Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

Proyecto Fin de Carrera

Real-time 3D Segmentation and Object Detection based on a Kinect Sensor

Eladio Estella Nonay

Director: **Michael Blaich**

Ponente: **Luis Montesano del Campo**

Ingeniería en Informática

Junio 2012

A toda la gente que me ha ayudado, apoyado y creído en mí, y en especial, como no, a mi familia y amigos y a mi sobrino al que pronto podremos ver.

Real-time 3D Segmentation and Object Detection based on a Kinect Sensor

Resumen

La actual aparición de sensores de bajo coste que combinan información de color y profundidad de la escena como es el sensor Kinect, ha generado grandes oportunidades para el desarrollo de aplicaciones relacionadas con la visión por computador. Este hecho abre las puertas al avance de su estudio en el ámbito de la robótica de servicio.

Este proyecto ha sido desarrollado en la Hochschule für Tecknik, Wirtschaft und Gestaltung Konstanz y se centra en la creación y análisis de diversos métodos de clasificación de objetos cotidianos para su integración en un robot móvil en los que la información del entorno es recogida utilizando el sensor Kinect. Para ello ha sido necesario un estudio de la literatura del estado del arte en lo que a visión por computador aplicada a robótica se refiere, así como de los métodos de clasificación utilizados. Dichos métodos, que son *Support Vector Machines* y *Adaptive Boosting*, utilizan la información de color y profundidad proporcionada por el sensor para realizar la clasificación.

La idea principal del proyecto es clasificar objetos cotidianos con métodos supervisados que utilicen la información de una base de datos propia creada con el sensor Kinect para crear sus clasificadores, así como la creación o utilización de técnicas de tratamiento de información 2D y 3D para poder realizar la clasificación de los objetos observados por el sensor. Además, se realiza el estudio de los resultados temporales y de clasificación obtenidos mediante diversas técnicas de análisis de resultados.

Debido al deseo de integración en un robot móvil, los métodos y técnicas utilizadas para la realización de los clasificadores han sido cuidadosamente elegidas, con el fin de llevar a cabo dicha clasificación en el periodo más corto posible para así permitir al robot tener tiempo suficiente para realizar otras tareas como la planificación de movimientos o la adquisición de información a través de otros sensores.

Los resultados desprendidos de este proyecto permiten diferenciar los clasificadores para poder elegir el más adecuado a la hora de realizar su integración como sistema de visión del robot móvil, teniendo en cuenta el ámbito de la clasificación de objetos cotidianos.

Índice

1 Introducción

La robótica tiene como objetivo la mejora del modo y la calidad de vida de las personas. Las aplicaciones de la robótica son amplias, tales como el desarrollo científico, trabajo industrial o seguridad. En particular, la robótica de servicio se encarga del desarrollo de robots capaces de asistir a las personas en su día a día, por ejemplo, en tareas domésticas o asistenciales. La detección de objetos cotidianos por un robot es un paso esencial para el desarrollo de este tipo de sistemas dado que requieren la manipulación de objetos. Esta detección de objetos es el primer paso para poder agarrar el objeto, moverlo, evitar un obstáculo o realizar tareas más complejas en las que se requiera una mayor precisión.

Este proyecto ha sido desarrollado en el departamento de robótica de la Hochschule für Tecknik, Wirtschaft und Gestaltung Konstanz (HTWG Konstanz) y se centra en el desarrollo de tres clasificadores de objetos usando el sensor Kinect de Xbox. Se han utilizado dos métodos para realizar la clasificación de dichos objetos, *Adaptive Boosting* (AdaBoost) y *Support Vector Machines* (SVM). La idea principal es crear un sistema de visión que pueda ser capaz de detectar y clasificar ciertos objetos cotidianos para su integración en un robot móvil, por lo que este proyecto se enmarca en el campo de la visión por computador aplicada a la robótica.

1.1 Motivación

El reconocimiento de patrones bebe de diferentes áreas como la inteligencia artificial o la visión por computador e incluye diversos campos tales como el reconocimiento corporal [1], reconocimiento facial [2], reconocimiento de dígitos manuscritos [3], reconocimiento de objetos y texturas [4] o incluso sismología [5]. La aparición de sensores RGB-D de bajo coste, como el sensor Kinect de Microsoft, han favorecido el desarrollo de aplicaciones relacionadas con el tratamiento de imágenes. Kinect ofrece la posibilidad de obtener imágenes de profundidad y color de la escena de forma sincronizada lo que facilita enormemente la investigación en el campo de la robótica, y en especial el de visión por computador aplicada a robótica.

Actualmente, la HTWG Konstanz está investigando con esta tecnología para su implantación en robots móviles. Ésta participa además anualmente en la competición Eurobot, en la cual, en su edición en 2011, la HTWG Konstanz utilizó un sistema de visión basado en la información proporcionada por el sensor Kinect pero todavía en una fase experimental.

Este proyecto fue ideado para realizar un estudio más exhaustivo de las propiedades de Kinect y para crear un nuevo sistema de visión más complejo que el ya existente. De esta manera se consideraría su integración en el robot móvil participante en el concurso para mejorar la fiabilidad y precisión de su sistema de visión. Este aspecto es muy importante, ya que en esta competición se combinan entre otras cosas la visión con la manipulación de objetos.

1.2 Objetivos

El objetivo del proyecto es la creación y comparación de tres reconocedores de objetos cotidianos sencillos utilizando diferentes métodos de clasificación multiclase (SVM, AdaBoost con la

estrategia One-Against-One y AdaBoost con la estrategia One-Against-All) para su integración en un robot móvil que interactúe de forma autónoma con el entorno utilizando el sensor Kinect de Microsoft para adquirir la información. Los tipos de objetos a reconocer son manzanas, plátanos, tazas de café, boles, vasos de plástico y paquetes de tabaco que se sitúan encima de una mesa para su clasificación. Objetos transparentes o semitransparentes como una botella de plástico o una de cristal fueron desechados ya que el sensor no es capaz de adquirir la información de profundidad de todo el objeto, lo que hubiera supuesto trabajar sincronizando la segmentación de la imagen en 3D y 2D con lo que la complejidad y tiempo de cálculo hubiese sido mayor para completar la detección en tiempo real. Para completar el objetivo principal se identifican varios sub-objetivos que son:

- La creación de una base de datos que contenga imágenes de los tipos de objetos que se quieren analizar. Esta base de datos servirá para el entrenamiento y prueba de los sistemas. Existen varias bases de datos disponibles en internet de forma gratuita como Caltech 101 [6], Labelme [7] o Imagenet [8] o incluso realizadas utilizando el sensor Kinect [9], para su utilización en sistemas de reconocimiento de objetos en 3D, pero existen dos principales razones por las que se decantó por la creación de una nueva base de datos con la que poder trabajar. La primera razón es que no se necesitan un número muy grande de clases de objetos en el ámbito del clasificador ya que la idea es tener una base de datos con un número de ellos similar al que se podría encontrar en la competición Eurobot. Además, al tratarse de clasificadores en tiempo real, se prefería tener los objetos físicos pertenecientes a la base de datos para poder realizar pruebas o ajustes con ellos. La segunda razón es que se quería crear un método para la creación de una base de datos utilizando la información que proporciona el sensor Kinect. De esta forma el procedimiento se podría repetir para ampliar la base de datos o crear una nueva con nuevos tipos de objetos de otro ámbito que no fuese el cotidiano.
- El pre-procesamiento de las imágenes para extraer la información útil y eliminar aquella que no corresponda al objeto, en particular la información alejada del sensor y la del plano de la mesa sobre la que se sitúan los objetos utilizando el algoritmo RANSAC.
- La utilización y/o creación de los métodos de clasificación. Además, para realizar la clasificación, se calculan previamente diferentes características de los objetos, siendo fáciles o complicadas de calcular dependiendo del método, que serán geométricas para los métodos de AdaBoost y una mezcla de forma y color para SVM denominadas *VOSCH* (*Voxelized Shape and Color Histograms*). Debido a esta diferencia las características se generan de manera distinta dependiendo del método de clasificación a utilizar.
- La creación de los módulos de entrenamiento y de test del sistema. El primero sirve para enseñar a los clasificadores los tipos de objetos a detectar por el sistema y el segundo para realizar el análisis y comparación de los clasificadores. Cada uno de estos módulos sirve para el entrenamiento y análisis de los clasificadores de forma independiente.
- La búsqueda de un esquema de clasificación en seudo-tiempo real que permita, utilizando el módulo de entrenamiento, la clasificación de nuevos patrones en el menor tiempo posible. Se denominan clasificadores en seudo-tiempo real ya que, aunque se trabaja con varias imágenes por segundo, estas clasificaciones no cumplen ningún requisito temporal específico. Se tiene en cuenta, además, la posibilidad de detectar y clasificar más de un objeto al mismo tiempo en vez de sólo uno. Esto se debe a que normalmente los robots móviles no interactúan con un solo objeto sino con varios a la vez para que, dependiendo de cada objeto, realice una tarea u otra.
- La evaluación de los resultados obtenidos en el módulo de test y la elección de la mejor opción para su integración en el robot. Se realizará teniendo en cuenta la corrección de los resultados utilizando el método *Reciever Operating Characteristic* (*ROC*) y las tasas de acierto y la rapidez de cómputo en los clasificadores en seudo-tiempo real.

1.3 Estructura

El proyecto se divide en tres partes principales. La primera explica la tecnología que se utiliza para la creación de los diferentes módulos y clasificadores, los fundamentos de los métodos empleados así como la utilización de dichos métodos. La segunda presenta cómo se prueban los clasificadores y los resultados que se generan. En la última parte se comentan las conclusiones obtenidas a partir de los resultados y las posibles mejoras. El anexo A contiene la explicación de la arquitectura del sistema y el anexo B la evolución temporal del proyecto. Este documento también contiene el Anexo C, en el cual se presenta la memoria completa en inglés desarrollada en la HTWG Konstanz y en la que se explican en profundidad algunos de los aspectos que aquí se introducen a lo largo de la memoria se hará referencia en el anexo.

2 Desarrollo

Del mismo modo que en la programación, existen varios paradigmas en el ámbito del reconocimiento de patrones, estos son, el sintáctico [10] y el estadístico [11]. El primero se basa en la teoría de los lenguajes formales mientras que el segundo trabaja teniendo en cuenta una cierta incertidumbre en los datos que utiliza. Debido a esto, al usar como métodos de clasificación SVM y AdaBoost se está utilizando el paradigma estadístico, ya que expresan los resultados de un reconocimiento de una forma probabilística. Los dos métodos utilizados son métodos supervisados, es decir, se utiliza un conjunto entrenamiento que contiene instancias de los objetos a analizar para poder crear el clasificador y reconocer posteriormente nuevos patrones de las clases de dicho conjunto. Se puede definir como patrón, una noción abstracta representada por un conjunto de descripciones y que en el caso de la inteligencia artificial se describe normalmente como un vector en el que sus componentes son las características del objeto [12] .

Ahora, se va a proceder a comentar las etapas de las que consta el campo del reconocimiento de patrones en métodos supervisados en la inteligencia artificial que son dos, la fase de entrenamiento y la fase de test. La fase de entrenamiento consta de:

- Recopilación de información: es el primer paso en el cual, se seleccionan las clases de patrones que se quieren reconocer y se crea o elige una base de datos ya existente con dichas clases. Este proyecto se centra en imágenes de objetos aunque los patrones pueden ser dígitos, caras o sonidos. Una vez que se dispone de la base de datos, se procede a dividirla en dos subconjuntos complementarios entre sí: el conjunto de datos de entrenamiento y el conjunto de datos de test. El primero se utiliza para enseñar al clasificador las diferentes clases con las que se quiere trabajar y el segundo para poder realizar pruebas de reconocimiento de esas clases. Existen diversos métodos para realizar esta partición siendo el más importante el denominado *validación cruzada*. Este método repite durante *k* iteraciones el análisis de las predicciones utilizando diferentes subconjuntos y calcula la media aritmética de los resultados de cada iteración para generar el resultado final. En este proyecto se utiliza el método de validación cruzada más elemental, el llamado *método de retención*, en el que la base de datos se divide en dos subconjuntos complementarios y el resultado final se obtiene en una sola iteración.

- Elección del método: en esta etapa se deben elegir las características que se desean tener en cuenta para la descripción de los patrones y el método de clasificación que se desea utilizar. Las características elegidas tienen que ser acordes con los patrones que se quieren analizar. Si se quiere distinguir entre limones y naranjas el color podría ser una buena característica, pero no así en una clasificación entre manzanas grandes y pequeñas, en donde el tamaño o volumen serían más adecuados. El método de clasificación se refiere al algoritmo o conjunto de algoritmos que describen dicho método que se van a utilizar para crear el clasificador.

- Entrenamiento del clasificador: una vez elegido el método, se utiliza el subconjunto de datos de entrenamiento como datos de entrada en el algoritmo elegido que serán los que definirán los parámetros del clasificador.

La fase de test puede dividirse en tres etapas que son:

- Selección de datos: esta etapa es dependiente de la etapa de recopilación de datos de la fase de entrenamiento. Los datos utilizados para validar el clasificador dependen del método de división de la base de datos. Es conveniente recordar que los subconjuntos siempre son complementarios, es decir, un dato utilizado para entrenar al clasificador en una determinada iteración nunca intervendrá en la fase de test de esa misma iteración.

- Clasificación: establecido el subconjunto de datos de test, éstos se utilizan como entrada del método de clasificación elegido. Las características de los datos son extraídas previamente a la utilización del algoritmo correspondiente que generará una salida por cada uno de los datos.

- Validación de los resultados: una vez que la clasificación se ha realizado y se han obtenido los resultados, se procede al análisis de los mismos. Este análisis puede realizarse utilizando diversos métodos entre los que destacan las matrices de confusión, tasas de acierto o el método ROC.

Además de las dos fases anteriores, se puede añadir una más, que describe el comportamiento de un reconocedor de patrones en tiempo real. Su estructura es ligeramente distinta a las anteriores. En el reconocimiento en tiempo real, se recibe la información del entorno ininterrumpidamente que va a ser procesada para obtener el resultado de la clasificación. Se pueden encontrar cuatro etapas que son:

- Lectura de la información: la recepción de información proveniente del entorno se realiza mediante un sensor que puede ser diferente dependiendo de la información que se quiera recibir, por ejemplo, imagen en 2D, 3D o imagen térmica. Para la utilización de estos sensores se necesitan algoritmos que interpreten la información que reciben.

- Pre-procesamiento: es posible que no toda la información recibida del sensor sea útil para la aplicación por lo que es necesario tratar los datos. Un ejemplo sencillo para describir esta etapa sucede en el reconocimiento de objetos en imágenes en 2D. El pre-procesamiento se encarga de descartar el fondo de una imagen (información no útil para el reconocedor) para tener sólo en cuenta los objetos (información útil). Después se realiza la extracción de las características de los datos para realizar el reconocimiento de patrones evitando usar la información completa del objeto. Esta extracción de atributos se suele incluir en la etapa de pre-procesamiento aunque hay veces en las que la diferencia con la aplicación del método no es muy marcada (por ejemplo en redes neuronales) [13].

- Clasificación: una vez que se dispone de los patrones de la información a reconocer se procede a la utilización del método seleccionado para realizar la clasificación. La forma de expresar el resultado de dicha clasificación será diferente dependiendo del método e incluso de la implementación del algoritmo que lo describe.

- Post-procesamiento: después de reconocer a que clase pertenece el patrón reconocido, puede suceder que se requiera más información de dicho patrón. Por ejemplo en el reconocimiento de objetos, podría ser interesante la distancia a la que se encuentra un objeto, su posición en una imagen o el espacio o saber si está en movimiento o no.

Las diferentes fases han servido para establecer las bases del desarrollo del proyecto. En el Diagrama 2.1 y en el Diagrama 2.2 se describe la estructura de las fases de entrenamiento y

reconocimiento en tiempo real respectivamente.

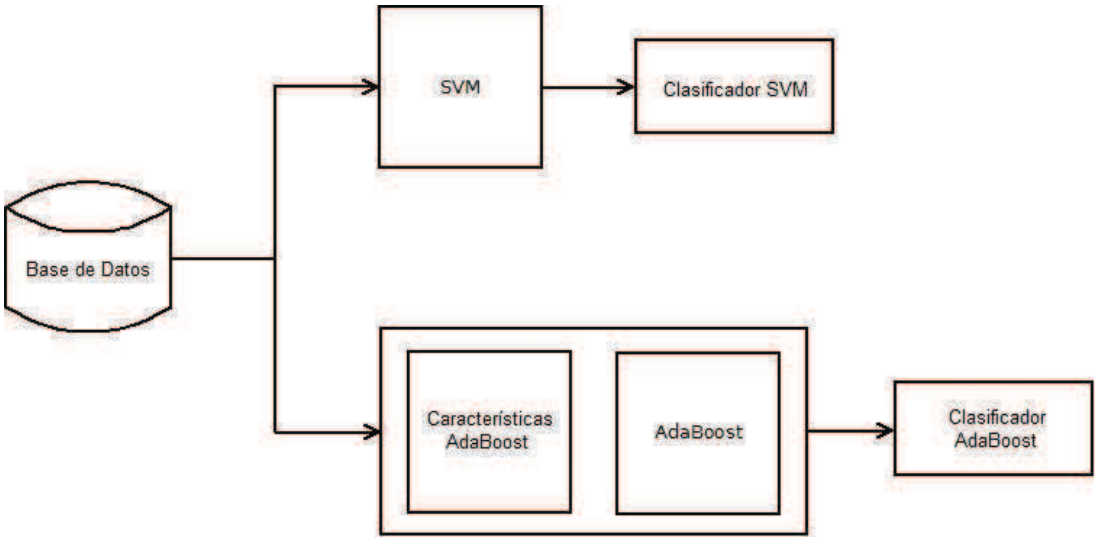

Diagrama 2.1 Estructura del proceso de entrenamiento de los métodos de clasificación.

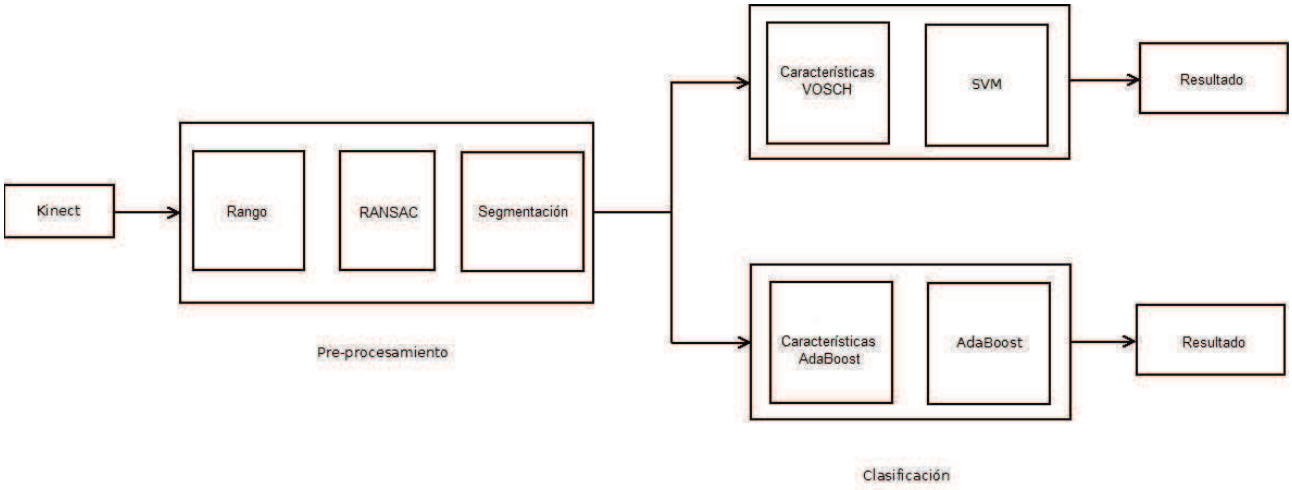

Diagrama 2.2 Esquema de los clasificadores en tiempo real.

A continuación se van a presentar los sistemas, bibliotecas y algoritmos utilizados para el desarrollo de los módulos y clasificadores de este proyecto. Luego se comentarán los métodos utilizados en la creación del clasificador y por último se explicarán las partes que conforman el entrenamiento y el reconocimiento en tiempo real.

2.1 Tecnología

El proyecto ha sido desarrollado en la HTWG Konstanz, en el departamento de Robótica. Se ha construido sobre el sistema operativo Ubuntu versión 10.04 LTS sobre un AMD Athlon 64 X2 Dual Core Processor 4800+. Para posibilitar la interacción con el sensor Kinect de Xbox, cuyas especificaciones se muestran en la Sección 2.1 del Anexo C, se han utilizado las bibliotecas MRPT (Mobile Robot Programming Toolkit) [14], que permiten leer y modificar la información que el

sensor recibe de su entorno. Estas bibliotecas han sido integradas en el sistema ROS (Robotic Operative System) [15], sobre el cual fue creado el reconocedor SVM. Los equipos del departamento de robótica de la universidad de Konstanz tienen a día de hoy el sistema ROS instalado, para facilitar el tratamiento de datos en el ámbito de la robótica y la inteligencia artificial. Para el reconocimiento de objetos mediante SVM, se utilizó la clase cvSVM de Open Source Computer Vision (OpenCV) [16] integrada en ROS, así como la biblioteca VOSCH (Voxalized Shape and Color Histograms) de ROS para la creación de las características para SVM. Se han utilizado diversos algoritmos y estructuras de la biblioteca OpenCV así como de Point Cloud Library (PCL) [17] ya que representan la punta de lanza del desarrollo de aplicaciones en visión por computador [18]. La programación tanto del método de AdaBoost como el de SVM ha sido facilitada gracias al IDE Eclipse.

2.2 Métodos seleccionados

Ya que una de las partes fijadas en este proyecto es la utilización de métodos de cálculo de características que sean complicados o simples dependiendo de la forma de clasificación utilizada, se va a proceder a explicar qué métodos han sido elegidos para la generación del clasificador. La elección de las características que describen un patrón es muy importante ya que es lo que permite trabajar siempre con una cantidad limitada y fija de datos. Este proceso en el que los datos de entrada se reducen para la utilización posterior de los mismos se denomina *reducción de la dimensionalidad*. Existen dos procedimientos para reducir esta cantidad de información, la selección de atributos, en la que simplemente se selecciona las características interesantes desechando el resto y la transformación de características, en la que se crea un nuevo espacio de características a partir de las variables originales. Debido a esto, éste último procedimiento se denomina *extracción de características* [19]. El uso de un conjunto de características limitado simplifica tanto la representación de los patrones como la complejidad del clasificador, lo que hará que éste último sea más rápido y utilice menos memoria [12]. Aunque se puede pensar que es mejor tener un gran número de características para poder hacer una mejor diferenciación, esto es falso. Si bien es cierto que incrementando el número de atributos, la precisión con la que las variables de entrada pueden ser especificadas aumenta, esto conlleva un crecimiento exponencial de la cantidad de datos de entrenamiento que se necesitan para especificar el mapeo de datos. Este fenómeno se denomina *maldición de la dimensionalidad* [20].

En el problema de clasificación de este proyecto, la cantidad de datos de entrada es 640x480x6=1843200, siendo 640x480 el tamaño de la imagen y 6 (3+3), la profundidad (x, y, z) y el color (r, g, b) que describen un punto de dicha imagen. La etapa en la que los datos de entrada de los patrones pasan a tener todos la misma dimensionalidad se llama normalización de los datos.

El departamento de robótica de la HTWG Konstanz quería trabajar con las características VOSCH debido a que se trata de un método nuevo para el cálculo de atributos de nubes de puntos desarrollado en la Universidad de Munich y utilizar AdaBoost con otro tipo de características ya que el departamento había trabajado previamente con este método en problemas de clasificación binaria [21]. Se decidió trabajar con las características VOSCH usando el método de SVM, ya que OpenCV proporciona una implementación multiclase usando la estrategia One-Against-One y porque la compatibilidad entre la forma de descripción de los atributos y su utilización en la clasificación es buena. Como los atributos VOSCH son complejos, para AdaBoost se decidió calcular otros más sencillos, siendo éstos propiedades sencillas, principalmente geométricas, de los objetos en 3D reconocidos.

A partir de ahora se van a explicar uno por uno los métodos de clasificación y de cálculo de atributos utilizados.

2.2.1 Support Vector Machines (SVM)

Support Vector Machines (SVM) es un concepto desarrollado a partir de la teoría de aprendizaje estadístico [11]. Es un sistema que usa un espacio de hipótesis de funciones lineales en un espacio de características de gran dimensionalidad inducido por un *kernel*, entrenado con un algoritmo de aprendizaje que implementa una desviación derivada de la teoría de aprendizaje estadística [22]. La idea del método es crear un hiperplano o conjunto de hiperplanos que permitan crear regiones que sirvan para diferenciar distintas clases para poder utilizarlos en tareas de clasificación o regresión.
En el Aprendizaje Automático, el objetivo de SVM es poder clasificar correctamente un nuevo patrón dentro la región designada para la clase a la que pertenece. Este patrón siempre es distinto a cualquiera que se haya utilizado previamente para el entrenamiento del sistema. Estos patrones o puntos del espacio se expresan normalmente como vectores de dimensión p y las diferentes clases están separadas por un hiperplano de dimensión (p-1). Para realizar la división entre regiones pueden existir infinitos hiperplanos que describan dicha división (Figura 2.1), por lo que es necesario elegir cuál de ellos es el óptimo.

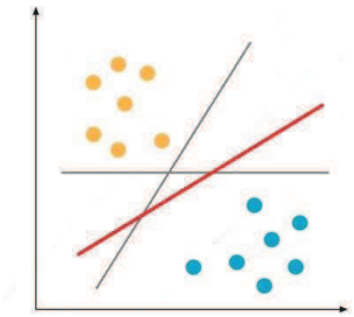

Figura 2.1 Se muestran tres posibles hiperplanos que separan las muestras de dos clases. El hiperplano marcado en rojo define el mejor hiperplano posible al encontrarse a la distancia máxima posible de las instancias más cercanas de ambas clases [23].

Este es el que representa la máxima separación o margen entre las regiones. SVM resuelve el problema de maximización de la distancia del hiperplano a los puntos más cercanos de cada región para el cálculo del hiperplano óptimo. El hiperplano obtenido se denomina hiperplano de margen máximo (Figura 2.2) y el clasificador que describe, clasificador de margen máximo. SVMs son los puntos que se encuentran más próximos al hiperplano. Una introducción a los fundamentos matemáticos de SVM para la clasificación binaria se encuentra en la Sección 2.3.1 del Anexo C.

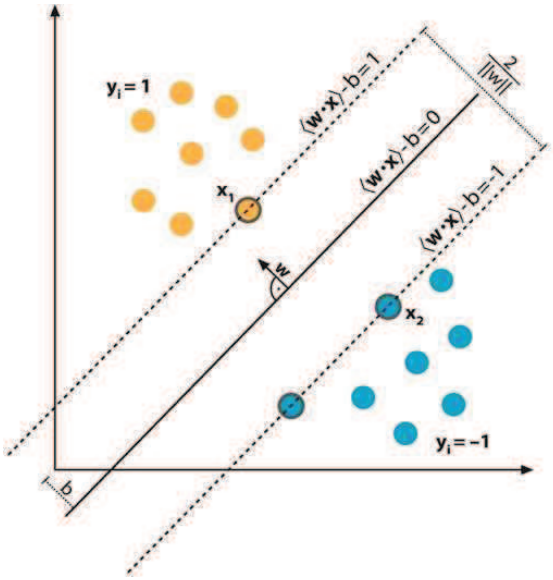

Figura 2.2 La figura muestra el hiperplano de margen máximo con la ecuación que lo describe ($\langle w \cdot x \rangle - b = 0$). Así como los hiperplanos que contienen los SVMs.

2.2.2 Adaptive Boosting (AdaBoost)

AdaBoost es un meta-algortimo utilizado en el aprendizaje de máquinas supervisado. Combina reglas sencillas de clasificación, también llamadas hipótesis, que destacan por su sencillez y poca precisión, para crear un clasificador final mucho más preciso y robusto. Un ejemplo sencillo que explica este principio se encuentra en [24]. Explica como, un corredor de apuesta de carreras de caballos, esperando maximizar sus ganancias decide crear un método para predecir el caballo ganador. Sin tener información alguna es muy difícil crear algún tipo de criterio de predicción, pero recogiendo información de carreras previas puede crear reglas sencillas que le permitan elegir al ganador, tales como el caballo favorito en las apuestas o el caballo que más veces ha ganado recientemente. Estas reglas son inexactas si se usan individualmente, pero combinándolas de manera adecuada se puede crear un procedimiento que acerque al apostante a la mejor predicción.

El algoritmo fue introducido en 1995 por Freund y Shapire [24] y calcula un clasificador final utilizando las hipótesis a las cuales se asigna un peso diferente dependiendo de su capacidad de diferenciación. Los datos de entrada del entrenamiento están formados por los atributos de una muestra $\boldsymbol{x} \coloneqq (x_1, x_2, \dots, x_n)$ y por la salida deseada y que toma valor 1 si la muestra es positiva o -1 si es negativa. Cada clasificador débil comprueba una coordenada de la muestra $\boldsymbol{x}$ y dependiendo de un umbral dado, clasifica la muestra como positiva o negativa. Para ello AdaBoost llama en cada iteración a un clasificador débil que mantiene una distribución o conjunto de pesos sobre el conjunto entrenamiento. Al principio del algoritmo todos los pesos tienen el mismo valor pero en cada ronda, el valor de los pesos de las muestras clasificadas incorrectamente se incrementan y el de las correctamente clasificadas disminuye para que el nuevo clasificador débil se centre en las muestras no clasificadas correctamente. Una vez que la hipótesis es seleccionada de acuerdo con el conjunto de pesos, el algoritmo elige un parámetro α_t que mide la importancia que tiene esa hipótesis sobre el clasificador final. Así pues, el clasificador se compone de la suma de las hipótesis con un peso α dado. Una explicación más profunda de AdaBoost, así como el seudo-algoritmo que describe el método puede encontrarse en Listing 2.1 de la Sección 2.3.2 del Anexo C.

2.2.3 Voxelized Shape and Color Histograms (VOSCH)

Voxelized Shape and Color Histograms (VOSCH) [25] es un método de extracción de atributos

de nubes de puntos que describen objetos en 3D en forma de histogramas. El método combina propiedades de la forma del objeto con propiedades de color. La utilización de estos atributos encaja perfectamente con la información en 3D (forma) y en 2D (color) que se puede obtener sincronizadamente mediante el sensor Kinect.

El método se basa en *Circular Color Cubic Higher-order Local Auto Correlation* (C^3- *HLAC)* [26] con una diferencia que reduce el número de atributos y hace el descriptor invariante respecto a la rotación y en *Global Radius-based Surface Descriptor* (*GRSD)* [27]. El descriptor C^3- *HLAC* es un vector de gran dimensionalidad que calcula la suma de los valores RGB de los voxels vecinos siguiendo varios criterios en una cuadrícula 3x3x3 a lo largo de una cuadrícula de tamaño arbitrario. De esta forma, el histograma contiene información del color de la nube de puntos dependiente de la forma del objeto. GRSD es un histograma que cuenta el número de transiciones entre diferentes tipos de voxels. Las clases de voxels se dividen en espacio, plano, cilindro, esfera, borde y ruido. La variación de C^3- *HLAC* crea 117 atributos mientras que GRSD 20, por lo que la dimensionalidad del descriptor VOSCH es de 137.

Estos descriptores guardan su información en ficheros de extensión pcd (véase Sección 4.1 del Anexo C), que contienen información que detalla nubes de puntos o histogramas. El uso de este tipo de archivos esta extensamente extendido en el ámbito de la programación con OpenCV.

2.2.4 Características en AdaBoost

Como se ha comentado en el Apartado 1.2, el objetivo con AdaBoost es trabajar con atributos sencillos. Para la generación de estos atributos, se calcula la altura, anchura, profundidad y número de puntos que posee la muestra obtenida por el sensor. Hay que remarcar que estos valores, al igual que los que se calcularán posteriormente, no son los que realmente contiene el objeto, si no los que describen la nube de puntos de ese objeto. En este caso el color se ha desechado ya que calcular un atributo sencillo dependiente del color, como el color medio de una observación, no aportaría información relevante o discriminatoria teniendo en cuenta que muchas de las clases pueden tener una gama de colores muy amplia.

Para la creación del clasificador final de AdaBoost, se utilizaron cuatro clasificadores débiles, de puntos, de proporción, de altura y de volumen. En un primer momento, se pensó en utilizar un valor umbral discriminante para cada clasificador, es decir, un valor que dijese si un objeto pertenece a una clase dada si se encuentra dentro de ese umbral, pero siendo una clasificación multiclase la idea no se ajusta bien a los clasificadores One-One, ya que para cada clase existiría un umbral y podría suceder que el rango que describiese dicho umbral estuviese contenido totalmente en el rango de la otra clase. Para los clasificadores One-All, poniendo como ejemplo la clase taza y el clasificador débil altura, se establecería como umbral la altura máxima de una posible taza llevando esto a clasificar como tazas todas las instancias que no lo son. En ambos casos, el error sería tal, que AdaBoost perdería el sentido para su utilización. La solución adoptada es el uso de un rango de valores para cada objeto, en los que para cada clase y clasificador se tendrá un valor mínimo y un valor máximo. Estos valores se obtienen mediante las mediciones del sensor y no con los valores reales del objeto, porque hay que recordar que las mediciones de un sensor no siempre son exactas y que siempre existen partes ocultas para el sensor, es decir, partes del objeto que no puede detectar. Los cuatro clasificadores débiles son:

- *Clasificador de puntos*: este clasificador discrimina los objetos en función de los puntos que contiene. Intuitivamente, se puede pensar que permite discernir entre objetos grandes, con más puntos, y pequeños, con menos puntos, pero además, dos objetos de aproximadamente

las mismas dimensiones pueden diferenciarse en las formas geométricas que componen su relieve. Un objeto con formas cóncavas y convexas contendrá más puntos que uno con formas rectas. Así pues, también se puede considerar como un clasificador simple de superficie de un objeto, aunque sólo sea de la parte que ve el sensor.

- *Clasificador de proporción*: para calcular el clasificador proporción se utilizan tres características principales del objeto, altura, anchura, profundidad; se suman los valores obtenidos del cálculo de la anchura y de la profundidad de la nube de puntos y se divide por la altura, con esto obtenemos el valor de nuestra proporción. Debido a esto los objetos más o menos simétricos tienen un rango de valores para la proporción pequeño, mientras que un rango de valores grande indicará gran diversidad en las instancias de una clase. En un primer momento se pensó en calcular la proporción entre la anchura y la altura del objeto o entre la profundidad y la altura, pero los valores obtenidos podían ser muy diferentes debido a la perspectiva que puede tener la muestra.

- *Clasificador de altura*: el clasificador diferencia simplemente mediante la característica de la altura.

- *Clasificador de volumen*: El clasificador se utiliza para diferenciar los volúmenes de los objetos, teniendo en cuenta que el volumen calculado se realiza usando los datos de la nube de puntos que el sensor distingue y no las dimensiones reales. El volumen se calcula de la siguiente forma; *altura x anchura x profundidad*, por lo que se puede interpretar como el volumen del hexaedro que contiene una nube de puntos dada. Puede existir una diferencia grande entre los límites del rango cuando, dependiendo de la perspectiva de la instancia, existan partes ocultas que añadan un volumen considerable. Esto es, por ejemplo, el asa de una taza, que puede estar oculta en una vista, o parcialmente o totalmente representada en otra.

Estos clasificadores débiles son los utilizados como hipótesis para la creación del clasificador final.

2.3 Entrenamiento

El entrenamiento es la fase en la que se utilizan instancias de las clases que se quieren reconocer posteriormente para enseñar al clasificador. Como muestra el Diagrama 2.1, se va a describir ordenadamente los diferentes módulos del sistema de entrenamiento. Primero se explicará qué es y cuál es la estructura de la base de datos, que es la fuente de las instancias, y después cómo se realiza el entrenamiento de los métodos de clasificación.

2.3.1 Data-set

El Data-set es una base de datos desarrollada específicamente en este proyecto. Consta de diferentes tipos imágenes y archivos de diferentes objetos, con las cuales se enseña y prueba nuestro sistema. Las imágenes fueron tomadas mediante el sensor Kinect, el cual estaba situado a una altura de 38cm sobre la mesa con un ángulo de inclinación de -15º. Se eligieron estos valores porque encajan perfectamente con la posición que tendría el sensor si se acoplase a un robot manipulador móvil mediano. Los objetos fueron colocados a una distancia entre 65 y 87 cm en el caso de profundidad y a una distancia máxima de 10 cm a izquierda y derecha respecto del centro del sensor. La explicación de las posiciones de los objetos sobre la mesa se pueden encontrar en la Sección 3.1.1.1 del Anexo C. El Data-Set creado contiene cuatro carpetas diferentes que son depth,

images, pointclouds y features. La carpeta depth contiene imágenes de tamaño 640x480 que muestra la información de profundidad en escala de grises con extensión JPG. En la carpeta images se encuentran las imágenes en color de tamaño 640x480 que recoge el sensor con extensión JPG. Pointclouds contiene los archivos pcd que describen la nube de puntos de un objeto sin información del entorno y features, los archivos pcd que describen las características calculadas de VOSCH para SVM. El Data-Set contiene seis clases diferentes. La clase apple, banana, bowl, mug, cup y tobacco. Las dos primeras corresponden a frutas, las tres siguientes a recipientes y la última a un paquete de tabaco. Cada una de las carpetas del set contiene una carpeta por cada clase, cuyo nombre es la clase que representan. Dentro de cada clase existen diferentes instancias. El número de instancias es distinto para cada clase. Cada instancia está descrita por ocho tomas, en las que para cada una de ellas se extrae la información de profundidad, de color, de nube de puntos y características VOSCH correspondientes. Los nombres de las imágenes de profundidad vienen dados por una d, de *depth*, seguido por el nombre de la clase a la que pertenecen, y seguido a su vez de un número. El nombre de las imágenes en color se crean uniendo el nombre de la clase con el número correspondiente. Para los archivos de nube de puntos y características añadiremos, respectivamente, una p, de *point cloud*, y una f, de *features*, antes del nombre de la clase. La designación de la numeración de las instancias de cada clase empieza en el 1, y para cada instancia le corresponde un rango de 8 números consecutivos, uno para cada toma del objeto. El rango de la numeración de una instancia de una clase será la misma para las cuatro carpetas existentes, y cada número corresponde a una toma determinada. A modo de ejemplo, apple6.jpg, muestra la sexta imagen en color de la primera instancia de la clase apple, mientras que fapple9.pcd, contiene las características VOSCH de la primera imagen de la segunda instancia de la clase apple. Para ayudar a la visualización de las nubes de puntos de las capturas se utilizó el programa *PCD Viewer* (véase Sección 4.5 del Anexo C).

Para la utilización de la base de datos en las etapas de entrenamiento y test, se utilizará el método de retención, por lo que se elige un subconjunto que contiene instancias de cada clase como conjunto de entrenamiento, y su complementario, que contiene también instancias de todas las clases como conjunto de test. Hay que remarcar que cuando se habla de instancia de una clase se refiere a un objeto concreto y ya que, cada objeto contiene información de ocho tomas, cuando se entrene al clasificador con, por ejemplo, dos instancias de una clase, en realidad se están utilizando 16 imágenes para su entrenamiento, ocho por cada instancia. Las imágenes reales de todas las instancias de las clases, así como el esquema de la estructura de la base de datos se encuentran en Figure 3.2 y Diagram 3.3 de la Sección 3.1.1.1 en el Anexo C respectivamente.

2.3.2 Creación de los clasificadores

Una vez construido el data-set y elegidos los dos subconjuntos para cada método, se procedió al entrenamiento del sistema, que dependiendo del método que se utilice en el módulo se realiza de una forma u otra. Una característica común a todos los métodos es que adquieren la información de entrenamiento siempre de la base de datos. A continuación se comentarán las diferencias entre el entrenamiento realizado para SVM y para las dos estrategias de AdaBoost.

SVM

Para la creación del clasificador SVM, la primera tarea a realizar es la elección del kernel que se quiere utilizar. Esta elección se debe realizar pensando, según el problema de clasificación planteado, con cuál de ellos se puede obtener mejores resultados. Según las conclusiones desprendidas de [28], la decisión es utilizar clasificadores lineales en el método multi-clase de SVM, ya que el número de instancias de entrenamiento es menor al número de características que se utilizan. El conjunto entrenamiento está compuesto por instancias de todas las clases, las cuales

se diferencian mediante la utilización de etiquetas diferentes para cada clase.

AdaBoost

A partir de la descripción para clasificación binaria del algoritmo AdaBoost, éste se extendió para ser capaz de realizar una clasificación multiclase de dos formas diferentes, One-Against-One y One-Agains-All.

En la primera forma, se generan una serie de clasificadores que son creados por todas las parejas posibles de clases de objetos. Siendo n el número de clases y k el tamaño de los conjuntos que se quieren crear, en este caso dos al ser One-One, $\binom{n}{k}$ describe el número de clasificadores totales que se crean. El cálculo de los pesos de los clasificadores débiles se realiza utilizando como casos positivos las instancias de una de las clases y como negativos las instancias de la otra clase.

En la segunda forma, se crea un vector de clasificadores binarios con tamaño el número de clases. El conjunto entrenamiento de cada clasificador-clase está formado por, para los casos positivos las instancias de esa clase del conjunto entrenamiento y como casos negativos las instancias del resto de clases.

2.4 Reconocimiento en tiempo real

Cada clasificador en tiempo real tiene dos partes principales. En la primera se establece la conexión con el sensor Kinect y se crean y entrenan los clasificadores. El entrenamiento se realiza siempre al principio de los clasificadores, antes de iniciarse la conexión con el sensor. Para el caso de SVM se leen instancias de la base de datos y se crea un archivo .xml que contiene el clasificador, que más tarde se leerá para realizar la clasificación. Para AdaBoost se calculan los valores alpha de cada clasificador para las dos variantes usando las instancias del conjunto entrenamiento.

En la segunda, se realiza de forma iterativa el reconocimiento de los objetos. En cada iteración se detectan y clasifican los objetos existentes en una imagen recogida por el sensor. Dentro de cada iteración, se pueden distinguir tres partes. Estas son las presentadas en el Diagrama 2.2: Pre-procesamiento, Clasificación y post-procesamiento.

2.4.1 Pre-procesamiento

El pre-procesamiento es la etapa en la que se tratan los datos antes de proceder a su clasificación. Podemos distinguir tres partes principales. La primera parte es la eliminación del rango de visión del sensor, la segunda, el cálculo de los puntos del plano de la mesa y su eliminación y la tercera, la segmentación de la imagen para la detección de los posibles objetos.

Eliminación del rango

El sensor Kinect tiene un rango de visión de alrededor 3,5 metros. En los clasificadores se ha decidido restringir ese rango de 0,5 metros a 1 metro. Existen dos motivos principales por los que se realizó este paso. El primero es, que no es necesario un rango de visión tan amplio, ya que nuestro interés se centra en objetos situados a una distancia media del sensor. El segundo es, que para una aplicación en tiempo real, el tiempo de cálculo del clasificador en el tratamiento de los datos será

menor cuanto menor sea el número de datos.

Random Sample Consensus algorithm (RANSAC)

Random Sample Consensus algorithm (RANSAC) [29] es un algoritmo desarrollado que estima, dado un conjunto de datos, un modelo matemático deseado. En este proyecto, el modelo matemático buscado es el plano de la mesa, y el conjunto de datos, la nube de puntos de los objetos detectada por el sensor. El principio sobre el que se basa el algoritmo en este proyecto es sencillo. Es un algoritmo iterativo, en el que en cada iteración se elige aleatoriamente un subconjunto de datos del conjunto total que formarán parte del hipotético modelo final. Después se comprueban los puntos restantes, para saber si encajan en ese modelo. La estimación de las iteraciones restantes cambia en cada iteración y significa el número de intentos necesarios para conseguir un conjunto en el que todos los datos pertenezcan al modelo con probabilidad *p*.

Eliminación del plano

En este paso los puntos calculados por el algoritmo RANSAC se eliminan de la nube de puntos final que contiene los objetos a clasificar. Respecto a las nubes de puntos, se puede decir que en el proceso de cálculo del plano con el algoritmo RANSAC, algunos de los puntos de la parte baja de los objetos pueden ser tomados como parte del plano. Esto hace que en objetos relativamente planos como pueden ser los plátanos, muchos puntos del objeto pueden ser eliminados y como consecuencia no se pueda alcanzar el tamaño mínimo necesario para poder ser detectado.

Segmentación

La segmentación es el proceso en el que una imagen se divide en grupos de pixeles, que equivalen a diferentes objetos, para diferenciarlos del fondo de la imagen. Cada uno de estos grupos tiene una etiqueta en común, normalmente un número, que se ocupa de diferenciar entre los grupos de pixeles.

Para realizar la segmentación de la imagen se utiliza un segmentador de imágenes desarrollado en la HTWG Konstanz [30]. Se utiliza tanto la información 2D como la información 3D proveniente del sensor. La segmentación se realiza calculando la distancia euclidea entre los puntos en el espacio que corresponden a dos píxeles vecinos de la imagen en 2D, para saber si pertenecen a un mismo objeto. Dependiendo si un pixel pertenece a un objeto u a otro, se le asigna una etiqueta distinta.

2.4.2 Clasificación

En esta etapa del reconocimiento, dependiendo del método de clasificación utilizado, se extraen las características VOSCH para SVM o las características presentadas en la Sección 2.2.4 para AdaBoost. Después se realiza la clasificación del objeto u objetos que se encuentran en la imagen que se está analizando para obtener el resultado final teniendo en cuenta el tamaño de la nube de puntos que se analiza, para evitar analizar objetos pequeños o ruido u objetos muy grandes.

Un aspecto común de los tres reconocedores es que, si éstos se encuentran con un objeto que pertenece a una clase desconocida, es decir, que no ha sido enseñada antes al sistema, el resultado de la clasificación que se devuelve es una de las clases usadas en el proceso de entrenamiento. Esto se debe a que el resultado obtenido se entiende como la probabilidad más alta de que una nueva muestra pertenezca a una de las clases enseñadas aunque esta probabilidad sea muy pequeña.

2.4.3 Post-procesamiento

En esta etapa, se trata la información del resultado para presentarla de una manera clara. La información se presenta en una ventana que contiene las imágenes de color y de profundidad en escala de grises a la izquierda. En la imagen de profundidad el negro significa los puntos no detectados al estar fuera de rango, pudiendo deberse a puntos que se encuentran demasiado cerca o demasiado lejos del sensor o una medición errónea. La parte central se reserva para pintar los puntos de los objetos. Pulsando la tecla i, podemos cambiar la imagen de profundidad por la imagen en infrarrojos del sensor y viceversa. Existe también la posibilidad de cambiar el ángulo de inclinación del sensor, mediante la tecla 'w', si queremos que el sensor gire hacia arriba, o 's', si queremos que el sensor gire hacia abajo, teniendo en cuenta que en sensor no puede sobrepasar los 31º ni los -31º respectivamente. En la parte superior derecha, se escriben, en orden de detección y uno debajo del otro los nombres de las clases que son reconocidas. En la parte inferior, se indica el grado de inclinación del sensor así como las teclas que se deben pulsar para realizar las acciones anteriormente nombradas. Las ventanas de clasificación se centran en escribir los nombres de las clases reconocidas, dibujar las nubes de puntos de dichos objetos y dibujar un círculo alrededor de ellos en la imagen en color.

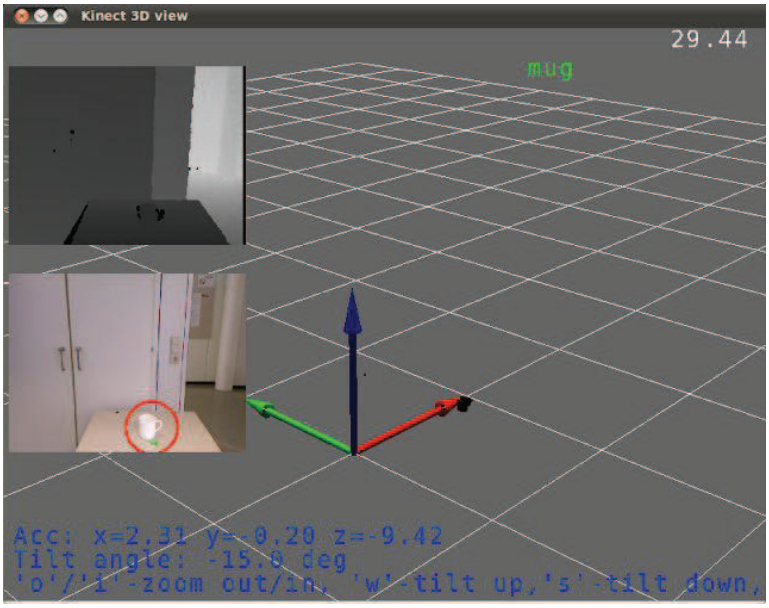

Figura 2.3 Ejemplo de detección de una taza, la cual ha sido rodeada por un círculo rojo en la imagen de color. Encima de ésta se encuentra la imagen en escala de grises que describe la información de profundidad. En el centro la nube de puntos de la taza ha sido dibujada en negro y el nombre de la clase ha sido escrito en verde en la parte superior derecha.

3 Test y resultados

En este capítulo se explica cómo se han desarrollado los programas de test para probar los clasificadores y los resultados obtenidos con dichos programas. Para comentar los resultados se utilizará el método *Receiver Operating Characteristic* (ROC) y tablas con las tasas de acierto de los programas, además de presentar los tiempos de las diferentes fases de reconocimiento obtenidos de los programas en tiempo real.

3.1 Programas de test

Para analizar cómo de buenos son los programas reconocedores se han creado tres programas de test, uno por cada método utilizado. La idea es simple y es la misma para todos los métodos, utilizar el conjunto de entrenamiento para entrenar los clasificadores y el conjunto de test para probarlos. Previamente, se ha realizado una selección de los conjuntos de entrenamiento para cada método entre diferentes conjuntos de entrenamiento (conjuntos con diferente número de instancias para cada clase) como el realizado en [31]. Los conjuntos seleccionados son diferentes para cada método, teniendo siempre instancias de todas las clases e intentando siempre que sea posible disponer de 16 imágenes de cada clase como mínimo para su test. Su selección se realiza comprobando cuál de ellos genera el mejor resultado (Sección 5.1 del Anexo C). La Tabla 3.1 muestra el número de instancias de cada clase para el entrenamiento de SVM, la Tabla 3.2 para AdaBoost One-Against-One (OaO) y la Tabla 3.3 para AdaBoost One-Against-All (OaA).

Training-set	Apple	Banana	Bowl	Mug	Plastic cup	Tobacco
SVM	8	16	24	32	8	16

Tabla 3.1 Número de imágenes de cada clase con las que se entrena al clasificador SVM

Training-set	Apple	Banana	Bowl	Mug	Plastic cup	Tobacco
OaO	8	16	16	16	8	16

Tabla 3.2 Número de imágenes de cada clase con las que se entrena al clasificador AdaBoost One-Against-One

Training-set	Apple	Banana	Bowl	Mug	Plastic cup	Tobacco
OaA	16	40	16	16	16	16

Tabla 3.3 Número de imágenes de cada clase con las que se entrena al clasificador AdaBoost One-Against-All

Una vez seleccionado el conjunto de entrenamiento, y por consiguiente el de test, que será su complementario respecto de la base de datos, se procede al entrenamiento de los clasificadores. Después se utiliza el conjunto de test, leyendo la ruta de las instancias, para generar los resultados de la clasificación. Los conjuntos de entrenamiento aquí seleccionados son los utilizados para el reconocimiento en tiempo real.

3.2 Herramientas de análisis de resultados

Antes de comentar los resultados se van a presentar las ideas en las que se basa el método ROC. Primero se comentarán las matrices de confusión y la información más importante que se desprende de ellas y que sirve para el uso del método ROC. Después se explicará como se utiliza dicho

método.

3.2.1 Matriz de confusión

Una matriz de confusión [32] es un método de representación utilizado en el aprendizaje supervisado que muestra la relación entre las clases reales y las clases predichas por un sistema de clasificación. En la Figura 3.1 se muestra la plantilla de una matriz de confusión para una clasificación binaria.

		Valor de la predicción	
		Positivo	Negativo
Valor real	Positivo	VP	FN
	Negativo	FP	VN

Figura 3.1 Esquema de representación de una matriz de confusión. Los valores reales se representan en las filas y los valores resultado calculados por el clasificador en las columnas.

Las filas representan la clase real de un objeto mientras que las filas representan la clase predicha por el clasificador en donde:

- *Verdadero positivo (VP)* es la predicción correcta de una muestra positiva también denominado éxito.
- *Falso negativo (FN)* es la predicción incorrecta de una muestra positiva, también denominado error de tipo II.
- *Falso positivo (FP)* es la predicción incorrecta de una muestra negativa, también denominado falsa alarma o error de tipo I.
- *Verdadero negativo (VN)* es la predicción correcta de una muestra negativa, también denominado rechazo correcto.

Las entradas de la matriz contienen el número de VPs, FNs, FPs y VNs de un sistema de clasificación. Con estos valores se pueden calcular otros más complejos que se utilizan en el método ROC que son, siendo P el número total de muestras positivas y N el número total de muestras negativas:

- *Tasa de Verdaderos Positivos (TVP)* o *sensibilidad* describe la proporción de las muestras positivas que han sido correctamente clasificadas, calculada usando la ecuación:

$$TVP = \frac{VP}{P} = \frac{VP}{(VP + FN)} \quad (3.1)$$

- *Tasa de Falsos Positivos (TFP)* es la proporción de las muestras negativas que han sido incorrectamente clasificadas, calculada usando la ecuación:

$$TFP = \frac{FP}{N} = \frac{FP}{(FP + VN)} \quad (3.2)$$

Otros valores que se pueden calcular a partir de la matriz de confusión se presentan en el Apartado 2.4.1 del Anexo C.

3.2.2 Receiver Operating Characteristic (ROC)

El método *Receiver Operating Characteristic* (*ROC*) es una representación gráfica entre la tasa de verdaderos positivos (TVP) en función de la tasa de falsos positivos (TFP) en el espacio ROC. El espacio ROC, representado en la Figura 3.2 está definido por dos ejes en el que TFP define el eje x y TVP el eje y. Cada punto dibujado en el espacio ROC representa los resultados de predicción de una clase, es decir, una instancia de la matriz de confusión para una clase determinada.

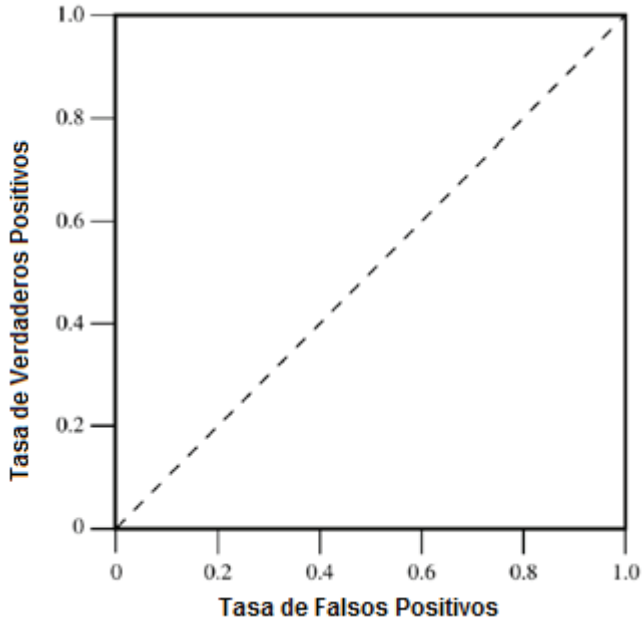

Figura 3.2 Espacio ROC. El eje X se representa por TFP y el eje Y por TVP. La línea de puntos representa el clasificador aleatorio [33]

Puede suceder que un método de clasificación esté definido por un parámetro, por lo que cambiando su valor pueden obtenerse diferentes clasificadores para un mismo método. En este caso, se podrán dibujar diferentes puntos en el espacio ROC, que uniéndolos darán lugar a una curva, denominada curva ROC. Si uno de estos clasificadores predice con un parámetro determinado todas las muestras como positivas, se obtiene el punto (1, 1) en el espacio ROC, y si predice todas como negativas se obtiene el punto (0, 0). El punto que describe la clasificación perfecta es el (0, 1), en el que no existen ni falsos positivos ni falsos negativos. Esto quiere decir que un clasificador será mejor cuanto más cerca esté de este punto.

El espacio ROC está dividido en dos partes por la diagonal ((0, 0)(1, 1)). Un clasificador en la diagonal representa un clasificador aleatorio, si se encuentra por debajo de la diagonal significa que genera peores resultados que el clasificador aleatorio y si está por encima genera mejores resultados. En la Figura 3.3 se muestran ejemplos de diferentes clasificadores:

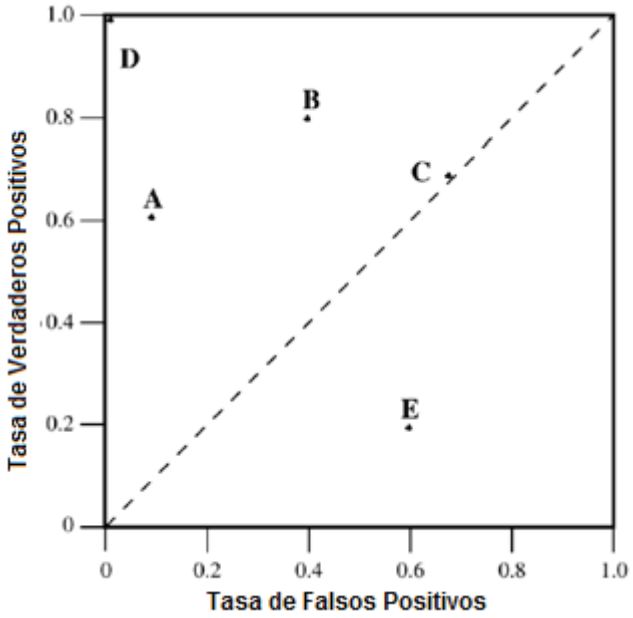

Figura 3.3 Ejemplo de clasificadores, nombrados de A a E en el espacio ROC. D representa el clasificador perfecto, C, el aleatorio y E un mal clasificador [33]

El punto D corresponde con el clasificador perfecto, el punto C con un clasificador aleatorio y el punto E con un clasificador peor que el aleatorio. Se puede añadir que para evitar un clasificador que genera malos resultados como E, se puede crear un clasificador simétrico respecto de la diagonal, E´, en el que los resultados positivos o negativos generados por E sean negativos o positivos respectivamente según E´.

En la Figura 3.4 se muestra un ejemplo de un método de clasificación con un parámetro variable. La aplicación de este tipo de curvas es frecuente en psicología, medicina o radiología, siendo cada vez más usado en el aprendizaje automático y la minería de datos.

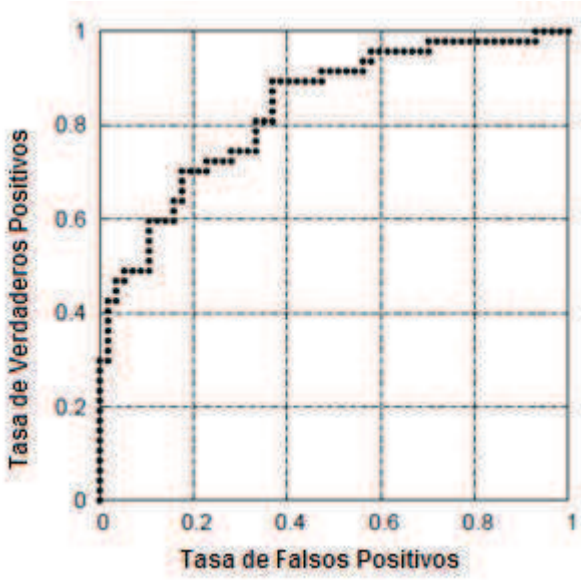

Figura 3.4 Ejemplo de curva ROC para un método paramétrico [34]

3.3 Resultados de la clasificación

Los resultados de la clasificación se obtienen utilizando el conjunto de test de la base de datos de imágenes. Esto quiere decir que las imágenes de test siempre son detectadas y situaciones como la presentada en la Sección 2.4.1, en la que un plátano no es detectado debido a su escaso número de puntos, no son tenidas en cuenta.

Para comenzar con el análisis, se presentan los diagramas ROC de los métodos de clasificación, en los que cada punto representa la clasificación para una clase. El Diagrama 3.1 muestra la información para SVM, el Diagrama 3.2 para AdaBoost One-Against-One y el Diagrama 3.3 para AdaBoost One-Against-All.

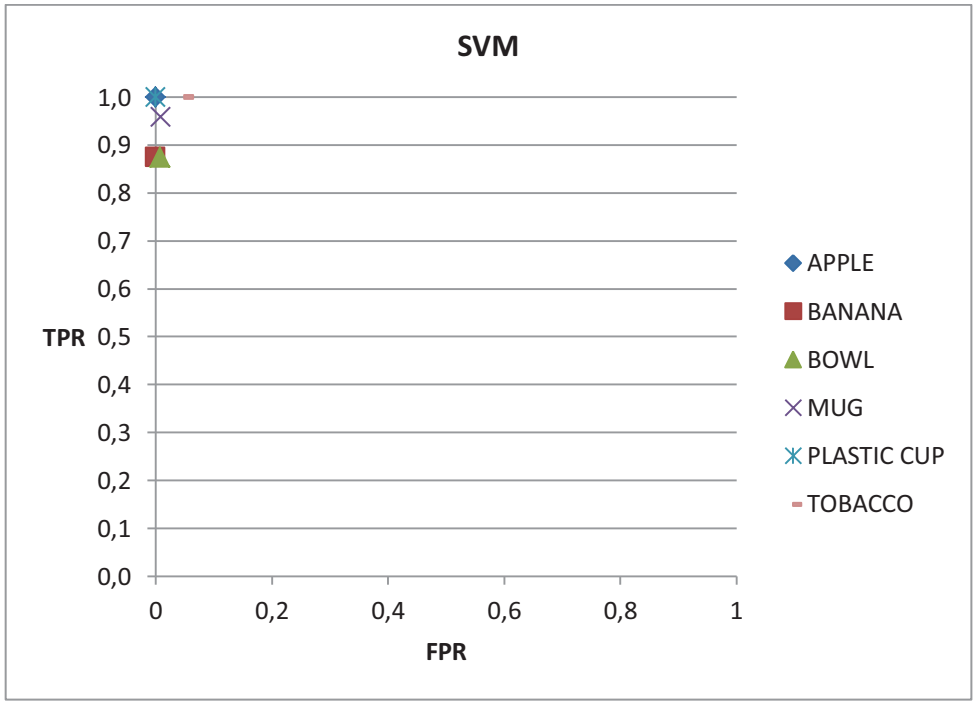

Diagrama 3.1 Representación ROC según clases para el método SVM

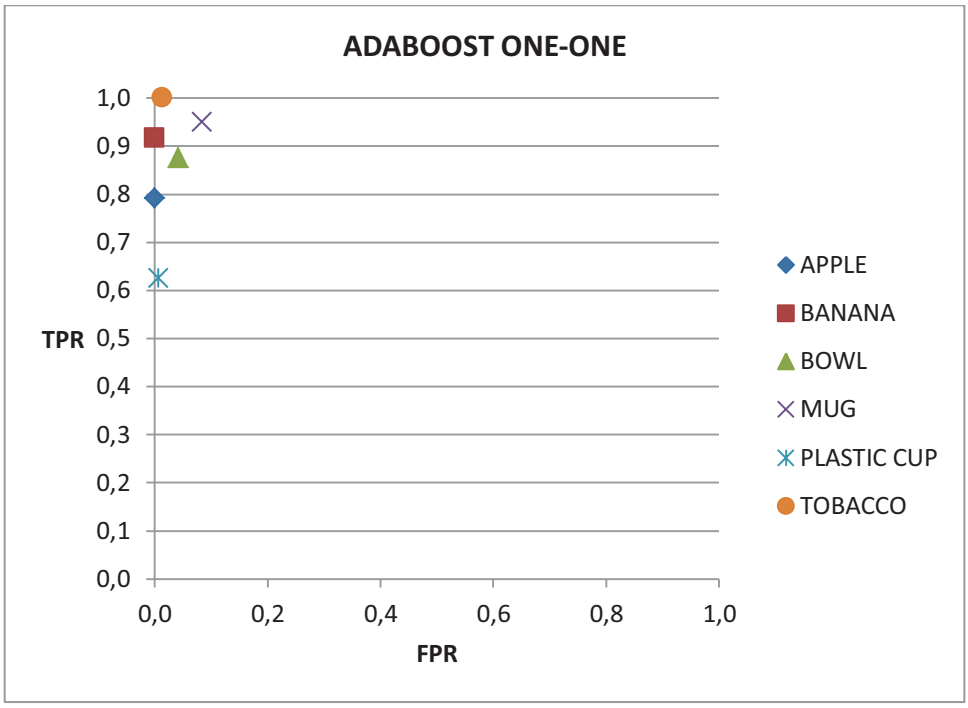

Diagrama 3.2 Representación ROC según clases para el método AdaBoost One-Against-One

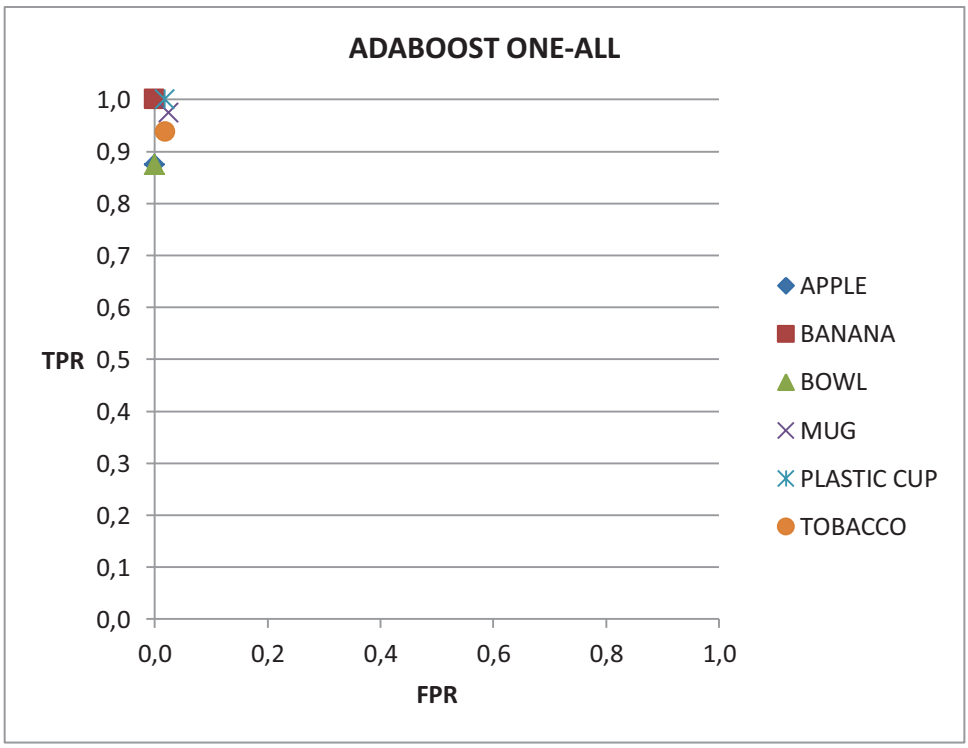

Diagrama 3.3 Representación ROC según clases para el método AdaBoost One-Against-All

El método ROC para SVM muestra como los clasificadores de Bowl y Banana son los que peores resultados generan, siendo los clasificadores Apple y Plastic Cup clasificadores perfectos. Para AdaBoost One-Against-One, los clasificadores Apple y sobre todo Plastic Cup son los peores, siendo el clasificador Tobacco el que mejores resultados proporciona con una clasificación perfecta.

En el caso de AdaBoost One-Against-All, los clasificadores Banana y Plastic Cup generan los mejores resultados siendo clasificadores perfectos, mientras que con los clasificadores Apple y Bowl se obtienen los peores resultados para este método. Los clasificadores de AdaBoost One-Against-One aunque generan resultados relativamente buenos, son peores que los obtenidos para los otros dos métodos al encontrarse más cerca de la diagonal.

Observando las matrices de confusión desprendidas de la clasificación se pueden comentar las equivocaciones más comunes entre clases. En SVM, algunos plátanos son reconocidos como paquetes de tabaco debido a que dependiendo de la orientación del paquete, la nube de puntos generada se asemeja a una nube de puntos que podría crear un plátano. A esto se le añade el hecho de que el método VOSCH es invariante respecto a la rotación (Sección 2.2.3) por lo que en la clasificación no se distingue si el paquete está en posición vertical. Otra confusión que se produce es entre taza y boles debido a la forma parecida que describen los recipientes. Para AdaBoost One-Against-One se producen varios errores de clasificación. Debido a las dimensiones de los objetos, las clases Mug, Bowl y Plastic Cup se confunden entre sí. Dependiendo de la orientación de la muestra ocurre el mismo problema entre las clases Apple, Mug y Tobacco. Por último existe una confusión entre la clase Bowl y Banana debido a la similitud de las nubes de puntos presentadas en Figure 4.7 y Figure 5.8. En cuanto a AdaBoost One-Against-All, los errores de clasificación son prácticamente los mismos que en la estrategia One-Against-One, como se podría esperar al trabajar con los mismos criterios de clasificación. La única diferencia es la clasificación errónea de algunas manzanas como tazas en vez de como vasos de plástico.

Aunque se puede distinguir que los clasificadores en SVM y AdaBoost One-Against-All son mejores que los que describen el clasificador de AdaBoost One-Against-One, se van a concretar los resultados presentando las tasas de acierto de los clasificadores. La Tabla 3.4 muestra las tasas de acierto según el método utilizado y la clase analizada. La primera columna describe el método utilizado, siendo OaO la estrategia One-Against-One y OaA One-Against-All. La última columna describe la tasa de acierto del método en general y el resto de columnas las tasas de acierto para cada clasificador en concreto:

Method	Apple	Banana	Bowl	Mug	Plastic Cup	Tobacco	Overall
SVM	1,0	0,875	0,875	0,958	1,0	1,0	0,951
AdaBoost OaO	0,792	0,917	0,875	0,95	0,625	1,0	0,859
AdaBoost OaA	0,875	1,0	0,875	0,975	1,0	0,938	0,944

Tabla 3.4 La información de las clasificaciones de cada método (filas) se muestran en función de la clase (columnas) en forma de porcentaje. La última columna describe el resultado final teniendo en cuenta todas las clases.

En la Tabla 3.4 se puede comprobar que los resultados de SVM y AdaBoost One-Against-All son similares, siendo mejores que los obtenidos con AdaBoost One-Against-One. Los mejores resultados se obtienen con el método de SVM. La existencia de clasificadores perfectos (tasa de acierto igual a uno) se debe entender, teniendo en cuenta que son los resultados obtenidos fruto de la utilización de la base de datos y no son generalizables. En principio, SVM es el método que debería elegirse debido a dichos resultados pero existe otro factor importante como es el tiempo en el reconocimiento en tiempo real que se debe tener en cuenta. En la Sección 3.4 se estudian los resultados de tiempo de los reconocedores en tiempo real.

3.4 Tiempo de la clasificación

En primer lugar se van a presentar los tiempos totales de entrenamiento de los programas, ya que

conforma la primera parte de los métodos supervisados. En la Tabla 3.5 se muestran los tiempos de todos los métodos:

SVM	AdaBoost OaO	AdaBoostOaA
31,090 ms	2145,699 ms	3985,678 ms

Tabla 3.5 Tiempos totales de entrenamiento dependiendo del método de clasificación.

Ya que como se ha explicado en la Sección 3.1, el número de instancias con las que se entrenan los clasificadores es diferente para cada método, la Tabla 3.6 muestra el tiempo empleado para entrenar al sistema con una instancia genérica. El cálculo se realiza dividiendo el tiempo total entre el número de instancias:

SVM	AdaBoost OaO	AdaBoostOaA
0,299ms	26,821ms	33,214ms

Tabla 3.6 Tiempos de entrenamiento de una instancia genérica dependiendo del método de clasificación.

En este proyecto, el entrenamiento de los clasificadores en tiempo real se realiza una vez han sido lanzados. Para poder ahorrar ese tiempo se podrían entrenar previamente los sistemas y almacenarlos en un fichero para que, al lanzar los programas, cargar simplemente los valores que describen a los clasificadores. Suponiendo esta mejora, los tiempos de entrenamiento de los clasificadores no se van a tener en cuenta a la hora del cómputo total del tiempo.

En el reconocimiento en tiempo real debe existir un compromiso entre la corrección de las clasificaciones y el tiempo empleado para calcularlas. La Tabla 3.7 muestra los tiempos empleados para el método SVM, la Tabla 3.8 para One-Against-One y la Tabla 3.9 para One-Against-All para poder hacer un análisis conjunto de estos dos factores. Cada columna de las tablas representa una etapa del proceso de reconocimiento. La columna *Rango* contiene el tiempo que se necesita para eliminar los puntos que no se encuentran el rango válido elegido y las columnas de *RANSAC* y *Plano* describen el tiempo empleado para calcular el plano mediante el algoritmo RANSAC y el tiempo que se tarda en eliminar dichos puntos respectivamente. *Segmentador* contiene el tiempo que se tarda en etiquetar los puntos válidos de la imagen. *Características* y *SVM* indican respectivamente, el tiempo necesario para calcular las características VOSCH y realizar su clasificación. La columna *Pintado* muestra el tiempo utilizado para dibujar las nubes de puntos en la ventana de la pantalla. Aunque no es necesario este paso para la clasificación, éste se muestra para presentar la información de tiempo del bucle de reconocimiento de la forma más completa posible. Por último *AdaBoost* contiene el tiempo para calcular las características sencillas de AdaBoost y su clasificación. Las filas representan las clases. Cada celda de la tabla contiene el tiempo medio que se tarda en calcular el paso descrito por la columna para la clase expuesta en la fila. La columna *Total* realiza la suma de los tiempos de cada etapa para la clase designada en la primera celda de la fila. La última fila corresponde con la media de tiempos de todas las clases en la etapa descrita en la columna.

	Rango	RANSAC	Plano	Segmentador	Características	SVM	Pintado	Total
Apple	11,316	1,971	8,956	45,770	465,577	3,583	0,138	537,311
Banana	10,424	1,413	14,290	43,606	164,496	3,780	0,073	238,083
Bowl	13,116	2,315	10,897	53,220	638,098	4,575	0,317	722,538
Mug	10,007	1,440	7,664	41,126	334,392	2,704	0,135	397,467
Plastic cup	10,118	1,125	7,624	45,208	230,162	3,285	0,102	297,624
Tobacco	11,157	1,179	7,849	41,610	211,523	2,685	0,091	276,093
	11,023	1,574	9,547	45,090	340,708	3,435	0,143	411,519

Tabla 3.7 Tiempos de reconocimiento con SVM para cada objeto dependiendo del paso en el proceso de reconocimiento en milisegundos. La media de tiempos en milisegundos para cada clase y para cada paso se describe en la última columna y última fila respectivamente.

	Rango	RANSAC	Plano	Segmentador	AdaBoost	Pintado	Total
Apple	34,279	3,829	14,567	62,287	16,243	0,180	131,384
Banana	31,332	3,635	13,750	62,002	12,698	0,107	123,525
Bowl	34,052	5,793	14,020	74,938	21,115	0,436	150,354
Mug	29,018	4,506	12,716	59,601	15,056	0,382	121,278
Plastic cup	31,041	5,006	12,418	55,041	14,751	0,195	118,452
Tobacco	29,600	3,500	11,902	55,285	11,020	0,174	111,481
	31,554	4,378	13,229	61,526	15,147	0,246	126,079

Tabla 3.8 Tiempos de reconocimiento con AdaBoost One-Against-One para cada objeto dependiendo del paso en el proceso de reconocimiento en milisegundos. La media de tiempos en milisegundos para cada clase y para cada paso se describe en la última columna y última fila respectivamente.

	Rango	RANSAC	Plano	Segmentador	AdaBoost	Pintado	Total
Apple	30,097	3,883	13,620	54,825	12,655	0,201	115,281
Banana	30,335	3,955	14,834	59,864	12,274	0,136	121,397
Bowl	31,284	5,905	12,377	56,709	16,405	0,476	123,155
Mug	31,306	5,313	12,925	60,974	15,383	0,463	126,363
Plastic cup	31,481	3,660	15,335	67,658	14,736	0,193	133,062
Tobacco	29,380	3,654	13,332	56,853	12,127	0,177	115,522
	30,647	4,395	13,737	59,480	13,930	0,274	122,463

Tabla 3.9 Tiempos de reconocimiento con AdaBoost One-Against-All para cada objeto dependiendo del paso en el proceso de reconocimiento en milisegundos. La media de tiempos en milisegundos para cada clase y para cada paso se describe en la última columna y última fila respectivamente.

La Tabla 3.7 muestra una diferencia considerable en la extracción de características dependiendo del objeto, que va desde 164ms a 638ms, Esto se debe a que el cálculo realizado por el método VOSCH depende de la forma del objeto y de la transición entre los diferentes tipos de puntos [25], es decir, cuantos más puntos y formas tenga un objeto, más tiempo se necesitará para su cálculo.

En el caso de que se detecten más de un objeto, al tiempo total de una de los reconocimientos habría que añadir el tiempo de cálculo de características y de aplicación del método de clasificación del otro objeto. A modo de ejemplo, suponiendo que con SVM se detectan dos boles, habría que

añadir al tiempo empleado para el reconocimiento de uno de ellos, que es 722ms, el tiempo de la extracción de características VOSCH y el de la clasificación del otro que son 638ms y 4,5ms respectivamente. Por lo tanto el tiempo final sería de 722ms + 638ms + 4,5ms= 1,3s.

Comparando los tiempos de los métodos de AdaBoost en la Tabla 3.8 y la Tabla 3.9, se encuentra una pequeña diferencia. La razón radica en el número diferente de clasificadores que utilizan. Mientras la estrategia One-Against-One utiliza seis, uno por cada clase, la estrategia One-Against-All utiliza 15, $\binom{6}{2}$ (Seccion 2.3.2). Aunque algunos clasificadores de la estrategia One-Against-One son triviales (Seccion 2.3.2) y su cálculo rápido, el número de clasificadores es 2,5 veces mayor.

La Figura 3.5 muestra una representación gráfica de las diferencias entre métodos para cada etapa. Para SVM, el cálculo de VOSCH y la clasificación se describen en *Método*. El eje x indica el paso en la clasificación y el eje y el tiempo en ms.

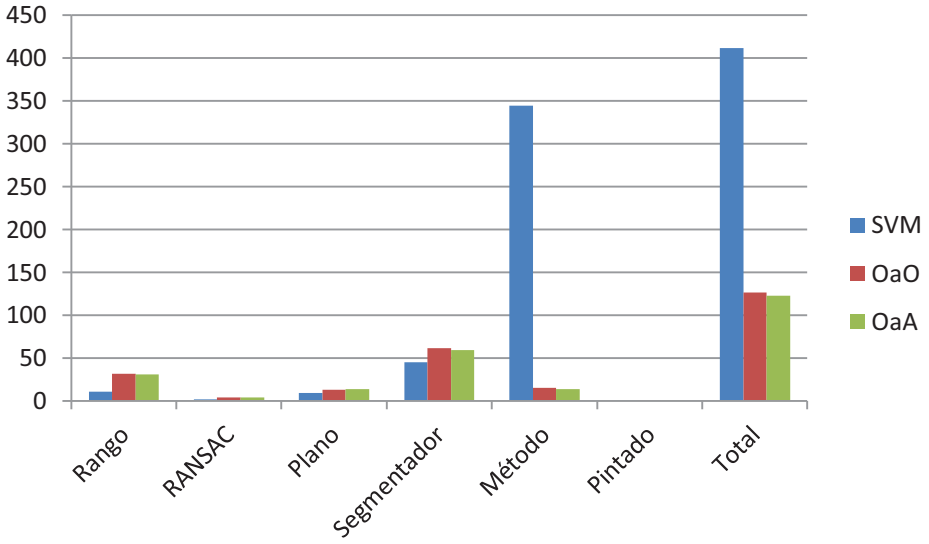

Figura 3.5 Cada parte del proceso de reconocimiento se describe en el eje X. Método indica la suma del cálculo de las características y de la aplicación del método y total muestra la suma de los tiempos de los pasos anteriores para cada método de clasificación.

Teniendo en cuenta la información de la Figura 3.5, SVM puede procesar de 1-3 imágenes por segundo y los métodos de AdaBoost de 6-8. Considerando que un robot móvil debe realizar otras tareas simultáneamente como planificación o adquisición de información de otros sensores, la opción preferida sería la que menos tiempo necesitase, ya que, permitiría al robot utilizar más tiempo en otras tareas.

De acuerdo con los resultados, el método seleccionado debería ser AdaBoost para poder clasificar los objetos lo antes posible.

4 Conclusiones

El objetivo principal del proyecto ha sido la implementación de un sistema de clasificación con información 3D que trabaje con tres métodos diferentes y permita su comparación para poder realizar la elección de la técnica más adecuada para su integración en un robot móvil. Este objetivo se ha cumplido satisfactoriamente identificando adecuadamente los sub-objetivos requeridos. Se ha creado una base de datos que ha permitido el suministro de información para el entrenamiento y test del sistema y se ha descrito la estructura y las características de la nueva base de datos. Se han identificado los módulos de entrenamiento y de test, que realizan el entrenamiento y test del sistema respectivamente, así como las partes de tratamiento de imágenes que se encargan del pre-procesamiento de la información adquirida por el sensor. Se ha alcanzado un esquema de seudo-tiempo real que libera de cálculos no críticos al sistema y que permite su implementación en un robot móvil. Por último, se han utilizado los métodos de análisis de resultados propuestos para realizar la elección del método de clasificación más adecuado.

Respecto a los resultados obtenidos, como se ha visto en la Sección 3.3 y en la Sección 3.4, la elección del método será SVM o AdaBoost dependiendo del criterio que se prefiera. Para encontrar la mejor solución se deben tener en cuenta ambos criterios. Para empezar se descarta AdaBoost One-Against-One, ya que su tasa de aciertos es alrededor del 85% mientras que la de los otros dos métodos es del 95% (Tabla 3.4).

Sobre los resultados de tiempos, existe una gran diferencia entre SVM y AdaBoost. El principal problema de SVM radica en las características que utiliza, ya que el tiempo que se necesita para calcularlas es muy grande. La solución más sencilla que se podría adoptar, sería la utilización de un procesador más rápido para poder realizar los cálculos más rápido. Una segunda solución sería cambiar el tipo de características a calcular ya que el tiempo que necesita para procesar una nube de puntos es muy elevado. En un reconocimiento de varios objetos a la vez, el tiempo necesitado sería demasiado alto para una aplicación en tiempo real. Sin embargo, VOSCH es una buena solución para problemas en los que no se requiere computación en tiempo real como puede ser el análisis de vídeos o para problemas en los que el tiempo no es un factor crítico. Respecto a la diferencia de tiempos de los clasificadores AdaBoost, se debe al número de clasificadores como se ha comentado en la Sección 3.4.

Los resultados de la detección obtenidos con SVM y AdaBoost One-Against-All son similares, mientras que el tiempo empleado es mucho menor para AdaBoost One-Against-All. Por estas razones y por la búsqueda de compromiso entre corrección y tiempo, la opción seleccionada para ser implantada en un robot móvil es AdaBoost con la estrategia One-Against-All. Este es el método que mejor se ajusta al problema, aunque la opción de SVM debe ser tenida en cuenta si se produce un cambio en la forma de generar los patrones a clasificar para reducir su tiempo.

Ahora se van a comentar varias ideas que servirían para mejorar o completar el sistema creado:

- Ampliación de la base de datos de objetos. En comparación con otras bases de datos existentes, la creada aquí es pequeña, pero cumple su función de entrenamiento y test del sistema. De todas formas, se pueden realizar varias mejoras. Una buena idea es ampliar la base de datos añadiendo más instancias para cada clase o añadiendo más capturas de cada objeto (tomando por ejemplo 16 imágenes en vez de ocho con rotación cada 22,75º). Otra

idea es crear una nueva base de datos siguiendo el método aquí expuesto en el que el ámbito de las clases esté acotado, como podría ser frutas, recipientes o elementos del concurso Eurobot.

- Aumento del número de clasificadores AdaBoost. El número de los clasificadores débiles de AdaBoost no es muy alto, por lo que se podría considerar la posibilidad de añadir nuevos, para que el clasificador final sea más complejo y se puedan analizar los nuevos resultados que dicho clasificador produzca. Hay que considerar que los resultados, aunque más costosos en cuanto a tiempo debido al mayor número de operaciones, podrían ser presumiblemente mejores.

- Reconocimiento de la clase desconocida. El clasificador siempre devuelve un resultado si detecta un objeto, por lo que se genera una clasificación errónea cuando el clasificador recibe información de un objeto perteneciente a una clase no enseñada previamente. Por esto, se podría implementar el sistema de tal forma que fuese capaz de identificar objetos que no pertenezcan a ninguna de las clases conocidas.

Bibliografía

[1] J. Shotton, A. Fitzgibbon, M. Cook, T. F. M. Sharp, R. Moore, A. Kipman and A. Blake, "Real-Time Human Pose Recognition in Parts from Single Depth Images".

[2] J. Ponce and A. Karahoca, State of the Art in Face Recognition, I-Tech Education and Publishing, 2009.

[3] A. K. Seewald, «On the Brittleness of Handwritten Digit Recognition Models,» 2011.

[4] D. Filliat, E. Battesti, S. Bazeille, G. Deceux, A. Gepperth, L. Harrath, R. Pereira, A. Tapus, C. Meyer, S.-H. Ieng, R. Benosman, E. Cizeron, J.-C. Mamanna and B. Pothier, "RGBD object recognition and visual texture classification for indoor semantic mapping".

[5] School of Geology & Geophysics University of Oklahoma, "http://geology.ou.edu/aaspi/".

[6] "http://www.vision.caltech.edu/Image_Data_sets/Caltech101/".

[7] «http://labelme.csail.mit.edu/index.html».

[8] "http://www.image-net.org".

[9] K. Lai, L. Bo, X. Ren y D. Fox, «A Large-Scale Hierarchical Multi-Viwe RGB-D Object Dataset».

[10] H. Bunke, "Structural an Syntactic Patter Recognition," in *Handbook of pattern recognition & computer vision*, World Scientific, 1993, pp. 163-209.

[11] V. Vapnik, The Nature of Statiscal Learning Theory, Springer, 2000.

[12] M. N. Murty and V. S. Devi, Pattern Recognition An Algorithmic Approach, Springer, 2011.

[13] C. M. Bishop, Neuronal Networks for Pattern Recognition, Oxford University Press, 2010.

[14] "http://www.mrpt.org".

[15] "http://www.ros.org/wiki/".

[16] OpenCV community, "http://opencv.willowgarage.com/wiki/".

[17] PCL developers, "http://pointclouds.org/".

[18] F. Tombari and L. Di Stefano, "Object recognition in 3D scenes with occlusions and clutter by Hough voting," 2010.

[19] F. Escolano, P. Suau and B. Bonev, Information Theory in computer Vision and Pattern Recognition, Springer, 2009.

[20] R. Bellman, Adaptive Control Processes: A Guided Tour, New Jersey: Adaptive Princeton Press, 1961.

[21] J. Hensler, «Bild- und laserbasierte Onjekterkennung am Beispiel der Türerkennung durch einen mobilen Roboter».

[22] N. Cristianini and J. Shawe-Taylor, Support Vector Machines and other kernel-based learning methods, Cambridge University Press, 2000.

[23] B. Universität, "http://www.precision-crop-protection.uni-bonn.de".

[24] Y. Freund and R. E. Schapire, "A short introduction to boosting," *Journal of Japanese Society for Artificial Intelligence,* pp. 771-778, 1999.

[25] A. Kanezaki, Z. Marton, D. Pangercic, T. Harada, Y. Kuniyoshi and M. Beetz, "Voxelized Shape and Color Histograms for RGB-D".

[26] A. Kanezaki, T. Suzuki, T. Harada and Y. Kuniyoshi, "Fast Object Dtection for Robots in a Clutteres Indoor Enviroment Using Integral 3D Feature Table," in *Proc. IEEE ICRA*, 2011.

[27] Z.-C. Marton, D. Pangercic, R. B. Rusu, A. Holzbach and M. Beetz, "Hierarchical Object Geometric Categorization and Appearance Classification for Mobile Manipulation," in *Proc. IEEE Int. Conf. on Humanoid Robots*, 2010.

[28] C.-W. Hsu, C.-C. Chang and C.-J. Lin, "A Pratical Guide to Support Vector Classification," 2010.

[29] M. A. Fischler and R. C. Bolles, "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography," *Communications of the ACM,* vol. 24, no. 6, pp. 381-395, 1981.

[30] M. Greuter, M. Rosenfelder, M. Blaich and O. Bittel, "Obstacle and Game Element Detection with the 3D-Sensor Kinect," *International Conference on Research and Education in Robotics,* vol. 161, 2011.

[31] G. M. Foody and A. Mathur, "A Relative Evaluation of Multiclass Image Classification by Support Verctor Machines," *IEEE Transactions on Geoscience and Remote Sensing,* vol. 42, no. 6, pp. 1335-1343, 2004.

[32] R. Kohavi and P. Foster, "Glosary of terms," in *Editorial for the Special Issue in Applications of Machine Learning and the Knowledge Discovery Process*, 1998.

[33] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognition Letters,* vol. 27, pp. 861-874, 2005.

[34] J. J. Rodríguez Diez, "Métodos y Tëcnicas de Minerïa de Datos. Metodología experimental".

[35] OpenKinect members, "http://openkinect.org".

Anexo A
Arquitectura del sistema

En esta sección se presentan las diferentes partes o módulos que describen el sistema y la relación que existe entre ellos. Primero se parte de una descripción general del sistema y después se comentan los módulos principales que pertenecen a dicho sistema. Por último se presentan las clases utilizadas más importantes.

El Diagrama A.1 muestra la relación general entre los módulos y las partes del sistema en tiempo real. Para poder utilizar la información que el sensor recoge del exterior se utiliza *Mobile Robot Programming Toolkit* (MRPT) [14]. MRPT proporciona un conjunto de librerías para el desarrollo de la robótica móvil. En este proyecto, la utilización de MRPT se centra en su relación con la librería *libfreenect* de openKinect [35] que permite a MRPT acceder a la información del sensor y en la utilización de las estructuras de datos y algoritmos referentes a la visión por computador que representan la información (Sección 4.2 del Anexo C).

Diagrama A.1 Esquema general del clasificador en tiempo real. Los diferentes módulos son los que proporcionan la información al clasificador para generar un resultado.

La función del test es comprobar el rendimiento del sistema a partir de la información de la base de datos tal y como se muestra en el Diagrama A.2. El módulo de entrenamiento se utiliza como entrada de los clasificadores para permitir generar los resultados del test.

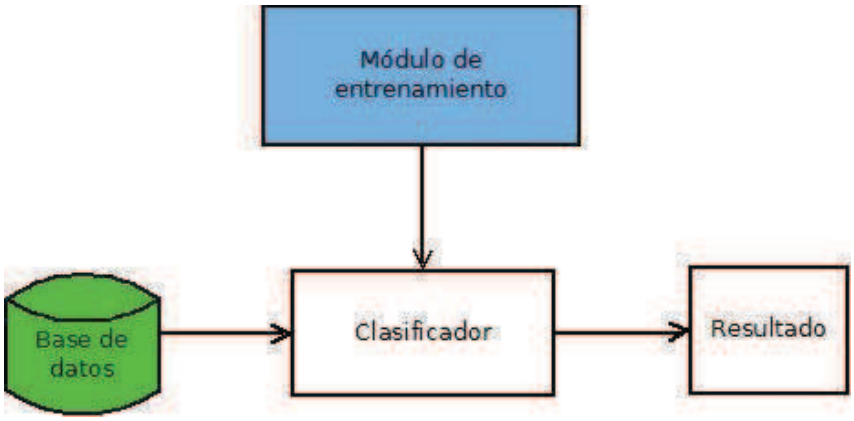

Diagrama A.2 Esquema que describe el test del sistema. Se muestra cómo para el análisis del clasificador se utiliza la información de la base de datos y el módulo de entrenamiento.

Para especificar qué partes han sido implementadas y en cuáles se han utilizado bibliotecas ya existentes, los diagramas muestran las partes implementadas en color verde, las partes que utilizan bibliotecas ya existentes en gris y los módulos en color azul.

El módulo de tratamiento de la imagen recibe como entrada la información proveniente del sensor y se encarga de tratar dicha información para hacer que el clasificador sea capaz de interpretarla para realizar su clasificación. El módulo de entrenamiento se encarga de proporcionar la información que permita al clasificador reconocer las diferentes clases de objetos. Con la información que el clasificador recibe de ambos módulos es capaz de proporcionar un resultado.

Módulo de tratamiento de imágenes

El módulo de tratamiento de imágenes se encarga de transformar la información del sensor Kinect para que pueda ser interpretada por un clasificador. El Diagrama A.3 muestra el esquema de este módulo, en el que primero se elimina de la información proveniente del sensor, los datos según un criterio de rango en la imagen utilizando las estructuras de datos de MRPT. A continuación se identifica y elimina el plano de la mesa, para lo cual se utiliza la clase *HTWGRansac*. La segmentación de la imagen que proporciona los objetos detectados se realiza gracias al método programado en [30]. La última parte del método es la extracción de características. Las características VOSCH se obtienen gracias a la biblioteca *vosch* que se encuentra integrada en Robot Operating System (ROS) [15] creando un fichero externo de extensión pcd que describe dichas características. ROS es un sistema que proporciona librerías y herramientas para el desarrollo de aplicaciones de robótica y en este caso de tratamiento de imágenes e información en 3D.

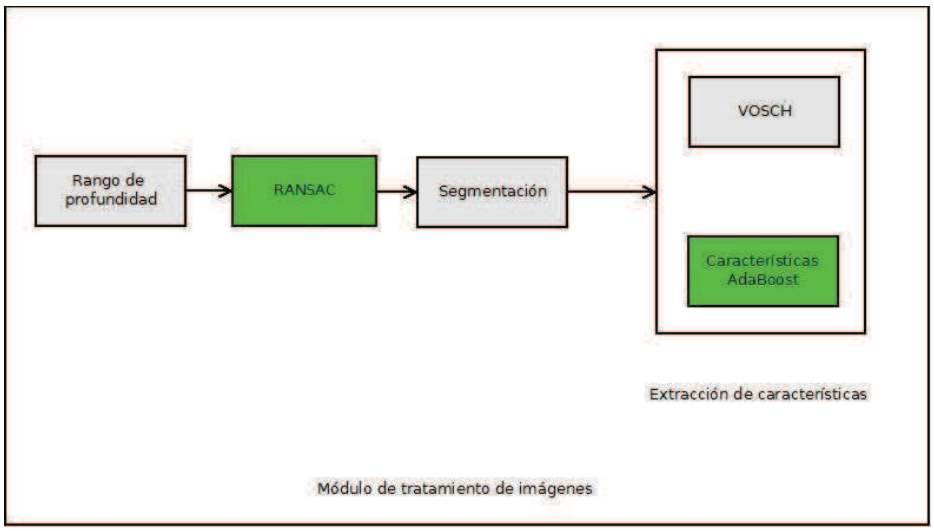

Diagrama A.3 Esquema de módulo de tratamiento de imágenes. Se muestran los cuatro pasos que hacen que la información del sensor sea legible por los clasificadores eliminando los datos no útiles para la clasificación

Este módulo se encarga del procesamiento de información proveniente del sensor Kinect para pueda ser tratada directamente por un sistema de clasificación. Para el tratamiento de información descrito mediante nubes de puntos se utiliza Point Cloud Library (PCL) [17], que es una librería que facilita su tratamiento y que está integrada en MRPT y ROS.

Módulo de entrenamiento

El módulo de entrenamiento es el encargado de crear la información necesaria para permitir a un

clasificador establecer los criterios de diferenciación entre clases. La base de datos proporciona la información a cada uno de los métodos para establecer dichos criterios. Para AdaBoost, el método de clasificación está descrito por la clase *AdaBoost*, mientras que para SVM se utiliza la clase *cvSVM* de Open Source Computer Vision (OpenCV) [16]. OpenCV es una librería que contiene el estado del arte de los algoritmos y estructuras aplicadas a la visión por computador en tiempo real. En el Diagrama A.4 se muestra el esquema del módulo de entrenamiento. La información que describe los criterios de clasificación de SVM se describe utilizando un archivo xml.

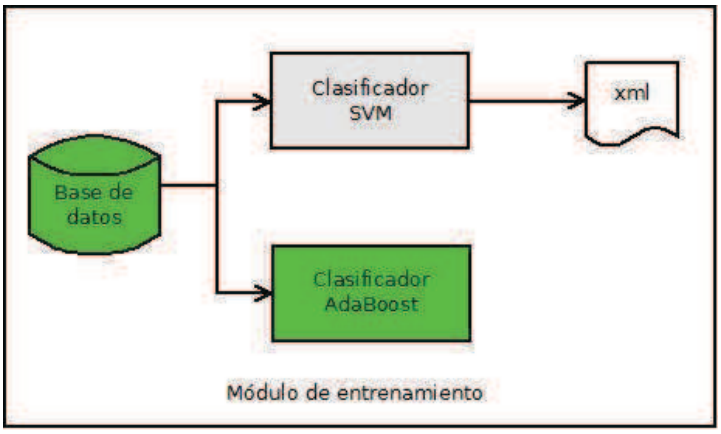

Diagrama A.4 Esquema del módulo de entrenamiento. Se muestra cómo la base de datos suministra la información a los métodos de clasificación para crear los clasificadores

Este módulo puede utilizarse de forma independiente para la creación de clasificadores o bien, como se muestra en el Diagrama A.2 puede utilizarse como entrada de un sistema de test.

Aspectos especiales de la implementación

En esta sección se comentan aspectos singulares de la compilación del sistema, de la documentación de las librerías utilizadas y por ultimo las clases programadas que son consideradas especiales en este proyecto.

En lo que concierne a la compilación del proyecto, el sistema ROS fue introducido durante el desarrollo del proyecto, por lo que las bibliotecas MRPT tuvieron que ser incluidas en el sistema para poder usarlas en cualquier proyecto en ROS. El problema era que, en principio, MRPT utilizaba VOSCH y SVM como programas externos y esto hacía que las llamadas a los programas añadiesen alrededor de 100ms cada una al tiempo total de reconocimiento. Al tratarse de un clasificador en tiempo real se quiso eliminar este problema. Para ello se realizó un profundo estudio de CMake, ya que es el sistema de archivos que utiliza ROS para compilar sus bibliotecas. Existieron muchas duplicidades en las bibliotecas de PCL que estaban definidas de la misma forma en ROS y MRPT pero con distinto nombre. El problema se resolvió renombrando bibliotecas y cambiando las dependencias entre ROS y PCL. Una vez realizado, se exportó el proyecto ROS a Eclipse.

Por último, ya que el desarrollo de aplicaciones con el sensor Kinect es una tecnología reciente, la información y documentación que se encuentra en Internet no es en muchos casos completa. Los comentarios en algoritmos o ejemplos no se encuentran ampliamente extendidos por lo que se debe realizar un estudio profundo de algoritmos, tipos de datos, estructuras, bibliotecas y dependencias que no están directamente relacionadas con el problema de reconocimiento de patrones.

Ahora se comentan las clases más especiales que se han considerado en el diagrama de clases para los métodos de AdaBoost (Diagram 4.1 en la Sección 4.3.1 del Anexo C) que son tres, la clase RansacHTWG, la clase Organizador y la clase AdaBoostObject.

La clase RansacHTWG

La clase RansacHTWG fue creada para la utilización del algoritmo RANSAC de MRPT. Está basada en el método *mrpt::math::RANSAC::execute*, que es el que calcula el modelo final. En un primer momento, se programó un algoritmo que calculaba el plano respecto de la matriz de la imagen, pero los tiempos obtenidos eran muy elevados. Por lo que se decidió utilizar otra estructura distinta, *CColouredPointsMap*. Esta estructura contiene una nube de puntos sin ordenación. Un aspecto importante en la utilización de la estructura es que se necesita información sobre la imagen en 2D ni de cualquier tipo de ordenación de los puntos para calcular el plano, por lo que podemos prescindir de ella. Además, la razón más importante es el tiempo de ejecución. En el caso de la matriz de la imagen se tienen que tratar un conjunto de 640x480 datos, mientras que con *CColouredPointsMap*, sólo se tratan los puntos contenidos en el rango. Esto es muy importante, ya que se eliminan para el cálculo entre 200.000 y 290.000 puntos que no tienen interés.

Los métodos más importantes de la clase son:

- *ransacHTWG3Dplane_fit*: crea el modelo del plano a partir de tres puntos aleatorios de la nube de puntos.
- *ransacHTWG3Dplane_distance*: comprueba los puntos de la nube para saber si pertenecen al plano o no, teniendo en cuenta un margen máximo, el atributo *distanceThreshold*, que es la distancia de un punto con el plano-modelo.
- *run_ransacHTWGD*: ejecuta el método RANSAC para calcular el modelo final del plano.
- *isNotFloor*: compara un punto con el plano resultado obtenido. El plano resultado se describe usando los coeficientes del plano.

Hay que añadir, que se puede ejecutar el algoritmo completo un número de iteraciones determinada para conseguir un resultado más exacto, aunque esto repercute en el tiempo final de cálculo del plano. En el ámbito del tiempo real hay que lograr un compromiso entre tiempo de ejecución y fidelidad del resultado. Además existen varios parámetros que pueden ser modificados para reducir el tiempo de ejecución, como son por ejemplo el número máximo de iteraciones a realizar por el algoritmo, el número mínimo de puntos pertenecientes al posible plano a calcular o la distancia mínima que indica si un punto pertenece al plano candidato. Con esto se evita que el tiempo de ejecución del algoritmo sea excesivo y que el resultado obtenido sea razonablemente bueno para nuestra aplicación. La elección de dichos parámetros se explican en la Sección 4.4.2 del Anexo C.

La clase Organizador

La clase Organizador fue creada para el posterior tratamiento de las nubes de puntos que conforman cada objeto y para posibilitar el reconocimiento en tiempo real de varios objetos en todos los clasificadores. Esta clase es principalmente un vector de vectores en el que cada entrada del vector se guarda los puntos de un objeto y el color detectado por el sensor para cada uno de esos puntos . De esta forma cada grupo de puntos identificados por el segmentador (con una etiqueta dada) pertenecientes a un mismo objeto son introducidos en la misma coordenada del vector. Para

AdaBoost, además de los valores de los puntos en el espacio, se almacenan los valores R, G, B del color y en el caso de SVM, un valor real correspondiente al color de los puntos. El cálculo de este valor real se muestra en Listing 4.1 de la Sección 4.1 del Anexo C, y su computación es necesaria ya que es la forma en la que los archivos pcd guardan la información de color de los puntos.

La clase contiene diferentes métodos, entre los más importantes se encuentran:

- *Size*: indica el número de objetos que contiene la instancia de Organizador.
- *IsInOrganizador*: comprueba si alguna entrada del vector corresponde con una etiqueta dada.
- *AddOrganizador*: añade una nueva entrada al vector debido a una nueva etiqueta.
- *AddPoint*: añade un punto de un objeto a la entrada correspondiente del vector.
- *Clear*: vacía el vector.

A cada entrada del vector, es decir a cada objeto, se le asigna además un color aleatorio (el mismo para todos los puntos) que se utiliza para diferenciar los objetos más tarde durante la visualización en pantalla.

Clase AdaBoostObject

La clase *AdaBoostObject* se utiliza en los métodos de clasificación de AdaBoost. Se construyó para facilitar el cálculo de las propiedades de las nubes de puntos de los objetos, para su posterior utilización en los distintos clasificadores.

Los atributos más importantes de la clase son la nube de puntos, de tipo *CColouredPointsMap*, que guarda cada uno de los puntos que conforman la nube, y la altura, la anchura, la profundidad, que se utilizan para el cálculo de las propiedades para los clasificadores débiles.

Los métodos más importantes que componen la clase son los siguientes:

- *AddPointCloud*: añade una nube de puntos de tipo *CColouredPointsMap* a la instancia. Se utiliza en el esquema de tiempo real, siendo la instancia de la clase Organizador la que proporciona dicha nube.
- *ReadObject*: lee la nube de puntos de un fichero pcd que se encuentra en una ruta determinada. Se utilizada en el módulo de entrenamiento y de test para trabajar con los objetos de la base de datos deseada.
- *CalculateValues*: calcula los valores de altura, anchura y profundidad de la nube de puntos que contiene la instancia.
- *PrintPoints*: imprime la nube de puntos que contiene la instancia de la clase.

Anexo B
Evolución temporal del proyecto

Este anexo muestra la información temporal de las diferentes partes del proyecto. Dichas partes están indicadas en la zona derecha de la Figura B.1, que representa el diagrama de Gantt, mientras que en la parte derecha se muestra la duración de las diferentes partes. Se puede destacar el tiempo invertido en el estudio de la documentación tanto de la transmisión y representación de la información proveniente del sensor Kinect como de los métodos de clasificación y extracción de características utilizados. Añadir que se necesitó un mes para la puesta a punto del sistema ROS y de su interacción con la biblioteca MRPT, además del consiguiente estudio de su documentación y forma de compilación.

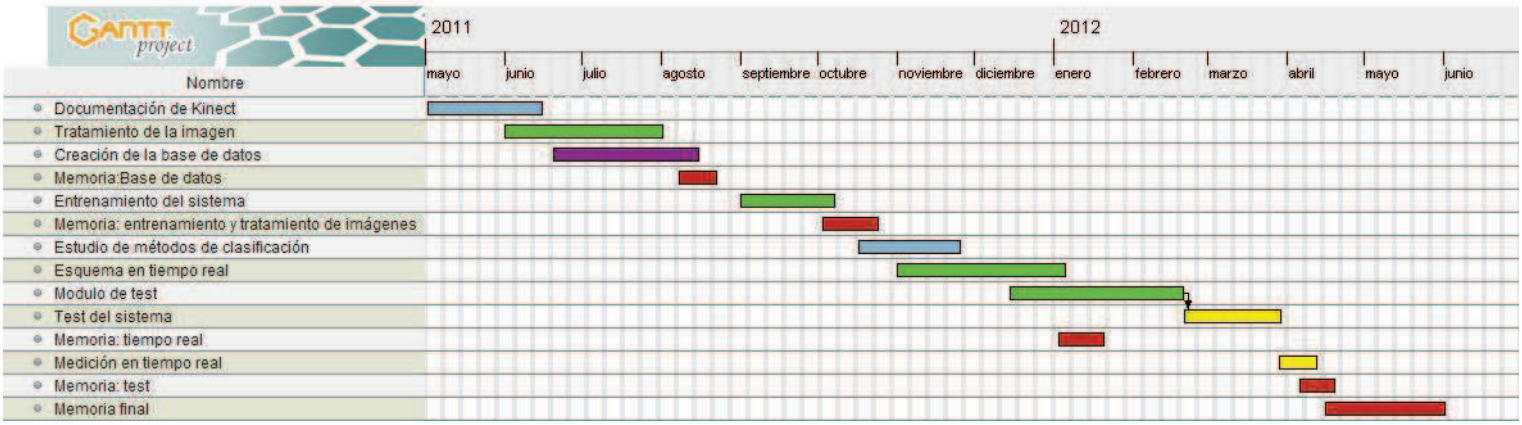

Figura B.1 Diagrama de Gantt que muestra la evolución del proyecto

Anexo C

Real-time 3D segmentation and Object Detection based on a Kinect Sensor

1 Introduction

The goal of robotics is to improve our way of living, helping us in science, discoveries, factory work, security or everyday tasks. Object detection is, for a robot, an essential step for the development of this kind of systems, in view of the fact that they require object handling to be able to carry out different complex tasks when it is interacting with its environment.

1.1 Motivation

Human beings have the important skill of recognizing objects, people or colors in a short period of time. This skill helps us to survive, learn new things, create tools or interact with other human beings and machines. The aim of the service robotics is the development of robots which are able to assist people, for example, in housework. For mobile service robots, the detection of everyday objects is an important challenge because it is the first step to develop advanced tasks, such us grab or move an object. The appearance of low-cost RGB-D sensors, like the Kinect sensor of Microsoft (Figure 1.1), has made possible obtaining synchronized depth and color information from the scene. This is helping researchers to improve detection systems in robotics due to various qualities that it possesses such as its small size.

Figure 1.1 Kinect sensor. It consists of a 3D depth sensor, a RGB camera and an array of microphones

1.2 Main Objective

The objective of this thesis is the development or use of object detection and classification methods in real-time programs as well as the study and comparison of the results between them. The objects are situated on a table to be analyzed and after that, classified. The different classes that can be recognized are described in a data-set which consists of different everyday objects. All the information coming from the objects is collected thanks to Kinect sensor. The used classification methods are supervised learning ones, these are: support vector machines (SVM) and adaptive boosting (AdaBoost). The aim of these methods is the use of complex or easy features. For SVM,

Voxelized Shape and Color Histograms (VOSCH) features are used. These features consist of a mixture of complex shape and colour characteristics. For AdaBoost, simple geometrical features are used. Because the program can classify different objects, the methods are used to make a multiclass classification. The results of this thesis consist of the first approach of integrating a vision system in a real mobile robot, making it able to grab, move or avoid an object.

1.3 Sub-Objectives

Apart from the study of the created programs, there are also two secondary objectives in this thesis. The first one is the development of a small data-set of everyday objects to be used as a source in this thesis. This data-set is fully explained in Section 3.1.1.1. The second one is the possibility of classifying more than one object in real-time at once. This function in the program has been developed because objects are not usually found alone on a table and if the program is executed by a mobile robot, this will interact with more than one object at once.

1.4 State of the art

In this part, some state of the art solutions in 3D pattern recognition connected with Kinect are going to be presented.
With the appearance of Kinect sensor, new data-sets have been created using this technology [1]. This data-sets use the color and depth information at the same time to create a sample of the object. They contain a huge amount of classes and instances in order to be able to use them to check how good a new classifier algorithm is. The state of the art of recognition algorithms can be mainly found in the Open Source Computer Vision library (OpenCV) [2]. This open source library contains a comprehensive set of both classic and state of the art computer vision and machine learning algorithms. These algorithms are widely used as in [3].
Pattern recognition research is been developed is very different fields, such as body recognition [4], face recognition [5], handwritten digit recognition [6], object and texture classification [7] or seismology [8]. This thesis is focused on object recognition and classification using OpenCV algorithms as a reference.

1.5 Structure

The structure of this document is mainly divided in five parts. In Chapter 2, the components and specifications of the Kinect sensor and the theoretical principles from machine learning are explained. In particular, theoretical basis of the classification and evaluation methods which are used in this thesis are presented. In Chapter 3, how the theoretical foundations of machine learning are used in each part of the recognizer is commented, in order to show a well-reasoned view of the structure of the classifier. It is also explored the structure of the data-set and how the data-set was created. The way of implementing these classification methods is explained in Chapter 4. It is shown the development of the different parts of the classifiers as well as the parts of each phase, the phases being: the learning phase and real-time classification phase. It is also discussed what kind of software (algorithms, libraries, compilers and programs) is used in order to program the classifiers because the algorithms are either implemented or used from different open source libraries. In Chapter 5, the test phase and the obtained results are studied in order to be able to get a conclusion from all the work developed.

2 Theoretical principles

The machine-learning principles that have been used or implemented in this project are going to be introduced in this chapter as well as the specifications of the sensor used in this thesis; the Kinect sensor. In this project two different methods are used, these are Support Vector Machines (SVM) and Adaptive Boosting (Adaboost). Both methods are supervised machine-learning methods, this means that a training set, in this case pictures of different objects, is used to train the algorithm in order to make a pattern recognition. A pattern is an abstract notion represented by a set of descriptions and it is usually described as a vector in which, its components are the attributes [9].
In machine learning there are mainly two parts in the development of a recognition program, the training phase and the test phase, which are presented in Section 2.2.1 and in Section 2.2.2.

2.1 Kinect Technical specifications

These are the Kinect technical specifications. It is shown the sensors that Kinect contains, the limitations of these sensors and the data that they receive:

- Sensor:
    - **Color VGA video camera:** This video camera aids in facial recognition and other features by detecting three color components: red, green and blue. Microsoft calls this an "RGB camera" referring to the color components it detects.
    - **Depth sensor**: An infrared projector and a monochrome CMOS (complimentary metal-oxide semiconductor) sensor work together to "see" the room in 3-D regardless of the lighting conditions.
    - **Multi-array microphone:** This is an array of four microphones that can isolate the voices of people from the background noise in a room.
- Field of view
    - Horizontal field of view: 57 degrees.
    - Vertical field of view: 43 degrees.
    - Physical tilt range ±27 degrees.
- Data streams
    - 320x240 16-bit depth @ 30 frames/sec.
    - 640x480 32-bit colour @ 30 frames/sec
    - 16-bit audio @16 kHz.

In Figure 2.1, a data transmission diagram of the Kinect sensor is shown:

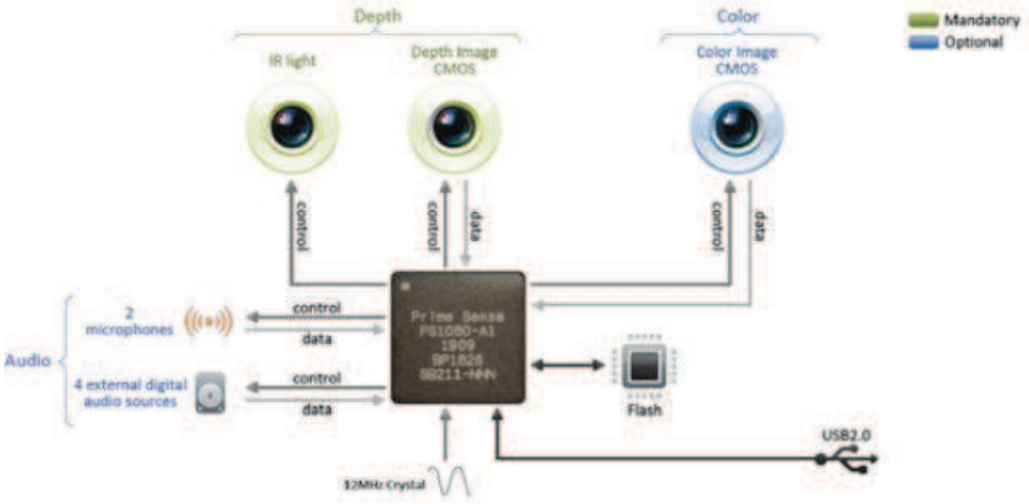

Figure 2.1 Data transmission in Kinect. It represents how the sensor receives the depth information (IR light and Depth Image CMOS), the color information (Color Image CMOS) and the sound (two microphones and our external digital audio sources) and how the sensor transmits information to other devices (USB 2.0), in this thesis, to a computer.

All the information that the sensor can obtain, allows it to distinguish different colours and distances, which make it possible to create different classifiers and recognizers depending on the task that it is wanted to be developed. Despite the fact that the precision of distance measurements is not always perfect (sometimes there can be found errors up to 1 cm in depth and 4 mm in height and width measurements), these differences are small enough to allow a robot to interact with its environment without significant problems using the data provided by the Kinect sensor as its vision system. The 3D information can be used by algorithms to enable a robot to recognize a face in order to situate it, respectively an object so it can grab it, or make a specific movement depending on the position of the recognized pattern. Because of all the advantages that the Kinect sensor provides, it is becoming one of the most popular sensors to be used in pattern recognition.

2.2 Machine learning working method

In this section, the two phases of the development and test of a machine learning application is presented. Each phase is divided in several parts. As it is a real-time program, the different parts of real-time applications is also commented.

2.2.1 Training phase

In this phase, the program is trained with the different instances of the objects to be able to recognize a different instance of a class afterwards. Training phase has mainly three parts: data collection, method choice and classifier training.

- Data collection: It has to be considered which is the field of classification, that is, what kind of classes it wants to be recognized and classified. The data-set will be different if a classification of objects, people or words is needed to be done. Datasets have a tree structure organization where each branch of the dataset is an easy to recognize class and the leaves of these branches are its different instances. There are several data-sets which can be found on the Internet depending on the field of application to evaluate the classifier. Some examples are Caltech 101 [10], LabelMe [11] or Imagenet [12]. Instead of using an already built dataset, an new dataset can be built to be able to work with a desired group of classes

or instances, maintaining always the tree structure. In the training phase not all the instances are used to train the classifier. A common method used to select what samples are going to be employed in the training phase are, for example, the holdout method, where a fixed subset of the dataset is used in the training phase and the rest of the samples are used in the test phase. Another method is crossing validation, where different subsets of the dataset are chosen to train the classifier, their complementary subsets to test it in different iterations. After this, the mean of all the results obtained from the iterations is computed.

- Method choice: this step consists of two parts- feature choice and principle choice. For the part it has to be considered what kind of features can represent a class properly and if both easy or complicated are required to be used. For example, if the classification between a glass and a bottle is needed to be done, a useful feature would be the height or between a green and red apple, colour would be a correct decision. After feature choice, the principle has to be chosen, this means, to choose the algorithm that is going to be used in the future classification.

- Classifier training: once previous steps are done, the method is applied using a training set from the dataset and the algorithm chosen in the previous step using the corresponding features. Information calculated by the corresponding algorithm or algorithms is saved to be able to distinguish among different classes and make a future classification.

2.2.2 Test phase

After training phase, the classifier has to be evaluated. This phase consists of three parts which are:

- Data selection: this part of the test phase depends on what instances have been chosen in data collection as well as the method of selection of the training and test sets. All the instances of the test set are used in this phase.

- Classification: once the test set is selected, the procedure is to extract the features of the instances and after this, use the algorithm that the selected method uses to classify them.

- Evaluation: after the classification of all the instances of the test set, the evaluation of the classifier is done. It can be evaluated, for example, using hit rates, confusion matrices or ROC curves. This way, the classifier can be compared with other ones or the precision of a classification of an instance is validated.

2.2.3 Real-time structure

When a recognition program is running, the structure is different to the training and test phases. The program is receiving information from the environment without a break and this data has to be processed in order to extract the features and make the classification. This structure, as shown in Diagram 2.1 can be summed up in:

- Data reading: to be able to receive information from the environment the program has to be connected to a sensor, for example thermographic, 2D or 3D cameras, which send the information to the recognizer. To use this data, algorithms are needed that can read the information and send it to the program. These algorithms are usually integrated libraries which make possible the information transmission between sensor and program. Sometimes different libraries or algorithms can be found to get the same information from a single

sensor.

- Pre-processing: once the complete sensor data can be transferred to the program, this information has to be selected owing to the fact that not all the data from the sensor will be used; just the important information from the sensor library to the recognizer has to be processed. After selecting the information from the sensor a new data processing may be needed because raw data can contain irrelevant parts for the program in the following steps. For example, in object recognition field, working with a sensor where just 2D images coming from the sensor are selected, a segmentation of the image would be done to get only the objects in the image and discard the background. After that, the attribute extraction of the objects can be done in order to make the pattern recognition just with feature information, avoiding using the whole information from the object. Although the feature extraction is considered a part of the pre-processing, sometimes it is not clear the difference between this stage and the application of the method (for example, neural networks) [13].

- Classification: with the relevant information from the sensor, the classification algorithms from the selected method can now be used to classify the data.

- Post-processing: after classification is done, not only the label of the recognized class can be required but also the name and further information of the class, such as the distance to the object, the specific position on a picture or in the space or if the recognized data is moving or not.

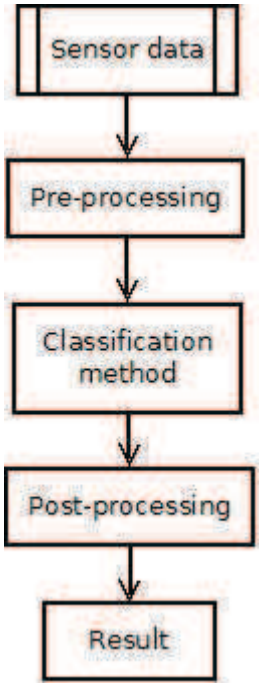

Diagram 2.1 The diagram shows the outline of a real-time recognition. The raw data coming from the sensor is processed so that the classifier can recognize the sample. The post-precessing can be done in order to clarify the information before the final result is given.

2.3 Classification methods

In this section, two classification methods are going to be introduced, Support Vector Machines and

Adaptive Boosting. Although Support Vector Machines can solve regression problems, just classification problems are going to be presented.

2.3.1 Support Vector Machines (SVM)

Support Vector Machines (SVM) is a concept developed from the Statistical Learning Theory (Vapnik & Chervonenkis) [14]. SVM are learning systems that use a hypothesis space of linear functions in a high dimensional feature space, trained with a learning algorithm from optimization theory that implements a learning bias derived from the Statistical Learning Theory [15]. This method creates a hyperplane or set of hyperplanes which separates regions with different class memberships in order to be used in different tasks such as classification or regression. In machine learning the goal of SVM is to be able to classify a new given point in the correct region. Points are viewed as p-dimensional vectors and, if the different classes can be separated with a $(p-1)$-dimensional hyperplane, a linear classifier will be found. In Figure 2.2, a binary linear classification example is shown, a line separates orange and blue objects:

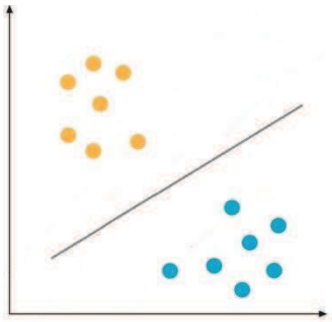

Figure 2.2 A hyperplane divides the region in two parts. New samples under the hyperplanes will be classified as "blue" and the rest of them will be classified as "orange" [16].

But a classifier can use more than one hyperplane to distinguish classes between two regions. In Figure 2.3 three of these possible hyperplanes are shown. Since the best classifier is aimed to be found, the hyperplane has to represent the largest separation, or margin, between the two regions. That is why, to find the best classifier, a maximization of the distance from the hyperplane to the nearest points on each side has to be done. This hyperplane is called maximum-margin hyperplane and the classifier it describes, maximum-margin classifier.

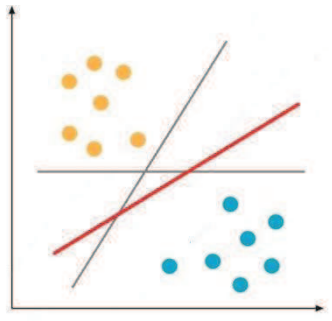

Figure 2.3 Three possible hyperplanes for the classification problem are shown in the figure. The hyperplane in red represents the perfect one due to the fact that the separation between the closest sample of both classes is the largest [16].

2.3.1.1 Mathematical principles

The mathematical principles to calculate a binary linear classifier are going to be presented.

Given a training set:

$$\mathcal{D} = \{(\boldsymbol{x_i}, \boldsymbol{y_i}) \mid \boldsymbol{x_i} \in \mathbb{R}^{\boldsymbol{p}},\ \boldsymbol{y_i} \in \{-\mathbf{1}, \mathbf{1}\}\}_{\boldsymbol{i=1}}^{\boldsymbol{n}} \quad (2.1)$$

Where $\boldsymbol{x}_i$ is a p-dimensional real vector which describes a point of the training set and y_i is the desired output value, which is 1 or -1, used to describe which class the point belongs to. A hyperplane can be described as:

$$\langle \boldsymbol{w} \cdot \boldsymbol{x} \rangle - \boldsymbol{b} = \mathbf{0} \quad (2.2)$$

where $\boldsymbol{x}$ is data from a object and $\boldsymbol{w}$ is the normal vector to the hyperplane. b describes the offset of the hyperplane from the origin along the normal vector $\boldsymbol{w}$. Figure 2.4 also describes the optimal hyperplane which maximizes margin ρ, that is the distance between $\langle \boldsymbol{w} \cdot \boldsymbol{x} \rangle - b = 1$ and $\langle \boldsymbol{w} \cdot \boldsymbol{x} \rangle - b = -1$. This margin can be calculated as, $\rho = \frac{2}{\|\boldsymbol{w}\|}$ .The problem of maximization of the margin can be presented now as a minimization problem of $\boldsymbol{w}$. Geometrically, Support Vectors are the training patterns that are closest to the decision boundary.

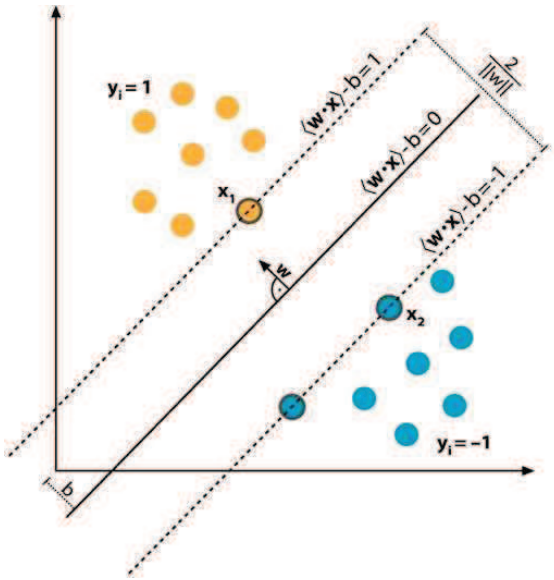

Figure 2.4 Example of hyperplane which divides the region in two parts with the largest margin ($\langle \boldsymbol{w} \cdot \boldsymbol{x} \rangle - \boldsymbol{b} = \mathbf{0}$) and the two hyperplanes which contain the support vector machines ($\langle \boldsymbol{w} \cdot \boldsymbol{x} \rangle - \boldsymbol{b} = \mathbf{1}$) and $\langle \boldsymbol{w} \cdot \boldsymbol{x} \rangle - \boldsymbol{b} = -\mathbf{1}$)). The blue samples are labeled as -1 and the orange ones as 1 [16].

To make a correct classification between the classes, these constraints have to be added,

$$\begin{aligned} \langle \boldsymbol{w} \cdot \boldsymbol{x_i} \rangle - b &\geq \ \ 1 \quad \text{if} \ \ y_i = \ \ 1 \\ \langle \boldsymbol{w} \cdot \boldsymbol{x_i} \rangle - b &\leq -1 \quad \text{if} \ \ y_i = -1 \end{aligned} \quad (2.3)$$

Or in a compact form,

$$y_i(\langle \boldsymbol{w} \cdot \boldsymbol{x_i} \rangle - b) \geq \ \ 1 \quad (2.4)$$

The support vector machines are the $\boldsymbol{x}_i$ values which satisfy the equation,

$$y^{(s)}\left(\langle \boldsymbol{w}\cdot \boldsymbol{x}^{(s)}\rangle - b\right) = 1 \qquad (2.5)$$

And given a new instance $\boldsymbol{x}$, the classifier is

$$f(x) = sign\ (\langle \boldsymbol{w}\cdot \boldsymbol{x}\rangle - b) \qquad (2.6)$$

In this optimization problem of $\|w\|$, the expression $\frac{1}{2}\|w\|^2$ can be used instead for mathematical convenience without changing the solution. To solve this problem the Lagrangian multipliers are used. In this problem w and b need to be minimized and α maximized. The corresponding Lagrangian function to the quadratic optimization problem is:

$$L(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \frac{\|\boldsymbol{w}\|^2}{2} - \sum_{i=1}^{n} \alpha_i [y_i(\boldsymbol{w}\cdot \boldsymbol{x}_i - b) - 1] \qquad (2.7)$$

where α_i are the Lagrangian multipliers, $\frac{1}{2}\|\boldsymbol{w}\|^2$ is the objective function, and $y_i(\boldsymbol{w}\cdot \boldsymbol{x}_i - b) \geq 1$ the constraint. This is called primal formulation of linear SVMs. It is a convex quadratic optimization problem with n variables, where n is the number of features. The solution can be expressed by terms of linear combination of the training vectors.

$$\boldsymbol{w} = \sum_{i=1}^{n} \alpha_i y_i\, \boldsymbol{x}_i \qquad (2.8)$$

Substituting equation (2.8) in function (2.7) the dual formulation of linear SVMs is obtained:

$$L(\boldsymbol{\alpha}) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i \boldsymbol{x}_j \qquad (2.9)$$

where the function tries to be maximized with respect to $\boldsymbol{\alpha}$, subject to the constraints $\alpha_i \geq 0$ and $\sum_{i=1}^{N} \alpha_i y_i = 0$. Solving this problem scales with the number of samples, N. $\boldsymbol{w}$ can be computed thanks to the $\boldsymbol{\alpha}$ terms. The final solution has the form:

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} \alpha_i y_i \boldsymbol{x}_i \boldsymbol{x} + b \qquad (2.10)$$

The classification depends on the sign of $f(\boldsymbol{x})$ for the input data $\boldsymbol{x}$.

2.3.1.2 Kernel method

One common pre-processing strategy in machine learning involves changing the representation of the data [15]. When the problem is not linearly separable, the kernel method can be used. The idea of this method is to transform the input space of the non-linearly separable data into a feature space so that data can be linearly classified using a nonlinear function Φ.

$$\begin{aligned} \Phi :\ & \mathbb{R}^n \rightarrow \mathbb{R}^m \\ & \boldsymbol{x} \mapsto \Phi(\boldsymbol{x}) \end{aligned} \qquad (2.11)$$

This step is equivalent to mapping the input space X into a new space, $F = \{\Phi(\boldsymbol{x}) | \boldsymbol{x} \in X\}$. Now the training set is

$$\mathcal{D} = \{(\Phi(\boldsymbol{x}_i), y_i)\}_{i=1}^{n} \quad (2.12)$$

And the hyperplane,

$$\boldsymbol{w}\,\Phi(\boldsymbol{x}_i) + b = 0 \quad (2.13)$$

Thus the hyperplane equation is,

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} \alpha_i y_i k(\boldsymbol{x}_i, \boldsymbol{x}) + b \quad (2.14)$$

where $k(x_i, x) = \Phi(x_i)\Phi(x)$. A visual example of this transformation is shown Figure 2.5, using the kernel,

$$\Phi(\mathbf{x}) = \begin{pmatrix} x_1^2 \\ \sqrt{2}\, x_1 x_2 \\ x_2^2 \end{pmatrix} \quad (2.15)$$

where, after the kernel transformation, the problem is linearly separable.

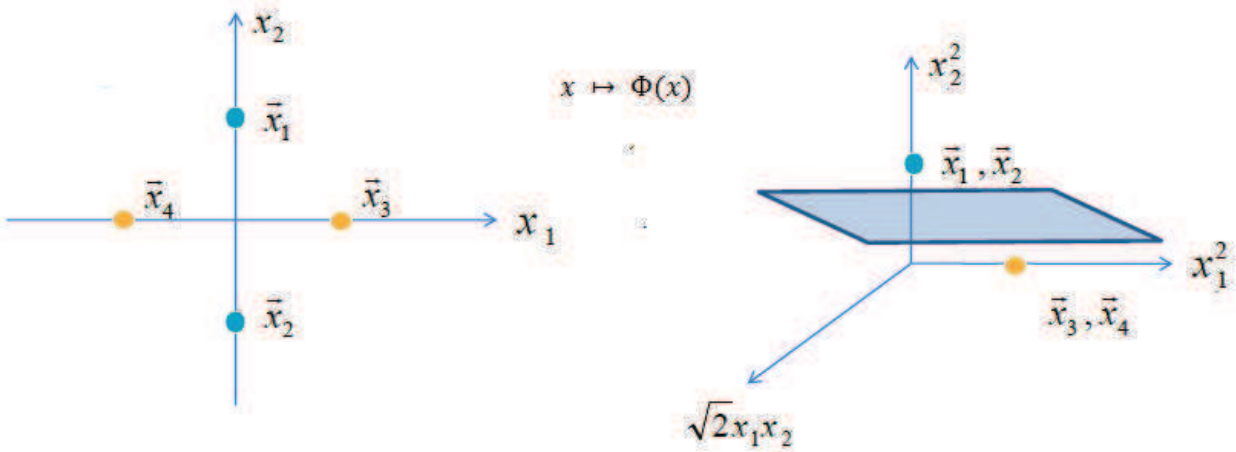

Figure 2.5 Visual example of kernel transformation. The non-linearly separable problem on the left is transformed in a linearly separable one using the kernel $\Phi(x) = \left(x_1^2, \sqrt{2}x_1x_2, x_2^2\right)^T$. This way, a hyperplane can be found in order to divide the space in two regions.

These are the most common used kernels:

- Polynomial, $k(\boldsymbol{x}_i, \boldsymbol{x}) = (\boldsymbol{x}\boldsymbol{x}_i + 1)^p$, where p is a specified parameter.
- Radial basis function, $k(\boldsymbol{x}_i, \boldsymbol{x}) = e^{-\frac{1}{2\sigma^2}\|x - x_i\|^2}$, where σ^2 is a specified parameter.
- Hyperbolic tangent, $k(\boldsymbol{x}_i, \boldsymbol{x}) = \tanh(\beta_0 \boldsymbol{x}\boldsymbol{x}_i + \beta_1)$, where β_0 and β_1 are parameters.

2.3.2 Adaptive Boosting (AdaBoost)

Boosting is a meta-algorithm used in supervised machine learning. It combines many rough rules of

thumb, which are called weak learners, to build a robust prediction rule. The typical example for boosting is presented in [17], where a horse-racing gambler, hoping to maximize his winnings, tries to create a method to predict the winner of a horse race. It is difficult for him to create and explain a successful betting strategy, but having the data from some races, the gambler can come up with a "rule of thumb" for that data, such as, bet on the horse that has recently won most of the races or bet on the horse with the most favored odds. These rules are inaccurate when used separately. The idea of boosting is to create a method by combining rough and moderate rules of the thumb that the gambler knows in order to make the best prediction.

AdaBoost, introduced in 1995 by Freund and Shapire [17], is one of the most used boosting algorithms owing to its good results. The pseudocode for AdaBoost is presented in Listing 2.1. The input training set is $(x_1, y_1), \dots, (x_n, y_n)$ where x_i belongs to the instance space X, and y_i is the value of the desired output belonging to $Y = \{-1, +1\}$. A weak classifier f, checks the value of a coordinate of a sample $\boldsymbol{x} \coloneqq (x_1, x_2, \dots, x_n) \in M$ depending on a threshold value l to classify the sample as positive, +1, or negative, -1, M being the feature space, as shown in Equation (2.16).

$$f(\boldsymbol{x}) = f(x_1, x_2, \dots, x_n) \coloneqq \begin{cases} +1 \; if \; x_j \geq l \\ -1 \; if \; x_j \leq l \end{cases} \qquad (2.16)$$

AdaBoost calls a given weak learning algorithm repeatedly in a series of rounds $t = 1, \dots, T$. One of the main ideas of the algorithm is to maintain a distribution or set of weights over the training set. The weight of this distribution of the training example i on round t is denoted $D_t(i)$. At the beginning of the algorithm, all weights have the same value, but at each round, the weights of incorrectly classified examples are increased and the ones for correctly classified examples are decreased so that the new weak classifier is forced to focus on the hard examples in the training set. Each round, once the weak hypothesis or rules of the thumb h_t are selected according with the distribution D_t, the algorithm chooses a parameter α_t, which measures the importance assigned to h_t.

Given: $(x_1, y_1), \dots, (x_m, y_m)$ where $x_i \in X, y_i \in Y = \{-1, +1\}$
Initialize $D_1(i) = \frac{i}{m}$ where m is the number of samples.
For $t = 1, \dots, T$

- Train weak learner using distribution D_t
- Get weak hypothesis $h_t: X \to \{-1, +1\}$ with error

$$\epsilon_t = Pr_{i \sim D_t}[h_t(x_i) \neq y_i]$$

- Choose $\alpha_t = \frac{1}{2} ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
- Update:

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha t} & \text{if } h_t(x_i) = y_i \\ e^{\alpha t} & \text{if } h_t(x_i) \neq y_i \end{cases}$$

$$= \frac{D_t(i) \exp\left(-\alpha_t y_t h_t(x_i)\right)}{Z_t}$$

Listing 2.1 Pseudocode of the AdaBoost algorithm for binary classification

Where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis:

$$H(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x)) \quad (2.17)$$

Thus, the aim of the algorithm is to build the classifier $H(x)$ using the weak hypothesis h_t.

2.4 Evaluation methods

In this point, some of the most common evaluation methods in machine learning are going to be presented. First confusion matrices are going to be explained and after that, different rates that can be calculated in pattern recognition using confusion matrices are going to be shown. Finally the ideas of Receiver Operating Characteristic (ROC) curves will be presented.

2.4.1 Confusion matrices

A confusion matrix [18] contains information about actual and predicted classifications done by a classification system. Performance of such systems is commonly evaluated using the data in the matrix. A binary classification can be described as in the Figure 2.6 Confusion matrix structure:

		Prediction outcome	
		Positive	Negative
Actual value	Positive	TP	FN
	Negative	FP	TN

Figure 2.6 Confusion matrix structure

Where:

- *True positive (TP)* is the correct prediction of a positive sample, also known as hit.
- *False negative (FN)* is the incorrect prediction of a positive sample, also known as Type II error.
- *False positive (FP)* is the incorrect prediction of a negative sample, also known as false alarm or Type I error.
- *True negative (TN)* is the correct prediction of a negative sample, also known as correct rejection.

The entries of the matrix contain the number of TPs, FPs, FNs and TNs and with these values further information from the confusion matrix can be calculated. The most important terms for this matrix are:

- The *true positive rate* (*TPR*), *sensitivity* or *recall* is the proportion of positive cases that were correctly identified, calculated using the equation:

$$TPR = \frac{TP}{P} = \frac{TP}{(TP + FN)} \quad (2.18)$$

- The *false positive rate* (*FPR*) or *fall-out* is the proportion of negative cases that were incorrectly classified as positive, calculated using the equation:

$$FPR = \frac{FP}{N} = \frac{FP}{(FP + TN)} \quad (2.19)$$

- The *true negative rate* (*TNR*) or *specificity* is defined as the proportion of negative cases that were classified correctly, calculated using the equation:

$$TNR = \frac{TN}{N} = \frac{TN}{(FP + TN)} = 1 - FPR \quad (2.20)$$

- The *false negative rate* (*FNR*) is the proportion of positive cases that were incorrectly classified as negative, calculated using the equation:

$$FNR = \frac{FN}{P} = \frac{FN}{(TP + FN)} \quad (2.21)$$

- The *accuracy* (*AC*) is the proportion of the total number of predictions that were correct. It is determinated using:

$$AC = \frac{(TP + TN)}{(T + N)} \quad (2.22)$$

- The *positive predictive value (PPV)* or *precision* is the proportion of the predicted positive cases that were correct, calculated using the equation:

$$PPV = \frac{TP}{(TP + FP)} \quad (2.23)$$

Other terms for this matrix are:

- *Negative predictive value (NPV):*

$$NPV = \frac{TN}{(TN + FN)} \quad (2.24)$$

- *False discovery rate (FDR):*

$$FDR = \frac{FP}{(FP + TP)} \quad (2.25)$$

2.4.2 Receiver Operating Characteristic (ROC)

A Receiver Operating Characteristic (ROC) is the graphical plot of the true positive rate (TPR) in function of the false positive rate (FPR) in the ROC space. ROC space, Figure 2.7 is defined by two axes, FPR as x-axis and TPR as y-axis. Each point which is drawn in this space represents a

prediction result of a class, that is, an instance of a confusion matrix which contains the information of sensitivity and specificity.

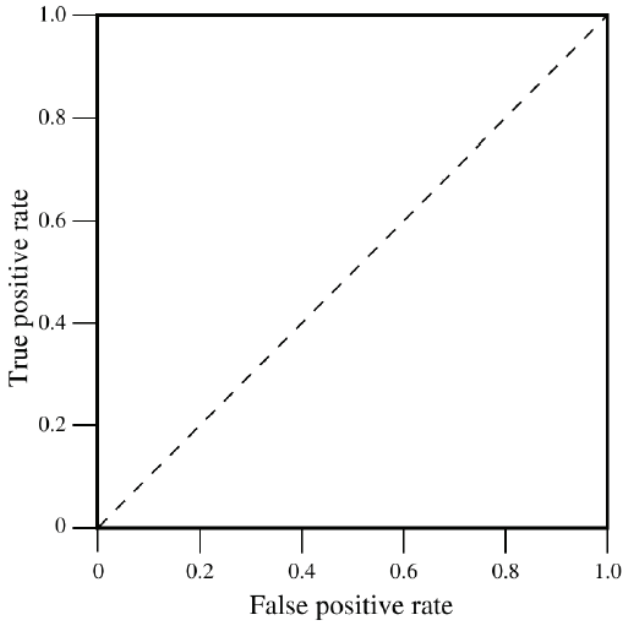

Figure 2.7 ROC space. The X-axis is represented by the FPR and the y-axis by the TPR. The dotted line represents the random classification [19].

When different thresholds are used to distinguish if a sample belongs to a class or not, different values of sensitivity and specificity can be calculated for a sample set to create the corresponding points in ROC space which will create a ROC curve. If a classifier with a threshold which classifies all the samples as positive is chosen, the point (1, 1) in ROC space will be obtained, and in case all the samples are classified as negative the point (0, 0) will be calculated. The perfect classification point is (0, 1) where sensitivity and specificity are 1 (no false negatives or false positives). The closest a classifier point is to the perfect classification point, the better the classifier. ROC space is divided by the diagonal ((0, 0), (1, 1)). A point on the diagonal means a random classification of the samples, points below the diagonal represent worse classifications than the random classification and points above the diagonal represent better classification results. In Figure 2.8, different classifiers in ROC space are presented:

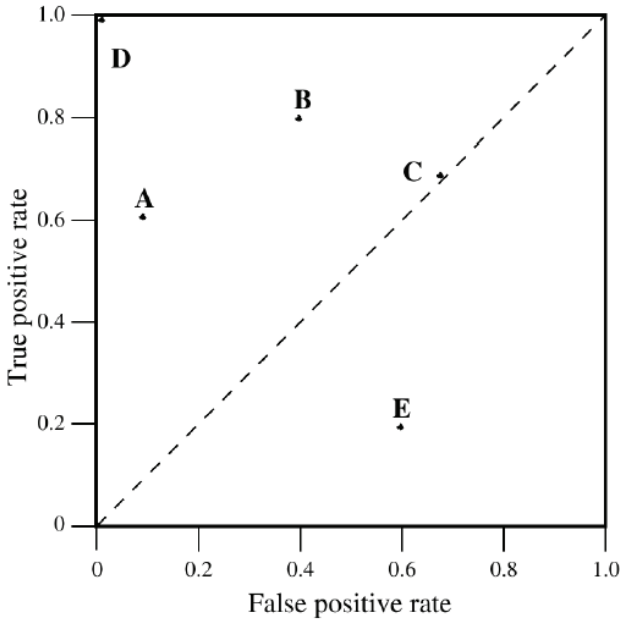

Figure 2.8 Example of classifers, named form A to E in ROC space. D classifiers represents the perfect classifier, C, the random classifier and E a bad classifier [19].

Point D corresponds to a perfect classifier, and point C, as it is situated on the diagonal corresponds to a random classifier. Point E is a classifier worse than random guessing, but building a symmetric classifier E' from the diagonal, the method can show the predictive power the classifier has, that is, where the method E predicts positive or negative, the method E' will predict negative or positive respectively.

If the method is defined by a parameter, the classifier produces a continuous output depending on the value of the parameter, as shown in Figure 2.9. The typical application fields of ROC analysis are psychology, medicine, radiology and biometrics and it is increasingly used in machine learning and data mining.

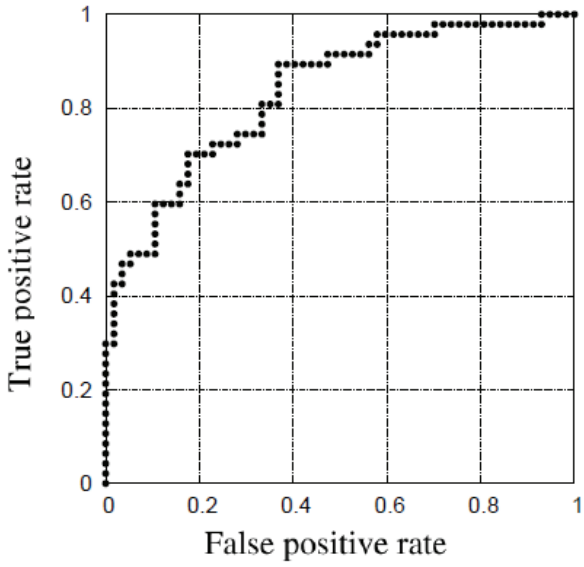

Figure 2.9 Roc curve for a parametric method [20].

3 Classification of everyday objects

As presented in Section 1.2, the objective is to detect objects on a table. The type of objects which it is going to work with is everyday objects. These objects are fruit or containers. Some kinds of objects, like a juice carton or a marker pen, were discarded because of their size. The classes of objects used in this thesis are apples, bananas, bowls, plastic cups, coffee mugs and tobacco packets. The sizes and shapes of the classes are sometimes similar, that is why it can be more difficult to distinguish one class from another and the decision boundaries have to be precise.
In this section it is going to be explained how the training and classifier programs are developed. The explanation will be done using Diagram 3.1 and Diagram 3.2, learning and real-time classifier diagrams respectively.

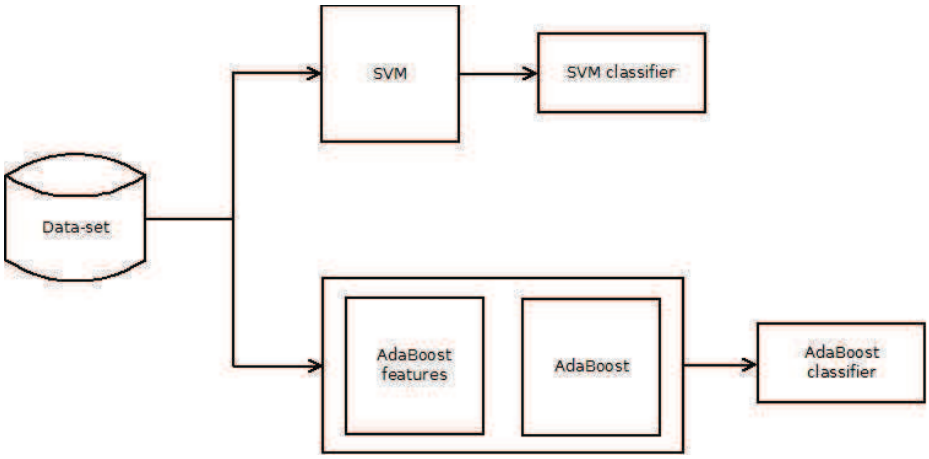

Diagram 3.1 Training structure. The data-set is the source of samples which is used to build the classifiers. After reading the data-set, each method creates the recognizer in order to be used in future classifications.

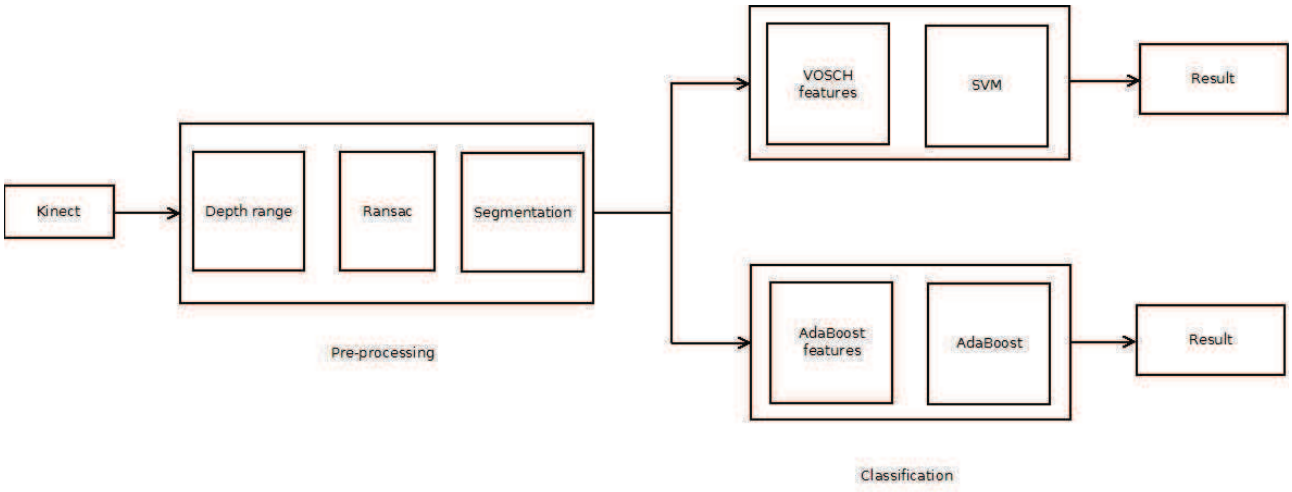

Diagram 3.2 Real-time structure. Kinect sensor receives the information from the environment. The raw data is processed so that the feature extraction and the classification can be done just with the desired objects.

Step by step, all the parts of each diagram will be commented on, following the outline from Diagram 2.1. A differentiation between the learning and classifier program is done to show the different parts and sources of each program.

3.1 Data collection

Data collection is the first part of a recognizer program because it is the source from which the program will read the information. The real-time program reads the information from the Kinect sensor and the training and test program read it from a data-set. The following section is focused on the explanation of the information source in the training phase of a recognition problem.

3.1.1 Training phase

Depending on the kind of program, data is read from one source or another. In the learning phase of this thesis data is read from its own database. In the following section, how the dataset is organized, and how exactly the information was taken, is thoroughly explained.

3.1.1.1 Data-set

The Data-set is a database specially developed for this project. It consists of different kind of images and files from different objects, with which the system can be trained and tested.
These images were taken using the Kinect sensor which was situated 38 cm high with a tilt angle of -15º (Figure 3.1). These parameters were chosen because they fit perfectly supposing the sensor that can be adapted in a middle-sized mobile robot.

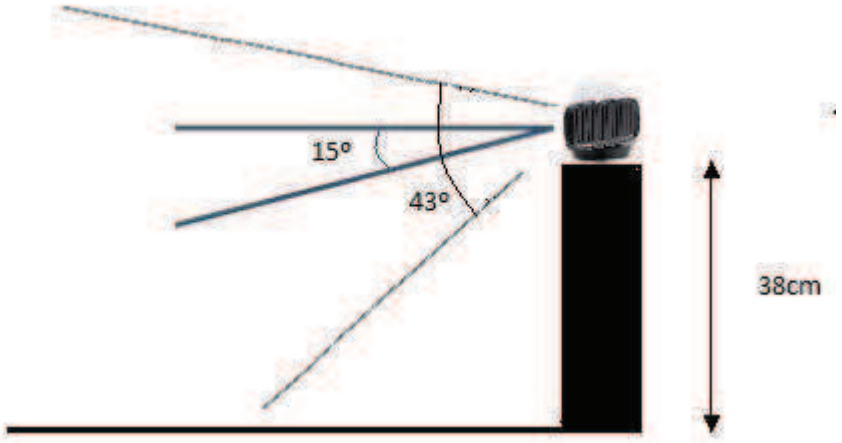

Figure 3.1 Position of the Kinect sensor with a tilt angle of -15º and a range vision of 43º

The Data-set contains four different folders, which are “Depth”,” Images”, “PointClouds” and “Features”. Each file inside a folder describes different information of an objects´ snapshot. Depth folder contains images, of 640x480 size, that show depth information in greyscale with .jpg extension. Color images with 640x480 size and extension .jpg can be found in the “images” folder. “Pointclouds” contains pcd files which describe the point cloud, and “features” contains the pcd files with the Voxelized Shape and Color Histograms (VOSCH) features for SVM of every snapshot.
The Data-set contains six different classes: apple, banana, bowl, mug, plastic cup and tobacco. The first two classes correspond to fruits, the following three classes to containers and the last one to a cigarette packet. Each folder of the database contains one folder for each class, which carries the name of the representing class. Inside each, different instances exist, the number of instances being

different for each class. Figure 3.2 shows all the instances of the data-set. For every object of a class there are eight snapshots, from which, depth and color information, the corresponding point cloud and the VOSCH features are extracted.
The names of the color images are built combining the name of the class and a number. The name of the depth images is built adding a ‘d’, from depth, before the color image name. For point cloud and feature files a ‘p’, from point cloud, and, respectively, an ‘f’, from features, are added before the name. Point cloud files save just the points of the object and none from the environment.

Figure 3.2 Pictures of the 30 instances of the six classes from the data-set ordered by class.

The numbering assignment to an instance of a class is the same for the four existing folders, each number belongs to a certain snapshot and starts from one. For each object a range of eight numbers is assigned, one for each snapshot. The structure of the data-set is shown in Diagram 3.3. As an example, apple6.jpg shows the sixth color image from the first apple of the apple class, and fapple9.pcd has the VOSCH features of the first image from the second apple of its class.

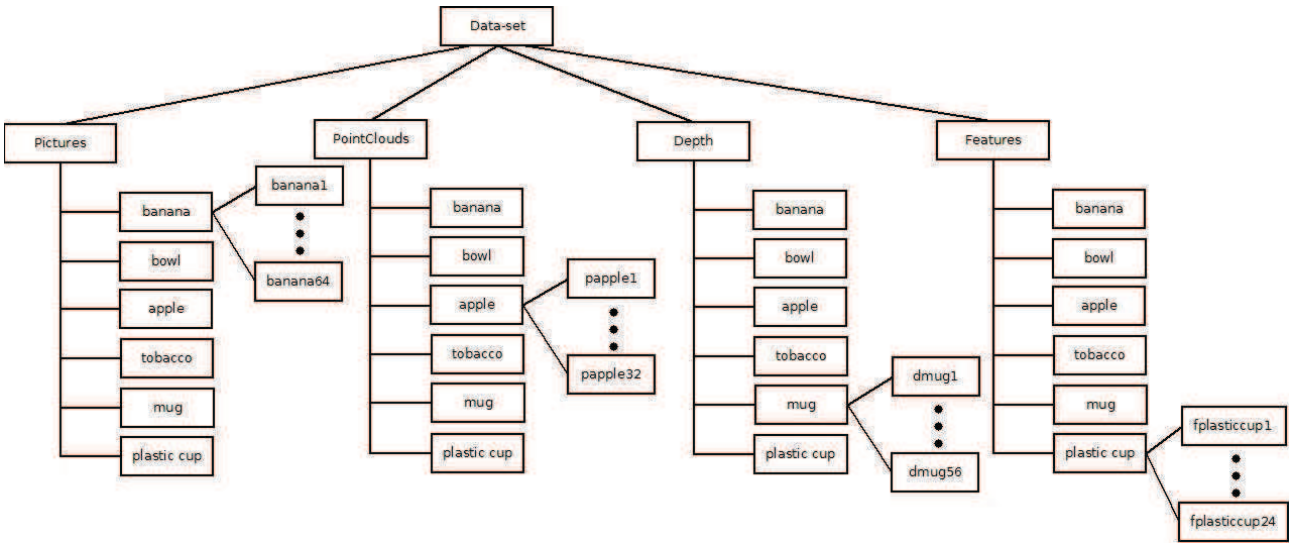

Diagram 3.3 Structure of the data-set. The main folder "Data-set" contains "Pictures", "PointClouds", "Depth" and "Features". Each one contains a folder for each class with all the instances inside.

Each picture of an object can be taken in a different position and from a different orientation. These positions and orientations are described using a diagram and a picture of the actual object is shown too. Each diagram consists of:

- Small images that symbolized the class each object belongs to.

- A small image of the Kinect sensor that represents the position and orientation of the sensor 38 cm high, this is, looking to the left with a tilt angle of -15º.

- A number next to each object image, which represents the number assigned to a picture of that class with that position and orientation. If the image has a range of numbers separated by a dash, it means that the position of different pictures is the same but the orientation changes.

- Thin lines which mean distances with regard to the sensor situation. The 2 upper lines on indicate a distance of 5 and 10cm from the middle of the sensor to the right, and the 2 lower lines, 5 and 10cm to the left. The 3 vertical lines on the lower part indicate a distance of 65, 76 and 87cm from the sensor respectively.

Two examples, one of an apple and one of a bowl, are presented in Diagram 3.4 and Diagram 3.5.

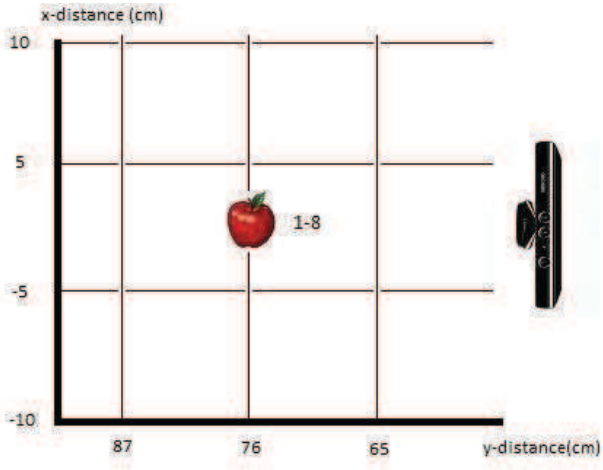

Diagram 3.4 Example of the pictures taken for the first apple of the data-set. The apple is situated at 76 cm from the sensor in front of it. The position of the apple is the same for the eight snapshots but the orientation changes 45º clockwise from the first picture. The range of pictures goes from one to eight.

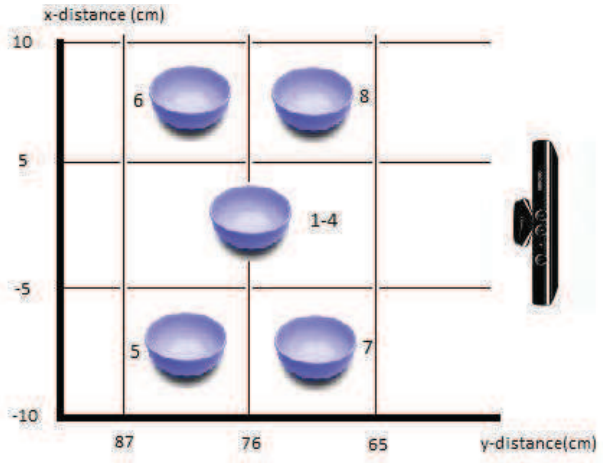

Diagram 3.5 Example of the pictures taken for the first bowl of the data-set. The snapshots from one to four are situated in front of the sensor at a distance of 76cm. the position for the first four pictures is the same but the orientation changes 90º clockwise. The position of the pictures from five to eight is the one shown on the diagram. For example, picture 6 is situated at a distance of 81,5cm from the sensor and 7,5cm to the right. The orientation of the last four snapshots is the same.

3.1.1.2 Holdout method for cross validation

To select which instances belong to the training set and which ones to the test set, holdout method is used. The holdout method is the simplest kind of cross validation. The data set is separated into two sets, called the training set and the testing set. The classifier parameters are calculated using the training set only. Then, in the test phase, the method predicts the output values for the data in the testing set and the error rates are calculated using these results. It is necessary to remind that eight pictures belong to each instance of an object, so if two instances of a class are learnt by the classifier, it will be actually learning 16 pictures. This way, all the pictures of an instance will be either in training set or in test set, but there will never be pictures of the same object in both sets. The advantage of this method is that it is fast to compute although the evaluation may be significantly different depending on how the division is made.

3.2 Real-time program

In this section the pre-processing of data made in the real-time programs is going to be presented. In this phase raw data is processed in order to get the information which the program can work with.

3.2.1 Pre-processing

In the learning phase, the information of the objects is already processed in the dataset, that is, the features which will be used in SVM are already calculated, and the point cloud of the object is contained in pcd files. This makes easier the learning program and saves time during the learning process. In real-time data collection, the sensor receives raw data from the environment, this data has to be processed in real time in order to be able to calculate the features just for the valid objects. This data pre-processing is calculated in three parts: elimination of points based on depth range, on RANSAC algorithm and segmentation of the corresponding image.

3.2.1.1 Depth range

Taking into account the position where the sensor is situated, the possible integration in a mobile robot in the future and that it is a real-time application, the program should get rid of useless information as soon as possible to reduce the amount of data, therefore the number of calculations. The solution to this problem is to think that points out of a fixed depth range are invalid points. The valid points in this application are the points situated in a distance between 0,5 m and 1 m, the rest of them, the points closer than 0,5m and further than 1 m, are considered as invalid points, as shown in Figure 3.3 . In addition, points at a distance of 3,5m or more are already considered invalid by the sensor due to sensor specification.

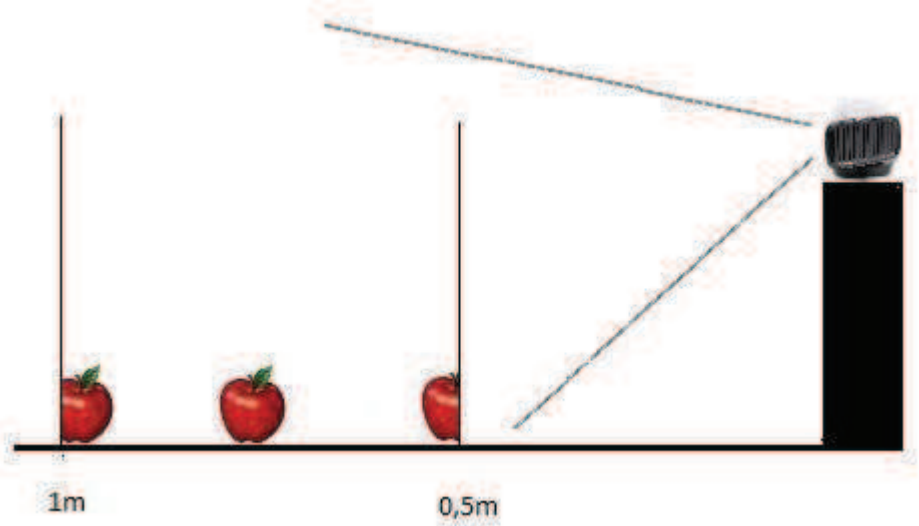

Figure 3.3 Example of the depth restriction of the classifiers. Three apples are trying to be recognize but just the parts of the apples which are between 0,5m and 1m are detected by the sensor

3.2.1.2 Random Sample Consensus algorithm (RANSAC)

Random Sample Consensus (RANSAC) [21] is an algorithm which can find mathematical models using an iterative method. Points which belong to the model are called inliers and if they don´t, they are called outliers. RANSAC uses a simple principle to calculate the model. In each iteration, a

small set of points is randomly chosen which will be part of a possible model. Then the remaining points of the set will be checked to know if they can be fit to the current model. The estimation of remaining iterations in order to get a good set of points which fits the model, changes in each iteration. This estimation means the number of trials needed to get a model with no outliers in it, with probability *p*.
Because in this part of the program, the set of points includes the corresponding points of the table onto which the objects are put, the RANSAC algorithm is used to get rid of these specific points. In this case, the mathematical model is a plane representing the table. In the Table 3.1, nine observations of objects were taken. It represents the number of points eliminated because of the depth range and the RANSAC algorithm as well as the final points which belong to the object. The mean of the values of the different objects is calculated to be able to show an estimation of the percentage of each step from the whole amount of points, 307200:

	Range	RANSAC	Object
	244176	58342	4682
	245745	56387	5068
	244893	57993	4314
	243776	60885	2539
	245254	60119	1827
	248449	56888	1863
	243149	59078	4973
	240699	62214	4287
	244095	58991	4114
Mean of points	244471	58989	3741
Percentage	79,580%	19,202%	1,218%

Table 3.1 Sample of the number of the eliminated points due to range and RANSAC and the final object points in nine classifications

Looking at the Table 3.1, it can be seen the need to get rid of those points as soon as possible in order to avoid computing the information with non-important points.

3.2.1.3 Segmentation

Segmentation is the step in which an image is divided in multiple segments to distinguish between relevant parts and the background. Different pixels of an image are labeled if they are considered they belong to a foreground region. Pixels from the same region will have the same label to distinguish among different regions. The aim of segmentation is the differentiation between foreground and background and the differentiation among objects in the same image. Different object have different labels in order to distinguish them in an image. The used algorithm for this task is [22] which combines the 3D information in a 2D image in order to label the pixels.

3.3 Method application

In this part, it is explained how the programs use the information coming from the pre-processing phase. First, it is presented how the information is processed to obtain the features of a sample and the ways which are used in this thesis to get it. Then, it is commented what is the goal of a classification method and the paradigms in pattern recognition.

3.3.1 Feature extraction

This stage is very important in order to work with a limited quantity of data. The process in which the input data is reduced is called *dimensionality reduction.* There are two major approaches to *dimensionality reduction*: feature selection and feature transform. Feature selection reduces the feature set by discarding features. Feature transform refers to building a new feature space from the original variables, therefore it is also called *feature extraction* [23]. The use of a limited feature set simplifies both the representation of patterns and the complexity of the classifiers. Consequently, the resulting classifier will be faster and use less memory [9]. Although, it can be intuitive to think that it would be better to recognize a pattern using a large number of features, this is wrong. It is true that by increasing the number of attributes, the precision with which the input variables can be specified increases. But this leads to an exponential grow of the quantity of training data needed to specify the mapping. This phenomenon has been termed the *curse of dimensionality* [24]. The amount of the input data of this thesis is 640x480x6=1843200. 640x480 is the size of the image, and 6 (3+3) are the depth (*x, y, z*) and colour (r, g, b) values of each point of the image. If it is wanted to make a classification among different patterns, the same dimensionality for all the patterns has to be chosen. This stage is called *normalization of data*. As an example, the dimensionality of VOSCH features is 137, as explained in Section 3.3.1.1.
Point clouds cannot give information by themselves, the information of the clouds that is going to be used is implicit. To be able to work with point clouds and detect different patterns using this information, some characteristics have to be chosen in order to make a differentiation among classes. These features have to be good enough to be able to describe the objects as well as possible so that the classification of the samples can be done.

3.3.1.1 Voxelized Shape and Color Histograms

Voxelized Shape and Color Histograms (VOSCH) [25] is a method which creates feature descriptors using shape and colour information. The idea fits well with the use of the Kinect sensor because the sensor provides this data in synchronization. The method is based on Circular Color Cubic Higher-order Local Auto Correlation (C^3- *HLAC)* [26] with a slight difference, which reduces the number of features and makes the descriptor to be rotation-invariant, and on the Global Radius-based Surface Descriptor (*GRSD)* [27]. C^3- *HLAC* descriptor is a high-dimensional vector that measures the summation of the multiplied RGB values of neighbouring voxels in a 3x3x3 grid around a voxel grid of arbitrary size. Each bin in the descriptor is differentiated by the RGB colour space and the relative position of the two neighbouring voxels. GRSD is a histogram that counts the number of transitions between different types of voxels. It counts transitions between the geometric classes of voxel surfaces, which are: free space, plane, cylinder, sphere, rim and noise. The variation of C^3-*HLAC* used in VOSCH creates 117 features for a point cloud and *GRSD* 20, this makes the dimensionality of the VOSCH descriptor 137. VOSCH descriptors are saved as a descriptor type pcd file.

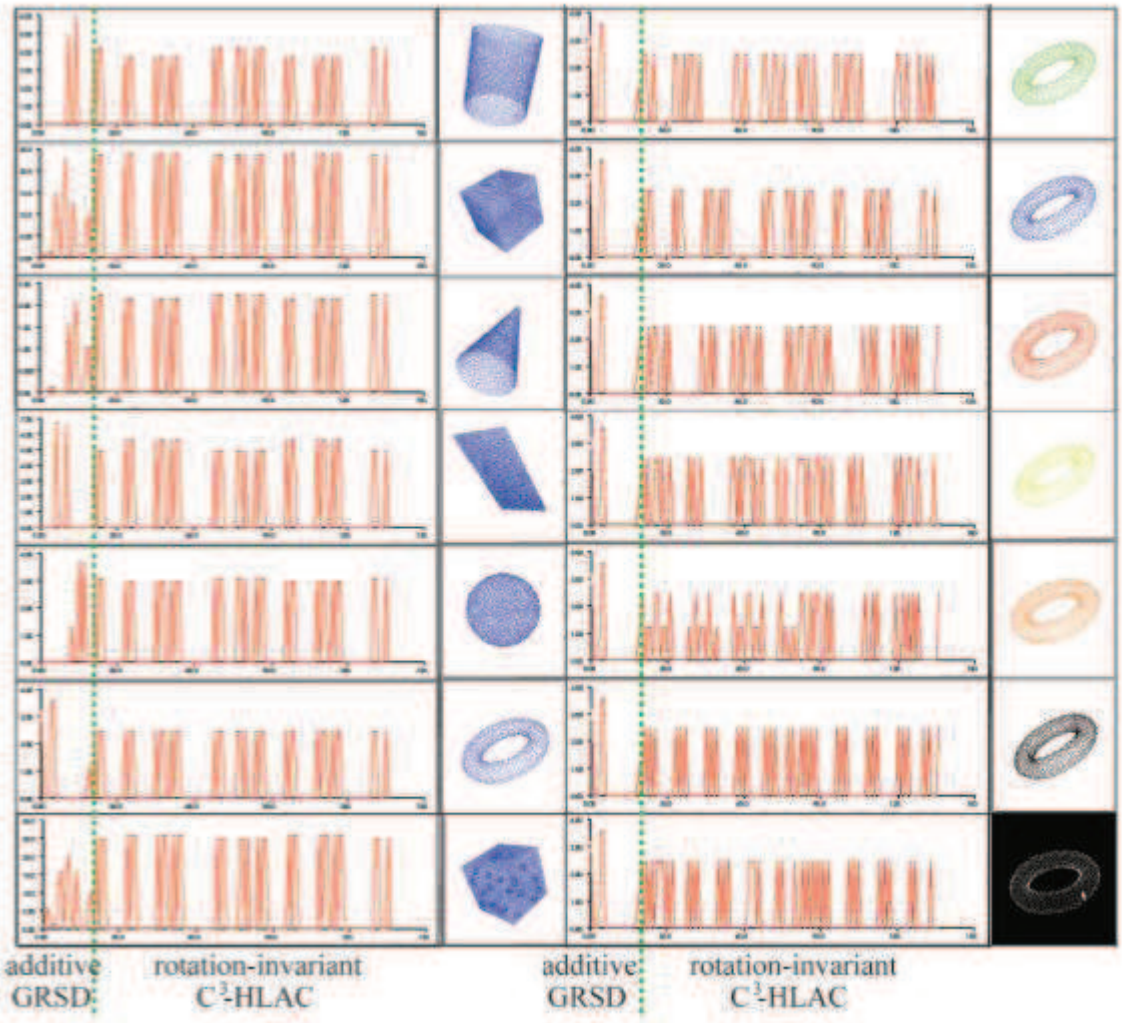

Figure 3.4 Example of scaled VOSCH histogram of different categories of objects (top-down: cylinder, cube, cone, plane, sphere, torus, die) have different values in the first 20 bins of their histograms. Right: different colors of the same category of a torus have different values in the last 117 bin in the histograms [25].

3.3.1.2 AdaBoost features

Since the goal of AdaBoost in this thesis is not to work with features as complex as VOSCH, new and easy features have been calculated, they are simply height, width, length and the number of points of the observation. In this approach, colour has been ruled out because the colour of some objects in the classes from the data-set can vary. This does not mean that the colour information is useless, it means that colour is more useful when objects which are trying to be detected have mainly a fixed colour. So AdaBoost detection rules are focused on the basic geometrical properties of an object. These are the height, the volume, the proportion of the object and its number of points:

- Height classifier: the classifier differentiates the classes using the height feature.

- Volume classifier: the volume of the objects is used to classify the samples, taking into account that the calculated volume is the one of the point cloud which describes the object and not the real dimensions. It is calculated multiplying height x width x length. This value can vary depending on the perspective of the sample, for example in mugs, in which sometimes the handle is seen and sometime it is not.

- Proportion classifier: to calculate the classifier, the three features are used. Width and length are summed and are divided by the height. This calculation generates the proportion value.

- Point classifier: the classifier tells the difference between object using the number of points of a point cloud. It differentiates not only between big and small object but also between object which can be the same size but different geometric shape or texture. In an object with concave or convex parts, more points will be detected as in one with straights parts. This way, this classifier can be considered as an easy surface classifier.

3.3.2 Classification method

Once the feature extraction is done, the classification method is applied. With its use, a pattern recognition using the new sample can be achieved. Pattern recognition can be defined as the classification of data based on knowledge already gained or on statistical information extracted from patterns and/or their representations [9]. In this case, "*knowledge already gained*" is the information of the extracted attributes which is done in the training phase.
The two most important paradigms in pattern recognition are statistical [14] and syntactic [28]. In this classification problem, the programs work with noisy data and uncertainty so statistical paradigm is used. On the other hand, the background for syntactic pattern recognition is provided by the formal language theory.

3.4 Post-processing

In this section, it is explained what kind of calculations are done after the classification in the programs of this thesis. It is commented how the recognition information is presented and how the important information is shown.

3.4.1 Sensor information representation

The information from the sensor is presented in a window. On the upper left part, the colour image which the sensor detects is shown and underneath, either depth image in greyscale or infrared image. This can be decided pressing key 'c'. On the lower part, the correspondence between some keys on the keyboard and their action is written. The main part of the window is occupied by the 3D representation of the detected objects. The point of view of the point cloud as well as the distance to it can be changed using the mouse. The x-axis is described by a red arrow, y-axis by a green arrow and finally z-axis by a blue arrow. The intersection of all of them represents Kinect position. The recognized class or classes are written in red on the upper-right part of the window, one under the other. If no objects were detected, it would be written five "?" symbols. An overview of the sensor information representation is presented in Figure 3.5:

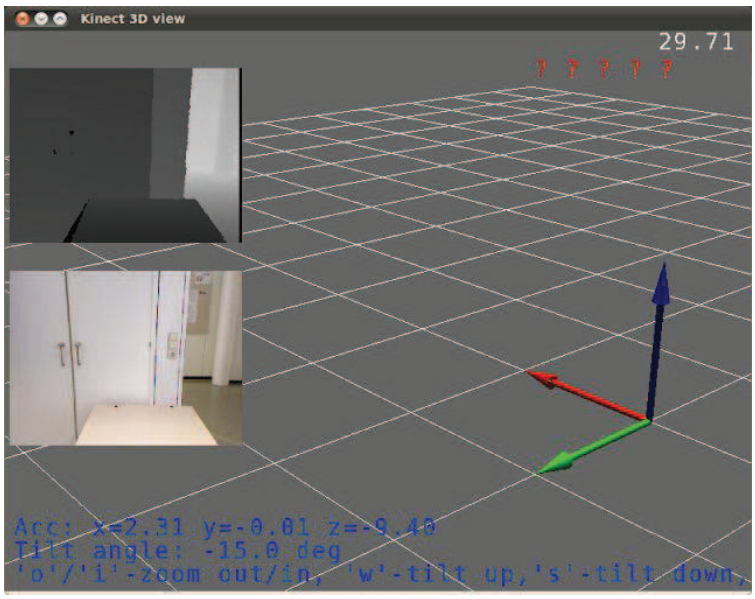

Figure 3.5 Recognition window. On the left, depth and colour information an on the middle the 3D space. No object is recognized in this picture, that is why the symbol "?" appears on the image

3.4.2 Detection representation

After detecting and classifying all objects in the scene, the name of the class of the objects is written on the upper right part of the window. On the colour image, a circle is drawn around every detected object and their class is written next to them in green. The classification information is represented as in Figure 3.6:

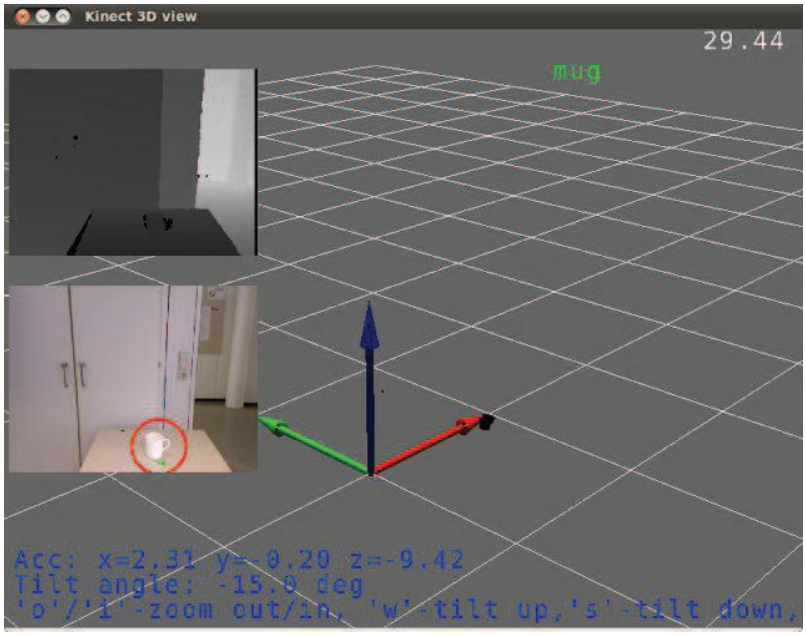

Figure 3.6 Recognition example of a mug in which the detected object is surrounded with a red circle with the name of the class "mug" in green. The point cloud is drawn in black in the space.

4 Implementation

This thesis has been developed using Linux operative system, Ubuntu 10.04 LTS on a AMD Athlon 64 X2 Dual Core Processor 4800+. To be able to interact with Kinect sensor, Mobile Robot Programming Toolkit (MRPT) [29] libraries have been used to make it possible to read and modify the information coming from the environment. These libraries have been integrated into Robotic Operative System (ROS) [30], which was used to create the SVM recognizer program. AdaBoost programs have been developed using Eclipse. The ROS project for SVM was also exported to Eclipse in order to be able to use all the facilities that programming with it offers. AdaBoost programs were compiled using g++ 4.4.3. and SVM using the open-source build system CMake 2.8.0.

4.1 Point Cloud Data (PCD) format

It is interesting to comment Point Cloud Data (PCD) files [31], because in further steps (learning phase and real-time application) these are going to be used. PCD files are text files that the Point Cloud Library [31] uses to work with 3D points. These files are the main source to create features in the selected methods. PCD files contain the following entries:

- **VERSION:** specifies the PCD file version.

- **FIELDS**: specifies the name of each dimension/field that a point can have.

- **SIZE**: specifies the size of each dimension in bytes. Examples:

    - *unsigned char*/*char* has 1 byte
    - *unsigned short*/*short* has 2 bytes
    - *unsigned int*/*int*/*float* has 4 bytes
    - *double* has 8 bytes

- **TYPE**: specifies the type of each dimension as a char. The current accepted types are:

    - **I** - represents signed types int8 (*char*), int16 (*short*), and int32 (*int*)
    - **U** - represents unsigned types uint8 (*unsigned char*), uint16 (*unsigned short*), uint32 (*unsigned int*)
    - **F** - represents float types

- **COUNT**: specifies how many elements does each dimension have. For example, *x* data usually has 1 element, but a feature descriptor like the *VFH* has 308. Basically this is a way to introduce n-D histogram descriptors at each point, and treating them as a single contiguous block of memory. By default, if *COUNT* is not present, all dimensions' count is set to 1.

- **WIDTH**: specifies the width of the point cloud dataset in the number of points. *WIDTH* has two meanings:

    - it can specify the total number of points in the cloud (equal with **POINTS**) for unorganized datasets;
    - it can specify the width (total number of points in a row) of an organized point cloud dataset.

- **HEIGHT**: specifies the height of the point cloud dataset in the number of points. *HEIGHT* has two meanings:

- it can specify the height (total number of rows) of an organized point cloud dataset;
- it is set to **1** for unorganized.

- **VIEWPOINT**: specifies an acquisition viewpoint for the points in the dataset. This could potentially be later on used for building transforms between different coordinate systems, or for aiding with features such as surface normal, that need a consistent orientation.

- **POINTS**: specifies the total number of points in the cloud. As of version 0.7, its purpose is a bit redundant.

- **DATA:** specifies the data type that the point cloud data is stored in. As of version 0.7, two data types are supported: *ascii* and *binary*.

The entries must be specified precisely in the above order. An example of a pcd file is shown in Listing 4.1, where the header and the first 2 points of the point cloud are presented:

```
# .PCD v.7 - Point Cloud Data file format
VERSION .7
FIELDS x y z rgb
SIZE 4 4 4 4
TYPE F F F F
COUNT 1 1 1 1
WIDTH 213
HEIGHT 1
VIEWPOINT 0 0 0 1 0 0 0
POINTS 213
DATA ascii
0.93773 0.33763 0 4.2108e+06
0.90805 0.35641 0 4.2108e+06
```

Listing 4.1 Structure of PCD files. The header of this type of files is shown as well as the first two points of 213 that the point cloud contains. In this file, x, y and z and the float describing the color are used to represent the point cloud.

RGB values of the coloured points are represented as a float in pcd files. Since the RGB values received from the sensor are three floats, one for each channel, the following operations have to be done to calculate to actual value for the pcd file. Future versions of pcd files will try to use integer values in order to avoid calculating the transformation shown in Listing 4.2:

```
//TYPE CASTING  FLOAT R G B -> INTEGER R G B -> FLOAT RGB

uint32_t colorr, colorg, colorb, rgb;

colorr=((imagen.getAsFloat(iter2,iter1,2))*255);
colorg=((imagen.getAsFloat(iter2,iter1,1))*255);
colorb=((imagen.getAsFloat(iter2,iter1,0))*255);

colorr=colorr<<16;
colorg=colorg<<8;

rgb = colorr | colorg | colorb;

float urgb = *reinterpret_cast<float*>(&rgb);
```

Listing 4.2 Transformation of color information in order to work with it in PCD files. First, the three floats that represent the color (RGB) and transformed as integer. Then, a transformation has to be done to calculate the final float of the RGB value.

4.2 Pre-processing Real-time classifier

In this case, instead of using stored data, the information is read in real time from the Kinect sensor. To be able to read the data, Mobile Robot Programming Toolkit (MRPT) is used. MRPT implements a common C++ interface to the sensor using the openKinect´s *libfreenect* [32] library to actually access the sensor. As the programs mainly use the picture and depth information, the explanation of the used structures is going to focus on this information. To read data from the Kinect, MRPT saves it in a *CObservation3DRangePtr* pointer, its structure containing all detected information from the Kinect in a specific time point. To be able to use the image information two attributes are read from the pointer, these are, *hasIntensityImage*, in which a Boolean value tells if the field *intensityImage* contains valid data, and *intensityImage*, which belongs to the *mrpt::utils::CImage* class. This class represents the image as a 480x640 matrix, in which each point of the matrix represents the colour saved in the following order: blue, green, red (bgr).

BGR(1, 1)	BGR (1, 2)	° ° °	BGR (1, 640)
° ° °			° ° °
BGR (480, 1)	° ° °		BGR (480, 640)

Figure 4.1 Structure of the matrix which contains the image. Each cell of the matrix contains a BGR value and represents the colour of the point in that position in a 480x640 colour image.

To get access to depth information *hasPoints3D* tells if the field *points3D* contains valid data. The information of *points3D* is saved in 3 different arrays, *points3D_x, points3D_y* and *points3D_z* which contain the x, y and z values of the coordinates in the image respectively. The size of each array is 307200 (640x480) and the depth information is saved one row after another. There is also further interesting information, such as, the *mrpt::math::CMatrix rangeImage,* which, if *hasRangeImage* is true, contains the floats with the range data as captured by the camera in meters or the attribute *timestamp*, which tells the specific time point when the information was taken. This attribute is used to be sure that the continuous information is read in a correct order.

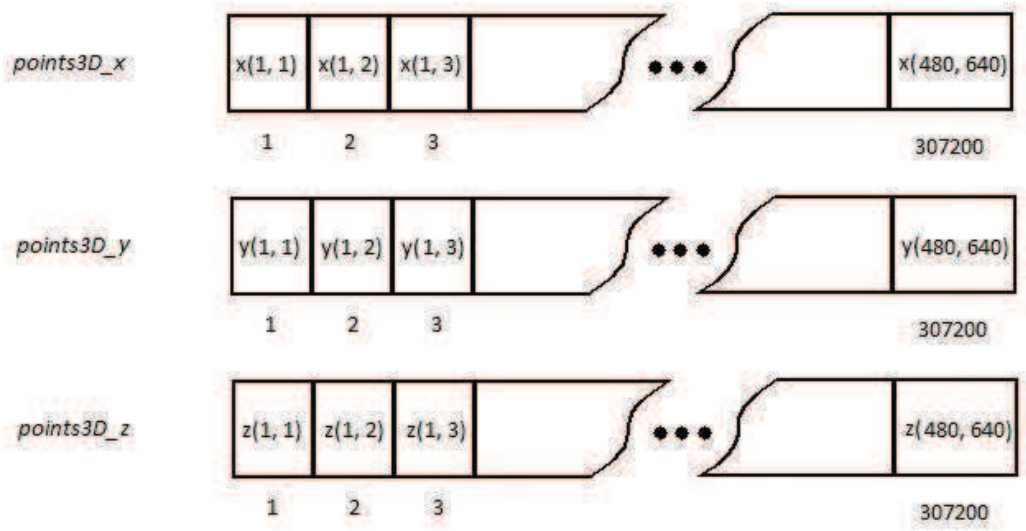

Figure 4.2 Structure of the arrays which contains the depth points detected by the Kinect sensor. These arrays can be considered as matrices because the values of the rows of the image are saved one after another

The possibility of taking information at a same time point makes it possible to work with depth data in synchronization with colour data. That is a reason why Kinect sensor offers huge possibilities in real-time applications.

4.2.1 Range depth elimination

To be able to use in following steps only the useful information for the program, invalid points are canceled using *points3D_x.* As this array contains the distance of all detected points (in which the value of non-detected points and further than 3,5m points is already 0), it is read and the values not corresponding to the subsequent range can be changed to 0 in order to represent them as invalid points.
In the programs, 2D images are pre-processed using depth information, that is, a 480x640 matrix is filled with the 3D coordinates of corresponding pixel. Class *Vertex* consists of a label as well as the x, y, z coordinates of a pixel in an image. This class is used for elements of the matrix. If it is wanted to set a point of the image as an invalid point, the values of *x, y, z* are changed to 0 and the value of the label to -1, otherwise the coordinate values are set to the values of *points3D*. In Figure 4.3, the colour image that the sensor detects is presented. The objects are situated at 1 meter and 0,5m so the detection of all the point of the object will not be complete. In Figure 4.4, it is shown the point clouds that the program recognize. Some parts of the object are not recognized because they are not in a valid range.

Figure 4.3 Color image of the recognition of two samples. The green and orange box is situated at 0,5m from the sensor and the bowl at 1m.

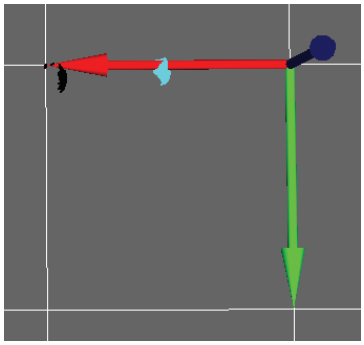

Figure 4.4 Point clouds of the two samples. Light blue cloud corresponds to the box of Figure 4.3 and black one to the bowl. In both cases, half of the object is discarded because their points are not in the valid range.

4.2.2 RansacHTWG class

For this thesis, the class *ransacHTWG* was created. It is based on the *mrpt::math::RANSAC::execute*, which is the method that calculates the final model. The algorithm receives a *CColouredPointsMap* pointer which contains the valid points of the observation in that moment. This pointer receives the data from the *points3D* arrays. Just the points with a value higher than zero in *points3D_x array* are included in that structure. *CColouredPointsMap* saves the points in an array without any order, that is, a point cloud. The main reason to use this structure instead of *points3D* or the *Vertex* matrix is the speed of the calculations. If the whole image was used, RANSAC should check if the point it is working at that moment with is valid or not. That is why the parameter contains just the point cloud. Another reason is that the input parameter is just a pointer and no a huge amount of memory is needed to call the RANSAC algorithm; the size of the parameter is independent from the point cloud.
RANSAC mainly uses two methods, *ransacHTWG::ransacHTWG3Dplane_fit,* which calculates,

using three points from the set, the possible model (plane) for each iteration and *ransacHTWG::ransacHTWG3Dplane_distance*, which calculates the distance between the plane and the rest of the points in order to add them or not to the result set. The used distance to discriminate between inliers and outliers is given by *distanceThreshold* which value is 1cm. It could be considered a large number but two things have to be taken into account: the measurement error, which can be up to 1cm and that it is a real-time program, so the plane needs to be calculated as fast as possible; making this threshold bigger reduces the number of iterations of the algorithm. The final result is described by the coefficients which represent the plane. Figure 4.5 show the times in relation to the chosen threshold for an object of approximately 3500 points.
Although the lower parts of the objects can be included as part of the plane, as in [1] where even small objects are completely merged, the solution model makes a fairly good approximation of the plane of the table. The worst example of failure in the point detection is the banana class. It is the lowest kind of object so, if the banana is not big enough, the amount of noise is big or the position and orientation are not favourable, the number of detected points for the banana will not be enough to be considered as a recognizable object (Section 4.2.3).

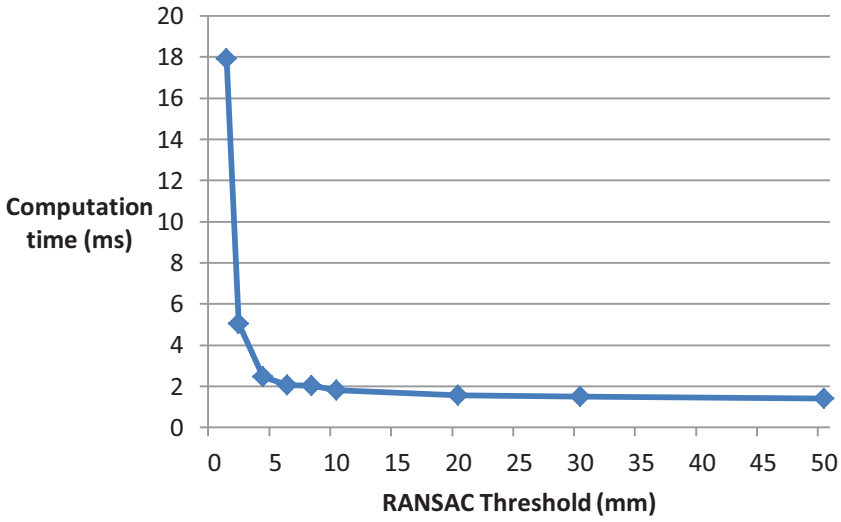

Figure 4.5 Evolution of the RANSAC computation time (Y-axis) in ms depending on the threshold (X-axis) in cm. From six millimeters, to choose a bigger threshold, do not make a proportional saving of time.

Once the plane is calculated, it is checked if the valid points of the *Vertex* matrix are part of it using the method *isNotFloor*. If a point is considered as a point of the plane, the *x, y, z* are set to 0 and the label to -1, as already done in the depth range elimination (Section 4.2.1).

4.2.3 Segmentation

In this step, just the points of the possible objects are saved in the *Vertex* matrix. In order to make the segmentation of the image which the matrix contains, it is used an algorithm which was developed in HTWG Konstanz [22]. It uses the *Vertex* matrix, to distinguish the different regions on the image. To check if a pixel belongs to the same region than its neighbour pixel, Euclidian distances are measured using the depth information of both pixels. In this manner, after segmentation a matrix is obtained, in which, points with different labels mean different regions (objects). In order to use the information of the different regions, *Organizador* class was created. This class is basically an array of arrays, which mainly saves all the points (*x, y, z*) of the same object in a component of the array and the colour associated to this objects in order to draw it in future steps. This colour will be the same for all points of an object.
Since the goal of the thesis is not to detect very small objects or big ones, only components with point clouds, those with a number of points between 800 and 6000 will be considered. This manner, bigger and smaller objects than the ones we are trying to detect will not be processed, neither will

the points which represent noise in the image.

4.3 Methods

In this section it is presented how the classifier is created using the data from the data-set and how the programs use the information from the classifier as well as from the sensor to make the recognition in real time. In supervised learning the information used in the system is a pair, whose values are the input-values, which are the calculated features, and the output values, which are 0 or 1 in binary classification depending if we want the system to classify the sample as a positive or negative. With this information and depending on the algorithm, different values are calculated (support vector machines, alpha values) in order to classify a new sample in the future.
There are three real-time programs; one of them uses SVM multiclass method from Open Source Computer Vision (OpenCV) [2]. In order to make a multiclass classification, OpenCV uses One-Against-One strategy. The two remaining programs use AdaBoost method, one using One-Against-One strategy and the other One-Against-All strategy. One-Against-One strategy creates a classifier for each pair of different classes which predicts in which class of the pair better fits a new sample. These classifiers are used to classify a new sample and the result of each one is saved. The class with more votes will be the predicted class. One-Against-All strategy creates classifier for each class (*n* classifiers) to know if a sample belongs to that class or not. Each classifier predicts a value, which can be interpreted as the probability of the new sample to belong to that class. The predicted class will be the class with the highest value. To be able to use libraries connected with point clouds and artificial intelligence Robot Operative System (ROS) was installed. ROS is a set of libraries which collect a huge amount of structures and algorithms used to work in robotics.

4.3.1 Training method

To create the SVM classifier, all the instances which belong to the training-set and are saved as VOSCH features in the data-set are added to an *n*x*m* matrix, where *n* is the number of instances and *m* is the number of VOSCH features, 137. Since *CvSVM* supports multiclass classification, the desired output for each instance is a label which is different for each class, being the label a float. Using this matrix, the desired output of the instances and some parameters for the type of classifier, *CvSVM* class from OpenCV is used to create the classifier using *predict* instruction. Since *CvSVM* supports multiclass classification, the desired output for each instance is a label which is different for each class. VOSCH features can be extracted using the VOSCH library of ROS using pcd file with a point cloud as an input. Taking into account the conclusions presented in [33], a linear classifier is created. The values of the SVMs are saved in a xml file named "*svm.xml*".
In AdaBoost, four rules of the thumb are created. These weak classifiers are *HeightClassifier, PointClassifier, VolumeClassifier* and *ProportionClassifier*. *HeightClassifier* and *PointClassifier* use the height of the observation and the number of points of the point cloud to distinguish between classes. *VolumeClassifier* distinguish between the volume of the smallest polyhedron of six faces in which the observation can fit inside. *ProportionClassifier* calculates a proportion between depth and width, and height of the observation. Although the last two weak classifiers do not use the real volume or real proportion of the object, the data-set sample and the real-time observation come from the same source, that is why the comparison can be made. In the Diagram 4.1, the class diagram for AdaBoost programs is presented.
Thresholds of the weak classifiers are given by an array, in which each component corresponds to a class, in which a range of values for each classifier is defined. This way, a weak classifier will consider a sample a member of that class if the sample values are in the range. AdaBoost One-Against-One and One-Against-All are implemented as arrays of AdaBoost classifiers, in which following the principle of AdaBoost presented in Listing 2.1, alpha values will be created for each classifier.

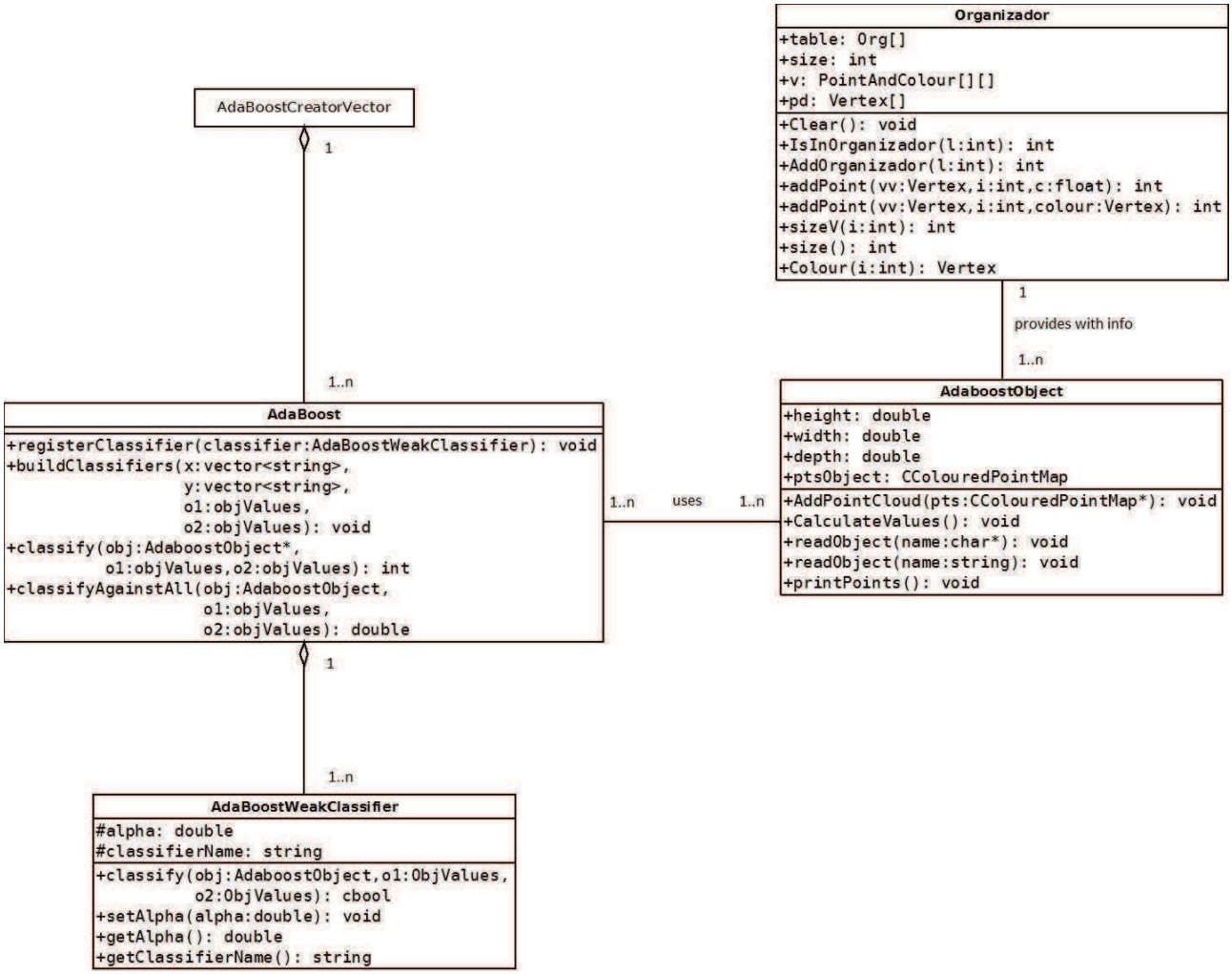

Diagram 4.1 AdaBoost class diagram

4.3.2 Real-time classifier

At this point, point clouds are saved in the *Organizador* instance (each object in a different array component). A loop will sequentially process the objects in both methods.
In SVM, each iteration has four main parts:

- Creation of a pcd file with the points of an object in order to be able to calculate VOSCH features using the file,
- Addition of the point cloud to a *CColouredPointsMap*, which is the array of coloured points which will be used to draw the points on the screen. It is reminded that all points of a same object have the same colour, which was attached by *Organizador* instance,
- VOSCH features calculation using the library from ROS,
- Classification of the features with *CvSVM* using as classifier the "*svm.xml*" file which has the SVMs.

In AdaBoost is slightly different:

- Creation of an *AdaboostObject* which saves the point cloud of an object contained in the component of *Organizador*,
- Addition of the point cloud to a *CColouredPointsMap*,
- Classification of the *AdaboostObject* using the array of classifiers(*One-Against-All* or *One-Against-One*) and registration of the results in an array of results with the same dimension as the number of classifiers,
- Calculation of the detected class depending of the results in the array. In AdaBoost One-Against-One, the result is the index with the highest amount of "votes". Each One-One classifier can "vote" for one the two classes the classifier consists of. This "vote" means that the component of the array which corresponds to the voted class will be increased by one (in the beginning all components are set to 0). In AdaBoost One-Against-All, the index with the highest predicted value for each One-All classifiers is the result of the recognition.

4.4 Post-Processing

As seen in Section 4.3.1 and Section 4.3.2, the way that a classifier uses to describe the recognized class is a number. In SVM, the number is a float which corresponds to the label attached to a class. In AdaBoost, the index of the result array indicates the recognized class.
In order to get the name of the recognized class, an easy calculation has to be done. The final label obtained by SVM has to be transformed to the word which represents the class. Since the results in AdaBoost are saved in an array, the transformation is done from the index of the array with the highest value to the corresponding word of the class.
Once the name of the class is known the name is written on the upper-right part of the detection window (Figure 3.6). On the left part of the window, a green circle (red in AdaBoost One-Against-One) is drawn around the detected object in the colour image. The method *drawCircle* of *CImage* class is used. As center of the circle, a close point to the last one of the point cloud of the object is chosen. Inside the circle, the name of the detected class is written again.
Valid point clouds which are contained in *Organizador* are drawn on the middle of the window loading the points using the instruction *loadFromPointsMap()*. In SVM program, point clouds are directly drawn on the window, on the other hand, in both AdaBoost programs a transformation is done. In this case, the point cloud is drawn at the same height of the point (0, 0, 0) (Kinect position). If no transformation is done, the point cloud is presented in a position which depends on the inclination of the sensor, that is, beneath the Kinect position. The calculation is done rotating the points of the point cloud 15º on Y-axis using the Equation (4.1):

$$R_y(\beta) = \begin{bmatrix} \cos\beta & 0 & sin\,\beta \\ 0 & 1 & 0 \\ -sin\,\beta & 0 & \cos\beta \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \qquad (4.1)$$

In which R_y is the rotation on Y-axis, β is the value of the angle, in this case 15º and $[x \quad y \quad z]^T$ are the coordinates of a point.
This way, the point cloud that the sensor detects is shown on the window as the Kinect was situated on the table pointing towards the object.

4.5 PCD Viewer

PCD viewer is a program from the stack *perception_pcl* of ROS which makes it possible to represent pcd files. The program helps to make a visual differentiation between the instances. It was used to compare point clouds from the same class and from different classes in order to check their similarity. This can be the reason of a misclassification. The following images (Apple14 is

described in Figure 4.6, Banana 62 in Figure 4.7, Bowl 12 in Figure 4.8, Mug48 in Figure 4.9, Plastic Cup11 in Figure 4.10 and Tobacco6 in Figure 4.11) describe an example for each class showing a point cloud and its corresponding VOSCH histogram.

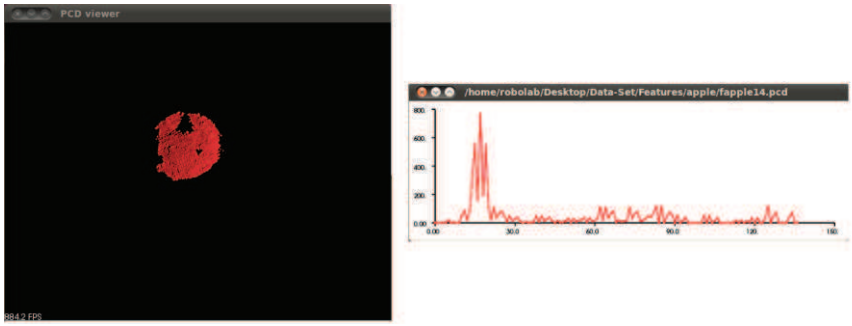

Figure 4.6 Apple14 point cloud and VOSCH histogram

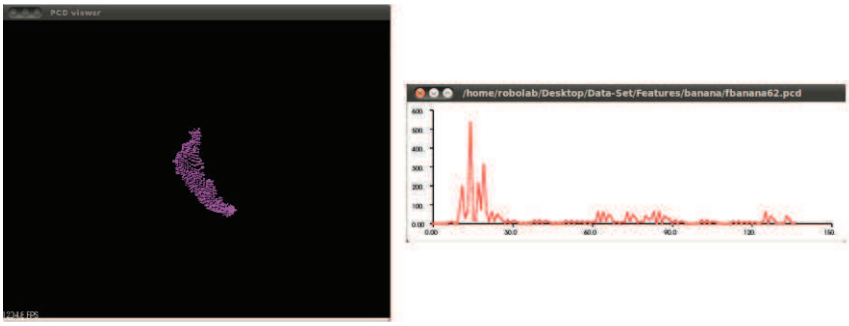

Figure 4.7 Banana62 point cloud and VOSCH histogram

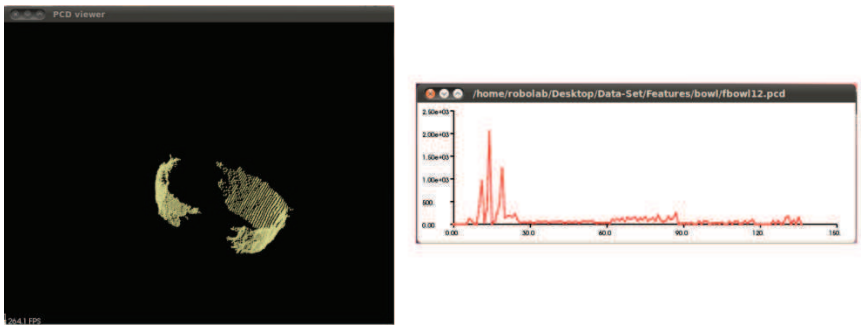

Figure 4.8 Bowl 12 point cloud and VOSCH histogram

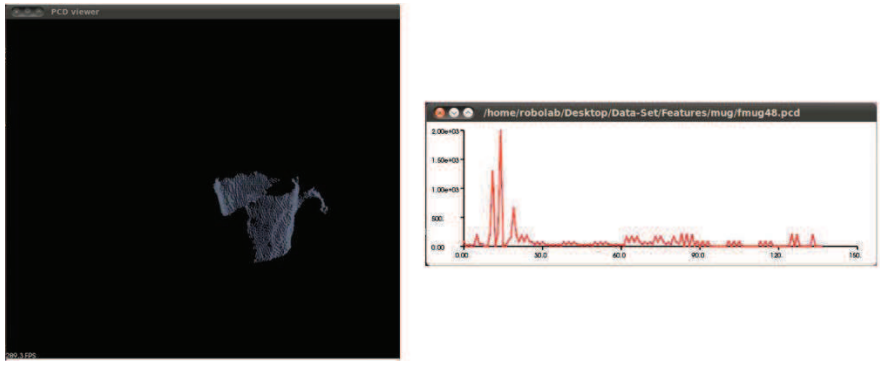

Figure 4.9 Mug 48 point cloud and VOSCH histogram

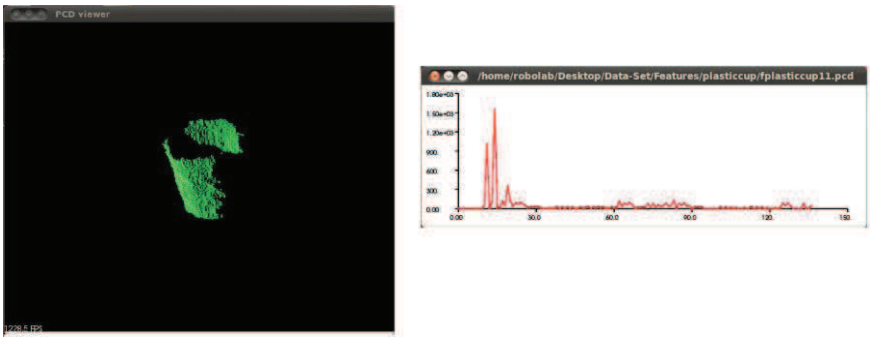

Figure 4.10 Plastic cup 11 point cloud and VOSCH histogram

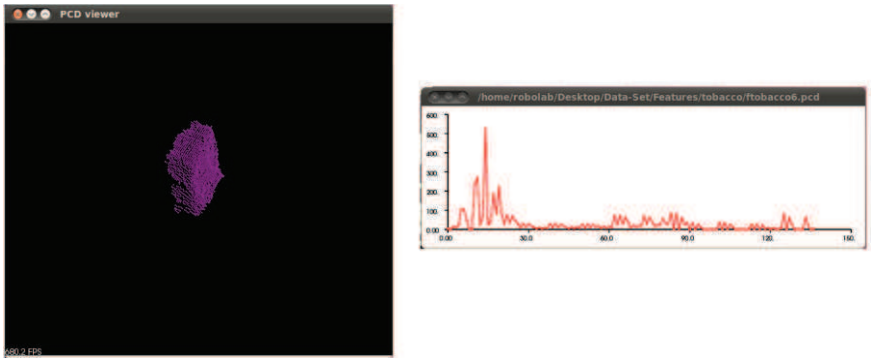

Figure 4.11 Tobacco 6 point cloud and VOSCH histogram

Since the first 20 values of the VOSCH histograms describe the shape of a sample, some visual comparison of the point clouds using PCD Viewer can be done. For example, the shape values of a mug and a plastic cup can be similar, as shown in Figure 4.9 and Figure 4.10.

4.6 Implementation characteristics

As introduced in Section 1.3, recognition programs can detect and classify more than one object at the same time. The classification is processed by the method sequentially in order of detection; first valid component of the *Organizador* instance is the first to be analyzed.
Figure 4.12 shows the classification of three samples using AdaBoost One-Against-All. The detected objects are a mug, a bowl and a tobacco packet.
A good property of VOSCH features is that the descriptor is rotation-invariant. An object can be

laid horizontally on the table and it could be still recognized. This could happen if the point cloud that the program detects is not very different from the ones in the data-set. If a mug is laid horizontally on the table, and the orientation is such that the sensor can see its bottom, the point cloud created will be very different from the ones the program has learnt. The result of such sample would be uncertain. Figure 4.13 shows a correct recognition of a plastic cup in these conditions, but Figure 4.14 shows a misclassification of a plastic cup with a slight different orientation than the one in Figure 4.13. In Figure 4.15, three different tobacco packets are recognized. The one in the middle is situated in a different position from the training samples. In the case of AdaBoost, this property does not exist. If an object is put with a different inclination than in the training-samples, the geometrical properties of the object can be totally different. If an apple is used an example, the properties will not be very different (due to the shape of the apple can be similar to a sphere), but if a tobacco packet is used instead, the dimensions of the object, taking the table as reference (the absolute dimensions, such as volume, are always the same) will be very different. Because of these reason, a good classification will not be normally achieved.

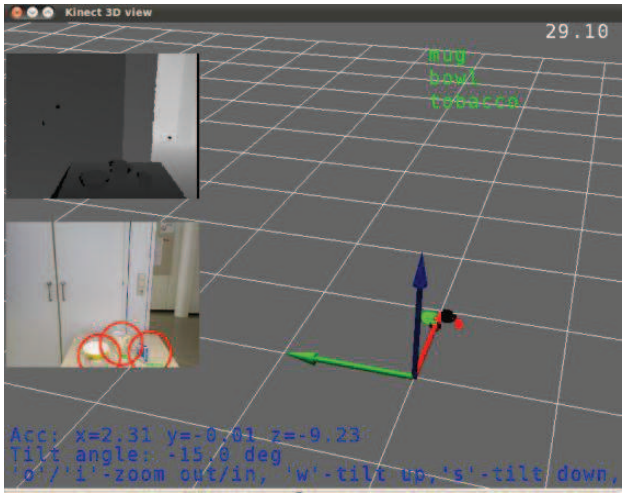

Figure 4.12 A correct multi-classification using SVM. The classified samples are a mug, a bowl and a tobacco packet.

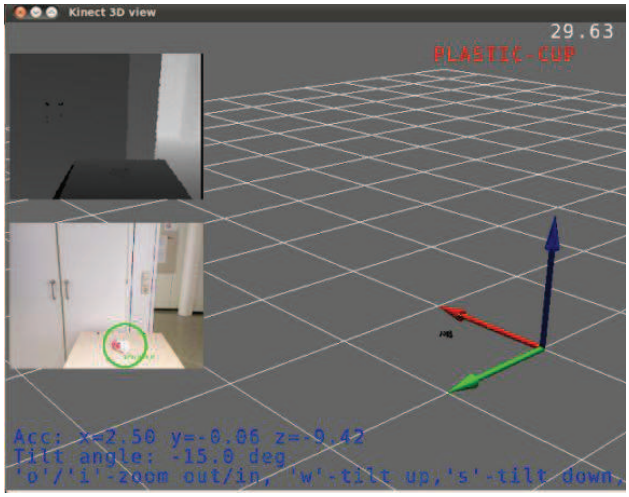

Figure 4.13 A plastic cup laid horizontally on the table is recognized by SVM. Although the inclination of the sample is different than the training-samples, the created point cloud is similar so the properties of shape and color will be also similar.

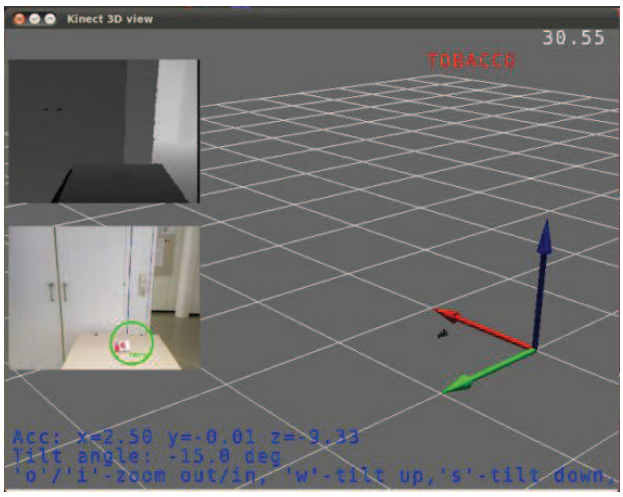

Figure 4.14 The picture shows a plastic cup which is recognized by SVM as a tobacco packet. Kinect detects jus the side of the cup so the shape of the recipient is lost

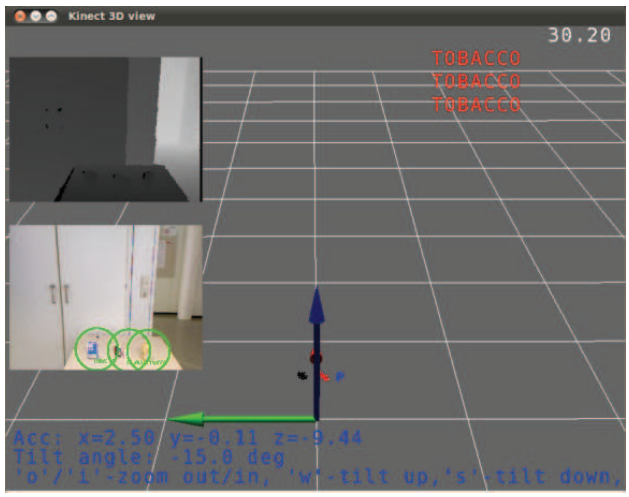

Figure 4.15 Another example of a detected object with a different inclination (tobacco packet in the middle). In this case, a multi-classification of three samples (tobacco packets) is done.

The same problem can be present if an object is just partially detected. In this thesis, there are two ways in which this can happen. The first one, it is presented in Figure 4.4, that is, if some parts of the object are out of the valid range. The second one, if an object is in front of another object that it is wanted to be detected. These ways, the dimensions of the point cloud will change as well. In AdaBoost, this will probably lead to a wrong classification. In SVM, The correct classification of the sample will depend on the part the other object is blocking.

4.7 Possible improvements of the implementation and difficulties

Although the programs can detect very different objects of the same class, there is an important limitation when they analyze an object which belongs to a class which has not been learnt by the classifier. In this case, the programs always classify the sample as one of the classes that the classifier knows, if the number of points of the sample is in the range. Despite the fact that the goal of the thesis is not to recognize new classes, a possibility to solve this could be to recognize the object as unknown, if the probability given by the classifier is not big enough to be sure that an object belongs to a known class. Figure 4.16 shows the misclassification of an unknown class using AdaBoost One-Against-One method and Figure 4.17 shows it using SVM. Both programs detect an object but the classification is wrong because the class they are trying to recognize has never been learnt by the classifiers.

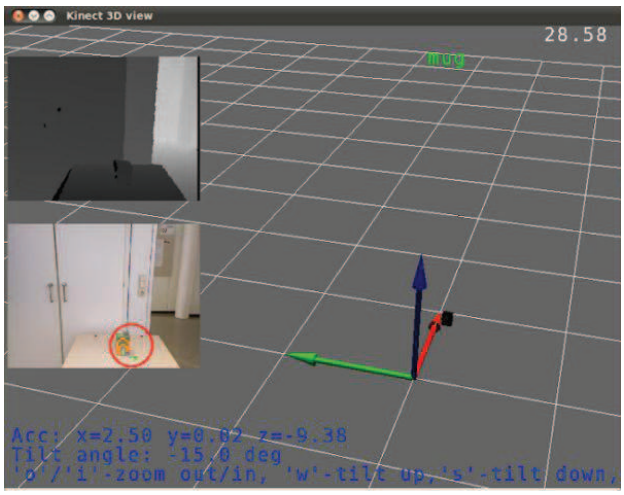

Figure 4.16 AdaBoost One-Against-One recognition of an unknown class (box) in which the predicted class is a mug.

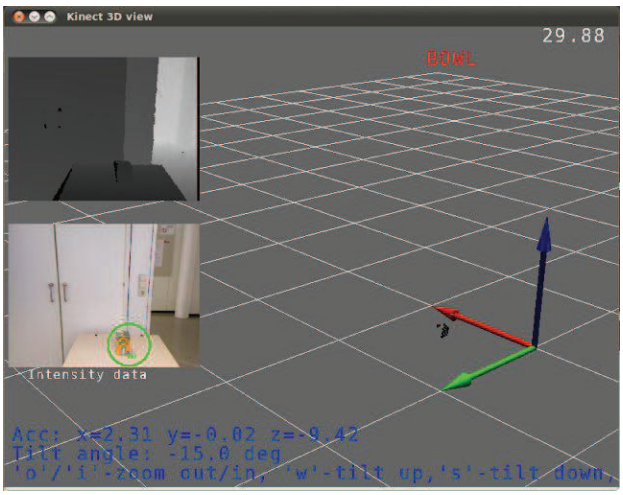

Figure 4.17 SVM recognition of an unknown class (box) in which the predicted class is a bowl.

Another interesting result is obtained in the program AdaBoost One-Against-One. Depending on the two classes which are part of a one-one classifier, a weak classifier can get no error in the training phase. This happens because features are simple and sometimes the values of the ranges are very different. As an example, banana-mug classifier can be presented if the samples used to train the weak classifier have no errors and describe normal objects of those classes. “Height”, weak classifier, will be able to make a perfect classification and the classification problem can be formulated as a trivial one, consequently this will be done depending only on the height, ignoring the rest of the other weak classifiers. A banana on a table will never be higher than a mug.

Talking about the project compilation, because of the installation of ROS, MRPT libraries had to be added to it in order to use them in ROS projects. The problem was that, at first MRPT used VOSCH and SVM methods as external programs. As it is a real-time approach, the call of an external program took almost 100ms each. Although library linking is apparently not a difficult problem to solve, first a deep study of CMake files had to be done because they are the type of files ROS uses to compile its libraries. Then there was a collision between PCL libraries defined in both ROS and MRPT which was solved renaming some libraries and changing dependencies between ROS and the Point Cloud Library. Finally, to be able to use Eclipse, the project created in ROS was exported to Eclipse.

As the Kinect development is a recent technology, the information and documentation that can be found on the Internet is to some extents not complete. Comments in algorithms or examples can hardly be found, so a deep study of algorithms, data types, data structures, libraries and dependencies which were not directly used was needed to be done.

5 Experimental results

In this chapter, the test programs of the different methods are presented. It is explained how they are built and the results they create. In the Section 5.3, the time that each real-time program needs to calculate a result is shown.

5.1 Test programs

The structure of the test programs is simple and it has always the same parts. First, the classifier is trained with the training-set. The way of training depends on the method which is going to be used. The idea is to create a training set in which, for each different class, the number of instances is between 30% and 75% trying to use at least 16 instances. Since these are multiclass-classifiers, a change in the training of a class can vary the results obtained by the classifier in another class. In order to get good results from the different methods, a study of the training-set has been done [34]. For each method, different numbers of instances from each class are used to train the classifier. Using ROC method, each combination of the number of instances can be analyzed in order to get a good training-set for the final test between methods. The criterion to choose the training-set is simple; the set with the highest hit-rate is selected. Once the classifier is created, the test-set is used, the classification results and the roots of the test pictures are written in a text file in order to be studied. In the following sections, the study of the different methods is presented.

5.1.1 SVM test

The following table represents eight different training-set combinations in which, the number of instances of each class is different (Table 5.1). The column on the left represents the name for the training-set with the specific number of instances on its right:

Training-set	Apple	Banana	Bowl	Mug	Plastic cup	Tobacco
A	8	8	8	8	8	8
B	8	16	16	16	8	8
C	8	16	16	16	8	16
D	8	16	24	16	8	16
E	8	24	24	16	8	16
F	8	16	24	24	8	16
G	8	16	24	32	8	16
H	8	32	24	16	8	16

Table 5.1 Different training-sets used for SVM named from A to H. Each cell contains the number of snapshots of a class (column) used to train the method with a training-set (row).

The different training-sets are chosen depending on the number of instances that each class has (Section 5.1). The following graphics represent the classification results of each class (Apple class is presented in Figure 5.1, Banana in Figure 5.2, Bowl in Figure 5.3, Mug in Figure 5.4, Plastic Cup in Figure 5.5 and Tobacco in Figure 5.6) in SVM using ROC method.

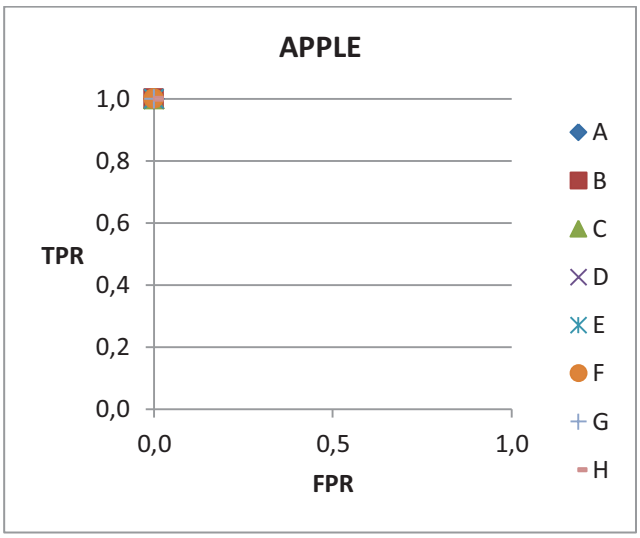

Figure 5.1 ROC representation for Apple class using SVM

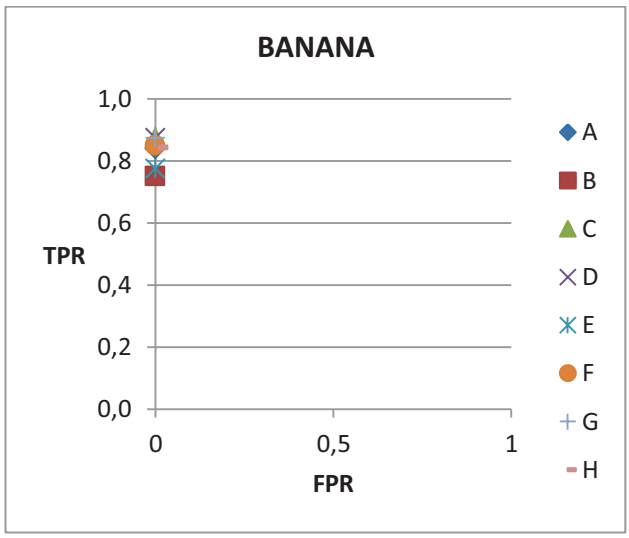

Figure 5.2 ROC representation for Banana class using SVM

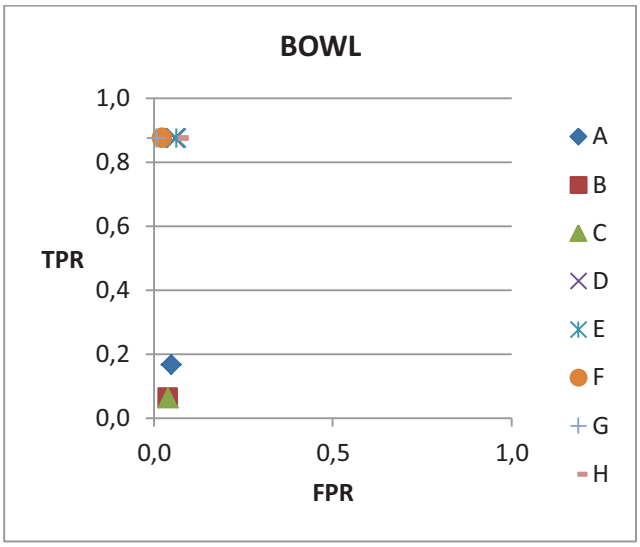

Figure 5.3 ROC representation for Bowl class using SVM

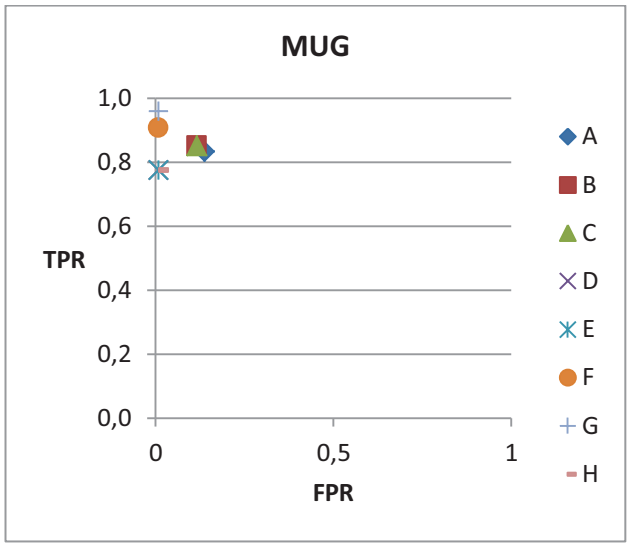

Figure 5.4 ROC representation for Mug class using SVM

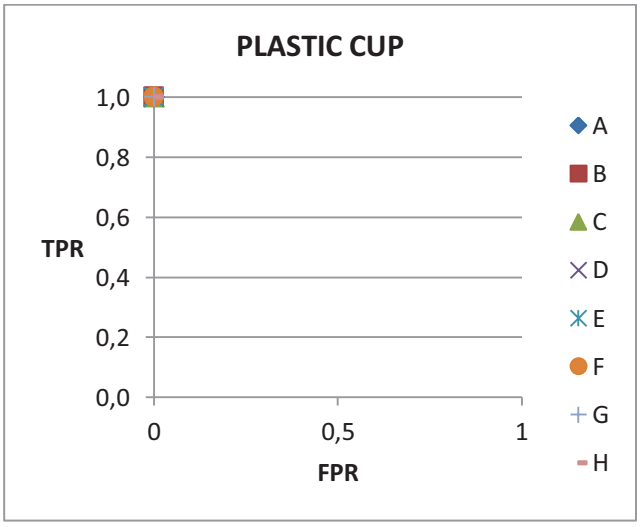

Figure 5.5 ROC representation for Plastic Cup class using SVM

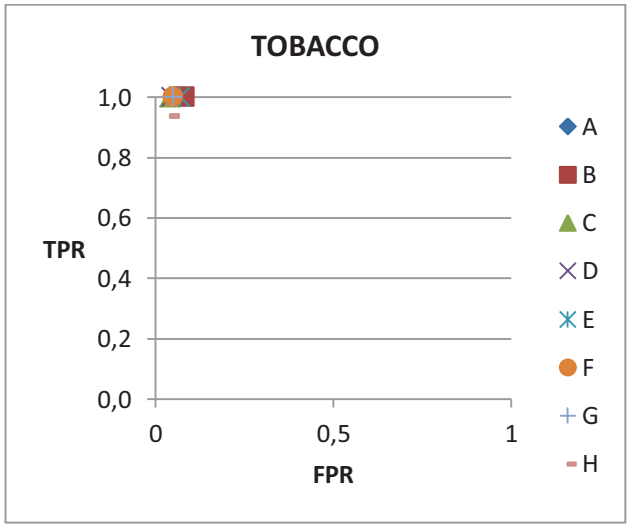

Figure 5.6 ROC representation for Tobacco class using SVM

The hit rates of all the possible training-sets are shown in Table 5.2:

Training-set	Apple	Banana	Bowl	Mug	Plastic cup	Tobacco	Overall
A	1,0	0,870	0,167	0,833	1,0	1,0	0,812
B	1,0	0,75	0,063	0,85	1,0	1,0	0,777
C	1,0	0,875	0,063	0,85	1,0	1,0	0,798
D	1,0	0,875	0,875	0,775	1,0	1,0	0,921
E	1,0	0,775	0,875	0,775	1,0	1,0	0,904
F	1,0	0,85	0,875	0,906	1,0	1,0	0,939
G	1,0	0,875	0,875	0,958	1,0	1,0	0,951
H	1,0	0,844	0,875	0,775	1,0	0,938	0,905

Table 5.2 Hit rates of the SVM method depending on the training-set (first column) and the class (first row). The last column indicates the hit rate counting all the classes.

The number of instances of apple and plastic cup are eight in all training-sets, because with that number of training instances, the classifiers obtain a perfect result. The hit rate for Bowl class is low if the number of instances is 16, but it increases if the recognizer is trained with 24.
Using the information from the graphics, the number of instances of G set is chosen in order to study SVM method:

Training-set	Apple	Banana	Bowl	Mug	Plastic cup	Tobacco
G	8	16	24	32	8	16

Due to the shape of the bowls, the difference between these instances is big. Its shape can be close to the shape of a mug (Figure 5.7) or can be almost as flat as a plate (Figure 5.8Figure 5.7) and sometimes Kinect can just detect a part of them. That is why the number of instances of bowl class is high compared with the number of bowl samples, otherwise the results of the recognition would be not very precise (Figure 5.3).

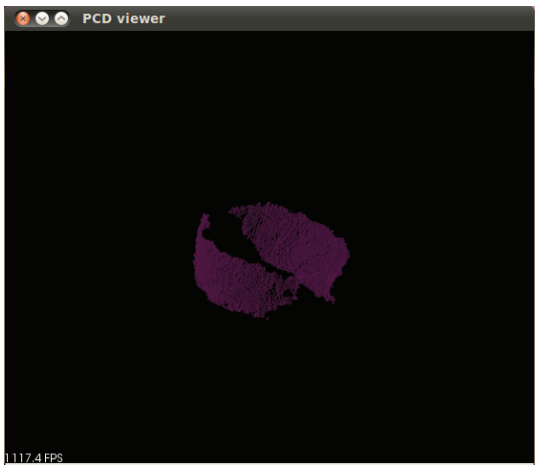

Figure 5.7 Detected point cloud of the third instance of the bowl class. The shape is similar to the mug one although the bowl is bigger.

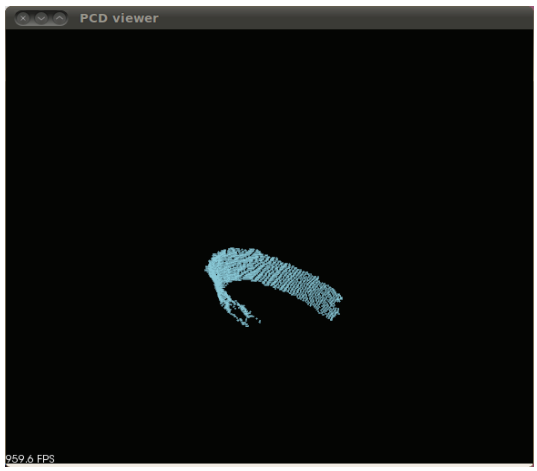

Figure 5.8 Detected point cloud of the first instance of the bowl class. The shape of the bowl is very flat so just one part of the bowl is detected.

5.1.2 AdaBoost One-Against-One test

In AdaBoost One-Against-One method, Table 5.3 represents the combinations of training sets depending on the number of instances for each class:

Training-set	Apple	Banana	Bowl	Mug	Plastic cup	Tobacco
A	8	8	8	8	8	8
B	16	16	16	16	16	16
C	16	24	16	16	16	16
D	16	32	16	16	16	16
E	16	40	16	16	16	16
F	16	16	16	24	16	16
G	16	16	16	32	16	16
H	8	16	16	16	8	16
I	16	16	16	16	8	16

Table 5.3 Different training-sets used for AdaBoost One-Against-One SVM named from A to I. Each cell contains the number of snapshots of a class (column) used to train the method with a training-set (row).

The training-sets used for AdaBoost One-Against-One are different from the ones used in SVM (Table 5.1). The idea is to try first with data-sets with eight and 16 instances per class, and after that, try to improve the results increasing the number of instances of class without changing the number of the others. In training-sets H and I, the number of apple and plastic cup snapshots is decreased to eight due to the number of pictures of those classes, respectively 32 and 24.
The following graphics represent the results of the different classes in AdaBoost One-Against-One using ROC method (Apple in Figure 5.9, Banana in Figure 5.10, Bowl in Figure 5.11, Mug in Figure 5.12, Plastic Cup in Figure 5.13 and Tobacco in Figure 5.14).

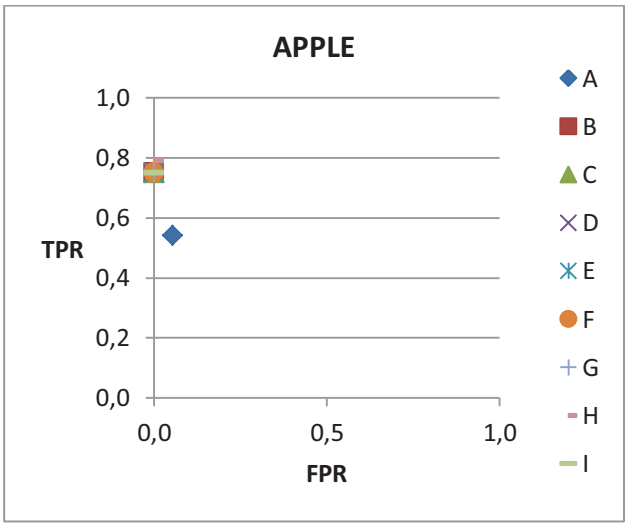

Figure 5.9 ROC representation for Apple class using AdaBoost One-Against-One

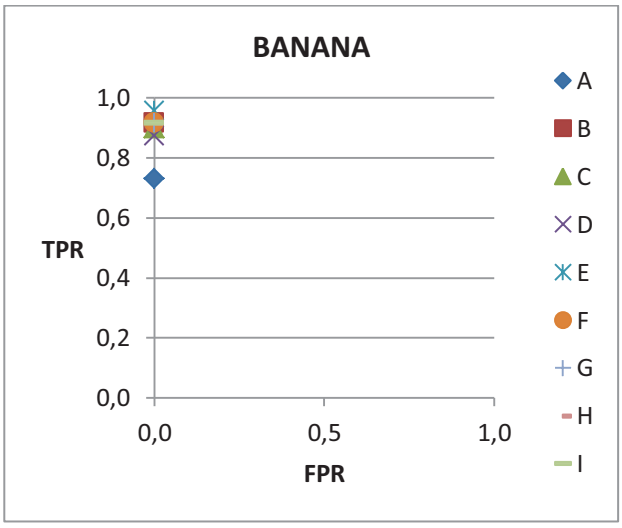

Figure 5.10 ROC representation for Banana class using AdaBoost One-Against-One

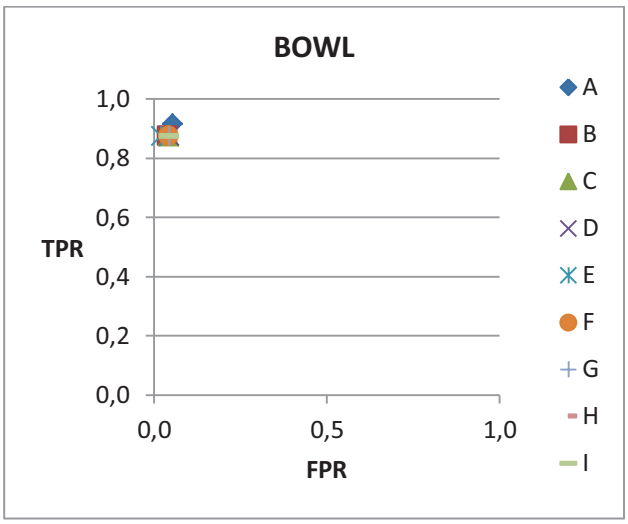

Figure 5.11 ROC representation for Bowl class using AdaBoost One-Against-One

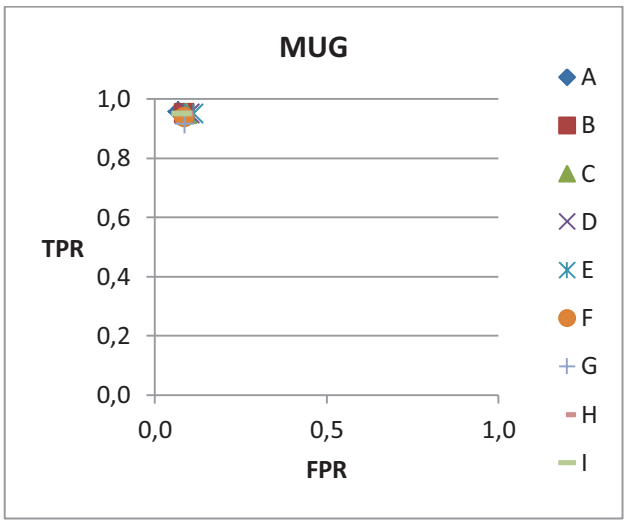

Figure 5.12 ROC representation for Mug class using AdaBoost One-Against-One

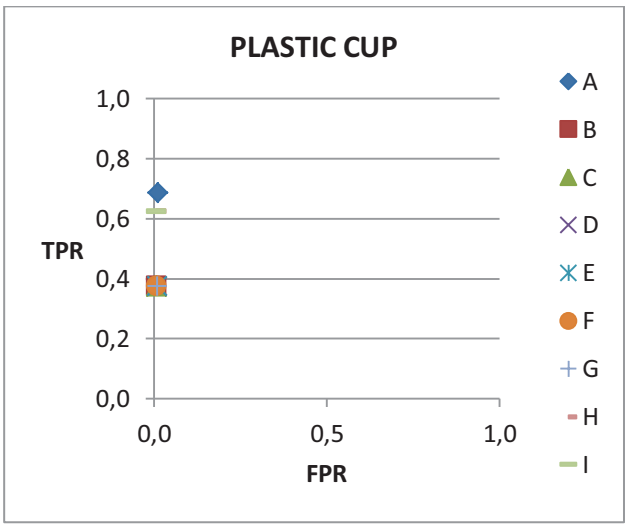

Figure 5.13 ROC representation for Plastic Cup class using AdaBoost One-Against-One

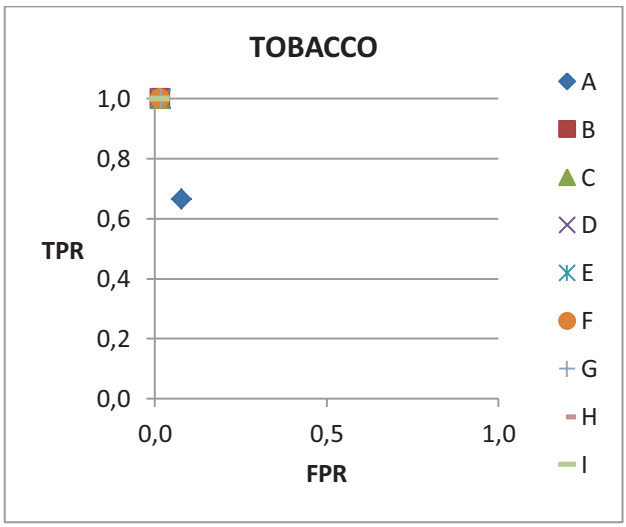

Figure 5.14 ROC representation for Tobacco class using AdaBoost One-Against-One

Hit rates of the classification with AdaBoost One-Against-One is presented in Table 5.4. Tobacco is always correctly classified. Plastic Cup results are better if the number of pictures to train the classifier is eight instead of 16. This means that these eight extra pictures (new ones from eight to 16) used in the training set do not lead to an improve of the results for Plastic Cup class.

Training-set	Apple	Banana	Bowl	Mug	Plastic cup	Tobacco	Overall
A	0,542	0,732	0,917	0,958	0,688	1,0	0,806
B	0,75	0,917	0,875	0,95	0,375	1,0	0,811
C	0,75	0,9	0,875	0,95	0,375	1,0	0,808
D	0,75	0,875	0,875	0,95	0,375	1,0	0,804
E	0,75	0,958	0,875	0,95	0,375	1,0	0,818
F	0,75	0,917	0,875	0,938	0,375	1,0	0,809
G	0,75	0,917	0,875	0,917	0,375	1,0	0,806
H	0,792	0,917	0,875	0,95	0,625	1,0	0,860
I	0,75	0,917	0,875	0,95	0,625	1,0	0,853

Table 5.4 Hit rates of the AdaBoost One-Against-One method depending on the training-set (first column) and the class (first row). The last column indicates the hit rate counting all the classes.

H set is the choice of the training-set for AdaBoost One-Against-One:

Training-set	Apple	Banana	Bowl	Mug	Plastic cup	Tobacco
H	8	16	16	16	8	16

5.1.3 AdaBoost One-Against-All test

As in Section 5.1.1 and Section 5.1.2, Table 5.5 represents the different training-sets used, in this case, in AdaBoost One-Against-All:

Training-set	Apple	Banana	Bowl	Mug	Plastic cup	Tobacco
A	8	8	8	8	8	8
B	16	16	16	16	16	16
C	16	24	16	16	16	16
D	16	32	16	16	16	16
E	16	40	16	16	16	16
F	16	16	16	24	16	16
G	16	16	16	32	16	16
H	8	16	16	16	8	16
I	16	16	16	16	8	16
J	16	40	16	32	16	16

Table 5.5 Different training-sets used for AdaBoost One-Against-All named from A to J. Each cell contains the number of snapshots of a class (column) used to train the method with a training-set (row).

A new training-set J has been added. This set is a mixture of the Banana instances of E and the Mug instances of G. Using it, it is checked if some improve in training phase is achieved.
The following graphics (Figure 5.15 for Apple, Figure 5.16 for Banana, Figure 5.17 for Bowl, Figure 5.18 for Bowl, Figure 5.19 for Plastic Cup and Figure 5.20 for Tobacco) represent the results of the different classes in AdaBoost One-Against-All using ROC method.

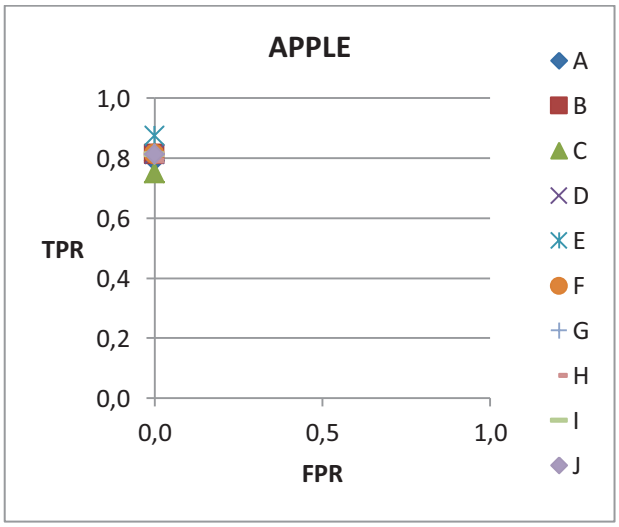

Figure 5.15 ROC representation for Apple class using AdaBoost One-Against-All

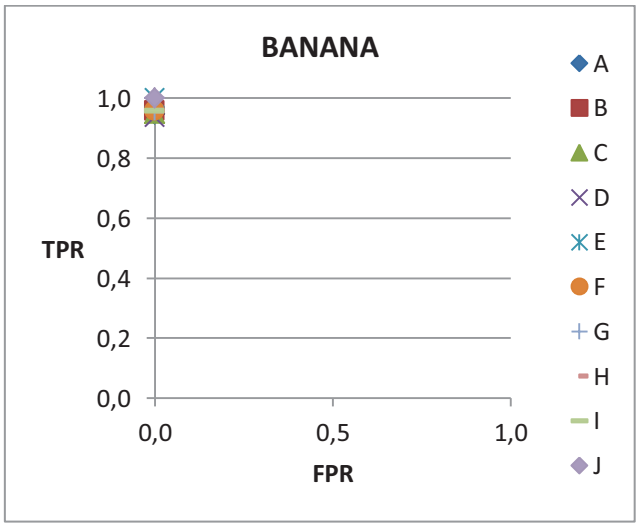

Figure 5.16 ROC representation for Banana class using AdaBoost One-Against-All

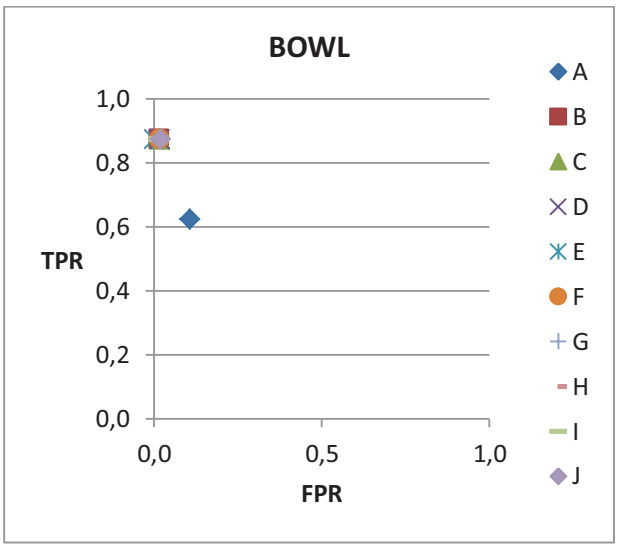

Figure 5.17 ROC representation for Bowl class using AdaBoost One-Against-All

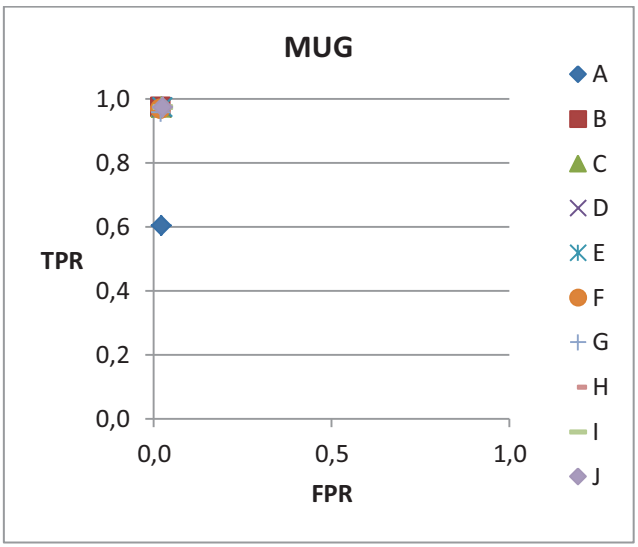

Figure 5.18 ROC representation for Mug class using AdaBoost One-Against-All

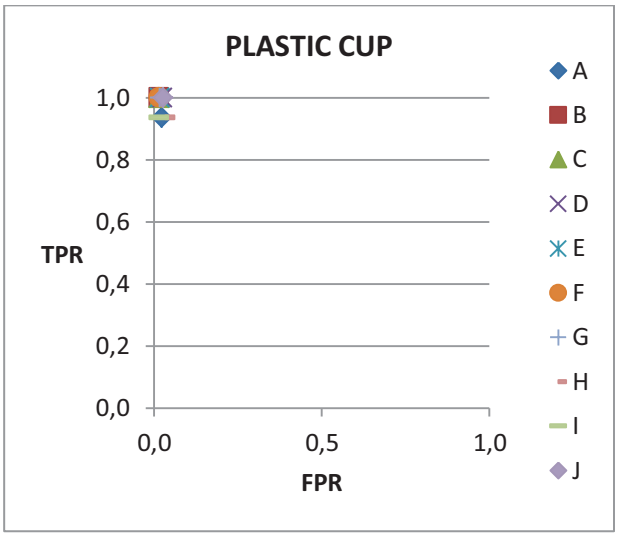

Figure 5.19 ROC representation for Plastic Cup class using AdaBoost One-Against-All

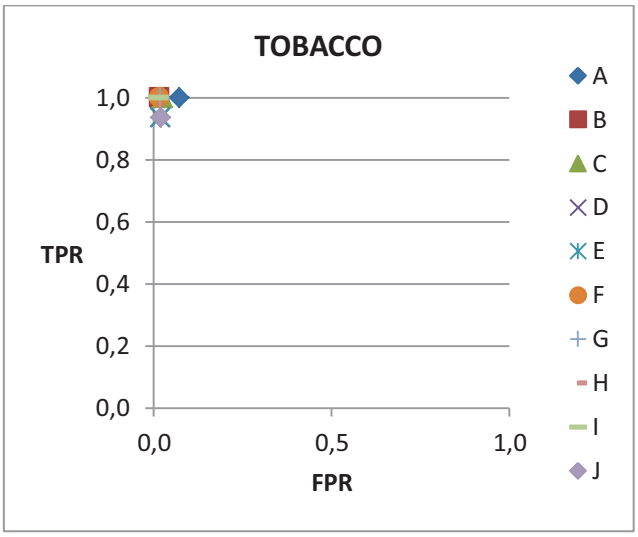

Figure 5.20 ROC representation for Tobacco class using AdaBoost One-Against-All

Table 5.6 presents the results for the different data-sets. Good recognition rates are obtained in almost all classes with all the sets, with the exception of, for example, Bowl and Mug classes that get lower rates for A set.

Training-set	Apple	Banana	Bowl	Mug	Plastic cup	Tobacco	Overall
A	0,792	0,946	0,625	0,604	0,938	1,0	0,817
B	0,813	0,958	0,875	0,975	1,0	1,0	0,937
C	0,75	0,95	0,875	0,975	1,0	1,0	0,925
D	0,813	0,938	0,875	0,975	1,0	0,938	0,923
E	0,875	1,0	0,875	0,975	1,0	0,938	0,944
F	0,813	0,958	0,875	0,969	1,0	1,0	0,936
G	0,813	0,958	0,875	0,958	1,0	1,0	0,934
H	0,792	0,958	0,875	0,975	0,938	0,938	0,913
I	0,813	0,958	0,875	0,975	0,938	1,0	0,926
J	0,813	1,0	0,875	0,975	1,0	0,938	0,933

Table 5.6 Hit rates of the AdaBoost One-Against-All method depending on the training-set (first column) and the class (first row). The last column indicates the hit rate counting all the classes.

The decision is to use E set for AdaBoost One-Against-All:

Training-set	Apple	Banana	Bowl	Mug	Plastic cup	Tobacco
E	16	40	16	16	16	16

5.2 Results

Once the number of instances for each test-set is chosen, the evaluation of the methods can be done. It has to be taken into account that the test is done using the test-set, so a classification of the sample will always be done. As it was explained in Section 4.2.2, some bananas will not be detected because of the size of the point cloud. In this case, instead of detecting a banana or any other class, the program will not count it as an object. This fact is not considered in the following analysis.
In order to start with the evaluation of the methods, first, ROC diagrams for each method are presented in order to show an idea of the capacity of the classifiers. ROC diagram of SVM is shown in Figure 5.21, of AdaBoost OaO in Figure 5.22 and of AdaBoost OaA in Figure 5.23.

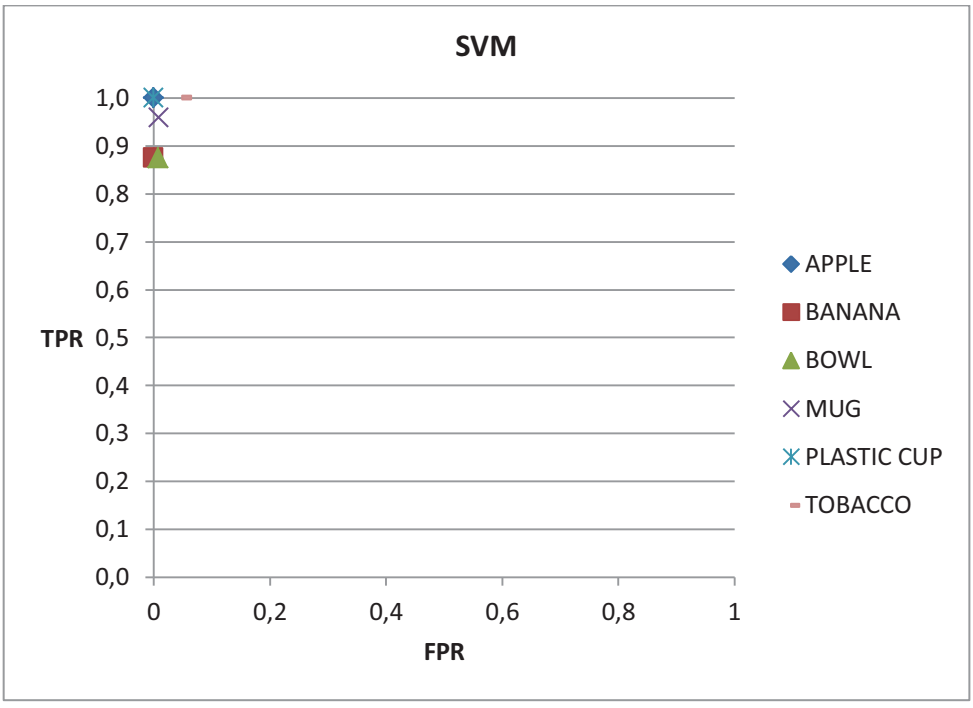

Figure 5.21 ROC representation for all classes using AdaBoost SVM

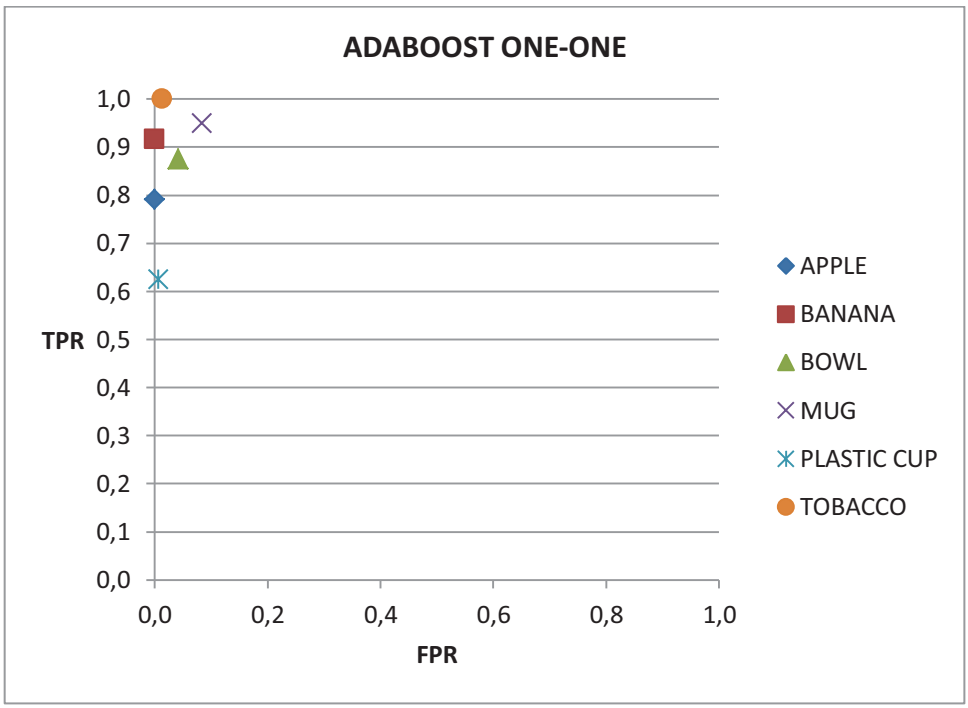

Figure 5.22 ROC representation for all classes using AdaBoost One-Against-One

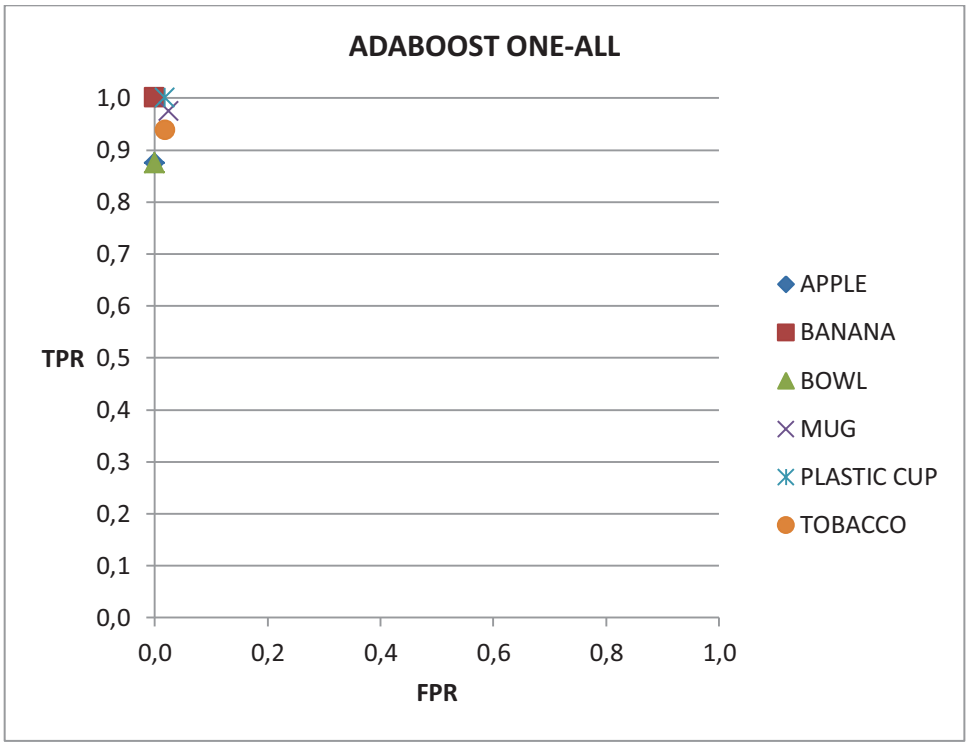

Figure 5.23 ROC representation for all classes using AdaBoost One-Against-All

To show more specific results, hit rates for each class-method, SVM, AdaBoost One-Against-One (OaO) and AdaBoost One-Against-All (OaO) and for the complete classifiers are shown in Table 5.7:

Method	Apple	Banana	Bowl	Mug	Plastic Cup	Tobacco	Overall
SVM	1,0	0,875	0,875	0,958	1,0	1,0	0,951
AdaBoost OaO	0,792	0,917	0,875	0,95	0,625	1,0	0,859
AdaBoost OaA	0,875	1,0	0,875	0,975	1,0	0,938	0,944

Table 5.7 Hit rates of the three methods depending on the class. Last column indicates the mean of hit rates of all the classes

In SVM, some bananas are recognized as a tobacco packet. This is caused because depending on the orientation of the sample, the point cloud can be similar as the tobacco one. It is reminded that VOSCH features extraction is an invariant-rotation method (Section 3.3.1.1). There is also a confusion between Bowl and Mug classes. Some bowls are recognized as mugs and vice versa. The reason is the similarity of the shapes of the object as described in Figure 5.7.
AdaBoost One-Against-One makes some misclassifications. Apples can be recognized as mugs or tobacco packets, because of the orientation, the dimensions are similar. The misclassification among bowls, mugs and plastic cup can also happen due to the reason of size. In bananas, the cause is different. The dimensions of a banana are different from the bowl one, but looking at the Figure 5.8 and Figure 4.7, it is shown that the point clouds that describe those objects are quite similar.
Concerning AdaBoost One-Against-All, the confusion in the recognition among the classes is almost the same as in AdaBoost One-Against-One method. The exception is that, instead of misclassifying some apples as mugs, this method can classify them as plastic cups.
The best method, talking about hit rate, is SVM, then AdaBoost One-Against-All and finally AdaBoost One-Against-One. Looking at the table, the decision of which classifier has to be used should be SVM, but more factors have to be considered. In the Section 5.3, these will be commented.

5.3 Recognition time

Since the first calculation of the programs is the classifier creation, the time that different methods need to build their classifiers is going to be presented. These times are calculated using the training-set calculated in section 4.2 for each method. Table 5.8 represents the time that each method needs to calculate its classifier in ms:

SVM	AdaBoost OaO	AdaBoostOaA
31,090 ms	2145,699 ms	3985,678 ms

Table 5.8 Mean of time that each method needs to be trained.

Since the training-set size is different for each method, it has been calculated how long it takes to each classifier to train a sample. Table 5.9 shows these results. Although the difference between AdaBoost methods is not important, the difference of times between SVM and AdaBoost methods is considerable:

SVM	AdaBoost OaO	AdaBoostOaA
0,299ms	26,821ms	33,214ms

Table 5.9 Mean time that each classifier needs to train one simple from the training-set.

Although in this thesis the training of the method is done when the real-time program is launched, a solution to avoid counting these times is to train the classifier before starting the program and save the values that describe the classifier in a file. This way, the programs would just have to load the

file in order to work with the classifier, avoiding its training. This is the main reason why this phase is not considered in the time analysis of the whole programs.

In real-time recognition it is not only important to obtain a good result but also to do it in a reasonable time. A trade-off between time and classification success has to exist. The following tables represent the time that SVM (Table 5.10), AdaBoost One-Against-One (Table 5.11) and AdaBoost One-Against-All (Table 5.12) need to classify an object. Each column of the tables represents a step of the recognition process and can vary from one method to another. *Range* column contains the time to eliminate the points which are not in the desired range. *RANSAC* and *plane* describe, respectively, the time in which the RANSAC algorithm needs to be calculated and the time needed to delete the points which belong to the plane calculated by RANSAC. *Labeler* is the time that the algorithm needs to label all the valid points of the point cloud. *Features* and *SVM* indicate the time needed to create the VOSCH features and to classify them. *Drawing* column shows the time the program needs to draw the points of an object. Although it is not necessary to count it because it is not an essential part of the recognition process, it is included in order to show the information as complete as possible. Finally, *AdaBoost* contains the time needed to calculate the easy features and classify them under Adaboost. Each row represents a class. The value of each cell represents the mean of times in the step of the corresponding column, using various samples of the class described by the row. Total column describes the total time needed to recognize the class designated by the first cell of the row. Last row of the tables corresponds to the mean of times of all classes in the step describe in that column.

	Range	RANSAC	Plane	Labeler	Features	SVM	Drawing	Total
Apple	11,316	1,971	8,956	45,770	465,577	3,583	0,138	537,311
Banana	10,424	1,413	14,290	43,606	164,496	3,780	0,073	238,083
Bowl	13,116	2,315	10,897	53,220	638,098	4,575	0,317	722,538
Mug	10,007	1,440	7,664	41,126	334,392	2,704	0,135	397,467
Plastic cup	10,118	1,125	7,624	45,208	230,162	3,285	0,102	297,624
Tobacco	11,157	1,179	7,849	41,610	211,523	2,685	0,091	276,093
	11,023	1,574	9,547	45,090	340,708	3,435	0,143	411,519

Table 5.10 SVM recognition times for every object depending on the step of the recognizing process. The mean of times for each class and for each step are described in the last column and in the last row respectively.

	Range	RANSAC	Plane	Labeler	AdaBoost	Drawing	Total
Apple	34,279	3,829	14,567	62,287	16,243	0,180	131,384
Banana	31,332	3,635	13,750	62,002	12,698	0,107	123,525
Bowl	34,052	5,793	14,020	74,938	21,115	0,436	150,354
Mug	29,018	4,506	12,716	59,601	15,056	0,382	121,278
Plastic cup	31,041	5,006	12,418	55,041	14,751	0,195	118,452
Tobacco	29,600	3,500	11,902	55,285	11,020	0,174	111,481
	31,554	4,378	13,229	61,526	15,147	0,246	126,079

Table 5.11 AdaBoost One-Against-One for every object depending on the step of the recognizing process. The mean of times for each class and for each step are described in the last column and in the last row respectively.

	Range	RANSAC	Plane	Labeler	AdaBoost	Drawing	Total
Apple	30,097	3,883	13,620	54,825	12,655	0,201	115,281
Banana	30,335	3,955	14,834	59,864	12,274	0,136	121,397
Bowl	31,284	5,905	12,377	56,709	16,405	0,476	123,155
Mug	31,306	5,313	12,925	60,974	15,383	0,463	126,363
Plastic cup	31,481	3,660	15,335	67,658	14,736	0,193	133,062
Tobacco	29,380	3,654	13,332	56,853	12,127	0,177	115,522
	30,647	4,395	13,737	59,480	13,930	0,274	122,463

Table 5.12 AdaBoost One-Against-All for every object depending on the step of the recognizing process. The mean of times for each class and for each step are described in the last column and in the last row respectively.

Table 5.10 shows a considerable difference of time in the feature extraction depending on which class is recognized. The reason why the times of attribute extraction in SVM goes from 164ms to 638ms is that VOSCH method depends on the shape of the object and the transition between the points [25]. This means, the more complicated the shape of an object is and the more points the point cloud has, the longer VOSCH algorithm requires to be calculated.
If more than one object is detected by the program, the time of feature creation and method application of each extra object have to be added to the total time of one recognition. For example, if two bowls are trying to be detected, the estimated time will be 722ms (mean of a bowl recognition) + 638ms (VOSCH features extraction of the second bowl) + 4,5ms (SVM classification of the second bowl) = 1,3 s.
Figure 5.24 helps to understand the time differences between the different methods. For SVM, the calculation of features and the SVM classification are described by *Method*. X-axis indicates the step of the classification and Y-axis the time in ms.

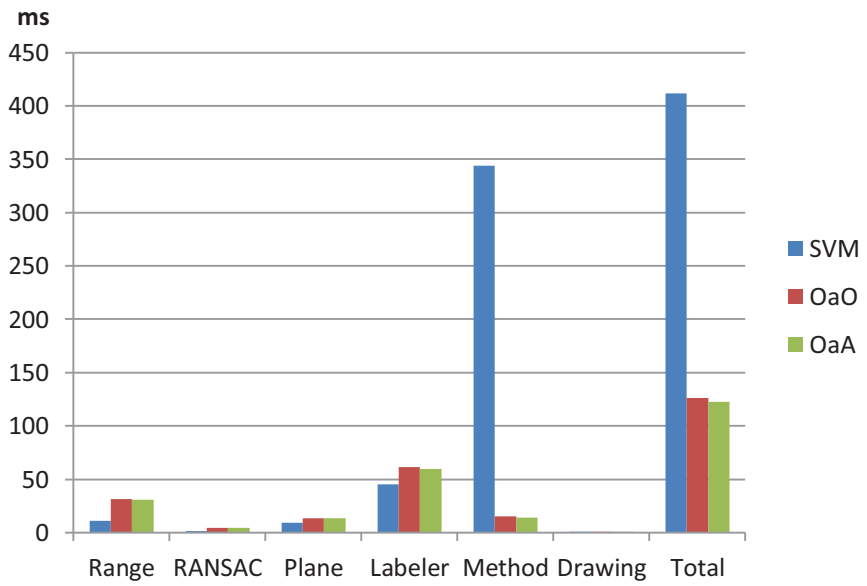

Figure 5.24 Each part of the recognizing process is described in X-axis. Method indicates the sum of the feature extraction and the method application and total shows the sum of times of the previous steps for each classification method.

Looking at the Table 5.11 and Table 5.12 a slight time difference between AdaBoost can be found. The reason is the number of classifiers the methods use. Being *n* the number of classes, in this case six, One-Against-All method will calculate a solution using *n* classifiers (one for each class) and One-Against-One $\binom{n}{k}$, where *k* refers to the number of elements of the One-One classifiers, in this case, two. Taking this into account, the methods will use six and 15 classifiers to get a classification result. Although, as explained in Section 4.7, some One-One classifiers are trivial and therefore the calculation time is short, the number of classifiers of AdaBoost One-Against-One is between 2,5 times bigger than the number of the One-Against-All method. This is the reason why One-Against-One takes longer to compute.

Looking at time information in Figure 5.24, SVM detection can process 1-3 images per second and AdaBoost methods 6-8. Considering that a mobile robot has to compute different tasks simultaneously, like planning or getting information from other sensors, the most suitable choice is the one that uses the shortest time to be computed. This way, the pattern recognition lets more time to the other tasks to be executed in the most precise manner.

According with the time results, the selected method should be AdaBoost in order to be able to identify the objects as soon as possible.

6 Conclusions and future work

As seen in the previous section, according to the classification or time results, SVM or AdaBoost method can be the best option for this subject.
Talking about classification results, AdaBoost One-Against-One can be discarded because it is far from the results which can be obtained using the other two methods. AdaBoost One-Against-All and SVM generate hit rates of about 95%, whereas One-Against-One method gets an 85% (Table 5.4).
Talking about time results, there is a huge difference between AdaBoost methods and SVM. The main problem for SVM is the time it needs to calculate the features. The easiest solution to this problem is to use a faster computer in order to do the calculations more quickly. A second solution could be to change the extracted characteristics. Instead of using VOSCH features, easier features could be chosen to avoid the computation time of that phase in the real-time program. Detecting several objects at once, the time needed becomes too extensive to be used in a real robot. Nevertheless, VOSCH features are a good solution working with non-real-time images or in real-time programs where a very fast detection is not required. For example, if a picture or a video already taken has to be analyzed in order to find some patterns in them. The difference between AdaBoost methods is slight, being the time for One-Against-One a little bit larger. This difference is caused by the number of classifiers due to the chosen strategy explained in Section 5.3.
The detection results obtained with AdaBoost One-Against-All are almost as good as the ones obtained with SVM and the time needed to compute those results is small enough to get a trade-off between hit rate and time. That is why, the favourite solution to this thesis is to use AdaBoost One-Against-All method in order to be able to include it in a real robot. The solution of using SVM cannot be discarded, if the computation of complex features does not take so long.
Regarding AdaBoost, the number of weak learners for the method is not very high, so it could be taken into account the possibility of creating new weak classifiers so that the final one can be more complex and new, possibly better, results can be obtained. This way, more calculations have to be done, but also better results could be obtained using more features discriminating the classes.
Another important part to comment on is the data-set. This data-set was created specifically for this thesis. Compared with other data-sets which can be found on the Internet, the data-set is smaller, but the objective of the thesis is not to create a huge set, it is only to use it to train and test the classifier. As desired the data-set has carried out that role.
Furthermore, some improvements can be made. A good idea is to expand the data-set increasing the number of instances of the classes and also the number of pictures for each everyday object (for example, taking 16 pictures instead, that is, adding more positions or turning the object every 22,75º). Another idea is to create a new data-set containing just a range of classes of a specific topic (for example fruits, containers or the elements of the Eurobot contest). The data-sets presented in Section 2.2.1 can be also used in order to do an intensive study of the methods. The amount of classes of objects and instances is huge, so the possibility of checking different algorithms and method using these sets can be considered.

7 References

[1] K. Lai, L. Bo, X. Ren and D. Fox, "A Large-Scale Hierarchical Multi-Viwe RGB-D Object Dataset".

[2] OpenCV community, "http://opencv.willowgarage.com/wiki/".

[3] F. Tombari and L. Di Stefano, "Object recognition in 3D scenes with occlusions and clutter by Hough voting," 2010.

[4] J. Shotton, A. Fitzgibbon, M. Cook, T. F. M. Sharp, R. Moore, A. Kipman and A. Blake, "Real-Time Human Pose Recognition in Parts from Single Depth Images".

[5] J. Ponce and A. Karahoca, State of the Art in Face Recognition, I-Tech Education and Publishing, 2009.

[6] A. K. Seewald, "On the Brittleness of Handwritten Digit Recognition Models," 2011.

[7] D. Filliat, E. Battesti, S. Bazeille, G. Deceux, A. Gepperth, L. Harrath, R. Pereira, A. Tapus, C. Meyer, S.-H. Ieng, R. Benosman, E. Cizeron, J.-C. Mamanna and B. Pothier, "RGBD object recognition and visual texture classification for indoor semantic mapping".

[8] School of Geology & Geophysics University of Oklahoma, "http://geology.ou.edu/aaspi/".

[9] M. N. Murty and V. S. Devi, Pattern Recognition An Algorithmic Approach, Springer, 2011.

[10] "http://www.vision.caltech.edu/Image_Data_sets/Caltech101/".

[11] "http://labelme.csail.mit.edu/index.html".

[12] "http://www.image-net.org".

[13] C. M. Bishop, Neuronal Networks for Pattern Recognition, Oxford University Press, 2010.

[14] V. Vapnik, The Nature of Statiscal Learning Theory, Springer, 2000.

[15] N. Cristianini and J. Shawe-Taylor, Support Vector Machines and other kernel-based learning methods, Cambridge University Press, 2000.

[16] B. Universität, "http://www.precision-crop-protection.uni-bonn.de".

[17] Y. Freund and R. E. Schapire, "A short introduction to boosting," *Journal of Japanese Society for Artificial Intelligence,* pp. 771-778, 1999.

[18] R. Kohavi and P. Foster, "Glosary of terms," in *Editorial for the Special Issue in Applications of Machine Learning and the Knowledge Discovery Process*, 1998.

[19] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognition Letters,* vol. 27, pp. 861-874, 2005.

[20] J. J. Rodríguez Diez, "Métodos y Tëcnicas de Minerïa de Datos. Metodología experimental".

[21] M. A. Fischler and R. C. Bolles, "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography," *Communications of the ACM,* vol. 24, no. 6, pp. 381-395, 1981.

[22] M. Greuter, M. Rosenfelder, M. Blaich and O. Bittel, "Obstacle and Game Element Detection with the 3D-Sensor Kinect," *International Conference on Research and Education in Robotics,* vol. 161, 2011.

[23] F. Escolano, P. Suau and B. Bonev, Information Theory in computer Vision and Pattern Recognition, Springer, 2009.

[24] R. Bellman, Adaptive Control Processes: A Guided Tour, New Jersey: Adaptive Princeton Press, 1961.

[25] A. Kanezaki, Z. Marton, D. Pangercic, T. Harada, Y. Kuniyoshi and M. Beetz, "Voxelized Shape and Color Histograms for RGB-D".

[26] A. Kanezaki, T. Suzuki, T. Harada and Y. Kuniyoshi, "Fast Object Dtection for Robots in a Clutteres Indoor Enviroment Using Integral 3D Feature Table," in *Proc. IEEE ICRA*, 2011.

[27] Z.-C. Marton, D. Pangercic, R. B. Rusu, A. Holzbach and M. Beetz, "Hierarchical Object

Geometric Categorization and Appearance Classification for Mobile Manipulation," in *Proc. IEEE Int. Conf. on Humanoid Robots*, 2010.

[28] H. Bunke, "Structural an Syntactic Patter Recognition," in *Handbook of pattern recognition & computer vision*, World Scientific, 1993, pp. 163-209.

[29] "http://www.mrpt.org".

[30] "http://www.ros.org/wiki/".

[31] PCL developers, "http://pointclouds.org/".

[32] OpenKinect members, "http://openkinect.org".

[33] C.-W. Hsu, C.-C. Chang and C.-J. Lin, "A Pratical Guide to Support Vector Classification," 2010.

[34] G. M. Foody and A. Mathur, "A Relative Evaluation of Multiclass Image Classification by Support Verctor Machines," *IEEE Transactions on Geoscience and Remote Sensing,* vol. 42, no. 6, pp. 1335-1343, 2004.