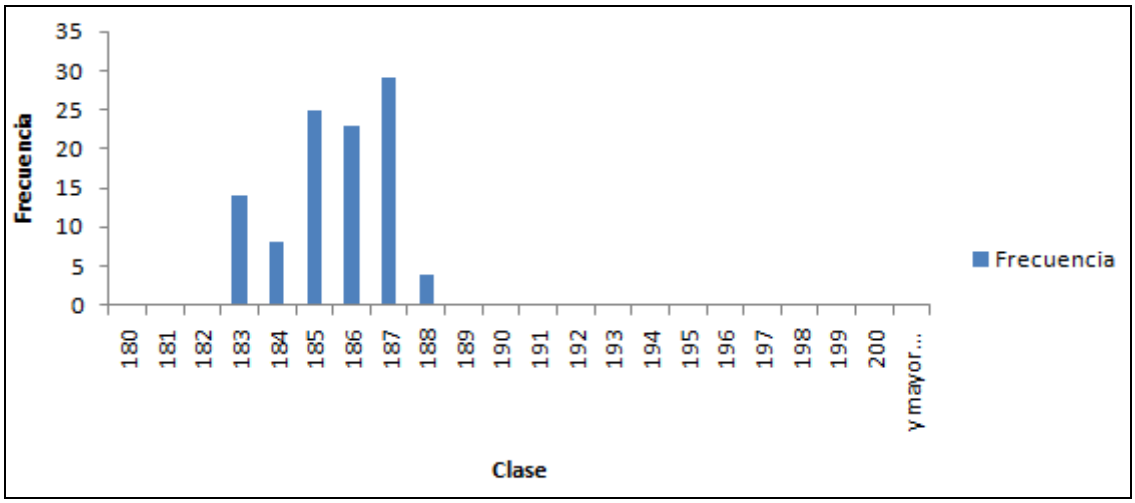


Figura 37: Histograma de las medidas tomadas en el segundo vídeo

En este segundo vídeo, como se puede comprobar en su correspondiente histograma mostrado en la figura 37, todas las medidas tomadas, es decir el 100% de ellas, se ajustan al error de ±3 centímetros impuesto con anterioridad, que supone el 1,6% de error.

6 Conclusiones y futuras líneas de investigación

6.1 Conclusiones

De acuerdo a los datos analizados en el apartado anterior, los objetivos logrados en este proyecto han sido ampliamente alcanzados, incluso sabiendo que la posibilidad de profundizar en el algoritmo y optimizado es posible.

Se ha conseguido crear una aplicación robusta e independiente, la cual pueda ser fácilmente exportable a cualquier máquina, para su posterior uso. Por ello, podría ser incluso comercializado, con una gran aceptación en el mercado actual.

Como conclusión se ha de decir que este proyecto ha supuesto un gran reto personal, ya que nunca había trabajado con librerías de OpenCV así como programas realizados bajo el lenguaje C++.

El uso de MATLAB© y C++ de manera conjunta ha supuesto una gran éxito para crear una solución rápida en ejecución y sencilla en la implementación. Por ello, el tiempo empleado para configurar ambos software para conseguir el final resultado, ha sido productivo, a pesar de los problemas encontrados.

6.2 Futuras líneas de investigación

Esta sección va a intentar mostrar los caminos a seguir en el futuro que se deben seguir si nuevos investigadores quieren profundizar en el software para una optimización en su funcionamiento . También se van a presentar algunas posibles aplicaciones de este software.

Como ha sido descrito a lo largo de esta memoria, la finalidad de este proyecto ha sido el unir diferentes y diversos campos de conocimiento y hacer una aplicación con todos ellos. No se ha tratado de buscar el mejor método de detección de personar, o el mejor detector de puntos de fuga, simplemente se ha intentado mostrar que sin la necesidad de tener un gran conocimiento en programación, es posible construir un algoritmo bastante robusto, capaz de realizar una tarea muy útil.

Si este proyecto se considera útil e interesante para siguientes investigadores, y quiere ser continuado para su mejora, o incluso comercializado, en la siguiente sección todas las características que puedan mejorar este algoritmo serán descritas. Estas mejoras han sido mencionadas en cada parte de la memoria, pero a continuación se analizan en detalle y se aporta más información.

Como se ha comentado con anterioridad, este proyecto está dividido en dos grandes campos. El primero de ellos es el encargado de detectar personas y el otro es el encargado de detectar tanto los puntos de fuga como la línea de horizonte.

En un principio, se van a presentar las posibles posibilidades para perfeccionar la detección de personas:

- El principal aspecto que se puede perfeccionar, es tratar de evitar las sombras de la imagen. Este aspecto es crucial, ya que al disponer de la máxima precisión del lugar en el que se encuentran los pies del sujeto, se realizará una medida de la altura de la persona con mucha mayor calidad.
  Este aspecto no ha sido realizado en este proyecto ya que puede emplear mucho tiempo encontrar el algoritmo adecuado, y no era uno de los objetivos del proyecto. Si somos capaces de eliminar todas las sombras de la imagen, muchas nuevas formas humanas van a ser detectadas y por tanto nuevas mediciones van a ser realizadas.

- Otra cuestión importante podría ser el intentar evitar las vibraciones de la cámara, usando algún algoritmo que pudiese reducirlas y por ello introducir menos error en las imágenes.

- Si un método de sustracción del fondo más complejo es usado, los resultados obtenidos podrían incrementar su precisión. En este proyecto, un algoritmo bastante sencillo ha sido usado, por lo que si se pudiera implementar otro más complejo o usar uno que ya esté creado, podría mejorar nuestras medidas de una manera significante.

Una vez presentadas todas las líneas de perfeccionamiento que pueden ser seguidas a la hora de detectar a los sujetos que puedan aparecer en el escenario, se van a explicar unos cuantos aspectos acerca de la detección de puntos de fuga.

En primer lugar, se debe remarcar que el algoritmo creado trabaja de una manera exitosa, incluso sabiendo que su complejidad no es muy elevada.

Los principales puntos a mejorar, son los mostrados a continuación:

- El algoritmo empleado tiene que funcionar con escenarios donde la mayoría de las líneas detectadas muestren la dirección de las líneas de fuga. En el caso de que muchas líneas correspondientes a objetos denominados como "ruido" apareciesen, podrían perturbar la detección de puntos de fuga.
- La etapa de detección de bordes todavía necesita ser optimizada, ya que usando las técnicas actuales se ha alcanzado un buen resultado, pero si es posible mejorar esta parte del software, pueden ser obtenidos tanto mayor precisión, por tanto, óptimos resultados

Destacando los principales atributos que pueden ser modificados para optimizar este software, próximos investigadores serán capaces de encontrar fácilmente el camino a seguir, sin tener que emplear mucho tiempo en descubrir los puntos a mejorar.

En cuanto a posibles usos en el futuro, en este proyecto simplemente se muestra el resultado final por pantalla, simplemente para demostrar su validez y fiabilidad. Pero, muchas otras aplicaciones pueden ser hechas usando este software.

Si en vez de mostrar cada medición por la pantalla, se decide almacenar todos ellos y tras ejecutar el programa analizarlos, se podrá realizar, por ejemplo, un estudio acerca de la población en un cierto país o ciudad.

La otra ventaja principal que puede ser tomada de aplicar todas las ideas propuestas, es hacer la aplicación en tiempo real. Si finalmente se consigue implementar este software en

tiempo real, muchas aplicaciones pueden ser halladas. También, tras haber ejecutado esta aplicación en tiempo real, es posible la fácil modificación del código para cambiar el objetivo a medir, cambiando una persona por un posible vehículo o animal.

Apéndice 1: Instalación de OpenCV con Microsoft Visual C++ 2008

Como ya se mencionó anteriormente, este proyecto ha sido llevado a cabo con la ayuda de muchas funciones incluidas en OpenCV. Para empezar a trabajar con OpenCV, en un principio es necesario instalarlo y hacerlo trabajar con el IDE que se está usando para la generación del código, en este caso Microsoft Visual Studio 2008. Este apéndice es un intento de aclarar y guiar a través del proceso que se debe seguir para configurar cada software. También existen tutoriales y blogs en internet, que pueden ayudar en este apartado. Los más recomendados son [8], [9], [10] y [11]

Es de necesaria mención que el OpenCV usado es la versión 2.1, y que el proceso de instalación puede variar dependiendo en la versión de las librerías de OpenCV, al igual que la versión IDE.

A continuación todas las acciones a realizar y los pasos necesarios para llevarlos a cabo van a ser enumeradas. Se realizará de un modo sencillo y guiado para evitar posibles confusiones.

Paso 1: Instalación de OpenCV

1. Descargar *OpenCV 2.1. Windows Installer*.
2. Instálese en una ruta primaria, por ejemplo “"C:\OpenCV2.1\". Hay que tener en cuenta que es importante el no introducir espacios en el nombre de la ruta.
3. Actívese la opción “Add OpenCV to the system PATH for all users” a lo largo de la instalación.
4.

Paso 2: El paquete de instalación de OpenCV, no incluye librerías pre-compiladas para Visual Studio. Por lo que es necesario crearlas usando *CMake*.

4- Descárguense los ficheros binarios de CMake e instálense. No importa la localización.

5- Ejecute la herramienta CMake GUI y configure OpenCV allí.

6- Seleccione *"C:\OpenCV2.1\"*, en el campo "Where is the source code".

7- Créese un directorio llamado *vs2008* en "C:\OpenCV2.1\" para almacenar los ficheros generados en el proyecto. ("C:\OpenCV2.1\vs2008").

8- De nuevo, dentro de CMakeGUI, en "Where to build the binaries", elíjase el directorio creado, (*"C:\OpenCV2.1\vs2008"*).

9- Se presiona el botón *Configure* y se elige "Visual Studio 9 2008".

10- Se ajustan las opciones.

11- Se presiona de nuevo Configure y entonces se selecciona *Generate.*

Paso 3: En este momento se configura Microsoft Visual Studio para el uso de OpenCV.

12- Se abre la solución creada "C:\OpenCV2.1\vs2008\OpenCV.sln".

13- Se construye el proyecto tanto en modo debug como en release.

14- *Se debe añadir "C:\OpenCV2.1\vs2008\bin\Debug"* and "C:\OpenCV2.1\vs2008\bin\Release" al path del sistema. (Una breve explicación de cómo se hace este paso es mostrado al final de este apéndice).

15- Dentro de *Tools–>Options–>Projects–>VC++ Directories–>Library files*, se añaden los siguientes ficheros:

"C:\OpenCV2.1\vs2008\lib\Release"

"C:\OpenCV2.1\vs2008\lib\Debug"

16- En *Tools–>Options–>Projects–>VC++ Directories–>Include files* se añade: "C:\OpenCV2.1\include\opencv"

17- En cada proyecto en el que se use OpenCV es necesario incluir en: *Project->Properties->Linker->Input->Additional Dependencies* los siguientes ficheros, separados por espacios:

cvaux210.lib

cv210.lib

cxcore210.lib

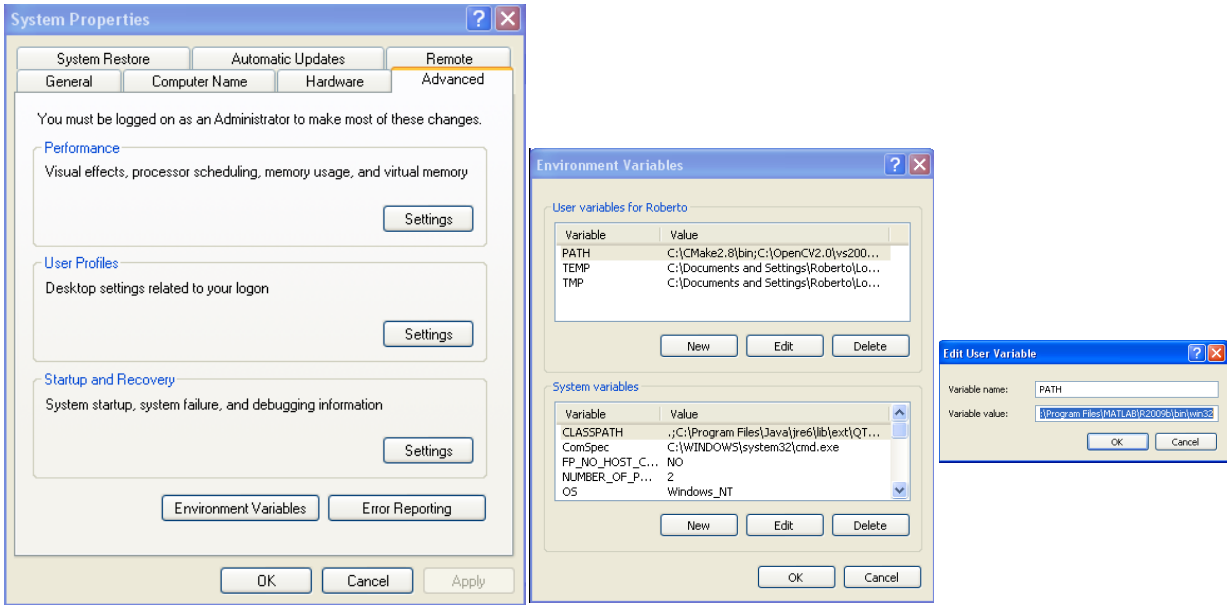
highgui210.lib

*Cómo añadir ficheros al path del sistema

Para realizar esta tarea simplemente hay que seguir el siguiente paso:

1- Clic con el botón derecho en *My Computer->Properties->Advanced->EnvironmentVariables->Path->Edit.* Añadir path al fichero en este caso:

"C:\OpenCV2.1\vs2008\bin\Debug" and *"C:\OpenCV2.1\vs2008\bin\Release"*



Apéndice 2: Cómo configurar Microsoft Visual C++ para llamar al motor MATLAB©

Como ya se comentó anteriormente, este proyecto ha sido desarrollado con Microsoft Visual C++ 2008 usado conjuntamente con MATLAB© R2009b. Tras dedicar gran parte del tiempo a configurar ambos programas para poner en contacto VC++ y MATLAB©, ha sido menester crear un apéndice dedicado a nuevos investigadores, explicando paso a paso la manera en la que han sido configurados ambos software.

En primer lugar, es importante aclarar qué cantidad y de qué manera se pueden comunicar VC++ y MATLAB©.

- **Llamando programas C desde MATLAB©:** Es posible llamar rutinas C o C++ desde la línea de comandos de MATLAB©. Estos programas son los denominados ficheros-MEX y pueden ser compilados desde MATLAB© o desde código C/C++. Son llamados desde MATLAB©

- **Llamando funciones MATLAB© desde C como DLLs:** Usando el compilador MATLAB© (MCC) para construir una librería compartida (DLL) desde un fichero m. Esta es la manera más recomendada, pero no la que vamos a seguir, ya que no es la más fiable y más difícil de utilizar. Si se consigue crear el fichero DLL a partir del fichero MATLAB©, será óptimo debido a las siguientes razones [12]:
    1- Con el fin de acelerar la ejecución de los programas, el código compilado en C o C++ es más rápido a la hora de ejecución que su fichero M equivalente.

2- Óptimo para crear aplicaciones independientes. Estas aplicaciones se aprovechan de las funciones de MATLAB©, pero no requieren de su presencia para ejecutarse.

3- Para crear librerías compartidas C o librerías estáticas en C++. El compilador deberá ser usado para crear una librería compartida o una librería estática en C++ a partir de un algoritmo desarrollado en MATLAB©.

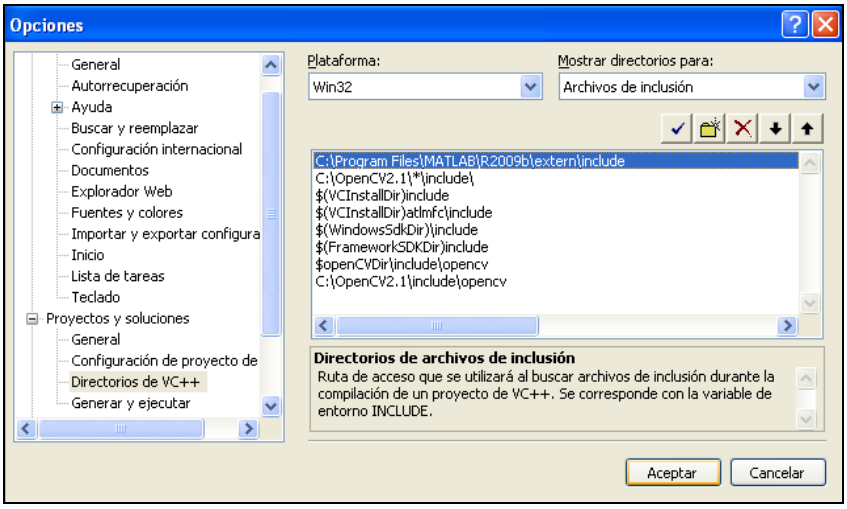
- **Llamando el motor MATLAB© desde programas C**: Las librerías del motor MATLAB© contienen rutinas que permiten llamar al software MATLAB© desde los propios programas creados en C, de esta manera MATLAB© es usado como un motor de cálculo. Los programas que usan este motor, son aplicaciones independientes en C/C++ que se comunican con procesos MATLAB© a través de pipes en sistemas UNIX, y a través de la interfaz Microsoft Component Object Model (COM), en sistemas Windows. El motor MATLAB© [13] opera ejecutándose en un proceso independiente al programa en C.

Este último método es el que ha sido usado. Para poder usar MATLAB© de esta forma, es necesario configurar algunas opciones en VC++. A partir de este momento, se explicará esta configuración paso a paso. Las rutas y algunas opciones pueden variar dependiendo de la versión del compilador y de MATLAB©. Por lo que, este es el procedimiento usado para configurar Microsoft Visual Studio 2008 version 9.0.21 y MATLAB© R2009b version 7.9.0.

A continuación se indican los pasos a seguir, aunque a grandes rasgos se ha seguido el siguiente tutorial [14].

1- Al crear rutas a MATLAB©, deberíamos añadir algunos ficheros include.

Tools->Options->Projects and Solutions->VC++ Directories. Seleccionar "Include files" desde la lista y añadir: *"C:\...\MATLAB\R2009b\extern\include".*

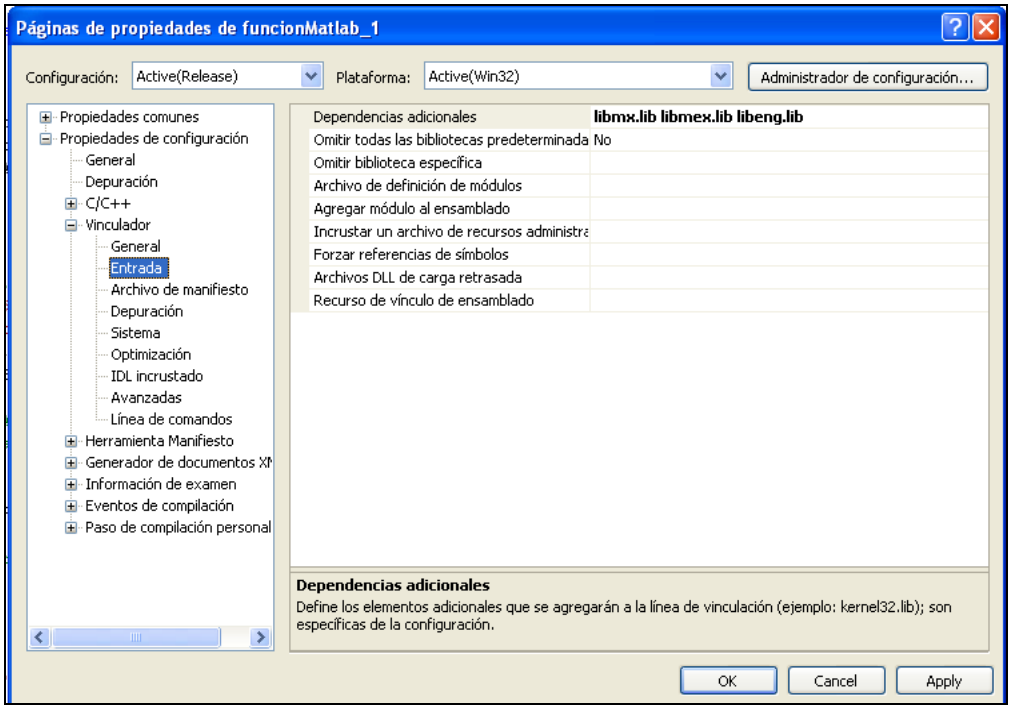


2- Inclusión de las librerías apropiadas:

Project properties (clic derecho en current Project) *->Linker->Input->Additional dependencies* , añadir tres nombres de ficheros:

- libmx.lib
- libmex.lib
- libeng.lib

Hay que destacar que estos "includes" no funcionarán si las correspondientes librerías dinámicas no existen en el fichero \system32. Por lo que, por ejemplo, si se necesita disponer de la librería libmx.dll en el fichero "*C:\WINDOWS\system32",* se debe conseguir la siguiente pantalla:


Páginas de propiedades de funcionMatlab_1
Configuración: Active(Release)
Plataforma: Active(Win32)
Administrador de configuración...
Propiedades comunes
Propiedades de configuración
General
Depuración
C/C++
Vinculador
General
Entrada
Archivo de manifiesto
Depuración
Sistema
Optimización
IDL incrustado
Avanzadas
Línea de comandos
Herramienta Manifiesto
Generador de documentos XM
Información de examen
Eventos de compilación
Paso de compilación personal
Dependencias adicionales libmx.lib libmex.lib libeng.lib
Omitir todas las bibliotecas predeterminada No
Omitir biblioteca específica
Archivo de definición de módulos
Agregar módulo al ensamblado
Incrustar un archivo de recursos administra
Forzar referencias de símbolos
Archivos DLL de carga retrasada
Recurso de vínculo de ensamblado
Dependencias adicionales
Define los elementos adicionales que se agregarán a la línea de vinculación (ejemplo: kernel32.lib); son específicas de la configuración.
OK
Cancel
Apply

Referencias

[1] Joseph L. Mundy and Andrew Zisserman, *Appendix-Projective geometry for machine vision*. Editors, Geometric Invariance in Computer Vision,. MIT Press, Cambridge MA, 1992.

[2] A. Criminisi, I. Reid and A. Zisserman, *Single View Metrology.* Kluwer Academic Publishers, 2000.

[3] J. *Canny, A. Computational Approach to Edge Detection*. IEEE Computer Society, 1986.

[4] Wikipedia web page: http://en.wikipedia.org/wiki/Video_tracking

[5] Massimo Piccardi, *Background subtraction techniques: a review.* The ARC Centre of Excellence for Autonomous Systems (CAS), 2004.

[6] Dr. Gary Rost Bradski and Adrian Kaehler, *Learning OpenCV*. O'Reilly Media, Inc., 2008.

[7] Wikipedia web page: http://en.wikipedia.org/wiki/Vitruvian_Man

[8] Web page: http://opencv.willowgarage.com/wiki/InstallGuide

[9] Web page: http://opencv.willowgarage.com/wiki/VisualC%2B%2B_VS2008

[10] Web page: http://sites.google.com/site/sanchohomesite/tutorials/installing-opencv-in-windows

[11] Web page: http://www.comp.leeds.ac.uk/vision/opencv/install-win.html

[12] Web page: http://technologyinterface.nmsu.edu/5_1/5_1f/5_1f.html

[13] MATLAB Web page: http://www.mathworks.com/access/helpdesk/help/techdoc/MATLAB_external/f29148.html

[14] MATLAB 7 External Interfaces, web page: http://www.mathworks.com.au/access/helpdesk/help/pdf_doc/MATLAB/apiext.pdf