UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

PROGRAMA DE MAESTRÍA Y DOCTORADO EN INGENIERÍA

Detección y reconocimiento de objetos para aplicaciones en robots móviles empleando técnicas de visión computacional

T E S I S

QUE PARA OPTAR POR EL GRADO DE:

MAESTRO EN INGENIERÍA

INGENIERÍA ELÉCTRICA - PROCESAMIENTO DIGITAL DE SEÑALES

P R E S E N T A :

LUIS ANGEL CONTRERAS TOLEDO

TUTOR: DR. JESÚS SAVAGE CARMONA

CO-TUTOR: DR. WALTERIO W. MAYOL-CUEVAS

2011

Universidad Nacional Autónoma de México
UNAM
Biblioteca Central
Dirección General de Bibliotecas de la UNAM

UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

JURADO ASIGNADO:

PRESIDENTE:	**Dr. PÉREZ ALCÁZAR PABLO ROBERTO**
SECRETARIO:	**Dr. ESCALANTE RAMÍREZ BORIS**
VOCAL:	**Dr. SAVAGE CARMONA JESÚS**
1er. SUPLENTE:	**Dra. MEDINA GOMEZ LUCIA**
2do. SUPLENTE:	**M. I. ESCOBAR SALGUERO LARRY**

Lugares en donde se realizó la tesis:

LABORATORIO DE BIO-ROBÓTICA
FACULTAD DE INGENIERÍA – UNAM

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE BRISTOL, INGLATERRA

TUTOR DE TESIS:

Dr. SAVAGE CARMONA JESÚS

_____
FIRMA

Resumen

En el presente trabajo se muestran técnicas desarrolladas para la detección y reconocimiento de objetos. Se establece el marco teórico de la técnica de Histogramas de Co-ocurrencias de Campos Receptivos (RFCH) y su aplicación en la detección de objetos, así como de un extractor de características invariante a la escala (SIFT) para el reconocimiento. Se presenta entonces una mejora para RFCH a través de la transformada Wavelet como descriptor de texturas, y se presenta un descriptor de características basado en la información de profundidad obtenida por medio de una cámara RGB-D como complemento a RFCH + Wavelet para cumplir con la tarea de reconocimiento, y se compara es rendimiento con el del descriptor SIFT. Finalmente, de presenta su aplicación en robots móviles tipo humanoide.

Abstract

In this thesis we present developed techniques for object detection and recognition. We first establish the theoretical framework of the receptive fields co-occurrence histograms (RFCH) and its application in the object detection task as well as an scale invariant feature extractor (SIFT) for object recognition. Then, we present an improvement for the RFCH through Wavelet Transform as a texture descriptor, and present a feature descriptor based on the depth information obtained through RGB-D camera as a complement to RFCH + Wavelet to meet the recognition task, and its performance is compared with the SIFT descriptor one. Finally, we show the application of these techniques in humanoid mobile robots.

Índice

Capítulo I. Introducción

1.1 Presentación

El control y la automatización en los procesos industriales se necesitan cada vez más, y en los países en desarrollo esta urgencia es más evidente. Al mismo tiempo, el avance en la robótica es cada vez más intenso, por lo que para tener una mayor competitividad la industria mexicana se debe actualizar y, por eso, se necesita de personal calificado. Esto ha sido entendido por las grandes empresas e institutos de investigación, los cuales han empezado a adoptar modelos actuales de control y robótica, que a su vez generan sistemas complejos robotizados para dar solución a problemas industriales. El objetivo de estos estudios es estar preparado para manipular y generar tecnología de punta en el ámbito local, teniendo en cuenta las condiciones particulares de México.

El rápido progreso en el desarrollo de la tecnología en cómputo ha permitido alcanzar estos objetivos de forma cada vez más eficiente. A su vez, con el desarrollo de la llamada "Inteligencia Artificial" (AI, *Artifical Intelligence*) se trata de resolver estos problemas imitando el comportamiento humano. Al principio, el hecho de resolver problemas reales de forma simbólica, empleando diversos algoritmos, fue considerado 'inteligencia'. Sin embargo, la complejidad del comportamiento humano sobrepasa por mucho la capacidad de estos sistemas (G. Fink, 2003). La inteligencia humana, pues, se considera no solo la respuesta a un estímulo, sino como la capacidad de procesamiento de múltiples entradas sensoriales simultáneas.

Como ejemplo tenemos la comunicación hablada, la interpretación de estímulos visuales y la interacción con el medio ambiente a través de la navegación espacial y la manipulación de objetos.

El presente trabajo se enfoca en el reconocimiento y la detección de objetos para obtener la información espacial necesaria que sirva para la posterior manipulación de estos objetos por medios mecánicos.

Un algoritmo de reconocimiento de objetos se diseña para permitir la clasificación de un objeto a una de varias clases predefinidas, suponiendo que la segmentación del objeto ya se ha realizado. La segmentación de una imagen implica la división o separación de la imagen en regiones de atributos similares. El atributo más básico para la segmentación es la amplitud de la luminancia para una imagen monocromática y componentes de color de una imagen en color (S. Ekvall *et al*., 2005).

La tarea de un algoritmo de detección de objetos es mucho más difícil. Su objetivo es la búsqueda de un objeto específico en una imagen de una escena compleja. La mayoría de los algoritmos de reconocimiento de objetos se puede utilizar para la detección mediante el uso de una ventana de búsqueda y digitalización de la imagen del objeto. En cuanto a la complejidad computacional, algunos métodos para la búsqueda son más adecuados que otros.

1.1.1 Objetivos

Esta investigación se enfocará en el desarrollo de un sistema de detección y reconocimiento de objetos con rotación, traslación y escala arbitrarias en ambientes humanos. Los objetivos generales son:

1. Definir las características más importantes que caracterizan a una imagen.
2. Conocer el estado del arte de los sistemas de visión computacional.
3. Analizar las técnicas de programación más recientes para la detección y reconocimiento de objetos.
4. Conocer las técnicas más recientes para la detección de objetos en tiempo real.

Basado en los objetivos generales, los objetivos particulares resultan:

1. Tener acceso a las investigaciones actuales como fuentes originales de información.
2. Desarrollar un sistema que involucre el procesamiento de imágenes para la toma de decisiones.
3. Aplicar el conocimiento teórico a un proceso real de manipulación robótica.

1.2 Motivación del trabajo de tesis

El uso de robots ha tenido un impacto considerable en la generación de productos y servicios, provocando que los costos y el tiempo de producción se reduzcan considerablemente (L. Contreras, 2007). Sabemos que estas reducciones implican una gran cantidad de efectos, entre los cuales está la reducción de precio para el consumidor final con lo que el recurso generado se hará más accesible para una mayor cantidad de personas.

Así mismo, en años recientes se ha visto incrementado el empleo de robots para tareas domésticas sencillas, teniendo que interactuar de forma autónoma en ambientes naturales. En la competencia internacional *RoboCup* (http://www.robocup.org/) existe la división RoboCup@Home enfocada en desarrollar tecnología para robots de servicio y asistencia, con un gran impacto en aplicaciones futuras para robots domésticos personales. Tales robots deben poder navegar en ambientes dinámicos y poblados, reconocer y/o evitar objetos y personas, interactuar por medio de voz con humanos, entre otras funciones.

Para que este tipo de robots cumpla con su tarea, es sabido que deben interactuar una serie de sistemas y procesos (S. Ekvall *et al*., 2006): procesos cognitivos de alto nivel para el razonamiento abstracto y la planeación; procesos motrices y de sensado de bajo nivel para la extracción de información y la ejecución de acciones; y procesos de medio nivel conectando ambos procesos.

El laboratorio de Bio-Robótica de la UNAM, entendiendo esta necesidad, ha venido desarrollando las herramientas necesarias para implementar un robot de servicio tipo humanoide, capaz de navegar por sensores láser y odometría, reconocer comandos de voz, sistemas de visión para reconocimiento de objetos y personas, y sistemas mecánicos para manipular objetos.

El presente trabajo de tesis pretende dar continuidad a este proyecto que, bajo el mando del Dr. Jesús Savage Carmona, se ha venido desarrollando a lo largo de 15 años.

Capítulo II. Fundamentos y antecedentes

2.1 Antecedentes

El Procesamiento Digital de Señales es una de las tecnologías más poderosas desarrolladas que marcará el paso de la ciencia e ingeniería en el siglo XXI (S. Smith, 1999; A. Antoniou, 2007). Las señales a procesar provienen, generalmente, de fenómenos físicos, como: vibraciones sísmicas, imágenes, aceleraciones de cuerpos, etc. El procesamiento de estos fenómenos se logra a través de una discretización de las características de interés y, con un sistema de sensores adecuado para la adquisición de datos, su posterior manipulación en el mundo digital.

Para el caso en el que pasamos del mundo real al mundo digital a través de la adquisición de imágenes, se tiene que una imagen digital (o simplemente imagen) es un arreglo 2D de valores que representan intensidad de luz. En el procesamiento digital de imágenes, un sensor de imágenes convierte una imagen en un número discreto de píxeles. Este sensor asigna numéricamente a cada píxel una posición y un valor en escala de gris o color que especifica el brillo o color del píxel. Una imagen digital tiene tres propiedades básicas: resolución, definición y número de planos.

La visión por computadora es una disciplina cuyo objetivo es tomar decisiones a partir de características del mundo real, extraídas a través de imágenes digitales del mismo. Dicho con otras palabras, la visión por computadora construye descriptores de una escena basándose en las características relevantes de una imagen o una secuencia de ellas, de tal manera que la caracterización esté libre de información irrelevante.

Avances recientes en los algoritmos y técnicas de visión por computadora han proveído de una gran cantidad de herramientas y soluciones útiles para distintas aplicaciones en diferentes campos profesionales (Nicholson *et al*., 1999; Ronacher *et al*., 2000; Müllier *et al*., 2001; Fukushi, 2001; Collett *et al*., 2001; Y. Chen *et al*., 2002; Graham *et al*., 2003; K. Kanda *et al*., 2005; F. Peña *et al*., 2006).

Estudiar visión no solo implica la extracción de información, sino también el almacenamiento de la misma para tenerla disponible como base para futuras tomas de decisiones.

Un algoritmo de reconocimiento de objeto está diseñado típicamente para clasificar un objeto de una serie de clases predefinidas, asumiendo que la segmentación del objeto ha sido realizada previamente. La segmentación de una imagen conlleva la división o separación de la imagen en regiones con características similares (S. Ekvall, 2005)).

Dado que para un buen reconocimiento de objetos se debe tener una buena segmentación, la primer parte de este proyecto se enfoca en este apartado. Las características más básicas para resolver este problema son color, textura, bordes y contornos, entre otras.

Una vez que se tienen detectadas las características de un objeto, se emplean algoritmos de detección, cuyo propósito es buscar un determinado objeto en una imagen de una escena compleja. Muchos de los algoritmos de reconocimiento de objetos se pueden emplear para la detección empleando ventanas de búsqueda y escaneando la imagen en búsqueda del objeto. Sin embargo, la complejidad de cómputo es alta para esta tarea, la cual se tiene que repetir una vez que se detecta una región de interés que puede incluir al objeto para después reconocerlo.

Una vez que se da una ubicación teórica de un objeto en una imagen, se segmenta la región de interés y se aplica un algoritmo de reconocimiento, como SIFT.

2.1.1 Histogramas

El término *histograma* surge como una herramienta estadística empleada para visualizar la distribución y variación de un conjunto de datos correspondientes a un proceso particular. Una distribución de frecuencia indica cuántas veces aparece cada diferente valor en el conjunto de datos. El objetivo principal de un histograma es clarificar la presentación de los datos, por ello éstos se presentan en forma gráfica, y detectar relaciones entre ellos, que de otra forma serían difíciles de observar. A su vez, es una

herramienta útil para dividir el conjunto de datos en regiones o clases para determinar la frecuencia de ciertos eventos o categorías de datos.

Con el histograma se puede, por ejemplo, detectar el valor o región que es más frecuente en un proceso en particular. También se puede observar la distribución de ocurrencias de los valores para diferentes conjuntos de datos que provienen de variaciones de parámetros de entrada del mismo proceso.

El histograma de un proceso se representa como un tipo de gráfica de barras, figura 1, en el que datos individuales son agrupados en clases para así tener una idea de qué tan frecuente los valores ocurren dentro de cada clase en el conjunto de datos. Barras altas indican más puntos dentro de esa clase y barras bajas indican menos puntos.

Figura 1. Histograma de un proceso arbitrario con 5 mil datos de salida con valor en números reales entre 0 y 50. Se observa la gráfica para el caso de 50 clases.

2.1.1.a Información estadística

Media	El promedio de todos los valores
Mínimo	El valor más pequeño
Máximo	El valor más grande
Desviación Estándar	Cantidad que expresa qué tanto se extienden los valores alrededor de la media.
Ancho de la clase	La distancia sobre el eje x entre el límite de la izquierda y el de la derecha de cada barra en el histograma.
Número de clases	El número de barras (incluyendo las de altura cero) en el histograma

2.1.1.b Construcción

• Contar el número de puntos *n* del conjunto de datos.

• Determinar el máximo X_{max} y el mínimo X_{min} de los datos adquiridos. Calcular la diferencia entre estos dos números (*intervalo*).

• Determinar el ancho de la clase. Para calcular el ancho, se divide el *intervalo* por el número de clases *k*.

• Conteo de valores val_i, $i = 1, 2, \ldots, n$, que pertenecen a cada clase C_j, $j = 1, 2, \ldots, k$. Si $C_{j\_min} \leq val_i < C_{j\_max}$, entonces val_i pertenece a la clase C_j.

• Normalizar el histograma. Para normalizar un histograma, se divide el número total de casos en cada clase C_j por el número de puntos *n* del conjunto de datos para obtener C_{j_norm}. Esto nos da un valor entre 0 y 1 (0 y 100%), en donde $\sum_{j=1}^{k} C_{j\_norm} = 1$.

• Vectorización del histograma. Para cuestiones de representación, se considera el vector *H* de dimensión *k*, en donde $H[j] = C_j$, $j = 1, 2, \ldots, k$.

2.1.1.c Consideraciones para la construcción

Hay dos características de los histogramas que se deben considerar al momento de su construcción. El primero es que los histogramas pueden ser manipulados para mostrar gráficas diferentes. Si se emplean muy pocas o muchas clases, el histograma puede ser malinterpretado. Esta es una etapa que requiere un poco de juicio, y tal vez un poco de experimentación, basado en la experiencia en el proceso.

Por otra parte, los histogramas también pueden ocultar las diferencias temporales entre los conjuntos de datos. Por ejemplo, si se examinan los niveles de iluminación en un objeto al aire libre, se pierden los cambios naturales día/noche, así como eventos aleatorios como cielo nublado/despejado. Asimismo, en el control de calidad, un histograma de un proceso cuenta sólo una parte de una larga historia del mismo.

2.1.1.d Intersección

Para comparar dos histogramas P. Chang emplea la técnica de intersección de histogramas, definida para un histograma arbitrario H^p y un modelo H^m como (M. J. Swain *et al.*, 1991):

$$I_{pm} = \sum_{j=1}^{n} \min\left[C_j^p, C_j^m\right].$$

La intersección indica qué tan bien el histograma H^p empata con el modelo H^m. Si las clases en el histograma arbitrario corresponden con todas las clases en el modelo, entonces la intersección será igual a la suma de todas las clases en el modelo, o sea I_{mm}. Para el caso de histogramas normalizados, se tiene $I_{mm} = 1$. Se pueden considerar umbrales de empatado, en el que dos histogramas se consideran iguales si su intersección excede cierto umbral $T = \alpha I_{mm}$.

2.1.2 Segmentación por k-medias

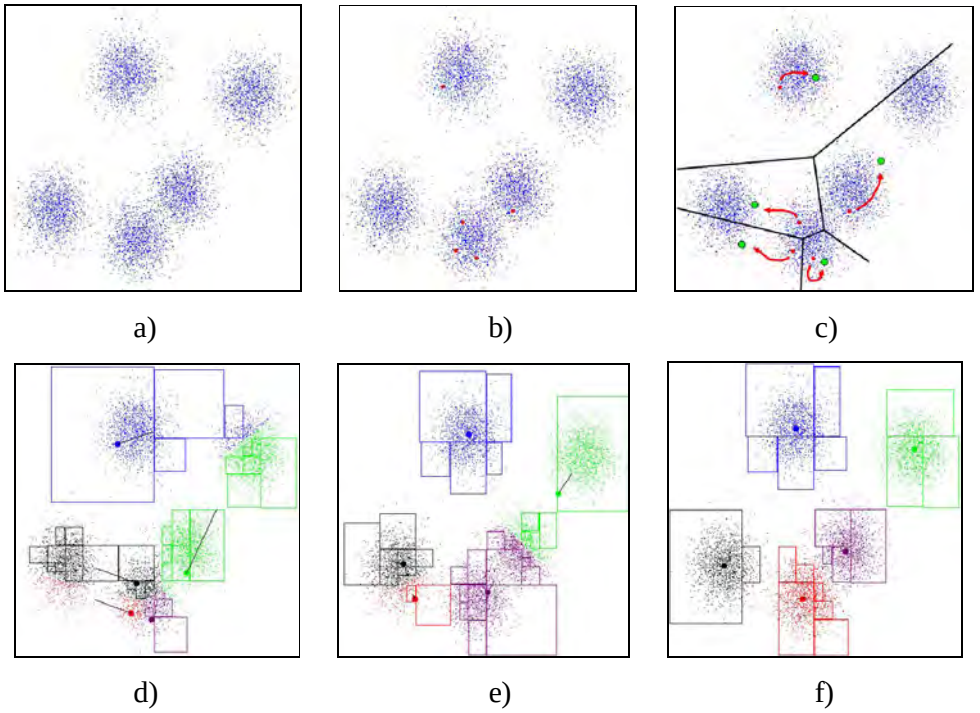
Segmentar un conjunto es subdividirlo de manera que se puedan obtener sus regiones o componentes constitutivas.
Existen dos enfoques principales:

- Segmentar por discontinuidad: buscar los límites de regiones, que corresponden a transiciones abruptas de características.
- Segmentar por similitud: formar regiones de características parecidas.

El análisis de conglomerados o *clusters* es una técnica iterativa que permite agrupar valores de un grupo de datos por su parecido o similitud. Al final, se generan *k* centroides en la nube de datos que mejor agrupen conjuntos de valores.

Este procedimiento se ilustra con el ejemplo de la Figura 2, en donde se tiene al a) conjunto de datos. Se plantea inicialmente una hipótesis aleatoria [puntos rojos en b)]; se calcula entonces la distancia euclidiana de cada punto a cada uno de los centroides y se les asigna pertenencia al centroide más cercano (T. Chen *et al.*, 2009), generándose regiones de dominio, como se muestra en c). Se repite este procedimiento hasta alcanzar

un número de iteraciones convenido o un criterio de convergencia (como puede ser la poca variación entre el centroide actual y el siguiente), como se observa en la figura 2 d), e) y f).



a) b) c)

d) e) f)

Figura 2. Etapas del algoritmo de k-medias [A. W. Moore, 2001].

2.1.3 Búsqueda

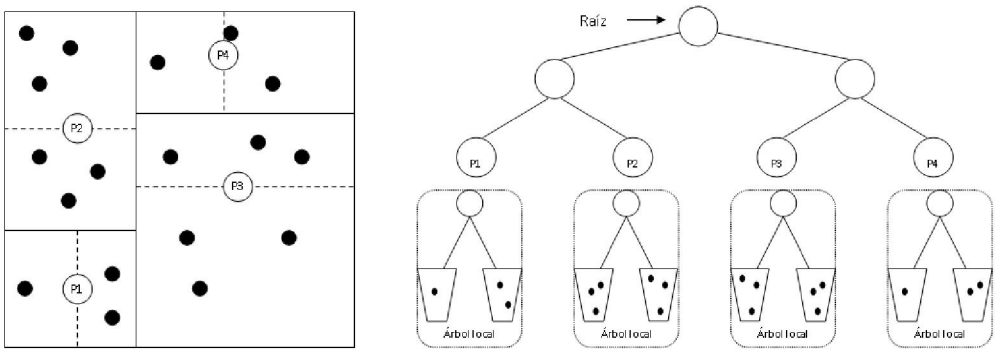
Si se tiene un conjunto de datos que pertenece a un proceso, en el que cada dato es un vector de dimensión *d*, éste se puede representar por los *k* centroides generados en un proceso de *clusterización*, por ejemplo por k-medias. Se dice que este conjunto de *k* centroides describe al proceso. Se puede saber además cuántos vectores pertenecen a cada centroide y obtener así su descriptor por histograma de ocurrencias.

Si se tiene otro conjunto de datos y se quiere saber si pertenece al mismo proceso, se puede saber al asignar a cada dato de este nuevo conjunto al centroide más cercano que describe a la referencia y obtener el histograma de ocurrencias; se comparan entonces los histogramas mediante la intersección para saber si corresponden los conjuntos de datos al mismo proceso.

En este caso, si se tiene muchos centroides y un conjunto de datos grande, el proceso de búsqueda de cada punto con su cluster más cercano consumirá una gran cantidad de tiempo. Para resolver esto, se emplea el algoritmo de búsqueda del vecino más cercano. La mayoría de los algoritmos rápidos utilizan una estructura de datos para almacenar el conjunto de entrenamiento, generalmente un árbol.

El árbol k-d (J. L. Bentley, 1975) es un árbol clásico de búsqueda del vecino más cercano. El nombre viene de *k-dimensional tree*, en el que la k representa la dimensión de los datos.

Los árboles k-d son árboles binarios, en el que cada nodo contiene información de una dimensión que divide en dos subconjuntos a los datos, generalmente con respecto a la media de los valores en esa dimensión. Los subconjuntos son de una dimensión $d_i - 1$, en donde $i = 1, 2, \ldots, k$, representa el nivel actual, figura 3. El proceso de búsqueda entonces se optimiza al descartar subconjuntos de centroides en lugar de centroides unitarios.



a) b)

Figura 3. Creación de subconjuntos por cada dimensión en un árbol k-d y su representación en un árbol de búsqueda.

2.2 Detección

En el nivel más bajo de abstracción, un objeto puede ser modelado empleando la imagen completa que lo contiene y comparado entonces, píxel por píxel, contra una imagen base del mismo. Para mejorar esta comparación, se pueden descartar detalles sin importancia utilizando plantillas (ignorando el fondo y la posición de la imagen), la

correlación normalizada (haciendo caso omiso de brillo) o mediante características de bordes (ignorando frecuencias espaciales bajas), etc. La abstracción en sí se manifiesta tanto en la representación del objeto y en la forma en que se compara con la imagen base (P. Chang *et al*., 1999).

El dilema viene a la hora de decidir qué abstraer. Se quiere ignorar suficientes detalles de la apariencia del objeto para que coincida con las imágenes base del objeto, pero no tantos como para que el algoritmo genere resultados falsos.

A su vez, se necesitan técnicas para la búsqueda de objetos cuando la distancia al objeto (escala) cambia de manera significativa, como sucede en el entorno de un robot móvil. Resulta complejo reconocer objetos que estén lejos de la cámara. Una forma de resolver este problema es con el uso del zoom óptico de la cámara y un algoritmo extractor de características por regiones para generar hipótesis sobre la presencia del objeto, antes de emplear algún algoritmo de reconocimiento. Al hacer zoom sobre un número de ubicaciones probables del objeto, los objetos lejos de la cámara pueden ser reconocidos (S. Ekvall *et al.*, 2006).

Otro enfoque para resolver este aspecto es el análisis multiescala, en el cual se emplean técnicas matemáticas en las que se extraen características espaciales que no varíen al cambio de escala (F. Cheikh *et al*., 2000; R. Willett *et al*., 2001; D. Donoho *et al*., 2002; B. Haar, 2003; M. Masood, 2005).

Para este trabajo, se emplean *campos receptivos*, en el que cada píxel de una imagen es representado por una combinación de su color y la respuesta a distintos filtros (extracción de característica). Aplicamos entonces *histogramas de coocurrencia* para representar a la imagen como una combinación de descriptores en un radio determinado, agregando así información geométrica de la imagen.

2.2.1 Histograma de Campos Receptivos

Cuando se trata de analizar la dispersión de una variable aleatoria, una representación adecuada es el histograma, que puede interpretarse como una representación discreta de

la función de densidad de probabilidad asociada a dicha variable. Se tiene una variable aleatoria X, entonces la probabilidad P(X = *a*) representa el número de veces que el valor *a* aparece en la dispersión.

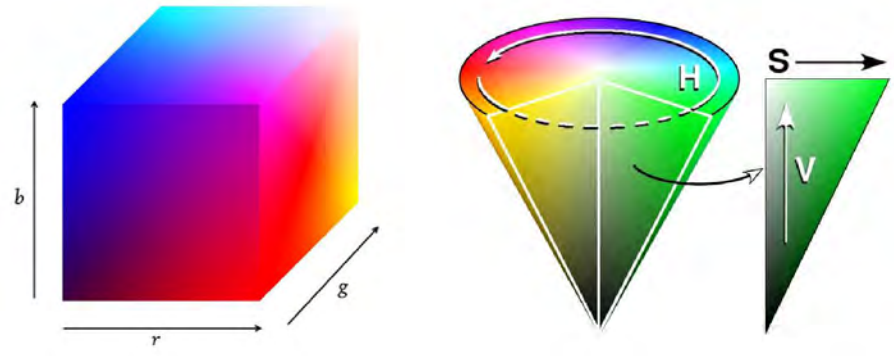
Un histograma de campos receptivos es una representación de la ocurrencia de múltiples descriptores dentro de una imagen; ejemplos de estos son la intensidad de color, la magnitud del gradiente o el Laplaciano de una imagen.

Se presentan a continuación los descriptores empleados para la tarea de detección de objetos. Estos descriptotes son invariantes a la traslación y rotación.

2.2.1.a Espacio de Color

La elección del espacio de color tiene una influencia significativa en la extracción de características de una imagen. El espacio de color más común es RGB, un espacio de 3 dimensiones en el que cada componente representa a uno de los colores primarios: R – Red, G – Green, B – Blue. Sin embargo, este espacio de color resulta poco robusto ante condiciones varientes en la intensidad de luz, texturas, etc.

El espacio de color HSV (Hue, Saturation, Value) ha mostrado mejores resultados en la tarea de segmentación pues conserva propiedades similares al sistema de visión humana (V. Mezaris *et al.*, 2004; T. Chen *et al.*, 2009). La figura 4 ilustra estos espacios.



a) b)

Figura 4. Espacios de color a) RGB y b) HSV.

2.2.1.b Magnitud del Gradiente y Laplaciano

Una característica aparte del color, es la presencia de bordes dentro de una escena: aquella región en donde aparece una fuerte variación en cambios de intensidad entre píxeles adyacentes. El fundamento para la detección de los bordes está en la aplicación del operador derivada en un entorno de vecindad, como se observa en la figura 5.

La derivada de una función nos indica el cambio de pendiente de ésta; en regiones homogéneas, este valor tiende a cero. Sin embargo, cuando hay un cambio drástico en intensidad, éste se refleja en un cambio drástico en la derivada. Si e aplica la segunda derivada, ésta nos indica el punto en donde hay cambios drásticos de intensidad.

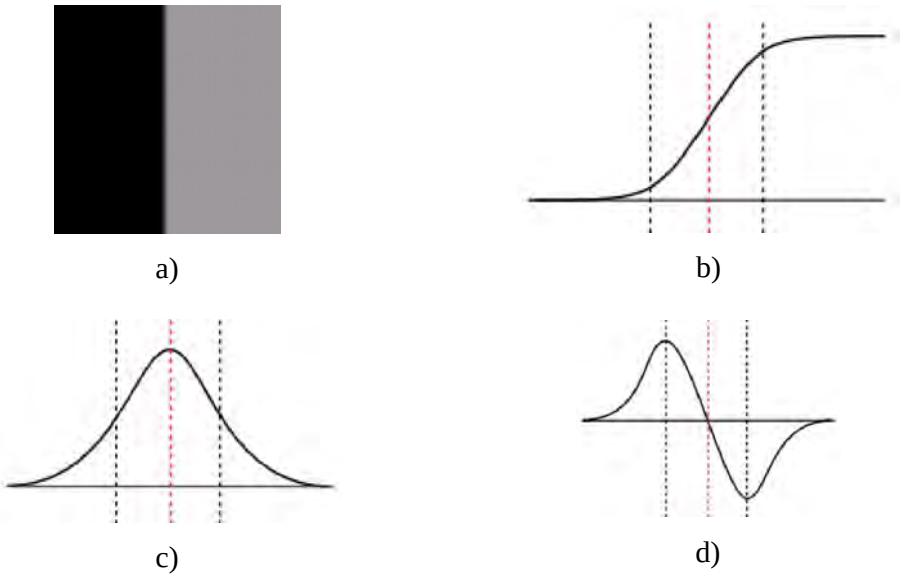


Figura 5. a) Dos regiones separadas por un borde vertical; b) nivel en escala de grises sobre una línea horizontal de la imagen; c) primera derivada y d) segunda derivada sobre el nivel de grises.

Cuando se aplica el operador gradiente en la imagen, se localizarán valores que tengan un gran valor, normalmente, en el módulo del gradiente. Por el contrario, al emplear el laplaciano se trata de detectar los bordes en una imagen (cambios drásticos de intensidad) en el que la segunda derivada, como se ilustra en la figura 5 d), se caracterizan por cambios de signo.

2.2.1.c Wavelets

La transformada Wavelet es una transformación sobre una función base con propiedades tanto en el tiempo como en la frecuencia. Esta transformada es eficiente para el análisis local de señales no estacionarias y de rápida transitoriedad, teniendo una representación de tiempo-escala, con un análisis de multi-resolución. El análisis de las frecuencias de mayor intervalo se realiza usando ventanas angostas y el análisis de las frecuencias de menor intervalo se hace utilizando ventanas anchas (Y. Sheng, 1996).

Se propone aquí emplear la transformada Wavelet de Haar (TWH), por ser la más rápida para el análisis en tiempo real. Esta consiste en implementar un banco de filtros, paso baja y paso alta, que se aplican a la imagen. Sea H (promedios) y G (diferencias) un filtro paso baja y paso alta, respectivamente, y el signo * la convolución, se tiene para el nivel n = 1, 2, 3, …, y una imagen D:

$$D_{n,0} = \left\lfloor H_x * \left\lfloor H_y * D_{n-1,0} \right\rfloor_{\downarrow 2,1} \right\rfloor_{1,2} \qquad D_{n,1} = \left\lfloor H_x * \left\lfloor G_y * D_{n-1,0} \right\rfloor_{\downarrow 2,1} \right\rfloor_{1,2}$$

$$D_{n,2} = \left\lfloor G_x * \left\lfloor H_y * D_{n-1,0} \right\rfloor_{\downarrow 2,1} \right\rfloor_{1,2} \qquad D_{n,3} = \left\lfloor G_x * \left\lfloor G_y * D_{n-1,0} \right\rfloor_{\downarrow 2,1} \right\rfloor_{1,2}$$

en donde ↓2,1 (o ↓1,2) denota un submuestreo sobre las filas (o columnas) por un factor de 2; D_{n-1} hace referencia a la imagen $D_{n-1,0}$ del nivel anterior, y $D_{n,i}$, i = 0, 1, 2, 3 el banco de filtros correspondiente. Cuando n = 1 entonces $D_{n-1} = D_0$ es la imagen original. En la figura 6 se ilustra este procedimiento.

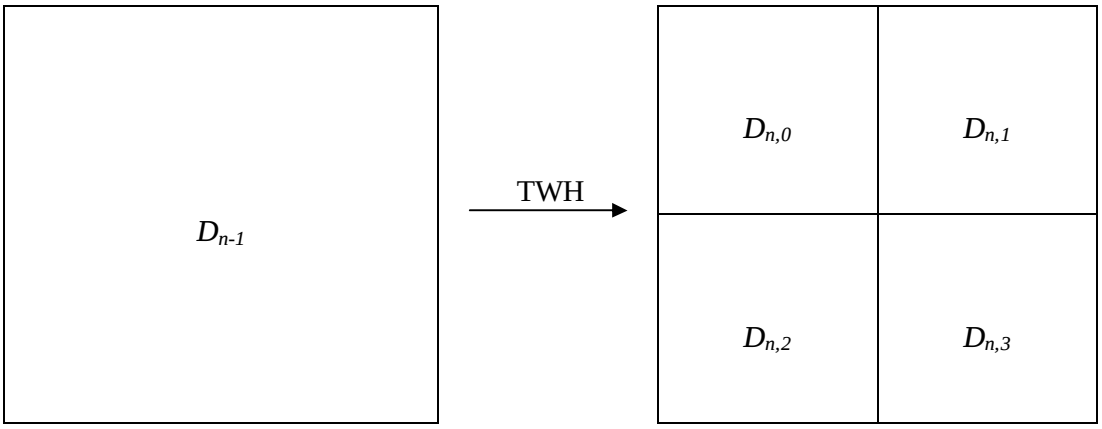


Figura 6. Aplicación de la transformada Wavelet a la imagen D_{n-1}. Se observa que, debido al submuestreo, el tamaño de la imagen resultante en cada caso es la mitad de la original.

Se tiene así que $D_{n,0}$ resulta ser una versión reducida en escala de la imagen original, mientras que $D_{n,1}$, $D_{n,2}$ y $D_{n,3}$ muestran los detalles de la imagen al aplicar los filtros en una dirección específica. $D_{n,1}$ en las verticales, $D_{n,2}$ en las horizontales y $D_{n,3}$ en las diagonales.

2.2.2 Histogramas de Co-ocurrencia de Campos Receptivos

Un histograma de co-ocurrencia de campos receptivos permite obtener, además de las características espaciales de un objeto, algunas de sus propiedades geométricas. Así pues, este tipo de histograma representa no solo qué tan común es la respuesta de cierto descriptor en la imagen, sino también qué tan común es que la respuesta a una combinación de descriptores dada ocurra en la misma zona.

En nuestro caso de estudio, cada píxel en la imagen se representa como un arreglo de sus descriptores $p_m = (A_m, B_m, C_m, \ldots)$, en donde A, B y C representan descriptores.

Entonces, como describe P. Chang *et al*. (1999), un histograma de co-ocurrencia (CH) representa el número de ocurrencias entre pares de arreglos separados por un vector en el plano de la imagen (Δx, Δy), como se muestra en la figura 7, y representado simbólicamente como CH(p_1, p_2, Δx, Δy).

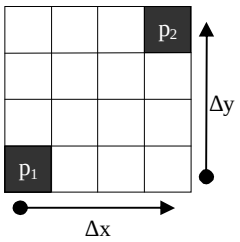


Figura 7. Arreglo de píxeles en una imagen. Los puntos p_1 y p_2 están separados por el vector (Δx, Δy).

Para hacer este histograma invariante a rotaciones en el plano de la imagen, se ignora la dirección de (Δx, Δy) y se monitorea solamente el valor de la magnitud $d = \sqrt{(\Delta x)^2 + (\Delta y)^2}$.

Además, clasificamos cada arreglo p_k en una serie de n_p clases $Q=\{q_1, q_2, ..., q_{n_p}\}$, y cuantizamos las distancias en una serie de n_d intervalos $D = \{[0,1),[1,2),...,[n_d - 1, n_d)\}$.

Para obtener una dispersión discreta de los arreglos de descriptores de cada imagen se emplea el algoritmo de clasificación por vecinos cercanos K-medias, en donde el número de cluster es n_c. Entonces, cada valor del histograma de co-ocurrencias se representa como CH(i, j, k), en donde i y j indican dos arreglos en Q, y k indica un intervalo de distancias en D.

Para la tarea de detección de objetos, a cada imagen base se le extrae su histograma de co-ocurrencia $CH_m(i, j, k)$. Luego, de la imagen en estudio, se obtiene su $CH_p(i, j, k)$, cuantizando ésta con el mismo algoritmo k-medias aplicando los centroides Q del modelo; finalmente, se calcula mediante su intersección (M. Swain *et al.*, 1991):

$$I_{pm} = \sum_{i=1}^{n_c} \sum_{j=1}^{n_c} \sum_{k=1}^{n_d} \min\left[\left(CH_p(i,j,k), CH_p(i,j,k)\right)\right]$$

La intersección indica qué tan bien se ajusta la imagen al modelo. Si la imagen corresponde a la del modelo, entonces la intersección será la suma de todos los valores del CH del modelo, o I_{mm}. Decimos que una imagen corresponde al modelo si la intersección sobrepasa un umbral $T = \alpha I_{mm}$.

2.3 Reconocimiento

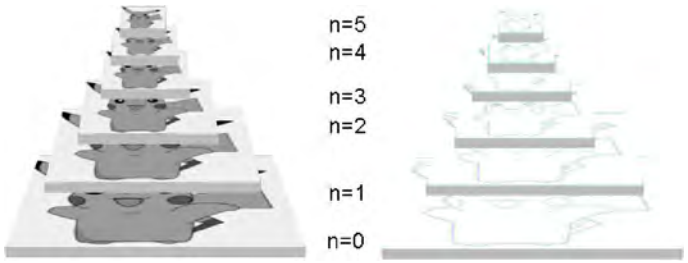
2.3.1 Transformación de características invariante a la escala.

Para determinar si dos imágenes son iguales se han de detectar puntos de interés en cada una de éstas y se obtendran características distintivas de cada uno de éstos; luego se comparan uno a uno: si se excede un umbral específico de puntos con características similares, se considera que ambas imágenes son la misma.

Se observa pues, que en los sistemas de empatado de imágenes se requiere primero de la detección de puntos de interés, la descripción de estos, su posterior comparación y la toma de decisiones. Cuando todas las imágenes son similares en naturaleza (misma

escala, orientación, etc.) puede funcionar un detector de esquinas sencillo. Pero cuando tenemos imágenes con diferentes escalas y rotaciones, se requiere de características invariantes a dichos cambios. Se emplea entonces un detector de puntos sobresalientes (figura 9) con una combinación de filtros de Gauss y de Laplace a distintas escalas (figura 8).

D. Lowe (2004) presenta su transformación de características invariante a la escala (*SIFT* por sus siglas en inglés), el cual es un método para extraer características distintivas invariantes de las imágenes que pueden ser usadas para ejecutar un empatado confiable contra diferentes vistas de un objeto o escena. Estas características aplicadas a cada punto sobresaliente son invariantes a escala y rotación, y muestran un nivel alto de empatado incluso ante distorsiones, cambio de punto de vista, ruido, y cambios de iluminación.



a) b)

Figura 8. Pirámides a) de Gauss y b) de Laplace para detectar puntos sobresalientes presentes en distintas escalas.

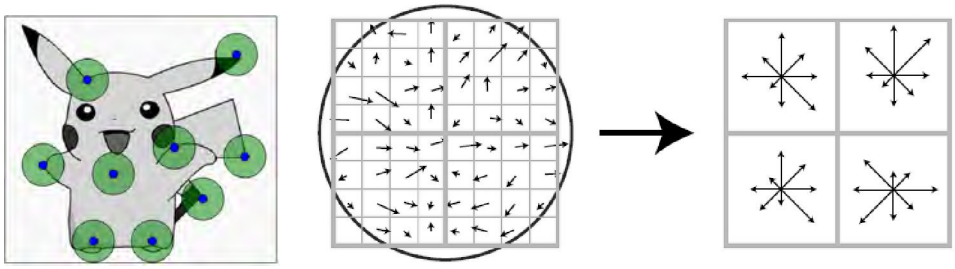


Figura 9. a) Puntos sobresalientes (azul) con su región de interés (verde) y b) descriptor SIFT para cada punto.

En este trabajo se emplea la aplicación desarrollada por A. Pacheco-Ortega (2011), en la que considera la gran dimensionalidad de los descriptores tipo SIFT, por lo que desarrolla técnicas para el almacenaje y búsqueda basadas en los trabajos de J. L. Bentley (1975) y J. S. Beis *et al*. (1997). Bentley desarrolló árboles de búsqueda que

permiten realizar comparaciones con un cúmulo de vectores almacenados en ellos de manera sistemática. Posteriormente Beis realizó modificaciones a los mismos que permitieron acelerar las búsquedas en espacios de alta dimensionalidad.

2.4 Cámaras RGB-D

Las cámaras RGB-D son un novedoso sistema que captura imágenes RGB junto con la información en profundidad por cada píxel. Este tipo de cámaras se basan en patrones de luz estructurada infrarroja, para generar una estimación de la profundidad asociada con cada píxel RGB. Cámaras pequeñas y de gran calidad han sido desarrolladas para el mundo de los videojuegos, por lo que pueden ser adquiridas rápidamente y a un bajo costo. La figura 10 muestra cómo se obtiene esta información.

Figura 10. a) Imagen RGB, y b) información de profundidad. Los sistemas actuales capturan imágenes con una resolución de 640x480 a 30 cuadros por segundo. Los pixeles negros en la imagen de la derecha no tienen valor de profundidad, debido principalmente a oclusión, máxima distancia, al material o ángulo relativo de la superficie.

En el contexto de la robótica, estas cámaras son usadas para construir mapas 3D de ambientes interiores. Dichos mapas pueden ser utilizados para navegación, manipulación, localización, entre otras aplicaciones. Sin embargo, aunque las nubes de puntos 3D permiten perfectamente una reconstrucción 3D, cuadro por cuadro, éstas ignoran información visual muy valiosa, información que no deja de obtenerse a través de cámaras RGB.

Aún cuando las cámaras RGB-D proveen una gran cantidad de información 3D del ambiente, presentan algunos inconvenientes que hacen aún difícil su aplicación en

detección y reconocimiento de objetos: Proveen información de profundidad en un intervalo limitado (más de 40cm); estos valores de profundidad son mucho más ruidosos que aquellos obtenidos por láser (especialmente en los contornos de los objetos); y su campo de visión (~60°) está, por mucho, más limitado que los de cámaras especializadas o escáneres láser usados en estas aplicaciones (~180°) (P. Henry *et al.*, 2010). En este trabajo se emplea la cámara desarrollada por PrimeSense incluida en el dispositivo Microsoft Kinect™.

Capítulo III. Implementación

3.1 Campos receptivos

El primer paso para implementar la solución propuestas, es la programación de los descriptores. Se empleará en este trabajo las funciones de OpenCV, unas bibliotecas especializadas para el procesamiento de imágenes.

La operación con los descriptores se obtendrá de las imágenes en escala de grises o en el espacio de color HSV, según se requiera, obtenidas éstas de la original en el espacio de color RGB, empleando los estándares de OpenCV basados en los modelos presentados por A. R. Smith (1978). Para el caso de convertir RGB a escala de grises, se emplea el estándar NTSC Rec.601:

$$\text{imgGray} = 0.299\,R + 0.587\,G + 0.114\,B$$

en donde de la imagen original se tienen los canales de color por separado, i.e. R, G, B; *imgGray* es la imagen en escala de grises (esto es, un solo canal). Para el caso de convertirla a HSV, se emplea el estándar de OpenCV, en donde:

$$V \leftarrow \max(R,G,B)$$

$$S \leftarrow \begin{cases} \frac{V-\min(R,G,B)}{V} & si \quad V \neq 0 \\ 0 & si \quad V = 0 \end{cases}$$

$$H \leftarrow \begin{cases} 60(G-B)/S & si \quad V = R \\ 120+60(B-R)/S & si \quad V = G \\ 240+60(R-G)/S & si \quad V = B \end{cases}$$

Si $H < 0$, entonces $H = H + 360$.

3.1.1 Descriptores

Una de las características que definen a un objeto son los bordes. Como se observó, para extraerlos de la imagen se emplean primeras y segundas derivadas.

La primera derivada (**operador Gradiente**) de una imagen digital está basada en varias aproximaciones del gradiente en 2D. El gradiente de una imagen *f*(*x*, *y*) en la posición (*x*, *y*) está definido como el vector:

$$\nabla f = \begin{bmatrix} f_x \\ f_y \end{bmatrix} = \begin{bmatrix} \partial f / \partial x \\ \partial f / \partial y \end{bmatrix}$$

El vector gradiente apunta a la dirección con la velocidad de cambio de *f* máxima en las coordenadas (*x*, *y*).

La magnitud y ángulo del gradiente están dados por:

$$\|\nabla f\| = \sqrt{f_x^2 + f_y^2} \approx |f_x| + |f_y| \qquad \theta = \arctan\left(\frac{f_x}{f_y}\right)$$

Una aproximación discreta al gradiente (filtro Sobel) queda:

$$\frac{\partial f(x,y)}{\partial x} \approx f(x+1,y) - f(x-1,y) \qquad \frac{\partial f(x,y)}{\partial y} \approx f(x,y+1) - f(x,y-1)$$

$$\text{Filtro Sobel: horizontal} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}, \quad \text{vertical} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

La segunda derivada (**operador Laplaciano**) de una imagen *f*(*x*, *y*) en la posición (*x*, *y*) está definido como el vector:

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

La aproximación discreta queda:

$$\frac{\partial^2 f(x,y)}{\partial x^2} \approx f(x+1,y) - f(x-1,y) - 2f(x,y)$$

$$\frac{\partial^2 f(x,y)}{\partial y^2} \approx f(x,y+1) - f(x,y-1) - 2f(x,y)$$

Filtro laplaciano 2D: $\begin{bmatrix} 0 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$

Para eliminar el ruido que pueda existir en la imagen y optimizar la detección de bordes (pues el ruido produce cambios drásticos de intensidad en píxeles adyacentes) se aplican los operadores Gradiente y Laplaciano a la imagen convolucionada con un filtro Gaussiano de la forma:

$$G(x,y) = -\exp\left(\frac{-x^2+y^2}{2\sigma^2}\right).$$

Las figuras 11 y 12 ilustran la aplicación de estos filtros. Se tiene entonces:

```
int sigma = 2;
función filtro_gauss( imgGray, imgGauss, sigma );
función filtro_gradiente( imgGauss, imgGrad );
función filtro_laplaciano( imgGauss, imgLap );
```

a) b) c)

Figura 11. a) Imagen original; b) imagen en escala de grises; c) imagen después de aplicar el filtro Gaussiano.

a) b)

Figura 12. a) Gradiene y b) Laplaciano de la imagen.

Estos valores se guardan en una matriz con *k* columnas, en donde *k* representa el número de descriptores empleados en cada caso de estudio. En este caso, *k* = 5.

```
//Obtener vector de características
//Entradas: imgHSV, imgGrad, imgLap
//Salida: Vector RF_VECTOR

int i, j, k = 0;
int Psize = 5; //Número de características

for ( i = 0; i < src->height; i++)
    for ( j = 0; j < src->width; j++)
    {
        RF_VECTOR [k * Psize + 0] = imgHSV->H( i, j );
        RF_VECTOR [k * Psize + 1] = imgHSV->S( i, j );
        RF_VECTOR [k * Psize + 2] = imgHSV->V( i, j );
        RF_VECTOR [k * Psize + 3] = imgGrad( i , j );
        RF_VECTOR [k * Psize + 4] = imgLap( i , j );
        k++;
    }
```

3. 2 Wavelets

Como se observó, una transformación Wavelet consiste en un banco de filtros tanto paso altas, como paso bajas. En el caso de la transformada Wavelet Haar, se trata de promedios y diferencias, por lo que se tiene el filtro paso bajas H = { ½ , ½ } y paso altas G = { –½ , –½ }. Para aplicar este filtro a la imagen completa se genera la siguiente matriz:

$$W_M \bar{v} = \sqrt{2} \cdot \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & \text{L} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & \text{L} \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & \text{L} \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & \text{L} \\ -\frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & \text{L} \\ 0 & 0 & -\frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & \text{L} \\ 0 & 0 & 0 & 0 & -\frac{1}{2} & \frac{1}{2} & 0 & 0 & \text{L} \\ 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{2} & \frac{1}{2} & \text{L} \\ \text{M} & \text{M} & \text{M} & \text{M} & \text{M} & \text{M} & \text{M} & \text{M} & \text{O} \end{bmatrix} \cdot \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \\ v_8 \\ \text{M} \end{bmatrix} = \begin{bmatrix} (v_1 + v_2)/2 \\ (v_3 + v_4)/2 \\ (v_5 + v_6)/2 \\ (v_7 + v_8)/2 \\ (v_2 + v_1)/2 \\ (v_4 + v_3)/2 \\ (v_6 + v_5)/2 \\ (v_8 + v_7)/2 \\ \text{M} \end{bmatrix}$$

en donde $W_M^{-1} = W_M^T$, dado que la transformada Wavelet es una transformación ortogonal.

Para el caso de 2D, se tiene:

$$B = WAW^T = \begin{bmatrix} H \\ G \end{bmatrix} A \begin{bmatrix} H \\ G \end{bmatrix}^T = \begin{bmatrix} H \\ G \end{bmatrix} A \begin{bmatrix} H^T \\ G^T \end{bmatrix} = \begin{bmatrix} HA \\ GA \end{bmatrix} \begin{bmatrix} H^T \\ G^T \end{bmatrix} = \begin{bmatrix} HAH^T & HAG^T \\ GAH^T & GAG^T \end{bmatrix}$$

En el primer cuadrante de la expresión anterior, HAH^T, se observa que se aplica un promedio HA en las columnas, y las filas de este resultado luego se promedian mediante H^T, con lo que se genera una versión comprimida de la imagen original.

Análisis similares para cada uno de estos productos nos lleva a:

$$HAH^T \Rightarrow \begin{bmatrix} a & b \\ c & d \end{bmatrix} \rightarrow 2 \cdot (a+b+c+d)/4$$

$$HAG^T \Rightarrow \begin{bmatrix} a & b \\ c & d \end{bmatrix} \rightarrow 2 \cdot (b+d-a-c)/4$$

$$GAH^T \Rightarrow \begin{bmatrix} a & b \\ c & d \end{bmatrix} \rightarrow 2 \cdot (c+d-a-b)/4$$

$$GAG^T \Rightarrow \begin{bmatrix} a & b \\ c & d \end{bmatrix} \rightarrow 2 \cdot (b+c-a-d)/4$$

Para calcular esta transformada en distintos niveles, se emplea la imagen escalada ontenida con HAH^T, lo que permite un análisis multi-escala de la misma. Se tiene así la siguiente función para calcular la transformada Wavelet de una imagen (figura 13):

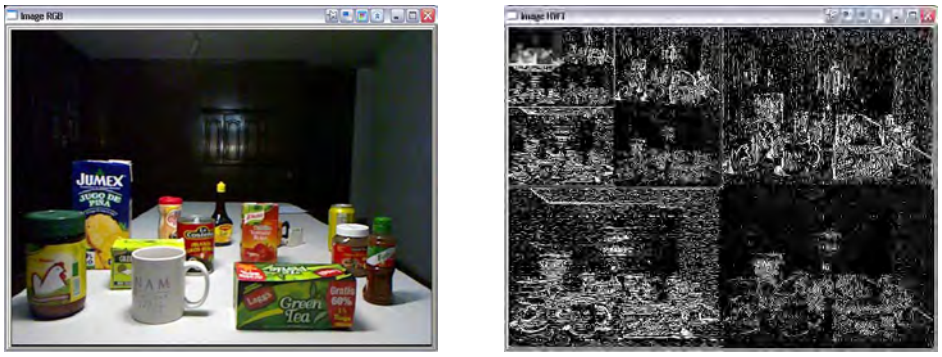
```
//Calcular la Transformada Wavelet de Haar en un nivel específico
//Entrada: SRC, DST, Level
   L = Level^2; //Escala de las dimensiones del primer cuadrante de acuerdo al nivel
   int H    = (src->height) / Level;
   int W    = (src->width) / Level;

   int a, b, c, d, i, j;

   int deltaX = (int) (W / 2); //Escala de la salida después del sub-muestreo
   int deltaY = (int) (H / 2); //Escala de la salida después del sub-muestreo

   for ( i = 0; i < (H - 1); i++ )
      for ( j = 0; j < (W - 1); j++ )
      {
         a = SRC [ i , j ];
         b = SRC [ i , (j+1) ];
         c = SRC [ (i+1) , j ];
         d = SRC [ (i+1) , (j+1) ];

         DST[ (i/2) , (j/2) ] = (a + b + c + d)/4;
         DST[ (i/2) , ((j/2) + deltaX) ] = (b + d - a - c)/8+128;
         DST[ ((i/2) + deltaY) , (j/2) ] = (c + d - a - b)/8+128;
         DST[ ((i/2) + deltaY) , ((j/2) + deltaX) ] = (b + c - a - d)/8+128;
      }
```



a) b)

Figura 13. a) Imagen original y b) imagen después de aplicar tres niveles de la transformada Wavelet Haar.

El descriptor de texturas se extrae a partir de la varianza $\sigma_{n,i}^2$ de los coeficientes $c_{n,i}$ de los detalles de la imagen obtenidos a partir de HAG^T, GAH^T y GAG^T en todos sus niveles n.

Para representar la textura de una imagen A, el vector del descriptor de textura queda:

$$T_{HWT} = \left[\left(\mu_{1,1},\sigma_{1,1}^2\right),\left(\mu_{1,2},\sigma_{1,2}^2\right),\left(\mu_{1,3},\sigma_{1,3}^2\right),\mathrm{L},\left(\mu_{M_{\max},3},\sigma_{M_{\max},3}^2\right)\right],$$

en donde $M_{\max}$ indica la máxima escala. En este trabajo $M_{\max} = 3$, y se tiene un vector de 9 componentes para describir a la textura.

Para comparar la textura entre dos imágenes, se obtiene la distancia euclidiana entre sus vectores respectivos.

3.3 Histograma de Co-ocurrencia de Campos Receptivos

El proceso de detección consiste en comparar las características de una imagen base con una imagen de prueba. Sin embargo, comparar toda la matriz de características resulta ineficiente, por lo que de ésta, como primer paso, se realiza una clasificación, obteniendo un número de clusters definido por el usuario.

```
//Calcular la clusterización por k-medias
//Entrada: points, MAX_CLUSTERS
//Salida: clusters, centers
    función K-Medias (points, MAX_CLUSTERS, clusters, centers);
```

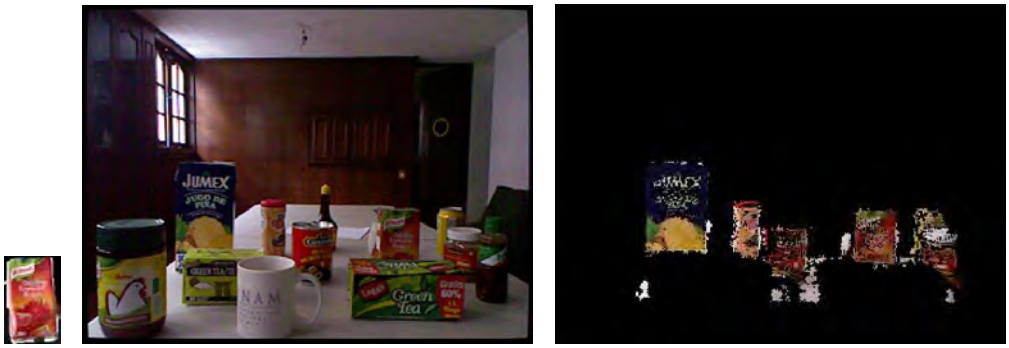
en donde points es la matriz de campos receptivos (un vector de características por fila, con tantas filas como pixeles en la imagen), MAX_CLUSTERS es el número de clusters a buscar, clusters contiene el índice del cluster al que pertenece cada píxel de la imagen, y centers contiene el valor del cluster en sí.

$$\text{points} = \begin{bmatrix} (H \quad S \quad V \quad |\nabla L| \quad |\nabla^2 L|)_{1,1} \\ (H \quad S \quad V \quad |\nabla L| \quad |\nabla^2 L|)_{1,2} \\ \dots \\ (H \quad S \quad V \quad |\nabla L| \quad |\nabla^2 L|)_{1,n} \\ (H \quad S \quad V \quad |\nabla L| \quad |\nabla^2 L|)_{2,1} \\ (H \quad S \quad V \quad |\nabla L| \quad |\nabla^2 L|)_{2,2} \\ \dots \\ (H \quad S \quad V \quad |\nabla L| \quad |\nabla^2 L|)_{m,1} \\ (H \quad S \quad V \quad |\nabla L| \quad |\nabla^2 L|)_{m,2} \\ \dots \\ (H \quad S \quad V \quad |\nabla L| \quad |\nabla^2 L|)_{m,n} \end{bmatrix} \text{ è } \text{cluster} = \begin{bmatrix} c_{1,1} \\ c_{1,2} \\ \dots \\ c_{1,n} \\ c_{2,1} \\ c_{2,2} \\ \dots \\ c_{m,1} \\ c_{m,2} \\ \dots \\ c_{m,n} \end{bmatrix}$$

en donde c = 0, 1, 2, …, MAX_CLUSTERS – 1.

Con estos centros obtenidos en el vector centers se clasifican los píxeles de la imagen de prueba. Los píxeles que no pertenezcan a alguno de los centros se descartan. Para determinar la pertenencia a un centro, durante el entrenamiento se obtiene la distancia media de todos los puntos que pertenecen al mismo cluster y, en la imagen en la que se busca el objeto, se obtiene su matriz campos receptivos. Para cada vector de características se le asocia el cluster más cercan, considerando la distancia Euclidiana, y si es menos que 1.5 veces la distancia media obtenida durante el entrenamiento, se considera que el punto puede pertenecer al objeto.

Con este procedimiento, se obtiene una imagen cuantizada, que permite realizar la detección del objeto de forma más eficiente, pues se limita el espacio de búsqueda a regiones con características de color similares y se descartan las regiones en negro. La figura 14 muestra el resultado del proceso de cuantización aplicado para una imagen completa para el proceso de búsqueda específica de una caja de color rojo (salsa de tomate).

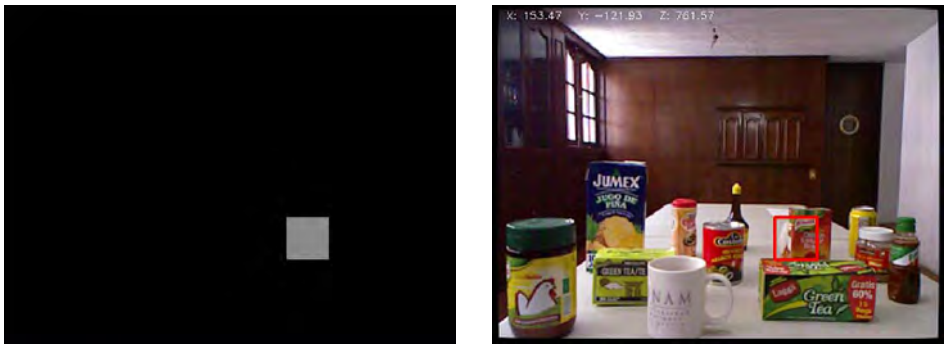


a) b) c)

Figura 14. a) Objeto a buscar; b) imagen de búsqueda; c) imagen después del proceso de cuantización.

Con la imagen cuantizada, se inicia un proceso de ventaneo, del que se extraen los histogramas de co-ocurrencia de campos receptivos (RFCH) de la imagen de referencia y de la ventana de búsqueda, así como el vector del descriptor de textura T_{HWT}. Luego, se realiza la intersección de histogramas I_{pm}, y se determina la distancia euclidiana entre los vectores de textura de la ventana y la referencia. A la ventana se le asigna un valor en escala de gris $255*I_{pm}$, si $I_{pm} > \alpha I_{mm}$.

En la figura 15 se muestra el resultado para el caso de la salsa de tomate. A la imagen resultante del proceso de cuantización (figura 14, c)) se le aplica el proceso de ventaneo. Se observa que objetos con similar distribución de color, pero textura notoriamente distinta se descartan.



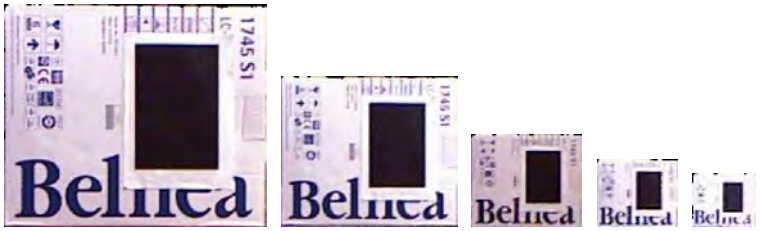
c) d)

Figura 15. a) resultado de la búsqueda por ventaneo; y b) correspondencia con la imagen completa.

3. 4 Función Distancia-Escala (Kinect)

Para acelerar el proceso de detección, se aprovechan las características del Kinect que nos da información en profundidad, lo que permite obtener el escalamiento apropiado del objeto a buscar de acuerdo a la distancia. Para esto, es necesario obtener la función de escala de acuerdo a la profundidad.

Se trabajó con un objeto de dimensiones conocidas, el cual se fue alejando a distancias conocidas y capturando su imagen (figura 16). Siendo un objeto rectangular, de la imagen obtenida se obtiene el número de milímetros por píxeles en el lado vertical y horizontal del objeto, luego se obtiene la raíz del área del mismo (con los valores anteriores) para obtener una representación lineal de la variación del objeto en la imagen de acuerdo a la profundidad.



a) b) c) d) e)

Figura 16. Objeto de dimensiones conocidas a distintas profundidades [mm]: a) 1052; b) 1574; c) 2532; d) 3475; e) 4521.

A partir de los datos anteriores se obtiene el siguiente modelo lineal:

$$y = 0.001716x + 0.2735,$$

en donde ***y*** representa el tamaño del objeto en la imagen en mm/píxel a una profundidad ***x*** en mm (figura 17).

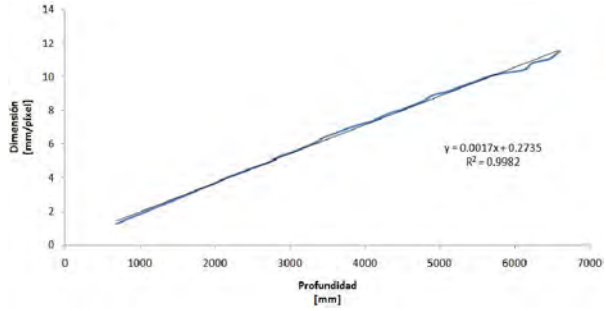


Figura 17. Cambio de dimensiones en mm/píxel de un objeto a distintas profundidades.

Entonces, para obtener el factor de escala de una imagen a una profundidad arbitraria x_i con respecto a una imagen de referencia con una profundidad fija x_0, como se muestra en la figura 18, se emplea la siguiente relación:

$$Escala = \frac{y_0}{y_i} = \frac{0.001716x_0 + 0.2735}{0.001716x_i + 0.2735}$$

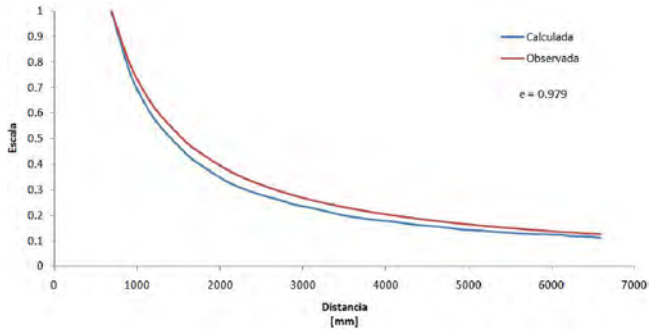
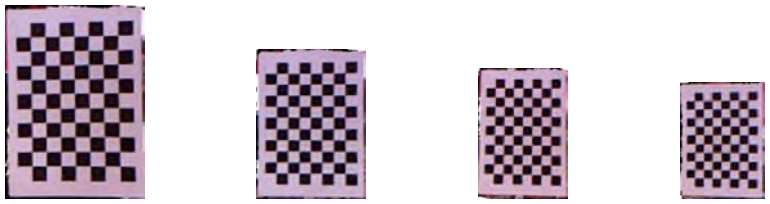


Figura 18. Escalamiento de la imagen de patrón a distintas profundidades, por observación (rojo) y a través del modelo experimental (azul). A la distancia de referencia de 691mm el factor de escala es 1, a partir de ahí, si la distancia de la cámara al objeto se incrementa, la escala (el tamaño del objeto observado en la imagen) se reduce y viceversa.

Este valor de escala indica por cuánto hay que re-dimensionar una imagen tomada a una determinada distancia para obtener su representación a la profundidad deseada.

Para comprobar el modelo, se obtuvieron varias tomas de un objeto con dimensiones conocidas a distintas distancias (figura 19) y se obtuvo el factor de escala tanto de forma experimental como a través del modelo (figura 20), obteniendo un error de 0.9815.



a) b) c) d)

Figura 19. Objeto de dimensiones conocidas a distintas profundidades: a) 736; b) 948; c) 1095; d) 1169.

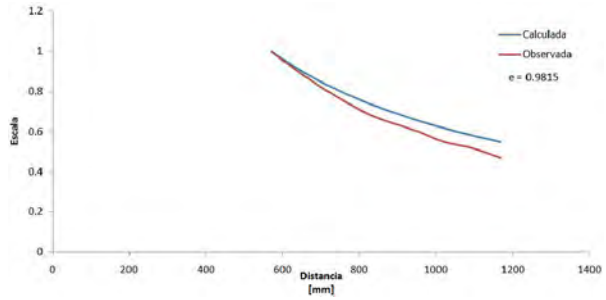


Figura 20. Ejemplo del escalamiento de una imagen a distintas profundidades, por observación (rojo) y a través del modelo experimental (azul).

3. 5 3D-Signature

Como se observó, de una imagen RGB-D, para cada píxel se tiene su información de profundidad la cual, a través de las bibliotecas de OpenNI.

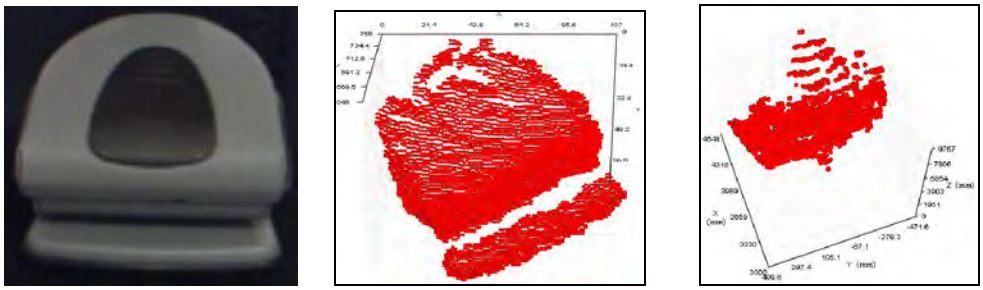
Para corregir la distorsión óptica en las imágenes RGB, se crearon las rutinas en la biblioteca *Undistorter.h*, que se pueden observar en el apéndice de este documento. Con esto, se tiene la información espacial y RGB de una escena respecto a la posición de la cámara, como se observa en la figura 21.

a) b) c)

Figura 21. a) Imagen RGB; b) imagen RGB corregida y su c) profundidad.

Se tiene entonces la función getFrame, que nos devuelve la imagen RGB y en profundidad, los parámetros intrínsecos y extrínsecos de la cámara y la posición en el espacio de cada píxel respecto a la posición de la cámara en mm.

Con esto se obtienen las características ópticas y espaciales de los objetos en el rango de visión de la cámara respecto a ésta, como se muestra en la figura 22.



a) b) c)

Figura 22. a) Imagen RGB y b) su profundidad Z en [mm] para cada píxel en XY. En la imagen c) se observa una vista parcial de su posición en el espacio XYZ en [mm].

3.5.1 Descriptor 3D de un punto.

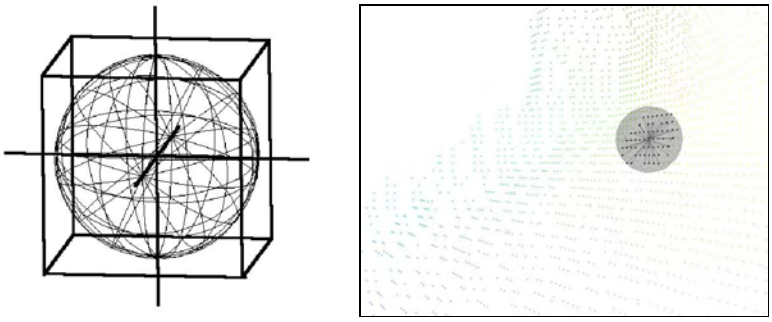
Con esta información, para incrementar la tasa de detección y aproximarla a la de reconocimiento, se emplea la información 3D del objeto. Para esto, primero se crea el descriptor de un punto:

1. Se genera una esfera de radio *r* centrada en el punto *p* del mapa de profundidad.

2. Se obtiene el histograma de distancias del punto en el centro de la esfera a todos los puntos contenidas en ésta.

Como la distancia es un valor real entre 0 y *r*, se realiza una cuantización dependiendo de cuántos niveles se quieran para el histograma de distancias. Se considera entonces como descriptor del punto un vector con los valores del respectivo nivel del histograma.

Para esto, se considera un cubo que contenga a la esfera de radio *r* centrada en el punto *p*, entonces se calcula la distancia *d* de *p* a cada punto dentro del cubo, considerándose para el conteo en el histograma solo los puntos cuya distancia $d \leq r$, como se muestra en la figura 23.



a) b)

Figura 23. a) Cubo que contiene a la esfera de radio *r* centrada en el punto *p*. Cálculo de la distancia *d* en un punto cualquiera del mapa de profundidad.

Los valores de distancia se almacenan en la matriz de distancias y de esta se obtiene su histograma de acuerdo a los parámetros de cuantización.

3.5.2 Descriptor 3D del mundo – vocabulario.

Como se observó, con el descriptor de un punto se obtiene información espacial del punto y el área alrededor de éste, lo que resulta de utilidad al describir objetos con geometrías variadas. Se requiere entonces generar una serie de vectores de referencia (clusters) en qué clasificar cada descriptor. Llamaremos a cada uno de esto clusters *palabra*, y al conjunto de todas ellas *vocabulario*.

Se captura entonces el mapa de profundidad de un escenario con múltiples geometrías, tal que contenga todas las formas que un objeto pudiera contener, como se muestra en la figura 24.

Figura 24. Arreglo típico de una escena compleja.

Con este mapa de profundidad complejo, de cada uno de los puntos que lo conforma se obtiene su descriptor en forma de vector (histograma de distancias del punto *p* a todos los puntos contenidos en una esfera de radio *r*), y se almacenan en una matriz *Wpoints* que contiene a todos los descriptores de la escena (también referida como *mundo*).

De esta matriz con todos los descriptores del *mundo*, se obtienen los clusters que mejor la representen por medio del método de k-medias. Estos clusters se almacenan en la matriz *centers*.

```
//Calcular la clusterización por k-medias
//Entrada: Wpoints, MAX_CLUSTERS
//Salida: clusters, centers
      función K-Medias (Wpoints, MAX_CLUSTERS, Wclusters, Wcenters);
```

A la matriz de centroides *Wcenters* se de denominará *vocabulario*, y a cada uno de los vectores de características contenido en *Wcenters* se le denominará *palabra*, se tienen *MAX_CH* palabras. Finalmente, para optimizar la asociación de un nuevo punto a uno de estos clusters, se emplean árboles de búsqueda KD. Para ello, con es necesario primero crear la estructura de búsqueda de una palabra en el vocabulario (referido como *diccionario*).

```
//Crear árbol KD
//Entrada: centers
//Salida: features
        Wfeatures = CreateKDTree(Wcenters);
```

3.5.3 Descriptor 3D de un objeto – frase.

Para describir un objeto, se obtiene el descriptor para cada uno de los puntos que lo conforman en la matriz *Opoints* y se les asocia el número del cluster correspondiente (palabra). Estos valores se almacenan en el vector *Oclusters*.

```
//Encontrar el vecino más cercano a Wcenters de los vectores en Opoints por medio de la
//búsqueda por árboles KD y almacenar el índice en Ocluster
//Entrada: Wfeatures, Opoints
//Salida: Ocluster, Odistances
         FindFeatures(Wfeatures, Opoints, Oclusters, distances);
```

Luego, se cuenta cuántas veces aparece cada palabra en el objeto, y se almacena en *clustersH*, y se normaliza en *HistObj*. A este histograma normalizado lo nombraremos

frase. Una frase está conformada por una secuencia de palabras, que en otra frase pueden o no aparecer en proporciones distintas, como se muestra en la figura 25.

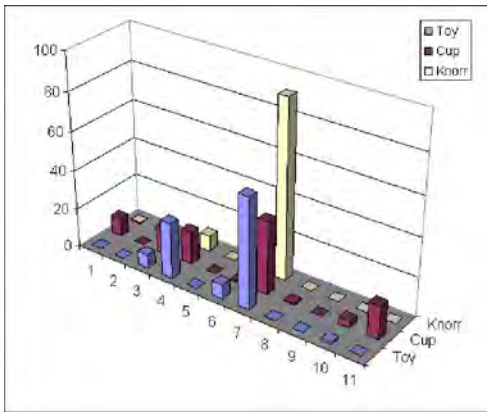


Figura 25. Se muestra la *frase* correspondiente a distintos objetos, en donde se pueden observar las diferencias entre ellas.

Capítulo IV. Resultados experimentales

4.1 Procedimiento

El proceso de detección requiere de una etapa de entrenamiento de cada objeto que se desee localizar. Para esto, se entrena un objeto en cada una de las etapas descritas. Se obtiene el árbol k-d de los centroides obtenidos de la clusterización de todos sus vectores de características, así como el histograma de co-ocurrencias de los mismos. A su vez, el vector de medias y varianzas de los coeficientes en cada uno de los niveles de la transformada wavelet, como descriptor de texturas. Finalmente, con l información de profundidad se obtiene la frase que describe al objeto, así como la distancia media a la que fue tomada la imagen, para poder hacer las variaciones en escala de acuerdo a la profundidad de búsqueda. La figura 26 ilustra este proceso.

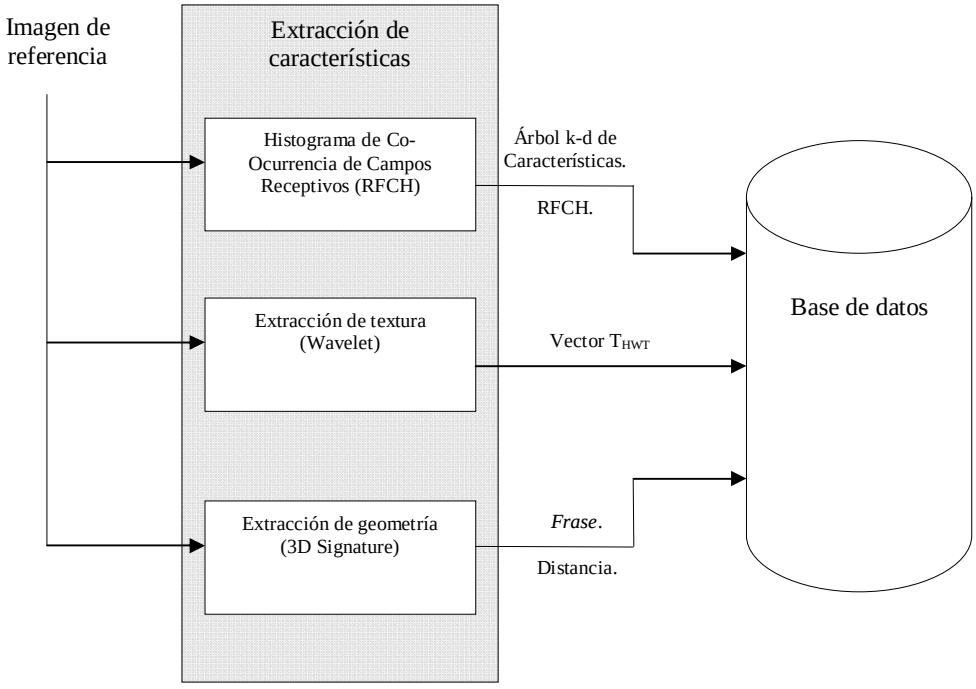


Figura 26. Diagrama de operación del proceso de entrenamiento.

Para el caso de detección, la figura 27 muestra las distintas etapas. Pasando primero por un análisis por ventanas de los valores de campos receptivos y textura en toda la

imagen, y un posterior análisis en profundidad de las regiones más probables de contener un objeto.

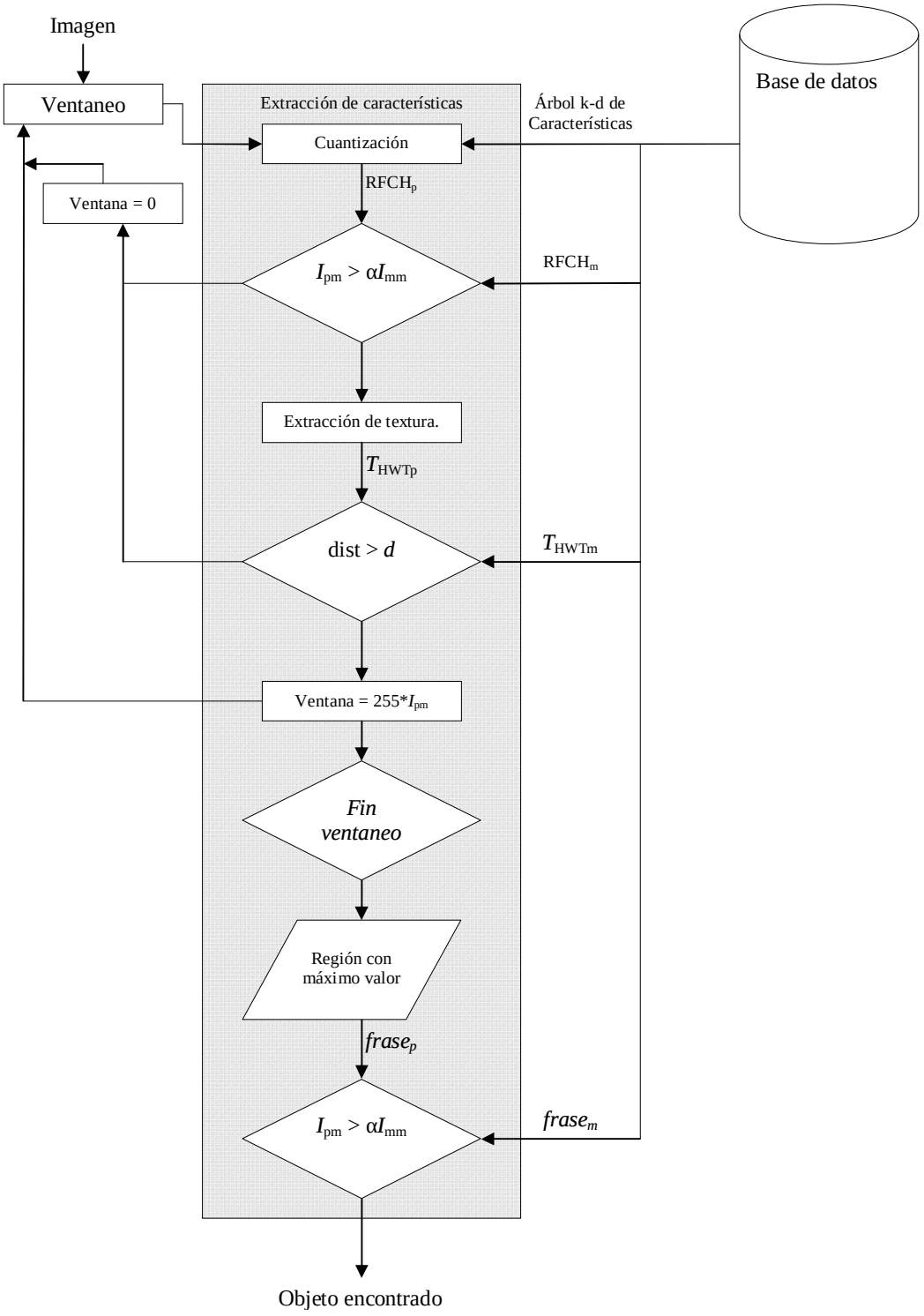


Figura 27. Diagrama de operación del proceso de detección.

4.2 Resultados

Para el caso del detector de objetos empleando el algoritmo de co-ocurrencias se observó un alto número de falsos positivos en zonas con distribución de color similar a la del objeto de referencia. En el caso de análisis con textura mediante la transformada Wavelet se tiene una reducción en el número de falsos positivos, sin embargo los objetos que lucen similares en la imagen son detectados como el mismo objeto.

Gracias a los descriptores 3D, la aquí referida como 3D-Signature, los cuales toman en cuenta la geometría del objeto, se alcanza una mejor discriminación entre objetos con patrones de color/textura similares. En la figura 28 se muestra cómo se distingue un objeto prismático regular y uno cilíndrico. En esta prueba, se observa cómo opera el algoritmo con la presencia de múltiples objetos a distintas profundidades y con oclusiones parciales.

a) b)

c) d)

Figura 28. Prueba de detección de objetos: a) Imagen de referencia; Detección de objetos por: b) RFCH, c)RFCH+Wavelet, d) RFCH+Wavelet+3D-Signature

El algoritmo opera en una Laptop AMD 64bit 2GHz y 2GB de memoria RAM, con una secuencia de video a 5 cuadros por segundo para la búsqueda de un objeto particular de entre los observados, logrando ser detectado el objeto desde diferentes posiciones y profundidades en cada cuadro (rastreo), como se observa en la figura 29.

a) b)

c) d)

Figura 29. a) Imagen de referencia. b), c) y d) Objeto detectado en distintas profundidades y con diferentes niveles de oclusión.

En la figura se observan las tasas de detección y reconocimiento alcanzadas por el algoritmo para el caso de una secuencia de video de 100 cuadros. Se entrena un objeto colocándolo frente a la cámara y obteniendo su histograma de co-ocurrencia de campos receptivos, información de textura y de geometría. Para ello, el área promedio que ocupa la imagen de entrenamiento en relación al cuadro completo es de 4.53% con una desviación estándar de 1.77%.

En este caso, debido a las dimensiones de la imagen de entrenamiento y la dimensión aún menos que ocupa el objeto a buscar en el cuadro completo, SIFT no alcanzó niveles

de detección apropiados y se descartó su uso para comparar con el nivel de reconocimiento. En este trabajo, entonces, se consideran los niveles de reconocimiento en relación con los de detección (nivel de error). Esto es, de las veces que el algoritmo detectó un objeto, cuántas era el objeto correcto.

El arreglo experimental consta de 14 objetos, 11 de los cuales se entrenaron para su búsqueda y tres se dejaron como objetos de control. Para una profundidad fija, se puede observar en la figura 30 que mientras los niveles de detección cuadro por cuadro son bajos en algunos casos (tasa de detección), cuando el algoritmo detecta un objeto la probabilidad de que este sea el correcto es alta (tasa de reconocimiento). En promedio, se tiene una probabilidad de que se presente un falso positivo de 4.76%.

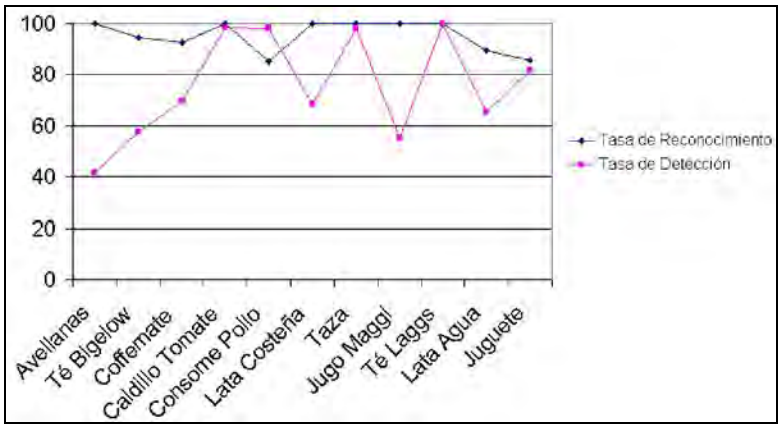


Figura 30. Tasas de reconocimiento y detección del algoritmo a una profundidad fija.

En la figura 31 se observa el rendimiento del algoritmo ante variaciones de profundidad para el caso de la caja de té (prisma rectangular horizontal), el caldillo de tomate (prisma rectangular vertical) y el bote de Coffemate (cilindro). Se observa que los niveles de detección y reconocimiento se mantienen regulares conforme la escala entre el objeto en la base de datos y en la escena cambia. Se logró detectar al objeto en escalas superiores al 50%, esto es, cuando el objeto ocupa el 2.25% del área total del cuadro de la imagen. Con SIFT no se logró reconocer al objeto con dimensiones tan pequeñas.

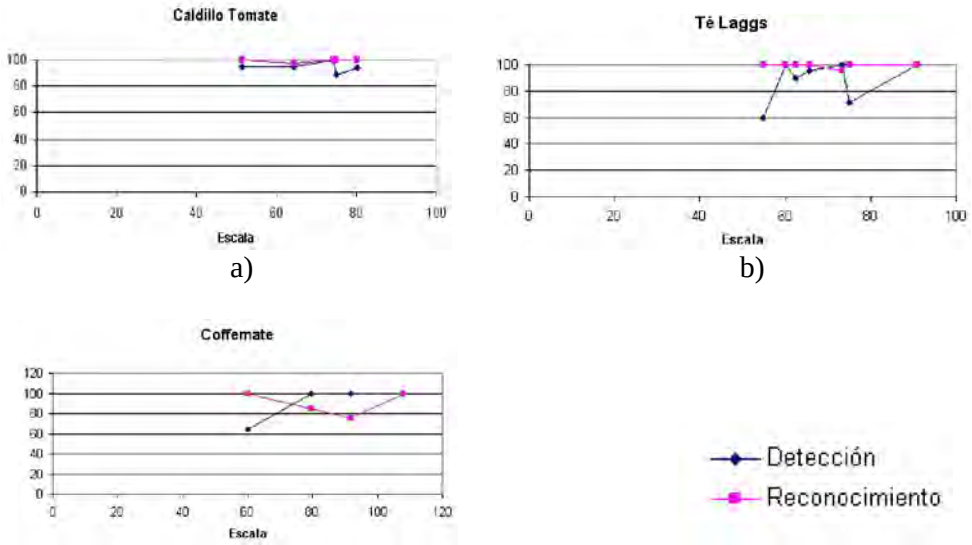


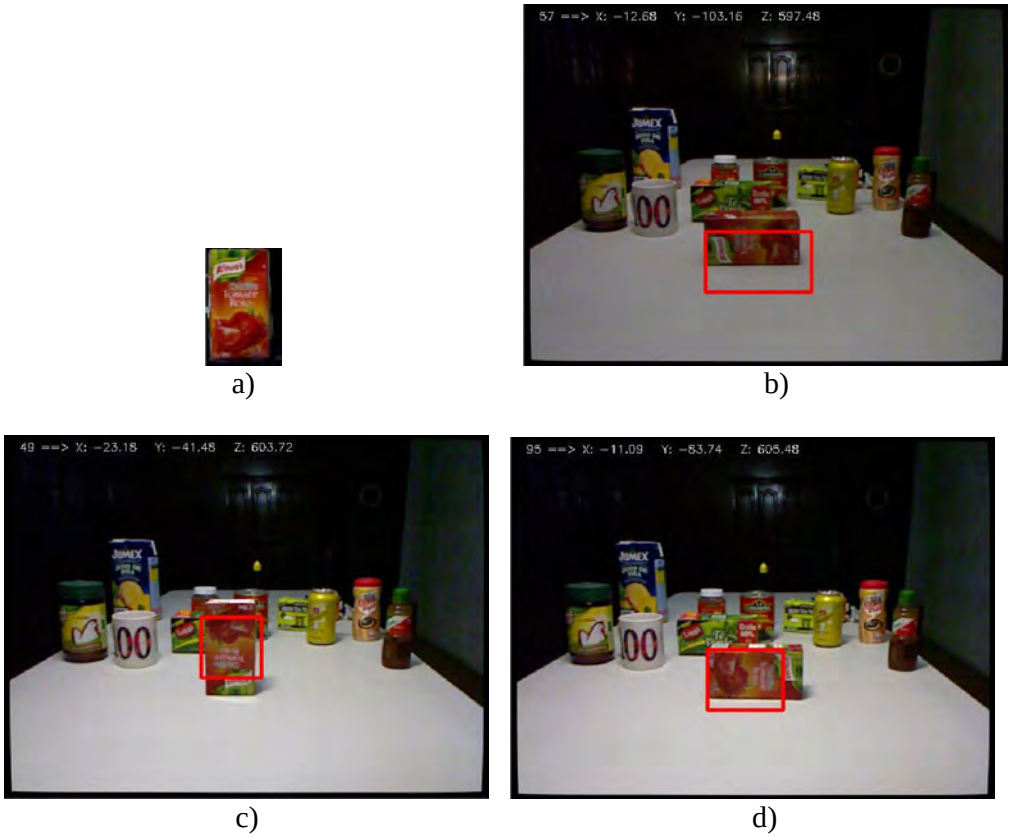
Figura 31. Tasas de detección y reconocimiento para el caso de a) la caja de caldillo de tomate, b) la caja de té Laggs y c) el bote de Coffemate.

Se puede observar en la figura 32 el desempeño del algoritmo a distintas profundidades, así como cierta robustez a los cambios en iluminación.

Figura 32. a) Objeto entrenado a una profundidad de 523 (mm); Objeto detectado a una profunidad de: b) 857 (mm), c) 869 (mm), d) 1347 (mm)}

Dado que esta es una aproximación experimental del descriptor 3D de una imagen, el desempeño obtenido varía de acuerdo al tamaño de la palabra para cada punto (asociado tanto al radio de la esfera centrada en el punto como al número de niveles en el histograma de distancias del descriptor), así como al tamaño del vocabulario (número de clusters elegido para describir al "mundo").

Para analizar el comportamiento del algoritmo ante rotaciones, se colocó el objeto frente a la cámara aproximadamente a la misma distancia en que se entrenó, y se rotó al mismo en ángulos de 0°, 90°, 180° y 270°, como se muestra en la figura 33.



a) b)

c) d)

Figura 33. Detección del objeto a) con rotaciones en b) 90°, c) 180° y d) 270°.

Para los mismos objetos que en el caso de variación de la profundidad (a saber, la caja de caldillo de tomate, la caja de té Laggs y el bote de Coffemate) se realizaron pruebas de detección. En la tabla 1 se muestran los porcentajes de detección en cada caso y para cada objeto, así como el promedio, el cual resulta mayor al 90% en casi todos los casos, por lo que se considera un detector confiable ante la presencia de variaciones de rotación entre el objeto entrenado y él mismo en la escena. Sin embargo se observa que

el menor nivel de detección ocurre para el caso en que el objeto está rotado 180° con respecto a la posición de entrenamiento, esto quizá a los cambio de iluminación cuando esta proviene desde una fuente superior.

Objeto\Rotación	0°	90°	180°	270°
Té Laggs	100	100	83	100
Caldillo de Tomate	100	79	64	96
Coffeemate	100	98	65	100
Promedio	100	92.33	70.67	98.67

Tabla 1. Porcentaje de detección a distintas variaciones de rotación del objeto en la escena.

En cuanto al cambio de perspectiva, los resultados no son contundentes, pues muestran cierta inconsistencia debido a que para objetos con estructura regular y colores homogéneos un cambio de perspectiva no lo afecta tanto como con objetos irregulares o con distribución de color irregular. En el caso normal se alcanzan niveles de detección aceptables con cambios de perspectiva de hasta 22.5°, como se ilustra en la figura 34, y en el caso óptimo, para objetos uniformes en color, textura y geometría, de hasta 45°, con tazas de reconocimiento superiores al 90%.

a) b) c)

Figura 34. Detección del objeto a) visto desde distintas perspectivas: b) a 0° y a c) 25.5° del original.

Finalmente, para una misma imagen, se realizaron pruebas de tiempo de ejecución al variar el radio y el tamaño del vocabulario, de los cuales se obtuvieron modelos de comportamiento lineales. Al variar el radio solamente, se obtuvo un proporcional con una pendiente m = 0.2522, esto es, cada variación de 3.96mm el tiempo de ejecución se duplica. Para el caso de la variación del tamaño del vocabulario, se obtuvo una pendiente m = 0.0449, siendo entonce que por cada incremento de 22.27 palabras se duplica el tiempo de ejecución.

Estos cambios en los tiempos de ejecución se deben a que mientras sea constante la imagen y, en consecuencia, su mapa de profundidad, al incrementar el radio de la esfera centrada en el punto de interés el número de puntos de los cuales hay que calcular la distancia al centro incrementa en forma cúbica (en relación al volumen de la esfera), al aumentas el número de palabras, este aumento es lineal, y el algoritmo de búsqueda de árboles KD puede lidiar de forma sencilla con variaciones pequeñas.

Capítulo V. Aplicación en robots móviles

Para la aplicación específica de manipulación de objetos con un brazo mecánico, se trabajó sobre un modelo por bloques, en el que varios subsistemas controlan la operación del robot. La interfaz ViRBot (J. Savage, *et al*., 1998, 2008) permite controlar sistemas reales y virtuales. Los robots virtuales pueden ejecutar las mismas órdenes, utilizando un lenguaje de programación del robot, que los robots reales, así como los comportamientos, las ecuaciones de movimiento y las lecturas de los sensores. La figura 35 ilustra las etapas requeridas para la operación de un robot móvil.

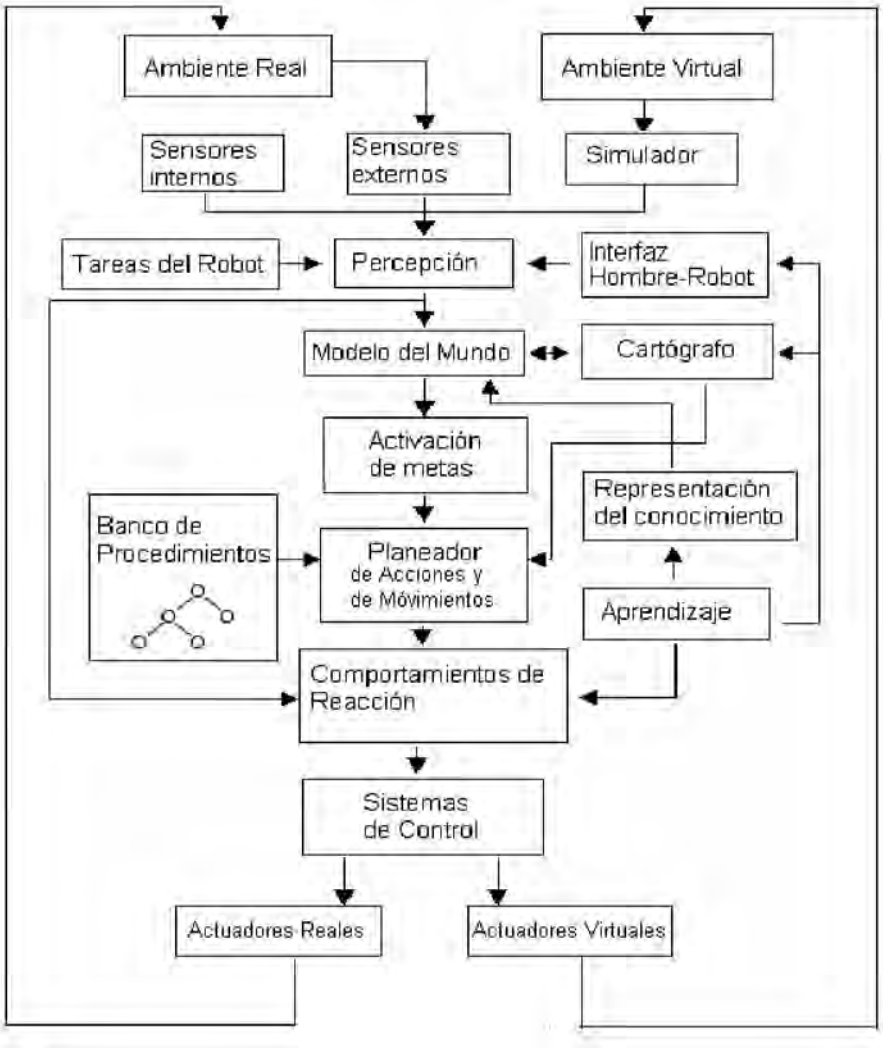


Figura 35. Sistema ViRBot

Una simplificación de la operación de este sistema es a través de una estructura modular con un sistema que distribuya el trabajo (*Blackboard*). Este tipo de estructura permite el enfoque por áreas en las distintas tareas que deben ser resueltas. En este caso, el sistema de visión resulta en un bloque cuya respuesta depende de los requisitos del *Blackboard*, como parte de una operación global compleja. La figura 36 muestra este sistema modular.

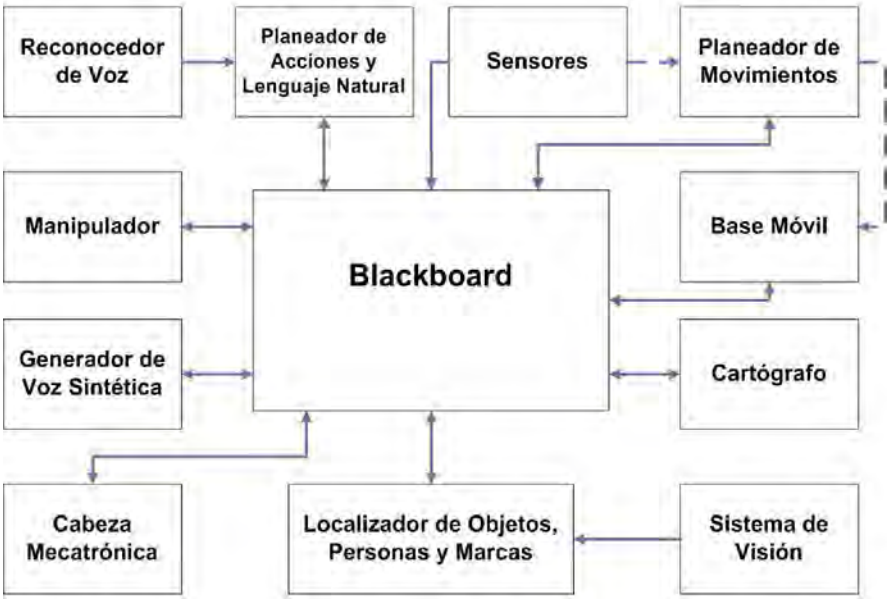


Figura 36. Sistema modular para la operación de un robot de servicio.

Se trabajó sobre el robot de servicio *Justina*, del laboratorio de Bio-Robótica, de la Facultad de Ingeniería – UNAM, figura 37. Entre otros sistemas, los que interesan para este trabajo es la cámara superior RGB-D (Microsoft Kinect™) y el brazo mecánico. Para probar la eficiencia del algoritmo aquí descrito, se consideró la tarea específica "Robot, limpia la mesa".

El escenario es el siguiente: se tiene una mesa con dos objetos, uno de ellos identificado como "basura". El robot es colocado en la proximidad de la mesa, la cual tiene la altura suficiente para poder manipular los objetos con el brazo mecánico. Al recibir la orden, el robot se debe acercar a la mesa, reconocer la presencia de objetos, reconocerlos y dar su posición 3D en relación con la posición de la cámara (conocida). Al saber la posición del objeto respecto de la cámara, se realiza una traslación/rotación de ejes para saber la posición de los objetos respecto al brazo y proceder a la manipulación de los objetos.

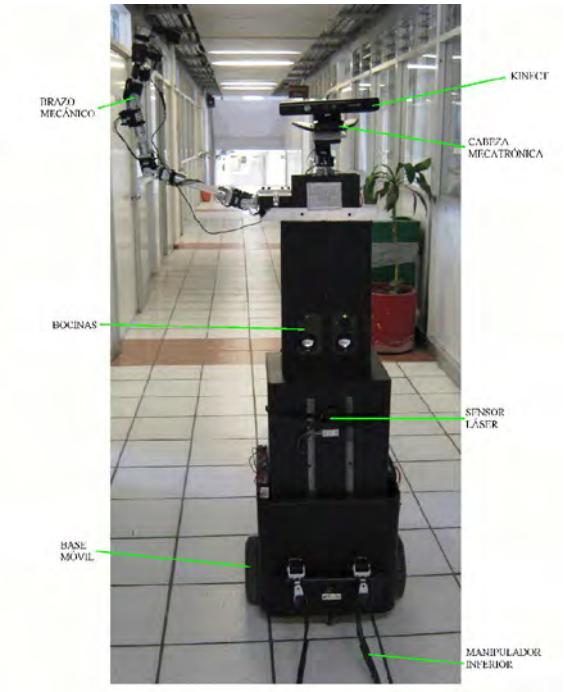
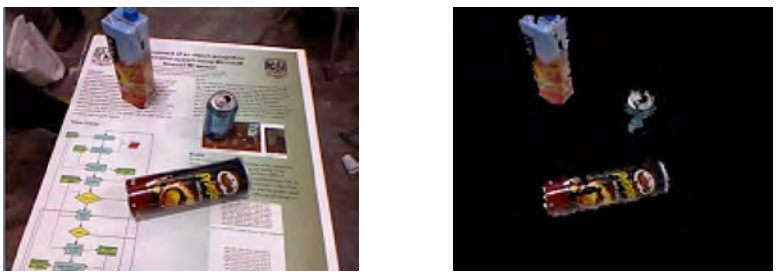


Figura 37. Robot “Justina”.

Por las condiciones de este problema, se tienen las siguientes etapas de operación:

1. Extracción del plano dominante.
2. Identificación de objetos (ubicación 3D).
3. Reconocimiento de objetos.

Para la extracción del plano, se considera que el robot se aproximó lo suficiente a la mesa y gira la cámara en un ángulo conocido, tal que la mesa con los objetos ocupa la mayor parte de la escena. Con estas premisas es posible extraer el plano dominante aplicando RANSAC, al considerarlo como aquel con la mayor cantidad de puntos en un nivel de profundidad dentro del rango de alcance del brazo mecánico, como se ilustra en la figura 38.



a) b)

Figura 38. a) Mesa con objetos. b) extracción del plano.

Se puede observar que los objetos resultantes ocupan un área pequeña de la imagen y en muchas ocasiones en ángulos en picada, por lo que resulta casi imposible identificarlos mediante algoritmos como SIFT. RFCH resultó eficiente en estos casos, pues con una base de datos que considere estas tomas, se puede identificar al objeto con un nivel de confianza razonable.

Aún así, cuando el proceso falla con RFCH, al saber la posición 3D de un objeto desconocido, el manipulador consigue acercar el objeto a la cámara y, al ocupar ahora un área mayor en la imagen, se emplea SIFT para identificarlo. Este no es deseable por el tiempo de cómputo que cuesta procesar esta operación; sin embargo, considerando los niveles bajo de detección para unos tipos de objetos, se hace necesario tener un plan de respaldo en operaciones en ambientes reales.

Conclusiones

En la primer parte de este trabajo se definieron las tareas de detección y reconocimiento, así como algunas de las técnicas más sobresalientes para realizarlas. Se presentó la transformación de características invariantes a escala (SIFT) y el histograma de co-ocurrencias de campos receptivos (RFCH).

Se desarrolló, entonces, una mejora a la técnica de detección de objetos RFHC, al incluir la información de texturas mediante descriptores estadísticos de la transformada Wavelet de tipo Haar (HWT), observándose una reducción en el número de falsos positivos. Sin embargo, en objetos con patrones de color/textura similares, el algoritmo sigue presentándolos.

Aprovechando la tecnología de cámaras RGB-D, se empleó el Microsoft Kinect™ y se desarrolló un descriptor que toma en cuenta la geometría 3D del objeto (3D-Signature), generando una esfera centrada en un punto de interés y obteniendo el histograma de distancias de cada punto en la esfera al del centro (palabra). Un objeto está descrito por una serie de palabras (frase) contenidas en un vocabularios del mundo. Con esto se logra discriminar falsos positivos, alcanzando niveles de reconocimiento que algoritmos como SIFT no alcanzan con variaciones drásticas de escala.

A su vez, se desarrolló una caracterización distancia-escala que nos permite manipular la dimensión de la imagen en pixeles en correspondencia a la profundidad en que se encuentran los mismos, muy útil en la tarea de localizar un objeto entrenado a una profundidad específica dentro de un escenario complejo con un rango amplio de profundidad.

Para probar el algoritmo, se enternan objetos que ocupan el 5 por ciento de la imagen completa, y con la combinación de RFCH + HWT + 3D-Signature se logró detectarlos con una escala de hasta el 50% con una probabilidad de falso positivos menor al 5%. Este proceso mostró ser también robusto a la s rotaciones.

Estas técnicas se probaron exitosamente en escenarios con condiciones de luz variables, y con niveles de oclusión parciales, obteniéndose niveles de reconocimiento confiables.

Como aplicación para robots móviles, este proceso se probó en el escenario planteado en la competencia de robótica RoboCup 2011, en Estambul, Turquía, en la categoría *@home* con el equipo PUMAS, obteniendo resultados satisfactorios.

Trabajo a futuro

Este trabajo presentó una aproximación experimental de un descriptor de la geometría de un objeto. Se pretende desarrollar un modelo matemático de los parámetros que definen la técnica 3D-Signature correlacionando las variables involucradas en el proceso (tamaño de palabra, frase, vocabulario).

La extracción de características a través de la información 3D de un objeto está teniendo un auge gracias al bajo costo de las cámaras RGB-D en dispositivos como el Microsoft Kinect™, por lo que es un área que se puede explotar aún más.

En este particular, se pretende generar un extractor de planos más eficiente que el desarrollado en este trabajo, considerando el análisis de normales sobre cada punto.

Por otro lado, se pretende general un modelo manipulación para cada objeto dependiendo de su geometría, para que el acercamiento y ángulo de agarre del brazo mecánico sea el adecuado dependiendo de la posición del objeto.

BIBLIOGRAFÍA

A. Antoniou, 2007. “On the roots of digital signal processing—part I,” IEEE Circuits and Systems Magazine, vol. 7, no. 1, pp. 8–18.

A. Antoniou, 2007. “On the roots of digital signal processing—part II,” IEEE Circuits and Systems Magazine, vol. 7, no. 4, pp. 8–18.

J. S. Beis and D. G. Lowe, 1997. “Shape Indexing Using Approximate Nearest-Neighbour Search in High-Dimensional Spaces”.

L. Benedetti, M. Corsini, P. Cignoni, M. Callieri, R. Scopigno, 2010. “Color to gray conversions in the context of stereo matching algorithms”. Machine Vision and Applications, pp. 1-22. Springer-Verlag.

J. L. Bentley, 1975. “Multidimensional binary search trees used for associative searching Commun”. ACM, ACM, 18, 509-517.

J. Bruce, T. Balch, y M. Veloso, 2000. “Fast and inexpensive color image segmentation for interactiva robot”. Proceedings of the 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems, October, pp. 2061-2066.

P. Chang, J. Krumm, 1999. “Object recognition with color cooccurrence histograms”. Proceedings of Conference IEEE Computer Vision and Pattern Recognition vol. 2, pp. 504.

F. Cheikh, A. Quddus y M. Gabbouj, 2000. “Multi-level shape recognition based on wavelet-transform modulus maxima”. Proc. of Image Analysis and Interpretation, pp. 8-12, April 2-4.

Y. Chen, K. Chao y S. Moon, 2002. “Machine vision technology for agricultural applications”. Computers and Electronics in Agriculture 36: 173-191.

T. Chen, Y. Chen, and S. Chien, 2009. “Fast Image Segmentation and Texture Feature Extraction for Image Retrieval,” in Proceedings of IEEE International Conference on Computer Vision Workshops (ICCV2009 Workshops), Kyoto, Japan, pp. 854–861.

T. Collett, M. Collett y R. Wehner, 2001. “The guidance of desert ants by extended landmarks”. The Journal of Experimental Biology, 204, 1635–1639.

L. Contreras, 2007. “Automatización de un invernadero utilizando componentes electrónicos comerciales (COTS)”. Tesis licenciatura, Facultad de Ingeniería, UNAM. México.

D. Donoho y X. Huo, 2002. “Beamlets and multiscale image analysis. In Multiscale and Multiresolution Methods”. Springer Lecture Notes in Computational Science and Engineering, 20: 149-196.

S. Ekvall, P. Jensfelt y D. Kragic, 2006. “Integrating Active Mobile Robot Object Recognition and SLAM in Natural Environments” IEEE/RSJ International Conference on Intelligent Robots and Systems. IROS'06.

S. Ekvall, D. Kragic, 2005. “Receptive Field Cooccurrence Histograms for Object Detection”. IEEE/RSJ International Conference on Intelligent Robots and Systems. IROS'05.

G. Fink, 2003. “Markov models for pattern recognition. From theory to applications”. Springer-Verlag Berlin Heidelberg.

T. Fukushi, 2001. “Homing in wood ants, Formica Japonica: Use of the skyline panorama”. The Journal of Experimental Biology, 204, 2063–2072.

P. Graham, K. Fauria y T. Collett, 2003. “The influence of beacon-aiming on the routes of wood ants”. The Journal of Experimental Biology, 206, 535-541.

B. Haar, 2003. “Front-End Vision and Multi-Scale Image Analysis”. Springer-Verlag Berlin Heidelberg.

P. Henry, M. Krainin, E. Herbst, X. Ren and D. Fox, 2010. “RGB-D Mapping: Using Depth Cameras for Dense 3D Modeling of Indoor Environments”. 12th International Symposium on Experimental Robotics (ISER), New Delhi, India.

K. Kanda, Y. Miyamoto, A. Kondo y M. Oshio, 2005. “Monitoring of earthquake induced motions and damage with optical motion tracking”. Smart Mater. Struct. 14 S32.

Z. Khan, T. Balch y F. Dellaert, 2005. “MCMC-Based Particle Filtering for Tracking a Variable Number of Interacting Targets”. IEEE Transactions on Pattern Analysis and Machine Intelligence, 27 (11), 1805-1918.

D. Lowe, 2004. “Distinctive Image Features from Scale-Invariant Keypoints”. Int. J. Comput. Vision, Kluwer Academic Publishers, 60, 91-110

M. Masood, 2005. “Multiscale analysis techniques in patter recognition systems”. King Fahd University of Petroleum and Minerals. Dhahran, Saudi Arabia.

V. Mezaris, I. Kompatsiaris and M. Strintzis, 2004. "Still Image Segmentation Tools for Object-based Multimedia Applications", International Journal of Pattern Recognition and Artificial Intelligence, vol. 18, no. 4, pp. 701-725.

A. W. Moore, 2001. “K-means and Hierarchical Clustering”. School of Computer Science, Carnegie Mellon University.

P. Müllier y D. Robert, 2001. “A Shot in the Dark - The Silent Quest of a Free-Flying Phonotactic Fly”. The Journal of Experimental Biology, 204, 1039–1052.

D. Nicholson, S. Judd, B. Cartwright y T. Collett, 1999. “Learning walks and landmark guidance in wood ants (formica rufa)”. The Journal of Experimental Biology, 202, 1831–1838.

A. Pacheco-Ortega, 2011. “Adecuación de técnicas de descripción visual utilizando información 3D y su aplicación en robótica”. Tesis maestría, IIMAS-UNAM, México.

F. Peña, P. Lourenço y J. Lemos, 2006. “Modeling the Dynamic Behavior of Masonry Walls as Rigid Blocks”. In: Mota Soares et al. (eds), III European Conference on computational Mechanics Solids, Structures and Coupled Problems in Engineering. Lisbon, Portugal.

Pumas-México Team, 2008. “ViRbot: A System for the Operation of Mobile Robots”. Team Description Paper for Robocup 2008, China.

B. Ronacher, K. Gallizzi, S. Wohlgemuth y R. Wehner, 2000. “Lateral optic flow does not influence distance estimation in the desert ant cataglyphis fortis”. The Journal of Experimental Biology, 203, 1113–1121.

J. Savage, M. Billinghurst, A. Holden, 1998. “The Virbot: A virtual reality mobile robot driven with multimodal commands”. Expert Systems with Applications 15, pp 413-419.

J. Sachs, 1999. “Using Curves and Histograms” Digital Light & Color.

Y. Sheng, 1996. “The Transforms and Applications Handbook”. CRC Press.

A. R. Smith, 1978. “Color gamut transform pairs,” Computer Graphics 12, pp 12–19.

S. Smith, 1999. “The Scientist and Engineer’s Guide to Digital Signal Processing”. Ed. California Technical Publishing, San Diego, California, USA. 2 ed.

M. J. Swain y D. H. Ballard, 1991. “Color Indexing,” International Journal of Computer Vision, vol. 7, pp. 11-32.

R. Willett y R. D. Nowak, 2001. "Platelets: A Multiscale Approach for Recovering Edges and Surfaces in Photon-Limited Medical Imaging," IEEE Transactions on Medical Imaging.

PrimeSense. http://www.primesense.com/

Apéndices

A.

Biblioteca con funciones de corrección de distorsión para cámaras RGB-D, basada en OpenCV y OpenNI.

Undistorter.h

```
#ifndef _UNDISTORTER_H_
#define _UNDISTORTER_H_

#include "cv.h"
#include "cxcore.h"

typedef struct  {
		double cx;
		double cy;
		double fx;
		double fy;
} Intrinsics;

class Undistorter{
private:
		IplImage* r ;
		IplImage* g ;
		IplImage* b ;
		IplImage* ru ;
		IplImage* gu ;
		IplImage* bu ;

		int width;
		int height;

public:
		CvMat* intrinsic;
		CvMat* distortion;
		CvMat* mapX;
		CvMat* mapY;
		Intrinsics coefIntr;

		Undistorter( int w, int h, CvMat* &intr, CvMat* &dist  ){

			this->width = w;
			this->height = h;

			this->intrinsic = intr ;
			this->distortion = dist ;
			coefIntr.fx  = cvmGet(intrinsic, 0, 0);
			coefIntr.fy  = cvmGet(intrinsic, 1, 1);
			coefIntr.cx  = cvmGet(intrinsic, 0, 2);
			coefIntr.cy  = cvmGet(intrinsic, 1, 2);

			this->r = cvCreateImage(cvSize(this->width, this->height), IPL_DEPTH_8U, 1);
			this->g = cvCreateImage(cvSize(this->width, this->height), IPL_DEPTH_8U, 1);
			this->b = cvCreateImage(cvSize(this->width, this->height), IPL_DEPTH_8U, 1);

			this->ru = cvCreateImage(cvSize(this->width, this->height), IPL_DEPTH_8U, 1);
			this->gu = cvCreateImage(cvSize(this->width, this->height), IPL_DEPTH_8U, 1);
			this->bu = cvCreateImage(cvSize(this->width, this->height), IPL_DEPTH_8U, 1);


			mapX = cvCreateMat(this->height, this->width, CV_32FC1);
			mapY = cvCreateMat(this->height, this->width, CV_32FC1);

			cvInitUndistortMap(intrinsic, distortion, mapX, mapY);
		}

		void undistor_rgb_image(IplImage* img, IplImage* out) {
```

```
                    cvSplit(img, r, g, b, NULL);
                    cvRemap( r, ru, mapX, mapY, CV_INTER_LINEAR + CV_WARP_FILL_OUTLIERS, cvScalarAll(0) );
                    cvRemap( g, gu, mapX, mapY, CV_INTER_LINEAR + CV_WARP_FILL_OUTLIERS,
cvScalarAll(0) );
                    cvRemap( b, bu, mapX, mapY, CV_INTER_LINEAR + CV_WARP_FILL_OUTLIERS,
cvScalarAll(0) );
                    cvMerge(ru, gu, bu, NULL, out);
            }
};
#endif
```

B.

Biblioteca para extracción y comparación de características para Campor Receptivos, basadas en OpenCV.

deTenshi.h

```
#include "cv.h"
#include "highgui.h"
#include <cmath>
using namespace std;

//Calculate the HWT from grayscale SRC and write the coeffcients
//into DST at the given level
//Level 0: Full SRC to 4/4 DST; Level 1: 1/4 SRC to 4/16 DST; etc.
void HWT (IplImage* src, IplImage* dst, int Level)
{
   Level = (int)(pow(2, double(Level)));

   int H    = (src->height)/Level;
   int W    = (src->width)/Level;
   int step = src->widthStep;

   int a, b, c, d, i, j;
   int deltaX = (int)(W/2);
   int deltaY = (int)(H/2);

            uchar *dSRC  = (uchar *)src->imageData;
            uchar *dDST = (uchar *)dst->imageData;

            for(i=0; i<(H-1); i++)
                        for( j=0; j<(W-1); j++)
                        {
                                    a = dSRC[i*step+j];
                                    b = dSRC[i*step+(j+1)];
                                    c = dSRC[(i+1)*step+j];
                                    d = dSRC[(i+1)*step+(j+1)];

                                    dDST[(i/2)*step+(j/2)] = (a + b + c + d)/4;
                                    dDST[(i/2)*step+((j/2)+deltaX)] = (b + d - a - c)/8+128;
                                    dDST[((i/2)+deltaY)*step+(j/2)] = (c + d - a - b)/8+128;
                                    dDST[((i/2)+deltaY)*step+((j/2)+deltaX)] = (b + c - a - d)/8+128;
                        }
}

//Calculate the mean and std. dev. vector VAL[3*Level][2]
//from HWT detail coefficients from SRC (HWT image)
void stadistics (IplImage* src, CvMat *val, int Level)
{
   int ii, jj, kk, pp, H, W;
   CvScalar Mu;
   CvScalar Sigma;

   pp = 0;
   for (kk = 0; kk < Level; kk++)
   {
      H = (int)((src->height)/(2*(int)pow(2, double(kk))));
      W = (int)((src->width)/(2*(int)pow(2, double(kk))));
      for (ii = 0; ii <= 1; ii++)
         for (jj = 0; jj <= 1; jj++)
         {
            if (ii == 0)
              jj++;

            cvSetImageROI(src, cvRect(ii*W, jj*H, W, H));
            cvAvgSdv(src, &Mu, &Sigma, 0);
            val->data.fl[pp * 2 + 0] = Mu.val[0];
            val->data.fl[pp * 2 + 1] = Sigma.val[0];

            cvResetImageROI(src);
            pp++;
```

```
        }
    }
}

//Calculate the Euclidean distance between vectors VAL1[Size][2]
//and VAL2[Size][2]
float distance (CvMat* val1, CvMat* val2, int Size)
{
    int ii, jj, idx;
    float dst, Dist = 0;

    for (ii = 0; ii < Size; ii++)
    {
        dst = 0;
        for (jj = 0; jj <= 1; jj++)
                                    dst = dst + pow(val1->data.fl[ii*2 + jj] - val2->data.fl[ii*2 + jj], 2);

        Dist = Dist + pow(double(dst), 0.5);
    }
    Dist = Dist / Size;
    return Dist;
}


//Calculate Receptive Field characteristics vector PTS[SRCsize][Psize]
//from SRC image
void RField(IplImage* src, CvMat* pts, int Psize)
{
            IplImage *imgGray, *imgNOR, *imgGrad, *imgLap;
            IplImage *imgTmp1,*imgTmp2,*imgTmp3;

    imgGray = cvCreateImage(cvGetSize(src),IPL_DEPTH_8U,1);
    cvCvtColor( src, imgGray, CV_RGB2GRAY );

    imgNOR = cvCreateImage(cvGetSize(src),IPL_DEPTH_8U,3);
    imgGrad = cvCreateImage(cvGetSize(src),IPL_DEPTH_16S,1);
    imgLap  = cvCreateImage(cvGetSize(src),IPL_DEPTH_8U,1);

    imgTmp1 = cvCreateImage(cvGetSize(src),IPL_DEPTH_8U,1);
    imgTmp2 = cvCreateImage(cvGetSize(src),IPL_DEPTH_16S,1);
    imgTmp3 = cvCreateImage(cvGetSize(src),IPL_DEPTH_16S,1);

    // GET HSV IMAGE
    cvCvtColor( src, imgNOR, CV_RGB2HSV );

    // GET IMAGE GRADIENT
    int sigma = 2;
    int ksize = (sigma*5)|1;

    cvSmooth( imgGray, imgTmp1, CV_GAUSSIAN, ksize, ksize, sigma, sigma );

    cvSobel( imgTmp1, imgTmp2, 1, 0, 3);
    cvSobel( imgTmp1, imgTmp3, 0, 1, 3);

    cvAdd(imgTmp2, imgTmp3, imgGrad, NULL);
    cvConvertScale (imgGrad, imgGrad, 255.0, 0);
    // ***************************

    // GET IMAGE LAPLACIAN
    cvLaplace( imgGray, imgTmp2, 1 );
    cvConvertScaleAbs( imgTmp2, imgLap, (sigma+1)*0.25, 0 );
    // ***************************

    // GET RF VECTOR
    int i, j, k = 0;
    int step = imgNOR->widthStep;
            for ( i = 0; i < src->height; i++)
        for ( j = 0; j < src->width; j++)
            {
                        pts->data.fl[k * Psize + 0] = imgNOR->imageData[i * imgNOR->widthStep + 3*j + 0];
                        pts->data.fl[k * Psize + 1] = imgNOR->imageData[i * imgNOR->widthStep + 3*j + 1];
                        pts->data.fl[k * Psize + 2] = imgNOR->imageData[i * imgNOR->widthStep + 3*j + 2];
                        pts->data.fl[k * Psize + 3] = imgGrad->imageData[i * imgGrad->widthStep + j];
                        pts->data.fl[k * Psize + 4] = imgLap->imageData[i * imgLap->widthStep + j];
            k++;
```

```
            }

    cvReleaseImage(&imgGray);
    cvReleaseImage(&imgNOR);
    cvReleaseImage(&imgGrad);
    cvReleaseImage(&imgLap);
    cvReleaseImage(&imgTmp1);
    cvReleaseImage(&imgTmp2);
    cvReleaseImage(&imgTmp3);
}

//Calculate the Coocurrence Histogram CHIST from clusterized image CLUST
void RFCH (CvMat* clust, CvMat* chist, int Width, int Height, int MAX_C, int Train)
{
    int a, b, c, i, j, k;
    int nd = 12;

    double maxVal;

    CvMat *DD = cvCreateMat (nd, 2, CV_32SC1);

    for (i = 1; i <= nd; i++)
    {
        DD->data.i[i * 2 + 0] = (i-1);
        DD->data.i[i * 2 + 1] = i;
    }

    c = 0;

    cvZero (chist);

    for (k = 0; k < nd; k++)
        for (i = 0; i < Height; i++)
            for (j = 0; j < Width; j++)
            {
                if ((i+DD->data.i[k*2 + 1]) < Height)
                {
                    //RIGHT SIDE
                    if (((j+DD->data.i[k*2 + 0]) <= Width)&&((j+DD->data.i[k*2 + 0]) >= 0))
                    {
                        a = clust->data.i[i*Width+j];
                        b = clust->data.i[(i+DD->data.i[k*2 + 1])*Width+(j+DD->data.i[k*2 + 0])];

                        if ((a>=0)&&(b>=0)&&(a < MAX_C)&&(b < MAX_C))
                        {
                            if (a <= b)
                                chist->data.fl[a*MAX_C+3*b+2]=chist->data.fl[a*MAX_C+3*b+2] + 1;
                            else
                                chist->data.fl[b*MAX_C+3*a+2]=chist->data.fl[b*MAX_C+3*a+2] + 1;
                        }
                    }

                    //LEFT SIDE
                    if (((j-DD->data.i[k*2 + 0]) >= 0)&&((j-DD->data.i[k*2 + 0]) <= Width))
                    {
                        a = clust->data.i[i*Width+j];
                        b = clust->data.i[(i+DD->data.i[k*2 + 1])*Width+(j-DD->data.i[k*2 + 0])];

                        if ((a>=0)&&(b>=0)&&(a < MAX_C)&&(b < MAX_C))
                        {
                            if (a <= b)
                                chist->data.fl[a*MAX_C+3*b+2]=chist->data.fl[a*MAX_C+3*b+2] + 1;
                            else
                                chist->data.fl[b*MAX_C+3*a+2]=chist->data.fl[b*MAX_C+3*a+2] + 1;
                        }
                    }
                }
                c++;
            }

                if (Train == 1)
                {
                        cvMinMaxLoc(chist, 0, &maxVal, 0, 0, 0);
                        for (i = 0; i < MAX_C; i++)
                                for (j = 0; j < MAX_C; j++)
```

```
                                        {
                                                if (chist->data.fl[i*MAX_C +3*j+2] == maxVal)
                                                  chist->data.fl[i*MAX_C +3*j+2] = 0;
                                                else
                                                  chist->data.fl[i*MAX_C +3*j+2] = 100*chist->data.fl[i*MAX_C +3*j+2]/(c-
maxVal);
                                        }
                }
                else
                {
                        for (i = 0; i < MAX_C; i++)
                                for (j = 0; j < MAX_C; j++)
                                        chist->data.fl[i*MAX_C +3*j+2] = 100*chist->data.fl[i*MAX_C +3*j+2]/c;
                }
    cvReleaseMat(&DD);
}

//Calculate the Histogram Intersection
float INTERCH (CvMat* ModCH, CvMat* ImgCH, int MAX_C)
{
    int i, j;
    float out = 0;

    for (i = 0; i < MAX_C; i++)
      for (j = 0; j < MAX_C; j++)
        {
          if (ModCH->data.fl[i*MAX_C +3*j+2] <= ImgCH->data.fl[i*MAX_C +3*j+2])
            out = out + ModCH->data.fl[i*MAX_C +3*j+2];
          else
            out = out + ImgCH->data.fl[i*MAX_C +3*j+2];
        }
   return out;
}

//Calculate the mean distance DIST from Receptive Field image PTS
//to each point corresponding cluster in CENT given by CLUST
void distPoints2Centers (CvMat* pts, CvMat* clust, CvMat* cent, CvMat* dist, int Size, int MAX_C, int Psize)
{
  int i, j, idx;
  float dst;
  cvZero (dist);

  for (i = 0; i < Size; i++)
  {
      idx = clust->data.i[i];

      dst = 0;
      for (j = 0; j < Psize; j++)
          dst = dst + pow(pts->data.fl[i*Psize + j] - cent->data.fl[idx*Psize + j],2);
      dist->data.fl[idx*2 + 0] = dist->data.fl[idx*2 + 0] + pow(double(dst), 0.5);
      dist->data.fl[idx*2 + 1] = dist->data.fl[idx*2 + 1] + 1;
  }

  for (i = 0; i < MAX_C; i++)
     dist->data.fl[i*2 + 0] = dist->data.fl[i*2 + 0] / dist->data.fl[i*2 + 1];
}

//Find the nearest cluster in FT to each point in PTS by KDTree algorithm
//and generate the clusterized image CLUST
//If this distance is greater than the average distance in DIST
//the pixel in SRC turns into 0, and the cluster in CLUST into -1
void newQuantize (IplImage* src, CvMat* pts, CvFeatureTree *ft, CvMat* dist, CvMat* clust, int Size)
{
  int i, j, x, y;
  CvScalar s = CV_RGB(0,0,0);

  CvMat *distances = cvCreateMat(Size,1,CV_64FC1);

  cvFindFeatures(ft, pts, clust, distances, 1, 250);

  i = 0;
  for (y = 0; y < src->height; y++)
     for (x = 0; x < src->width; x++)
  {
     j = clust->data.i[i];
```

```
      if ((float)(cvGetReal2D(distances, i, 0)) > 1.5*dist->data.fl[j*2 + 0])
      {
        cvSet2D(src,y,x,s);
        clust->data.i[i] = -1;
      }
      i++;
    }

    cvReleaseMat(&distances);
}

void train_object (IplImage* ImgSrc, CvMat* &CHists, CvMat* &hwt_vals, CvMat* &distances, CvMat* &centers,
CvFeatureTree* &features, int Level, int VSize, int MAX_C, float scale)
{
//VARIABLES
    //IMAGES
    IplImage *ImgTmp = 0, *ImgHWT = 0, *ImgScaled = 0;

    //TRAINING
    CvMat *clusters = 0, *points = 0;

    //GENERAL USE
    int ii, Size, Train = 1;

    ImgScaled = cvCreateImage(cvSize((int)(scale*ImgSrc->width), (int)(scale*ImgSrc->height)), ImgSrc->depth, 3);

    cvResize(ImgSrc, ImgScaled, 0);

    Size = ImgScaled->width * ImgScaled->height;

    if (hwt_vals != 0)
      cvReleaseMat (&hwt_vals);

    if (distances != 0)
      cvReleaseMat (&distances);

    if (CHists != 0)
      cvReleaseMat (&CHists);

    if (centers != 0)
      cvReleaseMat (&centers);

    //Image Descriptors
    hwt_vals = cvCreateMat (3*Level, 2, CV_32FC1);
            distances = cvCreateMat (MAX_C, 2, CV_32FC1);
    CHists = cvCreateMat (MAX_C*MAX_C, 3, CV_32FC1);
            centers = cvCreateMat (MAX_C, VSize, CV_32FC1);

    //Local Variables
            clusters = cvCreateMat (Size, 1, CV_32SC1);
            points = cvCreateMat (Size, VSize, CV_32FC1);

    //GET WAVELET COEFFICIENTS
    ImgTmp = cvCreateImage(cvGetSize(ImgScaled),IPL_DEPTH_8U,1);
    ImgHWT = cvCreateImage(cvGetSize(ImgScaled),IPL_DEPTH_8U,1);

    cvCvtColor( ImgScaled, ImgTmp, CV_RGB2GRAY );

    for (ii = 0; ii < Level; ii++)
    {
      HWT (ImgTmp, ImgHWT, ii);
      cvReleaseImage(&ImgTmp);
      ImgTmp = cvCloneImage(ImgHWT);
    }

    stadistics (ImgHWT, hwt_vals, Level);

    //GET RECEPTIVE FIELDS VALUES
    RField (ImgScaled, points, VSize);

            cvKMeans2 (points, MAX_C, clusters,
                    cvTermCriteria (CV_TERMCRIT_EPS + CV_TERMCRIT_ITER, 10, 1), 1, 0, 0, centers, 0);

    if (features != 0)
      cvReleaseFeatureTree (features);
```

```
    features = cvCreateKDTree(centers);

    distPoints2Centers (points, clusters, centers, distances, Size, MAX_C, VSize);

    //GET COOCURRENCE HIST.
    RFCH (clusters, CHists, ImgScaled->width, ImgScaled->height, MAX_C, Train);

    cvReleaseMat (&clusters);
    cvReleaseMat (&points);

    cvReleaseImage(&ImgHWT);
    cvReleaseImage(&ImgTmp);
    cvReleaseImage(&ImgScaled);
//*******************************
//*******************************
}

int find_object (IplImage* ImgSrc, IplImage* ImgZone, CvMat* CHists, CvMat* hwt_vals, CvMat* distances, CvMat*
centers, CvFeatureTree* features, int Level,  int vsize, int MAX_C, int sizew, int sizeh, int delta)
{
    //IMAGES
    IplImage *imgOut, *imgGray, *imgSRCGray, *imgTmp, *imgHWT;

    //TESTING
    CvMat *WINcluster, *cluster, *point, *center, *hwt_val, *CHist;
    float dd;

    //GENERAL USE
    int ii, jj, kk, xx, yy, val, Size, Obj = 0, Train = 0;

    //WINDOWING
    int next, temp;
    int wmax, hmax;

    int deltax, deltay;

    imgOut = cvCreateImage(cvGetSize(ImgSrc),IPL_DEPTH_8U,3);
    imgGray = cvCreateImage(cvGetSize(ImgSrc),IPL_DEPTH_8U,1);
    imgSRCGray = cvCreateImage(cvGetSize(ImgSrc),IPL_DEPTH_8U,1);

    Size = ImgSrc->width * ImgSrc->height;

    CHist = cvCreateMat (MAX_C*MAX_C, 3, CV_32FC1);
    hwt_val = cvCreateMat (3*Level, 2, CV_32FC1);

    center = cvCreateMat (MAX_C, vsize, CV_32FC1);
    cluster = cvCreateMat (Size, 1, CV_32SC1);
    point = cvCreateMat (Size, vsize, CV_32FC1);

    //GET IMAGE RECEPTIVE FIELDS VALUES
    RField (ImgSrc, point, vsize);

    cvCopy( ImgSrc, imgOut, 0);

    //IMAGE QUANTIZATION
    newQuantize (imgOut, point, features, distances, cluster, Size);

    cvCvtColor( imgOut, imgGray, CV_RGB2GRAY );
    cvCvtColor( ImgSrc, imgSRCGray, CV_RGB2GRAY );

    xx = 0; yy = 0;
    next = 1;

        if (sizew > sizeh) sizew = sizeh;
        else sizeh = sizew;

        if (sizew > 30)
        {
                sizew = 30;
                sizeh = 30;
        }
        else
        {
                delta = sizew / 2;
```

```
            }

            wmax = sizew;
    hmax = sizew;

    val = 0;
            cvZero (ImgZone);
    do
    {
                        cvSetImageROI(imgGray, cvRect(xx*delta, yy*delta, wmax, hmax));
                        ii = cvCountNonZero(imgGray);
                        cvResetImageROI(imgGray);

                        if (ii > 0.5 * wmax * hmax)
                        {
                                WINcluster = cvCreateMat (wmax*hmax, 1, CV_32SC1);
                                for (ii = 0; ii < hmax; ii++)
                                        for (jj = 0; jj < wmax; jj++)
                                        {
                                                WINcluster->data.fl[ii*wmax + jj] = cluster-
>data.fl[(yy*delta+ii)*(ImgSrc->width) + (xx*delta + jj)];
                                        }

                                cvZero(CHist);
                                RFCH (WINcluster, CHist, wmax, hmax, MAX_C, Train);

                                //COOCURRENCE HIST INTERSECTION
                                kk = (int)(2.55*INTERCH (CHists, CHist, MAX_C));

                                //GET TEXTURE SIGNATURE
                        imgTmp = cvCreateImage(cvSize(wmax,hmax),IPL_DEPTH_8U,1);
                                imgHWT = cvCreateImage(cvSize(wmax,hmax),IPL_DEPTH_8U,1);

                        cvSetImageROI(imgSRCGray, cvRect(xx*delta, yy*delta, wmax, hmax));
                        cvCopy(imgSRCGray, imgTmp, 0);
                        cvResetImageROI(imgSRCGray);

                                for (ii = 0; ii < Level; ii++)
                                {
                                        HWT (imgTmp, imgHWT, ii);
                                        cvReleaseImage(&imgTmp);
                                        imgTmp = cvCloneImage(imgHWT);
                                }

                                cvZero (hwt_val);
                                stadistics (imgHWT, hwt_val, Level);

                                //COMPARE TEXTURE SIGNATURES
                                dd = distance (hwt_vals, hwt_val, 3*Level);

                                if ((kk/2.55 > 50)&&(dd < 2))
                                {
                                        Obj = 1;
//                                      if (kk > val)
                                        {
                                        val = kk;
//                                      cvZero (ImgZone);

                                        for (ii = yy*delta; ii < (yy*delta + hmax); ii++)
                                                for (jj = xx*delta; jj < (xx*delta + wmax); jj++)
                                                        ImgZone->imageData[ii*ImgZone->width + jj] = kk;
                                        }
                                }
                                cvReleaseMat (&WINcluster);
                                cvReleaseImage (&imgHWT);
                                cvReleaseImage (&imgTmp);
                        }
                        //NEXT REGION
        if ((xx+1)*delta < ImgSrc->width - delta)
            xx++;
        else
        {
            xx = 0; yy++; wmax = sizew;
        }
```

```
        if ((xx*delta + wmax) > ImgSrc->width)
            wmax = ImgSrc->width - (xx*delta);

        if ((yy*delta + hmax) > ImgSrc->height)
            hmax = ImgSrc->height - (yy*delta);

        if ((yy*delta) > ImgSrc->height - delta)
            next = 0;

    } while (next != 0);

    cvReleaseImage (&imgOut);
    cvReleaseImage (&imgGray);

    cvReleaseMat (&cluster);
    cvReleaseMat (&point);
    cvReleaseMat (&center);
    cvReleaseMat (&hwt_val);
    cvReleaseMat (&CHist);

    return Obj;
}
```

C.

Biblioteca para extracción y comparación de características de geometría 3D, basadas en OpenCV.

exTenshi.h

```
#include "cv.h"
#include "highgui.h"

#include "Camera/CameraKinectThread.h"

char pointIsInsideContour(CvContour *contour, int x, int y)
{
	char found_left=0, found_top=0, found_right=0, found_bottom=0;
	int count, i;
	CvPoint *contourPoint;

	if(!contour)return 0;

	count = contour->total;

	for(i=0;i<count;i++)
  {
		contourPoint = (CvPoint *)CV_GET_SEQ_ELEM(CvPoint,contour,i);

		if(contourPoint->x == x)
    {
			if(contourPoint->y < y)found_top = 1;
			else found_bottom = 1;
		}
		if(contourPoint->y == y)
    {
			if(contourPoint->x < x)found_left = 1;
			else found_right = 1;
		}
	}

	return found_left && found_top && found_right && found_bottom;
}

CvContour* contourFromPoint(CvContour *contours, int x, int y)
{
	CvContour *contour;
	CvContour *tmpContour=0, *output=0;

	if(!contours)return 0;

	contour = contours;

	while(contour){
    // Determine if the selected point is inside the contour
		if(pointIsInsideContour(contour, x, y))
    {
			output = contour;

			if(contour->v_next)
      {
				tmpContour = contourFromPoint((CvContour *)contour->v_next, x, y);

				if (tmpContour)
          output = tmpContour;
			}
		}

		contour = (CvContour *)(contour->h_next);
	}

	return output;
}
```

```
void extract_object_world (KinectImages* capturaHilo, CvMat* &centers, CvFeatureTree* &features, int radius, int
MAX_CH, int sizeHIST, int minDepth, int maxDepth)
{
//3D Analisys
	//Distances Histogram
	XnVector3D point;

	IplImage *imgSRC, *imgHIST;
	uchar *data_hist;

	CvRect Cube;
	int ii, jj, kk, xx, yy, CbSide, step, idx;
	float XX, YY, ZZ, DD, MM;

	float range_0[]={1, radius - 1};
	float* ranges[] = { range_0 };
	CvHistogram *Hist = cvCreateHist(1, &sizeHIST, CV_HIST_ARRAY, ranges, 1);

	imgSRC = capturaHilo->getImageBetween(minDepth, maxDepth);

	kk = 0;
	for (xx = 0; xx < imgSRC->width; xx++)
	  for (yy = 0; yy < imgSRC->height; yy++)
	  {
		//DEPTH
		point = capturaHilo->get3DPoint(xx, yy);

		if ((point.Z >= minDepth)&&(point.Z <= maxDepth))
			kk++;
		//*****
	  }


	CvMat *DIST_vals = cvCreateMat (kk, sizeHIST, CV_32FC1);
	CvMat *clusters = cvCreateMat (kk, 1, CV_32SC1);

	centers = cvCreateMat (MAX_CH, sizeHIST, CV_32FC1);

	idx = 0;
	for (ii = 0; ii < imgSRC->height; ii++)
		for (jj = 0; jj < imgSRC->width; jj++)
	  {
		point = capturaHilo->get3DPoint(jj, ii);
		if ((point.Z >= minDepth)&&(point.Z <= maxDepth))
		{
			XX = point.X; YY = point.Y; ZZ = point.Z;

			CbSide = (int)(radius / (0.001716*point.Z + 0.2735));

			Cube.x = jj - CbSide;
			if (Cube.x < 0) {Cube.x = 0; Cube.width = jj;}
			else {Cube.width = CbSide;}

			Cube.y = ii - CbSide;
			if (Cube.y < 0) {Cube.y = 0; Cube.height = ii;}
			else {Cube.height = CbSide;}

			if (jj + CbSide >= imgSRC->width)
			{
				Cube.width = Cube.width + imgSRC->width - jj;
			}
			else
			{
				Cube.width = Cube.width + CbSide;
			}

			if (ii + CbSide >= imgSRC->height)
			{
				Cube.height = Cube.height + imgSRC->height - ii;
			}
			else
			{
				Cube.height = Cube.height + CbSide;
			}
```

```
                                imgHIST = cvCreateImage(cvSize(Cube.width, Cube.height), IPL_DEPTH_8U, 1);
                                data_hist = (uchar *)imgHIST->imageData;
                                step = imgHIST->widthStep/sizeof(uchar);

                                for (xx = Cube.x; xx < Cube.x + Cube.width; xx++)
                                        for (yy = Cube.y; yy < Cube.y + Cube.height; yy++)
                                        {
                                                point = capturaHilo->get3DPoint(xx, yy);

                                                if ((point.Z >= minDepth)&&(point.Z <= maxDepth))
                                                {
                                                  DD = pow(point.X - XX, 2) + pow(point.Y - YY, 2) +
pow(point.Z - ZZ, 2);

                                                  DD = pow(double(DD), 0.5);

                                                  if (DD < radius) {data_hist[(yy - Cube.y)*step + (xx -
Cube.x)] = (int)(DD);}
                                                  else {data_hist[(yy - Cube.y)*step + (xx - Cube.x)] = 0;}
                                                }
                                        }

                                cvCalcHist( &imgHIST, Hist, 0, NULL );
                                cvNormalizeHist( Hist, 100 );

/*                              MM = 0;
                                for (xx = 0; xx < sizeHIST; xx++)
                                        MM = MM + pow( cvGetReal1D(Hist->bins, xx) , 2);

                                MM = (float)(pow(double(MM), 0.5));

                                MM = 1;
*/
                                for (xx = 0; xx < sizeHIST; xx++)
                                        DIST_vals->data.fl[idx * sizeHIST + xx] =  cvGetReal1D(Hist->bins,
xx);

                                idx++;
                                cvReleaseImage (&imgHIST);
                          }
                        }

        cvKMeans2 (DIST_vals, MAX_CH, clusters,
                cvTermCriteria (CV_TERMCRIT_EPS , 10, 1), 1, 0, 0, centers, 0);

        features = cvCreateKDTree(centers);

        FILE* fp = fopen("World.hist", "w");
        for (yy = 0; yy < MAX_CH; yy++)
        {
          for (xx = 0; xx < sizeHIST; xx++)
          {
                        fprintf(fp, "%f ", centers->data.fl[yy * sizeHIST + xx]);
          }
          fprintf(fp,"\n", DD);
        }
        fclose(fp);

    return;
}

int extract_object_scene (KinectImages* capturaHilo, IplImage* &imgFOUND, CvMat *XYZpose, CvMat* &clustersH,
CvFeatureTree* features, int radius, int MAX_CH, int sizeHIST, int minDepth, int maxDepth)
{
    XnVector3D point;

    // Images
    IplImage *imgSRC, *imgMASK, *imgGRAY;
        uchar *data_mask, *data_gray, *data_found;

    int ii, jj, kk, xx, yy, obj = 0, pstep = 20;
        double area = 0, area1 = 0;

    // Secuencias para contornos
        CvMemStorage *storage = cvCreateMemStorage(0);
```

```
CvContour *ctr = 0, *ctrs = 0;
        CvSeq *ctrSEQ = 0;
CvRect rect;

// Crear Ventanas
imgSRC = capturaHilo->getImageBetween(minDepth, maxDepth);

        imgMASK = cvCreateImage(cvGetSize(imgSRC), IPL_DEPTH_8U, 1);
data_mask = (uchar *)imgMASK->imageData;

        imgGRAY = cvCreateImage(cvGetSize(imgSRC), IPL_DEPTH_8U, 1);
        data_gray = (uchar *)imgGRAY->imageData;

        cvCvtColor(imgSRC, imgGRAY, CV_BGR2GRAY);

        rect.x = 0;
        rect.y = 0;
        rect.width = 0;
        rect.height = 0;

XYZpose->data.fl[0] = 0;
XYZpose->data.fl[1] = 0;
XYZpose->data.fl[2] = 0;

for (ii = 0; ii < imgGRAY->height; ii++)
   for (jj = 0; jj < imgGRAY->width; jj++)
   {
      if (data_gray[ii*(imgGRAY->widthStep/sizeof(uchar)) + jj] > 0)
      {
      data_mask[ii*(imgMASK->widthStep/sizeof(uchar)) + jj] = 255;
      obj++;
      }
      else
      data_mask[ii*(imgMASK->widthStep/sizeof(uchar)) + jj] = 0;
   }

        if (obj < 25)
  return 0;

        cvDilate (imgMASK, imgMASK, 0, 10);
        cvErode (imgMASK, imgMASK, 0, 10);

        cvFindContours(imgMASK, storage, &ctrSEQ, sizeof(CvContour), CV_RETR_TREE,
CV_CHAIN_APPROX_NONE, cvPoint(0,0));
ctrs = (CvContour *)ctrSEQ;

pstep = 20;
for (ii = 0; ii < imgMASK->height; ii += pstep)
   for (jj = 0; jj < imgMASK->width; jj+= pstep)
   {
      ctr = contourFromPoint(ctrs, jj, ii);
      if (ctr)
        area = fabs(cvContourArea(ctr, CV_WHOLE_SEQ));

      if ((ctr)&&(area > area1))
      {
        area1 = area;
        rect = cvBoundingRect(ctr, 1);
      }
   }

if (area1 == 0)
  return 0;

if (imgFOUND != 0)
  cvReleaseImage (&imgFOUND);

if ((rect.width > 0)&& (rect.height > 0))
        {
          cvCopy(capturaHilo->getImageRGB(), imgSRC, 0);

          imgFOUND = cvCreateImage( cvSize(rect.width, rect.height), imgSRC->depth, 3);
          data_found = (uchar *)imgFOUND->imageData;

  cvSetImageROI(imgSRC, rect);
```

```
        cvCopy (imgSRC, imgFOUND, 0);
        cvResetImageROI(imgSRC);

//2D Analisys
                kk = 0;
        for (xx = rect.x; xx < rect.x + rect.width; xx++)
           for (yy = rect.y; yy < rect.y + rect.height; yy++)
           {
                                        //IMAGE
                                        if (data_mask[yy*(imgMASK->widthStep/sizeof(uchar)) + xx] == 0)
                                        {
                                                data_found[(yy - rect.y)*(imgFOUND->widthStep/sizeof(uchar)) + 3*(xx -
rect.x) + 0] = 0;
                                                data_found[(yy - rect.y)*(imgFOUND->widthStep/sizeof(uchar)) + 3*(xx -
rect.x) + 1] = 0;
                                                data_found[(yy - rect.y)*(imgFOUND->widthStep/sizeof(uchar)) + 3*(xx -
rect.x) + 2] = 0;
                                        }
                                        //*****

                                        //DEPTH
              point = capturaHilo->get3DPoint(xx, yy);

              if ((point.Z >= minDepth)&&(point.Z <= maxDepth))
              {
                XYZpose->data.fl[0] = XYZpose->data.fl[0] + point.X;
                XYZpose->data.fl[1] = XYZpose->data.fl[1] + point.Y;
                XYZpose->data.fl[2] = XYZpose->data.fl[2] + point.Z;

                                                kk++;
                                        }
                                        //*****
                        }

        XYZpose->data.fl[0] = XYZpose->data.fl[0] / kk;
        XYZpose->data.fl[1] = XYZpose->data.fl[1] / kk;
        XYZpose->data.fl[2] = XYZpose->data.fl[2] / kk;
//*****

//3D Analisys
                //Distances Histogram
                IplImage *imgHIST;
                uchar *data_hist;

                CvRect Cube;
                int CbSide, step, idx;
                float XX, YY, ZZ, DD, MM;

                float range_0[]={1, radius - 1};
                float* ranges[] = { range_0 };
                CvHistogram *Hist = cvCreateHist(1, &sizeHIST, CV_HIST_ARRAY, ranges, 1);

                CvMat *DIST_vals = cvCreateMat (kk, sizeHIST, CV_32FC1);
                CvMat *centers = cvCreateMat (MAX_CH, sizeHIST, CV_32FC1);
                CvMat *distances = cvCreateMat(kk,1,CV_64FC1);
                CvMat *clusters = cvCreateMat (kk, 1, CV_32SC1);

                clustersH = cvCreateMat (sizeHIST, 1, CV_32SC1);

                imgSRC = capturaHilo->getImageBetween(minDepth, maxDepth);

                idx = 0;
                for (ii = 0; ii < rect.height; ii++)
                        for (jj = 0; jj < rect.width; jj++)
                        {
                                point = capturaHilo->get3DPoint(jj + rect.x, ii + rect.y);
                                if ((point.Z >= minDepth)&&(point.Z <= maxDepth))
                                {
                                        XX = point.X; YY = point.Y; ZZ = point.Z;

                                        CbSide = (int)(radius / (0.001716*point.Z + 0.2735));

                                        Cube.x = jj - CbSide;
                                        if (Cube.x < 0) {Cube.x = rect.x; Cube.width = jj;}
                                        else {Cube.x = Cube.x + rect.x; Cube.width = CbSide;}
```

```
                                        Cube.y = ii - CbSide;
                                        if (Cube.y < 0) {Cube.y = rect.y; Cube.height = ii;}
                                        else {Cube.y = Cube.y + rect.y; Cube.height = CbSide;}

                                        if (jj + CbSide >= rect.width)
                                        {
                                                Cube.width = Cube.width + rect.width - jj;
                                        }
                                        else
                                        {
                                                Cube.width = Cube.width + CbSide;
                                        }

                                        if (ii + CbSide >= rect.height)
                                        {
                                                Cube.height = Cube.height + rect.height - ii;
                                        }
                                        else
                                        {
                                                Cube.height = Cube.height + CbSide;
                                        }

                                       imgHIST = cvCreateImage(cvSize(Cube.width, Cube.height), IPL_DEPTH_8U,
1);
                                        data_hist = (uchar *)imgHIST->imageData;
                                        step = imgHIST->widthStep/sizeof(uchar);

                                        for (xx = Cube.x; xx < Cube.x + Cube.width; xx++)
                                                for (yy = Cube.y; yy < Cube.y + Cube.height; yy++)
                                                {
                                                        point = capturaHilo->get3DPoint(xx, yy);

                                                        if ((point.Z >= minDepth)&&(point.Z <=
maxDepth))
                                                        {
                                                          DD = pow(point.X - XX, 2) + pow(point.Y - YY, 2)
+ pow(point.Z - ZZ, 2);
                                                            DD = pow(double(DD), 0.5);

                                                            if (DD < radius) {data_hist[(yy - Cube.y)*step +
(xx - Cube.x)] = (int)(DD);}
                                                            else {data_hist[(yy - Cube.y)*step + (xx -
Cube.x)] = 0;}
                                                        }
                                                }

                                        cvCalcHist( &imgHIST, Hist, 0, NULL );
                                        cvNormalizeHist( Hist, 100 );

                                        for (xx = 0; xx < sizeHIST; xx++)
                                                DIST_vals->data.fl[idx * sizeHIST + xx] =
cvGetReal1D(Hist->bins, xx);

                                        idx++;
                                        cvReleaseImage (&imgHIST);
                                }
                        }

                cvFindFeatures(features, DIST_vals, clusters, distances, 1, 250);

                cvZero(clustersH);
                for (xx = 0; xx < kk; xx++)
                {
                        yy = clusters->data.i[xx];
                        clustersH->data.i[yy]++;
                }
            }

    cvClearMemStorage(storage);

    cvReleaseImage (&imgSRC);
    cvReleaseImage (&imgMASK);
    cvReleaseImage (&imgGRAY);
```

```
    return 1;
}

void extract_object_hist (KinectImages* capturaHilo, CvRect Rect, CvMat* &clustersH, CvFeatureTree* features, int
radius, int MAX_CH, int sizeHIST, int minDepth, int maxDepth)
{
    XnVector3D point;

    // Images
    IplImage *imgSRC;
            imgSRC = capturaHilo->getImageBetween(minDepth, maxDepth);

    int ii, jj, kk, xx, yy;

            if ((Rect.width > 0)&&(Rect.height > 0))
            {
              kk = 0;
      for (xx = Rect.x; xx < Rect.x + Rect.width; xx++)
        for (yy = Rect.y; yy < Rect.y + Rect.height; yy++)
        {
                                    point = capturaHilo->get3DPoint(xx, yy);
                                    if ((point.Z >= minDepth)&&(point.Z <= maxDepth))
                                            kk++;
                            }
//3D Analisys
                    //Distances Histogram
                    IplImage *imgHIST;
                    uchar *data_hist;

                    CvRect Cube;
                    int CbSide, step, idx;
                    float XX, YY, ZZ, DD, MM;

                    float range_0[]={1, radius - 1};
                    float* ranges[] = { range_0 };
                    CvHistogram *Hist = cvCreateHist(1, &sizeHIST, CV_HIST_ARRAY, ranges, 1);

                    CvMat *DIST_vals = cvCreateMat (kk, sizeHIST, CV_32FC1);
                    CvMat *centers = cvCreateMat (MAX_CH, sizeHIST, CV_32FC1);
                    CvMat *distances = cvCreateMat(kk,1,CV_64FC1);
                    CvMat *clusters = cvCreateMat (kk, 1, CV_32SC1);

                    clustersH = cvCreateMat (sizeHIST, 1, CV_32SC1);

                    idx = 0;
                    for (ii = 0; ii < Rect.height; ii++)
                            for (jj = 0; jj < Rect.width; jj++)
                            {
                                    point = capturaHilo->get3DPoint(jj + Rect.x, ii + Rect.y);
                                    if ((point.Z >= minDepth)&&(point.Z <= maxDepth))
                                    {
                                            XX = point.X; YY = point.Y; ZZ = point.Z;

                                            CbSide = (int)(radius / (0.001716*point.Z + 0.2735));

                                            Cube.x = jj - CbSide;
                                            if (Cube.x < 0) {Cube.x = Rect.x; Cube.width = jj;}
                                            else {Cube.x = Cube.x + Rect.x; Cube.width = CbSide;}

                                            Cube.y = ii - CbSide;
                                            if (Cube.y < 0) {Cube.y = Rect.y; Cube.height = ii;}
                                            else {Cube.y = Cube.y + Rect.y; Cube.height = CbSide;}

                                            if (jj + CbSide >= Rect.width)
                                            {
                                                    Cube.width = Cube.width + Rect.width - jj;
                                            }
                                            else
                                            {
                                                    Cube.width = Cube.width + CbSide;
                                            }

                                            if (ii + CbSide >= Rect.height)
                                            {
                                                    Cube.height = Cube.height + Rect.height - ii;
```

```
                                        }
                                        else
                                        {
                                                Cube.height = Cube.height + CbSide;
                                        }

                                       imgHIST = cvCreateImage(cvSize(Cube.width, Cube.height), IPL_DEPTH_8U,
1);
                                        data_hist = (uchar *)imgHIST->imageData;
                                        step = imgHIST->widthStep/sizeof(uchar);

                                        for (xx = Cube.x; xx < Cube.x + Cube.width; xx++)
                                                for (yy = Cube.y; yy < Cube.y + Cube.height; yy++)
                                                {
                                                        point = capturaHilo->get3DPoint(xx, yy);

                                                        if ((point.Z >= minDepth)&&(point.Z <=
maxDepth))
                                                        {
                                                          DD = pow(point.X - XX, 2) + pow(point.Y - YY, 2)
+ pow(point.Z - ZZ, 2);
                                                           DD = pow(double(DD), 0.5);

                                                           if (DD < radius)
                                                           {data_hist[(yy - Cube.y)*step + (xx - Cube.x)] =
(int)(DD);}
                                                           else {data_hist[(yy - Cube.y)*step + (xx -
Cube.x)] = 0;}
                                                        }
                                                }


                                        cvCalcHist( &imgHIST, Hist, 0, NULL );
                                        cvNormalizeHist( Hist, 100 );

                                        for (xx = 0; xx < sizeHIST; xx++)
                                                DIST_vals->data.fl[idx * sizeHIST + xx] =
cvGetReal1D(Hist->bins, xx);

                                        idx++;
                                        cvReleaseImage (&imgHIST);
                                }
                        }

                cvFindFeatures(features, DIST_vals, clusters, distances, 1, 250);

                cvZero(clustersH);
                for (xx = 0; xx < kk; xx++)
                {
                        yy = clusters->data.i[xx];
                        clustersH->data.i[yy]++;
                }

                }
                  //*****

   cvReleaseImage (&imgSRC);
   return;
}

void select_object (IplImage* ImgZone)
{
        // Images
        IplImage *imgMASK, *imgClone;
        CvPoint maxloc;
        CvScalar s;

        // Secuencias para contornos
        CvMemStorage *storage = cvCreateMemStorage(0);
        CvContour *ctr = 0, *ctrs = 0;
        CvSeq *ctrSEQ = 0;
        CvRect rect;

        cvMinMaxLoc( ImgZone, 0, 0, 0, &maxloc, 0 );
```

```
		s = cvGet2D(ImgZone, maxloc.y + 2, maxloc.x + 2);
		if (s.val[0] != 0)
		{
			maxloc.x = maxloc.x + 2; maxloc.y = maxloc.y + 2;
		}
		else
		{
			s = cvGet2D(ImgZone, maxloc.y + 2, maxloc.x - 2);
			if (s.val[0] != 0)
			{
				maxloc.x = maxloc.x - 2; maxloc.y = maxloc.y + 2;
			}
			else
			{
				s = cvGet2D(ImgZone, maxloc.y - 2, maxloc.x + 2);
				if (s.val[0] != 0)
				{
					maxloc.x = maxloc.x + 2; maxloc.y = maxloc.y - 2;
				}
				else
				{
					s = cvGet2D(ImgZone, maxloc.y - 2, maxloc.x - 2);
					if (s.val[0] != 0)
					{
						maxloc.x = maxloc.x - 2; maxloc.y = maxloc.y - 2;
					}
					else
					{
						return;
					}
				}
			}
		}

		// Crear Imágenes
		imgMASK = cvCloneImage(ImgZone);
		imgClone = cvCloneImage(ImgZone);
		cvZero (ImgZone);

		cvFindContours(imgMASK, storage, &ctrSEQ, sizeof(CvContour), CV_RETR_TREE,
CV_CHAIN_APPROX_NONE, cvPoint(0,0));
		ctrs = (CvContour *)ctrSEQ;
		ctr = contourFromPoint(ctrs, maxloc.x + 2, maxloc.y + 2);
		rect = cvBoundingRect(ctr, 1);


		cvSetImageROI(ImgZone, rect);
		cvSetImageROI(imgClone, rect);
		cvCopy (imgClone, ImgZone, 0);
		cvResetImageROI(ImgZone);
		cvResetImageROI(imgClone);

		cvClearMemStorage(storage);
		cvReleaseImage (&imgMASK);
		cvReleaseImage (&imgClone);

		return;
}
```