UNIVERSITAT POLITÈCNICA DE VALÈNCIA


UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Departamento de Ingeniería de Sistemas y Automática
Máster de Automática e Informática Industrial

Tesina de final de Máster:

# DETECCIÓN DE PERSONAS CON VISIÓN ARTIFICIAL Y SENSORES DE RANGO

Autora: Laura Arnal Benedicto

Directores: Leopoldo Armesto Ángel
Pau Muñoz Benavent

Septiembre 2012

# Índice

# Índice de figuras

# Índice de cuadros

# 1. Introducción

La detección de personas es un aspecto fundamental en múltiples aplicaciones. En el sector de la vigilancia puede resultar útil para encontrar o seguir la trayectoria realizada por las personas para así alertar al personal de seguridad. Dentro del campo de la realidad virtual la detección es un punto clave para la correcta interacción con el entorno. También se utiliza la detección del movimiento de personas en animación, entrenamiento deportivo y videojuegos, como por ejemplo los videojuegos que utilizan el sensor Kinect. En robótica la detección de personas es clave para aquellos robots que interactúan o comparten entorno con ellas, así como para construir sistemas de seguridad en robots industriales y células automatizadas para minimizar el riesgo de accidentes.

Una aplicación interesante de la detección de personas son los sistemas avanzados de asistencia a la conducción (también conocidos como "advanced driving assistance systems" o ADAS). Un sistema de asistencia a la conducción puede alertar al conductor de un peligro próximo o, si no hay reacción por parte del conductor tras un aviso, modificar la trayectoria del vehículo para evitar un posible accidente. Además puede ayudar al conductor a mantener la distancia de seguridad, alertar de un cambio de carril involuntario o asistir en los adelantamientos, por nombrar algunas de sus otras aplicaciones. Otro de los aspectos en los que se centra la investigación actualmente es la protección de peatones, importante debido a lo desprotegido que está este colectivo en el entorno vial. Por esta razón la detección de obstáculos y personas es una parte crucial dentro de los sistemas ADAS.

## 1.1. Objetivos, justificación y problemática asociada

Esta tesina de final de máster se sitúa dentro del proyecto de investigación "Sistemas Avanzados de Seguridad Integral en Autobuses SAFEBUS". Dicho proyecto tiene como objetivo mejorar la seguridad activa y pasiva en los autobuses urbanos e interurbanos, ofreciendo una mayor comodidad a los usuarios y reduciendo la frecuencia y la gravedad de los accidentes, tanto en el interior como en el exterior del vehículo. Uno de los objetivos específicos del proyecto es el desarrollo de un sistema de asistencia a la conducción para la detección de personas y objetos a bajas velocidades y en los momentos de carga y descarga, objetivo claramente relacionado con la temática de esta tesina.

Según la documentación del proyecto, en la Unión Europea se producen alrededor de 1.300.000 de accidentes cada año, provocando 40.000 muertos y 1.700.000 heridos. De estas víctimas mortales, el 14 % son peatones. En España se producen 500 muertes cada año entre los peatones, aproximadamente un 15 % de las víctimas mortales. Dado que los errores humanos están presentes en entre el 85 y el 95 % de los accidentes, la existencia de un sistema avanzado de asistencia a la conducción con módulo de protección de peatones puede contribuir a la disminución del número de accidentes y la gravedad de los mismos, siendo especialmente útil en zonas cercanas a colegios, por ejemplo.

La detección de personas suele ser un problema bastante difícil. Por ejemplo, cuando se trata de detectar un objeto en un entorno industrial (como en una cadena de montaje) éstos suelen cumplir una serie de características de tamaño y forma que sufren muy pocas variaciones. En cambio, las personas pueden variar tanto en tamaño como en la postura en la que se encuentran, además de llevar distintos tipos de ropa y objetos (sombreros, maletas, paraguas, etc.) que aumentan aún más su

variabilidad. Todo esto complica el proceso de detección.

A la hora de desarrollar un sistema de protección de peatones hay que considerar otras dificultades además de la variabilidad de las personas. Una de las principales es el entorno: al desarrollarse en entornos urbanos, éste es dinámico y contiene muchos elementos como postes y señales de tráfico que pueden confundir al detector. Esto no ocurre en otras aplicaciones, como las relacionadas con la vigilancia, en las que el entorno suele ser estático y las cámaras están fijas o su movimiento es mínimo. En estos casos se pueden aplicar técnicas sencillas relacionadas con la sustracción de fondo, ya que éste es conocido y no suele sufrir grandes variaciones. En cambio, en sistemas ADAS el entorno puede cambiar constantemente y estar formado por diferentes elementos que dificultan la detección. De la misma manera hay que tener en cuenta la iluminación de la escena, que también puede variar en gran medida debido a las condiciones climatológicas, la hora del día o a la aparición de túneles u otros elementos. Además, al ser entornos muy complejos, es posible que los peatones no se vean completamente y sólo se puedan distinguir parcialmente. Por último hay que añadir que, debido al tipo de aplicación de la que se está hablando, se requieren altos requisitos de rendimiento en cuanto a tiempo de respuesta y robustez. El tiempo de respuesta ha de ser bajo para que el sistema sea capaz de alertar a tiempo al conductor, y se deberá evitar producir un alto número de falsos positivos para no avisar al conductor innecesariamente.

El objetivo de esta tesina es realizar un estudio de los algoritmos que se pueden encontrar en la literatura y paquetes software existentes en la plataforma ROS (Robot Operating System) para la detección de personas, y comprobar el funcionamiento de alguno de ellos. El estudio se centrará principalmente en algoritmos que se basen en visión (tanto monocular como estéreo) y sensores de rango. Dado que la tesina está asociada al proyecto de seguridad en autobuses SAFEBUS se ha prestado especial atención a algoritmos adecuados para sistemas ADAS.

## 1.2. Objetivos logrados

Con la realización de este trabajo se ha logrado conocer técnicas existentes para la detección de personas, distinguiendo las diferentes fases que forman este proceso, y conociendo los diferentes enfoques (detección por partes frente detección del cuerpo entero, distintos métodos de generación de candidatos, etc.). Además del conocimiento a nivel teórico, se han podido probar algunos algoritmos. Se ha utilizado un algoritmo para detección de objetos verticales como método de generación de candidatos para uno de los algoritmos de detección de personas más conocidos (Histogram of Oriented Gradients o HOG), y se ha conseguido mejorar el tiempo de cómputo que alcanzaba la implementación original.

De manera paralela a estos objetivos se ha trabajado con una webcam estéreo USB, dispositivo poco común, y con cámaras individuales y framegrabbers; adquiriendo y reforzando conocimientos de visión artificial y reconstrucción 3D. Por último también se ha aprendido a utilizar la plataforma ROS, dado que gran parte de la implementación se ha hecho sobre esta plataforma, y se han conocido librerías como OpenCV y Point Cloud Library.

## 1.3. Contenido de la memoria

La memoria se ha estructurado en cinco apartados. Tras esta introducción y justificación, en la sección 2 se revisa el estado del arte en el campo de la detección

de personas, centrándose principalmente en aplicaciones ADAS. Tras el estado del arte se realiza una descripción de los paquetes existentes en ROS para la detección de personas y se comentan algunas librerías y algoritmos utilizados durante el desarrollo en la sección 3. A continuación en la sección 4 se describe el equipo utilizado, los algoritmos probados y los resultados obtenidos. Para finalizar, en la sección 5 se comentan algunas conclusiones y posibles trabajos futuros.

# 2. Estado del arte

En esta sección se van a comentar las técnicas existentes en el área de la detección de personas a través de artículos revisados, centrándose principalmente en algoritmos adecuados para la aplicación concreta de un sistema de ayuda a la conducción. Por esta razón, no se han tenido en cuenta métodos de detección de caras o de manos, también tratados ampliamente en la bibliografía, al no ser de utilidad para el caso concreto de la conducción. En estos métodos se pueden distinguir una serie de fases comunes: generación de candidatos, clasificación y seguimiento (tracking). Aunque esto ocurre en la mayoría de los casos, algunos métodos no incluyen la fase de tracking o añaden una fase de confirmación de resultados entre las fases e clasificación y tracking. Existen múltiples estudios publicados que revisan el estado del arte en la detección de personas, como [1, 2, 3], centrados en principalmente en visión.

A continuación, primero se comentan las características de los sensores que se suelen emplear en la detección de personas. Posteriormente se comentan cada una de las fases en las que se divide la detección: generación de candidatos, clasificación, verificación y seguimiento. Por último se comentarán algunos métodos que incluyen fusión sensorial.

## 2.1. Sensores

Los sensores de visión se han utilizado ampliamente en esta área, tanto en el espectro visible como en visión nocturna, y en configuraciones monocular y estéreo. Entre sus principales ventajas están el bajo coste del hardware y un buen campo de visión. Además las cámaras que trabajan en el espectro visible proporcionan información de textura. En cambio los algoritmos que utilizan estos sensores suelen tener un alto coste computacional. Otro inconveniente es que están muy influenciados por la iluminación y las condiciones meteorológicas, factor a tener en cuenta dado que se está pensando en una aplicación de tipo ADAS. En condiciones de conducción con baja visibilidad o nocturna, las cámaras infrarrojas son una opción a considerar. Sin embargo también están afectadas por las condiciones meteorológicas, especialmente por la temperatura ambiente.

Otros sensores ampliamente utilizados son los sensores láser. Este sensor de tipo activo posee un amplio campo de visión y no le afectan los cambios en la iluminación, sin embargo no aporta mucha información sobre las personas que detecta. Los algoritmos que utilizan láser son poco costosos a nivel de procesamiento, sin embargo el coste del hardware es alto.

El radar no está influenciado por la meteorología o la iluminación, por lo que pueden ser útiles en condiciones de escasa visibilidad; en cambio, sí que les afecta la humedad o el polvo en suspensión.

En la tabla 1, obtenida de [2], se resumen muy bien las características de los sensores utilizados en detección de personas.

| **Sensor** | **Campo de visión** | **Res. angular** | **Rango detección** | **Rango de resolución** | **Iluminación** | **Coste HW** | **Coste comp.** |
|---|---|---|---|---|---|---|---|
| visión convencional | Medio | Medio/alto | Bajo/-medio | Medio | Reflectivo pasivo, necesita iluminación ambiente | Bajo | Alto |
| Cámara de campo de visión amplio | Alto | Bajo/-medio | Bajo | Bajo | Reflectivo pasivo, necesita iluminación ambiente | Medio | Alto |
| Infrarrojo cercano | Medio | Medio/alto | Medio | Medio | Iluminación activa, funciona en la oscuridad | Bajo | Alto |
| Infrarrojo térmico | Medio | Bajo/-medio | Bajo/-medio | Bajo | Emisivo, funciona en la oscuridad | Alto | Medio |
| Radar | Bajo | Bajo | Alto | Alto | Emisivo, funciona en oscuridad, lluvia, niebla. | Medio | Bajo |
| Láser | Alto | Medio | Medio | Alto | Emisivo, funciona en la osucridad | Alto | Bajo |

Cuadro 1: Características de sensores usados en detección (tabla de [2])

## 2.2. Generación de candidatos

La primera fase en la detección de personas es la generación de candidatos. En esta fase el objetivo es obtener un conjunto de regiones candidatas (*regions of interest* o ROI) que serán las que se clasificarán en la siguiente fase. El objetivo será reducir al mínimo este conjunto para facilitar la clasificación, reduciendo al máximo posible las zonas de fondo y tratando de evitar hacer una selección tan fuerte que tenga como consecuencia la pérdida de candidatos.

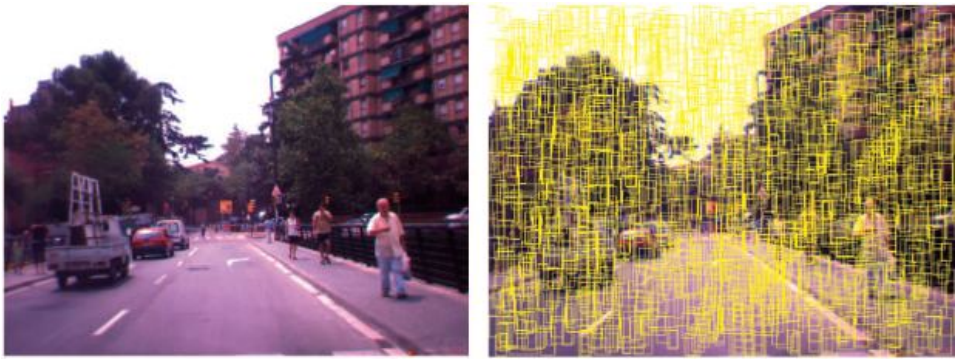

Figura 1: Generación de ROIs mediante escaneo exhaustivo, figura de [3]

En visión el método más simple es el método de **ventana deslizante o escaneo exhaustivo**, consistente en recorrer la imagen con una ventana de varios tamaños. Por ejemplo, en [4] los autores utilizan una ventana de 64x128 píxeles. El tamaño de estas ventanas suele guardar una razón de aspecto similar a la del cuerpo humano. Esta técnica presenta como principales inconvenientes el alto número de regiones candidatas generadas, hecho que complica emplear este tipo de métodos en aplicaciones de tiempo real debido al incremento de coste computacional que supone, así como la generación de candidatos con regiones irrelevantes de la imagen (como por ejemplo, regiones de fondo). En la figura 1 se muestra un ejemplo de este método en el que solo se muestra un 10 % del total de regiones generadas, lo que da una idea de la gran cantidad de candidatos que se puede llegar a generar. En [5], como mejora del escaneo exhaustivo, se propone un mecanismo que filtra las ventanas basándose en la textura de la imagen.

Otro método bastante estudiado está relacionado con la **detección de simetrías verticales**, aprovechando que el cuerpo humano cumple esta característica. En el trabajo de Bertozzi et al. [6] en un primer paso se limita la búsqueda a una porción de la imagen debido a que se conocen los parámetros intrínsecos del sistema y la asunción de que la escena es plana, de forma que se eliminan zonas de cielo y suelo. Los parámetros considerados son un rango de distancias o de tamaños. Tras este paso se consideran las columnas de la imagen como posibles ejes de simetría de los rectángulos de delimitación. Para cada uno de estos ejes de simetría se consideran diferentes ROIs dentro de un rango específico de distancias a la cámara y un rango de alturas y anchuras para una persona. No se consideran todos los ejes de simetría, solo aquellos que se encuentran en zonas con alta densidad de bordes. Para los ejes de simetría restantes se elige el mejor candidato de entre sus correspondientes rectángulos delimitadores (variando su posición y tamaño, pero compartiendo dicho eje). Para evaluar y decidir cuál es el mejor rectángulo de delimitación los autores proponen una combinación lineal de dos medidas: los valores de escala de grises y los valores de los gradientes horizontales. El mejor ROI será aquel que maximice esa función. Por último, se ajusta el tamaño de estos rectángulos debido a que suelen ser mayores que la persona a detectar. Para ello para cada eje de simetría se escoge el tamaño que maximiza una nueva función consistente en el producto de la simetría de los bordes verticales y la densidad de los bordes verticales.

El trabajo expuesto en [7] también se apoya en la detección de simetrías. En este caso se generan tres mapas, uno para las aristas verticales (figura 2) y dos para las horizontales (figura 3); y, mediante la combinación de ambos, se obtiene un tercer mapa que es el que se estudia para obtener las posiciones de las personas. Para generar ROIs se calculan los bordes laterales y el límite inferior a partir del mapa de aristas y la cabeza y los hombros asociando con modelos. Para mejorar esta medida se aprovecha el sistema estéreo: a partir de la estimación de la posición en la imagen izquierda, se busca la región correspondiente en la imagen derecha y comparando estos dos rectángulos delimitadores se calcula la distancia al peatón. Utilizando esta distancia se filtran las detecciones descartando aquellas que no cumplen el tamaño y razón de aspecto esperados para una persona a esa distancia.

En el trabajo anterior se asumía que la carretera era plana, hecho que permitía limitar tamaño y perspectiva. Para generalizar el sistema, los mismos autores proponen en [8] un método basado en estéreo. Primero, en las dos imágenes, se extraen aristas, se binarizan y se expanden horizontalmente. Después se calcula la correlación entre las líneas epipolares de ambas imágenes, obteniéndose una imagen de

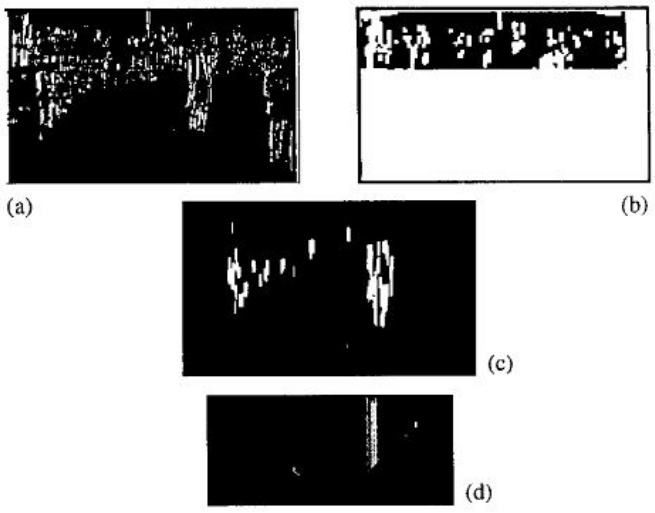


Figura 2: Aristas verticales (a), máscara para los objetos del fondo (b), aristas verticales restantes (c) y mapa de simetrías verticales (d), figura de [7]

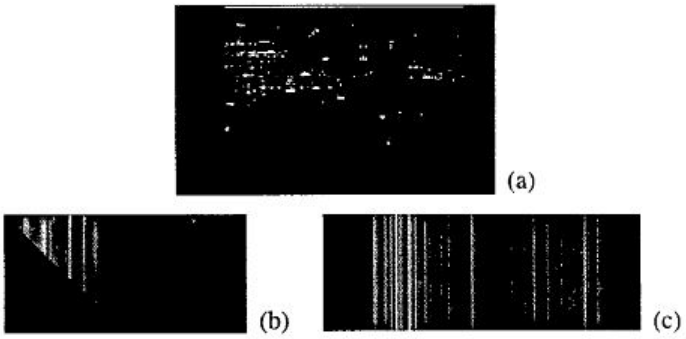


Figura 3: Simetrías verticales (b) y mapa de densidades lineales (c) para las aristas horizontales (a), figura de [7]

correlación en la que se pueden distinguir lineas que corresponden con la carretera, el fondo o los objetos. Para dejar sólo los objetos en la imagen, se extrae el fondo y el suelo mediante una transformada de Hough en la imagen de correlación. Una vez eliminados, se vuelve a obtener la imagen de correlación de forma que sólo quedan objetos verticales, que se pueden extraer mediante un histograma vertical. El área de cada obstáculo se puede obtener gracias a un histograma vertical (para los límites laterales) y a un histograma horizontal (para los límites superior e inferior). Este método tiene problemas en entornos con texturas muy pobres (puede hacer que no se detecte el suelo) o con sombras provocadas por objetos.

Otros sistemas se basan en **procedimientos estéreo** para limitar el número de regiones candidatas. En [9] se opta por obtener los candidatos a partir de puntos de interés 3D mediante un sistema estéreo. La representación 3D de los puntos relevantes se calcula en dos fases. En la primera fase los puntos relevantes se extraen usando el algoritmo Canny con umbrales adaptativos. En la segunda fase se crea un

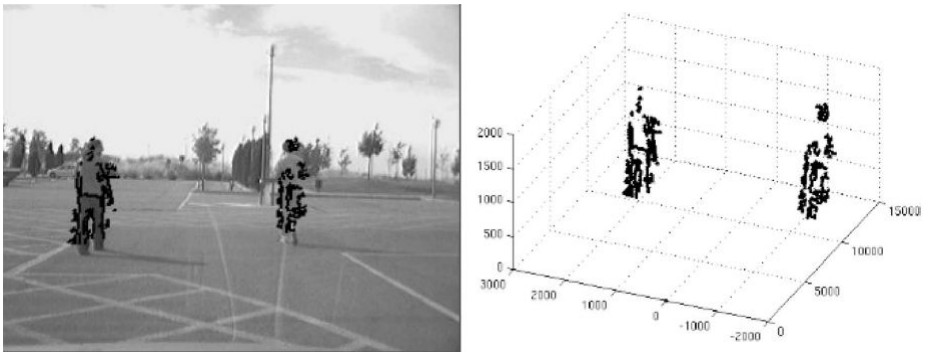


Figura 4: Puntos de interés 3D, figura de [9]

mapa 3D tras resolver el problema de la correspondencia, obteniéndose un resultado como el de la figura 4. Tras obtener los puntos relevantes se propone utilizar un método de clustering sustractivo, en el que los objetos en el espacio 3D se modelan mediante funciones gaussianas. Una vez realizado el clustering, las áreas de interés se hallan como las proyecciones en el plano de la imagen de las regiones 3D candidatas. Para cada uno de los clusters se generan varias ROIs para compensar el efecto que la precisión del área de este ROI pudiera tener en la fase de clasificación. En consecuencia se generan un total de 15 ROIs por cluster (figura 5, parte superior) variando ligeramente tanto el tamaño como la posición de la misma. En la fase de clasificación se considerará que es un peatón si se han clasificado un mínimo de 5 de sus ROIs (valor obtenido de forma experimental). Este tipo de representación permite una segmentación de objetos robusta siempre que el número de puntos relevantes en la imagen sea lo suficientemente alto. La principal ventaja es que los puntos aislados se pueden filtrar fácilmente en el espacio 3D, lo que lo hace poco sensible al ruido.

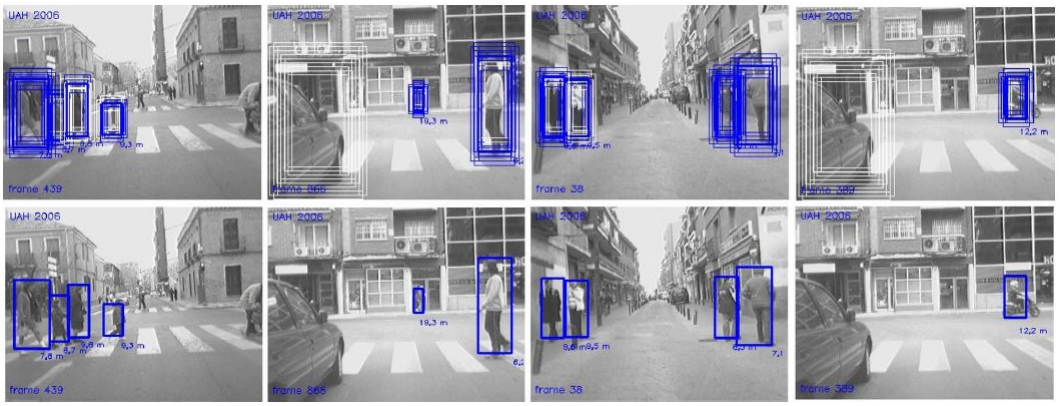


Figura 5: Generación de 15 ROIs por cada región (parte superior) y resultado de la clasificación (parte inferior), figura de [9]

En PROTECTOR [11], un conocido sistema ADAS, se calcula un mapa de profundidades en tiempo real. Este mapa de profundidades se multiplexa en N rangos discretos los cuales se escanean con ventanas en relación con el tamaño mínimo y máximo de los peatones, teniendo en cuenta la posición en el plano del suelo en ese

rango en particular. Si en una posición el número de características de profundidad sobrepasa un determinado porcentaje de la ventana, se añade el ROI que corresponde a esa posición. Un nuevo trabajo del mismo equipo presentado en el año 2007 [12] opta por el cálculo de la imagen de disparidad, en este caso mediante el algoritmo presentado en [13] aunque los autores señalan que podría optarse por otro algoritmo de cálculo de disparidad. Como resultado obtienen un mapa de disparidad poco denso.

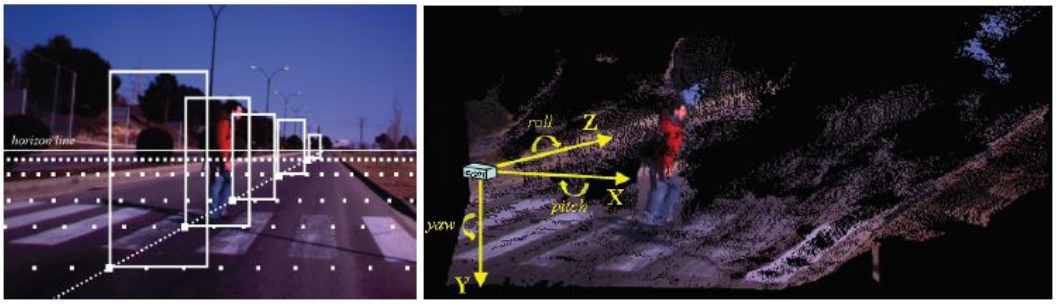


Figura 6: Generación de 5 ROIs por cada región (imagen izquierda) y nube de puntos correspondiente (imagen derecha), figura de [14]

Gerónimo, Sappa et al [14] también utilizan un sistema estéreo. Su propuesta consiste en encontrar el plano de la carretera mediante un algoritmo basado en RANSAC para estimar la posición relativa de la cámara. Una vez obtenida se generan 5 ROIs dentro de un rango de tamaños cada 0.5 metros tanto en X como en Z, como se muestra en la figura 6.

Si se emplean sensores infrarrojos térmicos un procedimiento común y sencillo es aplicar un **umbralizado**. Por ejemplo, en [15] se comienza detectando los puntos calientes en la imagen y aplicando una umbralización. La extracción de candidatos se realiza tomando un área algo mayor que el punto de calor detectado. En la figura 7 se muestra en a) la imagen original. Posteriormente, se calcula su histograma y se segmenta en regiones (figura 7b) para poder extraer los posibles peatones (figura 7 c y e) y hallar la estimación de las mismas (figura 7 d y f). En [16] Broggi et al. también aplican un umbralizado para la extracción de candidatos.

Al utilizar sensores de rango, como un sensor láser, en [17, 18, 19] se **divide en segmentos** cada linea del escáner realizado, por ejemplo utilizando el algoritmo de clustering "Jump Distance Clustering". Este procedimiento inicia un nuevo segmento cuando la distancia entre dos puntos consecutivos sobrepasa un umbral determinado.

Por último se pueden generar regiones candidatas **analizando el movimiento** de los objetos. Es el caso de [20], donde Liu y Fujimura utilizan un sistema de visión nocturna estéreo. Para detectar el movimiento relativo de los objetos, se realiza una detección en tres pasos. Primero, tras aplicar una umbralización sobre las imágenes, se realiza la correspondencia estéreo. El segundo paso consiste en realizar la correspondencia con el frame anterior. Para ello, para cada burbuja detectada se busca en un área de búsqueda cercana a su centroide la burbuja que tiene un área similar. Si esta fase no da buenos resultados se realiza una búsqueda basada en correspondencia de silueta. El tercer paso consiste en detectar si el objeto se ha movido. Para ello se estima la nueva posición compensando el movimiento del vehículo, de forma que si tras esta compensación el movimiento es mayor que un determinado umbral (para tener en cuenta errores debido a que se considera la posición como la posición del

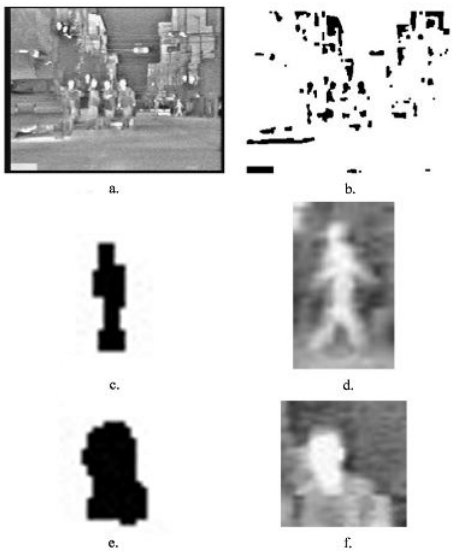


Figura 7: Generación de ROIs con infrarrojos, figura de [15].

centroide) es que el objeto se ha movido. Dado que se detectan todo tipo de objetos móviles, hace falta aplicar una fase más para discernir si el objeto es una persona u otro objeto capaz de moverse. Los autores proponen utilizar la razón de aspecto de la burbuja o un método de clasificación más complejo, como los basados en silueta que se comentarán en la siguiente sección.

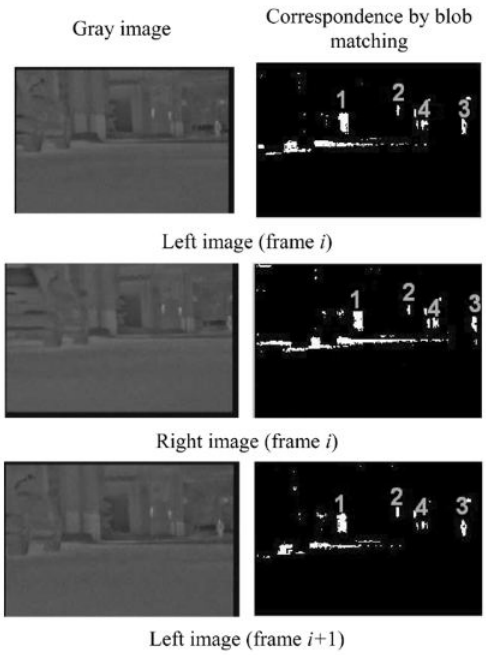


Figura 8: Correspondencia de blobs entre frames, figura de [20].

## 2.3. Clasificación

En esta fase el objetivo es determinar si las regiones obtenidas en la fase anterior son personas o no lo son, minimizando el número de falsos positivos (clasificar como persona una ROI que no lo es) y los falsos negativos (no clasificar una persona que sí aparece). Se puede diferenciar entre dos grandes enfoques: basados en silueta o basados en apariencia.

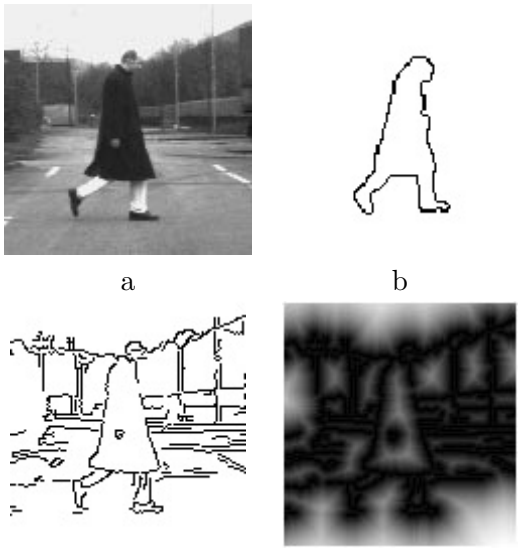


Figura 9: Sistema PROTECTOR: imagen original (a), plantilla (b), imagen de características (c), imagen DT (d) (figuras de [10])

El primer enfoque se basa en **relacionar la región de interés con una silueta**. Este método es el utilizado por Gavrila et al. en el sistema PROTECTOR [10, 11] donde la apariencia de las personas se modela mediante plantillas, de forma que asociar una imagen con una plantilla implica calcular la imagen de características y aplicar una transformación de distancia para obtener una imagen DT (figura 9d). Una transformación de distancia convierte una imagen binaria (consistente en píxeles que pertenecen a las características y píxeles no pertenecientes) en una imagen donde el valor de cada píxel indica la distancia al píxel de la característica más cercana. En este caso la métrica utilizada es la distancia *chamfer*, que calcula una aproximación de la distancia euclídea usando cálculo integral. Tras calcular la transformación de la distancia, se aplica una traslación sobre la plantilla correspondiente y se posiciona sobre la imagen DT calculada. Para valorar la validez de la correspondencia se contabiliza el valor de los píxeles de la imagen DT que recaen sobre los píxeles de la plantilla transformada. En la aplicación final se considera que una imagen corresponde con una plantilla si este valor es menor que un umbral determinado por el usuario. Según los autores, la principal contribución del método es el uso de una estructura jerárquica de plantillas (figura 10), de forma que se puede hacer una búsqueda eficiente entre todas las plantillas disponibles. De esta manera se puede obtener un conjunto de plantillas off-line. Las plantillas similares se agrupan bajo una plantilla prototipo y un parámetro de distancia. Este parámetro refleja la disimilitud entre la plantilla prototipo y la plantilla que representa. Una vez obtenida

la jerarquía de plantillas, en la aplicación on-line se obtiene la plantilla que mejor corresponde con la imagen realizando una búsqueda en una estructura en árbol.

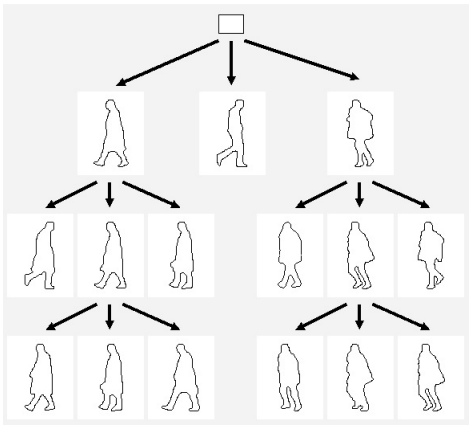

Figura 10: Sistema PROTECTOR: jerarquía de plantillas (figura de [10])

Otro trabajo basado en la silueta es [7] en el que, como se ha comentado en la sección 2.2, Bertozzi et al. hacen uso de modelos como los de la figura 11 de los hombros y la cabeza para acabar de generar las regiones de interés.

Figura 11: Siluetas de la parte superior del cuerpo (figura de [7])

El segundo enfoque se basa en la **apariencia de las personas**: se define un conjunto de características de interés o descriptores y se entrena un clasificador con ellos. Dichos descriptores pueden hacer referencia al cuerpo completo (métodos de tipo holístico) o a partes individuales. Siguiendo el **enfoque holístico** se encuentran métodos como el propuesto por Dalal y Triggs [4], donde presentan un conjunto de características adecuado para la detección de personas llamado Histogram of Oriented Gradients (HOG). El método se basa en evaluar histogramas locales normalizados de una imagen de gradientes orientados. La imagen se divide en pequeñas celdas cada una de las cuales acumula direcciones del histograma de gradiente u orientaciones de los bordes de los píxeles de las celdas. Se recomienda para una mejor respuesta normalizar el contraste en unas zonas más grandes (denominadas bloques) y utilizar dicho resultado para normalizar las celdas del bloque. Estos bloques de descriptores normalizados son lo que los autores denominan descriptores HOG (figura 12). Por último, se utilizan los descriptores HOG de la ventana de detección como entrada a un clasificador SVMLight [21], que es una implementación de un SVM (support vector machine) adecuada para trabajar con grandes conjuntos de datos. SVM es una técnica para entrenar clasificadores capaces de discernir entre patrones de la clase a detectar y cualquier otro patrón. El algoritmo SVM presenta el problema de entrenamiento como un problema de encontrar entre todas las superficies de separación posibles la que maximiza la distancia entre los elementos más próximos de dos clases. Esto lo consigue resolviendo un problema de programación cuadrática.

Los descriptores HOG están inspirados en las características SIFT [22]. Los au-

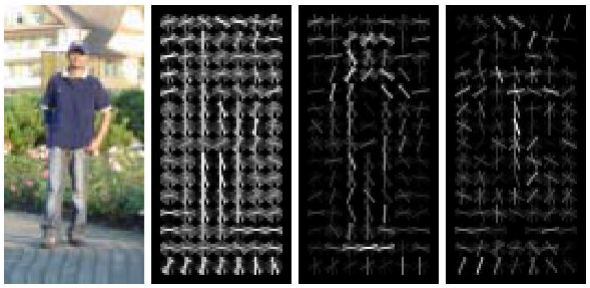

Figura 12: Ejemplo extracción de descriptores HOG, figura de [4].
De la imagen de test, se muestra el cálculo de los descriptores HOG y los descriptores ponderados según los pesos positivos y negativos del clasificador SVM.

tores remarcan como principal ventaja de estos descriptores que captan estructuras características de la forma a detectar y que, al trabajar con celdas pequeñas, se mantiene invariante ante transformaciones exceptuando la orientación, por lo que es adecuado para detectar personas que se encuentren en posición vertical.

Otro método holístico es el presentado por Papageorgoiu y Poggio en [23]. En él se extraen características y se clasifican con un SVM. Las características son en este caso Haar Wavelets (HW). Los Haar Wavelets se pueden definir como un filtro que calcula la diferencia de nivel de gris entre dos zonas definidas. Algunos ejemplos de configuraciones de filtro se pueden ver en la figura 13. Su uso permite propagar restricciones entre regiones circundantes y describir patrones complejos. Los autores justifican la utilización de Haar Wavelets debido a que captan características de la forma de los objetos y de su interior que son invariantes a determinadas transformaciones. Como se ve en la figura 14, donde se muestran la media de los coeficientes para los Haar Wavelets, los wavelets verticales identifican los lados del cuerpo, los horizontales la cabeza y los hombros, y los diagonales la cabeza, hombros, manos y pies. En cuanto al clasificador, utilizan Support Vector Machines (SVM). Este método consigue en detección de personas un 90 % de acierto, con 1 falso positivo cada 10.000 muestras procesadas.

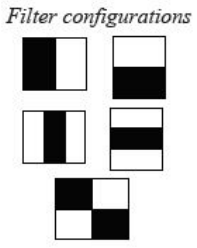


Figura 13: Ejemplo filtros de descriptores HW, figura de [14].

En [14] también se utilizan Haar Wavelets, en este caso junto con histogramas de orientación de aristas (EOH). Los EOH (figura 15) se apoyan en la información de las aristas, por lo que se mantienen invariantes a cambios de iluminación. En este caso debido a la perspectiva hay ROIs de tamaños diferentes, por lo que es necesario normalizar el tamaño de los Haar Wavelets para poder establecer una equivalencia. Con

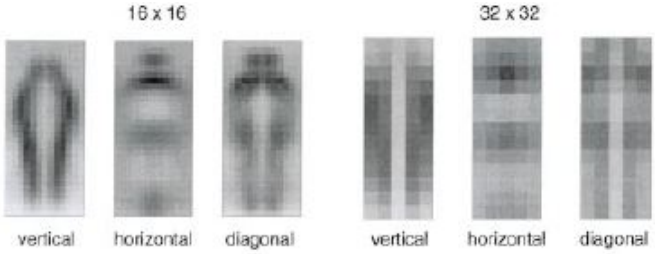


Figura 14: Media de los coeficientes de los Haar Wavelets, figura de [23].

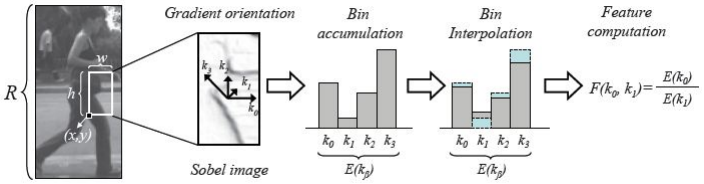


Figura 15: Cálculo de los descriptores EOH, figura de [14].

estos dos descriptores se entrena un clasificador mediante el algoritmo AdaBoost.

Siguiendo con el enfoque holístico, en [24] se utiliza un radar UWB (Ultra-Wideband) para la detección de personas. Para ello, se requiere un modelo de la propagación y la interacción con el cuerpo humano de la onda del radar. La señal radar que se obtiene de objetos con formas complejas como las personas consiste en observaciones de múltiples componentes. Para estimar el tiempo de llegada y la amplitud se puede aplicar el algoritmo CLEAN [25] a los valores escaneados. Para reducir el coste computacional se aplica un sistema para eliminar aquellas respuestas que claramente no corresponden con las respuestas obtenidas al percibir una persona. En [26] se utilizan dos sensores radar cuyas áreas de detección se solapan ligeramente. Los únicos objetos que se detectan son coches, personas, bicicletas y árboles, debido a la reflexión. Para diferenciar entre ellos se utiliza como medidas la velocidad y la "Radar Cross Section" (RCS), que determina la capacidad reflectante de los objetos. Para cada uno de los objetos mencionados se conoce el rango habitual de RCS.

En [5] Shashua et al. utilizan una **clasificación por partes** en contraposición a una representación holística. De esta manera se pasa a tener múltiples partes cada una con una determinada variabilidad que es menor que la variabilidad de la persona considerada como un todo global. Así pues, se divide la región de interés en nueve zonas (las numeradas del 1 al 9 en la figura 16) y se entrena un clasificador para cada una de ellas. Además se obtienen cuatro zonas adicionales (las numeradas del 10 al 13 en la figura 16) combinando las zonas anteriores, y se entrena un clasificador para cada una. En total se generan 13 clasificadores débiles cuyos resultados se combinan mediante AdaBoost, utilizando todo el conjunto de entrenamiento. Las características que se extraen de cada una de las regiones son características SIFT [22].

En [9] se propone extraer seis subregiones para cada ROI (figura 17). El valor de seis se ha obtenido de forma empírica. Para cada una de estas subregiones se aplica un método de extracción de características: para la cabeza y la zona inferior central se

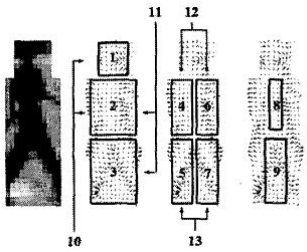


Figura 16: División en trece partes presentada por Shashua et al., figura de [5].

utiliza el método NTU [27], para los brazos el histograma de diferencias de intensidad y para las piernas descriptores HOG [4]. Por lo tanto para cada subregión se utilizará un método de extracción, y se entrenará un clasificador SVM independiente para dicha región. Estos seis clasificadores se combinan en una segunda parte en la que se puede utilizar otro clasificador SVM o un criterio de distancia.

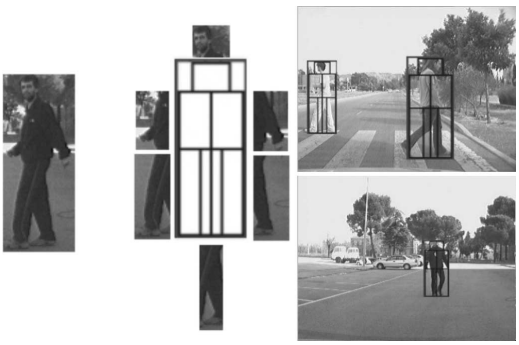

Figura 17: División en seis partes presentada por Alonso et al., figura de [9].

Wu y Nevatia [33] también proponen una detección por partes basada en la detección de "edgelets". Un "edgelet" (figura 18) es un segmento corto de una línea o una curva. Las características de este tipo que utilizan son lineas, circunferencias, medias circunferencias, un cuarto de circunferencia, un octavo de circunferencia y sus pares simétricos. Los autores de este trabajo utilizan como partes a detectar la zona de la cabeza y los hombros, el tronco y las piernas. Se construye un clasificador adaBoost para cada una de las zonas, además de un clasificador conjunto que tiene en cuenta las relaciones espaciales entre partes (figura 19).

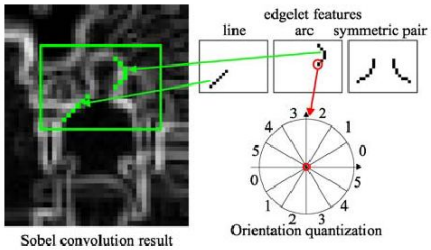


Figura 18: Ejemplo de características de tipo edgelet, figura de [33].

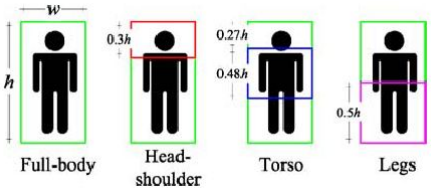


Figura 19: Partes detectadas en Wu & Nevatia y relaciones espacial entre ellas, figura de [33].

Los autores de [15] proponen dos variantes para detección a partir de la imagen obtenida mediante visión nocturna. En la primera se generan un total de seis clasificadores SVM: tres por cada tipo de peatón (peatón cruzando la carretera, peatón andando a lo largo de la carrera y ciclistas) y, para cada uno de estos tipos, dos clasificadores, uno para cuerpo completo (enfoque holístico) y otro para partes del cuerpo (enfoque basado en detección de partes). En la segunda variante se genera un solo clasificador SVM para todos los tipos de peatón. Experimentalmente muestran que mediante esta última opción se consigue una mayor tasa de acierto.

En relación a sensores de rango existe múltiples métodos que detectan piernas a partir de las medidas de un láser, principalmente calculando **características geométricas** de los segmentos. En [17] se comprueba que el segmento es un semicírculo y además que la distancia entre los puntos extremos está dentro del rango esperado de diámetro de una pierna. En [18], Arras et al. definen las características siguientes:

1. numero de puntos del segmento.
2. desviación típica: para dar una medida del centro de gravedad del segmento.
3. promedio de la desviación media de la mediana: mide lo compacto que es el segmento.
4. distancia de salto entre el segmento actual y el anterior.
5. distancia de salto entre el segmento actual y el posterior.
6. ancho del segmento.
7. linealidad.
8. circularidad.
9. radio de la circunferencia que se ajusta al segmento.
10. longitud del contorno: longitud de la polilínea del segmento.
11. regularidad del contorno.
12. curvatura media.
13. diferencia angular media: mide la concavidad/convexidad del segmento.
14. velocidad media: determina la velocidad de cada punto del segmento a lo largo de su rayo.

Los autores de este trabajo construyen un clasificador utilizando el algoritmo AdaBoost a partir de diez clasificadores débiles, a diferencia de otros métodos en los que la clasificación se realizaba simplemente considerando un umbral para cada una de las características. Una de las conclusiones que obtienen es que las características que mejor representan el problema son el radio de la circunferencia, la diferencia angular media, distancia de salto entre el segmento actual y el anterior, el promedio de la desviación media de la mediana y la distancia de salto entre el segmento actual y el posterior.

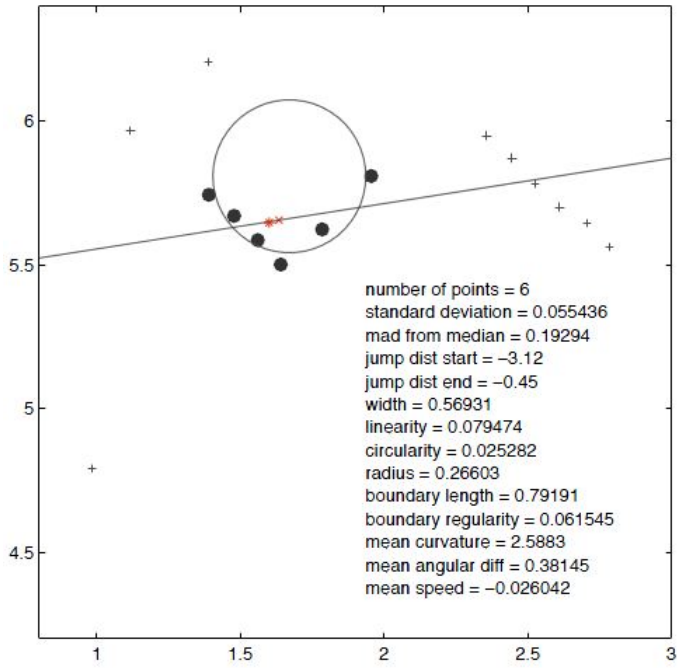


Figura 20: Cálculo de características de un segmento láser, figura de [18].

Los sistemas que se basan sólo en la detección de piernas colocan los sensores a una altura a la que se espera encontrar las piernas. El problema con estos sistemas es que a esa altura las piernas se representan por segmentos cortos que también pueden corresponder con otros objetos del entorno. En [28] Mozos et al. proponen un sistema en el que se utilizan varios sensores láser 2D a diferentes alturas (figura 21). Cada láser se utiliza para detectar una parte del cuerpo, como las piernas, el tronco o la cabeza. En cada una de las capas el escáner obtenido se segmenta y se extraen características geométricas de cada uno de los segmentos obtenidos. En este caso se calculan once características de los segmentos: número de puntos, desviación típica, promedio de la desviación media de la mediana, distancia euclídea entre el primer y el último punto del segmento, linealidad y circularidad del segmento, radio del segmento, longitud y regularidad de la frontera del segmento, curvatura media del segmento y diferencia angular media. De esta forma, siguiendo el método propuesto en [18], se obtiene un clasificador AdaBoost para cada sección capaz de clasificar los segmentos correspondientes a la parte del cuerpo de la persona que está a esa altura.

Para combinar los resultados obtenidos de los clasificadores individuales se sigue un proceso de voto probabilista. También es necesario obtener un modelo de la forma de la persona que especifica la distancia entre las partes del cuerpo. Para ello

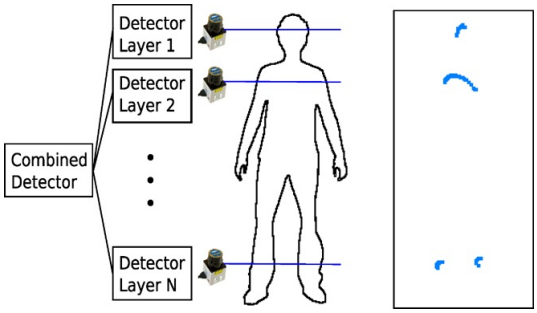


Figura 21: Sistema propuesto por Martínez Mozos, Kurazume y Hasegawa, figura de [28].

se proyectan los segmentos correspondientes en el plano horizontal 2D, obteniendo la distancia entre las distintas capas, de forma que se obtiene una relación entre las partes del cuerpo. Para cada una de estas relaciones se obtiene una función de test. Finalmente la persona detectada es la que tiene una mayor puntuación positiva en el proceso de voto.

En este trabajo se habla de tres capas de detección, correspondiente con piernas, tronco y cabeza, pero se pueden utilizar más para, por ejemplo, detectar niños (que podría no ser detectado por las capas más altas al ser de menor altura que un adulto). En principio está pensado para detectar a sólo una persona, pero no esta limitado a ello. Según la experimentación realizada por los autores se obtiene una mayor tasa de acierto que en el caso de utilizar una sola capa.

En [19] también se propone una detección por capas, sin embargo en este caso se añade la restricción de que éstas sean paralelas, como se muestra en la figura 22. Las capas no necesariamente corresponden con partes del cuerpo humano, como tronco o extremidades. Con este modelo, se propone un clasificador para cada una de estas partes utilizando AdaBoost. De cada segmento se calculan 17 características geométricas: las características 1-3 y 6-13 que se utilizaban en [18], además del ajuste a una spline cuadrática, ajuste a una spline cúbica, el centroide de Kurtosis, la relación PCA (relación entre el segundo mayor y el mayor autovalor de la matriz de dispersión), el área de la zona de delimitación y el área del casco convexo.

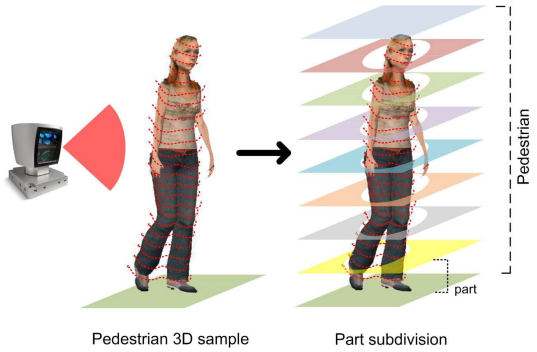


Figura 22: Sistema propuesto por Spinello et al., figura de [19].

Posteriormente los clasificadores se combinan mediante un modelo de voto. Estos votos se aprenden a partir de la distancia entre el segmento y el centro de la persona

en la fase de entrenamiento del clasificador. En la fase de detección, al no conocerse el centro, se debe estimar. Cada segmento se clasifica y aporta un determinado voto (aprendido en el entrenamiento). Con todos los votos se estima la moda de la distribución de votos mediante Mean Shift [29] con un kernel esférico uniforme. Para cada una de estas modas se calcula una ponderación, cuanto mayor sea este valor más seguridad se tiene sobre la existencia de una persona en ese punto.

## 2.4. Verificación

Algunos métodos añaden esta etapa tras realizar la clasificación para verificar los resultados obtenidos y filtrar falsos positivos. Por ejemplo en [10] se vuelve a utilizar la imagen original para extraer una ventana rectangular correspondiente al rectángulo que delimita la plantilla asociada durante la fase de clasificación. Esta ventana se normaliza y se clasifican los píxeles resultantes utilizando una aproximación local basada en RBF (Radial Basis Functions); aunque también se podría realizar con otros clasificadores.

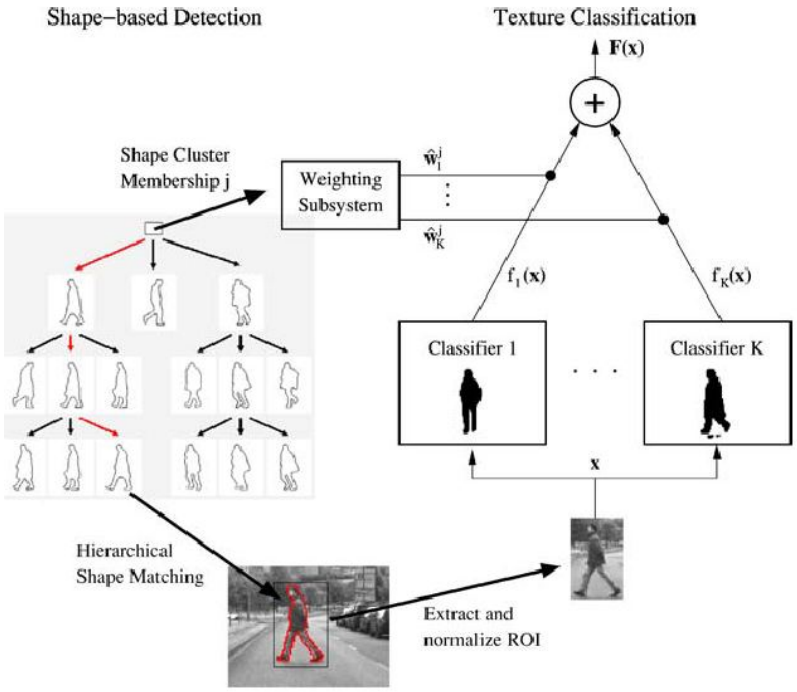


Figura 23: Sistema PROTECTOR con verificación por texturas, figura de [12].

En el sistema PROTECTOR [11, 12] se realiza una clasificación basada en la textura para verificar las detecciones realizadas por el sistema chamfer, cuya arquitectura se muestra en la figura 23. Como clasificador se utiliza una red neuronal con campos receptivos locales, descartando las detecciones para las cuales la salida de la red neuronal está por debajo de un valor de confianza determinado por el usuario. Posteriormente se realiza una segunda etapa de verificación basada en estéreo para filtrar falsos positivos. La plantilla con la que se asoció en la etapa de detección se utiliza como máscara para realizar una correlación con la otra imagen estéreo, utilizando la distancia aproximada a la que se encuentra la detección y los datos de

disparidad. Esto se hace con el objetivo de filtrar falsas detecciones que contienen mucho fondo.

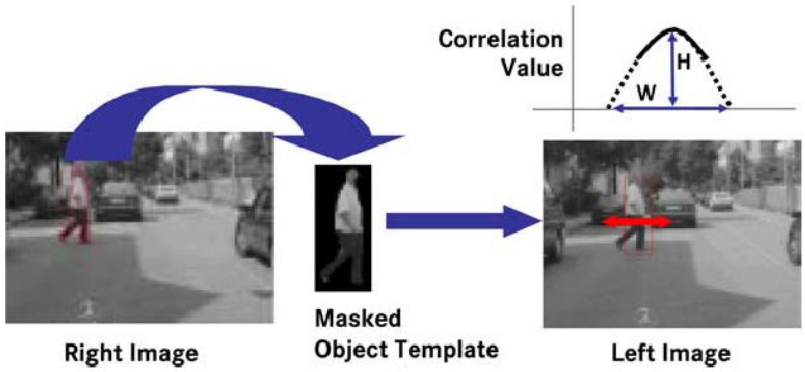


Figura 24: Verificación en el sistema PROTECTOR: segunda etapa basada en estéreo. Figura de [12].

Bertozzi et al. [6] también aprovechan el sistema estéreo: para cada rectángulo delimitador se busca una zona de la otra imagen que tenga un contenido similar mediante una medida de correlación entre píxeles. Una vez se ha encontrado la zona correspondiente se determina la distancia al sistema de visión mediante triangulación, por lo que se puede realizar un ajuste del rectángulo delimitador basándose en las restricciones de calibración y la perspectiva. Tras esto se realiza un segundo paso. Este método se basaba en la detección de objetos verticales simétricos, por lo que se pueden detectar objetos de estas características además de personas. Por esta razón es necesario realizar un filtrado posterior para eliminar falsos positivos. Estos filtros se basan en el análisis de la distribución de los bordes dentro del rectángulo delimitador y en la segmentación y clasificación de la región del rectángulo. Por último, se ha detectado que en este sistema a veces el rectángulo delimitador no cubre completamente a la persona, concretamente la altura puede ser algo menor. Este hecho influye directamente en la estimación de la posición, por lo que debe tenerse en cuenta. La solución propuesta establece una relación entre el rectángulo de delimitación y las coordenadas de posición del peatón utilizando parámetros de calibración. Finalmente se utiliza un filtro de Kalman para realizar la estimación de la posición de la persona. Justifican esta elección argumentando que este estimador tiene en cuenta los errores, además de permitir predicción en la posición.

Por último, en [5] se utiliza información de múltiples frames para corroborar las detecciones obtenidas en la fase de clasificación. Con este fin se realiza un proceso de clasificación basado en un clasificador de tipo árbol de decisión, con características como la forma del paso, el movimiento de la persona y tracking.

## 2.5. Seguimiento (tracking)

Muchos sistemas también incluyen un método para realizar seguimiento de personas y así poder conocer la dirección del movimiento y consolidar las detecciones en el tiempo, entre otros.

En este módulo el método más utilizado es el **filtro de Kalman**. Es el caso de [9] o [15], en este último se utiliza además el método Mean-shift para ajustar mejor la posición de la persona. En la figura 25 se muestran detecciones obtenidas

tras la fase de tracking. En la imagen de la izquierda se marcan con circunferencias las detecciones realizadas en la fase de detección, mientras que en la imagen de la derecha se muestran las detecciones obtenidas en la fase de tracking (rectángulos).

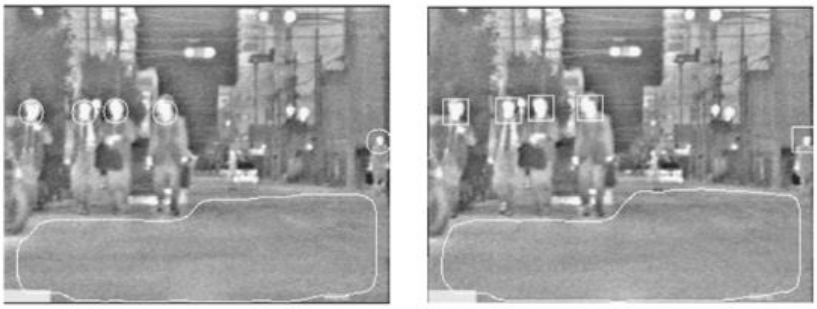


Figura 25: Detección y tracking en el sistema propuesto por Xu, Liu y Fujumura, figura de [15].

En [11, 12] se utiliza un $\alpha - \beta$ tracker (filtro de Kalman simplificado) basado en el recuadro de delimitación obtenido en la fase de verificación. Para tratar asociaciones del tipo una medida - varios tracks (o viceversa) se utiliza el método Húngaro utilizando como medida de similitud una combinación lineal ponderada de la distancia euclídea entre los centroides de los objetos y la distancia Chamfer. En este sistema utilizan los valores obtenidos del tracking para evaluar el riesgo que supone cada persona detectada. En [24] se utiliza un algoritmo "Expectation-Maximization Kalman Filter" (EMKF) para el tracking de un número fijo de personas en el que el algoritmo de expectación-maximización asocia de forma simultánea las observaciones individuales a cada objetivo y estima el estado de cada uno de ellos.

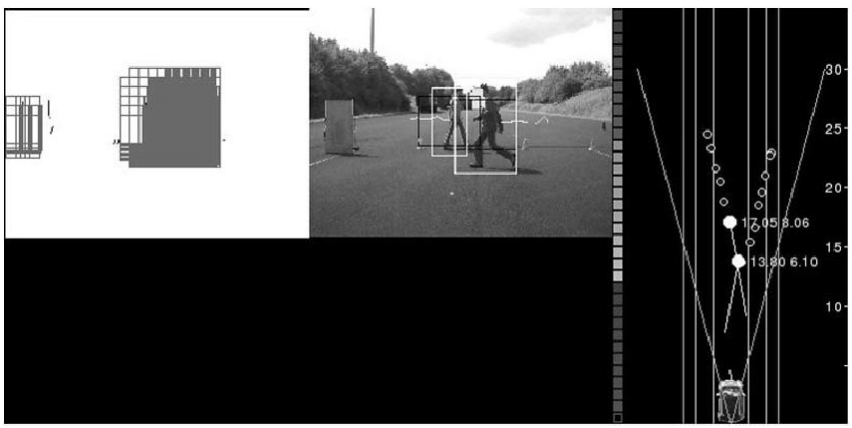


Figura 26: Detección y tracking en el sistema protector para la evaluación de riesgos, figura de [12].

Otro método utilizado para la fase de tracking son los **filtros de partículas**. En [30] se utiliza una variante de estos filtros conocido como el algoritmo "Condensation" [31] para seguir siluetas cuyo contorno se ha modelado como un conjunto de B-Splines (figura 27). El método "Condensation" ofrece una estimación dinámica del

estado donde la función de densidad de probabilidad subyacente no es gaussiana. En este caso, esta función se representa mediante un modelo de Monte Carlo, donde la función de densidad de probabilidad se representa por un conjunto de muestras aleatorias. En Munder et al. [32] también se utilizan filtros de partículas. Concretamente, se asocia un filtro individual para cada persona a seguir, infiriendo la velocidad o posición del objeto, así como si se trata del objeto a seguir o forma parte del fondo de la escena.

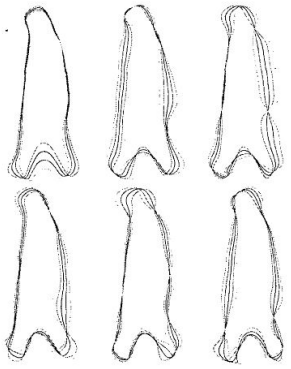

Figura 27: Siluetas y variaciones utilizadas para tracking en [30].

Por último, Wu y Nevatia [33] realizan el seguimiento mediante asociación de datos y el algoritmo Mean Shift [29]. Primero se intenta realizar el tracking asociando las hipótesis de los objetos con las detecciones de personas. Si no se puede asociar con una persona completa, se intenta asociar con una detección de una parte de una persona, ya que en este método se realiza una detección por partes. Si este segundo paso falla, se utiliza el método Mean Shift para seguir al objetivo.

## 2.6. Fusión sensorial

Todos los tipos de sensores presentan ventajas y desventajas. Utilizando una combinación de sensores se pueden compensar los efectos negativos y construir un sistema más robusto que aproveche las ventajas de cada sensor. Esta integración se puede hacer mediante una arquitectura de sensores en paralelo o secuencial.

En [34] se utiliza una cámara infrarroja FIR y un láser en paralelo. Tanto la fusión sensorial como el tracking se realiza mediante un conjunto de filtros de Kalman. El escáner láser proporciona la profundidad y la reflexión en cada dirección acimutal donde se sobrepasa un cierto umbral de reflexión. El algoritmo busca escalones en la señal utilizando la primera y la segunda derivada y genera los objetos a partir de estos escalones. En la imagen FIR se utiliza un umbral para binarizar la imagen. Sobre esta se agrupan los píxeles adyacentes de forma que se genera un conjunto de regiones. De estas regiones sólo se aceptan las que tienen una determinada área y orientación vertical.

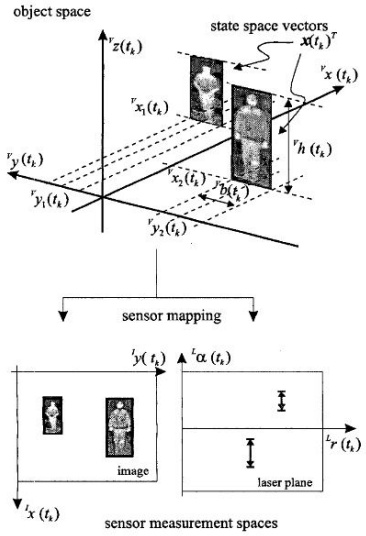


Figura 28: Fusión sensorial en [34].

En cambio, en [26] se utiliza una estructura secuencial consistente en dos radares y un sistema de visión. La arquitectura de dicho sistema se muestra en la figura 2.6: el sistema radar detecta coches, personas, bicicletas y árboles, y se puede discernir entre ellos a partir de la capacidad reflectante de los objetos. Posteriormente los candidatos generados por el radar se comprueban mediante un sistema de visión utilizando un modelo 2D de la silueta de una persona. Se entrenan dos modelos: uno para la vista frontal de una persona y otro para la vista lateral. Los modelos utilizados fueron propuestos inicialmente por Kass et al en [35] y se conocen como contornos activos o "snakes". Estos modelos se definieron como unas curvas paramétricas cerradas que minimizan energía guiadas por fuerzas externas.

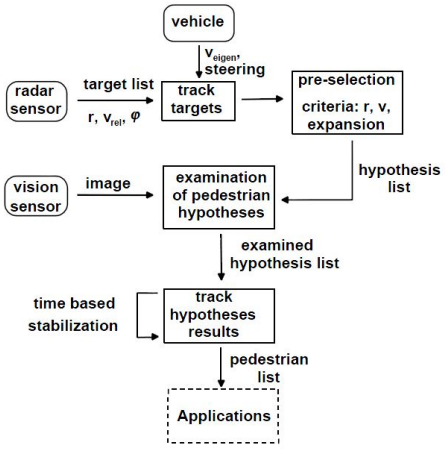


Figura 29: Arquitectura del sistema presentado por Milch y Behrens, figura de [26].

# 3. Software para la detección de personas

En esta sección se detalla el software utilizado durante el desarrollo de la tesina. Por un lado, se describen los paquetes software existentes en ROS que realizan detección de personas. Primero se describen las principales características de ROS para posteriormente comentar los nodos implementados en cada stack o paquete ROS y su funcionamiento. De la misma manera que en el apartado anterior, no se han considerado los paquetes que tratan de reconocimiento de caras, por no ser de utilidad para un sistema de ayuda a la conducción. Al final de esta sección se describen dos herramientas software utilizadas en la experimentación, como son OpenCV y Point Cloud Library (PCL).

## 3.1. ROS: Robot Operating System

ROS (Robot Operating System) [36] es un meta-sistema operativo para robótica. Proporciona servicios que se esperan de un sistema operativo, como abstracción hardware, control de dispositivos a bajo nivel, implementación de funcionalidades comúnmente utilizadas, paso de mensajes entre procesos y gestión de paquetes. También aporta herramientas y librerías para obtener, construir, escribir y ejecutar código entre múltiples ordenadores. Una de las principales ventajas de este framework es la capacidad de desarrollar nodos software reutilizables., además de proporcionar herramientas y algoritmos de uso típico en robótica.

Este framework está disponible de forma estable bajo Ubuntu y de forma experimental en otros sistemas como OS X (Homebrew), Fedora, Gentoo, OpenSUSE, Arch Linux o Windows; así como para múltiples robots. ROS no es un framework de tiempo real, pero sí que se puede integrar con código de tiempo real. De hecho, es fácil su integración y ya está integrado con otras plataformas como OpenRAVE, Orocos o Player. ROS es, además, independiente del lenguaje de programación utilizado: actualmente soporta Python, C++ y Lisp, y de forma experimental Java y Lua.

ROS es de código abierto, bajo licencia BSD. Está disponible desde Enero de 2010 con la versión 1.0 y hasta la actualidad se han lanzado otras versiones: Box Turtle (Marzo 2010), C Turtle (Agosto 2010), Diamondback (Marzo 2011), Electric Emys (Agosto 2011) y Fuerte (Abril 2012), la última versión lanzada.

El grafo de ejecución de ROS es una red punto a punto de procesos que están débilmente acoplados utilizando la infraestructura de comunicación de ROS (basada en TCP/IP). En ROS hay implementados varios estilos de comunicación, como comunicación síncrona de tipo RPC (llamada a procedimiento remoto, del inglés *Remote Procedure Call*) a través de servicios, comunicación asíncrona por medio de topics (o tópicos) y almacenamiento de datos en un servidor de parámetros que se puede consultar desde cualquier nodo.

A continuación se describen algunos de los conceptos más importantes de este framework.

### 3.1.1. Sistema de ficheros de ROS

El código que se genera para ejecutar en ROS sigue una determinada organización a nivel de sistema de ficheros. En esta organización se pueden distinguir paquetes (o packages), stacks, ficheros de manifiesto de paquete y de stacks y descripción de mensajes y servicios.

- Paquetes (packages): los paquetes es la principal unidad para organizar el software en ROS. Un paquete puede contener nodos (la unidad de ejecución en ROS), librerías, data sets o ficheros de configuración, entre otros.
- Ficheros de manifiesto (manifest.xml): los manifiestos proporcionan meta datos de un paquete, incluyendo información de licencia, dependencias con otros paquetes y opciones de compilación.
- Stacks: Un stack es un conjunto de paquetes que proporcionan una funcionalidad. Un ejemplo de stack es la *navigation stack* disponible en los repositorios de ROS.
- Ficheros de manifiesto de un stack (stack.xml): de forma similar al manifiesto de un paquete, este tipo de ficheros proporcionan información información del stack, como la versión, la licencia y las dependencias.
- Descripción de mensajes: estos ficheros describen los mensajes a utilizar por los nodos, definiendo la estructura de datos del mismo. Se suelen ubicar dentro de la carpeta *msg* de un paquete y tienen la extensión ".msg".
- Descripción de servicio: se utilizan para describir los servicios, definiendo la estructura de datos para la petición y la respuesta del servicio. Se suelen ubicar dentro de la carpeta *srv* de un paquete y tienen la extensión ".srv".

### 3.1.2. Grafo de computación

Como se ha comentado anteriormente, el grafo de computación de ROS es una red de comunicación punto a punto de procesos. En este grafo se pueden distinguir nodos, Máster, servidor de parámetros, mensajes, servicios, *topics* y *bags*.

- Nodo: un nodo es un proceso que realiza una determinada funcionalidad. Mediante los nodos podemos diseñar un sistema modular de manera que el sistema de control del robot requiera la participación de varios nodos que puedan ser reutilizables. Como ejemplo, un nodo puede encargarse de la planificación de movimientos, otro de un sensor láser, un nodo para controlar los motores de las ruedas y otro para realizar la localización. Un nodo se escribe mediante una librería cliente de ROS, como *rosccp* para programar en C++ o *rospy* para hacerlo en Python. Los nodos se lanzan introduciendo en una consola "rosrun paquete nombre_nodo".
- Máster: El Máster de ROS permite hacer registro de nombres y búsqueda al resto del grafo de computación. Almacena información de registro de topics y servicios de nodos ROS. Los nodos se comunican con el Máster y pueden recibir información sobre otros nodos para conectarse con ellos si es oportuno. Cabe señalar que los nodos se conectan entre sí y no a través del máster, éste último sólo realiza una función similar a la de un servidor DNS de Internet. Los nodos que se subscriben a un topic pedirán conexiones a nodos que publiquen ese topic y se conectarán a ellos a través de un protocolo. El protocolo más común es TCPROS, basado en sockets TCP/IP. Tecleando "roscore" en una terminal se inicia el máster.
- Servidor de parámetros: permite almacenar datos, de forma que se pueden actualizar y consultar por cualquier nodo. Se inicia junto con el máster.

- Mensajes: son una forma de comunicación entre nodos. Los mensajes se definen como una estructura de datos en un fichero de definición de mensajes. Los tipos que se permiten son desde tipos básicos como enteros o reales a estructuras C más complejas y vectores. Existe una gran variedad de mensajes ya definidos en diferentes paquetes, pero el usuario puede definir mensajes propios que se adapten a sus necesidades combinando los ya existentes o creándolos desde cero.

- Topic o tópico: el topic es el canal de comunicación entre nodos. De esta manera un nodo publica un mensaje en un topic al que se ha suscrito un segundo nodo para leer los mensajes allí depositados. Un topic se identifica por un nombre. Puede haber varios publicadores en un topic, así como varios suscriptores. Tampoco hay restricciones de este tipo a nivel de nodo, de forma que un nodo puede suscribirse y/o publicar en varios topics. Normalmente los nodos suscriptores o publicadores no conocen si existen otros nodos utilizando el topic que están usando. En la figura 30 se puede ver un esquema de esta comunicación en la parte inferior.

- Servicios: además de la comunicación mediante mensajes, ROS también permite comunicación de tipo petición-respuesta. Este tipo de comunicación se realiza mediante servicios. Los servicios se definen especificando un par de estructuras de mensajes: una para la petición y otra para la respuesta. De esta manera, un nodo ofrece un servicio con un nombre específico y otro nodo puede solicitar dicho servicio mediante un mensaje de petición. También se puede ver este sistema de comunicación como una llamada a procedimiento remoto (RPC). En la figura 30 se muestra la relación entre nodos mediante un servicio en la parte superior.

- Bags: los *bags* son un formato para almacenar y reproducir mensajes ROS. Son especialmente útiles para almacenar datos de sensores para después probar algoritmos de forma off-line.

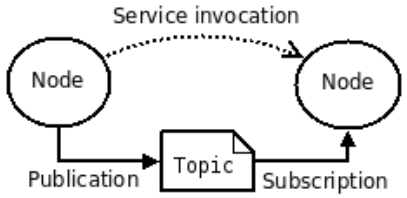


Figura 30: ROS: Comunicación entre nodos.

## 3.2. Stack PeopleExperimental

Este stack [37, 38] está desarrollado por WillowGarage y compuesto por cuatro paquetes: LegDetector, pedestrianDetectorHOG, HeightTracker y PeopleTracking-Filter.

### 3.2.1. LegDetector

En este paquete se implementa el nodo "legDetector". Este nodo detecta piernas a partir de las medidas de un sensor láser. El nodo espera la llegada de mensajes procedentes del sensor por el topic "scan", y mensajes procedentes del nodo "people_tracking_node" enviados por el topic "people_tracker_filter". También realiza tracking de las piernas detectadas a partir de un mensaje del nodo "people_tracking_filter", asociando piernas a la persona que se está siguiendo.

**Procesamiento de los datos del láser** Cuando recibe un mensaje procedente del láser examina las muestras recibidas y elimina las que sobrepasan el rango del láser o están en la máscara. Con las muestras restantes se forman grupos (o segmentos) de manera que quedan en un mismo conjunto las muestras que están más próximas entre sí (especificando un umbral para esta distancia de salto). Los conjuntos que tienen menos de cinco elementos, se eliminan.

Una vez procesados los datos del láser se borran de entre las características que tenía almacenadas de otras iteraciones las que son de hace más de 0.5 segundos, y de las restantes se estima su posición actual mediante un filtro de Kalman Extendido.

El siguiente paso es calcular cuáles son candidatos a ser una pierna a partir de los clusters extraídos del láser. Para ello se calcula una serie de características de los segmentos y se clasifican mediante un clasificador CvRTree (Random Trees [39], implementados en la librería OpenCV), de forma que si el resultado de la clasificación es positivo, se obtiene un conjunto de datos candidato a ser una pierna. En total se calculan 14 características de cada segmento:

1. desviación típica
2. promedio de la desviación media de la mediana.
3. distancia de salto entre el segmento actual y el anterior.
4. distancia de salto entre el segmento actual y el posterior.
5. ancho del segmento.
6. linealidad.
7. circularidad.
8. radio.
9. longitud del contorno.
10. regularidad del contorno.
11. curvatura media.
12. diferencia angular media.
13. IAV (Inscribed Angle Variance).
14. desviación típica del IAV.

Analizando estas características se comprueba que las doce primeras características son las que se utilizaban en [18], exceptuando el número de puntos y la velocidad media. Las dos últimas características que se calculan en este caso corresponden a la técnica presentada por Xavier et al en [17].

Una vez clasificados los segmentos, se pasa a la fase de tracking. Para cada uno de los candidatos se busca el tracker existente más cercano dentro de un umbral y se actualiza su filtro de Kalman asociado. Siempre se considera la pareja tracker-candidato con una distancia menor, por lo que si hubiera más de un candidato asociado a un tracker, se consideraría que el correcto es el que está más cerca. Si no se ha encontrado ningún track, se inicia uno nuevo ya que se supone que es una nueva pierna que ha entrado en el escenario.

Por último, se publican los resultados: en el topic "people_tracker_mesurements" se publica la posición de las piernas (un mensaje por cada pierna que esté etiquetada) y en "kalman_filtered_cloud" se obtiene una nube de puntos con las posiciones de todas las piernas detectadas.

**PeopleCallback** Al recibir un mensaje de tipo people_msg::PositionMesurements desde el topic "people_tracker_filter" (publicado por el nodo people_tracking_node que se comentará más adelante) se llama a la callback PeopleCallback. Esta callback se encarga de asociar, si existe, un par de piernas a la persona que se está siguiendo asignando etiquetas. Para ello:

- Si hay un par de piernas con la etiqueta correcta y dentro de la distancia máxima, esa persona ya tenía un par de piernas asociadas y no se hace nada.
- Si hay una pierna con la etiqueta correcta y dentro de la distancia máxima, se busca un par entre las piernas no etiquetadas que está dentro de la distancia máxima. Si no encuentra un par, solo se etiqueta una pierna.
- Si no hay piernas con la etiqueta adecuada y dentro de la distancia máxima, se busca un par de piernas sin etiquetar que estén cerca de la persona y cerca entre ellas, y se les asigna la etiqueta. Si no se encuentran dos piernas, pero sí que se encuentra una que cumpla las condiciones, se devuelve sólo esa pierna.

### 3.2.2. PedestrianDetectorHOG

En este paquete se implementa un nodo que hace uso del "HOGDetector" disponible en OpenCV. El nodo simplemente se suscribe a los topics por los que se envían las imágenes para después ejecutar el detector. Los resultados no se publican por ningún topic, aunque sí que se crea una imagen en la que se han añadido los rectángulos que marcan las detecciones a la imagen original.

El nodo está preparado para funcionar en sistemas monoculares y estéreo. Aunque en el método original de Dalal y Triggs solo se necesita una imagen, el nodo se suscribe a cuatro topics para las imágenes de un sistema estéreo y la información de calibración de las dos cámaras. Sin embargo, solamente se utiliza una de las dos imágenes, por lo que es posible que el nodo esté en desarrollo. En el caso de utilizar un sistema monocular se crean los mensajes restantes como mensajes vacíos y se ejecuta el mismo procedimiento que para el método estéreo.

El detector HOG incluido en OpenCV implementa el método presentado por Dalal y Triggs en [4], y devuelve un conjunto de rectángulos que indican la posición de las personas detectadas. Los principales parámetros que se pueden modificar son:

- Umbral para la distancia entre las características y el plano del clasificador SVM.
- Desplazamiento o paso de la ventana, por defecto se establece a 8 píxeles. Debe ser múltiplo del tamaño de bloque.
- Coeficiente para el incremento del tamaño de la ventana de detección.
- Coeficiente para agrupar detecciones similares del mismo objeto (y evitar que el mismo objeto esté cubierto por más de un rectángulo)

### 3.2.3. HeightTracker

Este nodo a partir de una nube de puntos estéreo realiza el seguimiento de personas usando el método Mean shift a partir de la proyección de la nube de puntos sobre el plano. Se suscribe a dos topics: "stereo/cloud", para recibir una nube de puntos, y "people_tracker_filter", para controlar la distancia al objeto que se está siguiendo.

Cuando se recibe una nube de puntos se preprocesa extrayendo los puntos que están fuera del área de recorte. A continuación se proyectan los puntos restantes en el plano correspondiente al XY, se realiza el Mean shift tracking y se calcula y envía la medida de la posición por el topic "people_tracker_mesurements". Mean shift es un método para localizar el máximo de una función de densidad a partir de datos discretos de dicha función. Este método va iterando para encontrar el centro del objeto dada la proyección de los puntos y una posición inicial de la ventana de búsqueda. Las iteraciones continúan hasta que el centro de la ventana de búsqueda se mueve menos que el valor dado y/o hasta que la función ha hecho el número máximo de iteraciones.

El segundo topic al que se suscribe el nodo es "people_tracker_filter" y se reciben posiciones. A partir de la posición recibida, se calcula la distancia al centro de la ventana de seguimiento, y si esta distancia es grande (mayor que 1000) se considera que se ha perdido el rastro del objeto y se inicia uno nuevo.

### 3.2.4. PeopleTrackingFilter

En este paquete se implementa el nodo "people_tracking_node", utilizado para unir las detecciones realizadas por los nodos implementados en los paquetes anteriores. Este nodo se ejecuta de forma iterativa, encargándose de actualiza los trackers ya existentes. También se calcula la calidad del tracker y si es menor o igual a cero, se descarta. Para cada tracker utiliza un filtro de Kalman extendido (EKF), aunque también se podría utilizar un filtro de partículas. También publica los nuevos valores por el topic "people_tracker_filter", suscrito por el nodo de detección de piernas.

El nodo se suscribe al topic "people_tracker_mesurements" por el que se reciben mensajes con posiciones. Cuando se recibe uno de estos mensajes se utiliza este valor para actualizar el filtro asociado. Si no tiene un filtro asociado se crea uno nuevo o se asocia a uno existente que esté próximo. En este método también se publican las posiciones mediante un tipo de mensaje de visualización para poder comprobar los resultados en un visor, como por ejemplo "rviz".

## 3.3. Stack iri_perception

El stack iri_perception [40] está desarrollado por el Institut de Robòtica e Informàtica Industrial de la Universidad Politécnica de Cataluña. El stack engloba una serie de algoritmos de percepción útiles en robótica. A continuación se describen los paquetes que hacen referencia a detección de personas.

### 3.3.1. iri_laser_people_detection

El nodo implementado en este paquete se basa en un sensor láser. En cuanto se recibe un mensaje del sensor por el topic correspondiente se divide el escáner en segmentos según proximidad y extrae características geométricas para clasificarlos mediante un clasificador AdaBoost. En total se extraen 14 características:

1. numero de puntos del segmento.
2. desviación típica.
3. promedio de la desviación media de la mediana.
4. distancia de salto entre el segmento actual y el anterior.
5. distancia de salto entre el segmento actual y el posterior.
6. ancho del segmento.
7. linealidad.
8. circularidad.
9. radio de la circunferencia que se ajusta al segmento.
10. longitud del contorno.
11. regularidad del contorno.
12. curvatura media.
13. diferencia angular media.
14. anchura del par formado por el segmento y su segmento más cercano.

Las trece primeras características coinciden con las trece primeras propuestas en el trabajo de Arras et al. en [18]. Por último, se calcula la posición de las personas a partir de los resultados de la clasificación de los segmentos y se publican los resultados por el topic “people”.

### 3.3.2. iri_camera_people_detection

En este paquete se implementa un nodo que combina láser y visión monocular en paralelo para la detección de personas, por lo que está suscrito a dos topics para recibir la información de los sensores correspondientes.

Cuando se recibe la imagen y las detecciones láser, éstas últimas se transforman al sistema de coordenadas de la imagen. Después se detectan las personas a partir de la imagen mediante un HOGDescriptor de OpenCV creado con los parámetros por defecto. A continuación se ponderan las detecciones obtenidas mediante

el sensor láser basándose en las detecciones obtenidas en la imagen, calculando la distancia entre el centro de la detección de la imagen y los datos del láser. Con esto se comprueba si las detecciones del láser corresponden con alguna de las detecciones realizadas en la imagen. Tras esto se validan los datos del láser a partir de los pesos, de forma que solo se validan las detecciones cuya ponderación pasa un umbral establecido. Por último, se añaden a las detecciones validadas las detecciones láser que están próximas a la cámara o que están fuera del rango de la misma. Las posiciones de personas detectadas y validadas se publican por el topic "image detection".

El nodo también realiza funciones de visualización de resultados, señalando en la imagen tanto las detecciones obtenidas por el método HOG como las detecciones láser, distinguiendo entre validadas y no validadas.

### 3.3.3. iri_laser_people_labeler_trainer

Este nodo no realiza labores de detección o tracking, sin embargo se utiliza para reproducir un fichero de bag de ROS para etiquetar piernas de personas procedentes de medidas de un láser, generando tres ficheros con los resultados. A su vez también permite el entrenamiento de un clasificador AdaBoost.

### 3.3.4. iri_laser_multi_detection

Este nodo recibe las detecciones realizadas por dos nodos como los implementados en el paquete "iri_laser_people_detection", de forma que tiene dos listas de personas detectadas. También se suscribe a dos topics para recibir la información de los sensores, para realizar una calibración y saber la posición relativa entre los dos sensores y así poder comparar las dos listas realizando las transformaciones necesarias.

Cuando el nodo recibe las dos listas de personas las fusiona por proximidad: para cada persona de cada una de las listas se calcula la distancia entre ellas, y si es menor que una distancia de error se considera que son la misma. También se añaden a la lista de personas finales aquellas detecciones de las listas iniciales que no se han fusionado con otra detección. En consecuencia se obtiene una sola lista en la que están todas las detecciones, eliminando los posibles duplicados debido al solape de sensores. Por último se publica en los topics "peopleMarkers" y "people" los resultados obtenidos.

### 3.3.5. iri_people_tracking

Este nodo se suscribe a dos topics. Del primero de ellos (topic "odom") se obtienen los datos de la odometría del robot. Desde el segundo topic al que se suscribe, el topic "people_detection", se reciben las posiciones de las personas detectadas por otros nodos. Con estos datos se estiman y actualizan las posiciones de las personas detectadas mediante filtros de partículas. Una vez realizado esto se publican las posiciones estimadas en los topics "peopleSet" y "particleSet" (para visualización de resultados).

## 3.4. Stack iri_navigation

Este stack está desarrollado por el Institut de Robòtica e Informàtica Industrial de la Universidad Politécnica de Cataluña, de la misma manera que el visto en

el apartado anterior. Dentro de este stack se van comentar dos paquetes que se encargan de realizar el seguimiento de una persona.

### 3.4.1. iri_people_follower

El nodo se suscribe al topic "target_pose" por el que le llegan posiciones objetivo a seguir. Cuando llega un mensaje por este topic se establece como posición destino la posición indicada en el mensaje, teniendo en consideración una distancia de margen por seguridad.

El nodo a su vez puede estar en una serie de estados para controlar el seguimiento:

- libre ("idle"): en este estado el nodo está a la espera de una nueva petición. En cuanto llega se pasa al estado de solicitud de movimiento.
- solicitud de movimiento de la base: se espera la llegada de una posición por el topic "target_pose". Cuando esto ocurre, se actualiza la posición objetivo, y se llama a la función que realiza el movimiento del robot para que se vaya a la posición indicada, pasando al estado de seguimiento. Si se produce algún problema se pasa al estado libre.
- seguimiento: si llega el evento que indica que se ha llegado a la posición objetivo se pasa al estado "éxito". Por el contrario si se da el evento de "adelanto" se pasa al estado del mismo nombre. En cambio, si se ha recibido una nueva posición se calcula si se ha de actualizar el objetivo, y si hay que actualizar se pasa al estado correspondiente. Por último, si estando en este estado no se reciben nuevos mensajes con la posición a alcanzar durante un número determinado de iteraciones, se pasa al estado de adelanto.
- éxito: Se espera la llegada del evento que indica que se ha seguido al objetivo con éxito y se pasa al estado libre.
- actualización: Se cancela el objetivo actual y se pasa al estado de solicitud de movimiento.
- adelanto: Se cancela el objetivo actual y se pasa al estado libre.

### 3.4.2. iri_people_follower_client

Este nodo se suscribe al topic "people_tracker". Cuando llega un mensaje por este topic se calcula entre las personas recibidas la más cercana y que esté parada. La velocidad de la persona se proporciona en el mensaje, por lo que para saber si la persona está parada se comprueba que dicha velocidad es menor que un umbral establecido. Cuando ha encontrado la persona adecuada, se construye un mensaje con su posición y se envía por el topic "target_pose", con lo que el nodo "iri_people_follower" puede comenzar al seguimiento de la persona. Si no se encuentra ningún objetivo, se cancela el servicio.

## 3.5. Stack twoLevelMTTDStack

El stack "twoLevelMTTD_stack" [42] está desarrollado por Tinne De Laet con el objetivo de detectar y seguir múltiples objetivos utilizando el algoritmo "twoLevelMTTL" (multi target tracking and localization). Este algoritmo está desarrollado en componentes de Orocos (Open Robot Control Software) pero está integrado en

ROS gracias al paquete “MTTD_interface”, que utiliza una versión modificada del paquete de ROS para la integración con Orocos. La estructura resultante es la que se muestra en la Figura 31, donde se ve a la parte izquierda los componentes de Orocos y su interconexión y a la parte derecha los nodos y topics de ROS.

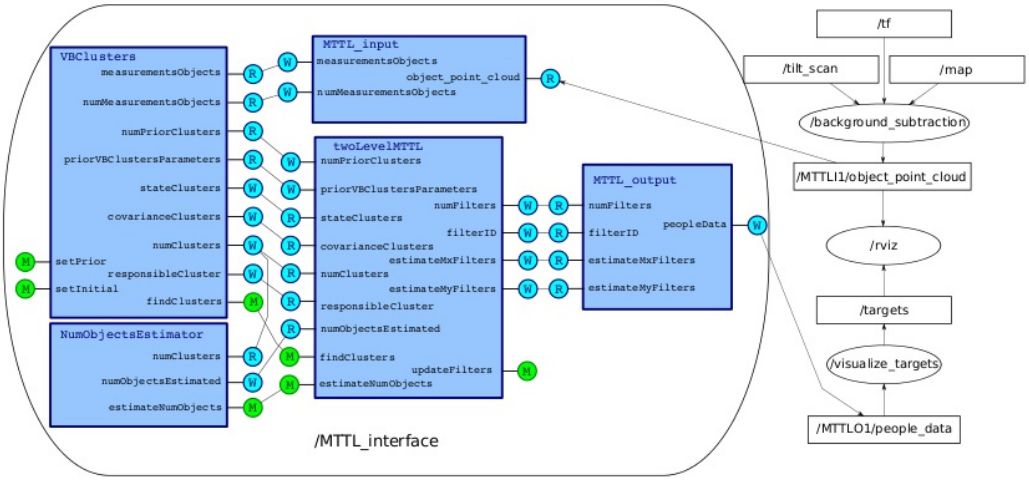


Figura 31: Estructura de “2 level MTTL”.

El objetivo final es obtener, a cualquier instante de tiempo t, estimaciones del estado de un número desconocido de objetivos. Con este fin el algoritmo parte de las medidas de los sensores (uno o varios, que pueden ser cámaras, láser, etc.) cuyos datos se preprocesan extrayendo el fondo. Tras este paso se agrupan las medidas en clusters para posteriormente realizar la asociación de datos y el tracking. Este proceso se realiza de forma iterativa, utilizando en cada iteración la información de la etapa de asociación de datos y tracking de la iteración anterior.

La sustracción de fondo se hace a partir de un entorno conocido que se obtiene mediante el servicio “static_map” del nodo de ROS “map_server”, dentro del paquete del mismo nombre del stack “navigation”. Una vez obtenido el mapa del entorno la nube de puntos obtenida mediante los sensores se transforma al sistema de referencia del mapa. Después elimina todos aquellos puntos que estén fuera del área del mapa o que no estén libres en el mapa. Para que se considere que un punto está libre hay que comprobar que la posición del mapa correspondiente y las de al rededor están libres (por defecto, se comprueban $\pm 8$ posiciones en el mapa). Los componentes que se encargan de esta función son “PreparateSeparatedMesurement” y “SeparateObjects”.

El nivel de clustering de bajo nivel agrupa las medidas preprocesadas en M clusters, cada uno de ellos definidos como una gausiana. El número total de clusters no se conoce de inicio, pero se determina de forma automática a partir de los datos. Este clustering utiliza información del clustering de alto nivel del instante anterior (posiciones de los objetivos y formas del estado estimado actual) como probabilidad a priori y un modelo de predicción. El componente que implementa el clustering es VBClusters, proporcionando tamaños y posiciones de los clusters usando “Variational Bayesian Cluster Finding”; mientras que la estimación del número de objetos la realiza el componente “NumberObjectsEstimator”, que implementa un filtro discreto para realizar esta estimación.

La fase de clustering de alto nivel asocia clusters con trackers en vez de con medidas individuales. Este nivel es el que realiza el seguimiento y detección de nuevos objetivos. Para ello debe incluir asociación de datos para asociar las medidas con los targets, estimar el número de targets, estimar la posición y la forma de los targets, ser robusto frente a oclusiones y mantener una correcta identificación de los targets. El problema de la asociación de datos se basa en un algoritmo similar al "joint probabilistic data-association filter" (JPDAF) si se asumen densidades con distribución gausiana, asociando a cada tracker un filtro de Kalman Extendido. Si están modeladas como distribuciones de Monte Carlo, el algoritmo es similar al algoritmo de tracking "Sample-based Joint Probabilistic Data Association" (SJPDAF) y a cada tracker se le asocia un filtro de partículas.

El componente principal que realiza este algoritmo recibe el nombre de "twoLevelMTTL" y es el que se encarga de lanzar los métodos de los otros componentes cuando son necesarios y mantener los trackers individuales. Hay cuatro versiones de este componente según el filtro de asociación de datos y el modelo utilizado:

- EstimationGaussian: Implementa el algoritmo para el caso en el que se usa el filtro de kalman extendido para para cada tracker individual y como filtro de asociación de datos un JPDAF.
- EstimationMCPdf: Implementa el algoritmo para el caso en el que se usan filtros bootstrap para cada tracker individual. Como filtro de asociación de datos se utiliza una función de densidad de probabilidad de Monte Carlo (MCPdf), con modelo de medida con ruido gausiano aditivo.
- EstimationRBBM: el mismo caso que en el anterior, con la diferencia del modelo de medida, que ahora es una función de densidad de probabilidad condicional no lineal para modelo de haz bayesiano.
- EstimationRBBM_Radius: el mismo caso que los dos anteriores, pero variando una vez más el modelo de medida por una función de densidad de probabilidad condicional no lineal para modelo de haz bayesiano con radio desconocido.

## 3.6. Paquete "head_follow_people"

En este paquete [43] se implementa el nodo "train_head", encargado de realizar seguimiento de las posiciones recibidas. Este nodo se suscribe al topic "people_tracker_mesurements" y cuando recibe un mensaje con una posición de una persona se comprueba si ésta ya está en la lista interna de personas a seguir. Si ya está en la lista se actualiza su información con la recibida, en cambio, si no está en la lista, se añade la nueva persona a seguir.

El nodo publica en dos topics: "head_goal" y "target_cloud", publicando respectivamente la posición objetivo a seguir y las posiciones que tiene almacenadas. En cada iteración de la ejecución de este nodo se visitan todos los trackers que se tienen en ese instante y se eliminan los antiguos (de más de 0.5 segundos). Con los puntos restantes se genera una matriz que representa las posiciones a seguir mediante una distribución gaussiana.

## 3.7. Paquete "people_detector_node"

El nodo implementado en este paquete [44] encapsula el método de detección de piernas a partir de características geométricas presentado por Óscar Mozos en [18].

El nodo se suscribe a un topic para recibir mediciones de un sensor láser. Cuando recibe un mensaje por este topic se divide en segmentos el contenido del mismo según proximidad, descartando los que tienen menos de tres elementos. Con los segmentos restantes se ejecuta el algoritmo de detección.

Los resultados se publican por el topic "output_scan_topic", de forma que en un visor como rviz se puede comprobar el resultado obtenido.

## 3.8. OpenCV

OpenCV (Open Source Computer Vision) [45, 46, 47] es una librería de funciones para visión artificial en tiempo real. Esta librería es de código abierto bajo licencia BSD e incluye más de 2500 de algoritmos para visión, como funciones generales para el procesamiento de imágenes, transformaciones, segmentación, calibración de cámaras, procesamiento estéreo o algoritmos de detección y reconocimiento. Aunque no está específicamente destinada a la detección de personas, estos algoritmos proporcionados pueden resultar de utilidad en este tipo de aplicaciones.

Esta librería apareció en 1999, cuando apareció su primera versión, y actualmente aún está en desarrollo, estando al cargo del mismo Willow Garage. La última versión lanzada es la 2.4.2 (Julio de 2012). Proporciona un API para los lenguajes de programación C, C++ y Python, con el API de Java en desarrollo. La librería está disponible para Windows, Linux, Android y Mac.

OpenCV está dividida en una serie de módulos, entre los cuales se encuentran:

- core: módulo que define las estructuras de datos básicas y funciones básicas que utilizan otros módulos.
- imgproc: módulo para procesamiento de imágenes, incluyendo filtrado lineal y no lineal, transformaciones geométricas, conversión de espacio de colores y cálculo de histogramas, entre otros.
- video: módulo para el análisis de secuencias de video. Proporciona estimación de movimiento, sustracción de fondo y algoritmos de seguimiento de objetos.
- calib3d: algoritmos de calibración de cámaras monoculares o estéreo, estimación de posición de objetos, algoritmos de correspondencia estéreo y elementos de reconstrucción 3D. Es de especial interés para la aplicación que se está considerando, por lo que se explicará en más detalle a continuación.
- features2d: detectores de características, descriptores y comparadores de descriptores.
- objdetect: detección de objetos e instancias de las clases predefinidas, como caras, ojos y coches.
- highgui: interfaz para la captura de vídeo, imagen y codecs de vídeo. También proporciona herramientas de interfaz de usuario simples.
- gpu: algoritmos de otros módulos acelerados mediante GPU.

### 3.8.1. Calibración

Ninguna cámara es perfecta y debido a su construcción se dan, principalmente, dos tipos de distorsiones en la imagen: la distorsión radial y la distorsión tangencial.

Mediante la calibración de las cámaras se pueden compensar sus efectos, proceso necesario para poder relacionar los píxeles de la imagen con los puntos del mundo real. OpenCV posee funciones para calcular los parámetros necesarios para ello en el módulo "calib3d". A continuación se van a comentar las principales funciones que obtienen estos valores.

La función que realiza la calibración en OpenCV es cvCalibrateCamera2. Esta función calcula los parámetros intrínsecos de una cámara a partir de imágenes tomadas de un objeto con puntos claramente identificables. Para ello el método más común es utilizar un patrón de calibración de tipo tablero de ajedrez con un número conocido de casillas de un determinado tamaño, de forma que los puntos a identificar serán las esquinas de las casillas. Se utilizan patrones de este tipo por su alto contraste, lo que facilita la búsqueda de puntos de interés. Se deberán tomar varias imágenes de este patrón en diferentes posiciones e inclinaciones, intentando cubrir toda el área de visión de la cámara. Es preferible que el patrón de calibración tenga algo de espacio en blanco al rededor del tablero de ajedrez para que la detección sea más robusta. La función OpenCV que busca estos puntos es:

```
int cvFindChessboardCorners(const void* image,
                            CvSize patternSize,
                            CvPoint2D32f* corners,
                            int* cornerCount=NULL,
                       int flags=CV_CALIB_CB_ADAPTIVE_THRESH)
```

en la que se indica la imagen en la que ha de buscar los puntos y el tamaño del patrón de calibración (puntos por fila y puntos por columna), y devuelve las coordenadas y el número de los puntos encontrados. El tamaño del patrón de calibración se especifica según el número de esquinas internas, por lo que en un tablero de 9x6 casillas el tamaño seria de 8x5. Como flags se pueden utilizar tres valores:

- CV_CALIB_CB_ADAPTIVE_THRESH: utiliza una umbralización adaptativa para convertir la imagen a blanco y negro (en función de un umbral calculado según el brillo medio de la imagen)
- CV_CALIB_CB_NORMALIZE_IMAGE: normaliza el gamma de la imagen con "EqualizeHist" antes de aplicar un umbralizado adaptativo o fijo.
- CV_CALIB_CB_FILTER_QUADS: utiliza un criterio adicional para filtrar los falsos cuadrados extraídos en la fase de obtención de contornos.

La posición de los puntos es algo aproximada, por lo que hay que ajustar su valor usando la función "FindCornerSubPix":

```
void cvFindCornerSubPix(const CvArr* image,
                        CvPoint2D32f* corners,
                        int count, CvSize win,
                        CvSize zero_zone,
                        CvTermCriteria criteria)
```

cuyos parámetros son:

- image: la imagen de entrada.
- corners, count: las coordenadas de las esquinas y el número de las mismas. Tras finalizar la función, las nuevas coordenadas se devuelven en "corners".

- win: la mitad de la longitud de un lado de la ventana de búsqueda.
- zero_none: la mitad de un lado de la región muerta en el medio de la zona de búsqueda. A veces se utiliza para evitar posibles singularidades en la matriz de autocorrelación.
- criteria: criterio para terminar el proceso iterativo. Puede ser un valor de precisión y/o un número máximo de iteraciones.

Esta función itera hasta encontrar la localización exacta de los subpíxeles de las esquinas o de los puntos de silla de la imagen. Para ello se basa en la observación de que cada vector del centro $q$ al punto $p$ situado en la vecindad de $q$ es ortogonal al gradiente de la imagen en $p$ (figura 32)

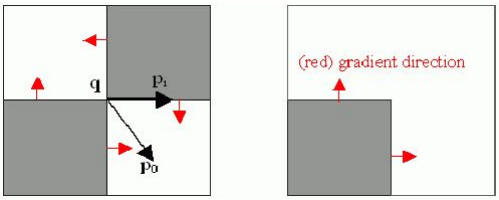


Figura 32: Localización exacta de las esquinas con OpenCV

Es posible que estos algoritmos no encuentren todas las esquinas (o incluso que no encuentre ninguna) por lo que para realizar una calibración adecuada habrá que considerar las imágenes en las que se han encontrado todas las esquinas del tablero. Para comprobar las esquinas encontradas en el tablero se puede utilizar la siguiente función:

```
void cvDrawChessboardCorners(CvArr* image,
                             CvSize patternSize,
                             CvPoint2D32f* corners,
                             int count,
                             int patternWasFound)
```

en la que hay que especificar la imagen, el tamaño del patrón de calibración, la posición y el número de las esquinas y si se ha encontrado completamente o no el patrón completo. Esta función dibuja las esquinas detectadas con círculos rojos y, si se ha encontrado el patrón de calibración completo, conecta las esquinas con líneas de colores. En la figura 33 se puede ver un ejemplo en el que se han detectado completamente todas las esquinas del patrón de calibración utilizado.

Una vez se ha logrado obtener un conjunto apropiado de imágenes ya se puede realizar la calibración mediante la función:

```
double cvCalibrateCamera2(const CvMat* objectPoints,
                          const CvMat* imagePoints,
                          const CvMat* pointCounts,
                          CvSize imageSize,
                          CvMat* cameraMatrix,
                          CvMat* distCoeffs,
                          CvMat* rvecs=NULL,
                          CvMat* tvecs=NULL,
                          int flags=0)
```

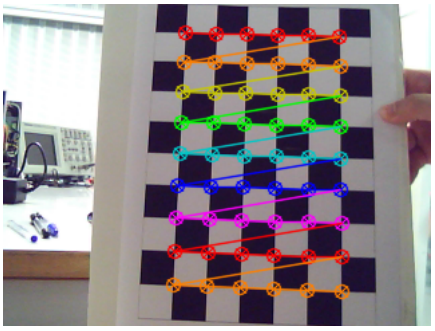

Figura 33: Detección de las esquinas del patrón de calibración

cuyos parámetros son:

- objetctPoints: los puntos obtenidos del patrón de calibración en el sistema de coordenadas del modelo.
- imagePoints: los puntos obtenidos del patrón de calibración en el sistema de coordenadas de la cámara.
- pointCounts: un vector con el número de puntos en cada imagen en particular.
- imageSize: tamaño de la imagen, utilizado para inicializar la matriz intrínseca de la cámara.
- cameraMatrix: la matriz intrínseca de la cámara que se obtiene como resultado.
- distCoeffs: los vectores de coeficientes de distorsión (cuatro o cinco valores).
- rvecs: los vectores de rotación estimados para cada imagen del patrón.
- tvecs: los vectores de traslación estimados.

La función cvCalibrateCamera2 calcula los parámetros intrínsecos y extrínsecos de una cámara a partir de los puntos detectados en las imágenes. Primero se calculan unos valores iniciales de los parámetros intrínsecos, estableciendo los coeficientes de distorsión a cero. A continuación se estima la posición inicial de la cámara como si los parámetros intrínsecos ya se conocieran. Esto se realiza de forma interna mediante la función “FindExtrinsicCameraParams2”, que estima dicha posición como la posición que minimiza el error de proyección. Además de la posición inicial también calcula la matriz de la cámara y los coeficientes de distorsión.

En el caso de utilizar estéreo, se ha de utilizar esta función para cada una de las dos cámaras y después la función cvStereoCalibrate:

```
double cvStereoCalibrate(const CvMat* objectPoints,
                         const CvMat* imagePoints1,
                         const CvMat* imagePoints2,
                         const CvMat* pointCounts,
                         CvMat* cameraMatrix1,
                         CvMat* distCoeffs1,
                         CvMat* cameraMatrix2,
                         CvMat* distCoeffs2,
                         CvSize imageSize,
                         CvMat* R, CvMat* T, CvMat* E=0,
                         CvMat* F=0,
                         CvTermCriteria term_crit,
                         int flags=CV_CALIB_FIX_INTRINSIC)
```

Esta función estima la transformación entre las dos cámaras: a partir de las posiciones relativas de un objeto a cada una de las cámaras, calcula la transformación y rotación relativas entre las dos cámaras. Lo hace minimizando el error de proyección total para todos los puntos disponibles de ambas cámaras. Aunque la función podría realizar la calibración completa de las dos cámaras, es recomendable calibrar primero las dos cámaras por separado, como ya se ha comentado. Opcionalmente también calcula la matriz fundamental y la esencial. La matriz esencial tiene información de la traslación y rotación relativas entre las dos cámaras, y la matriz fundamental contiene la misma información que la esencial además de información de los parámetros intrínsecos de las dos cámaras. En consecuencia, la matriz fundamental relaciona las dos cámaras en relación a las coordenadas de la imagen (píxeles). La matriz fundamental también se puede obtener mediante la función "cvFindFundamentalMat".

Figura 34: Imagen distorsionada (izquierda) y corregida (figura de [47])

Una vez obtenida la calibración se puede solucionar el problema de la distorsión de las imágenes. Un ejemplo de este efecto es el mostrado en la figura 34. En la imagen izquierda se ve la imagen "raw" distorsionada y en la imagen derecha se ve que se ha corregido este efecto. Se puede ver el efecto de la distorsión en la ventana superior, cuya arista se ve curvada debido a este efecto. La manera de corregir la imagen en OpenCV es mediante la función:

```
void cvUndistort2(const CvArr* src, CvArr* dst,
                  const CvMat* cameraMatrix,
                  const CvMat* distCoeffs,
                  const CvMat* newCameraMatrix=0)
```

que transforma una imagen distorsionada (src) en una imagen sin distorsionar

(dst) conociendo la matriz de la cámara y los coeficientes de distorsión obtenidos de la calibración.

### 3.8.2. Disparidad y reconstrucción 3D

El módulo "calib3D" también proporciona herramientas para reconstruir entornos en tres dimensiones a partir de un sistema de visión estéreo.

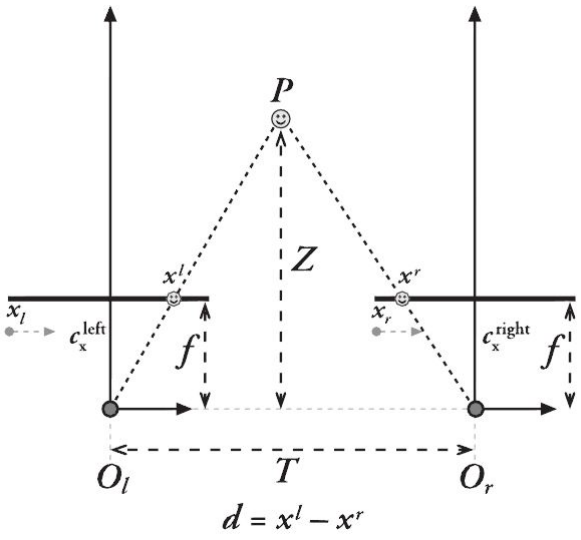


Figura 35: Cálculo de la profundidad por triangulación (figura de [47])

Mediante un proceso de triangulación se puede saber la profundidad a la que está un determinado píxel. Si se suponen dos imágenes sin distorsionar y rectificadas (alineadas horizontalmente) entonces se puede conocer la profundidad de un determinado píxel mediante triángulos semejantes. En la figura 35 se muestra un ejemplo en el que el punto P tiene dos píxeles correspondientes $x^l$, $x^r$ (como las coordenadas en la imagen izquierda y derecha, respectivamente). Según la notación de la imagen y por triángulos semejantes se obtiene la siguiente expresión:

$$Z = \frac{fT}{x^r - x^l} \tag{1}$$

De esta manera, si calculamos la disparidad como $d = x^r - x^l$, se puede comprobar que la disparidad es inversamente proporcional a la distancia. Como consecuencia, un sistema estéreo tiene mayor resolución de profundidad en objetos próximos (donde su disparidad será mayor) que en objetos alejados (que tendrán un nivel de disparidad muy bajo).

Como se ha comentado, para poder calcular la profundidad se necesitan imágenes no distorsionadas y rectificadas. El problema de la distorsión se ha comentado en la sección anterior: mediante la calibración se obtienen los parámetros necesarios para solucionar este problema. Sin embargo, falta resolver el problema de la rectificación de las imágenes para que ambas estén horizontalmente alineadas. Mediante la rectificación el problema de la correspondencia es mucho más fácil de resolver debido a que solo se ha de buscar el píxel correspondiente en una línea de la imagen en vez de en la imagen completa, lo cual facilita a su vez el cálculo de la disparidad. Para ello

son necesarias las líneas epipolares, además de la matriz fundamental (comentada en la sección anterior).

El epipolo es la proyección del centro de proyección de la otra cámara en el plano de la imagen. La restricción epipolar determina que dado un punto $x^r$ en la imagen de la derecha su correspondiente en la imagen izquierda debe estar sobre la línea epipolar resultante de intersectar el plano epipolar y el plano de la imagen derecha. El plano epipolar es el formado por el punto real que se está visualizando y los epipolos.

En OpenCV las líneas epipolares se pueden obtener mediante la función siguiente:

```
void cvComputeCorrespondEpilines(const CvMat* points,
                                 int whichImage,
                                 const CvMat* F,
                                 CvMat* lines)
```

Esta función necesita los puntos, la imagen sobre la que están definidos, la matriz fundamental y, como último parámetro devuelve las líneas epipolares definidas por tres parámetros (coeficientes a, b y c en la ecuación de una recta $ax + by + c = 0$).

Tras introducir estos conceptos ya se puede hablar más en detalle de la rectificación. Mediante la rectificación se busca que las imágenes estén alineadas horizontalmente, de forma que buscar la correspondencia entre píxeles se reduzca a buscar en una sola línea y no en toda la imagen. En OpenCV hay implementados dos métodos para obtener los parámetros necesarios para la rectificación. El primero de ellos es el algoritmo de Hartley (para cámaras sin calibrar) y el segundo implementado es el algoritmo de Bouguet, que utiliza los parámetros de traslación y rotación obtenidos en la calibración. Dado que en este caso (y en general en la robótica) se pueden calibrar las cámaras, el algoritmo a utilizar sería el algoritmo de Bouguet. Este algoritmo intenta minimizar los cambios producidos por la proyección en cada una de las imágenes a la vez que se intenta maximizar el área común que se visualiza, utilizando las matrices de rotación y traslación entre las imágenes estéreo.

Las matrices necesarias para realizar la rectificación se pueden obtener mediante la función:

```
void cvStereoRectify(const CvMat* cameraMatrix1,
                     const CvMat* cameraMatrix2,
                     const CvMat* distCoeffs1,
                     const CvMat* distCoeffs2,
                     CvSize imageSize,
                     const CvMat* R, const CvMat* T,
                     CvMat* R1, CvMat* R2, CvMat* P1,
                     CvMat* P2, CvMat* Q=0,
                     int flags=CV_CALIB_ZERO_DISPARITY,
                     double alpha=-1,
                     CvSize newImageSize=cvSize(0, 0),
                     CvRect* roi1=0, CvRect* roi2=0)
```

Esta función necesita las matrices de las cámaras, los coeficientes de distorsión y las matrices de rotación y traslación obtenidos con "cvStereoCalibrate"; así como el tamaño de la imagen utilizado en las imágenes de calibración. Como resultado se obtienen las matrices R1 y R2 que representan la transformación para lograr la rectificación para la primera y la segunda cámara, respectivamente. P1 y P2 son las matrices de proyección en el nuevo sistema de coordenadas. Q es la matriz

que permite proyectar puntos en dos dimensiones en puntos en tres dimensiones, siguiendo la expresión:

$$Q \begin{bmatrix} x \\ y \\ d \\ 1 \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix} \tag{2}$$

de forma que las coordenadas en 3D serán $(X/W, Y/W, Z/W)$.

Con esta última función obtenemos los parámetros necesarios para la rectificación, pero no la rectificación en si. Para ello se realiza un remapping de la imagen original para obtener la imagen rectificada y sin distorsión. Esto se hace obteniendo, para cada píxel de la imagen destino, su píxel correspondiente en la imagen original. Para ello se puede utilizar la función de OpenCV "cvInitUndistortRectifyMap":

```
void cvInitUndistortRectifyMap(const CvMat* cameraMatrix,
                               const CvMat* distCoeffs,
                               const CvMat* R,
                               const CvMat* newCameraMatrix,
                               CvArr* map1, CvArr* map2)
```

Esta función recibe la matriz de la cámara, los coeficientes de distorsión y las matrices R y P obtenidas por "cvStereoRectify". En "map1" y "map2" se obtienen los mapas que indican como interpolar los píxeles para realizar adecuadamente la rectificación en x y en y. Esta función se ha de ejecutar dos veces, una por cada cámara, obteniéndose dos mapas para cada una de ellas. Para posteriormente aplicar la rectificación se pueden utilizar funciones como cvWarpPerspective o cvRemap. En la imagen 36 se muestran dos imágenes rectificadas, donde los puntos aparecen prácticamente en la misma linea horizontal.

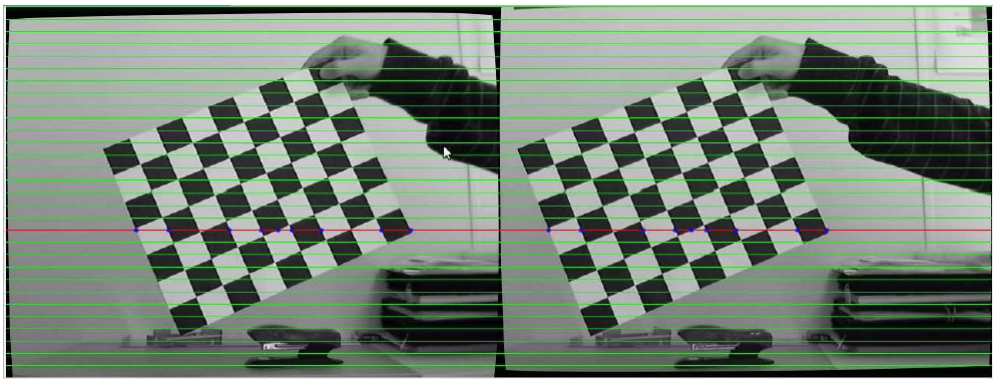

Figura 36: Imágenes rectificadas.

Rectificar las imágenes facilita mucho el proceso de correspondencia estéreo. La correspondencia estéreo consiste en relacionar un punto 3D en las dos imágenes obtenidas con las cámaras, y sólo se puede calcular en las zonas de la imagen que se solapan. OpenCV implementa un algoritmo de correspondencia estéreo basado en asociación de bloques, utilizando ventanas pequeñas de "suma de diferencias absolutas" (sum of absolute difference, SAD) para encontrar la relación entre puntos en las imágenes rectificadas. La suma de diferencias absolutas [48] utiliza una ventana $W$ de un tamaño determinado centrada en un píxel de la imagen izquierda, traslada

la ventana un valor $d$ y compara los valores de intensidad de los píxeles en $W$ en la imagen izquierda y en la ventana trasladada en la imagen derecha, calculando la diferencia entre ellos. Este proceso se muestra en la figura 37. La comparación suele seguir la siguiente expresión:

$$SAD : \sum(Il(x,y), Ir(x+d,y)) = \sum |Il(x,y) - Ir(x+d,y)| \tag{3}$$

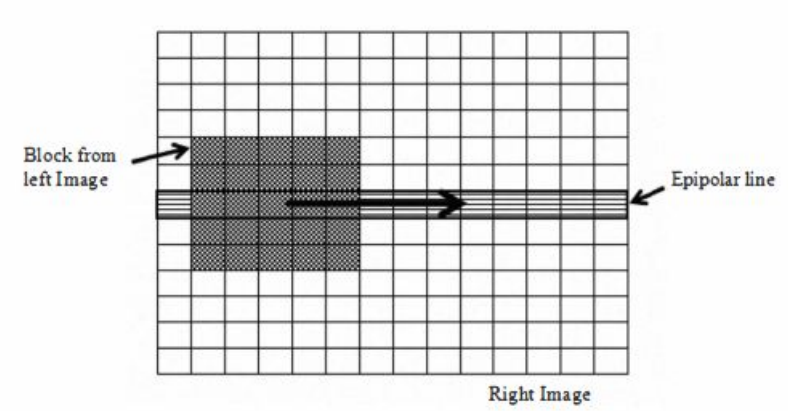


Figura 37: Cálculo de SAD, figura de [48].

La comparación se realiza entre todos los píxeles de la ventana y se suman todos estos valores, por lo que el valor calculado en un píxel mide la diferencia entre dicho píxel y los píxeles circundantes. Esta suma se realiza para todos los valores posibles en esa línea y finalmente se toma como píxel con un mejor valor de asociación aquel con un menor valor de SAD. En el caso ideal, al encontrar la zona correspondiente en la otra imagen, la suma de diferencias absolutas debe dar cero. Una vez realizada la correspondencia se calcula la disparidad entre los píxeles asociados.

En OpenCV, la función que realiza esto es:

```
void cvFindStereoCorrespondenceBM(const CvArr* left,
                                  const CvArr* right,
                                  CvArr* disparity,
                                  CvStereoBMState* state)
```

que recibe como parámetros las imágenes izquierda y derecha, y devuelve la imagen de disparidad resultante en "disparity", del mismo tamaño que las imágenes originales. Concretamente, esta función realiza tres fases:

1. Realizar un prefiltrado para normalizar el brillo de la imagen y aumentar la textura de la imagen.

2. Realizar la búsqueda de correspondencia a lo largo de las líneas epipolares horizontales usando una ventana SAD.

3. Postfiltrado para eliminar malas correspondencias.

El parámetro "state" corresponde con una estructura de datos de tipo "cvStereoBMState", que contiene buffers internos y parámetros como el tamaño de ventana del SAD o el tamaño del filtro de la primera fase.

Si además se necesita hacer la reconstrucción 3D del entorno, a partir de la disparidad, un punto 2D y la matriz Q presentada al inicio de esta sección se puede obtener la profundidad 3D utilizando la ecuación 2; donde las coordenadas en 3D serán $(X/W, Y/W, Z/W)$. OpenCV proporciona dos funciones para realizar esta conversión. La primera de ellas es "cvPerspectiveTransform".

```
void cvPerspectiveTransform( const CvArr *pointsXYD,
                             CvArr* result3DPoints,
                             const CvMat *Q );
```

La función anterior recibe como parámetros los puntos y sus disparidades asociadas y la matriz Q, mientras que en "result3DPoints" devuelve las coordenadas 3D de los puntos.

La segunda función que implementa OpenCV opera con la imagen de disparidad directamente, al contrario que la anterior que trabajaba con puntos.

```
void cvReprojectImageTo3D( CvArr *disparityImage,
                           CvArr *result3DImage,
                           CvArr *Q );
```

La función "cvReprojectImageTo3D" recibe una imagen de disparidad y la matriz Q y devuelve las coordenadas 3D en una imagen del mismo tamaño que la original.

Algunas de estas funciones son las utilizadas por nodos de ROS, por ejemplo para la calibración o hallar la disparidad, mientras que otras, como la última función vista, se utilizan en el software "v4l2stereo".

## 3.9. Efficient Large-Scale Stereo Matching (ELAS)

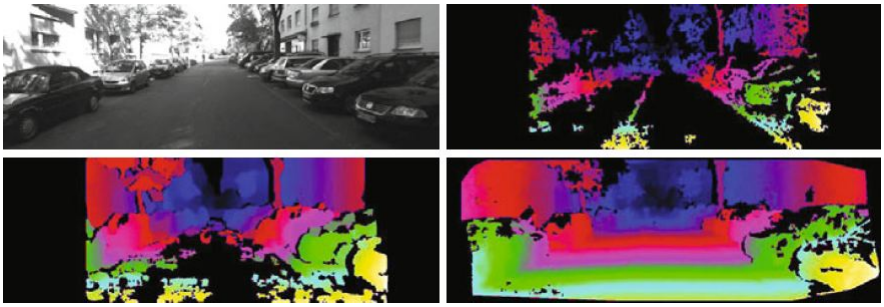

Figura 38: Cálculo de la disparidad con el algoritmo ELAS (abajo derecha) frente a otros algoritmos (figura de [49])

Efficient Large-Scale Stereo Matching (ELAS)[49] es un algoritmo para realizar la asociación estéreo en imágenes de alta resolución. Este algoritmo es el utilizado para el cálculo de la imagen de disparidad en el software "v4l2stereo" que se comentará más adelante. Un ejemplo del resultado de este método frente a otros se muestra en la figura 38. Mientras que en la figura superior derecha no hay una buena tasa de asociación entre píxeles y en la inferior izquierda los bordes no aparecen bien definidos, estos problemas no son tan notables aplicando el algoritmo ELAS.

Suponiendo como entrada imágenes ya rectificadas, el algoritmo basa su funcionamiento en hallar la correspondencia de puntos que son más fáciles de asociar debido a su textura y características. Estos puntos reciben el nombre de puntos de

soporte. Asumiendo disparidades suaves por partes, estos puntos de soporte contienen información para la estimación de los puntos restantes (más ambiguos). Las coordenadas en la imagen de los puntos de soporte se usan para crear una malla 2D vía triangulación de Delaunay.

Los autores proponen un modelo generativo que permite, a partir de los puntos de soporte y una imagen de referencia generar las disparidades. Con este mapa de disparidad y la imagen de referencia, se pueden generar muestras de la imagen opuesta.

### 3.9.1. Puntos de soporte o support points

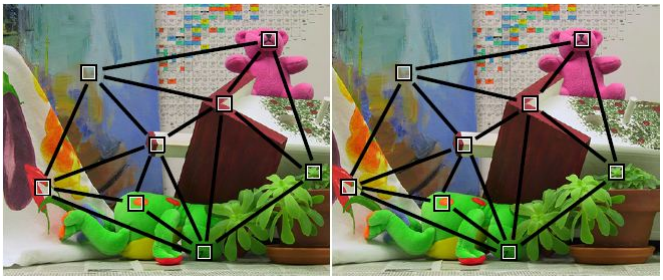

Figura 39: puntos de soporte utilizados en ELAS (figura de [50])

Los puntos de soporte son puntos que se pueden asociar de forma robusta debido a su textura y características, como los que se muestran en la figura 39. Los autores han concluido en sus experimentos que para encontrar este conjunto de puntos un buen método consiste en asociar los puntos de soporte mediante una red regular usando la distancia $l_1$ entre los vectores formados a partir de la concatenación de las respuestas de los filtros de Sobel horizontales y verticales de ventanas de 9x9 píxeles. Los filtros de Sobel calculan el gradiente de la intensidad de la imagen, cuyo resultado puede dar una idea acerca de cómo cambia la imagen (de manera suave o brusca) en ese punto, y por tanto, la posibilidad de que en ese punto haya una arista y su orientación.

En los experimentos los autores utilizan máscaras de Sobel de 3x3 y una red de tamaño fijo de 5 píxeles. Para no imponer restricciones sobre la disparidad, se establece un rango de búsqueda amplio, de la mitad del ancho de la imagen.

Para conseguir robustez se imponen restricciones de consistencia: las correspondencias solo se mantienen si se pueden relacionar de derecha a izquierda o de izquierda a derecha. También se eliminan asociaciones ambiguas, si el ratio entre el mejor matching y el segundo mejor supera un umbral de 0.9. Para cubrir la imagen completa se añaden puntos de soporte en las esquinas, con disparidad calculada a partir de sus vecinos más cercanos.

### 3.9.2. Modelo generativo para correspondencia estéreo.

Este modelo generativo se puede utilizar para dibujar muestras de la otra imagen, a partir de una imagen de referencia y los puntos de soporte.

Sea S=$\{s_1, ..., s_m\}$ el conjunto de puntos de soporte, con cada punto de soporte $s_m = (u_m, v_m, d_m)^T$, con $(u_m, v_m)$ como las coordenadas en la imagen y $d_m$ como su disparidad; y sea O=$\{o_1, ..., o_m\}$ un conjunto de observaciones de la imagen, con

cada observación $o_n = (u_n, v_n, f_n)^T$ como la concatenación de las coordenadas en la imagen y f un vector de características, por ejemplo la intensidad del píxel. Se denota $o_n^{(l)}$ y $o_n^{(r)}$ como las observaciones en la imagen izquierda y derecha, respectivamente. La distribución conjunta de este modelo que proponen los autores se puede factorizar de la siguiente manera:

$$p(d_n, o_n^{(l)}, o_n^{(r)}, S) \propto p(d_n|S, o_n^{(l)}) p(o_n^{(r)}|o_n^{(l)}, d_n) \tag{4}$$

con $p(d_n|S, o_n^{(l)})$ como la probabilidad a priori y $p(o_n^{(r)}|o_n^{(l)}, d_n)$ la probabilidad de la imagen. La probabilidad a priori se determina mediante la ecuación:

$$p(d_n|S, o_n^{(l)}) \propto \begin{cases} \gamma + exp(-\frac{(d_n - \mu(S, o_n^{(l)}))^2}{2\sigma^2}) & if\ |d_n - \mu| < 3\sigma \vee d_n \in N_s \\ 0 & otherwise \end{cases} \tag{5}$$

donde $N_s$ representa el conjunto de puntos de soporte en una zona de 20x20 cercanos al píxel $o_n^{(l)}$ y la condición $d_n \in N_s$ permite tratar mejor discontinuidades en la disparidad en lugares en los que aparecen discontinuidades. $\mu(S, o_n^{(l)})$ se define como una función media que asocia los puntos de soporte y las observaciones. Esta función es lineal por partes, e interpola las disparidades usando triangulación de Delaunay calculado en los puntos de soporte. Para cada triángulo se obtiene un plano definido por la siguiente ecuación:

$$\mu_i(o_n^{(l)}) = a_i u_n + b_i v_n + c_i \tag{6}$$

Tras definir la probabilidad a priori, falta definir la probabilidad de la imagen, que se expresa como una distribución de Laplace con restricciones

$$p(o_n^{(r)}|o_n^{(l)}, d_n) \propto \begin{cases} exp(-\beta \left\| f_n^{(l)} - f_n^{(r)} \right\|_1) & if \begin{pmatrix} u_n^{(l)} \\ v_n^{(l)} \end{pmatrix} = \begin{pmatrix} u_n^{(r)} + d_n \\ v_n^{(r)} \end{pmatrix} \\ 0 & otherwise \end{cases} \tag{7}$$

donde $\beta$ es una constante y $f_n^{(l)}$ es el vector de características. La condición indica que se busca en el mismo píxel de la otra imagen más una disparidad, es decir, se busca en la misma línea epipolar.

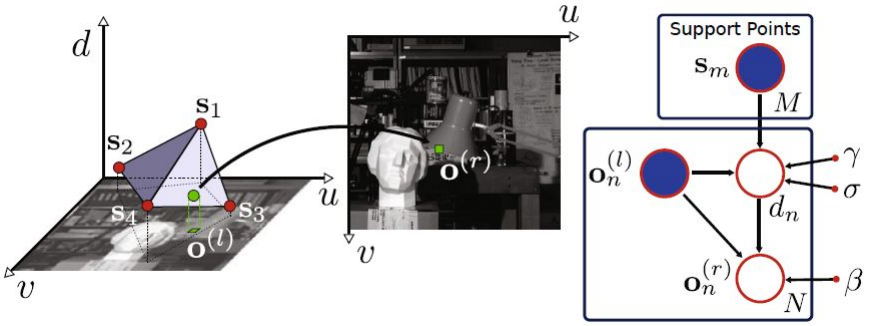


Figura 40: Esquema del modelo generativo (figura de [49])

Mediante el modelo generativo representado por la ecuación 4 a partir de un conjunto de puntos de soporte y una imagen de referencia se pueden generar muestras

de la otra imagen. En la figura 40 se muestra el proceso: a partir de un conjunto de puntos de soporte y observaciones de la imagen de referencia (en este caso, la izquierda), se genera una disparidad $d_n$; y a partir de la disparidad y la imagen de referencia se generan observaciones $o_n^{(r)}$ de la otra imagen.

### 3.9.3. Estimación de la disparidad

Aprovechando el desarrollo anterior se puede estimar la disparidad realizando una estimación "maximum a posteriori" (MAP) para calcular las disparidades. Dicha estimación sigue la expresión:

$$d_n^* = argmax\, p(d_n|o_n^{(l)}, o_1^{(r)}, ..., o_N^{(r)}, S) \tag{8}$$

donde

$$p(d_n|o_n^{(l)}, o_1^{(r)}, ..., o_N^{(r)}, S) \propto p(d_n|S, o_n^{(l)}) p(o_1^{(r)}, ..., o_N^{(r)}|o_n^{(l)}, d_n) \tag{9}$$

donde el primer término se calcula según la ecuación 5 y el segundo término que se puede modelar como

$$p(o_1^{(r)}, ..., o_N^{(r)}|o_n^{(l)}, d_n) \propto \sum_{i=1}^{N} p(o_i^{(r)}|o_n^{(l)}, d_n) \tag{10}$$

Sustituyendo la ecuación 7 en 10 y ésta última y la ecuación 5 en la expresión 9 y tomando el logaritmo negativo se define una función de energía mostrada en la ecuación 11. Minimizando esta ecuación se puede obtener un mapa de disparidad denso. Además esto se puede hacer para cada píxel independientemente, por lo que es fácil de paralelizar.

$$E(d) = \beta \left\| f^{(l)} - f^{(r)}(d) \right\|_1 - log \left[ \gamma + \exp(-\frac{\left[ d - \mu(S, o^{(l)}) \right]^2}{2\sigma^2}) \right] \tag{11}$$

Calcular la imagen de disparidad de la imagen derecha se hace de forma análoga. En el método final se calculan los dos mapas de disparidad y se realiza una comprobación de consistencia entre ambas para eliminar la disparidad en las regiones ocultas y posibles desajustes aislados.

## 3.10. Point Cloud Library (PCL)

Otra de las librerías utilizadas durante la experimentación es Point Cloud Library (PCL) [51]. La librería PCL es un proyecto de código libre bajo licencia BSD para procesamiento de imágenes 2D y 3D y nubes de puntos. Esta librería contiene numerosos métodos como algoritmos de filtrado, reconstrucción de superficies, segmentación o árboles kd, divididos en una serie de módulos que se pueden compilar de forma separada. Los módulos más importantes son:

- Filters: para eliminar ruido y puntos aislados en nubes de puntos 3D. Es el módulo que se ha utilizado durante la experimentación, por lo que se entrará más en detalle más adelante.

- Features: contiene herramientas y estructuras de datos para estimar características 3D a partir de una nube de puntos 3D. Las características 3D son representaciones en un determinado punto 3D o posición en el espacio que

describe patrones geométricos basándose en los datos disponibles al rededor del punto.

- Keypoints: contiene algoritmos para detección de puntos de interés. Los puntos de interés son puntos en una imagen o en una nube de puntos que son estables, característicos e identificables por un criterio de detección bien definido. Por lo general el número de puntos de interés en una nube de puntos es mucho menor que el número total de puntos de la nube, y si se utiliza en combinación con descriptores de características, se puede lograr una representación compacta y descriptiva de los datos originales.

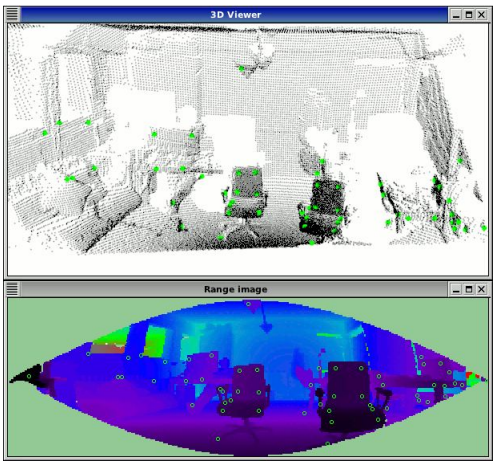


Figura 41: Detección de puntos de interés mediante PCL

- Registration: proporciona herramientas para combinar varios data sets en un modelo global.

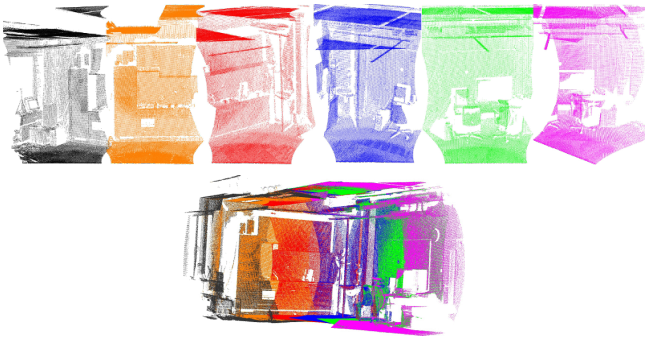

Figura 42: Combinar nubes de puntos en un solo modelo, mediante PCL

- Kd-tree: implementación de la estructura de árbol kd para realizar búsquedas de los vecinos más próximos.
- Octree: proporciona métodos eficientes para la creación de un árbol jerárquico a partir de una nube de puntos. Esto permite particionado espacial, downsampling y operaciones de búsqueda.

- Segmentation: implementación de algoritmos para segmentar la nube de puntos en clusters.

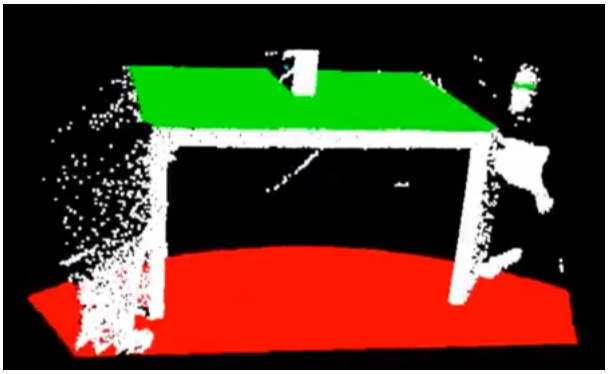

Figura 43: Escena segmentada mediante PCL

- Sample_consensus: módulo que contiene algoritmos como RANSAC (Random Sample Consensus) y modelos como planos y cilindros.
- Surface: implementa funciones de reconstrucción de las superficies originales a partir de escáners 3D.
- Range_image: contiene dos clases para representar y operar con imágenes de profundidad.
- Io: módulo con utilidades para leer y almacenar nubes de puntos a partir de ficheros PCD, así como para capturar nubes de puntos a partir de sensores.
- Visualización: para la visualización de resultados que operan con nubes de datos 3D.

PCL es multiplataforma y se ha compilado en sistemas Linux, MacOS, Windows y Android/iOS.

### 3.10.1. El módulo "filters"

En el desarrollo práctico realizado se han utilizado herramientas del módulo de filtrado, por lo que se van a indicar las principales herramientas que implementa este módulo.

El mecanismo más sencillo es realizar un filtrado mediante un filtro paso banda en una determinada dimensión. La forma básica de crear un filtro es la siguiente:

```
pcl::PassThrough<pcl::PointXYZ> filtro;
filtro.setInputCloud (nP);
filtro.setFilterFieldName ("y");
filtro.setFilterLimits (0.0, 5.0);
filtro.filter (*nPfiltrada);
```

En este fragmento de código se declara un filtro llamado "pass" para trabajar con nubes de puntos pcl::PointXYZ. A continuación se indica cual es la nube de

puntos sobre la que se quiere realizar el filtrado, en este caso "nP". El siguiente paso es especificar la coordenada sobre la que se va a hacer el filtrado (en este caso, coordenada z) y los límites en los que hay que eliminar los puntos. Por último se ejecuta el filtrado, almacenando el resultado en "nPfiltrada". Un ejemplo de esto se ve en la figura 44, donde se ha eliminado parte de la superficie de la mesa.

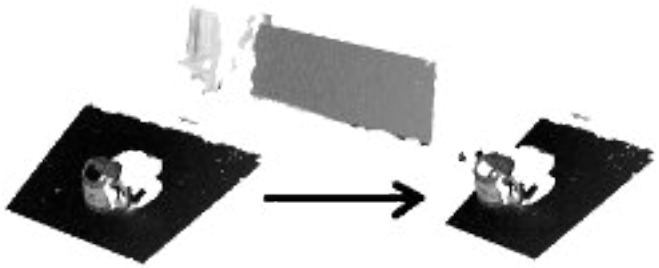

Figura 44: Filtrado de una escena mediante PCL

Otra herramienta interesante es el downsampling. Mediante este método se puede reducir de manera drástica el número total de puntos de la nube. PCL realiza este proceso dividiendo el espacio cubierto por la nube de puntos en cubos llamados "vóxeles" de forma que todos los puntos que quedan dentro de esa región se aproximan como un solo punto, correspondiente con el centroide de los puntos. Aunque podría ser más rápido aproximar las regiones con el centro del vóxel, a la hora de representar una superficie esta aproximación obtiene mejores resultados. El código básico para realizar esto es el siguiente:

```
pcl::VoxelGrid<sensor_msgs::PointCloud2> vox;
vox.setInputCloud (nP);
vox.setLeafSize (0.2f, 0.2f, 0.2f);
vox.filter (*nPfiltrada);
```

De manera análoga a como ocurría en el filtrado, se crea el objeto para el downsampling (con la clase "VoxelGrid") y se establece la nube de puntos de entrada. A continuación se establece el tamaño de los vóxeles con el método "setLeafSize". En este caso el tamaño en todas sus dimensiones es de 20 centímetros. Por último, se realiza el filtrado, obteniendo la nube resultante en "nPfiltrada". Un ejemplo de downsampling es el mostrado en la figura 45.

Debido a la precisión y a los errores en las medidas es posible que aparezcan puntos aislados. Estos puntos pueden complicar la estimación de características como el cambio en la curvatura de una superficie, por lo que conviene eliminarlos. En PCL esto se hace analizando estadísticamente los puntos y su vecindad, eliminando aquellos que no cumplen un determinado criterio. Por ejemplo, para eliminar puntos dispersos primero se calcula la distancia media de un punto a sus vecinos. Después se calcula la media y la desviación típica de las distancias calculadas. Como último paso se eliminan aquellas puntos de la nube que pasen un umbral determinado por $\bar{d}+k*\sigma$, donde k es un multiplicador definido mediante el método "setStddevMulThresh". A continuación se muestra un ejemplo:

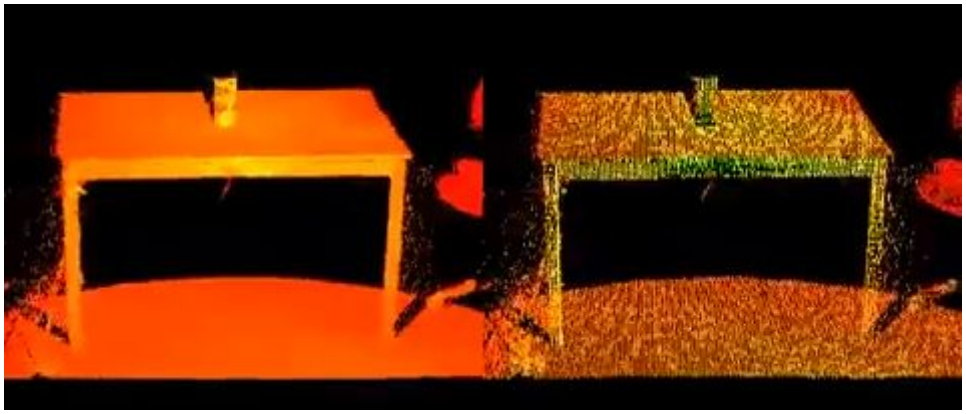

Figura 45: Downsampling de una escena mediante PCL

```
pcl::StatisticalOutlierRemoval<pcl::PointXYZ> filtro;
filtro.setInputCloud (nP);
filtro.setMeanK (10);
filtro.setStddevMulThresh (1.0);
filtro.filter (*nPfiltrada);
```

Primero se define el objeto para realizar el filtrado (en este caso, "filtro") y se indica la nube inicial a filtrar. Mediante el método "setMeanK" se establece el número de vecinos sobre los que hay que calcular la distancia, en este caso para cada punto se calcula la distancia a sus 10 vecinos más cercanos. Por último se establece el multiplicador para la desviación típica y se filtra la nube. En la figura 46 se muestra un ejemplo en el que se han eliminado puntos aislados en los extremos del mueble.

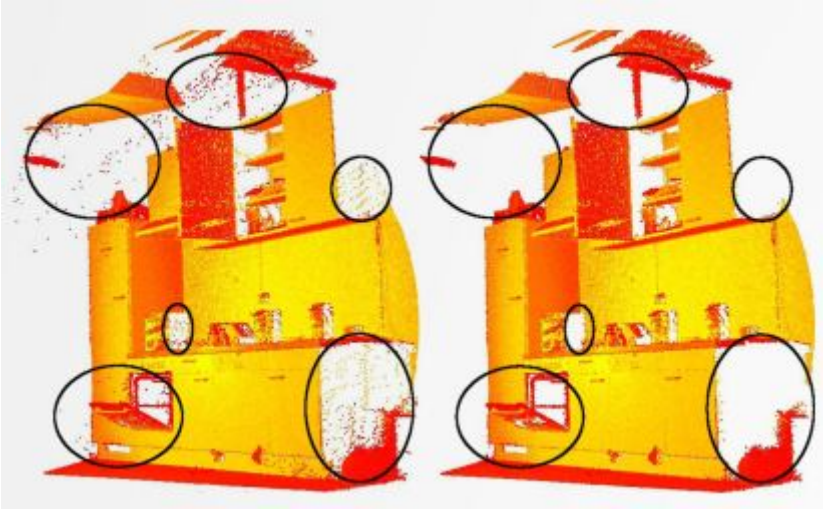

Figura 46: Filtrado de puntos aislados mediante PCL

Esta no es la única forma de la que se puede realizar este filtrado. También se puede aplicar un filtrado radial que elimine los puntos que tengan menos de un número especificado de vecinos en un radio determinado. En el ejemplo a continuación se eliminan aquellos puntos que tengan menos de tres vecinos en un radio de 1 metro.

```
pcl::RadiusOutlierRemoval<pcl::PointXYZ> fradio;
fradio.setInputCloud(nP);
fradio.setRadiusSearch(1.0);
fradio.setMinNeighborsInRadius(3);
fradio.filter(*nPfiltrada);
```

Otra posible opción es mediante un filtro condicional en el que se indica mediante secuencias "or" o "and" las condiciones que deben cumplir los puntos para que no sean eliminados. En el siguiente ejemplo se mantienen en la nube de puntos aquellos puntos que tengan una z mayor que 0 y menor que 0.8

```
ConditionAnd<PointXYZ>::Ptr range_cond (new
                  ConditionAnd<PointXYZ> ());
range_cond->addComparison(
                FieldComparison<PointXYZ>::ConstPtr
                (new FieldComparison<pcl::PointXYZ>
                ("z", ComparisonOps::GT, 0.0)));
range_cond->addComparison(
                FieldComparison<PointXYZ>::ConstPtr
                (new FieldComparison<PointXYZ>
                ("z", ComparisonOps::LT, 0.5)));
ConditionalRemoval<PointXYZ> condrem (range_cond);
condrem.setInputCloud (nP);
condrem.setKeepOrganized(true);
condrem.filter(*nPfiltrada);
```

Por último se pueden filtrar los puntos que corresponden con un modelo obtenido mediante un algoritmo de segmentación. En el siguiente fragmento de ejemplo se eliminan los puntos contenidos en "inliers", que se han obtenido por alguno de los métodos del módulo de segmentación.

```
pcl::ExtractIndices<pcl::PointXYZ> filtro;
filtro.setInputCloud (nP);
filtro.setIndices (inliers);
filtro.setNegative (false);
filtro.filter (*nPfiltrada);
```

En la figura se muestra un ejemplo en el que se han obtenido dos conjuntos por segmentación (correspondientes a dos planos) que se podrían filtrar mediante este método.

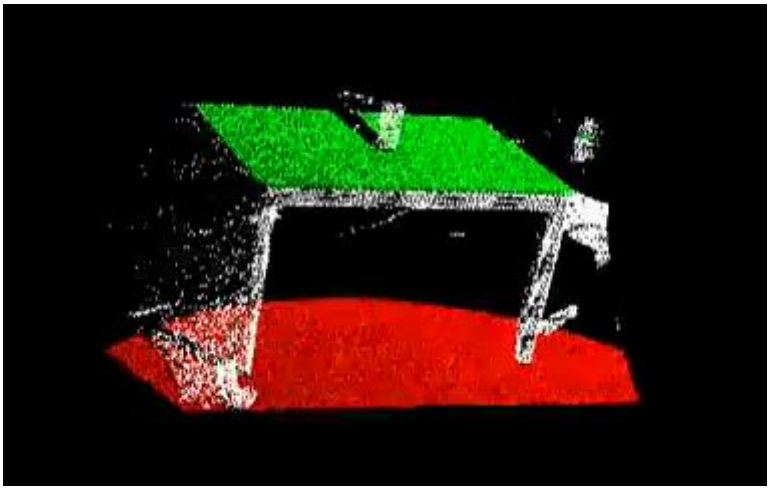

Figura 47: Filtrado mediante índices mediante PCL

# 4. Experimentación

A continuación se van a describir las pruebas realizadas con diferentes algoritmos. Primero se citan las características del equipo informático utilizado, las cámaras Shark Eye y la webcam Minoru utilizada en algunas de las pruebas. A continuación se describe el data set escogido para evaluar los métodos y los nodos implementados en ROS para poder integrar el data set con los nodos de detección. A continuación se evalúa el funcionamiento del detector HOG a partir del nodo del stack "people Experimental". Posteriormente se describe el funcionamiento de la aplicación "v4l2stereo" con la webcam Minoru para detección de objetos verticales. Por último se describe una implementación que combina la detección de objetos verticales que realiza "v4l2stereo" y el detector HOG. Para las implementaciones realizadas se ha utilizado C++ como lenguaje de programación.

## 4.1. Equipo utilizado

El equipo utilizado para la realización de los experimentos y las demos consta de un procesador Intel(R) Core(TM) i5-2400 de 4 núcleos a 3.10GHz, 6MB de memoria caché y 3GB de memoria. El sistema operativo es Ubuntu 10.10 (Maverick), con la versión de kernel Linux 2.6.35-32-generic y Gnome 2.32.0.

## 4.2. Cámaras Shark-Eye

Como sistema de visión se disponía de dos cámaras submarinas "Shark-Eye" como la de la figura 48. Estas cámaras tienen un sensor Sony CCD II de 1/3" de 550 líneas de resolución, sistema de vídeo PAL/NTSC y sensibilidad de 0.05 lux. Las dos cámaras están colocadas en un soporte como el que se muestra en la figura 49 con una separación de 20 centímetros entre ellas.

Figura 48: Cámara Shark-Eye

Junto con las cámaras se disponía de un framegrabber para cada cámara como el mostrado en la figura 50. El framegrabber utilizado es Falcon de "IDS Imaging" [52] con interfaz PCI, una entrada S-VHS y dos entradas CVBS. La resolución máxima es de 768x576 píxeles en PAL y 640x480 en NTSC. Soporta cámaras tanto monócromas como a color, con varios modos de color: escala de grises (8 bits), RGB de 16 bits, RGB True Color de 16 bits o RGB True Color de 32 bits.

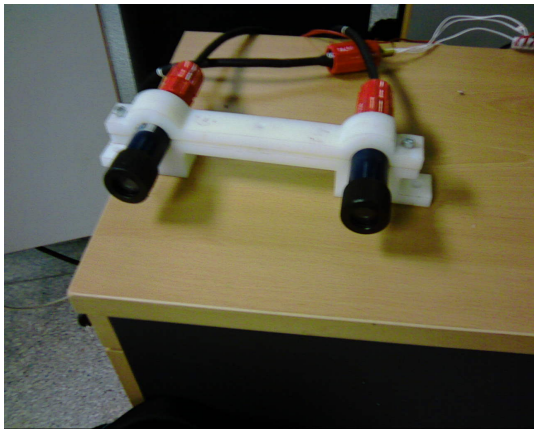

Figura 49: Cámaras Shark-Eye en el soporte para visión estéreo.

Figura 50: Framegrabber FALCON

Para utilizar las cámaras como entrada a otros nodos de ROS se implementó una clase "Cámara" que ofrecía la funcionalidad básica para su integración en ROS, utilizando el API proporcionado por el driver del framegrabber. La clase cámara tiene como miembros parámetros de configuración del framegrabber, como el tamaño de la imagen o la resolución de color, y miembros para la comunicación mediante topics de ROS. A continuación se describen las partes más relevantes del nodo implementado.

### 4.2.1. Interacción con el framegrabber

**Inicialización** La primera acción que hay que realizar para utilizar la cámara es iniciar el framegrabber. Para ello se utiliza la función "is_InitBoard(&id,NULL)", que necesita como parámetro el identificador del framegrabber. Este identificador se puede determinar mediante una aplicación proporcionada por el fabricante.

Para establecer el color, el tamaño, la posición y el escalado de la imagen se utilizan las siguientes funciones:

```
is_SetColorMode(framegrabber, IS_SET_CM_RGB24);
is_SetImageSize(framegrabber, width, height);
is_SetImagePos(framegrabber,0,0);
is_SetScaler(framegrabber, IS_SET_SCALER_OFF,
             IS_SET_SCALER_OFF);
```

A continuación hay que asignar memoria para las imágenes. Esto se lleva a cabo con las funciones:

```
is_AllocImageMem(framegrabber, width, height, bpp,
                 ppcImgMem+i, id_mem+i);
is_SetImageMem(framegrabber,ppcImgMem[i],id_mem[i]);
```

La primera función reserva la memoria necesaria, y la segunda establece la memoria asignada como activa. Si la memoria no se marca como activa no se puede utilizar. Hay que asignar tantas regiones de memoria como buffers se vayan a asignar. Los buffers de memoria se tienen que añadir mediante la función siguiente:

```
is_AddToSequence(framegrabber,ppcImgMem[i],id_mem[i]);
```

A continuación se elige el modo de adquisición y el modo de vídeo. El framegrabber utilizado permite modos entrelazados y no entrelazados, así como un modo estéreo en el caso de trabajar con dos cámaras sincronizadas. La función que proporciona el API para realizar esto es:

```
is_SetCaptureMode(framegrabber, MODE);
is_SetVideoMode(framegrabber,IS_SET_VM_PAL);
```

**Finalización** Al terminar el programa no hay que olvidar liberar la memoria que estaba utilizando el framegrabber. Para ello se ha incluido en el destructor de la clase las funciones del API que realizan esta función.

```
is_ClearSequence(framegrabber);
is_FreeImageMem(framegrabber,ppcImgMem[i], id_mem[i]);
is_ExitBoard(framegrabber);
```

La función "is_ClearSequence" elimina todos los buffers de memoria que se habían establecido con "is_AddToSequence", mientras que "is_FreeImageMem" libera la memoria reservada con "AllocImageMem". La función "is_ExitBoard" realiza la función contraria a "is_InitBoard", liberando el manejador del framegrabber y liberando la memoria y estructuras de datos en uso.

**Captura y manejo de memoria** La función "is_CaptureVideo" digitaliza las imágenes de vídeo, copiándolas en memoria. Para acceder a estas imágenes es recomendable bloquear la posición de memoria que se está leyendo para evitar que se sobreescriba durante el proceso de lectura. Para bloquear y desbloquear se utilizan, respectivamente, las siguientes funciones:

```
is_LockSeqBuf(framegrabber, this->id_mem[id-1],
              this->ppcImgMem[id-1]);
is_UnlockSeqBuf(framegrabber, this->id_mem[id-1],
              this->ppcImgMem[id-1]);
```

Para conocer la posición de memoria sobre la que se está escribiendo se utiliza la función "is_GetActSeqBuf". Esta función nos informa de la posición actual, por lo que hay que determinar cuál ha sido la última posición de memoria sobre la que se ha escrito. Dicho cálculo se indica a continuación, y depende del número de buffers en total que se está utilizando.

```
ret = is_GetActSeqBuf(framegrabber, &current,
      &pmemcurrent, &pmemlast);
int previous = ((current + NB_MEMORY_BUFFERS - 2)
              % NB_MEMORY_BUFFERS)+1;
```

**Cambio de parámetros** Otros parámetros a modificar del framegrabber son el brillo, el contraste y la saturación, tanto en el canal U como en el V. Para esto se utilizan las siguientes funciones:

```
is_SetBrightness(this->framegrabber,brightness);
is_SetContrast(this->framegrabber,contrast);
is_SetSaturation(this->framegrabber,satU,satV);
```

### 4.2.2. Integración en ROS

Para su utilización en ROS se ha implementado un nodo que se encarga de instanciar cada una de las dos cámaras (y su framegrabber correspondiente) y, en cada iteración, leer la última imagen almacenada y enviarla por el topic correspondiente.

En total se publica en cuatro topics, ya que cada cámara publica en dos topics, uno para las imágenes y otro para la información de calibración. También se ofrecen dos servicios, uno por cada cámara. Concretamente, la cámara izquierda realiza las siguientes funciones de ROS:

- publica imágenes en el topic /stereo/left/image_raw (mensaje de tipo sensor_msgs::Image)
- publica información de calibración en el topic /stereo/left/camera_info (mensaje de tipo sensor_msgs::CameraInfo)
- ofrece el servicio /stereo/left/set_camera_info, para almacenar los parámetros de calibración (petición y respuesta de tipo sensor_msgs::SetCameraInfo)

La cámara derecha publica en los mismos topics y servicios, solo que cambiando la palabra "left" por "right".

Para registrarse como publicador de imágenes se debe crear una instancia de la clase "ImageTransport", mientras que para publicar la información de calibración se hace mediante la metodología estándar de ROS.

```
ros::NodeHandle nh;
imageTransport = new image_transport::ImageTransport(nh_);
string ad = "/stereo/" + nombre + "/image_raw";
imagePub = imageTransport->advertise(ad,1);

string ad2 = "/stereo/" + nombre + "/camera_info";
cInfoPub = nh.advertise<sensor_msgs::CameraInfo>(ad2, 1);
```

Para la calibración se ha utilizado el nodo de ROS "cameracalibrator.py" del paquete "camera_calibration". Este nodo utiliza para realizar la calibración funciones de OpenCV (descritas en la sección 3.8.1) y permite tanto calibración monocular como estéreo, de forma que va tomando imágenes si encuentra las esquinas del patrón de calibración y cuando tiene suficientes datos para calibrar calcula los parámetros. Una vez calculados, la herramienta permite comprobar el error que se ha cometido. De esta manera, volviendo a pasar el tablero de calibración, detecta las esquinas y muestra el error epipolar. Si se está conforme con los resultados obtenidos, haciendo clic en el botón "commit" se llama al servicio del nodo de las cámaras para almacenar estos nuevos valores.

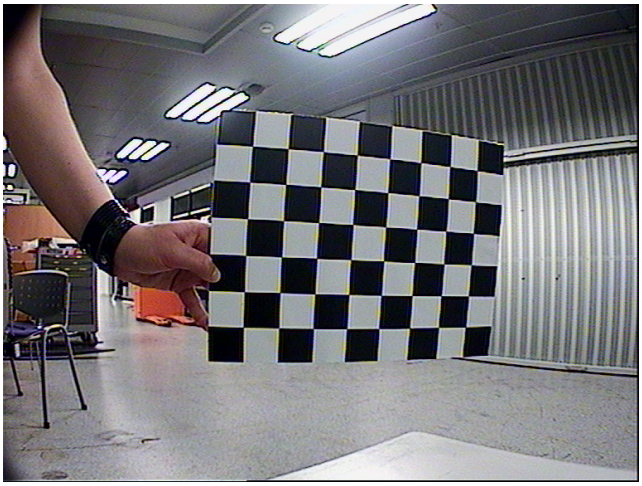

Figura 51: Calibración de una de las cámaras Shark Eye

Este nodo de calibración necesita que el nodo responsable de las cámaras oferte un servicio que permita modificar los parámetros de calibración. Para ofertar este servicio se debe realizar lo siguiente:

```
string s1 = "stereo/"+nombre+"/set_camera_info";
setCInfo = nh.advertiseService(s1,
                           &Camara::set_camera_info, this);
```

También hay que implementar la función que realiza este servicio. En este caso es un método de la función cámara llamada "set_camera_info".

```
bool Camara::set_camera_info(
  sensor_msgs::SetCameraInfo::Request &req,
  sensor_msgs::SetCameraInfo::Response &res)
{
   msgCInfo = req.camera_info;
   res.status_message = "";
   res.success = true;
   string nomFich = nombre+".yaml";
   return save_camera_info(nomFich, msgCInfo);
}

bool Camara::save_camera_info(string filename,
  sensor_msgs::CameraInfo &info)
{
     return camera_calibration_parsers::writeCalibration(
                              filename,this->nombre,info);
}
```

Cuando se produce una petición al servicio, se recibe en "req" los parámetros de calibración con los que se puede rellenar el mensaje de tipo "sensor_msgs::CameraInfo". En la respuesta "res" al servicio sencillamente se indica que todo ha ido correctamente. Al mismo tiempo se almacena en un fichero los datos de calibración mediante la función "writeCalibration" de la clase de ROS "camera_calibration_parsers", del paquete del mismo nombre. De esta manera si se reinicia el nodo

se pueden leer los parámetros de calibración de este fichero, mediante el método "loadCalibration" de "camera_calibration_parsers", y no es necesario realizar de nuevo la calibración.

Para cambiar dinámicamente los parámetros de las cámaras se ha utilizado el paquete "dynamic_reconfigure" de ROS. Mediante este paquete se pueden cambiar dinámicamente los parámetros que especifiquemos en un fichero de configuración. En este caso, se ha elegido poder modificar el brillo, el contraste y la saturación en U y en V. El primer paso a realizar es crear un fichero de configuración (de extensión .cfg) estableciendo los parámetros que se pueden cambiar. Este fichero tiene la siguiente estructura:

```
#!/usr/bin/env python
PACKAGE = "visualizacion"
import roslib;
roslib.load_manifest(PACKAGE)
from dynamic_reconfigure.parameter_generator import *
gen = ParameterGenerator()
# left parameters
gen.add("BrightnessLeft", int_t, 0,
        "brightness between 0 and 255", 128, 0 ,255)
gen.add("ContrastLeft", int_t, 0,
        "brightness between 0 and 200", 216, 0 ,511)
gen.add("SaturationULeft", int_t, 0,
        "Saturation U between 0 and 511", 254, 0 ,511)
gen.add("SaturationVLeft", int_t, 0,
        "Saturation V between 0 and 511", 180, 0 ,511)
#right parameters... de manera análoga a los izquierdos
...
exit(gen.generate(PACKAGE, "camara", "camara"))
```

Para cada parámetro se indica su nombre, el tipo de datos que se maneja, el nivel, una breve descripción de ayuda, el valor por defecto, y los valores mínimo y máximo.

Tras editar el fichero de configuración en el nodo se crea el servidor del servicio de "dynamic_reconfigure":

```
/*código en el nodo ROS*/
dynamic_reconfigure::Server<visualizacion::camaraConfig>
                            server;
dynamic_reconfigure::Server<visualizacion::camaraConfig>
                            ::CallbackType f;
f = boost::bind(&cbReconfigure, _1, _2);
server.setCallback(f);

/*callback*/
void cbReconfigure(visualizacion::camaraConfig &config,
                           uint32_t level) {
  c_left->updateParams(config.BrightnessLeft,
                   config.ContrastLeft,
                   config.SaturationULeft,
                   config.SaturationVLeft);
  c_right->updateParams(config.BrightnessRight,
                        config.ContrastRight,
                        config.SaturationURight,
                        config.SaturationVRight);
}
```

Con estas acciones se indica la función que se ejecutará cuando se varíe algún parámetro mediante "dynamic_reconfigure". Los nuevos parámetros llegan a la función como miembros de la clase "config" y tendrán los nombres indicados en el fichero de configuración. Con estos valores se llama a una función de la clase cámara llamada "updateParams", que cambia los parámetros con la ayuda de las funciones del driver del framegrabber (descritas en la sección anterior).

Para cambiar los parámetros hay que ejecutar un nodo del paquete "dynamic_reconfigure". Por ejemplo, ejecutando la orden "rosrun dynamic_reconfigure reconfigure_gui" aparecerá una pequeña interfaz gráfica con los parámetros que hemos especificado en el fichero de configuración. En el caso de las cámaras aparece una ventana como la de la figura 52, en la que se pueden ajustar los parámetros indicados.

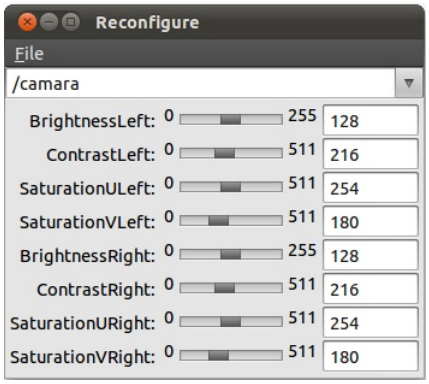


Figura 52: Configuración dinámica de parámetros

### 4.2.3. Problemas encontrados

Durante el desarrollo del nodo descrito se encontraron una serie de problemas que finalmente provocaron que se descartara la utilización de estas cámaras.

Los primeros problemas encontrados fueron durante la instalación del driver del framegrabber. Dicho driver no está muy desarrollado para Linux debido a la baja demanda. Por ejemplo, no funcionaba correctamente con kernels de linux de tipo "-pae". Este tipo de kernel se instala si en nuestro equipo se detectan más de 3GB de memoria, además de mejorar el rendimiento de CPUs de 64-bits. Esto hizo instalar la versión "-generic" del kernel que se ha indicado en la sección 4.1, limitando la memoria del equipo a 3GB. Una vez solucionado este problema y tras realizar la instalación las cámaras funcionaban correctamente. No obstante, al reiniciar el equipo dejaban de funcionar, problema que al final también se pudo solucionar mediante un script.

Sin embargo, el mayor problema encontrado fue la no sincronización de las cámaras. Tras realizar la calibración se probó a realizar la disparidad mediante el nodo de ROS "stereo_image_proc", del paquete del mismo nombre dentro del stack "image_pipeline", que realiza la rectificación, la disparidad y la nube de puntos recibiendo como entradas las imágenes "raw" y los parámetros de calibración. Los resultados obtenidos no fueron muy buenos, debido a la falta de sincronización. Este hecho finalmente nos llevó a probar otras opciones, como es el caso de la webcam Minoru que se describirá a continuación.

## 4.3. Webcam Minoru

Minoru [53] es una webcam estéreo USB compuesta por dos cámaras (sensores CMOS VGA 640x480) separadas 6 centímetros entre ellas y un micrófono. Actualmente soporta hasta 30 frames por segundo con una resolución de 320x240 o de 15 frames por segundo con una resolución de 640x480. Una de sus principales ventajas es su bajo precio y la conexión USB que facilita su instalación, mientras que como desventaja se puede mencionar la no sincronización de la adquisición de las imágenes. Pese a la falta de sincronización, al contrario que en el caso de las cámaras Shark Eye, los resultados de disparidad eran bastante buenos. No obstante, si existiera esta sincronización, posiblemente se obtendrían mejores resultados.

Figura 53: Webcam Minoru

Mediante esta webcam es posible realizar vídeos y fotografías que se pueden ver en tres dimensiones con la ayuda de gafas 3D, si bien el software para ello sólo es compatible con el sistema operativo Windows. Aunque esta función no está

disponible en Linux, existe un proyecto de código abierto y bajo licencia GNU que permite realizar disparidad y sustracción de fondo, entre otros tipos de procesado útiles para la robótica. Esta aplicación se llama "v4l2stereo" y se comentará en detalle en la sección 4.6. Una de las desventajas de esta aplicación es que requiere un alto coste de CPU, por lo que si se necesitara utilizar en un sistema con recursos limitados habría que adaptarlo adecuadamente.

### 4.3.1. Posición de la cámara y campo de visión

Dado que el desarrollo está situado dentro del proyecto SAFEBUS de seguridad de autobuses, se ha intentado simular las condiciones en las que se situaría la cámara en la aplicación final. Para ello, la cámara se ha situado en un vehículo de tipo carretilla industrial a una altura de 2.16 metros, como se muestra en la figura 54. El ángulo de inclinación es de 60º aproximadamente, orientada hacia el suelo.

Figura 54: Posición de la webcam sobre la carretilla

El campo de visión de la webcam es de unos 40º, de acuerdo a lo indicado en la especificación. Desde esta posición, el área de visión se podría representar por un rectángulo de 147 cm de largo por 175 cm de ancho, estando la longitud hasta el punto medio a unos 51cm.

## 4.4. Daimler Stereo Pedestrian Detection Benchmark

Para realizar pruebas de algunos de los métodos encontrados se ha utilizado el data set "Daimler Stereo Pedestrian Detection Benchmark". Este data set se puede encontrar en [55] y se presenta en [56]. Dicho data set extiende el presentado en [1] para visión monocular.

Las imágenes se tomaron durante 27 minutos de conducción por un entorno urbano. El conjunto de entrenamiento consta de un total de 15,660 muestras positivas de peatones. A partir de imágenes de video se obtuvieron 3915 imágenes de peatones con una resolución de 48x96, de cada una de las cuales se obtuvieron cuatro

imágenes tras variar de forma aleatoria la región que contenía dicha muestra unos píxeles en vertical y horizontal. Estas muestras no cumplen con ninguna restricción en cuanto a pose o iluminación, sin embargo en todas ellas se ven los peatones completamente en posición vertical. En la figura 55 se muestran algunas imágenes de entrenamiento. Además, se incluyen 7129 imágenes que no contienen peatones como muestras negativas para el entrenamiento.

Figura 55: Ejemplo de imágenes de entrenamiento del data set, figura de [1].

En cuanto al conjunto de test, éste está formado por 21790 imágenes de 640x480 píxeles que contienen un total de 56492 peatones etiquetados de forma manual y 259 trayectorias de peatones, algunos ejemplos se pueden ver en la figura 56. Las imágenes se encuentran organizadas en dos directorios diferentes, de manera que en el directorio "c0" se encuentran las imágenes correspondientes a la cámara izquierda y en el "c1" las imágenes correspondientes a la cámara derecha. Los nombres de los archivos corresponden con la marca de tiempo en la que fueron tomadas, por ejemplo la imagen "01m_41s_297510u.pgm" se tomó en el instante de tiempo 1 minuto, 41 segundos y 297510 microsegundos.

Figura 56: Ejemplos de imágenes de test del data set.

El data set proporciona además dos documentos con el ground truth, uno para el visión monocular y otro para visión estéreo. En ellos, se hace una distinción entre peatones, grupos de peatones, peatones parcialmente visibles, ciclistas, y motoristas. Para el ground truth 2D se proporciona la localización del rectángulo que contiene al peatón, dando las coordenadas de dos de sus vértices opuestos. En el ground truth para visión estéreo se proporcionan las coordenadas 3D en las que se encuentra dicho peatón, calculadas manualmente a partir del centro de gravedad y los parámetros del sistema estéreo. Por último, el data set contiene los datos de calibración de las cámaras con las que se tomaron las imágenes y datos del vehículo para cada frame (velocidad y orientación).

### 4.4.1. Reproducción del data set

Para la reproducción de las imágenes del data set se ha implementado un nodo ROS. Este nodo se encarga de leer las imágenes de ambas cámaras y enviarlas por dos topics: "/stereo/left/image_raw" y "/stereo/right/image_raw". Entre envíos se realiza una espera en función del tiempo indicado en los nombres de las imágenes, para representar las condiciones reales de conducción. A continuación se detalla la implementación.

Tras iniciar el nodo ROS, se debe registrar como publicador en los topics correspondientes. Para ello se deben crear instancias de la clase "ImageTransport" y "Publisher", del paquete "image_transport". Este paquete proporciona herramientas para el transporte de imágenes en ROS, incluyendo compresión y varios formatos; y se debe usar para el envío de imágenes en lugar del procedimiento estándar para el paso de mensajes en ROS. A continuación se muestra el código para ello.

```
ros::NodeHandle nh;
image_transport::ImageTransport* imageTransportLeft;
image_transport::ImageTransport* imageTransportRight;
image_transport::Publisher imagePubLeft;
image_transport::Publisher imagePubRight;

imageTransportLeft = new ImageTransport(nh);
imageTransportRight = new ImageTransport(nh);
string adLeft = "/stereo/left/";
string adRight = "/stereo/right/";
imagePubLeft = imageTransportLeft->advertise(adLeft +
    "image_raw",1);
imagePubRight = imageTransportRight->advertise(adRight +
    "image_raw",1);
```

Para obtener una lista de todas las imágenes se utiliza la función POSIX "scandir", indicando como cuarto parámetro en la llamada "alfasort" para que ordene los ficheros adecuadamente. De esta manera se obtiene una lista con los descriptores de todas las entradas que contiene el directorio. Hay que señalar que también incluye las entradas "." y "..", referentes al directorio actual y al directorio padre, por lo que las imágenes comienzan en la segunda entrada. A continuación se muestra el código necesario correspondiente al directorio de las imágenes de la cámara izquierda, para el otro directorio se haría de forma análoga:

```
char nDirLeft[] = "/host/Dataset/TestData/c0/";
int total_img_left = scandir(nDirLeft, &f_list_left,
    NULL, alphasort);
```

A continuación se realiza un bucle hasta que se han leído todas las imágenes. En cada iteración se leen las imágenes mediante funciones de OpenCV y se crea un mensaje ROS para cada una de ellas mediante la clase "CvBridge", encargada de hacer la conversión entre mensajes ROS y imágenes OpenCV (IplImage). La marca de tiempo se obtiene del nombre del fichero y se especifica en la cabecera del mensaje ROS. El código para realizar esto para la imagen izquierda es el siguiente:

```
sensor_msgs::ImagePtr msgImageLeft;
strcpy(nImgLeft, nDirLeft);
strcat(nImgLeft,f_list_left[numImg]->d_name);
img_left = cvLoadImage(nImgLeft, CV_LOAD_IMAGE_COLOR);
/*message*/
msgImageLeft = sensor_msgs::CvBridge::cvToImgMsg(img_left);
msgImageLeft->header.stamp.sec = sec;
msgImageLeft->header.stamp.nsec = nsec;
```

Por último se publica el mensaje por el topic correspondiente, se espera la diferencia de tiempo correspondiente y se continúa con la siguiente imagen. La espera se realiza mediante la función de OpenCV "cvWaitKey", que se utiliza para la visualización en ventanas de las imágenes y también permite espera hasta que se ha pulsado una determinada tecla.

```
imagePubLeft.publish(msgImageLeft);
int ret = cvWaitKey(dTiempo) & 255;
```

### 4.4.2. Comparación con el Ground Truth

El data set proporciona dos ficheros con el ground truth, uno para ser utilizado en un sistema monocular y un segundo para un sistema estéreo. En cada uno de ellos se indica para cada imagen las personas, ciclistas, motoristas, peatones parcialmente visibles y grupos de peatones que se pueden encontrar. Para poder comprobar los resultados obtenidos de la clasificación, se ha implementado un nodo ROS que se suscribe a un topic por el que se reciben mensajes con las posiciones de las personas detectadas y compara con el ground truth, llevando una cuenta de detecciones, falsos positivos y falsos negativos.

Para cada imagen el formato para indicar los objetos en el fichero de ground truth es el siguiente:

```
nombre_imagen
ancho_imagen alto_imagen
0 num_objetos
# clase_obj
id_obj id_unico
confidence
min_x min_y max_x max_y
0
...
```

donde "num_objetos" es el número de objetos que hay en ese frame, "clase_obj" puede tomar como valor 0 (persona visible por completo), 1 (ciclista), 2 (motorista), 10 (grupo de peatones) o 255 (peatón, ciclista o motorista parcialmente visible); "id_obj" es un identificador para el objeto; "id_unico" es un identificador para esa ocurrencia en concreto; "min_x", "min_y", "max_x" y "max_y" delimitan el rectángulo delimitador en el que se encuentra el objeto y "confidence" es un valor para indicar si es un objeto requerido (1) u opcional (0) a detectar. Un objeto se considera que es "requerido" si es un peatón completamente visible de al menos 72 píxeles de altura. Para cada imagen puede haber varios objetos.

Se ha creado una clase llamada "Objeto" que representa a cada uno de los objetos de una imagen. Además, se ha creado la clase "Frame" para representar una imagen,

conteniendo una referencia a los objetos que aparecen en ella. Los frames se agrupan en un conjunto ordenado, para que su búsqueda sea eficiente. Los frames que no tienen objetos, no se insertan.

El nodo se suscribe al topic correspondiente, donde esperará la llegada de mensajes de tipo "PolygonStamped", del paquete de ROS "geometry_msgs". Este mensaje contiene una cabecera y una estructura para representar un polígono con un número de puntos definidos al crear el mensaje. En este caso, se enviarán dos puntos por cada detección, correspondientes a la esquina superior izquierda y a la inferior derecha del rectángulo que contiene a la persona detectada. La cabecera contiene un campo para indicar la marca de tiempo, que se utilizará como identificador de la imagen a la que corresponde la detección. El nodo que realiza la detección deberá enviar un mensaje de este tipo por cada imagen. Si en una imagen no ha detectado ningún objeto se envía un mensaje solo con la cabecera, sin indicar ningún polígono, para facilitar la recogida de estadísticas.

Este nodo comparará los rectángulos de detecciones recibidas con las del ground truth, ahora bien, esta comparación no es trivial. La región señalada en el ground truth no tiene por qué ser del mismo tamaño que la que se ha detectado por el sistema, o puede estar desplazada algunos píxeles en alguna dirección. Por esta razón, siguiendo el criterio indicado por los autores del data set en [1], se ha establecido la función de comparación entre los rectángulos como la razón entre el área del rectángulo intersección y el área del rectángulo unión:

$$\Gamma(a_i, a_j) = \frac{A(a_i \cap a_j)}{A(a_i \cup a_j)} \tag{12}$$

Las operaciones de unión e intersección están implementadas en OpenCV. Si esta razón es mayor que un umbral definido se considera que el rectángulo de la detección y el rectángulo del objeto ground truth corresponden con el mismo objeto. Este umbral se ha establecido en 0.25, siguiendo de nuevo las indicaciones de los autores del data set.

Con las consideraciones explicadas anteriormente se comparan los resultados recibidos con el ground truth. Primero se busca la marca de tiempo el conjunto donde se han almacenado los objetos del ground truth. Si no está, es porque no hay objetos en esa imagen, por lo que si hay detecciones habrá que contabilizarlas como falsos positivos. Si la marca de tiempo sí que está, para cada detección se compara con los objetos de dicha imagen. Si se cumple el umbral para alguno de ellos significa que el objeto se ha detectado correctamente. En cambio, si no se supera el umbral para ningún objeto de la imagen se trata de un falso positivo. Los falsos negativos se calculan a partir del total de detecciones y las detecciones correctas al finalizar el nodo. El proceso completo se muestra en la figura 57 mediante un diagrama de flujo.

Al finalizar el programa se muestran los datos contabilizados, distinguiendo entre tipos de objetos y valor de confianza uno o cero. También se muestra el total de imágenes procesadas.

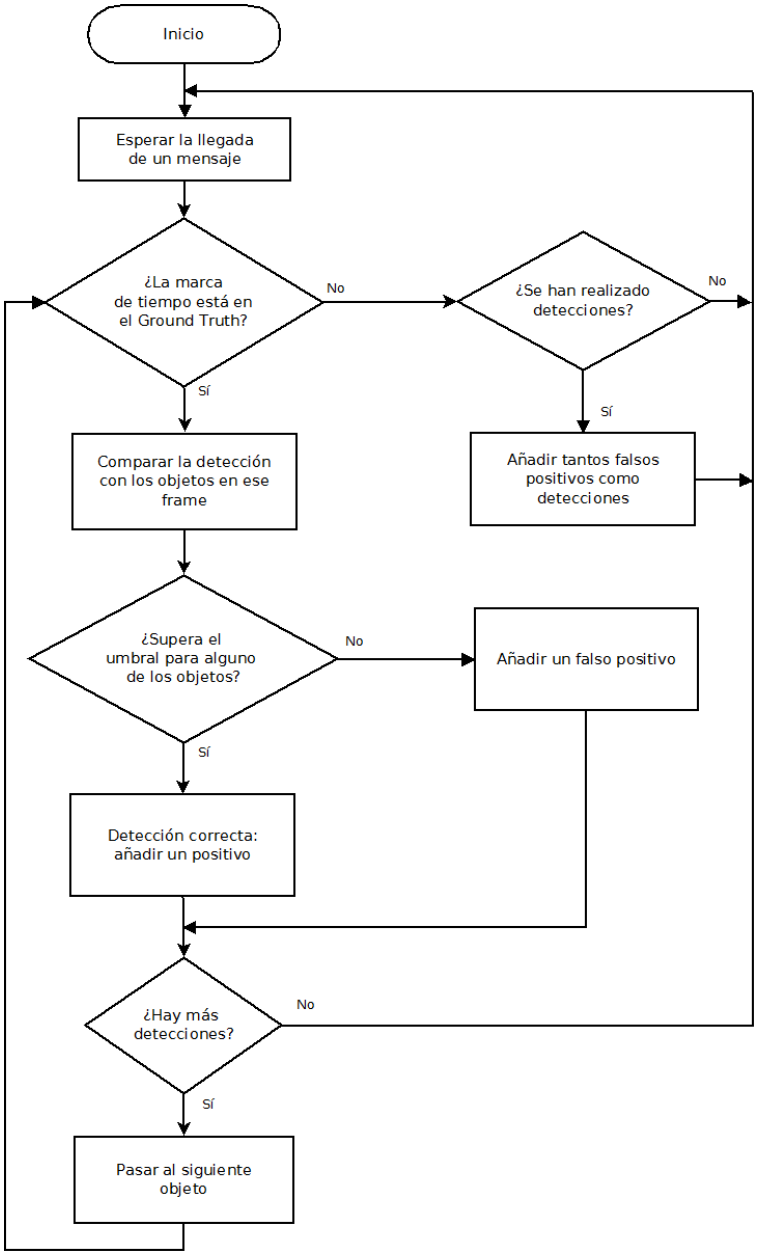


Figura 57: Diagrama de flujo del nodo de comparación de resultados

## 4.5. HOG

A continuación se ha evaluado el funcionamiento del detector HOG de OpenCV para el data set considerado. Para ello se ha utilizado el nodo de ROS "pedestrianDetectorHOG" del paquete del mismo nombre del stack "peopleExperimental" comentado en la sección 3.2.2.

Al nodo ya existente se le ha añadido el código necesario para enviar los resultados al nodo de comparación del data set. Para ello se ha registrado como publicador en el topic "people_rectangle" y se crea un mensaje de tipo "geometry_msgs::PolygonStamped" por cada imagen, indicando la marca de tiempo y los vértices superior izquierda e inferior derecha de las detecciones. Si no se ha detectado ningún objeto

se envía el mensaje solo con la marca de tiempo.


a b

c d

Figura 58: Ejemplos de ejecución del detector HOG

En la figura 58 se ven algunos ejemplos del funcionamiento del detector. En la figura 58(a) se muestra un peatón correctamente detectado, mientras que en (b) se ve, además de una detección correcta, un falso positivo: el sistema está detectando un objeto (un árbol) como persona. En (c) se ve otro falso positivo. Por último, en (d) se da un falso negativo: el sistema no detecta ninguna persona, pero al fondo de la imagen hay tres peatones. En este caso puede que no se detecten debido a que la distancia a ellos aun es grande, por lo que las personas aparecen muy pequeñas, y si el clasificador no se ha entrenado para personas de este tamaño será difícil que las detecte.

### 4.5.1. Resultados (ejecución en tiempo real)

Para evaluar el funcionamiento de este método se han realizado un total de siete pruebas de este nodo, variando parámetros del detector. Como ya se indicó en la sección 3.2.2, hay cuatro parámetros principales a modificar: umbral para la distancia entre las características y el plano del clasificador SVM, desplazamiento de la ventana, coeficiente para el incremento del tamaño de la ventana de detección y coeficiente para agrupar detecciones. Este último parámetro no se ha modificado para que no haya un mismo objeto detectado varias veces. En cuanto al resto de parámetros, el umbral por defecto está establecido a cero y se ha probado con umbrales

de 0.1, 0.3 y 0.01; el coeficiente de incremento de la ventana por defecto es 1.05 y se ha establecido a 1 (sin incremento); y el paso de la ventana por defecto está a 8x8 píxeles y se ha probado con 16x16 y 4x4. Para evaluar el método se ha utilizado el data set comentado en la sección anterior, considerado solo los objetos que en el ground truth estaban marcados con un valor de confianza de 1. Primero se ha evaluado su funcionamiento en tiempo real, utilizando el nodo para la reproducción del dataset explicado en la sección 4.4.1, es decir, enviando imágenes en función del tiempo entre frames especificado por el dataset.

| | detecciones | Falsos positivos/frame | t. medio de ejecución/frame |
|---|---|---|---|
| por defecto | 110/172 (63.9535 %) | 0.25 | 442 ms |
| sin incremento de ventana | 20/766 (2.61097 %) | 0.002 | 49 ms |
| paso de ventana 16 píxeles | 72/281 (25.6228 %) | 0.034 | 270 ms |
| paso de ventana 4 píxeles | 39/52 (75 %) | 0.766 | 1.518 s |
| coeficiente SVM 0.1 | 112/173 (64.7399 %) | 0.17 | 448 ms |
| coeficiente SVM 0.01 | 109/171 (63.7427 %) | 0.24 | 446 ms |
| coeficiente SVM 0.3 | 84/170 (49.4118 %) | 0.08 | 446 ms |

Cuadro 2: Resultados HOG (tiempo real)

Los resultados para la evaluación en tiempo real se muestran en la tabla 2. En el caso del clasificador por defecto el acierto es el 63.9535 %, con 0.25 falsos positivos por frame. El tiempo de evaluación de cada frame es de 442 ms. A continuación se ha evaluado el detector sin incremento de ventana. Como se aprecia, la tasa de detección disminuye considerablemente, hasta el 2.6 %. Sin embargo, también lo hace el tiempo de ejecución, debido a que no tiene que volver a buscar personas con un tamaño de ventana diferente. La tasa de falsos positivos también desciende notablemente, hasta producirse un falso positivo cada 500 frames. Como era de suponer, al no haber incremento en el tamaño de la ventana se realizan muchas menos evaluaciones, disminuyendo tanto el número de aciertos como de falsos positivos.

A continuación se ha variado el tamaño de paso de ventana (relativo al método de escaneo exhaustivo). Tras variar el valor de paso de ventana de los 8x8 píxeles por defecto a 16x16 píxeles, el tiempo de ejecución se ha reducido a los 270 ms (frente a 442 ms en la primera versión). Sin embargo, también disminuye el número de detecciones hasta el 25.62 % y la tasa de falsos positivos, dándose uno cada 29.4 frames. También se ha probado a reducir este valor hasta los 4x4 píxeles. El principal inconveniente que presenta es el alto tiempo de ejecución, de más de un segundo y medio por frame, lo que hace que su aplicación en un sistema de conducción real no sea aconsejable. Aunque la tasa de detección es la más alta de las pruebas realizadas (75 %), también lo es la cantidad de falsos positivos por frame.

El último parámetro variado es la distancia que tiene que haber al plano del detector SVM para que se dé un positivo, que por defecto está establecido a cero. Primero se ha incrementado este coeficiente a 0.1. Mientras que el tiempo de ejecución se mantiene prácticamente sin variación, la tasa de falsos positivos baja, pasando a tener un falso positivo cada 4 frames a uno cada 5.88. Además, la tasa de detección aumenta ligeramente frente a la versión por defecto, hasta un 64.7399 %. El siguiente valor probado es de 0.01, obteniendo una tasa de detección y de falsos positivos algo menor pero similar que en el caso inicial. El tiempo de ejecución se mantiene prácticamente idéntico. Por último, el coeficiente SVM se ha cambiado a 0.3. En este caso ha bajado la tasa de detección considerablemente, así como la tasa de falsos positivos. El tiempo de ejecución es prácticamente idéntico al caso original.

### 4.5.2. Resultados (evaluación del dataset completo)

A continuación se vuelve a evaluar el detector HOG en las siete combinaciones de parámetros de la sección anterior, pero evaluando las 21790 imágenes del dataset. En este caso no se envía una nueva imagen hasta que el nodo detector no ha acabado de analizar la imagen anterior.

| | detecciones | Falsos positivos/frame | t. medio de ejecución/frame |
|---|---|---|---|
| por defecto | 496/766 (64.752 %) | 0.250 | 442 ms |
| sin incremento de ventana | 20/766 (2.61097 %) | 0.002 | 49 ms |
| paso de ventana 16 píxeles | 171/766 (22.3238 %) | 0.034 | 271 ms |
| paso de ventana 4 píxeles | 612/766 (79.895 %) | 0.765 | 1.494 s |
| coeficiente SVM 0.1 | 454/766 (59.2689 %) | 0.170 | 442 ms |
| coeficiente SVM 0.01 | 487/766 (63.577 %) | 0.242 | 439 ms |
| coeficiente SVM 0.3 | 375/766 (48.9556 %) | 0.078 | 441 ms |

Cuadro 3: Resultados HOG (dataset completo)

Los resultados son, en general, similares al caso anterior. La mayor tasa de acierto sigue dándose al decrementar el tamaño de paso de ventana hasta los 4 píxeles, que ahora es de un 79.895 %, pero también se obtiene la mayor tasa de falsos positivos. La principal diferencia con los resultados de la sección anterior es en el caso del coeficiente SVM. Viendo los resultados de la tabla 2 podía parecer que el mejor valor de entre los probados de este coeficiente es 0.1. Sin embargo, viendo estos nuevos resultados se deduce que incrementar este valor empeora la tasa de detección, aunque disminuye el número de falsos positivos.

Tras las pruebas realizadas se concluye que el mejor detector (en cuanto a tasa de acierto) de los considerados es el de tamaño de paso de ventana de 4. La razón principal de ello es que, al ser el tamaño de paso menor, se generan muchas más

regiones de interés, pudiéndose detectar más peatones. Sin embargo, el alto tiempo de ejecución por frame hace que no sea posible aplicarlo en un sistema con requisitos de tiempo tan altos como los de un sistema de asistencia a la conducción.

El parámetro que más parece influir en los resultados son los relacionados con la generación de candidatos: a menor paso de ventana, mayor número de candidatos y tasa más alta de detección. A tamaño de paso de ventana más alto, menor número de candidatos, por lo que se podrían estar perdiendo candidatos válidos. La misma pérdida de candidatos puede producirse al no incrementar el tamaño de ventana. Por otro lado, la variación del incremento o del paso de ventana afectan directamente al tiempo de ejecución, como era de suponer, debido a que al generar las regiones de interés mediante el método de la ventana deslizante estos dos parámetros afectan directamente al número de ROIs generadas que se deben analizar. Otra consecuencia es que baja la tasa de detección, así como la de falsos positivos, debido a que se analizan menos regiones.

Tras las pruebas realizadas, si se ha de optar por una solución de compromiso entre una tasa de acierto razonable y tiempo de ejecución, probablemente la mejor opción sea establecer los parámetros por defecto o el coeficiente SVM en 0.01.

Una de las posibles razones por las que no se llega a un porcentaje más alto de detección es el tamaño de ventana. En la implementación utilizada ésta es de 64x128, sin embargo en el data set se considera que una persona de como mínimo 72 píxeles ha de ser detectada (valor de confianza de 1). Si el detector no se ha entrenado para detectar peatones de este tamaño, es posible que no los detecte o lo haga con muy poca frecuencia.

Una última consecuencia que se deduce de los resultados (tanto en tiempo real como al evaluar el dataset completo) es la relación directa entre los falsos positivos por frame y las detecciones correctas: en los casos en los que mayor tasa de acierto se consigue, también se alcanza un mayor número de falsos positivos.

## 4.6. Aplicación v4l2stereo

La aplicación "v4l2stereo" es un programa que implementa utilidades para visión estéreo. Es de código abierto bajo licencia GNU y está dentro del proyecto "libv4l2cam", que consiste en una librería que utiliza el API de "Video4Linux 2" para gestionar de forma sencilla los ajustes básicos de una cámara. La aplicación contiene parámetros intrínsecos de la webcam Minoru, por lo que es una buena opción para comprobar su funcionamiento. A continuación se detalla el proceso de prueba y calibración de la cámara, probando también una de las características más interesantes para la temática de la tesina: detección de objetos verticales. Esta función puede ser útil, por ejemplo, en el proceso de carga y descarga de pasajeros en un autobús.

Al conectar la webcam vía USB aparecerán dos dispositivos en "/dev/video" correspondientes con cada una de las cámaras. Se puede iniciar el programa "v4l2stereo" introduciendo en un terminal uno de los siguientes comandos:

```
v4l2stereo --dev0 /dev/video2 --dev1 /dev/video1
v4l2stereo --0 /dev/video2 --1 /dev/video1
```

donde el primer parámetro es la cámara izquierda y el segundo parámetro es la cámara derecha. Este comando puede variar ligeramente en función de dónde se monten los dispositivos. Además, si tenemos alguna otra cámara es posible que dé problemas y sólo funcione la primera orden.

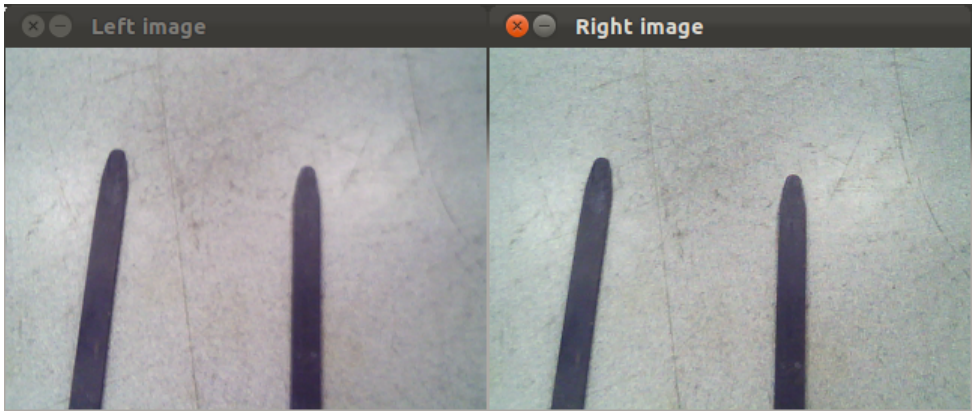


Figura 59: Imagen sin rectificar

Como resultado se obtienen las imágenes de las dos cámaras sin rectificar (figura 59). Otra de las características a probar es añadiendo la opción "–features", que detecta bordes. El resultado obtenido es el mostrado en la figura 60.

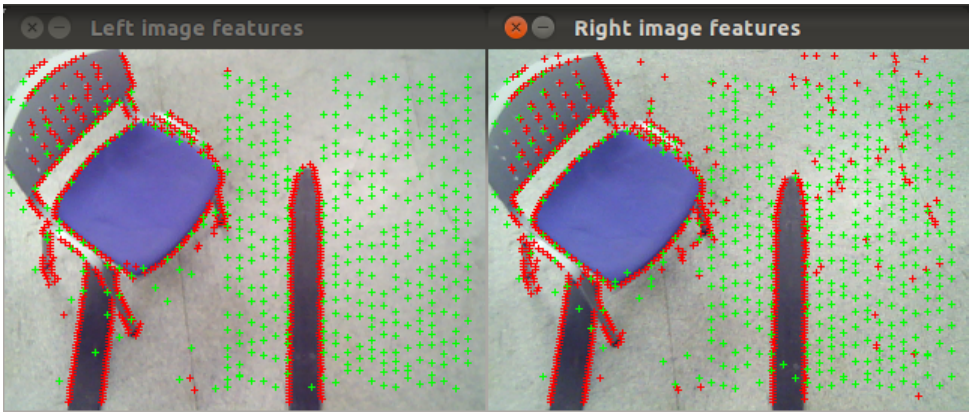


Figura 60: Detección de bordes en la imagen

### 4.6.1. Calibración

Para la obtención de una imagen de disparidad correcta primero hay que realizar la calibración de la cámara. Para ello, se puede utilizar el programa de demostración con la opción "–calibrate" como se muestra a continuación:

```
v4l2stereo --dev0 /dev/video2 --dev1 /dev/video1
--calibrate "6 9 815" --calibrationimages 120
--calibrationfile calibration.txt
```

Mediante la opción –calibrate "x y size" se indica que se va a realizar la calibración con un patrón de calibración de "x" cuadrados en horizontal e "y" cuadrados en vertical de tamaño "size" (en milímetros). En este caso, el patrón de calibración utilizado es de 6x9 cuadrados con un tamaño de 81.5 milímetros, como el que se muestra en la figura 61. Además de las características del patrón de calibración, se

puede establecer el número de imágenes a utilizar para la calibración con el argumento "–calibrationimages", que por defecto está establecido a 20. En este caso se ha calibrado con 120 imágenes para obtener un mejor resultado.

Figura 61: Patrón de calibración utilizado

Una vez lanzada la aplicación para calibrar las cámaras, hay que intentar mover el patrón por toda el área de visión, con diferentes inclinaciones y acercando y alejando el patrón de la imagen. En la figura 62 se muestra un instante del proceso. Una vez se han tomado las imágenes necesarias, automáticamente se calculan parámetros como los parámetros intrínsecos de la cámara, la homografía, las rotaciones y las traslaciones relativas. Por último aparece una ventana en la que se puede ajustar la rectificación vertical (figura 63).

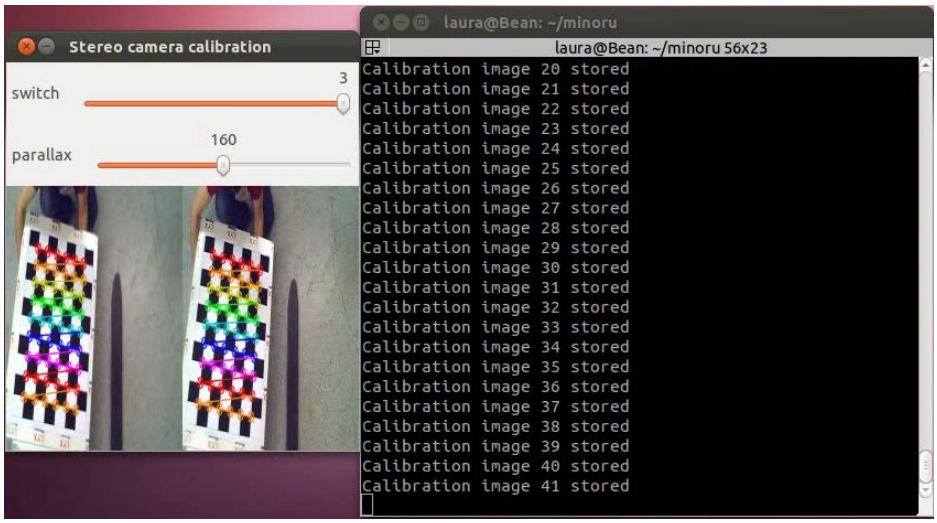


Figura 62: Proceso de calibración

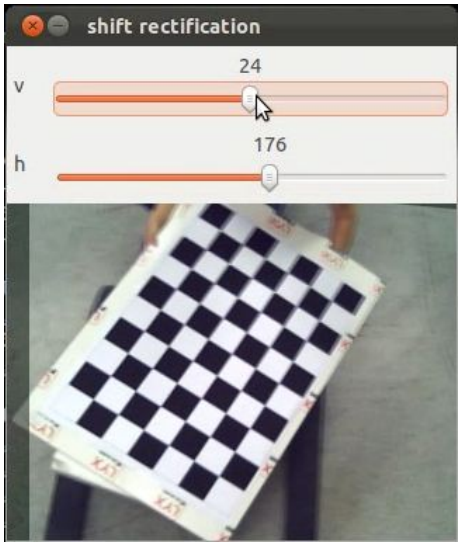


Figura 63: Ajuste de la rectificación

Tras ajustar los resultados, los parámetros de calibración se almacenan en un fichero. Por defecto, este fichero se almacena en el directorio de trabajo actual con el nombre de "calibration.txt", aunque también se puede especificar este valor con el parámetro "–calibrationfile nomFich.txt". Esta opción también se puede utilizar para especificar el fichero de calibración a utilizar cuando se está calculando la disparidad o la detección de objetos.

Como nota final hay que señalar que el fichero de calibración se genera con números decimales con una coma separando parte entera y decimal. Hay que modificar este fichero de forma que en los números la parte entera y la decimal esté separada con un punto, de lo contrario procesamientos más avanzados como la distinción de los objetos verticales no funcionan de forma adecuada.

### 4.6.2. Imagen de disparidad

```
v4l2stereo --dev0 /dev/video2 --dev1 /dev/video1 --disparitymap
--calibrationfile calibration.txt
```

Mediante el programa de demostración, y tras calibrar las cámaras, se puede calcular la imagen de disparidad con la opción "–disparitymap". Esto mostrará el mapa de disparidad, mostrando una imagen con diferentes colores en función de la distancia a la que se encuentren. El algoritmo que utiliza para el cálculo de la disparidad es el ELAS[49, 50].

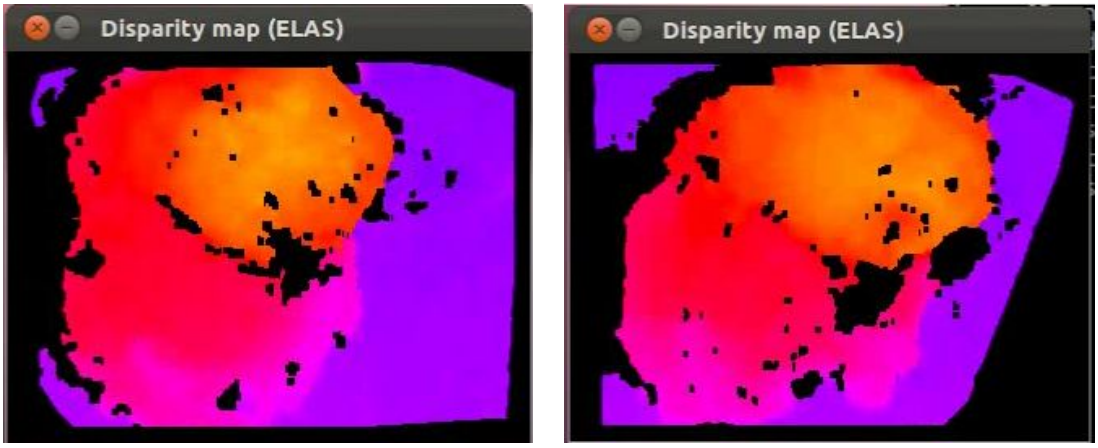


Figura 64: Imagen de disparidad (izquierda) y de disparidad con ecualización del histograma (derecha)

Para mejorar los resultados de la disparidad se puede utilizar la opción "–equal", que realiza una ecualización del histograma. En la figura 64 se ven los resultados obtenidos para las dos opciones. En ambos casos se puede apreciar la cabeza de la persona en colores naranja y rojo y parte del cuerpo en color morado claro, mientras que el fondo aparece en un tono más oscuro.

También se puede establecer un umbral sobre la imagen de disparidad. De esta manera, si se va aumentando el umbral, se ve como paulatinamente se va eliminando el fondo. Por ejemplo, a partir de un umbral de 5, sólo aparece una pequeña parte del fondo y el cuerpo de la persona aparece por completo, como se muestra en la figura 65. Las dos pequeñas manchas que aparecen en la parte inferior son debidas a las palas de la carretilla. Por otro lado, aumentando más este valor empiezan a desaparecer las piernas de la persona, por lo que habrá que elegir con cuidado este umbral ya que se podrían estar eliminando regiones de interés. El umbral establecido por defecto es cero.

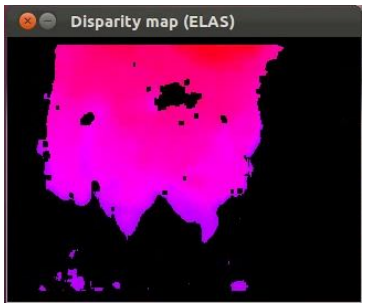


Figura 65: Imagen de disparidad con valor del umbral 5

### 4.6.3. Sustracción de fondo

Otra de las opciones que permite la demo es realizar una sustracción de fondo. Para ello, primero hay que aprender el fondo mediante la opción "–learnbackground archivo.dat". Hay que asegurar que cuando se aprenda el modelo no hay ningún objeto en la zona de visión.

```
v4l2stereo --dev0 /dev/video2 --dev1 /dev/video1
--learnbackground archivo.dat --equal
```

Una vez se ha aprendido el modelo, se puede utilizar mediante la opción "–backgroundmodel archivo.dat –disparitythreshold num", donde especificamos el archivo donde se almacenó el modelo del fondo. También hay que especificar un umbral que se utilizará para discriminar entre fondo y objetos. Por ejemplo, si se establece un umbral de 10 sólo se mostrarán los píxeles con una disparidad relativa al fondo mayor de 10. Disminuir el valor del umbral hace que la persona se detecte en cuanto entra en el campo de visión de la cámara, aunque si este valor es muy bajo aparecen algunas partes del fondo. Por otro lado, aumentar el umbral implica que la persona debe estar más cerca de la carretilla para ser detectada.

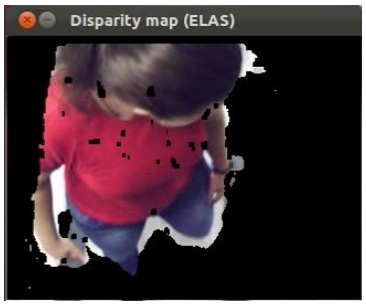


Figura 66: Sustracción de fondo, umbral 3

Por último, señalar que este método solo sirve para entornos estáticos, dado que se tiene que aprender previamente un modelo del entorno.

### 4.6.4. Overhead

También se puede mostrar una vista superior de la proyección del mapa de disparidad. Esto puede ser útil para actualizar mapas de ocupación. La opción a elegir en este caso es "–overhead", obteniéndose imágenes como las de la figura 67.

```
v4l2stereo --dev0 /dev/video2 --dev1 /dev/video1
--calibrationfile calibration.txt --equal --overhead
```

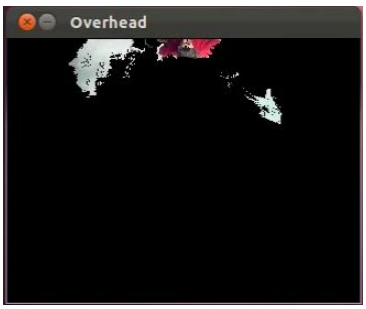


Figura 67: Overhead

### 4.6.5. Cámara virtual

El programa de demostración también es capaz de calcular la nube de puntos. Una forma de visualizar dicha nube es mediante la opción "–vcamera".

```
v4l2stereo --dev0 /dev/video2 --dev1 /dev/video1
--calibrationfile calibration.txt --equal --vcamera
```

Se puede mover esta cámara virtual con las siguientes teclas:

- a (arriba)
- z (abajo)
- x (hacia adelante)
- s (hacia atrás)
- . (izquierda)
- , (derecha)
- 1 (inclinar hacia abajo)
- 2 (inclinar hacia arriba)
- 3 (desplazar hacia la izquierda)
- 4 (desplazar hacia la derecha)
- 5 (girar en sentido antihorario)
- 6 (girar en sentido horario).

Según la documentación, en situaciones en las que la iluminación es baja y el tiempo de exposición es bastante alto, la cámara virtual y el overview puede haber errores en la profundidad debido a que las cámaras no están sincronizadas. Cuando se da este caso, es posible que la imagen oscile en profundidad.

### 4.6.6. Nube de puntos

La próxima opción a considerar con este software es la creación de nube de puntos. Con una orden como

```
v4l2stereo --dev0 /dev/video2 --dev1 /dev/video1 --equal
--vcamera --poserotation 60 0 0 --pointcloud fichero.dat
```

se almacena en el fichero archivo.dat una nube de puntos correspondiente considerando la orientación de la cámara que se indica en "poserotation". Esta nube después se puede visualizar mediante la herramienta "pcloud", también de libre distribución y del mismo proyecto. Los comandos para mover la cámara vistos para la cámara virtual también se pueden utilizar con esta herramienta.

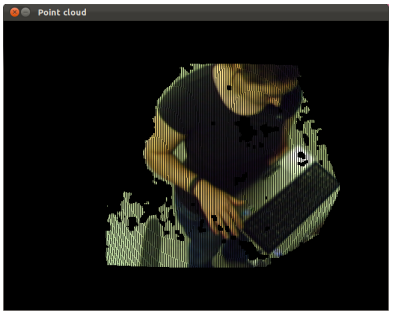


Figura 68: Nube de puntos

### 4.6.7. Detección de obstáculos verticales (cámara orientada al plano del suelo)

Si se posiciona la cámara de forma que esté mirando hacia abajo a una superficie plana, se pueden detectar obstáculos u objetos que tengan estructura vertical. En este caso, se ha orientado la cámara hacia el suelo, con una orientación aproximada de 60 grados con respecto a la horizontal. Para ello, la opción a utilizar es "–obstacles". También se puede especificar la orientación de la cámara (en grados) mediante la opción "–poserotation "tilt pan roll"".

```
v4l2stereo --dev0 /dev/video2 --dev1 /dev/video1
--calibrationfile calibration.txt --equal
--obstacles --poserotation "60 0 0"
--obstaclethreshold 3000 --obstaclecellsize 40
```

Otras opciones a probar son "–obstaclethreshold obsThres" y "–obstaclecellsize cs", que determinan el umbral y el tamaño de la celda. De esta manera, partiendo de la nube de puntos obtenida a partir del par de imágenes, se calcula la proyección de los puntos sobre el plano que representa el suelo. Con esta proyección se va actualizando el mapa de obstáculos aplicando una función gausiana: en la celda del mapa de obstáculos que corresponde con la proyección se incrementa el contador en cuatro unidades, y en las celdas colindantes se incrementa en una unidad. Posteriormente se contabilizan cuántos puntos hay dentro de cada celda de tamaño "cs", y si hay más de obsThres puntos, se considera que es un objeto vertical y aparece en la imagen. En las zonas en las que este umbral no se cumple, no se muestra nada.

De esta manera, ajustando los valores de los parámetros de tamaño de celda y el umbral se puede conseguir que se elimine el fondo y sólo aparezcan los objetos o personas a detectar. Tras las pruebas realizadas, se ha comprobado que para un mismo umbral, disminuyendo el tamaño de la celda se detectan menos objetos. Por otra parte, si se aumenta este valor cada vez se detecta a la persona de manera más nítida, aunque si el valor se aumenta demasiado se empieza a mostrar también partes del fondo y del suelo. Estos resultados se muestran en la figura 69, donde para un tamaño de celda de 10 apenas se ve a la persona, mientras que para un valor de 80 aparece gran parte del fondo, aun cuando no hay ninguna persona en el área de visión. Un valor de tamaño de celda alrededor de 40 presenta un buen resultado, ya que el contorno de la persona aparece por completo y no aparece apenas fondo.

Por otra parte, fijando un valor de tamaño de celda a 40 y variando el umbral, se comprueba que cuando este valor es muy bajo aparece el fondo de la escena.

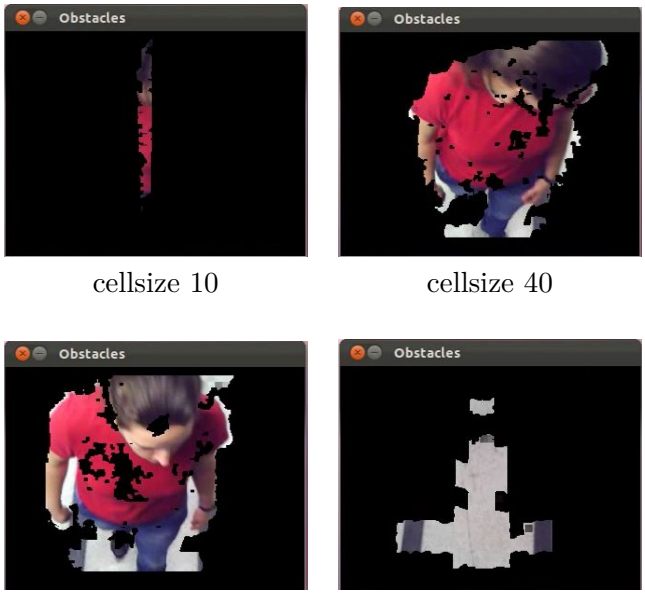


Figura 69: Detección de obstáculos: variación del tamaño de celda

En la figura 70 se muestran algunas de las pruebas realizadas. Para la situación de la cámara que se ha especificado, aparecen partes del fondo hasta un umbral de 1200 (para el tamaño de celda fijo a 40). En este valor, aunque la persona aparece bien definida, aun aparece algo de fondo de forma intermitente. Estas zonas son de pequeño tamaño y generalmente corresponden con las palas de la carretilla. Si se continua aumentando el valor del umbral, el fondo se elimina por completo, pero la persona tiene que estar cada vez más próxima a la cámara. Para el caso de un umbral de 1500 la persona aparece cuando está a una distancia de 1.40 metros, mientras que para un umbral de 5000, la persona no aparece hasta que está a unos 80 centímetros. Si se sigue aumentando este umbral la persona cada vez aparece peor definida, pudiendo llegar a no aparecer.

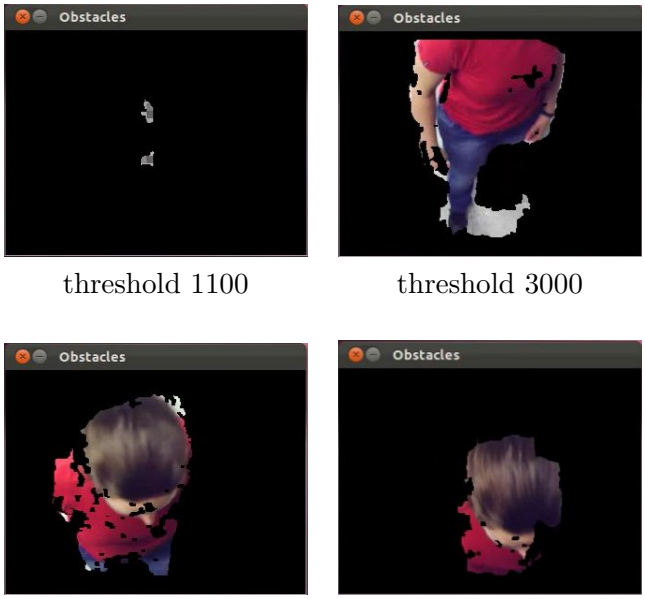


Figura 70: Detección de obstáculos: variación del umbral

Con este método también se pueden detectar múltiples objetos. Sustituyendo la opción "–obstacles" por la opción "–objects" se discriminan los diferentes objetos representando cada uno de ellos con un color diferente.

Por último, señalar que en este método también se pueden utilizar los comandos para mover la cámara virtual para desplazar la imagen.

### 4.6.8. Detección de obstáculos verticales (cámara orientada a la línea del horizonte)

En la sección anterior se ha mostrado que la detección de obstáculos verticales con los parámetros de ajuste adecuados da como resultado una imagen en la que se ve a la persona claramente definida. Esta situación podría ser útil en la carga y descarga de viajeros del autobús o para cubrir posibles ángulos muertos donde el conductor no puede ver si hay alguna persona, utilizando una cámara estéreo orientada hacia el plano del suelo. Otra posible situación en la que es útil la detección de personas es durante la conducción. En este caso la cámara estará orientada hacia el horizonte. A continuación se prueba este algoritmo con la secuencia de imágenes del data set considerado.

Se ha creado un nuevo nodo ROS suscrito a las imágenes procedentes del data set. En este caso, al utilizar un sistema estéreo, hay que esperar a que lleguen las dos imágenes, por lo que el método de suscripción es algo más complicado. Como se muestra a continuación, hay que crear un filtro de tipo "TimeSynchronizer" que recoja las imágenes cuando hayan llegado las dos que tengan la misma marca de tiempo. El procedimiento que se ejecuta cuando esto sucede es el indicado en "registrerCallback". En este procedimiento es donde se ejecuta el código necesario para la detección de objetos verticales.

```
message_filters::Subscriber<Image>
    image_sub_left(nh, "stereo/left/image_raw", 1);
message_filters::Subscriber<Image>
    image_sub_right(nh, "stereo/right/image_raw", 1);
TimeSynchronizer<Image, Image>
    sync(image_sub_left, image_sub_right, 10);
sync.registerCallback(boost::bind(&callback, _1, _2));
```

Para adaptar el código de "v4l2stereo" a este caso hay que conocer los parámetros de calibración. En la documentación del data set se pueden encontrar tanto los parámetros intrínsecos como la posición y la orientación de la cámara, indicando que la cámara está inclinada 0.033 radianes con respecto a la horizontal a una altura de 1.17 metros del suelo y a 2 metros de la parte frontal del vehículo.

Lo primero a realizar cuando llegan dos imágenes es convertirlas al formato de OpenCV, ya que se reciben como mensajes de ROS. Esto se hace con el método "imgMsgToCv" de la clase de ROS "CvBridge". Tras esto, se realiza una ecualización del histograma de la imagen y se calcula la disparidad mediante el algoritmo ELAS. A partir de la imagen de disparidad se halla la nube de puntos 3D transformando los puntos con respecto al sistema de coordenadas de la cámara y, a partir de esta nube, se obtiene el mapa de ocupación proyectando los puntos sobre el plano del suelo. Una vez obtenido el mapa de ocupación se obtiene la imagen de objetos verticales.


Figura 71: Detección de obstáculos verticales: data set

En la figura 71 se ve un ejemplo de una imagen del data set tras aplicar el algoritmo. La imagen de la izquierda es la imagen original, mientras que la de la derecha es la imagen con los objetos verticales. Como se aprecia en esta última no aparece el cielo, sin embargo, además e los objetos verticales como farolas y árboles aparece gran parte del suelo; región que, además de no ser un objeto vertical, no interesa analizar.

Analizando la nube de puntos correspondiente (figura 72X se obtiene una posible explicación para este problema. En la zona del suelo aparece una gran cantidad de puntos y, al acumularlos en celdas, superan fácilmente el umbral establecido. Una posible solución es aumentar este umbral, pero se correría el riesgo de eliminar objetos verticales.

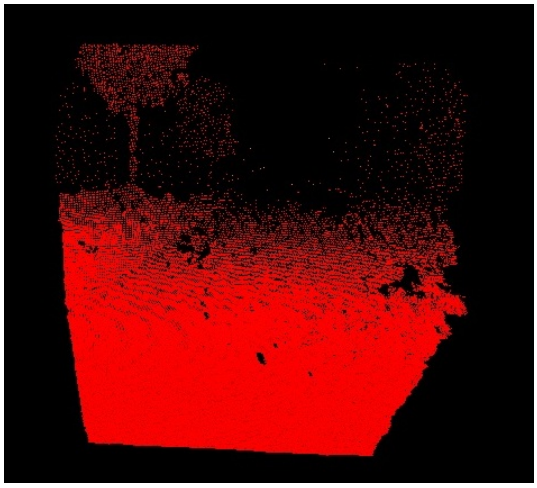

Figura 72: Nube de puntos correspondiente a la Figura 71

Una solución más adecuada es aplicar un filtro a la nube de puntos para eliminar el suelo. Para ello, se ha utilizado la librería PCL.uDado que la nube de puntos obtenida mediante el software "v4l2stereo" está almacenada en una imagen de OpenCV, hay que hacer una conversión entre este formato y el formato de nube de puntos de PCL. La nube de puntos original está almacenada en una imagen IplImage de tres canales, de forma que para cada píxel i se almacenan las coordenadas x, y, z del punto de la nube correspondiente, en lugar de almacenar la intensidad del color. Para crear una nube de puntos PCL se ha de elegir un tipo de punto. La librería PCL dispone de varias estructuras para ello; en este caso se ha elegido la estructura "pcl::PointXYZI" que además de las coordenadas del punto posee un entero adicional que se utilizará para almacenar el píxel al que corresponde dicho punto, para posteriormente poder hacer la correspondencia entre nube de puntos e imagen.

Una vez obtenida la nube de puntos en el formato PCL ya se pueden aplicar las herramientas que proporciona esta librería. Como ya se ha comentado, se va a aplicar un filtro sobre la coordenada y, de manera que se eliminen aquellos puntos cuya coordenada y está entre -1000 y 0.40. La forma de aplicar este filtro es la siguiente:

```
pcl::PassThrough<pcl::PointXYZI> filtro;
filtro.setFilterFieldName("y");
filtro.setFilterLimits(-1000, 0.40);
filtro.setInputCloud(cloud);
filtro.setFilterLimitsNegative(false);
filtro.filter(*cloudFilt);
```

Sobre la nube filtrada también se va a aplicar un método de downsampling, para reducir aún más el número de puntos. El método utilizado es la clase "VoxelGrid" de la librería PCL. Esta clase crea cajas 3D en el espacio de manera que todos los puntos que recaen en esa caja se aproximan con su centroide. Hay que establecer el tamaño de los vóxeles, en este caso se ha establecido a 50 centímetros.

```
pcl::VoxelGrid<sensor_msgs::PointCloud2> voxgrid;
voxgrid.setInputCloud(c_ini);
voxgrid.setLeafSize (0.5f, 0.5f, 0.5f);
voxgrid.filter (point_cloud_msg);
```

En las figuras 73 y 74 se ve el resultado de aplicar estos dos filtros a la nube de puntos. Como se ve, se ha eliminado gran parte del suelo. Ajustando los parámetros de tamaño de celda y umbral se mejora más aún el resultado. Un ejemplo de ello es la figura 75, donde después de ajustar el tamaño de celda a 5 centímetros y el umbral a 20000 puntos el peatón que cruza la calzada aparece prácticamente aislado del resto de la imagen.

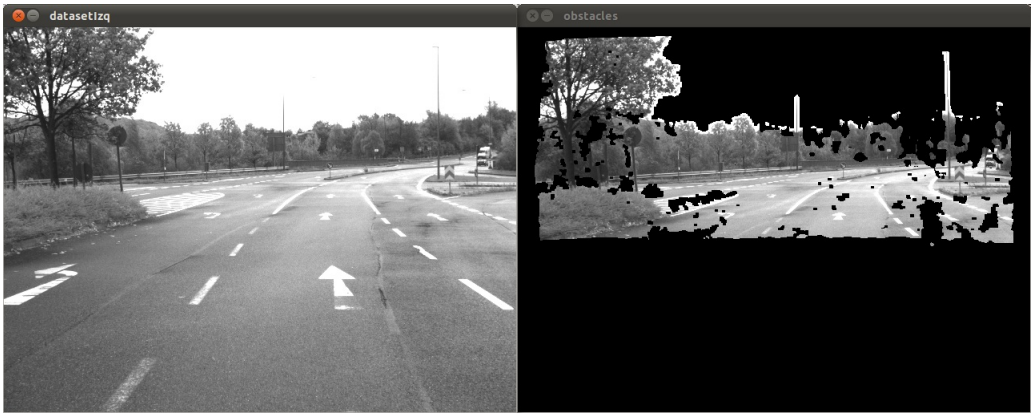


Figura 73: Detección de obstáculos verticales tras filtrado y downsampling

Figura 74: Nube de puntos correspondiente a la Figura 73


Figura 75: Detección de obstáculos verticales tras filtrado y ajuste de parámetros

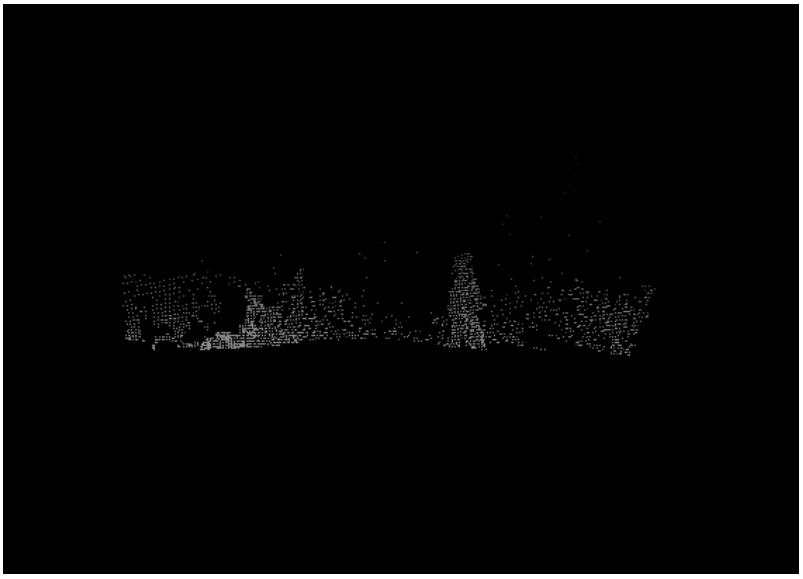

Figura 76: Nube de puntos correspondiente a la Figura 75

Para eliminar zonas pequeñas y mejorar el resultado se aplican operaciones de erosión y dilatación sobre la imagen binarizada. Concretamente, primero se erosiona diez píxeles para posteriormente dilatar 10 píxeles. En la imagen 77 se ve el proceso de aplicar estas operaciones sobre una de las imágenes del dataset. En la parte superior se muestra la imagen original, en la parte inferior izquierda se muestra el resultado antes de aplicar la erosión y la dilatación, mientras que la imagen inferior derecha se ve el resultado de aplicar estas operaciones.

Si bien hasta aquí se ha hablado de objetos verticales, falta discernir cuales de ellos son personas y cuales no. En la siguiente sección se comentará este aspecto.

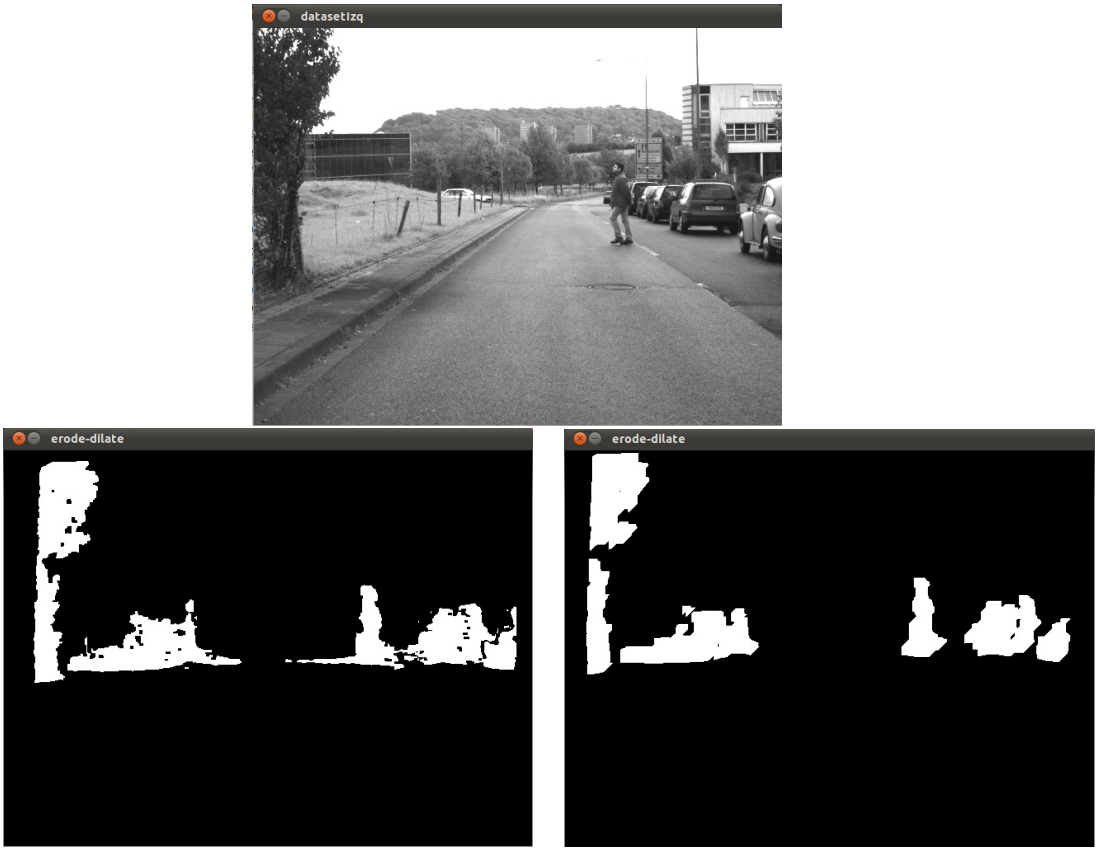


Figura 77: Operaciones de erosión y dilatación

## 4.7. Detección de objetos verticales y detector HOG

Tras el apartado anterior se ha conseguido distinguir los objetos verticales del fondo, pero dado que las personas no son los únicos objetos verticales que puede haber en un entorno urbano hay que aplicar un paso más. En este caso se ha optado por calcular el histograma de gradientes orientados y clasificarlos con un clasificador SVM, es decir, la técnica presentada por Dalal y Triggs [4]. A diferencia de lo expuesto en la sección 4.5 donde dicho método se aplicaba a la totalidad de la imagen, en este caso solo se aplicará a las zonas con objetos verticales.

El primer paso es obtener las regiones de interés de la imagen. Tras la detección de objetos verticales se obtiene una serie de "burbujas" de la imagen original en las que se ven los objetos verticales. La idea será obtener un contorno rectangular de estas burbujas, crear una imagen con cada una de ellas y aplicarles el detector HOG de OpenCV.

La librería OpenCV implementa la función "cvFindContours" que obtiene contornos a partir de una imagen binaria. Para cada contorno obtenido, la función "cvBoundingRect" obtiene el rectángulo que contienen la zona de la imagen a analizar. A dicha zona se le añade un offset, haciendo más grande la región de interés y asegurando que se toma completamente. Dado que en la fase de detección de objetos verticales se filtra el suelo, en algunos casos las piernas de los peatones no aparecen completamente. Por esta razón se añade una zona por debajo de la región de interés inicial de 35 píxeles. Al borde superior se le añaden 25 píxeles y a los límites laterales 30 píxeles.

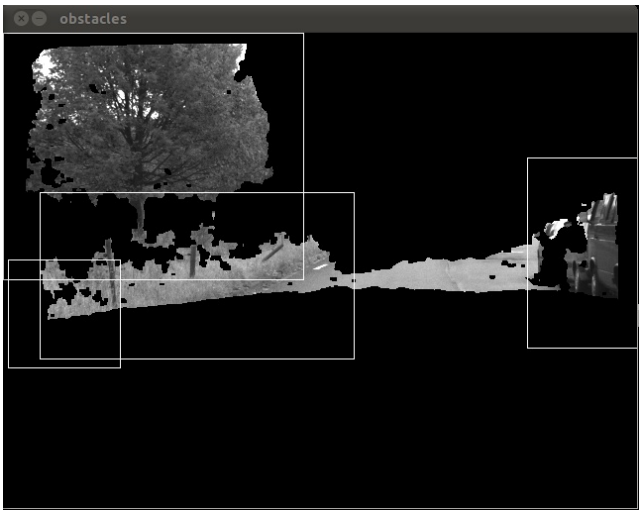


Figura 78: Ejemplo de regiones internas o muy solapadas

Las regiones de interés obtenidas a partir de los contornos se procesan para que no aparezcan ROIs dentro de otras ROIs o dos regiones que se solapan casi en su totalidad, como ocurre en la figura 78. En ambos casos se calcula el rectángulo intersección de las dos regiones a comparar. Si el área de la intersección entre el área del rectángulo es mayor que 0.95 (coinciden en un 95 %), la región se elimina del conjunto de regiones por ser una región interna a otra. Por otro lado, si la relación entre el área del rectángulo intersección y el área del rectángulo es mayor que 0.5, los rectángulos se fusionan, generando una sola región.

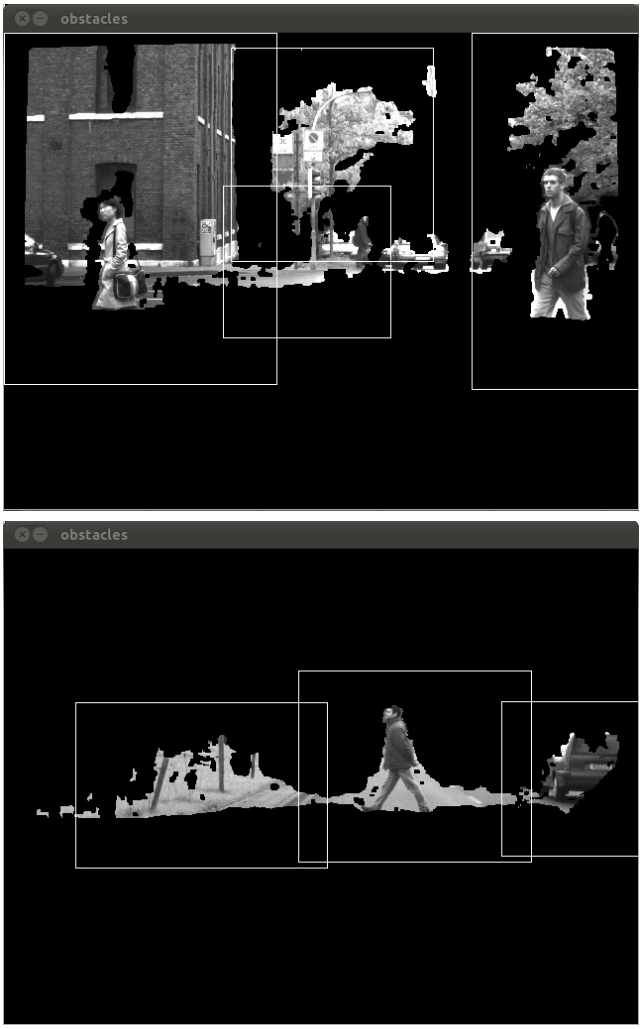


Figura 79: Detección de obstáculos: regiones extraídas

Como último paso, se aplica el detector HOG de OpenCV. En la figura 79 se muestran ejemplos de las regiones de interés a analizar. El detector requiere zonas que sean al menos igual de grandes que el tamaño de ventana mínimo, que en el HOG por defecto es de 64x128. Para extraer estas regiones y que se analicen mediante el algoritmo HOG de OpenCV se extraen regiones de interés de la imagen original mediante funciones de OpenCV del tamaño y posición hallados durante la extracción de contornos

```
subimg = cvCreateImage(cv::Size(rectangle.width,
                        rectangle.height),
                        original_left_image->depth,3);
cvSetImageROI(original_left_image, caja);
cvCopy(original_left_image, subimg, NULL);
cvResetImageROI(original_left_image);
hog_.detectMultiScale(subimg, found, hit_threshold_,
                      cv::Size(4,4), cv::Size(24,16),
                      1.05, group_threshold_);
```

Para que no se produzcan errores por utilizar imágenes muy pequeñas, se redi-

mensiona la imagen al tamaño mínimo, manteniendo la razón de aspecto. Por otro lado, tampoco tendría sentido analizar regiones de interés muy pequeñas, ya que no es lógico que contengan personas. Por esta razón se limita el tamaño mínimo de la región a analizar, analizando sólo aquellas con tamaño mayor que la mitad del tamaño de ventana por defecto del detector HOG.

### 4.7.1. Resultados

De manera análoga a lo realizado con el nodo que implementaba HOG, se ha evaluado el funcionamiento de este método con el data set "Daimler Stereo Pedestrian Detection Benchmark", comparando con el ground truth 2D con los objetos con confianza 1 (peatones de al menos 72 píxeles de altura). Se ha comparado su funcionamiento tanto en tiempo real (los 27 minutos de conducción que representa el dataset) como frame a frame. Los resultados en cada caso se muestran en las tablas 4 y 5 respectivamente, en las que se compara con el detector HOG con parámetros por defecto y con paso de ventana de 4 píxeles, que era el que mayor porcentaje de acierto lograba.

| | detecciones | Falsos positivos/frame | t. medio de ejecución/frame |
|---|---|---|---|
| HOG (paso de ventana 4 píxeles) | 39/52 (75 %) | 0.766 | 1.518 s |
| objetos verticales + HOG (paso 4 píxeles) | 55/72 (76.3889 %) | 0.513 | 0.966743 s |
| HOG por defecto | 110/172 (63.9535 %) | 0.25 | 442 ms |
| objetos verticales + HOG por defecto | 78/138 (56.5217 %) | 0.1868 | 513 ms |

Cuadro 4: Resultados HOG - v4l2stereo + hog (tiempo real)

| | detecciones | Falsos positivos/frame | t. medio de ejecución/frame |
|---|---|---|---|
| HOG (paso de ventana 4 píxeles) | 612/766 (79.895 %) | 0.765 | 1.498 s |
| objetos verticales + HOG | 607/766 (79.2428 %) | 0.682 | 1.09531 s |
| HOG por defecto | 496/766 (64.752 %) | 0.250 | 442 ms |
| objetos verticales + HOG por defecto | 471/766 (61.4883 %) | 0.231 | 524ms |

Cuadro 5: Resultados HOG - v4l2stereo + hog (dataset completo)

En el caso de HOG con paso de ventana de 4 píxeles, tanto en "tiempo real" como con el dataset completo se presentan tasas de acierto muy similares, en torno al 75 % en la ejecución en tiempo real y del 79 % detectando frame a frame. En este

último caso, en la versión con extracción de objetos verticales se han detectado en total cinco peatones menos que con el HOG estándar.

El tiempo de ejecución se ha visto reducido a pesar de haber añadido la parte de detección de objetos verticales (unos 230 milisegundos de tiempo de cómputo añadido). En el caso del dataset completo, el tiempo de ejecución medio ha sido de 1.095 segundos, 403 milisegundos menos que sin la extracción de objetos verticales, llegando en algunos casos en los que no había objetos verticales a 230 milisegundos por frame. El número de falso positivos también se ha visto reducido ligeramente, pasando de 0.765 falsos positivos por frame a 0.682.

De las pruebas realizadas en tiempo real se pueden extraer conclusiones similares: tasa de acierto muy parecida al HOG, tiempo de ejecución menor (en este caso, más de medio segundo menos) y tasa de falsos positivos reducida hasta 0.513 falsos positivos por frame. En este caso la tasa de aciertos sube ligeramente frente al HOG estándar debido al menor tiempo de ejecución, ya que esto hace que se puedan analizar más imágenes, por lo que el número total de personas detectadas y esperadas aumenta.

Probando el método de segmentación de objetos verticales con el HOG con parámetros por defecto se obtienen tasas de acierto algo menores que con el HOG simple en ambos casos. Analizando frame a frame (tabla 5), la tasa de acierto solo baja un 3.3 %; disminuyendo ligeramente la tasa de falsos positivos de 0.250 a 0.231. Sin embargo, el tiempo de cómputo aumenta en 82 milisegundos. Teniendo en cuenta que la fase de objetos verticales supone 230 milisegundos aproximadamente, el tiempo de cómputo de la parte del detector HOG sí se ha visto reducido. No obstante, haría falta realizar más optimizaciones sobre el código de detección de objetos verticales para que el tiempo global fuera menor. En el caso de ejecución en tiempo real, la tasa de aciertos baja de 63.95 % a 56.52 % y la tasa de falsos positivos también baja de 0.25 a 0.1868. El tiempo de cómputo es 71 milisegundos mayor que en el caso del HOG simple.

# 5. Conclusiones y trabajos futuros

Los principales objetivos de la presente tesina eran conocer los algoritmos existentes en cuanto a detección de personas, tanto a nivel de bibliografía como de implementaciones existentes, así como la experimentación e implementación de algunos de estos métodos. Debido a que la tesina se encuentra asociada al proyecto SAFEBUS de seguridad de autobuses la tesina se ha orientado hacia la detección de personas en sistemas de asistencia a la conducción (ADASs), ya que uno de los objetivos de este proyecto de investigación es la implementación de un sistema de este tipo.

Primero se ha realizado un estado del arte, para conocer métodos existentes en la bibliografía que puedan resultar adecuados para esta aplicación y se han comentado las características más relevantes de los sensores que se suelen utilizar (sensores de visión, láser y radar); siendo en este caso los más utilizados los sensores de visión. Se han presentado las diferentes fases que se dan en una aplicación de detección. La primera de ellas es la generación de candidatos, que abarca desde métodos más básicos, como el de ventana deslizante en visión o la segmentación por distancia de salto en láser, a otros más complejos basados en hallar el perfil de la carretera. La segunda fase es la de la clasificación, en la que se han distinguido tres vertientes principales: comparación con siluetas, detección de cuerpo completo (métodos holísticos) o por partes. En general estos dos últimos generan un conjunto de características que después se clasifican mediante clasificadores como SVM (Support Vector Machines) o AdaBoost. Tras la clasificación, algunos métodos añaden fases de verificación o tracking, para filtrar falsos positivos, realizar predicción de colisiones y consolidar detecciones en el tiempo. También se han estudiado algoritmos ya implementados en la plataforma ROS, y la aplicación "v4l2stereo", de código abierto, capaz de detectar objetos verticales.

En cuanto a experimentación se ha probado mediante la webcam Minoru (una webcam estéreo USB) la detección de objetos verticales disponible en el software "v4l2stereo". La cámara se ha colocado orientada hacia el suelo y a la altura aproximada de un autobús para simular una situación real. Mediante esta aplicación se han podido aislar personas del fondo de la escena, siendo posible su utilización, por ejemplo, en los momentos de carga y descarga del autobús.

Posteriormente se ha probado uno de los métodos principales en cuanto a detección de personas. Se trata de "Histogram of Oriented Gradients", presentado por Dalal y Triggs en 2005. Se ha evaluado la eficacia de dicho detector ejecutándolo sobre un dataset consistente en 27 minutos de conducción en áreas urbanas, analizando cómo influyen los principales parámetros de ajuste en la tasa de detección, los falsos positivos por frame y el tiempo de ejecución. El máximo porcentaje de detección obtenido es del 79.8 %, aunque a costa de un tiempo de ejecución demasiado elevado para una aplicación ADAS que posee importantes restricciones de tiempo. Una solución más económica desde el punto de vista de tiempo de cómputo alcanza el 64.75 % con un falso positivo cada 4 frames.

También se ha probado el algoritmo de detección de objetos verticales con las imágenes del dataset. Sobre el método original se han aplicado una serie de filtros para mejorar los resultados obtenidos, llegándose a segmentar bastante bien los objetos verticales del suelo y el fondo. En la última parte de la tesina se ha aplicado este método como un método de generación de candidatos para ser detectados mediante el algoritmo HOG. Se han logrado resultados de acierto similares a los del

HOG pero disminuyendo el tiempo de cómputo y la tasa de falsos positivos.

Entre los trabajos futuros se encontrarían la inclusión de un módulo de verificación o de tracking, que podría mejorar las tasas de detección al cubrir los espacios entre ejecuciones del detector HOG, además de seguir optimizando el proceso de segmentación de objetos verticales. También se podría investigar el efecto que tendría utilizar una técnica más avanzada para detectar el suelo. En la implementación comentada se realiza un filtrado por altura, asumiendo que la carretera es plana. Esto puede hacer que en carreteras con mucha pendiente se filtre parte de los objetos presentes.

Otro de los trabajos futuros relacionados con el proyecto sería adaptar los algoritmos para que se puedan ejecutar sobre un dispositivo de tipo OMAP (Open Multimedia Applications Platform) como placas de prototipado como "BeagleBoard" o "PandaBoard". Este tipo de dispositivos suelen consistir en un procesador ARM de propósito general y uno o más co-procesadores especializados, como DSPs y aceleradores gráficos (GPU). Algunos productos comerciales que utilizan este tipo de circuitos integrados son smart phones como Samsung Galaxy SII o tablets como BlackBerry PlayBook.

Como objetivo final, el trabajo a realizar sería la implementación de un sistema de protección de peatones para utilizar en autobuses o vehículos de conducción autónoma.

# Referencias


[1] M. Enzweiler and D. M. Gavrila. “Monocular Pedestrian Detection: Survey and Experiments”. *IEEE Trans. on Pattern Analysis and Machine Intelligence, vol.31, no.12, pp.2179-2195, 2009.*

[2] T. Gandhi, M.M. Trivedi, "Pedestrian collision avoidance systems: a survey of computer vision based recent studies," *Intelligent Transportation Systems Conference, 2006. ITSC '06. IEEE , vol., no., pp.976-981, 17-20 Sept. 2006.*

[3] D. Gerónimo, A.M. López, A.D. Sappa, T. Graf, "Survey of Pedestrian Detection for Advanced Driver Assistance Systems," *Pattern Analysis and Machine Intelligence, IEEE Transactions on , vol.32, no.7, pp.1239-1258, July 2010.*

[4] N. Dalal, B. Triggs. “Histograms of Oriented Gradients for Human Detection”. *Proc. IEEE Conf. Computer Vision and Pattern Recognition, vol. 1, pp. 886-893, 2005.*

[5] A. Shashua, Y. Gdalyahu, G. Hayun, "Pedestrian detection for driving assistance systems: single-frame classification and system level performance," *Intelligent Vehicles Symposium, 2004 IEEE , vol., no., pp. 1- 6, 14-17 June 2004.*

[6] M. Bertozzi, A. Broggi, R. Chapuis, F. Chausse, A. Fascioli and A. Tibaldi. “Shape-based pedestrian detection and localization”. *Proc. IEEE International Conference Intelligent Transportation Systems, pp. 328-333, 2003.*

[7] M. Bertozzi, A. Fascioli, M. Sechi. “Shape-based pedestrian detection”. *Proceedings of the IEEE Intelligent Vehicles Symposium, pp. 215-220, 2000.*

[8] A. Broggi, A. Fascioli, I. Fedriga, A. Tibaldi, M.D. Rose. "Stereo-based preprocessing for human shape localization in unstructured environments," *Intelligent Vehicles Symposium, 2003. Proceedings. IEEE , vol., no., pp. 410- 415, 9-11 June 2003.*

[9] I.P. Alonso, D.F. Llorca, M.A. Sotelo, L.M. Bergasa, Pedro Revenga de Toro; J. Nuevo, M. Ocana, M.A.G. Garrido, "Combination of Feature Extraction Methods for SVM Pedestrian Detection," *Intelligent Transportation Systems, IEEE Transactions on , vol.8, no.2, pp.292-307, June 2007*

[10] D. M. Gavrila. “Pedestrian Detection from a Moving Vehicle”. *Proc. of the European Conference on Computer Vision, pp. 37-49, Dublin, 2000.*

[11] D. M. Gavrila, J. Giebel, S. Munder. “Vision-based Pedestrian Detection: The PROTECTOR System”. *IEEE Intelligent Vehicles Symposium, 2004*

[12] D. M. Gavrila, S. Munder. “Multi-Cue Pedestrian Detection and Tracking from a Moving Vehicle”. *International Journal of Computer Vision, vol. 73, 2007.*

[13] U. Franke. “Real-time stereo visión for urban traffic scene understanding”. *Proceedings of the IEEE Intelligent Vehicle Symposium, Detroit, 2000.*

[14] D. Gerónimo, A.D. Sappa, A.L. López, D. Ponsa. “Adaptive Image Sampling and Windows Classification for On-board Pedestrian Detection”. *Proceedings. Fifth International Conf. Computer Vision Systems, 2007.*

[15] F. Xu, X. Liu, K. Fujimura. “Pedestrian Detection and Tracking with Night Vision”. *IEEE Trans. Intelligent Transportation Systems, Mar. 2005.*

[16] A. Broggi, A. Fascioli, M. Carletti, T. Graf, M. Meinecke,"A multi-resolution approach for infrared vision-based pedestrian detection," *Intelligent Vehicles Symposium, 2004 IEEE , vol., no., pp. 7- 12, 14-17 June 2004.*

[17] J. Xavier, M. Pacheco, D. Castro, A. Ruano, U. Nunes, "Fast Line, Arc/Circle and Leg Detection from Laser Scan Data in a Player Driver," *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on , vol., no., pp. 3930- 3935, 18-22 April 2005*

[18] K.O. Arras, O.M. Mozos, W. Burgard, "Using Boosted Features for the Detection of People in 2D Range Data," *Robotics and Automation, 2007 IEEE International Conference on , vol., no., pp.3402-3407, 10-14 April 2007*

[19] L. Spinello, K.O. Arras, R. Triebel, R. Siegwart. “A Layered Approach to People Detection in 3D Range Data”. In *AAAI, 2010.*

[20] Xia Liu; K. Fujimura. "Pedestrian detection using stereo night vision," *Vehicular Technology, IEEE Transactions on , vol.53, no.6, pp. 1657- 1665, Nov. 2004.*

[21] T. Joachims. “Making large-scale svm learning practical”. *B. Schlkopf, C. Burges, and A. Smola. Advances in Kernel Methods - Support Vector Learning. The MIT Press, Cambridge, MA, USA, 1999.*

[22] D. Lowe. “Distinctive Image Features from Scale-Invariant Keypoints”. *International Journal of Computer Vision, vol 60, no. 2, pp.91-110, 2004.*

[23] C. Papageorgiou, T. Poggio. “A Trainable System for Object Detection,”. *International Journal of Computer Vision 38(1), pp. 15–33, 2000.*

[24] S. Chang, R. Sharan, M. Wolf, N. Mitsumoto, J. W. Burdick. “People Tracking with UWB Radar Using a Multiple-Hypothesis Tracking of Clusters (MHTC) Method”. *International Journal of Social Robotics, vol.2, pp. 3-18, 2010.*

[25] S. Chang, N. Mitsumoto, J.W. Burdick. “An algorithm for UWB radar-based human detection”. *Proc. IEEE RadarCon, Pasadena, 2009.*

[26] S. Milch and M. Behrens, “Pedestrian Detection with Radar and Computer Vision”, 2001.

[27] L. Wang, “Texture unit, texture spectrum and texture analysis,” *IEEE Trans. Geosci. Remote Sens., vol. 28, no. 4, pp. 509–512, Jul. 1990.*

[28] O. Martinez Mozos, R. Kurazume, T. Hasegawa. “Muti-Part People Detection Using 2D Range Data”. *International Journal of Social Robotics 2(1), pp. 31-40, 2010.*

[29] D. Comaniciu, D, P. Meer, “Mean shift: A robust approach toward feature space analysis”. *IEEE Transactions on Pattern Analysis Machine Intelligence, vol. 24, pp. 603–619, 2002.*

[30] V. Philomin, R. Duraiswami, L. Davis, "Pedestrian tracking from a moving vehicle," *Intelligent Vehicles Symposium, 2000. IV 2000. Proceedings of the IEEE, pp.350-355, 2000.*

[31] M. Isard, A. Blake. “Contour tracking by stochastic propagation of conditional density”. *Proc. European Conference on Computer Vision, pages 343- 356, Freiburg, Germany, 1996.*

[32] S. Munder, C. Schnorr, D.M. Gavrila, "Pedestrian Detection and Tracking Using a Mixture of View-Based Shape–Texture Models," *Intelligent Transportation Systems, IEEE Transactions on , vol.9, no.2, pp.333-343, June 2008.*

[33] B. Wu, R. Nevatia, “Detection and Tracking of Multiple, Partially Occluded Humans by Bayesian Combination of Edgelet based Part Detectors”, *International Journal of Computer Vision, vol. 72, no. 2, 2007.*

[34] U. Scheunert, H. Cramer, B. Fardi, G. Wanielik. "Multi sensor based tracking of pedestrians: a survey of suitable movement models," *Intelligent Vehicles Symposium, 2004 IEEE , vol., no., pp. 774- 778, 14-17 June 2004.*

[35] M. Kass, A. Witkin, D. Terzopoulos, "Snakes, Active contour models", *First International Conference on Computer Vision, pp. 259-268, IEEE, Computer Society Press, 1987.*

[36] ROS. `http://www.ros.org`

[37] Stack de ROS “PeopleExperimental”, `http://www.ros.org/wiki/people_experimental`

[38] Stack de ROS “PeopleExperimental”, código fuente, `https://code.ros.org/svn/wg-ros-pkg/branches/trunk_cturtle/stacks/people_experimental/`

[39] Random Trees, `http://www.stat.berkeley.edu/users/breiman/RandomForests/`

[40] Stack de ROS “iri_perception”, `http://www.ros.org/wiki/iri_perception`

[41] Stack de ROS “iri_navigation” `http://www.ros.org/wiki/iri_navigation`

[42] Stack de ROS “twoLevelMTTD_stack” `http://www.ros.org/wiki/twoLevelMTTD_stack`

[43] Paquete de ROS “head_follow_people” `http://www.ros.org/browse/details.php?name=head_follow_people`

[44] Paquete de ROS “people_detector_node” `http://www.ros.org/browse/details.php?name=people_detector_node`

[45] OpenCV. `http://opencv.willowgarage.com/wiki/`

[46] Documentación de OpenCV. `http://opencv.willowgarage.com/documentation/`

[47] G. Bradski y A. Kaehler, “Learning OpenCV”, Ed. O’Reilly. ISBN: 978-0-596-51613-0.

[48] R.A. Hamzah, R.A. Rahim, Z.M. Noh; “Sum of Absolute Differences algorithm in stereo correspondence problem for stereo matching in computer vision application,” *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on , vol.1, no., pp.652-657, 9-11 July 2010.*

[49] A. Geiger, M. Roser, R. Urtasun, “Efficient Large-Scale Stereo Matching”. *Asian Conference on Computer Vision, November 2010.*

[50] Librería “libelas” `http://www.rainsoft.de/software/libelas.html`

[51] Point Cloud Library. `http://pointclouds.org/`

[52] IDS imaging. `http://www.ids-imaging.es/`

[53] Webcam Minoru. `http://www.minoru3d.com/`

[54] Proyecto libv4l2cam y aplicación v4l2stereo, `http://code.google.com/p/libv4l2cam/`

[55] Daimler Stereo Pedestrian Detection Benchmark Dataset, `http://www.gavrila.net/Research/Pedestrian_Detection/Daimler_Pedestrian_Benchmark_D/Daimler_Stereo_Ped__Detection_/daimler_stereo_ped__detection_.html`

[56] C. Keller, M. Enzweiler, and D. M. Gavrila, “A New Benchmark for Stereo-based Pedestrian Detection," *Proc. of the IEEE Intelligent Vehicles Symposium, Baden-Baden, Germany, 2011.*