



## **Facultad de Ingeniería**

### **Departamento de Electrónica y Automática**

### **Temas Específicos de Control 1: Visión Artificial**

### ***DETECCIÓN DE PERSONAS PARA UN ROBOT TURTLEBOT USANDO OPENCV***

**Alumnos:**

***Pablo Aguado – 23724 – aguadopd@hotmail.com***

***Fabricio Emder – 23764 – elector102@gmail.com***

**Profesor:**

***Adrián Orellana - orellana@inaut.unsj.edu.ar***

**Abril – 2016**

# Índice

<b>1. Introducción</b>	<b>3</b>
<b>2. Objetivos</b>	<b>3</b>
<b>3. Con qué se trabajó</b>	<b>3</b>
3.1. ¿Qué es un TurtleBot? . . . . .	3
3.2. ¿Qué es un MICROSOFT KINECT? . . . . .	4
3.2.1. Funcionamiento . . . . .	5
3.3. ¿Qué es una RASPBERRY PI 2? . . . . .	6
3.3.1. Cámara . . . . .	6
3.4. ¿Qué es ROS? . . . . .	7
<b>4. Alternativas iniciales</b>	<b>7</b>
4.1. Desarrollo de un detector . . . . .	7
4.2. Detectores integrados en OPENCV . . . . .	8
4.3. OPENNI TRACKER de ROS . . . . .	9
4.4. SKELTRACK . . . . .	9
4.5. HUMAN TRACKER de ROS . . . . .	9
4.6. OPENPTTRACK . . . . .	9
<b>5. Algoritmos evaluados</b>	<b>10</b>
5.1. HOG + SVM . . . . .	10
5.1.1. Funcionamiento . . . . .	10
5.1.2. Entrenamiento . . . . .	11
5.1.3. Implementación . . . . .	12
5.2. Cascada de clasificadores . . . . .	13
5.2.1. Funcionamiento . . . . .	13
5.2.2. Entrenamiento . . . . .	14
5.2.3. <i>Haar-like features</i> . . . . .	14
5.2.4. <i>LBP features</i> . . . . .	14
5.2.5. Implementación . . . . .	15
5.3. Preprocesamiento . . . . .	17
5.3.1. Escalado inicial . . . . .	17
5.3.2. Transformación a escala de grises . . . . .	17
5.3.3. Ecualización de histograma . . . . .	17
5.3.4. Filtrado por convolución . . . . .	17
5.4. Postprocesamiento . . . . .	18
5.4.1. Estimación de la altura real . . . . .	18
<b>6. Pruebas</b>	<b>18</b>
6.1. Metodología . . . . .	18
6.2. Evaluación . . . . .	18
6.2.1. Igualdad . . . . .	18
6.2.2. Medidas básicas . . . . .	19
6.2.3. Medidas a optimizar . . . . .	19
6.3. Sets de imágenes . . . . .	20
6.4. Parámetros . . . . .	21
6.4.1. HOG + SVM . . . . .	21
6.4.2. LBP + Cascada . . . . .	21
<b>7. Implementación</b>	<b>22</b>
7.1. Conceptos generales de ROS . . . . .	22
7.2. Información desde el TURTLEBOT . . . . .	23
7.3. Información desde la RASPBERRY PI . . . . .	23
7.4. Información desde una PC . . . . .	24
7.5. Nodo de detección . . . . .	24

## Índice

<b>8. Resultados</b>	<b>24</b>
8.1. Pruebas y ajuste de parámetros . . . . .	24
8.1.1. HOG + SVM . . . . .	24
8.1.2. LBP + Cascada . . . . .	26
8.2. Pruebas con los mejores parámetros . . . . .	26
8.3. Pruebas sobre el set de validación . . . . .	27
8.4. Implementaciones en línea . . . . .	27
<b>9. Conclusiones</b>	<b>27</b>
9.1. Resultados . . . . .	27
9.2. Mejoras al trabajo realizado . . . . .	30
9.3. Mejores alternativas . . . . .	30
9.4. Comentarios . . . . .	31
<b>Referencias</b>	<b>32</b>
<b>A. Apéndice: Estructura de archivos</b>	<b>35</b>
<b>B. Apéndice: Nodos</b>	<b>36</b>

---

## 1. Introducción

### 1. Introducción

Este es el informe del trabajo integrador final de la materia TEMAS ESPECÍFICOS DE CONTROL I: VISIÓN ARTIFICIAL, dictada en la carrera Ingeniería Electrónica de la Universidad Nacional de San Juan. La consigna es llevar a cabo un proyecto que en el se apliquen todos los conocimientos adquiridos en la materia, asimilándolos con práctica y complementándolos con más conceptos.

Se planteó como objetivo principal del proyecto realizar la detección de personas desde un robot móvil con un grado aceptable de aciertos. Para ello se probaron la eficiencia y eficacia de distintos algoritmos que fueron implementados en OPENCV sobre un robot TURTLEBOT, equipado con una cámara RGB+D MICROSOFT KINECT y trabajando bajo el sistema ROS (*Robot Operative System*). El programa desarrollado fue luego adaptado para poder utilizarse en otras cámaras sin información de profundidad, como una *webcam* de una PC o una mini-computadora RASPBERRY PI 2 con cámara.

El informe se extiende sobre algunas alternativas de trabajo, conceptos de los detectores utilizados y también sobre ROS, ya que fue una herramienta de uso constante. Se abordan también las pruebas realizadas, su evaluación e interpretación, terminando con algunas conclusiones obtenidas y posibles mejoras.<sup>1</sup>

### 2. Objetivos

El detector de personas a implementar intenta satisfacer las siguientes especificaciones:

1. Funciona en línea.
2. Detecta personas adultas
  - a) en posición vertical, ya sea caminando o paradas;
  - b) a una distancia mayor a 1 metros y menor a 7 metros.<sup>2</sup>
3. Funciona bajo condiciones normales de iluminación, entendiendo como normal el nivel de iluminación presente en entornos cerrados donde personas circulen y/o realicen trabajos manuales — es decir, oficinas, fábricas, laboratorios, etc.
4. Funciona bajo ROS sobre un robot TURTLEBOT (versión 1).

### 3. Con qué se trabajó

El TURTLEBOT trabaja bajo el sistema ROS (*Robot Operating System*), por lo cual la implementación final deberá ser compatible con ROS. Durante el desarrollo de este trabajo también se probó el sistema utilizando *webcams* de computadoras personales y también una mini-computadora RASPBERRY PI con una cámara *SCI* externa.

#### 3.1. ¿Qué es un TurtleBot?

TURTLEBOT es un kit de robótica de bajo costo construido a partir de electrónica de consumo, por lo que se puede replicar de forma sencilla. Es capaz de recorrer interiores con una decente autonomía de batería y con funciones de visión tridimensional. Se le pueden sumar accesorios para añadir funcionalidad.

El TURTLEBOT está compuesto por:

- Base móvil iROBOT CREATE. [1]
- Netbook ASUS 1215N. [2]
- Controlador MICROSOFT KINECT. [3] Está montado a 30 cm del suelo.

<sup>1</sup>El código fuente de todos los programas, los sets de imágenes utilizados y los resultados obtenidos, pueden descargarse desde <https://github.com/GERUNSJ/deteccion-de-personas-con-turtlebot-y-opencv-1>

<sup>2</sup>Originalmente el rango planteado fue de 1 a 5 metros, pero fue extendido en vista de las posibilidades de los detectores.

### 3. Con qué se trabajó

Figura 1: TURTLEBOT 1



- Estructura de madera y aluminio para llevar la *netbook*, el controlador KINECT y accesorios que se agreguen.

#### 3.2. ¿Qué es un Microsoft Kinect?

Es un controlador de juego y entrenamiento desarrollado por MICROSOFT (*en adelante será referido como el [controlador] KINECT o como la [cámara] KINECT*). [4, 3, 5] El dispositivo permite al usuario de una consola de videojuegos XBOX o una PC interactuar con juegos o programas sin tener contacto físico con un controlador. Algunos de los componentes principales de la versión utilizada, KINECT FOR XBOX 360 o KINECT V1:

- Cámara RGB

**Formato** VGA (640 \* 480 px)

**Resolucion** 8 bits

**Tasa** 30 cuadros por segundo

**Campo de visión** 62,7 ° horizontal

**Distancia focal** 525 píxeles

- Cámara infrarroja

**Formato** VGA (640 \* 480 px)

**Resolucion** 11 bits

**Tasa** 30 cuadros por segundo

**Campo de visión** 57,8 ° horizontal

**Distancia focal** 580 píxeles

- Proyector láser infrarrojo con una matriz de difracción

Es importante notar que la KINECT está diseñada para trabajar en interiores. La presencia de fuentes de luz infrarroja significa una disminución del rango útil de detección e incluso puede llegar a hacerla totalmente imposible.

### 3. Con qué se trabajó

Figura 2: Sensor Kinect

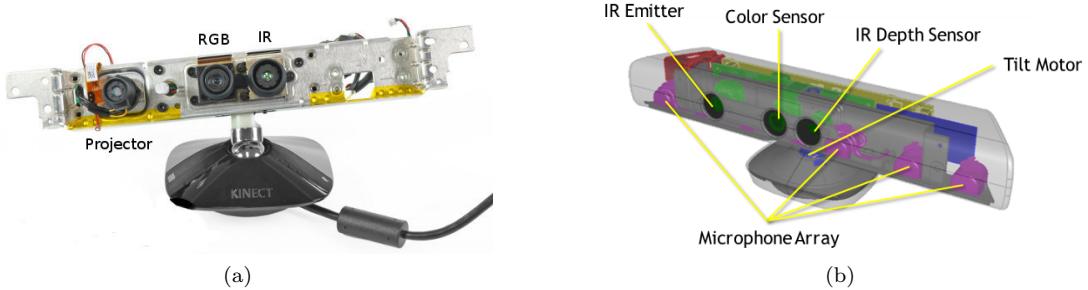


Figura 3: Patrón de puntos proyectado — Luz estructurada.

Obsérvese que la imagen está dividida como una matriz de  $3 \times 3$  regiones, y cada una tiene un punto brillante en su centro. Se cree que esto sirve para una orientación rápida del sistema.



#### 3.2.1. Funcionamiento

Para la obtención de la imagen de profundidad o mapa de profundidad no se utiliza la información de la cámara RGB, si no que un proyector láser infrarrojo con una matriz de difracción constante proyecta un patrón de puntos que luego es captado por la cámara infrarroja. Ver Figura 3.

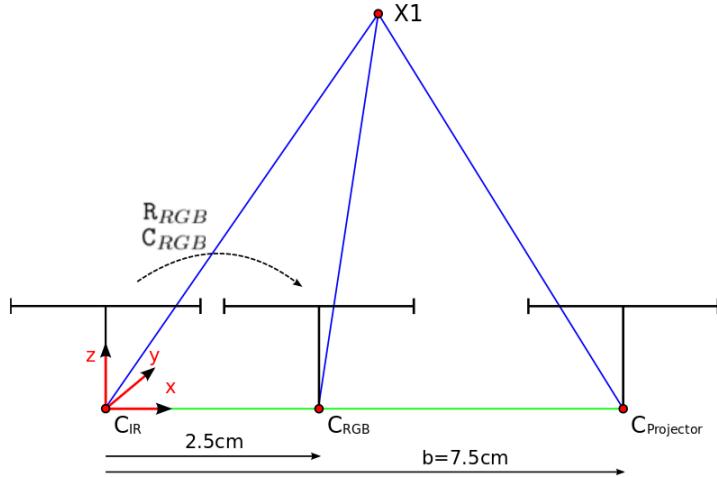
Se conoce a priori cuál es la visualización del patrón ante una superficie plana a una distancia determinada. Para conocer la profundidad de los objetos en la imagen se procede a correlacionar cada porción del patrón observado contra todo el patrón conocido; al encontrar la coincidencia se mide el desfasaje o *disparidad*, y a partir de esto y por triangulación se calcula la distancia real de la porción en cuestión.<sup>3</sup>[6, 7, 8, 9, 10, 11, 5, 12]

La imagen de profundidad es una imagen de un solo canal cuyos valores corresponden a distancias (en mm desde la cámara) y es transmitida como una imagen de 16 bits, pero sólo los 11 bits menos significativos son útiles. La distancia de trabajo efectiva del sensor KINECT v1 es desde 0,8 m a 3,5 m, con un error de 1 cm en profundidad y de hasta 3 mm en las mediciones de ancho y alto.

<sup>3</sup>El modo real de funcionamiento no ha sido explícitamente publicado, y mucha de esta información está basada en ingeniería inversa, análisis de expertos y patentes de la empresa PRIMESENSE, desarrolladora del *System On a Chip* PS1080, encargado de todo el procesamiento de imagen y audio de la KINECT. Un interesante análisis en español de la KINECT se encuentra en el Apéndice A de [5].

### 3. Con qué se trabajó

Figura 4: Modelo geométrico de las cámaras de la KINECT



### 3.3. ¿Qué es una Raspberry PI 2?

RASPERRY PI es una serie de computadoras de placa única (*Single Board Computer, SBC*), las cuales incluyen todo en una misma placa y son del tamaño de una tarjeta de crédito. En este proyecto se usó la RASPERRY PI 2 versión B [13], cuyas características principales son:

- Procesador Quad core ARM Cortex-A7 de 900mhz
- 1GB de memoria RAM
- 17 GPIO - salida HDMI - 4 puertos USB - Ethernet
- Puerto SCI para cámara.

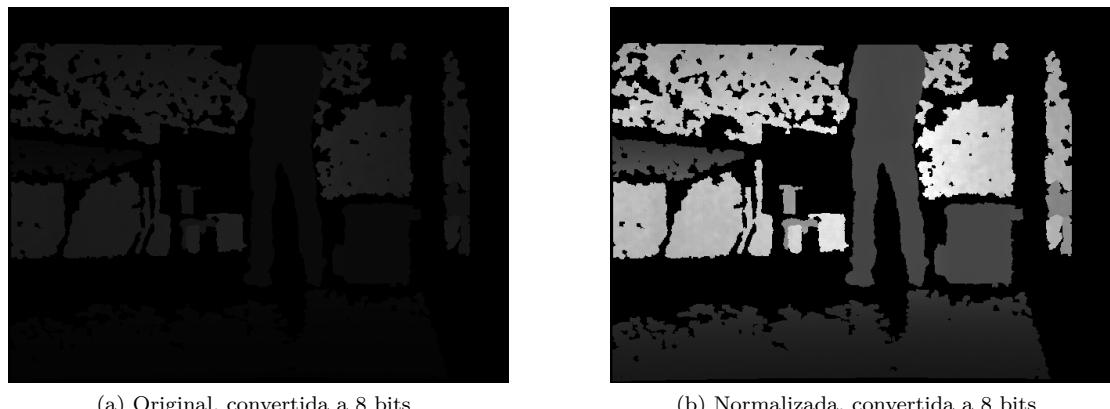
#### 3.3.1. Cámara

La cámara conectada a la RASPERRY PI tiene las siguientes especificaciones:

**Sensor** OMNIVISION OV5647

**Formatos** QVGA (320 \* 240 px) @ 120 FPS, VGA (640 \* 480 px) @ 90 FPS, 720P @ 60 FPS, 960P @ 45 FPS, 1080 @ 30 FPS, QSXGA (2592 \* 1944 px) @ 15 FPS

Figura 5: Imágenes de profundidad de la KINECT



(a) Original, convertida a 8 bits

(b) Normalizada, convertida a 8 bits

#### 4. Alternativas iniciales

**Resolucion** 8-/10 bits

**Lente** LS-2716CS

**Campo de visión** 81 ° horizontal

**Distancia focal** 4 mm

**Interfaz** SCI

Figura 6: RASPBERRY PI 2 Y CÁMARA



### 3.4. ¿Qué es ROS?

A pesar de su nombre (*Robot Operating System*, Sistema Operativo Robótico o Sistema Operativo de Robots), no se puede decir que ROS [14, 15] sea un sistema operativo (SO) propiamente dicho, ya que la funcionalidad de este depende de un SO base, como pueden ser : OS X, ANDROID, WINDOWS y distintas distribuciones de LINUX/UNIX — la más difundida y soportada es UBUNTU. Se puede entonces definir a ROS como un *framework* (entorno de trabajo) flexible para el desarrollo de software para robots, que provee funcionalidades de un SO en un *cluster* heterogéneo. Está conformado por una colección de herramientas, bibliotecas y convenciones que tienden a simplificar la creación de funciones robóticas complejas. ROS es de código abierto al igual que OPENCV, lo cual, en conjunto con sus herramientas y estandarización, ha permitido un gran crecimiento de la comunidad involucrada en robótica, facilitando y motivando el desarrollo colaborativo.

## 4. Alternativas iniciales

En base a las herramientas disponibles y a los objetivos buscados, las siguientes alternativas iniciales fueron contempladas:

### 4.1. Desarrollo de un detector

Desarrollo desde cero de un detector utilizando funciones básicas de OPENCV (*Open Computer Vision library*)[16]. Este enfoque, iniciado y luego descartado, permitió un mejor entendimiento del problema, a través de la búsqueda creativa de patrones y características de las imágenes que aportaran información útil para distinguir a una persona — e incluso implicó explorar cuál es la definición de persona, desde un punto de vista visual. El algoritmo desarrollado consistió<sup>4</sup> en:

<sup>4</sup>Sólo se trabajó con la imagen de profundidad.

#### 4. Alternativas iniciales

1. Segmentación según el histograma de intensidad de la imagen de profundidad.
  - a) Generación del histograma.
  - b) Suavizado del histograma con varias etapas de filtros promediadores. Así se elimina ruido y se agrupan los segmentos que están cercanos en el histograma y probablemente corresponden al mismo objeto.
  - c) Búsqueda de máximos y mínimos locales del histograma.
  - d) Segmentación del histograma. Cada segmento corresponde a los valores de intensidad que se encuentran entre dos mínimos consecutivos.
  - e) Segmentación de la imagen según los segmentos de histograma, a través de *look-up tables*.
  - f) Mayor segmentación de cada segmento, separando los distintos objetos que se encuentran a la misma profundidad.
    - 1) Operaciones morfológicas: etapas de apertura y cierre para eliminar bloques pequeños y llenar “agujeros”.
    - 2) Búsqueda de contornos y segmentación por contornos.
2. Filtrado por relación de aspecto.
  - a) Creación de rectángulo contenedor de cada contorno.
  - b) Cálculo de relación de aspecto y eliminación si no es similar al de una persona de pie.
  - c) Filtrado según la altura real estimada.
    - 1) Estimamos la altura real del objeto a partir de la distancia respecto de la cámara, la altura en píxeles del objeto y la distancia focal de la cámara. Ver [5.4.1](#).
    - 2) Eliminación si es mayor o menor a límites de altura prefijados.
3. Filtrado según la dispersión del gradiente.
  - a) Cálculo de gradientes horizontales con *kernel* simple  $[-1, 0, 1]$ , en cada zona de interés.
  - b) Separación en gradientes positivos y negativos.
  - c) Cálculo de la dispersión de los valores de gradiente.
  - d) Filtrado bajo la suposición de que dispersiones pequeñas implican superficies con caras planas (eliminando así muchas mesas y sillas), mientras que dispersiones grandes corresponden a una, cuya superficie tiene muchas irregularidades.

*El proceso de segmentación y filtrado fue satisfactorio en el corto rango en que la persona se veía completa y se veía en la imagen de profundidad — aproximadamente desde los 3 a los 4 metros desde el robot. Además de este limitado rango, las estrategias de filtrado no fueron capaces de eliminar grandes porciones de paredes y objetos de contornos similares a los de una persona. El filtrado por dispersión de gradiente fue probado satisfactoriamente en MATLAB pero no implementado en OPENCV; las pocas pruebas realizadas sugieren que la idea es válida siempre y cuando haya suficiente información de profundidad. El desarrollo no fue continuado debido a los resultados mucho más prometedores de parte de los detectores en cascada.*

## 4.2. Detectores integrados en OpenCV

Utilización de un detector integrado en OPENCV — sin aprovechar la información existente en las imágenes de profundidad. La biblioteca OpenCV incluye, dentro del módulo `objdetect`, a los siguientes detectores:

**Detector en cascada** Cascada de clasificadores de complejidad en incremento, utilizando características (*features*) tipo Haar o patrones binarios locales (*Linear Binary Patterns, LBP*). [17, 18, 19]

#### 4. Alternativas iniciales

**Detector HOG** Clasificador tipo máquina de vectores soporte (*Support Vector Machine, SVM*), utilizando como vector de características a un histograma de gradientes orientados (*Histogram of Oriented Gradients, HOG*). [20, 21]<sup>5</sup>

**Latent SVM** Detector basado en modelos de partes deformables; usa *HOG* para detectar las partes. Este sistema fue desecharo porque es muy lento<sup>6</sup> y tiene poca precisión<sup>7</sup>, comparado con los anteriormente nombrados. [22, 23]

*Esta es la opción elegida para mayor desarrollo. Implicó profundizar el conocimiento en la biblioteca OPENCV, que fue la utilizada durante el cursado de la materia, y además conocer conceptos de detectores estándar de personas (u otros objetos), como lo son los detectores en cascada y los detectores HOG. Como se verá en las secciones siguientes, se añadió además información de profundidad para mejorar la precisión de los detectores, pero el corto rango útil de esta perjudica a los resultados en lugar de mejorarlo.*

#### 4.3. OpenNI Tracker de ROS

Utilizar el paquete OPENNI TRACKER de ROS [24], que publica (ver conceptos de ROS en 7.1) las coordenadas de las extremidades de las personas que detecta. Utiliza OPENNI<sup>8</sup> y el módulo NITE<sup>9</sup> para hacer la detección, segmentación y seguimiento. El proyecto fue adquirido por APPLE y discontinuado; la documentación remanente es escasa. Las versiones de NITE existentes al comenzar este trabajo requerían que los usuarios se colocaran en una posición predeterminada para detectarlos y comenzar a seguirlos. Versiones posteriores permitían usar un archivo de configuración pre-guardado para el usuario, mientras que las últimas son capaces de iniciar el seguimiento desde cualquier posición.

*Esta posibilidad fue evaluada y desechada al comenzar este trabajo, ya que requería la cooperación inicial de los sujetos en escena para poder detectarlos. Si bien versiones que aparecieron posteriormente prescinden de esta cooperación, el área de operación efectiva es muy limitada por las limitaciones propias de la información de profundidad de la KINECT.*

#### 4.4. Skeltrack

La biblioteca de código abierto SKELTRACK [25]. Está basada en [26] y utiliza información heurística para encontrar a la persona. Si bien la detección es muy rápida, tanto el trabajo original como la implementación en SKELTRACK están limitadas a un usuario en cámara y necesitan que el usuario sea el único objeto en escena.

*Esta opción fue desechada debido a las limitaciones que imponía.*

#### 4.5. Human Tracker de ROS

El paquete HUMAN TRACKER [27, 28] para ROS, disponible en los repositorios de ROS INDUSTRIAL. Consiste en una cascada de detectores de complejidad ascendente, y se vale tanto de la información de color como de la de profundidad.

*Tiene poca documentación y no pudo hacerse funcionar en ROS FUERTE con el controlador KINECT. Por estas causas, la opción fue descartada en favor de los detectores de OPENCV.*

#### 4.6. OpenPTrack

El proyecto OPENPTRACK [29, 27, 30, 31], que permite la detección y seguimiento de múltiples personas utilizando arreglos de cámaras. Algunos de sus desarrolladores formaron parte del equipo de HUMAN TRACKER, por lo que los algoritmos utilizados son similares.

<sup>5</sup>[20] se refiere a la versión del algoritmo para *GPUs*. La versión para *CPUs* no está documentada pero se utiliza de la misma manera que la de *GPUs*.

<sup>6</sup>Aproximadamente 1 segundo para una imagen de 640 \* 480 px.

<sup>7</sup>Ver el trabajo original de Felzenszwab. En nuestras pruebas con OPENCV, el número de falsos positivos lo tornaba inútil para el objetivo buscado.

<sup>8</sup>*Open Natural Interface*, plataforma que facilita el desarrollo de interfaces naturales, que son aquellas basadas en gestos del usuario. Estandariza formatos y forma de acceder a los dispositivos.

<sup>9</sup>Plugin para OPENNI que permite el seguimiento de esqueleto e interpretación de gestos manuales. De código cerrado, desarrollado por PrimeSense.

## 5. Algoritmos evaluados

Se pudo hacer funcionar en la instalación de ROS INDIGO existente, pero ocasionalmente se cerraba con errores. Además, el sistema está diseñado para arreglos de cámaras en altura y orientadas hacia abajo — condición no cumplida por los 30 cm de las cámaras en el TURTLEBOT — por lo que las detecciones fluctuaban al situar al robot en el suelo (que es su condición habitual).

## 5. Algoritmos evaluados

A continuación se presentan (conceptual y superficialmente) los detectores que tuvieron mejores resultados y se enuncian los parámetros de cada uno. También mencionamos algunas estrategias de pre y post procesamiento que fueron probadas.

### 5.1. HOG + SVM

#### 5.1.1. Funcionamiento

El descriptor visual o vector de características (*feature vector*) es un histograma de gradientes orientados. A grandes rasgos, su construcción (según el trabajo original de Dalal y Triggs) consiste en : [21, 20, 32]

1. Se divide una imagen en pequeños bloques.
2. Para cada bloque:
  - a) Se calculan los gradientes (magnitud y ángulo) en cada píxel;
    - 1) El gradiente horizontal se puede calcular (hay diversas definiciones) como la resta entre el valor a la derecha del píxel y el que tiene a su izquierda (o al revés, lo importante es mantener el criterio). Como ejemplo, ver Figuras 7 y 8.
    - 2) Análogamente para el gradiente vertical, utilizando los píxeles que están arriba y debajo del píxel de interés.
    - 3) A partir de lo anterior se obtiene el vector gradiente. Para el ejemplo en cuestión (ver Figura 9):
$$dx = 94 - 56 = 38$$
$$dy = 93 - 55 = 38$$
$$\text{Gradiente} = \sqrt{38^2 + 38^2} \approx 53,74$$
$$\text{Ángulo} = \arctan(38/38) = 45^\circ$$

- b) Se agrupan las magnitudes en un histograma de 9 clases o *bins*, correspondientes a un intervalo de orientaciones o ángulos (ver Figura 10);
- c) Se normalizan los valores para hacer el descriptor más robusto ante cambios de contraste.

3. Se agrupan los resultados de cada bloque en un único vector descriptor.

Luego este vector ingresa a una Máquina de Vector Soporte o Máquina de Soporte Vectorial [33] (*Support Vector Machine, SVM*), que es un clasificador binario — su salida es la clase “persona” o la clase “no persona”.

## 5. Algoritmos evaluados

Figura 7: Cálculo del gradiente horizontal y vertical para un píxel específico.

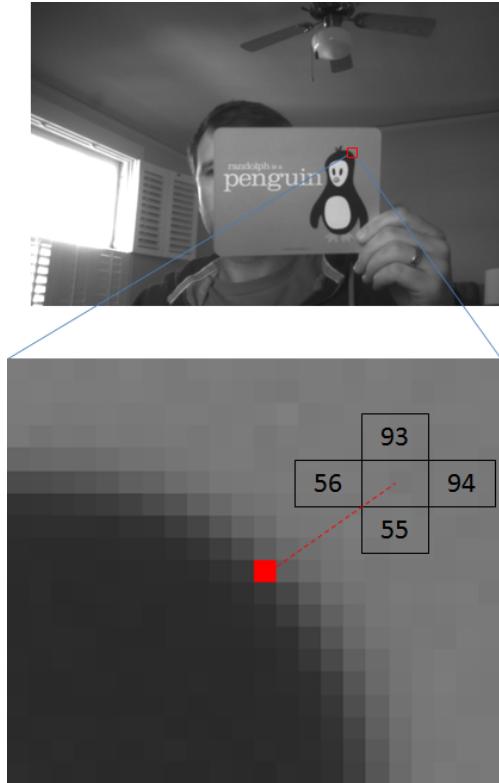
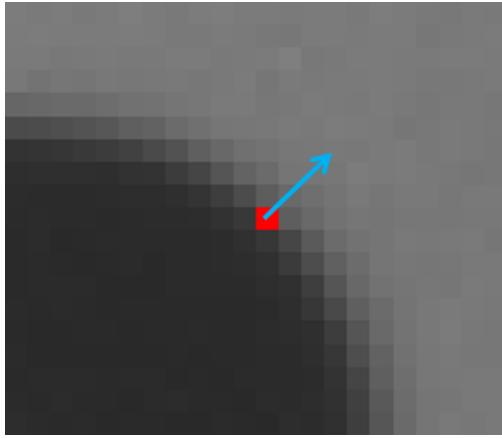


Figura 9: Vector gradiente resultante



### 5.1.2. Entrenamiento

La *SVM* se entrena previamente con una gran cantidad de muestras positivas y negativas etiquetadas (se indica si cada imagen tiene una persona o no)<sup>10</sup>, a las cuales se les calcula el vector descriptor anteriormente mencionado. El subespacio vectorial obtenido se utiliza para optimizar un hiperplano de decisión que permita la (mejor) separación entre las clases.

Para buscar personas se recorre la imagen objetivo con una ventana patrón — dicho de otra manera, se hace deslizar la ventana sobre la imagen objetivo. Nótese que el sistema se diseña y entrena para un tamaño fijo de imagen. Para hacerlo multiescala se recurre a achicar la imagen en donde se buscan los objetos, o a escalar la ventana patrón adonde se intenta detectar la persona.

Figura 8: Imágenes de las componentes del vector gradiente, normalizadas a 256 valores

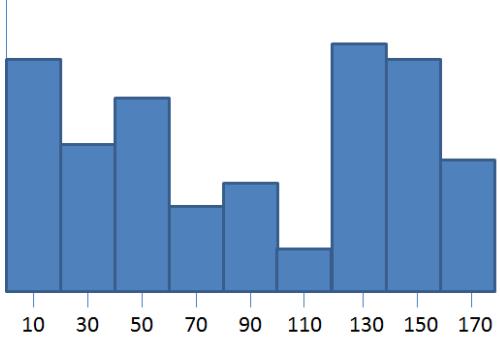


(a) Gradiente horizontal



(b) Gradiente vertical

Figura 10: HOG: Histograma de 9 clases



<sup>10</sup>A esto se le denomina entrenamiento supervisado.

## 5. Algoritmos evaluados

### 5.1.3. Implementación

El detector *HOG* de OPENCV está en el módulo `objdetect`. [20]<sup>11</sup> Su constructor es:

```
HOGDescriptor( cv::Size winSize ,  
                cv::Size blockSize ,  
                cv::Size blockStride ,  
                cv::Size _cellSize ,  
                int nbins ,  
                int derivAperture=1,  
                double winSigma=-1,  
                int histogramNormType=HOGDescriptor::L2Hys ,  
                double L2HysThreshold=0.2 ,  
                bool gammaCorrection=false ,  
                int nlevels=HOGDescriptor::DEFAULT_NLEVELS );
```

Estos argumentos corresponden a parámetros internos del detector. La mayoría no se puede cambiar en la versión 2.4.x ya que solo se ha implementado la versión original de Dalal y Triggs. Sí se puede elegir el parámetro `winSize`, que corresponde al tamaño de la ventana patrón, y debe coincidir con el tamaño del modelo usado. OPENCV trae dos modelos incrustados en la librería:

- `DEFAULTPEOPLEDETECTOR`, entrenado con imágenes de  $64 * 128\text{ px}$ . Puede usarse con `HOGDescriptor::getDefaultPeopleDetector`.
- `DAIMLERPEOPLEDETECTOR`, entrenado con imágenes de  $48 * 96\text{ px}$ . Puede usarse con `HOGDescriptor::getDaimlerPeopleDetector`.[34]

Para detectar en una imagen, se utiliza:

```
void detectMultiScale( const cv::Mat& img ,  
                      std::vector<cv::Rect>& foundLocations ,  
                      double hitThreshold = 0 ,  
                      cv::Size winStride = cv::Size() ,  
                      cv::Size padding = cv::Size() ,  
                      double scale=1.05 ,  
                      double finalThreshold=2.0 ) ;
```

Los argumentos son:

**img** Imagen objetivo.

**foundLocations** Vector de `cv::Rect` con las personas detectadas.

**hitThreshold** Umbral de distancia entre los vectores de *features* y el (hiper)plano de clasificación — si la distancia es menor al umbral, la muestra es rechazada. Usualmente es 0, y debe estar especificado en los coeficientes del detector, como el último coeficiente libre. En caso de estar omitido ahí, se puede especificar en este argumento. *Es usado como parámetro variable en este trabajo.*

**winStride** Avance de la ventana de evaluación. Debe ser un múltiplo del parámetro `blockStride` del detector. *En este trabajo es `Size(8,8)` en todos los casos, ya que es el menor paso posible.*

**padding** Píxeles de relleno en los bordes de la imagen. *En este trabajo es `Size(0,0)`.*

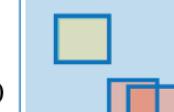
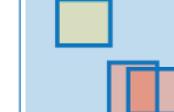
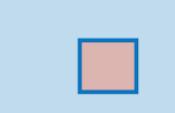
**scale** Coeficiente de incremento de la ventana de detección. *Es usado como parámetro variable en este trabajo.*

---

<sup>11</sup>Ver nota 5.

## 5. Algoritmos evaluados

Figura 11: Funcionamiento de `groupRectangles`

<i>MergeThreshold</i>	Detections	Returned Bounding Boxes
0		
1		
2		
3		

**finalThreshold** Umbral de cantidad para agrupación de detecciones. Hace un llamado a la función `groupRectangles`<sup>12</sup>, que agrupa los rectángulos de un vector según su cercanía espacial y similaridad de lados. La desventaja es que usa un factor `eps` fijo de 0.2 — la alternativa es usar `finalThreshold = 0` y luego usar `groupRectangles` con los valores deseados. `finalThreshold` es usado como parámetro variable en este trabajo.

## 5.2. Cascada de clasificadores

### 5.2.1. Funcionamiento

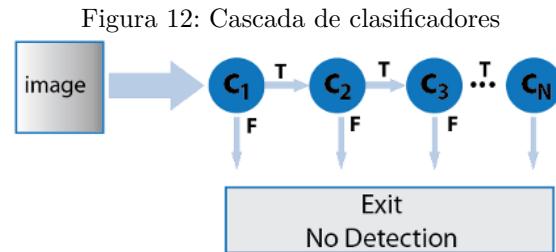
En inglés, *cascade of boosted classifiers*, aunque también se conoce como algoritmo de Viola-Jones, en honor a los autores de la idea[18]. Consiste en una cascada o serie de clasificadores binarios cuya complejidad va en incremento — en su trabajo los clasificadores son árboles de decisión de al menos 2 hojas. El sistema es efectivo por el conjunto completo de clasificadores, que ha sido entrenado previamente para detectar un objeto específico; y es eficiente porque los objetos que son rechazados por alguna etapa no son evaluados por las etapas subsiguientes. A diferencia del clasificador *SVM*, en este caso no se calcula todo el vector descriptor (o vector de características, *feature vector*) antes de ingresarlo al clasificador, si no que las componentes del vector se van generando sucesivamente — cada componente es la entrada a un clasificador — hasta terminar el vector o hasta que la respuesta de un clasificador sea negativa. OPENCV soporta *features* del tipo *Haar* (*Haar-like*) o *LBP*, según se comenta en las siguientes secciones.[17]

Al igual que el detector *HOG*, detecta personas a través de una ventana deslizante que recorre la imagen objetivo. Para la detección multi-escala se incrementa gradualmente el tamaño de la ventana deslizante, o se decremente gradualmente el tamaño de la imagen objetivo, según la implementación. El detector está entrenado con imágenes de un tamaño fijo, generalmente muy pequeño.

<sup>12</sup>`cv::groupRectangles(vector<Rect>& rectList, int groupThreshold, double eps=0.2)` — Agrupa los rectángulos cuya relación entre lados sea menor a `eps` y crea un rectángulo promedio de los agrupados. Luego de la agrupación o *clustering*, solo retiene aquellos grupos con más de `groupThreshold` rectángulos. Un valor de 0 en `groupThreshold` implica no hacer nada. Ver Figura 11.

## 5. Algoritmos evaluados

Nótese que cada *feature* está asociada a una posición fija dentro de una imagen de tamaño fijo. Entonces, cada *feature* o descriptor específico está determinado por su valor y por información implícita no variable: su posición, su escala (no confundir con el escalado de toda la imagen o clasificador, para la detección) y su figura — estos dos últimos solo para *features* Haar. Por ejemplo, una característica podría ser del tipo 1(a) de la Figura 13, de un tamaño total de  $20 * 20\text{ px}$  y con su esquina superior en las coordenadas (2,12) de la imagen objetivo. Otra característica podría ser del mismo tipo, pero de un tamaño de  $38 * 38\text{ px}$  y situada en (43,33). El valor de cada una ingresa a un clasificador binario distinto, y no necesariamente ambas se calcularán en caso de una imagen que no contiene el objeto buscado.



### 5.2.2. Entrenamiento

El entrenamiento del clasificador es supervisado: se ingresa una gran cantidad de imágenes positivas y negativas, todas etiquetadas (clase “persona” o clase “no persona”); luego el sistema se optimiza iterativamente para reducir la diferencia entre las salidas del clasificador y las etiquetas.

Se evalúan las características o *features* en todas las posiciones posibles, para todas las imágenes; cada característica es la entrada a un clasificador binario distinto. Estos son llamados clasificadores *débiles*, porque individualmente no permiten una buena clasificación, y muchas veces apenas pueden superar a una decisión aleatoria. Sin embargo, en conjunto forman un clasificador *fuerte*.

Durante las iteraciones de entrenamiento, cada *feature* (y en cada posición) se va pesando según su capacidad de detección y se priorizan las imágenes que han sido incorrectamente clasificadas. Al finalizar el entrenamiento, solamente las mejores características se retienen, pues son las que mejor permiten la identificación del objeto — además se reduce la dimensionalidad del clasificador y se hace más general o menos sobreentrenado. El detector de caras presentado por Viola y Jones tiene más de 160.000 *features* posibles, pero sólo se utilizan alrededor de 6,000 — afirman que con 200 ya obtienen una precisión del 95 %. [18]

### 5.2.3. Haar-like features

OPENCV utiliza los algoritmos y los descriptores propuestos en el trabajo de Lienhart y otros [35], que presentan una mejora al trabajo de Viola y Jones. Los descriptores se denominan simil-Haar o *Haar-like*, debido a su similitud con ondeletas Haar; por simplicidad también son llamados *features* Haar.

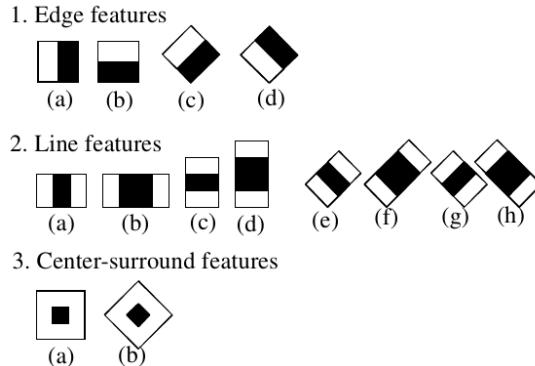
Consisten en áreas rectangulares agrupadas en conjuntos simples. La Figura 13 muestra los descriptores básicos propuestos en [35] — se escalan a diferentes tamaños para formar nuevos *features*. Según su forma, los descriptores indicarán la presencia de líneas, bordes y puntos. En cada caso, la suma de los valores de los píxeles bajo los rectángulos blancos es restada de la suma de los píxeles bajo los rectángulos negros. Pueden calcularse de forma muy eficiente a través del uso de las llamadas imágenes integrales (*integral images*) o tablas de áreas sumadas.

### 5.2.4. LBP features

Los patrones binarios locales [19] (*Local Binary Patterns, LBP*) son descriptores originalmente utilizados para discriminar texturas. Naturalmente robustos ante cambios de iluminación, fueron mejorándose para ser invariantes ante rotaciones y otros factores, e implementados de diferentes formas para hacerlos útiles en la detección de objetos, reconocimiento facial, análisis temporal de imágenes, entre otras aplicaciones.

## 5. Algoritmos evaluados

Figura 13: Prototipos de *features Haar-like* usados en OPENCV

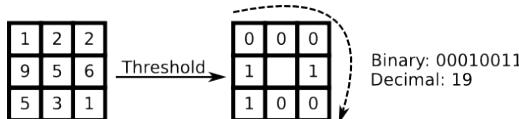


Básicamente, se analiza cada píxel de una imagen o región de una imagen, de la siguiente manera: [36, 37, 38]

1. Se evalúa la diferencia entre cada uno de los píxeles circundantes (según la implementación, inmediatamente adyacentes, a un radio mayor o incluso interpolando valores) y el píxel central del análisis.
2. A los píxeles cuyo valor es mayor o igual al píxel central se les asigna un valor de 1, mientras que aquellos cuyo valor sea menor reciben un valor de 0. Nótese que la diferencia se mantendrá aunque cambie el contraste o el brillo.
3. Los valores obtenidos se concatenan en una única palabra binaria que es la que representa al píxel — se reemplazó su valor original por uno que codifica la información propia y la de su entorno.

Otras implementaciones analizan sólo conjuntos no intersectados de píxeles; es decir, cada píxel sólo pertenece a un vecindario. También hay versiones que incluyen información multiescala, incluyendo en el vector descriptor los valores correspondientes a más de un radio de análisis.

Figura 14: Principio de funcionamiento de los patrones binarios locales



Luego se puede computar el histograma de la “imagen” obtenida y así codificar todo en un solo vector descriptor. Para no perder información localizada se recurre a dividir en bloques la imagen objetivo, y se calcula el histograma de cada bloque; cada histograma es la entrada a un clasificador de la cascada.<sup>13</sup> En otros casos (como el reconocimiento facial) y para otros clasificadores, todos los histogramas se concatenan en un mismo vector.

Empíricamente se ha encontrado (en el análisis de texturas) que la mayoría de la información útil se encuentra en los **patrones uniformes**, que son aquellos que tienen menos de 3 transiciones entre 1s y 0s: 00000000, 001110000, 10000000, 11111101, ... Agrupando todas las transiciones que son equivalentes (iguales si se rotan un número de veces) como si fuesen el mismo patrón, se logra una reducción del descriptor y además invarianza rotacional.

### 5.2.5. Implementación

El detector en cascada está en el módulo `objdetect` de OPENCV. [17] Su constructor:

```
CascadeClassifier();
```

<sup>13</sup>Estimamos que es de esta manera en OPENCV, pero no podemos afirmarlo. *LBP* no está documentado en la sección de detectores en cascada.

## 5. Algoritmos evaluados

Se puede cargar un modelo de cascada entrenado con el método `load`:

```
detector.load( direccion_a_cascada );
```

El archivo cargado determinará si se utilizan *features LBP* o *Haar-like*. En este trabajo se probaron los siguientes detectores:

- HAARCASCADE\_FULLBODY, entrenado con imágenes de  $14 * 28\text{ px}$ , de cuerpo completo.[39] Incluido en el código fuente de OPENCV (`opencv/data/haarcascades/ haarcascade_fullbody.xml`).
- HAARCASCADE\_LOWERBODY, entrenado con imágenes de  $19 * 23\text{ px}$ , de la parte inferior del cuerpo.[39] Incluido en el código fuente de OPENCV (`opencv/data/haarcascades/ haarcascade_lowerbody.xml`).
- HAARCASCADE\_MCS\_UPPERBODY, entrenado con imágenes de  $14 * 20\text{ px}$ , de cabeza y hombros. [39][40] Incluido en el código fuente de OPENCV (`opencv/data/haarcascades/ haarcascade_mcs_upperbody.xml`).
- HAARCASCADE\_UPPERBODY, entrenado con imágenes de  $18 * 20\text{ px}$ .[39] Incluido en el código fuente de OPENCV (`opencv/data/haarcascades/ haarcascade_upperbody.xml`).
- VISION-ARY.NET, entrenada con imágenes de  $26 * 74\text{ px}$ . [41]

La detección multi-escala en una imagen se realiza con:

```
void detectMultiScale( const Mat& image ,  
                      vector<Rect>& objects ,  
                      double scaleFactor=1.1 ,  
                      int minNeighbors=3 ,  
                      int flags=0 ,  
                      Size minSize=Size() ,  
                      Size maxSize=Size() );
```

Los argumentos de la función son:

**image** Imagen objetivo.

**objects** Vector de `cv::Rect` con las personas detectadas.

**scaleFactor** Coeficiente de incremento de la ventana de detección o de la imagen objetivo. *Es usado como parámetro variable en este trabajo.*

**minNeighbors** Umbral de cantidad para agrupar detecciones. Ver `finalThreshold` en 5.1.3. *Es usado como parámetro variable en este trabajo.*

**flags** Opciones sólo documentadas en versiones antiguas (<2.0) de OPENCV. [42] *En este trabajo no se utilizarán, y por tanto flags = 0.*

`cv::CASCADE_DO_CANNY_PRUNING=1` — Si está activada, se utiliza un detector de bordes tipo Canny para rechazar regiones de imagen que tengan demasiados o muy pocos bordes y por lo tanto sea poco factible que contengan el objeto buscado. Los umbrales de esta opción están ajustados a la detección de caras. Sirve para acelerar el procesamiento.

`cv::CASCADE_SCALE_IMAGE=2` — Se escala la imagen en vez de agrandar la ventana patrón. No puede usarse simultáneamente con ninguna de las otras opciones.

`cv::CASCADE_FIND_BIGGEST_OBJECT=4` — Sólo devuelve el objeto más grande detectado, o ninguno si no hubo detecciones.

`cv::CASCADE_DO_ROUGH_SEARCH=8` — Solo puede usarse si `CASCADE_FIND_BIGGEST_OBJECT` está activada y `minNeighbors` es mayor a 0. Cuando está activada, la función realiza una búsqueda tosca, aproximada: no se buscarán candidatos de menor tamaño una vez que se encontró el objeto (con suficientes detecciones vecinas) para la escala vigente.

## 5. Algoritmos evaluados

**minSize cv::Size**, tamaño mínimo de las detecciones. Acota la búsqueda junto a **maxSize**. Es usado como parámetro variable en este trabajo, proporcional a la escala inicial.

**maxSize cv::Size**, tamaño máximo de las detecciones. Es usado como parámetro variable en este trabajo, proporcional a la escala inicial.

### 5.3. Preprocesamiento

#### 5.3.1. Escalado inicial

Se reduce el tamaño de la imagen por un factor arbitrario. Esta función es especialmente importante en cámaras de gran resolución, donde se puede acortar de manera importante el tiempo de procesamiento sin que esto tenga un impacto significativo en los resultados.

#### 5.3.2. Transformación a escala de grises

Es especialmente útil para hacer que los algoritmos sean más ágiles en procesar. También es utilizada como etapa obligatoria previa a la ecualización de histograma en OPENCV 2.4.x.

#### 5.3.3. Ecualización de histograma

Normalización de la imagen de forma tal que su histograma se reparta uniformemente en todo el rango de grises. Si bien en OPENCV 2.4.x la ecualización solo puede realizarse sobre imágenes de un único canal, podría aplicarse a imágenes en color si se trabaja en modelos de color que separan la intensidad de la información de color, como HSV — en este caso se ecualiza el histograma del canal de intensidad o brillo.

#### 5.3.4. Filtrado por convolución

En el filtrado por convolución, cada píxel de una imagen es reemplazado por la suma pesada de sí mismo y una cantidad de píxeles cercanos. El peso asignado a cada píxel y la cantidad y posición de píxeles cercanos que influyen en cada operación son determinados por una matriz cuadrada llamada *kernel* o núcleo, o simplemente filtro o matriz. La matriz de convolución se desplaza sobre toda la imagen original, fila por fila y columna por columna, de manera tal que al recorrer todos los píxeles se ha creado una nueva imagen filtrada.

Matemáticamente, la operación es una convolución bidimensional entre la imagen y la matriz de convolución, que generalmente es de un tamaño mucho menor a la imagen a filtrar.

En este trabajo utilizamos *kernels* estándar, especificados a continuación. En todos los casos el píxel ancla o *anchor* es el punto central del *kernel*.

##### Promediado (*box blur*)

$$K = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

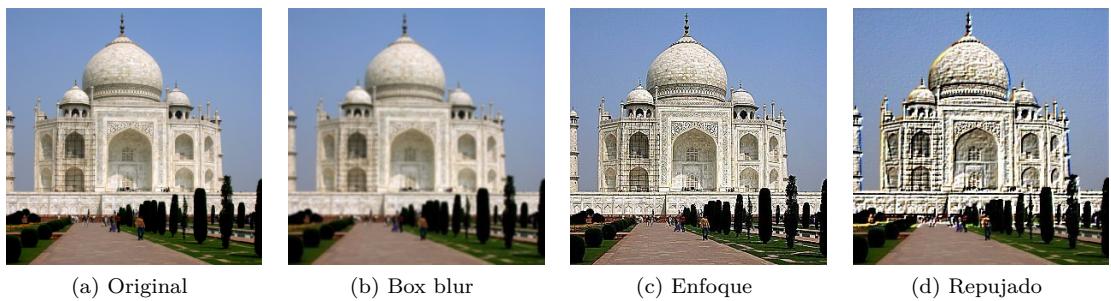
##### Enfoque (*sharpen*)

$$K = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

##### Repujado (*emboss*)

$$K = \begin{bmatrix} -2 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix}$$

Figura 15: Filtrado por convolución



## 6. Pruebas

### 5.4. Postprocesamiento

#### 5.4.1. Estimación de la altura real

Ya que se cuenta con la imagen de profundidad, se puede estimar la altura real de cada objeto detectado.[43] Así, se pueden eliminar objetos que sean mayores o menores a un rango de alturas válido para personas.

$$L_r : \text{Longitud real del objeto [m]}$$

$$L_p : \text{Longitud del objeto [px]}$$

$$D_r : \text{Distancia al objeto [m]}$$

$$F_p : \text{Distancia focal [px]}$$

$$L_r = \frac{L_p * D_r}{F_p}$$

Se utilizó una distancia focal  $F_p = 570 \text{ px}$ , de acuerdo a lo sugerido en [7] y otros. La medida de distancia se toma en el centro horizontal de cada cuadro, y a  $2/3$  de altura desde la base.

## 6. Pruebas

Se evaluaron los resultados de los detectores mencionados en la sección 5, para distintas combinaciones de parámetros, sobre sets de imágenes tomados desde el robot TURTLEBOT. El objetivo es analizar la influencia de cada parámetro y elegir la mejor combinación para implementarla en un nodo de ROS que se ejecutará en el robot. Los resultados se presentan en la sección 8.

### 6.1. Metodología

- Por simplicidad, los parámetros de cada detector se consideran independientes entre sí. Se evalúa la incidencia de cada uno sobre los resultados totales, manteniendo todos los demás constantes en valores relativamente estándar.
- El programa que realiza las detecciones es **dp\_opencv**. Este programa guarda en un archivo *csv* los datos de las detecciones, y en un *txt* los parámetros que se utilizaron.
  - Fue diseñado para que fácilmente puedan añadirse y utilizarse otros detectores. De hecho, para extraer manualmente las coordenadas reales se utilizó una clase detectora.
- El programa **dp\_resultados** compara las detecciones de un csv con las coordenadas reales del set. Guarda un archivo con los resultados.
- Un conjunto de *scripts* de bash ejecuta **dp\_opencv** y **dp\_resultados** para distintas combinaciones de parámetros y junta los resultados en un único archivo para posterior visualización.

### 6.2. Evaluación

#### 6.2.1. Igualdad

Cada detección es un rectángulo contenido en la imagen de entrada. Sea  $e$  una detección estimada y  $r$  el rectángulo real que contiene a una persona en una imagen,

$$e \cong r \text{ si y solo si}$$

1. La distancia euclídea entre centros es menor a una proporción del ancho de  $r$

$$\sqrt{(centro_r^x - centro_e^x)^2 + (centro_r^y - centro_e^y)^2} < P\_RADIO * anchor_r$$

2. La diferencia entre anchos es menor a una proporción del ancho de  $r$

$$|anchor_r - anchor_e| < P\_ANCHO * anchor_r$$

## 6. Pruebas

3. La diferencia entre alturas es menor a una proporción de la altura de  $r$

$$|alto_r - alto_e| < P\_ALTO * alto_r$$

Se ajustaron<sup>14</sup> los factores a:

$$\begin{aligned}P\_RADIO &= 0,5 \\P\_ANCHO &= 1 \\P\_ALTO &= 0,4\end{aligned}$$

### 6.2.2. Medidas básicas

Presentamos algunas medidas útiles para la definición de las métricas a optimizar:

**Verdadera positiva** Una persona estimada que coincide con una persona real en la imagen.  
*En este trabajo se cuentan como verdaderas todas las detecciones que coincidan con una persona, aunque sea la misma persona.*

**Falsa positiva** Una detección estimada que no coincide con ninguna persona real.

**Falsa negativa** Una persona real que no tuvo ninguna detección que coincida.

A partir de lo anterior se definen:

**Precisión** Proporción verdadera de todas las detecciones realizadas.

$$\text{Precisión} = \frac{\text{verdaderas positivas}}{\text{verdaderas positivas} + \text{falsas positivas}}$$

**Recuperación** También llamada exhaustividad o *recall*. Proporción detectada de todas las personas reales marcadas.

$$\text{Recuperación} = \frac{\text{verdaderas positivas}}{\text{verdaderas positivas} + \text{falsas negativas}}$$

### 6.2.3. Medidas a optimizar

Las métricas que se intentarán optimizar son  $F_1$  y  $F_{0,5}$ , que pueden considerarse casos específicos de  $F_\beta$ , una media pesada de la precisión y la recuperación. Mide la efectividad de recuperación para un usuario que le da  $\beta$  veces más importancia a la recuperación que a la precisión.  $F_1$  es la media armónica de la precisión y la recuperación.  $F_{0,5}$ , entonces, prioriza la precisión sobre la recuperación. Se consideró que para la mayoría de las posibles aplicaciones del TURTLEBOT que usen este detector, es más importante la precisión que la recuperación — especialmente si luego se hace seguimiento de la detección. Las definiciones:

$$F_\beta = (1 + \beta^2) * \frac{\text{precisión} * \text{recuperación}}{\beta^2 * \text{precisión} + \text{recuperación}}$$

$$F_1 = (2) * \frac{\text{precisión} * \text{recuperación}}{\text{precisión} + \text{recuperación}}$$

$$F_{0,5} = (1 + 0,5^2) * \frac{\text{precisión} * \text{recuperación}}{0,5^2 * \text{precisión} + \text{recuperación}}$$

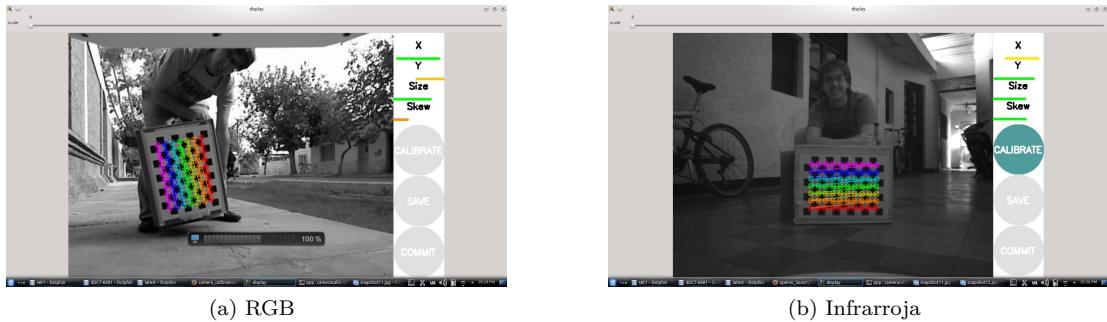
<sup>14</sup>Por prueba y error. Para esto desarrollamos el programa `dp_cuadros`, que marca las personas reales y las estimadas. *Consideramos que la inspección visual fue herramienta esencial para la detección de errores en el programa.*

## 6. Pruebas

### 6.3. Sets de imágenes

Se tomaron varios sets de imágenes para la optimización de parámetro y su posterior validación. Se construyó un nodo de ROS para capturar en forma sincronizada las imágenes RGB y de profundidad, teniendo el menor desfase temporal posible.<sup>15</sup> Las cámaras RGB e infrarroja se calibraron para reducir la distorsión geométrica. La alineación o registro de las imágenes de profundidad respecto al punto de vista de la cámara RGB es realizada por la KINECT en forma automática.

Figura 16: Calibración de cámaras



Las imágenes se tomaron con el robot TURTLEBOT en el suelo y el sensor KINECT en su montaje original, a 30 cm del suelo. Las escenas son espacios interiores con sillas, mesas y otros elementos que pueden ocluir a las personas que circulan. Las personas se marcaron manualmente; el criterio elegido fue etiquetar si más del 50 % de la persona es visible en la imagen.

Los sets:

**Set1:** 24 imágenes. Una sola persona en escena. Tomado con el robot en el suelo, en una sala cerrada, de cara a un ventanal con iluminación natural. Se utilizó en las fases iniciales de este trabajo.

**Set2:** 300 imágenes. Una sola persona en escena. Tomado con el robot en el suelo en una sala cerrada. Iluminación artificial con lámparas de bajo consumo.

**Set3:** 300 imágenes. Dos personas en escena. Tomado con el robot en el suelo en una sala cerrada. Iluminación artificial con lámparas de bajo consumo.

**Set4** 300 imágenes. Dos personas en escena. Tomado con el robot sobre una mesa, en una sala cerrada. Iluminación artificial con lámparas de bajo consumo. No se usó.

**Set5:** 300 imágenes. Dos personas en escena. Tomado con el robot en el suelo en una sala cerrada. Iluminación artificial con lámparas de bajo consumo.

**Set6:** 300 imágenes. Dos personas en escena. Tomado con el robot en el suelo, en un laboratorio. Iluminación artificial con lámparas de bajo consumo.

**Set7:** 300 imágenes. Dos personas en escena. Tomado con el robot en el suelo, en un laboratorio. Iluminación artificial con lámparas de bajo consumo.

**Set8:** 300 imágenes. Dos personas en escena. Tomado con el robot en el suelo, en un pasillo ancho. Iluminación natural a través de ventanales.

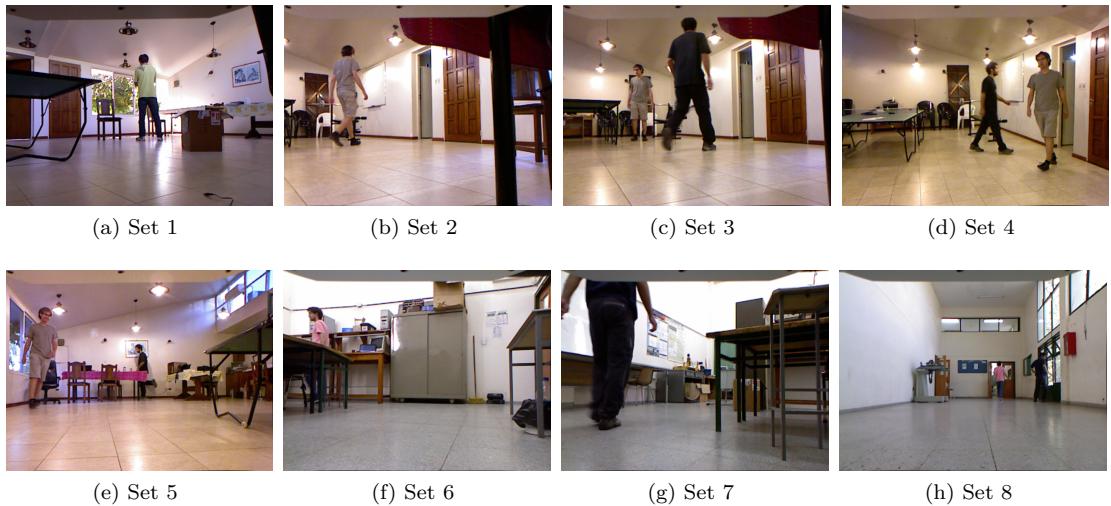
Los sets 2,3 y 5 se agruparon en uno nuevo, el set235. Este será el utilizado para las pruebas y ajustes de parámetros. Los sets 6 y 7 se agruparon en el set67, utilizado como set de validación. El set 8 también es usado como set de validación, pero separado del set67 debido al perjudicial efecto de la luz solar presente (ver 3.2).

Debe aclararse que los sets son muy limitados, en el sentido de que no tienen diversidad de personas, vestimentas, superficies, iluminación, objetos presentes, entre otros factores.

<sup>15</sup>Los mensajes a los que el programa se suscribe son /camera/rgb/image\_rect\_color y /camera/depth\_registered/hw\_registered/image\_rect\_raw

## 6. Pruebas

Figura 17: Sets de imágenes



### 6.4. Parámetros

#### 6.4.1. HOG + SVM

Los parámetros variados son:

**escala\_inicial** Ver [5.3.1](#)

**pasoEscala** Ver `scale` en [5.1.3](#)

**umbralAgrupamiento** Ver `finalThreshold` en [5.1.3](#)

**hit\_threshold** Ver `hitThreshold` en [5.1.3](#)

**setSVMClassifier** Ver `setSVMClassifier` en [5.1.3](#)

**convertir\_a\_gris** Ver [5.3.3](#)

**ecualizar\_histograma** Ver [5.3.3](#)

**blurear** Ver [5.3.4](#)

**tamano\_blur** Tamaño del filtro cuadrado de suavizado. Ver [5.3.4](#)

**filtro\_enfoque** Ver [5.3.4](#)

**filtro\_repujado** Ver [5.3.4](#)

**filtro\_enfoque\_y\_repujado** Aplicación del filtro de enfoque y luego del de repujado. Ver [5.3.4](#)

**filtro\_repujado\_y\_enfoque** Aplicación del filtro de repujado y luego del de enfoque. Ver [5.3.4](#)

**usar\_profundidad\_altura** Ver [5.4.1](#)

#### 6.4.2. LBP + Cascada

Los parámetros variados son:

**direccion\_a\_cascada** Ver [5.2.5](#)

**escala\_inicial** Ver [5.3.1](#)

## 7. Implementación

**minSize** Determinamos un cuadro mínimo de  $50 * 100\text{px}$ .<sup>16</sup> Se escala según `escala_inicial`.

**maxSize** Determinamos un cuadro máximo de  $180 * 360\text{px}$ . Se escala según `escala_inicial`.

**convertir\_a\_gris** Ver [5.3.3](#)

**ecualizar\_histograma** Ver [5.3.3](#)

**scaleFactor** Ver `scaleFactor` en [5.2.5](#)

**minNeighbors** ver `minNeighbors` en [5.2.5](#)

**usar\_profundidad\_altura** Ver [5.4.1](#)

**blurear** Ver [5.3.4](#)

**tamano\_blur** Tamaño del filtro cuadrado de suavizado. Ver [5.3.4](#)

**filtro\_enfoque** Ver [5.3.4](#)

**filtro\_repujado** Ver [5.3.4](#)

**filtro\_enfoque\_y\_repujado** Aplicación del filtro de enfoque y luego del de repujado. Ver [5.3.4](#)

**filtro\_repujado\_y\_enfoque** Aplicación del filtro de repujado y luego del de enfoque. Ver [5.3.4](#)

## 7. Implementación

Una vez realizadas las pruebas para poder determinar qué algoritmo y qué parámetros se adaptaban mejor a nuestro objetivo, implementamos el detector como un nodo de ROS para ejecutarlo en línea. **Para mayor compatibilidad solo se usaron funciones de *CPU*, ninguna de *GPU*.**

### 7.1. Conceptos generales de ROS

ROS es un sistema diseñado para ser modular, por lo que el sistema está compuesto de múltiples **nodos**. Los nodos son programas que realizan alguna acción específica, y cada equipo puede estar ejecutando varios nodos. Para poder trabajar conjuntamente necesitan comunicarse por algún medio, este medio son los **mensajes**. Un mensaje es una estructura de datos estrictamente tipificada y pueden estar formados por tipos primitivos (int, float, bool, etc), o tipos más complejos, como vectores, matrices o incluso por otros mensajes o vectores de otros mensajes. Para ordenar un poco la interacción de los nodos con los mensajes existen los **topics** o temas, títulos que agrupan a mensajes del mismo tipo y significado. Múltiples nodos pueden estar publicando mensajes en un mismo tema y más de un nodo puede estar suscripto a un mismo tema. De esta manera la comunicación es indirecta, y generalmente los nodos no conocen quienes son los publicadores o los suscriptores en un determinado *topic*. Hay métodos para asegurar una comunicación directa entre nodos evitando la publicación.

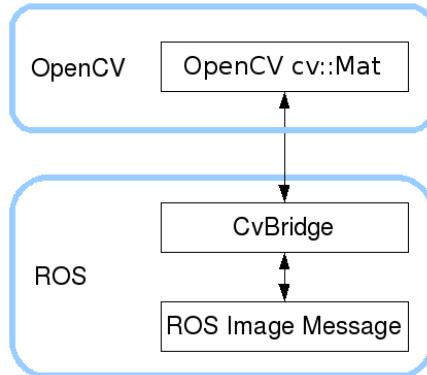
Para los nodos es totalmente transparente la topología sobre la cuál se ha implementado el sistema bajo ROS. Así, una computadora montada sobre un robot puede estar ejecutando los nodos de control general del robot, una mini-computadora o microcontrolador ejecuta un nodo de control específico de un componente, mientras que una computadora de escritorio con mayor capacidad ejecuta nodos de planificación. Todos dispositivos deben estar conectados en red. A pesar de lo que sugiere lo anterior, ROS no es un sistema distribuido, ya que requiere del funcionamiento de un único nodo central: el nodo núcleo o *core*. El dispositivo que ejecute el nodo **roscore** será denominado dispositivo maestro o **master**.

Para poder desarrollar aplicaciones de OPENCV en ROS se deben convertir los mensajes de tipo imagen de ROS a algún tipo nativo de OPENCV. Esto se realiza mediante el paquete CvBridge de ROS. [44]

<sup>16</sup>De acuerdo a mediciones preliminares, teniendo en cuenta el tamaño de imagen de  $640 * 480\text{px}$  de la KINECT, la porción visible de la imagen (debido a la plataforma superior del TURTLEBOT), y el tamaño de una persona en el rango objetivo.

## 7. Implementación

Figura 18: Funcionamiento de CvBridge de ROS



### 7.2. Información desde el TurtleBot

La computadora conectada al TURTLEBOT puede transmitir las imágenes recibidas desde la KINECT. Para esto, desde esta computadora, se ejecutan los nodos necesarios para alimentar el robot y el sensor, y luego transmitir las imágenes. Estando ROS instalado y configurado para el robot TURTLEBOT [45], se ejecuta:<sup>17</sup>

```
$ rosrun turtlebot_bringup minimal.launch
```

Seguidamente se ejecuta el nodo que pone en funcionamiento la KINECT:

```
$ rosrun turtlebot_bringup 3dsensor.launch
```

Con esto ya se estarán publicando los mensajes con la imágenes en distintos formatos. Para la imagen en color en este trabajo utilizaremos el topic `/camera/rgb/image_rect_color` y para la imagen de profundidad será `/camera/depth_registered/hw_registered/image_rect_raw`.

Se pueden visualizar las imágenes publicadas con el nodo `rqt_image_view`:

```
$ rosrun rqt_image_view rqt_image_view
```

### 7.3. Información desde la Raspberry Pi

En esta mini-computadora se instaló UBUNTU para ARM, y ROS INDIGO. Luego se utilizó el paquete RASPICAM [46], el cual contiene el driver de la cámara *SCI* de la RASPBERRY y le agrega la interfaz para manejarlo como un nodo de ROS, con servicios y mensajes que permiten explotar todas las funcionalidades de la cámara. Para publicar los mensajes con las imágenes, en la RASPBERRY se ejecuta<sup>18</sup>:

```
$ rosrun raspicam raspicam_node
```

Luego de esto se dispone dos servicios principales para usar las funcionalidades de la cámara, los cuales se pueden llamar desde la computadora maestra o cualquier otro equipo en la red:

```
$ rosservice call /camera/start_capture // Comienza la captura de video y lo publica.
```

```
$ rosservice call /camera/stop_capture // Detiene la captura de video y la publicación.
```

Cuando usamos el primero publica distintos mensajes con información. El *topic* necesario para ver la imagen de la cámara es `/camera/image`. Para ver más opciones y características del paquete, remitirse a su documentación en [46].

Nuevamente, se pueden visualizar las imágenes con `rqt_image_view`.

<sup>17</sup>Este ejecutará el nodo `roscore` si no está ya funcionando en alguna máquina de la red.

<sup>18</sup>Remotamente con `ssh` o conectando monitor y teclado a la RASPBERRY.

## 8. Resultados

### 7.4. Información desde una PC

Como agregado se decidió incorporar un nodo capaz de publicar el video de una *webcam* al sistema de ROS con el topic `/camera/image`. Este nodo detecta si existe una *webcam* conectada y publica su contenido a 10 *FPS*. Para ejecutar este nodo ejecutamos:

```
$ rosrun detpeople publicador_video
```

Este nodo puede ser ejecutado en cualquier PC con sistema ROS.

### 7.5. Nodo de detección

El nodo desarrollado para cumplir el objetivo de detectar personas, llamado `detpeople`, se suscribe al mensaje de alguno de los tres publicadores mencionados anteriormente (*webcam*, RASPBERRY PI 2 o TURTLEBOT); siempre debe ejecutarse uno de ellos. En el nodo `detpeople` se implementaron los dos algoritmos con mejores resultados para probarlos en linea, en clases separadas para aprovechar el encapsulamiento. Además se incorporaron métodos para que en caso de que exista una imagen de profundidad (como la generada por la KINECT), se estime la altura del objeto detectado. Con esto eliminamos de manera importante los falsos positivos a costa de disminuir un poco la capacidad de recuperación, como se ha demostrado en los resultados.

Este nodo de detección de personas se ejecuta de esta forma:

```
$ rosrun detpeople detpeople
```

El nodo muestra y publica la imagen con las personas detectadas marcadas para así poderlas visualizar en cualquier otra PC o incluso desde un celular o una *tablet* con ANDROID.<sup>19</sup>

Para hacer funcionar todo el entorno de ROS se debe ejecutar `roscore` en la computadora que se decida como maestra. Este nodo se encargara de ejecutar todos los servicios necesarios para que sea posible el funcionamiento del sistema.

Para mayores detalles referirse al código adjunto y al Apéndice B.

## 8. Resultados

### 8.1. Pruebas y ajuste de parámetros

#### 8.1.1. HOG + SVM

En la Tabla 1 se observan los resultados obtenidos para el detector *HOG* en el set 235.<sup>20</sup>

Algunas observaciones:

- Como se preveía, en casi todos los casos un aumento de precisión implica un decremento de recuperación, y viceversa.
- El detector DAIMLER tuvo un peor desempeño que el DEFAULT. Manteniendo el resto de los parámetros igual, el detector Daimler mejoró la recuperación (80 % contra 66 %) pero tuvo un gran decremento en precisión (19 % contra 96 %). Hay que recordar (ver 5.1.3) que están entrenados con imágenes de distinto tamaño y tal vez eso está relacionado con los otros parámetros.
- La ecualización de histograma mejora los resultados.
- La mejor escala inicial no fue 1 (sin escalado) sino que fue 1,5. Sugiere una relación entre la escala inicial y el paso de escala o factor de escalado.
- La altura estimada a partir de la profundidad aumentó la precisión en 3 %, a costa de un 1 % de recuperación.

Las 2 mejores combinaciones están en la Tabla 3, una con filtrado por altura estimada y la otra sin esta opción.

<sup>19</sup>Ver aplicación Ros IMAGE VIEW, disponible en el PLAY STORE de GOOGLE

<sup>20</sup>Las tablas completas se encuentran en [evaluacion/resultados/evaluacion/](#)

## 8. Resultados

Tabla 1: HOG + SVM: Resultados de la evaluación de parámetros

Parámetro	Tipo	Variación	Valor inicial	Valor final	Precisión	Recuperación	F1	F0.5	Mejor valor (F0.5)	Tiempo
tamaño blur	int	Incremento	Sin blur	9	Decrece	Decrece	Decrece	Decrece	Sin blur	Crece
escala_inicial	float	Incremento	1	3	Crece	Crece y luego decrece	Crece y luego decrece	Crece y luego decrece	1,5	Decrece
hit_threshold	float	Incremento	0	4	Crece	Decrece	Decrece	Decrece	0	Constante
umbralAgrupamiento	int	Incremento	1	8	Crece	Decrece	Crece y luego decrece	Crece y luego decrece	2	Constante
pasoEscala	float	Incremento	1,05	4	Decrece	Decrece	Decrece	Decrece	1,05	Decrece
clasificador	-	-	DAIMLER	-	-	-	-	-	Default	-
filtros	bool	-	Sin filtros	Con filtros	-	-	Decrece	Decrece	Sin filtros	-
convertir_a_gris	bool	-	No	Sí	Decrece	Crece	Decrece	Decrece	No	Decrece
eualizar_histograma	bool	-	No	Sí	Constante	Crece	Crece	Crece	Sí	Constante
usar_profundidad_altura	bool	-	No	Sí	Crece	Decrece	Crece	Crece	Sí	Constante

Tabla 2: LBP + Cascada: Resultados de la evaluación de parámetros

Parámetro	Tipo	Variación	Valor inicial	Valor final	Precisión	Recuperación	F1	F0.5	Mejor valor (F0.5)	Tiempo
tamaño blur	int	Incremento	Sin blur	9	Decrece	Crece	Crece	Decrece	Sin blur	Crece
cascada	-	Distintas cascadas	-	-	-	-	-	-	Vision-ARY	-
escala_inicial	float	Incremento	1	4	Crece	Decrece	Decrece	Crece y luego decrece	1,2	Decrece
filtros	bool	-	-	-	Crece	Decrece	Decrece	Decrece	Sin filtros	Decrece
minNeighbors	int	Incremento	1	8	Crece	Decrece	Crece y luego decrece	Crece y luego decrece	5	Constante
scaleFactor	float	Incremento	1,05	4	Crece	Decrece	Decrece	Crece y luego decrece	1,1	Decrece
eualizar_histograma	bool	-	No	Sí	Crece	Crece	Crece	Crece	Sí	Constante
usar_profundidad_altura	bool	-	No	Sí	Crece	Decrece	Crece	Crece	Sí	Constante

## 8. Resultados

Tabla 3: HOG+SVM: Mejores combinaciones en las pruebas iniciales

Parámetro	Mejor	Segunda
	Valor	
pasoEscala	1.050000	1.050000
umbralAgrupamiento	2	2
hit_threshold	0.000000	0.000000
setSVMDetector	DEFAULT	DEFAULT
escala_inicial	1.300000	1.500000
convertir_a_gris	true	true
ecualizar_histograma	true	true
blurear	false	false
tamanio_blur	3	3
filtro_enfoque	false	false
filtro_repujado	false	false
filtro_enfoque_y_repujado	false	false
filtro_repujado_y_enfoque	false	false
usar_profundidad_altura	true	false
F0.5	0.899069	0.881625
Tiempo promedio [ms]	59.569961	41.347358
Precisión	0.993234	0.978022
Recuperación	0.651865	0.632327

Tabla 4: LBP + Cascada: Mejores combinaciones en las pruebas iniciales

Parámetro	Mejor	Segunda
	Valor	
direccion_a_cascada	VISION-ARY	VISION-ARY
escala_inicial	1.300000	1.300000
tamanio_minimo	[38 x 77]	[38 x 77]
tamanio_maximo	[138 x 277]	[138 x 277]
convertir_a_gris	true	true
ecualizar_histograma	true	true
scaleFactor	1.100000	1.100000
minNeighbors	4	5
usar_profundidad_altura	true	false
blurear	false	false
tamanio_blur	3	3
filtro_enfoque	false	false
filtro_repujado	false	false
filtro_enfoque_y_repujado	false	false
filtro_repujado_y_enfoque	false	false
F0.5	0.801030	0.794961
Tiempo promedio [ms]	43.176355	43.099704
Precisión	0.972426	0.944923
Recuperación	0.469805	0.486271

### 8.1.2. LBP + Cascada

En la Tabla 2 se observan los resultados obtenidos para el detector *LBP* en el set235.

Algunas observaciones:

- Como se preveía, en casi todos los casos un aumento de precisión implica un decremento de recuperación, y viceversa.
- El modelo entrenado de VISION-ARY muestra resultados mucho mejores que los del resto de las cascadas. FULLBODY, LOWERBODY, MCS\_UPPERBODY tuvieron F<sub>0.5</sub> de 0,26, 0,15 y 0,04 respectivamente. UPPERBODY no tuvo detecciones. VISION-ARY es una cascada más moderna y con más entrenamiento. [41]
- La ecualización de histograma mejora los resultados.
- La mejor escala inicial no fue 1 (sin escalado) sino que fueron 1,2 y 1,3. Sugiere una relación entre la escala inicial y el paso de escala o factor de escalado.
- La altura estimada a partir de la profundidad aumentó la precisión en 6 %, a costa de un 4 % de recuperación.

Las 2 mejores combinaciones están en la Tabla 4, una con filtrado por altura estimada y la otra sin esta opción.

## 8.2. Pruebas con los mejores parámetros

En función de las variaciones observadas previamente, elegimos los mejores valores para cada parámetro, intentando optimizar F<sub>0.5</sub>. Las Tablas 5 y 6 muestran esas combinaciones de parámetros y sus resultados sobre el mismo set235.

Los resultados son aproximadamente iguales, e incluso peores, lo que demuestra que los parámetros no son totalmente independientes. Tanto como para la cascada con *LBP* como para el detector *HOG*, las combinaciones que utilizan filtrado por altura estimada muestran mejores resultados. El aumento en precisión que aporta supera a la disminución en recuperación que causa.

## 9. Conclusiones

Tabla 5: HOG+SVM: Parámetros optimizados sobre set235

Parámetro	Valor	
pasoEscala	1.050000	1.050000
umbralAgrupamiento	2	2
hit_threshold	0.000000	0.000000
setSVMDetector	DEFAULT	DEFAULT
escala_inicial	1.500000	1.500000
convertir_a_gris	true	true
ecualizar_histograma	true	true
blurear	false	false
tamanio.blur	3	3
filtro_enfoque	false	false
filtro_repujado	false	false
filtro_enfoque_y_repujado	false	false
filtro_repujado_y_enfoque	false	false
usar_profundidad_altura	false	true
F0.5	0.881625	0.886561
Tiempo promedio [ms]	42.712951	42.399328
Precisión	0.978022	0.995696
Recuperación	0.632327	0.616341

Tabla 6: LBP + Cascada: Parámetros optimizados sobre set 235

Parámetro	Valor	
direccion_a_cascada	VISION-ARY	VISION-ARY
escala_inicial	1.300000	1.300000
tamanio_minimo	[38 x 77]	[38 x 77]
tamanio_maximo	[138 x 277]	[138 x 277]
convertir_a_gris	true	true
ecualizar_histograma	true	true
scaleFactor	1.100000	1.100000
minNeighbors	5	5
usar_profundidad_altura	false	true
blurear	false	false
tamanio.blur	3	3
filtro_enfoque	false	false
filtro_repujado	false	false
filtro_enfoque_y_repujado	false	false
filtro_repujado_y_enfoque	false	false
F0.5	0.794961	0.796807
Tiempo promedio [ms]	43.761759	43.912127
Precisión	0.944923	0.984526
Recuperación	0.486271	0.452043

### 8.3. Pruebas sobre el set de validación

Para validar los resultados, las mismas combinaciones de 8.2 fueron probadas sobre los sets set67 y set8. Referirse a las Tablas 7, 8, 9 y 10.

En el set67 podemos ver que, al contrario que en las pruebas sobre el set235,  $F_{0.5}$  decrece al usar la información de profundidad. Por el contrario, crece para el detector en cascada. Esto es porque muchas de las personas en este set están parcialmente ocluidas, y la distancia que se estima (según lo explicado en 5.4.1) es errónea. *HOG* tiende a tener buena precisión y recuperación, por lo que el filtrado por altura estimada (a partir de distancia errónea) lo perjudica. Por otro lado, el detector en cascada indicaba muchos falsos positivos en todas las pruebas, y el filtrado correcto de estos compensa las verdaderos positivos que se pierden por estimación errónea de altura. Es decir, el aumento en precisión supera por mucho al decremento en recuperación, mientras que para el detector *HOG* la relación es inversa: la pérdida en capacidad de recuperación es mayor al incremento en precisión.

El mismo efecto se observa en las pruebas sobre el set8. Esta vez, los sujetos no están ocluidos pero la información de profundidad es escasa (ver 6.3) y lleva al rechazo de muchos verdaderos positivos. Para el caso del detector en cascada, el filtrado por altura permite rechazar la detección continua de un matafuegos cercano (ver Figura 20g), que es la causa de un  $F_{0.5}$  de 0.30.

### 8.4. Implementaciones en línea

El desempeño de las implementaciones en línea fue satisfactorio y muy similar a lo obtenido en las pruebas.

## 9. Conclusiones

### 9.1. Resultados

Evaluamos las distintas opciones que ofrece OPENCV 2.4.x para la detección de personas (detector en cascada con *features* tipo *Haar*, detector en cascada con *features* tipo *LBP* y el detector *HOG*) para un caso específico: la detección de personas desde un robot TURTLEBOT

## 9. Conclusiones

Tabla 7: HOG+SVM: Parámetros optimizados sobre set67

Parámetro	Valor	
pasoEscala	1.050000	1.050000
umbralAgrupamiento	2	2
hit_threshold	0.000000	0.000000
setSVMClassifier	DEFAULT	DEFAULT
escala_inicial	1.500000	1.500000
convertir_a_gris	true	true
ecualizar_histograma	true	true
blurear	false	false
tamanio.blur	3	3
filtro_enfoque	false	false
filtro_repujado	false	false
filtro_enfoque_y_repujado	false	false
filtro_repujado_y_enfoque	false	false
usar_profundidad_altura	false	true
F0.5	0.887170	0.844407
Tiempo promedio [ms]	44.327279	42.162186
Precisión	0.958231	0.987055
Recuperación	0.684211	0.535088

Tabla 8: LBP + Cascada: Parámetros optimizados sobre set67

Parámetro	Valor	
direccion_a_cascada	VISION-ARY	VISION-ARY
escala_inicial	1.300000	1.300000
tamanio_minimo	[38 x 77]	[38 x 77]
tamanio_maximo	[138 x 277]	[138 x 277]
convertir_a_gris	true	true
ecualizar_histograma	true	true
scaleFactor	1.100000	1.100000
minNeighbors	5	5
usar_profundidad_altura	false	true
blurear	false	false
tamanio.blur	3	3
filtro_enfoque	false	false
filtro_repujado	false	false
filtro_enfoque_y_repujado	false	false
filtro_repujado_y_enfoque	false	false
F0.5	0.826228	0.853116
Tiempo promedio [ms]	36.226137	35.497596
Precisión	0.860465	0.950413
Recuperación	0.712785	0.605263

Tabla 9: HOG+SVM: Parámetros optimizados sobre set8

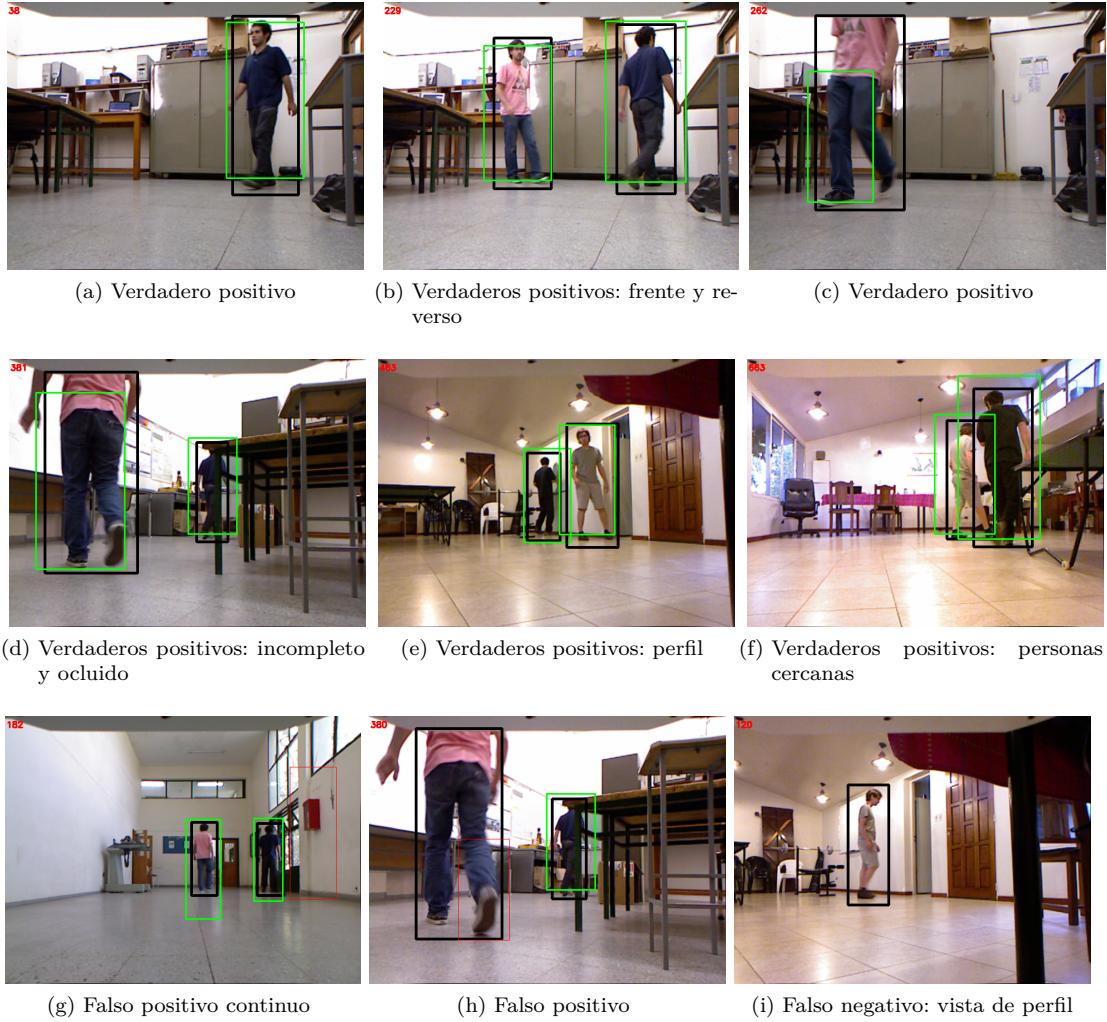
Parámetro	Valor	
pasoEscala	1.050000	1.050000
umbralAgrupamiento	2	2
hit_threshold	0.000000	0.000000
setSVMClassifier	DEFAULT	DEFAULT
escala_inicial	1.500000	1.500000
convertir_a_gris	true	true
ecualizar_histograma	true	true
blurear	false	false
tamanio.blur	3	3
filtro_enfoque	false	false
filtro_repujado	false	false
filtro_enfoque_y_repujado	false	false
filtro_repujado_y_enfoque	false	false
usar_profundidad_altura	false	true
F0.5	0.712223	0.679612
Tiempo promedio [ms]	41.782576	43.574079
Precisión	0.954839	0.992126
Recuperación	0.353222	0.300716

Tabla 10: LBP + Cascada: Parámetros optimizados sobre set8

Parámetro	Valor	
direccion_a_cascada	VISION-ARY	VISION-ARY
escala_inicial	1.300000	1.300000
tamanio_minimo	[38 x 77]	[38 x 77]
tamanio_maximo	[138 x 277]	[138 x 277]
convertir_a_gris	true	true
ecualizar_histograma	true	true
scaleFactor	1.100000	1.100000
minNeighbors	5	5
usar_profundidad_altura	false	true
blurear	false	false
tamanio.blur	3	3
filtro_enfoque	false	false
filtro_repujado	false	false
filtro_enfoque_y_repujado	false	false
filtro_repujado_y_enfoque	false	false
F0.5	0.300000	0.654699
Tiempo promedio [ms]	38.492498	39.950105
Precisión	0.286239	0.939394
Recuperación	0.371429	0.295943

## 9. Conclusiones

Figura 19: Detecciones



con sensor KINECT a 30 cm del suelo, en ambientes interiores. Se recurrió a los clasificadores entrenados incluidos en OPENCV para los detectores *Haar* y *HOG*, mientras que el detector *LBP* utilizó una cascada externa; todos están entrenados para detección de peatones, tarea que llevan a cabo en forma muy satisfactoria.

Los resultados en el limitado set de validación son de  $F_{0.5}$  mayor a 0,7, con precisiones de hasta el 90 % y recuperación del 60 %. Más pruebas en línea mostraron tendencia a error en patas de muebles, y en reflejos y sombras sobre superficies verticales. En promedio, el detector *HOG* produjo mejores resultados. El sistema es preciso, y en general los falsos positivos que se observan no son continuos en el tiempo. Si bien consideramos que los resultados obtenidos no son confiables para la mayoría de las posibles aplicaciones, creemos pueden mejorarse para hacerse útiles. Algunos factores que influyeron negativamente, en orden de importancia:

1. Los clasificadores utilizados fueron entrenados para condiciones muy distintas a los objetivos de este trabajo. Detectan personas **completas**, de pie, de frente o de espalda, con poca tolerancia a vistas de perfil o intermedias. Suponemos fueron entrenados con imágenes tomadas desde una altura promedio de ojos humanos o alturas superiores como las utilizadas para posicionar cámaras de vigilancia. Las diferencias entre esta perspectiva y la de la cámara del TURTLEBOT se acentúan a corta distancia — las habituales en los ambientes objetivo de este trabajo.
2. Los sets de imágenes utilizados para optimizar no aportaron la **diversidad** suficiente. Fue variada la orientación, posición y completitud de las personas en escena, pero fueron

## 9. Conclusiones

constantes la iluminación, la vestimenta, la altura de las personas, textura y color de piso y paredes.

3. Para simplificar el análisis, los parámetros de los detectores fueron considerados **independientes** entre sí, pero en realidad existen relaciones complejas entre varios de ellos.

### 9.2. Mejoras al trabajo realizado

Proponemos las siguientes mejoras:

- Elevar la posición de la cámara, para reducir la distancia a partir de la cual una persona se observa completa en la imagen. Esto incrementará tanto la precisión como la recuperación.
- Hacer una mejor estimación de la altura real (ver 5.4.1). En lugar de usar la profundidad de un solo punto, debería hacerse un promedio de varios o buscar una alternativa más robusta ante occlusiones.
- Superar las limitaciones del clasificador usado, ya sea con un modelo entrenado en interiores y con mayor tolerancia a las vistas laterales, o entrenando uno específicamente para la aplicación. En este caso, se puede decidir elevar el sensor KINECT para que los resultados sean aplicables a una mayor proporción de otros desarrollos.  
Para el entrenamiento, sea cual sea el clasificador elegido y una vez acotado el ámbito de acción, se debe recopilar una gran cantidad de muestras positivas, asegurándose de que contengan diversidad de posiciones, alturas, distancias, vestimenta, occlusiones, iluminación. De igual manera, es importante añadir una todavía mayor cantidad de muestras negativas, asegurándose de incluir a los elementos que más pueden confundirse con una persona; en este trabajo observamos a las patas de sillas y mesas, las sombras de personas sobre la pared y reflejos de personas sobre superficies reflectantes.
- Utilizar más de un detector, en cascada y siguiendo el concepto utilizado por el detector en cascada. Implicaría relajar los parámetros de los detectores para que tengan mayor recuperación, aunque pierdan precisión, que se logra al utilizar un detector sobre los resultados del otro.
- Utilizar la información temporal disponible al trabajar con videos. Una opción es hacer un promedio temporal de los cuadros detectados, con la intención de eliminar aquellos falsos positivos que aparecen esporádicamente. Otra, más compleja, es detectar el movimiento de personas a través de la diferencia entre cuadros, pero se debe limitar, conocer o estimar el movimiento del robot.
- Hacer seguimiento de las personas detectadas. Puede ser una buena alternativa combinado con un detector muy confiable, sacrificando capacidad de recuperación, o con el uso del filtrado temporal.

### 9.3. Mejores alternativas

A lo largo de este trabajo obtuvimos un mejor entendimiento del problema y de las posibles soluciones. Algo importante fue notar que los entornos típicos de oficina o laboratorio son muy cerrados, y la escasa distancia existente entre el robot y una persona hacen difícil obtener una imagen completa de esta, y por tanto es difícil pretender usar efectivamente detectores de personas completas. Podría solucionarse con lentes de gran angular, pero aún así existen escenarios en donde la persona se ve incompleta durante la mayor parte del tiempo. Creemos que es una buena decisión acudir a herramientas avanzadas que están diseñadas para este tipo de trabajo.

- OPENNI y NITE son herramientas avanzadas que permiten reconocer a una persona en posición natural incluso con occlusiones importantes, y luego realizar seguimiento. Si bien la documentación existente no es mucha, OPENNI ha continuado lentamente su desarrollo y continúa siendo de código abierto. El problema es que NITE, el *middleware* que realiza la detección y seguimiento de personas, desapareció al ser comprada su empresa desarrolladora; todavía se pueden usar las versiones antiguas que se distribuyen de manera no oficial,

## 9. Conclusiones

pero el producto ha sido discontinuado y eso implicará falta de compatibilidad con otras plataformas de software o hardware.

- Otra alternativa es recurrir al KINECT SDK (*Software Developement Kit*) de MICROSOFT, pero únicamente está disponible para MICROSOFT WINDOWS y las últimas versiones no soportan al sensor KINECT para XBOX 360. Sin embargo, está en activo desarrollo por quienes crearon y comercializan los sensores KINECT, permite interacción con MATLAB y OPENCV, y se permite su uso (bajo ciertas condiciones) en aplicaciones comerciales.
- OPENPTRACK es una alternativa abierta muy interesante que permite arreglos multi-cámara. Los autores nos informaron que lo han usado satisfactoriamente desde un robot, con la cámara a 1,3 m de altura. Otro usuario afirma que, desde 0,5 m de altura, su problema es perder el seguimiento al desaparecer las cabezas de las personas.

### 9.4. Comentarios

*Para la realización de este trabajo tuvimos la necesidad de conocer y aprender a usar herramientas muy útiles y de amplio uso industrial: OPENCV y ROS. Tanto el desarrollar en ellos como buscar alternativas, nos permitieron conocer características que desconocíamos e inferir potenciales aplicaciones.*

*También fue nuestra primer experiencia en LINUX (usando UBUNTU, XUBUNTU Y KUBUNTU); pudimos apreciar sus características de primera mano. No solo desarrollamos programas complejos en los entornos ECLIPSE y NETBEANS, sino que también recurrimos a otras herramientas, como scripts de bash, ssh, entre otros. Incluso, para la escritura de este informe, utilizamos LATEX a través de LyX. Para el control de versiones aprendimos GIT, y siempre trabajamos colaborativamente con las herramientas de GITHUB<sup>21</sup>. Si bien todo esto significó más tiempo de trabajo y aprendizaje, pensamos que ha sido una buena inversión dirigida a incrementar nuestras aptitudes profesionales y capacidades personales.*

*Intentamos aplicar un procedimiento científico ordenado para poder lograr conclusiones concretas. Si bien los resultados no fueron los esperados, esperamos que sirvan de base para mejores implementaciones. Recomendamos acudir continuamente al consejo de personas experimentadas en la temática, para orientar bien los esfuerzos y el tiempo invertido. Esperamos en un futuro cercano probar las alternativas mencionadas en 9.3 para comprobar su eficacia en la aplicación objetivo de este trabajo.*

*Agradecemos al GRUPO ESTUDIANTIL DE ROBÓTICA DE LA UNIVERSIDAD NACIONAL DE SAN JUAN (GERUNSJ) por permitirnos disponer de 2 robots TURTLEBOT, y al profesor Adrián Orellana por sus apoyo y útiles recomendaciones. Agradecemos también a todas las personas que comparten día a día sus problemas, soluciones y conocimientos, en diversas comunidades de internet.*

<sup>21</sup>El código fuente de todos los programas, los sets de imágenes utilizados y los resultados obtenidos, pueden descargarse desde <https://github.com/GERUNSJ/deteccion-de-personas-con-turtlebot-y-opencv-1>

## Referencias

- [1] Wikipedia, “iRobot Create — Wikipedia, The Free Encyclopedia.” [https://en.wikipedia.org/wiki/IRobot\\_Create](https://en.wikipedia.org/wiki/IRobot_Create), 2016. [Internet; descargado abril-2016]. 3
- [2] ASUS, “Asus 1215N.” [https://www.asus.com/Notebooks/Eee\\_PC\\_1215N](https://www.asus.com/Notebooks/Eee_PC_1215N). [Internet; descargado abril-2016]. 3
- [3] Wikipedia, “Kinect — Wikipedia, The Free Encyclopedia.” <https://en.wikipedia.org/wiki/Kinect>, 2016. [Internet; descargado marzo-2016]. 3, 4
- [4] OpenKinect wiki, “Imaging Information.” [https://openkinect.org/wiki/Imaging\\_Information](https://openkinect.org/wiki/Imaging_Information), 2013. [Internet; descargado marzo-2016]. 4
- [5] M. Sebastián Magallón and E. Montijano Muñoz, “Sistema interactivo para manejo de electrodomésticos en entornos domésticos.” <http://zaguán.unizar.es/record/12845>, 2013. 4, 5
- [6] ROS wiki, “Kinect accuracy.” [http://wiki.ros.org/openni\\_kinect/kinect\\_accuracy](http://wiki.ros.org/openni_kinect/kinect_accuracy), 2011. [Internet; descargado marzo-2016]. 5
- [7] K. Konolige and P. Mihelich, “Technical description of Kinect calibration.” [http://wiki.ros.org/kinect\\_calibration/technical](http://wiki.ros.org/kinect_calibration/technical), 2012. [Internet; descargado marzo-2016]. 5, 18
- [8] B. Freedman, A. Shpunt, M. Machline, and Y. Arieli, “Depth mapping using projected patterns.” <http://www.google.com/patents/US20100118123>, May 13 2010. US Patent App. 12/522,171. 5
- [9] D. Scharstein and R. Szeliski, “High-Accuracy Stereo Depth Maps Using Structured Light,” in *CVPR (1)*, (<http://research.microsoft.com/pubs/75606/Scharstein-CVPR03.pdf>), pp. 195–202, IEEE Computer Society, 2003. 5
- [10] Kinect for Matlab, “Modo de funcionamiento normal.” <http://kinectformatlab.es.tl/MODO-DE-FUNCIONAMIENTO-NORMAL.htm>. [Internet; descargado abril-2016]. 5
- [11] J. Nuño Simón, “Reconocimiento de objetos mediante sensor 3D Kinect.” <http://e-archivo.uc3m.es/handle/10016/16917>, 2012. 5
- [12] J. Smisek, M. Jancosek, and T. Pajdla, “3D with Kinect.,” in *ICCV Workshops*, (<http://cmp.felk.cvut.cz/ftp/articles/pajdla/Smisek-CDC4CV-2011.pdf>), pp. 1154–1160, IEEE, 2011. 5
- [13] R. P. Foundation, “Raspberry Pi 2 model B.” <https://www.raspberrypi.org/products/raspberry-pi-2-model-b>. [Internet; descargado abril-2016]. 6
- [14] Open Source Robotics Foundation, “Robot Operating System.” <http://www.ros.org>, 2016. 7
- [15] Wikipedia, “Sistema Operativo Robótico — Wikipedia, La enciclopedia libre.” [https://es.wikipedia.org/wiki/Sistema\\_Operativo\\_Robótico](https://es.wikipedia.org/wiki/Sistema_Operativo_Robótico), 2015. [Internet; descargado abril-2016]. 7
- [16] Itseez, “Open Source Computer Vision Library.” <http://opencv.org>. 7
- [17] OpenCV, “OpenCV 2.4 Documentation: Cascade Classification.” [http://docs.opencv.org/2.4/modules/objdetect/doc/cascade\\_classification.html](http://docs.opencv.org/2.4/modules/objdetect/doc/cascade_classification.html). [Internet; descargado marzo-2016]. 8, 13, 15
- [18] P. Viola and M. Jones, “Rapid Object Detection using a Boosted Cascade of Simple Features,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, (Hawaii), 2001. 8, 13, 14

## Referencias

- [19] T. Ojala, M. Pietikäinen, and T. Mäenpää, “Multiresolution gray-scale and rotation invariant texture classification with local binary patterns,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 24, no. 7, pp. 971–987, 2002. 8, 14
- [20] OpenCV, “OpenCV 2.4 Documentation: HOG.” [http://docs.opencv.org/2.4/modules/gpu/doc/object\\_detection.html#gpu-hogdescriptor](http://docs.opencv.org/2.4/modules/gpu/doc/object_detection.html#gpu-hogdescriptor). [Internet; descargado marzo-2016]. 9, 10, 12
- [21] N. Dalal and B. Triggs, “Histograms of Oriented Gradients for Human Detection,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, (San Diego, USA), 2005. 9, 10
- [22] OpenCV, “OpenCV 2.4 Documentation: Latent SVM.” [http://docs.opencv.org/2.4/modules/objdetect/doc/latent\\_svm.html](http://docs.opencv.org/2.4/modules/objdetect/doc/latent_svm.html). [Internet; descargado marzo-2016]. 9
- [23] P. F. Felzenszwalb, R. B. Girshick, D. A. McAllester, and D. Ramanan, “Object Detection with Discriminatively Trained Part-Based Models.,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 9, pp. 1627–1645, 2010. 9
- [24] Tim Field and Marcus Liebhardt, “ROS package: Openni tracker.” [http://wiki.ros.org/openni\\_tracker](http://wiki.ros.org/openni_tracker), 2013. [Internet; descargado marzo-2016]. 9
- [25] Igalia, “Skeltrack.” <https://github.com/joaquimrocha/Skeltrack>, 2013. 9
- [26] A. Baak, M. Müller, G. Bharaj, H. P. Seidel, and C. Theobalt, “A data-driven approach for real-time full body pose reconstruction from a depth camera,” in *Computer Vision (ICCV), 2011 IEEE International Conference on*, pp. 1092–1099, Nov 2011. 9
- [27] M. Munaro, C. Lewis, D. Chambers, P. Hvass, and E. Menegatti, “RGB-D Human Detection and Tracking for Industrial Environments.,” in *IAS* (E. Menegatti, N. Michael, K. Berns, and H. Yamaguchi, eds.), vol. 302 of *Advances in Intelligent Systems and Computing*, pp. 1655–1668, Springer, 2014. 9
- [28] M. Munaro, C. Lewis, D. Chambers, P. Hvass, E. Menegatti, and S. Edwards, “ROS package: Human tracker.” [https://github.com/ros-industrial/human\\_tracker](https://github.com/ros-industrial/human_tracker), 2013. [Internet; descargado marzo-2016]. 9
- [29] University of California, “OpenPTrack.” <http://openptrack.org>, 2016. 9
- [30] M. Munaro, F. Basso, and E. Menegatti, “OpenPTrack: Open source multi-camera calibration and people tracking for RGB-D camera networks.,” *Robotics and Autonomous Systems*, vol. 75, pp. 525–538, 2016. 9
- [31] M. Munaro, A. Horn, R. Illum, J. Burke, and R. B. Rusu, “OpenPTrack: People Tracking for Heterogeneous Networks of Color-Depth Cameras,” 2014. 9
- [32] C. McCormick, “HOG person detector tutorial.” <https://chrisjmccormick.wordpress.com/2013/05/09/hog-person-detector-tutorial>, 2013. [Internet; descargado abril-2016]. 10
- [33] Wikipedia, “Máquinas de vectores de soporte — Wikipedia, La enciclopedia libre.” [https://es.wikipedia.org/wiki/M%C3%A1quinas\\_de\\_vectores\\_de\\_soporte](https://es.wikipedia.org/wiki/M%C3%A1quinas_de_vectores_de_soporte), 2015. [Internet; descargado abril-2016]. 10
- [34] M. Enzweiler and D. M. Gavrila, “Monocular Pedestrian Detection: Survey and Experiments,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 31, no. 12, pp. 2179–2195, 2009. 12
- [35] R. Lienhart, A. Kuranov, and V. Pisarevsky, “Empirical Analysis of Detection Cascades of Boosted Classifiers for Rapid Object Detection,” in *Proceedings of the 25th DAGM Symposium on Pattern Recognition*, (Magdeburg, Germany), 2003. 14
- [36] M. Pietikäinen, “Local Binary Patterns,” *Scholarpedia*, vol. 5, no. 3, p. 9775, 2010. revision #137418 [http://www.scholarpedia.org/article/Local\\_Binary\\_Patterns](http://www.scholarpedia.org/article/Local_Binary_Patterns). 15

## Referencias

- [37] P. Wagner, “Local Binary Patterns.” [http://bytefish.de/blog/local\\_binary\\_patterns](http://bytefish.de/blog/local_binary_patterns), 2011. [Internet; descargado abril-2016]. 15
- [38] A. Rosebrock, “Local Binary Patterns with Python and OpenCV.” <http://www.pyimagesearch.com/2015/12/07/local-binary-patterns-with-python-opencv>, 2015. [Internet; descargado abril-2016]. 15
- [39] H. Kruppa, M. Castrillon-Santana, and B. Schiele, “Fast and robust face finding via local context,” pp. 157–164, 2003. 16
- [40] M. Castrillón, O. Déniz, C. Guerra, and M. Hernández, “ENCARA2: Real-time detection of multiple faces at different resolutions in video streams.,” *J. Visual Communication and Image Representation*, vol. 18, no. 2, pp. 130–140, 2007. 16
- [41] Vision-Ary Team, “Boost the World: Pedestrian Detection.” <http://www.vision-ary.net/2015/03/boost-the-world-pedestrian>, 2015. [Internet; descargado abril-2016]. 16, 26
- [42] E. CV, “Emgu CV 1.5 Library Documentation: HAAR\_DETECTION\_TYPE Enumeration.” <http://www.emgu.com/wiki/files/1.5.0.0/Help/html/e2278977-87ea-8fa9-b78e-0e52cfe6406a.htm>, 2008. [Internet; descargado abril-2016]. 16
- [43] Wikipedia, “3D Projection — Wikipedia, The Free Encyclopedia.” [https://en.wikipedia.org/wiki/3D\\_projection#Diagram](https://en.wikipedia.org/wiki/3D_projection#Diagram), 2016. [Internet; descargado abril-2016]. 18
- [44] ROS wiki, “Using CvBridge to convert between ROS images and OpenCV images.” [http://wiki.ros.org/cv\\_bridge/Tutorials/UsingCvBridgeToConvertBetweenROSImagesAndOpenCVImages](http://wiki.ros.org/cv_bridge/Tutorials/UsingCvBridgeToConvertBetweenROSImagesAndOpenCVImages), 2015. [Internet; descargado marzo-2016]. 22
- [45] ROS wiki, “Robots: TurtleBot.” <http://wiki.ros.org/Robots/TurtleBot>, 2015. [Internet; descargado marzo-2016]. 23
- [46] fpasteau, “Groovy ROS node for camera module of Raspberry Pi.” [https://github.com/fpasteau/raspicam\\_node](https://github.com/fpasteau/raspicam_node), 2015. [Internet; descargado abril-2016]. 23

*A. Apéndice: Estructura de archivos*

- A. Apéndice: Estructura de archivos**
- `informe_Aguado_Emder.pdf` — Este informe
  - `evaluacion/` — Lo referido a la evaluación de algoritmos
  - `evaluacion/clasificadores/` — Los clasificadores utilizados
  - `evaluacion/ejecutables/` — Ejecutables compilados para Linux x64
  - `evaluacion/resultados/` — Resultados sobre los sets de prueba
  - `evaluacion/scripts/` — *Scripts* de evaluación
  - `evaluacion/src/` — Código fuente de los programas de evaluación
  - `implementacion/` — Nodos de ROS para funcionamiento en línea. Ver Apéndice B
  - `otros/calibracion_turtlebot/` — Archivos de calibración de las cámaras
  - `otros/raspicam_node-master.zip` — Nodo RASPICAM
  - `resultados/` — Imágenes y videos de los resultados sobre los sets de prueba y validación

## B. Apéndice: Nodos

El *workspace* de ROS contiene el nodo de detección y un nodo publicador de video para la *webcam*. Para poder ejecutarlos, primero hay que compilarlos.

Los pasos a seguir:

1. Modificar el código (`src/detpeople/src/ImageConverter.cpp`, líneas 48 y 51) para que se suscriba a los mensajes de la KINECT o de la *webcam* o de la cámara de la RASPBERRY Pi. Por defecto está suscripto a los mensajes bajo el topic `/camera/image` , que son los de la *webcam* o de la cámara de la RASPBERRY.
2. Compilar:
  - a) Situarse dentro de la carpeta `implementacion`
  - b) Ejecutar `catkin_make` . Esto compila los nodos.
  - c) Se le informa a ROS la existencia de esos paquetes con `source devel/setup.bash`
3. Se ejecutan los nodos <sup>22</sup> con `rosrun`
  - a) [Opcional] `rosrun detpeople publicador_video` — Ejecuta un nodo que publica video desde la *webcam* de la PC.
  - b) `rosrun detpeople detpeople` — Ejecuta el detector.

Para recibir imágenes desde el TURTLEBOT, ver [7.2](#). Para transmitir imágenes desde la RASPBERRY Pi, referirse a [7.3](#); el nodo RASPICAM se puede compilar de forma similar a lo realizado en el punto 2 anterior.

---

<sup>22</sup>Cada nodo se ejecuta en una terminal o consola diferente.