UNIVERSIDAD DE GRANADA

TRABAJO FIN DE GRADO

INGENIERÍA EN INFORMÁTICA

Implementación de tareas de manipulación precisa con brazos robóticos

Autor
Cristina María Garrido López

Directores
Eduardo Ros Vidal
Francisco Barranco Expósito

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, a 18 de Junio de 2018

*

Implementación de tareas de manipulación precisa con brazos robóticos

Cristina María Garrido López

Palabras clave: robótica, clasificación, clustering, k-medias, planificación de trayectorias, filtrado por color.

Resumen

Este proyecto final de grado presenta el desarrollo de tareas de manipulación de objetos y evaluación de prestaciones, así como análisis de eficiencia de cada algoritmo programado. Se utiliza el robot Baxter de Rethink Robotics y el sistema operativo de robots (ROS).

Se fija como objetivo principal la automatización de tareas de clasificación de objetos en distintos entornos utilizando uno de los brazos robóticos de Baxter para realizar los procesos de visión y control. Posteriormente, se realizarán comparaciones con distinto tamaño y número de objetos en los diferentes entornos.

Se explicarán las funcionalidades del framework ROS, MoveIt! y los componentes hardware de Baxter, así como los métodos utilizados de visión por computador.

Puesto que es difícil encontrar código abierto en robótica, los algoritmos desarrollados en este proyecto serán liberados de forma que puedan ser utilizados por otros investigadores, esperando que así sea más útil para la comunidad.

Precise handling tasks implementation with robotic arms

Cristina María Garrido López

Keywords: Robotics, classification, clustering, k-means, trajectory planning, colour filter.

Abstract

This Bachelor thesis presents the development of various manipulation tasks and performance evaluation, where the Baxter Robot from Rethink Robotics and the Robot Operating System (ROS) are used.

The main objective is to automate object classification tasks in different environments, using one of Baxter's two robotic arms to carry out computer vision tasks and operational level control. Later, tests will be made with different size and number of objets in the different environments.

ROS functionalities, MoveIt! and Baxter's hardware components will be explained, as well as the computer vision methods.

Since it is difficult to find open source in the field of robotics, the developed algorithms in this thesis will be released so that they can be used by other researchers, hoping it is useful for the comunity.

*

Yo, **Cristina María Garrido López**, alumna de la titulación Ingeniería en Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI *, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Cristina María Garrido López

Granada a 17 de Junio de 2018.

*

D. **Eduardo Ros Vidal**, Profesor del Área de Arquitectura y Tecnología de Computadores del Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Granada.

D. **Francisco Barranco Expósito**, Profesor del Área de Arquitectura y Tecnología de Computadores del Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado ***Implementación de tareas de manipulación precisa con brazos robóticos***, ha sido realizado bajo su supervisión por **Cristina María Garrido López**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 18 de Junio de 2018.

Los directores:

Eduardo Ros Vidal **Francisco Barranco Expósito**

Agradecimientos

Sobre todo, me gustaría agradecer todo el apoyo, tiempo invertido y consejo proporcionado por los tutores de este proyecto, Eduardo Ros y Francisco Barranco, que nunca han faltado cuando he necesitado su ayuda. Por supuesto, agradezco además el hecho de permitirme trabajar en el laboratorio del CITIC y aportarme los materiales necesarios para realizar mi proyecto.

También me gustaría expresar mi gratitud hacia mis amigos de la facultad, a mi familia y a mis amigas por todo el apoyo durante el curso.

Declaración de autoría y originalidad del TFG

Yo, Cristina María Garrido López, con DNI *, declaro que el presente documento ha sido realizado por mí y se basa en mi propio trabajo, a menos que se indique lo contrario. No se ha utilizado el trabajo de ninguna otra persona sin el debido reconocimiento. Todas las referencias han sido citadas y todas las fuentes de información y conjuntos de datos han sido específicamente reconocidos.

Fdo: Cristina María Garrido López

Granada a 17 de Junio de 2018.

Índice

1. Introducción **13**
1.1. Motivación 14
1.2. Situación inicial 15
1.3. Objetivos 15
1.4. Procedimiento 15
1.5. Historia de la robótica 16
1.5.1. Actualidad 18

2. Entorno operacional **20**
2.1. ROS 20
2.1.1. Conceptos 20
2.2. Baxter 21
2.2.1. Introducción 21
2.2.2. Componentes hardware 22
2.3. MoveIt! y RViz 23
2.4. Modelos de robot 24
2.4.1. Formato universal de descripción robótica 24
2.4.2. Formato semántico de descripción robótica 24

3. Gestión del proyecto **25**
3.1. Metodología del software 25
3.2. Planificación 25
3.3. Presupuesto 27
3.3.1. Recursos hardware y software 27
3.3.2. Recursos humanos 28
3.3.3. Coste total 28
3.4. Gestión de problemas 28

4. Análisis y diseño del sistema **31**
4.1. Restricciones del sistema 32
4.1.1. Restricciones hardware 32
4.1.2. Restricciones software 32
4.2. Especificación de casos de uso 33
4.2.1. Entorno estático 34
4.2.2. Entorno dinámico 36
4.3. Especificación de requisitos 37
4.3.1. Requisitos funcionales 38
4.3.2. Requisitos no funcionales 39

5. Diseño **41**
5.1. Módulo de reconocimiento 42
5.2. Módulo de control de Baxter 42

6. Implementación **44**
6.1. Esquema general del escenario 45
6.2. Visión por computador 46
6.2.1. Reconocimiento de objetos 46
6.2.2. Procesamiento de imágenes 48
6.3. Procedimientos de control 49
6.3.1. *Pick and place* adaptado 49
6.3.2. Clasificación en entorno estático 50
6.3.3. Clasificación en entorno dinámico con cinta transportadora 51

7. Pruebas **54**
7.1. Introducción 54
7.2. Medidas de rendimiento 54
7.2.1. Precisión frente a velocidad en clasificación dinámica . 54
7.2.2. Precisión y exhaustividad 55
7.2.3. Robótica adaptativa 58
7.3. Medidas de error 59
7.3.1. Clasificación estática 59
7.3.2. Clasificación dinámica 61

8. Conclusión **64**
8.1. Competencias adquiridas 64
8.2. Trabajos futuros 65

9. Anexo. Algoritmos de clustering **66**
9.1. Algoritmo K-means 66
9.1.1. Cómo funciona 66
9.1.2. Por qué elegir K-means 66

Índice de figuras

1. Brazo robótico aislado 13
2. Gráfica que muestra el crecimiento del mercado de robots industriales [1]. El eje horizontal representa el año y el eje vertical miles de millones de dólares 14
3. Año 1738. Autómata con forma de pato de Jacques de Vaucanson. 17
4. Año 1956. Robot industrial Unimate. 17
5. Baxter trabajando cerca de una persona. 18
6. Robot Baxter. 21
7. Articulaciones de Baxter. 23
8. Diagrama de Gantt. 26
9. Posición inicial de Baxter. 31
10. Diagrama de casos de uso. 33
11. Diagrama de flujo. 41
12. Esquema general del escenario. 45
13. Captura que muestra el reconocimiento de los objetos y la definición de sus perímetros. 47
14. Imagen que muestra la configuración de objetos de la imagen anterior 6.2.1 desde fuera. 47
15. Capturas de la ejecución de *pick and place* adaptado. 49
16. Paleta de cartón. 50
17. Capturas de la clasificación en entorno estático. 50
18. Capturas de la ejecución de la segmentación con paleta en entorno dinámico. 51
19. Cuña de cartón. 52
20. Capturas de la ejecución de la segmentación con cuña en entorno dinámico. 53
21. Gráfica que muestra la velocidad frente a la eficiencia en clasificación dinámica con cuña donde la línea sigue un ajuste polinómico de grado 6. 55
22. Ejemplo de precisión y exhaustividad en el que se ven los objetos que toma como verdaderos positivos (a la izquierda) y los que toma como negativos (a la derecha). 56
23. Gráfica que muestra la exhaustividad frente a la precisión en clasificación estática 57
24. Gráfica que muestra el número de objetos grandes frente a la precisión en clasificación estática, representando la media con una columna y la desviación típica con una barra. 60
25. Gráfica que muestra el número de objetos pequeños frente a la precisión en clasificación estática, representando la media con una columna y la desviación típica con una barra. 61

26. Gráfica que muestra el número de objetos grandes frente a la precisión en clasificación dinámica con cuña, representando la media con una columna y la desviación típica con una barra. 62
27. Gráfica que muestra el número de objetos pequeños frente a la precisión en clasificación dinámica con cuña, representando la media con una columna y la desviación típica con una barra. 63

Índice de cuadros

1. Planificación de proyecto. 27
2. Tabla de costes. 28
3. Coste total del proyecto 28
4. Caso de uso CU-001. 34
5. Caso de uso CU-002. 34
6. Caso de uso CU-003. 35
7. Caso de uso CU-004. 35
8. Caso de uso CU-005. 35
9. Caso de uso CU-006. 36
10. Caso de uso CU-007. 36
11. Caso de uso CU-008 37
12. Caso de uso CU-009. 37
13. Requisito funcional RF-001. 38
14. Requisito funcional RF-002. 38
15. Requisito funcional RF-003. 38
16. Requisito funcional RF-004. 38
17. Requisito funcional RF-005. 39
18. Requisito funcional RF-006. 39
19. Requisito funcional RF-007. 39
20. Requisito no funcional RNF-001. 39
21. Requisito no funcional RNF-002. 40
22. Tabla que muestra la precisión que se le indica al programa en cada ejecución y la precisión obtenida realmente. 59

1. Introducción

Para realizar este trabajo se ha utilizado a Baxter, un robot desarrollado por Rethink Robotics, como robot de investigación, encargado de realizar las diferentes tareas mostradas a lo largo del documento.

La automatización de procesos es cada vez más frecuente en la industria, es por ello que este tipo de robots seguros, que no necesitan de una infraestructura para aislarse de las personas, se utilizan. Además de la posibilidad de trabajar colaborando con un humano, son fáciles de programar, de desplazar y tienen un bajo coste comparado con el coste usual de los robots industriales.
Estos robots se pueden programar a través del entorno ROS, del cual se puede encontrar bastante documentación en internet, además de tener una gran comunidad, útil para consultar o solventar cualquier duda existente.

Figura 1: Brazo robótico aislado

Como se ve en la siguiente imagen (obtenida de [1]), se espera que el mercado de robots industriales crezca considerablemente en los próximos años. Algunas fuentes indican que este crecimiento puede llegar a ser exponencial, llegando a aumentar de 16850 millones de dólares en 2017 a 48170 millones en 2025 [2].

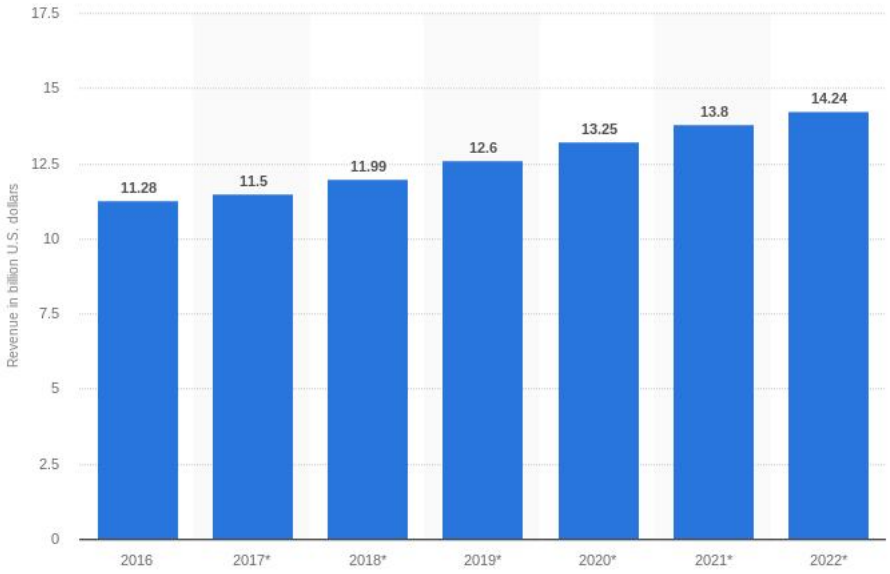

Figura 2: Gráfica que muestra el crecimiento del mercado de robots industriales [1]. El eje horizontal representa el año y el eje vertical miles de millones de dólares

En este documento se explicarán las plataformas software y hardware utilizadas para realizar las tareas de clasificación de objetos con Baxter, en entorno estático y dinámico, además de realizar una comparativa de eficiencia con distintos parámetros.

1.1. Motivación

A día de hoy, en la industria existen muchos trabajos realizados por personas, a pesar de que pueden resultar peligrosos o dañinos para ellas [3]. Debido a esto, la automatización de estas tareas resulta interesante, siendo realizadas por robots industriales que, en muchas ocasiones, suelen superar los 100000 euros.

En este proyecto se ha utilizado a Baxter, un robot desarrollado por Rethink Robotics, menos costoso y más seguro [4], simulando una cadena de producción en la que Baxter se encarga de clasificar los objetos por color y posición, empleando el framework ROS (Robotic Operating System), MoveIt! y RViz.

1.2. Situación inicial

Como ya se ha dicho, en este proyecto se pretende realizar una clasificación de objetos en entorno estático y dinámico. Se dispone de una cinta transportadora que se utilizará en las pruebas de entorno dinámico, del robot Baxter y de las plataformas MoveIt! y RViz. Para el desarrollo de los algoritmos, se parte de un filtro de color [5], al que se le añadirán las funcionalidades de segmentación por color y posición, planificación y ejecución de trayectorias y se realizarán distintos programas encargados de controlar a Baxter para realizar la clasificación.

1.3. Objetivos

En este trabajo se realiza un estudio de las prestaciones de Baxter en tareas que incluyen clasificación de objetos y visión por computador. Para ello han sido fijados los siguientes objetivos:

- Estudio de las funcionalidades y servicios del framework ROS.
- Integración de un filtro de color que permita realizar la clasificación basada en esta característica, y cálculo de grupos de objetos segmentados y recta que los separa.
- Desarrollo de diversos programas de clasificación de objetos y *pick and place*.
- Clasificación de objetos según un parámetro que especifica el porcentaje de objetos de un color a obtener.
- Robótica adaptativa. Con la misma configuración de objetos, el esquema de control adaptará un parámetro a partir del grado de éxito de la ejecución anterior en entorno estático, lo que proporcionará cierto grado de adaptación y optimización. Este parámetro modificará el valor del brazo en el eje de ordenadas.
- Realización de pruebas, estudio de prestaciones y barras de error de las tareas realizadas.
- Liberación del software desarrollado y documentación alojados en *GitHub*. Creación de una web con el contenido para que pueda ser utilizado por otros investigadores.

1.4. Procedimiento

Estos objetivos se pueden englobar en dos módulos, **módulo de reconocimiento**, encargado de realizar la segmentación por color y posición y enviar

los datos obtenidos al **módulo de control**, cuya función será controlar a Baxter para realizar los movimientos definidos por la planificación de trayectorias.
Así pues, puntualizamos varias funcionalidades:

- Identificación de objetos.
- Segmentación en dos grupos por color y posición.
- Obtención del punto medio entre ambos grupos de objetos.
- Planificación y ejecución de trayectorias.

Tras cumplir estas competencias, cada algoritmo procederá como le corresponda según tipo de entorno.

1.5. Historia de la robótica

El comienzo de la historia de la robótica data del 250 a. C. [6] En esta fecha Ctesibius de Alejandría, un físico e inventor griego, ideó una de las primeras aproximaciones de lo que hoy conocemos como autómata: el reloj de agua o *clepsydra*, que medía el tiempo según el flujo del agua hacia/desde un recipiente graduado.

Aunque se inventaron muchos más autómatas en aquellas épocas tan tempranas, casi ninguno de ellos se conserva en la actualidad, a diferencia del Gallo de Estrasburgo, del año 1352, que formaba parte del reloj de la catedral de Estrasburgo y movía las alas y el pico al dar las horas, que hoy se sigue conservando en esta catedral.

Durante los siglos XVII y XVIII, los autómatas que se construían seguían teniendo el mero objetivo de entretener o divertir a la gente y frecuentemente eran elaborados por relojeros.
Jacques de Vaucanson en el año 1738 inventó un autómata con forma de pato capaz de comer, excretar y moverse (3). Más adelante comenzaron a construirse autómatas que podían servir el té, disparar con arco, etc.

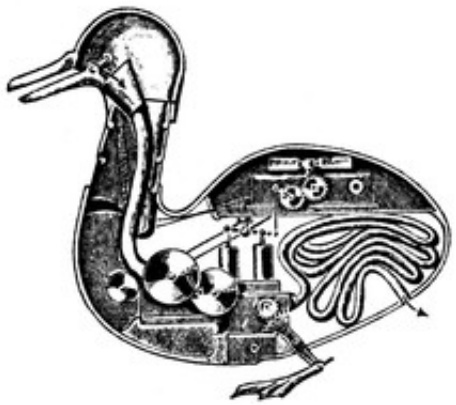

Figura 3: Año 1738. Autómata con forma de pato de Jacques de Vaucanson.

Con la Revolución Industrial y el desarrollo de la tecnología comenzaron a crearse robots más complejos con motores y energía eléctrica, con formas humanoides. En el año 1917, Karel Čapek publicó Rossum's Universal Robots, que dio lugar al término *robot*; además, Isaac Asimov, escribió un libro, "Runaround" ("Círculo vicioso"), en 1942, en el que aparecía por primera vez la palabra *robótica* y se establecían las leyes de los robots.

Joe Engleberger junto con George Devol, lanzaron al mercado el primer robot que se comercializaría (4), con lo que Joe Engleberger comenzó a ser considerado el padre de la robótica, ya que creó la primera empresa dedicada a este campo, *Unimation* con robots preparados para la industria en los años 1950 y 1960.

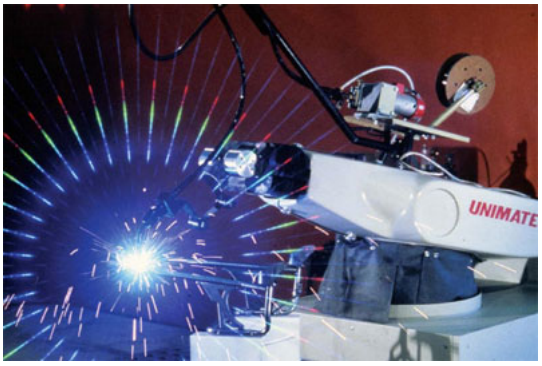

Figura 4: Año 1956. Robot industrial Unimate.

Siguieron construyéndose robots utilizados en distintas ramas, como la industria, el espacio o destinados a la interacción con el ser humano.
Entre algunos impulsores de la robótica, se encuentra Rodney Brooks [7], el director y presidente de Rethink Robotics; fue un estudiante de matemáticas que recibió un Ph.D. en informática y que más tarde ejerció de profesor de robótica. Fundó el laboratorio de Ciencias de la Computación e Inteligencia Artificial y cofundó iRobot. En 2008 fundó la empresa Rethink Robotics.

1.5.1. Actualidad

La evolución del desarrollo de robots ha permitido que estos trabajen en muchas áreas, como la industria, permitiendo una más rápida adaptación al cambio, así como un trabajo más preciso y con menos errores. Es frecuente a día de hoy decir que los robots generan desempleo, sin embargo, no existen para sustituir a una persona, sino para realizar aquellos trabajos en los que la repetición de procesos y la necesaria cualidad de que se hagan de forma rápida abundan. Cabe decir que muchos de estos trabajos suelen ser incluso peligrosos para las personas [8], ya que es probable que con la reiteración que supone el realizar estas tareas, el humano se equivoque, cuando un robot no lo hace [9].
Puesto que la robótica está obteniendo una gran presencia en la actualidad, es necesario que la sociedad se adapte a este nuevo paradigma, generando nueva formación y empleo de más valor añadido, como por ejemplo ingenieros industriales, ingenieros de automatización, programadores, etc. Además, conforme este proceso se haga más complejo, se hará necesaria la supervisión continuada por parte de especialistas, técnicos, etc.

Figura 5: Baxter trabajando cerca de una persona.

En cuanto a Baxter, es un robot con dos brazos robóticos que, a parte de ser un robot seguro [10] y que puede trabajar con personas como se verá en próximas secciones, no es tan caro. Según RobotWorx [11], una compañía

centrada en robótica de alta calidad a bajo precio, un robot industrial nuevo consistente en un sólo brazo robótico puede llegar a costar entre 50000 y 80000 dólares, sin incluir periféricos de aplicación específicos, con los que el precio puede aumentar hasta los 150000 dólares. En cambio, Baxter cuesta 35000 dólares incluyendo un año de garantía y un año de actualizaciones software, además de las pinzas y la bomba de vacío.

Estos robots industriales en principio tienen este precio, pero después es necesario añadir el coste de las infraestructuras que los rodeen para mantener a los humanos seguros. Generalmente son más difíciles de programar mientras que Baxter no requiere tanta experiencia y puede programarse para realizar varias tareas rápidamente.

2. Entorno operacional

En este apartado se explicará el software utilizado así como el robot Baxter y sus componentes. Se comenzará con ROS, el middleware que permite integrar las tareas de control, y a continuación se hablará de MoveIt! y RViz y las plataformas utilizadas para realizar la manipulación y visualización del modelo y del movimiento.

2.1. ROS

Robotic Operating System [12] [13] es un framework libre orientado a desarrollo de aplicaciones para robots. Actúa como un sistema operativo incluyendo abstracción de hardware, transferencia de mensajes entre procesos, administración de paquetes, etc. Para el desarrollo de software cuenta con librerías y herramientas que permiten construir, ejecutar y escribir código en cualquier lenguaje de programación.
Cada versión de ROS está montada sobre un sistema basado en Unix, aunque actualmente se está integrando en otros sistemas como Windows.

2.1.1. Conceptos

En ROS existen una serie de conceptos básicos a integrar:

- Paquetes. Son la unidad básica de organización de ROS y contienen los nodos, librerías y ficheros de configuración como CMakeLists.txt y package.xml. Se pueden construir a partir de la orden catkin_create_package, que crea automáticamente los archivos de configuración.
- Nodos. Son los procesos de ROS, que pueden actuar como *publishers* o *subscribers* para enviar o recibir datos entre ellos.
- Mensajes. Son las estructuras de datos con las que se comunican los nodos.
- Tópicos. Son la vía de transporte de los mensajes, donde estos se publican.
- Servicios. Son otra forma de comunicación de los nodos vía petición/-respuesta.
- Comandos. Son una serie de órdenes para navegar por el sistema de ficheros y modificar o mostrar tópicos y mensajes.

2.2. Baxter

Figura 6: Robot Baxter.

2.2.1. Introducción

Baxter [14] es un robot de automatización de procesos, desarrollado por Rethink Robotics y a la venta en el año 2012. Presenta siete grados de libertad en cada brazo y tres cámaras que se pueden utilizar o integrar en sus dos versiones disponibles: Baxter para fabricación, que incluye un entorno gráfico de fácil acceso al usuario que le permite ser programado de forma rápida y sencilla, y Baxter para investigación, para el que es necesario constar de una estación de trabajo con un entorno SDK de ROS.

Dos de las tres cámaras a color están situadas en los brazos y la tercera en la cabeza, donde además presenta un conjunto de sensores de ultrasonidos. Aunque consta de tres cámaras [15], sólo se pueden utilizar dos a la vez debido a las limitaciones de los USB en un entorno 64-bit.
Baxter es un robot seguro, cuenta con un botón de emergencia que deshabilita automáticamente la energía que le llega a sus componentes, de forma que cesa el proceso actual. Además consta de mecanismos de prevención de colisiones consigo mismo y, gracias a los sensores de ultrasonidos, con personas. [16]

2.2.2. Componentes hardware

Como se ha mencionado antes, cada brazo robótico [17] tiene siete grados de libertad, lo que le permite realizar casi todos los movimientos que puede realizar un brazo humano, aunque este último conste de unos 26 grados de libertad.

Se utilizan actuadores elásticos que consisten en introducir un resorte entre el motor y los engranajes de las articulaciones de Baxter, proporcionando más seguridad, menos ruido de control y más estabilidad.

En cada brazo hay sensores infrarrojos, un acelerómetro, botones para navegar en la interfaz y, como se ha dicho anteriormente, cámaras.

Para nombrar cada una de las articulaciones se utilizan varias letras: S para los "hombros" (*shoulder*), E para los "codos" (*elbow*) y W para las "muñecas" (*wrist*), con un número que indicará el ángulo de navegación [18], un tipo de ángulo de Euler que se describe a continuación.

- Dirección o *yaw*. Rotación contraria a las agujas del reloj en el eje Z.
- Elevación o *pitch*. Rotación contraria a las agujas del reloj en el eje Y.
- Ángulo de alabeo o *roll*. Rotación contraria a las agujas del reloj en el eje X.

En la figura (7) vemos cómo se denomina cada articulación de los brazos de Baxter.

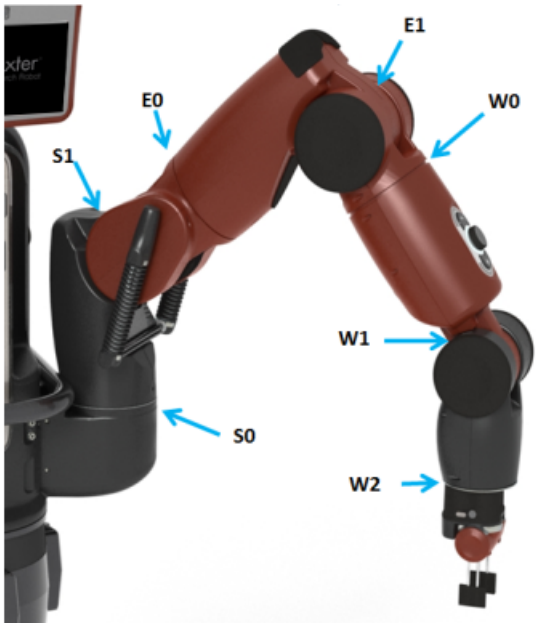

Figura 7: Articulaciones de Baxter.

S0 - Shoulder Roll, S1 - Shoulder Pitch, E0 - Elbow Roll, E1 - Elbow Pitch, W0 - Wrist Roll, W1 - Wrist Pitch, W2 - Wrist Roll.

Existen dos tipos de pinzas o *grippers* que se pueden colocar en los brazos de Baxter: los eléctricos y los aspiradores. Los eléctricos son dos palas paralelas que pueden levantar hasta 2 kilos aproximadamente; en cuanto a los aspiradores, se utilizan para coger objetos succionando, para lo que es necesario conectar un suministro de aire externo.

2.3. MoveIt! y RViz

MoveIt! [19] es una plataforma de código abierto montada sobre ROS enfocada a manipulación de robots, control, cinemáticas, detección de colisiones y navegación. En este proyecto se ha utilizado para la definición de trayectorias, planificación y ejecución de las mismas, así como la detección de colisiones.
Para utilizar este software es necesario proporcionar el archivo URDF (es decir, el formato de modelos de robots de ROS) del robot utilizado, y MoveIt! se encarga de generar todos los archivos para comenzar a trabajar.
MoveIt! utiliza verificadores de colisión como FCL, una librería que detecta solapamientos entre modelos [20].

El plugin RViz [21] es una GUI de ROS (ROS Visualizer) que sirve para

visualizar el modelo de robot, configurar la planificación de movimientos, evitar colisiones, además de consultar valores de los tópicos de ROS como los de las cámaras, sensores infrarrojos, etc. Normalmente se utiliza como librería de planificación la librería OMPL, de código abierto y alta calidad de planificación aleatoria.

2.4. Modelos de robot

2.4.1. Formato universal de descripción robótica

Este archivo URDF (*Universal Robotic Description Format*), contiene la representación en formato XML del modelo de robot en estructura de árbol [22], que, en caso de Baxter, se genera dinámicamente al iniciarlo, y actualizado cuando se conectan o desconectan los *grippers* o pinzas o cuando se coge o suelta un objeto [23]. Contiene información sobre la descripción cinemática o dinámica del robot, representación visual, modelo de colisión y enlaces entre las articulaciones [24] [p. 60-61].

2.4.2. Formato semántico de descripción robótica

El archivo SRDF (*Semantic Robot Description Format*) se genera dinámicamente al iniciar MoveIt! y complementa al archivo URDF. Contiene los parámetros necesarios para establecer los límites de las articulaciones de los brazos y articulaciones virtuales, información sobre cinemática y controladores y verificadores de colisión adicionales [24] [p. 122].

3. Gestión del proyecto

En este apartado se expone la planificación seguida en el proyecto, dividido en las tareas que lo componen, mostrando algunas dificultades, problemas y tiempo invertido en cada una de ellas, así como el presupuesto estimado del trabajo realizado.

3.1. Metodología del software

La metodología utilizada en este proyecto ha sido el desarrollo en espiral, en el que cada iteración representa un conjunto de actividades y la siguiente se elige en función del análisis de riesgo de las anteriores. De esta forma, este modelo itera repetidamente mejorando cualquier tipo de conflictos en la implementación, reduciéndolos conforme se completan las iteraciones [25].

3.2. Planificación

Para organizar la planificación seguida en este proyecto, se muestra el diagrama de Gantt (figura 3.2) desde el 07/03/2018 hasta el 06/06/2018:

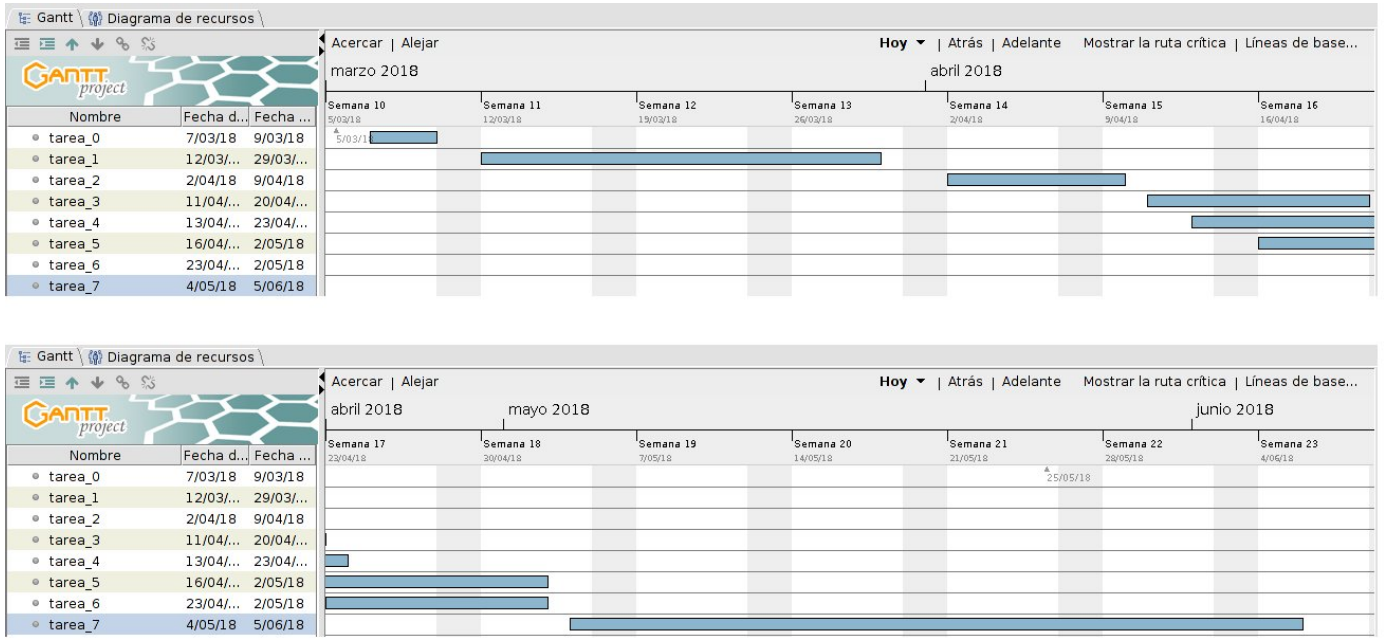

Figura 8: Diagrama de Gantt.

Tarea	Nombre	Fecha de inicio	Fecha de fin	Estimación
tarea_0	Instalación y toma de conceptos de MoveIt! y ROS	07/03/2018	10/03/2018	4 días
tarea_1	Adaptación de software *pick and place* utilizando módulos de visión	12/03/2018	30/03/2018	18 días
tarea_2	Estudio de visión por computador	02/04/2018	10/04/2018	8 días
tarea_3	Estudio de algoritmo k-medias e implementación en C++	11/04/2018	21/04/2018	10 días
tarea_4	Desarrollo de software de visión por computador	13/04/2018	23/04/2018	10 días
tarea_5	Desarrollo de software de clasificación en entorno estático	16/04/2018	03/04/2018	17 días
tarea_6	Desarrollo de software de clasificación en entorno dinámico	23/04/2018	03/05/2018	11 días
tarea_7	Realización de pruebas y corrección de errores	04/05/2018	06/06/2018	30 días

Cuadro 1: Planificación de proyecto.

3.3. Presupuesto

3.3.1. Recursos hardware y software

Para hacer el presupuesto de este proyecto se ha utilizado la fórmula siguiente:

$$(D/V) * C$$

Donde:

- D es el número de meses que se ha utilizado el producto para el proyecto.
- V es la vida útil del producto en número de meses.
- C es el coste total del producto en euros.

Artículo	**Coste** (euros)	**Dedicación** (meses)	**Vida Útil** (meses)	**Coste Aplicable** (euros)
Baxter	24078€	4	120	802,6€
Ordenador de sobremesa	1500€	4	60	100€
Cinta transportadora	350€	2	120	5,83€
Total				868,43€

Cuadro 2: Tabla de costes.

3.3.2. Recursos humanos

En este trabajo se han empleado aproximadamente 80 días, contando con que una jornada laboral común suelen ser 8 horas al día y con que además de este proyecto se ha trabajado en otras asignaturas, se considera que en media se han invertido 4 horas al día.

Suponiendo que un ingeniero informático recién graduado en España representa un coste de 2000 euros al mes, las 320 horas del proyecto estarían valoradas en 1731.2 euros en total.

3.3.3. Coste total

COSTE TOTAL	
Recursos HW y SW	868€
Recursos humanos	1731€
Coste total	2599€

Cuadro 3: Coste total del proyecto

Por lo tanto, añadiendo un 20 % referido a gastos, el coste total del proyecto sería de:

PRECIO TOTAL: 3120€(IVA no incluido)

3.4. Gestión de problemas

En este proyecto se han enfrentado muchos problemas de los que se hablará en este apartado. Se hacía imprescindible su resolución, ya que suponían un

impacto grave en el funcionamiento del código.

Problema 1. Incompatibilidades a la hora de utilizar las plataformas MoveIt!, Gazebo y ROS sobre Ubuntu 16.04 LTS en el portátil personal. No se propuso ninguna solución, puesto que en el laboratorio había un servidor disponible con Ubuntu 14.04.5 LTS y ROS Indigo.

Problema 2. Actualización en el servidor del laboratorio. Debido a una actualización de Windows, la gráfica de este computador quedó inutilizable. Para resolver este problema, se formateó el servidor, instalando ROS Kinetic sobre Ubuntu 16.04 LTS y MoveIt! (esta vez no resultó necesario instalar el simulador Gazebo, ya que este servidor se encuentra en el mismo lugar que el robot Baxter).

Problema 3. Planificación de MoveIt!. A la hora de utilizar OMPL, el algoritmo de planificación de MoveIt!, el resultado obtenido no siempre es el esperado; en ocasiones, al intentar ejecutar movimientos simples, el algoritmo recalcula la posición de las articulaciones de Baxter de forma que esta sea la óptima, sin embargo, resulta en un movimiento del brazo robótico más espaciado que impide realizar la clasificación correctamente. Para solucionarlo se implementan como movimientos pequeñas variaciones con respecto a la posición que el brazo tenía anteriormente.

Problema 4. Permisos de ejecución. ROS suele expresar los errores como *"Process is dead"*. Normalmente especifica anteriormente el porqué, pero en este caso no lo hacía. El problema era que el programa escrito en Python no tenía permisos de ejecución.

Problema 5. Iluminación. La iluminación es una característica clave en un programa que utiliza visión por computador, y son muchos los problemas que esta puede dar, un ejemplo son los focos, ya que aunque no los veamos, están parpadeando constantemente, esto influye al capturar las imágenes de los objetos. Existen focos de luz azul capaces de solucionar esto; en mi caso, utilicé la luz natural y alguna artificial de la sala, intentando crear sombras para que esta no incidiese directamente sobre los objetos.

Problema 6. Entrada del algoritmo K-means. El problema principal consistía en que el único tipo de dato que se le puede pasar a este algoritmo es *Mat* de OpenCV. Puesto que se estaba trabajando con *PoseArray* de ROS para obtener las coordenadas en la imagen, se intentó adaptar para pasárselo a K-means. Este consiste en un array de *PoseStamped*, un tipo de dato que incluye unas coordenadas para especificar la posición del objeto y otras para la orientación. Puesto que el objetivo más básico del proyecto era realizar una segmentación por color y posición y al algoritmo no se le

podían pasar valores RGB ya que la salida resultaría ser otro valor intermedio RGB y no una posición entre los dos clústers, se definió específicamente un valor para cada objeto que indicaba su color y se mandaba a través de la coordenada z, ya que no es necesario especificarla porque los objetos siempre se mantienen en la misma z (sobre la mesa o cinta transportadora). Así, la posición, orientación y color del objeto se incluyen en la matriz *Mat* para obtener los dos clústers tras ejecutar K-means.

4. Análisis y diseño del sistema

En esta sección se especificarán los casos de uso que componen el sistema, además de los requisitos funcionales y no funcionales para ambas clasificaciones, estática y dinámica, en este último caso utilizando una cuña como herramienta de separación.

Las principales características funcionales son:

- Mover los brazos de Baxter a la posición inicial (figura 4) con la que comenzará todas sus ejecuciones.

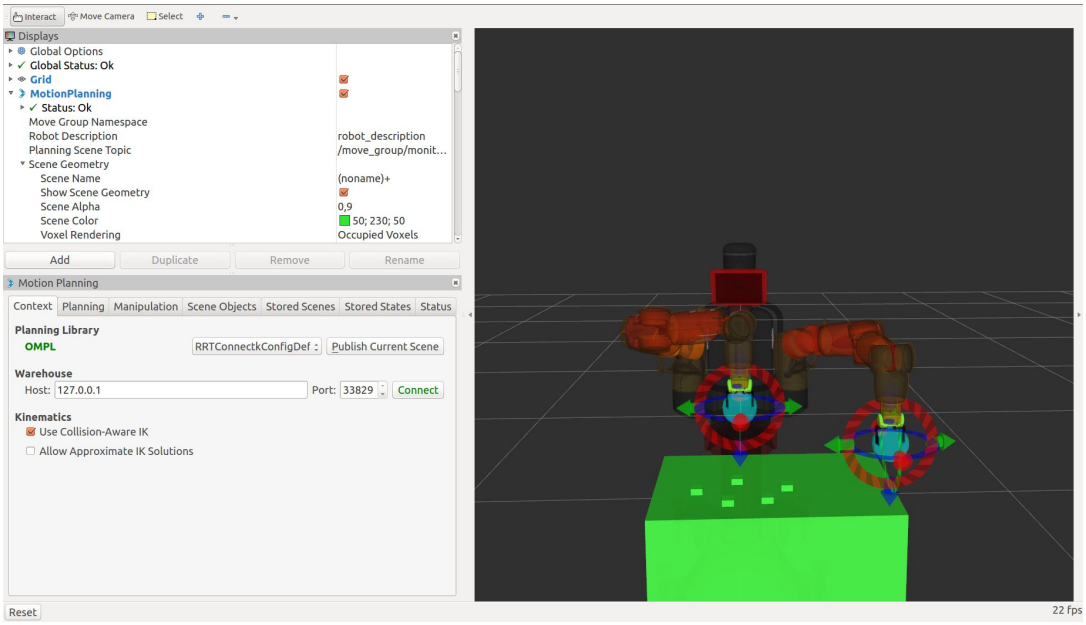

Figura 9: Posición inicial de Baxter.

- Visión por computador para obtener la imagen de los objetos sobre la mesa o cinta transportadora.
- Reconocimiento de objetos.
- Si no se reconoce ningún objeto, Baxter no hará nada hasta que sean identificados.
- Obtención de información acerca de la posición de los objetos a través de los tópicos de ROS.
- Si se trata de clasificación en entorno estático, se procederá a situar el brazo de Baxter entre los objetos con la orientación adecuada para separarlos.

- Si, por contrario, se trata de clasificación en entorno dinámico, Baxter llevará la cuña a un punto cercano al punto medio, y conforme los objetos vayan llegando al punto medio obtenido, moverá la cuña con la orientación necesaria para separarlos.

4.1. Restricciones del sistema

4.1.1. Restricciones hardware

- El rango de movimiento del brazo derecho (el brazo utilizado) de Baxter es limitado. A la hora de calcular la posición en la que pueden estar los objetos tras un determinado tiempo es posible obtener una posición a la que Baxter no pueda llegar.
- No se puede controlar la planificación OMPL realizada por MoveIt!, por lo que no se puede conocer a ciencia cierta cómo va a realizar el movimiento o cuánto va a tardar en realizarlo, siendo la velocidad máxima 1m/s.
- Baxter tiene que estar conectado a la corriente.
- La cinta transportadora utilizada es estrecha, por lo que no caben muchos objetos.
- La cinta transportadora consta de 10 velocidades que se controlan mediante una rueda.
- Es importante disponer de una iluminación suficiente como para que el sistema de reconocimiento identifique los objetos, evitando la aparición de reflejos y artefactos.

4.1.2. Restricciones software

- Es necesario realizar este proyecto sobre un directorio RSDK (conexión entre el PC y el robot mediante ROS) de ROS.
- En este caso se ha utilizado ROS Kinetic, por lo que sólo se ha podido montar sobre Ubuntu 16.04 LTS, Xenial.
- El lenguaje de programación utilizado para la parte de visión por computador será C++, y para el control, Python.

4.2. Especificación de casos de uso

Los casos de uso describen cómo interactúa un actor con el entorno. [26]
Las tareas de este proyecto son realizadas por Baxter y el programa de visión por computador.

- **Software de reconocimiento.** Este programa se encarga de procesar la información que llega de las cámaras de Baxter y enviarla ya procesada de nuevo al robot.
- **Baxter.** Se encarga de realizar los movimientos correspondientes utilizando la información que le llega del software de visión.

A continuación se muestra el diagrama de casos de uso a seguir para describir cada uno de ellos.

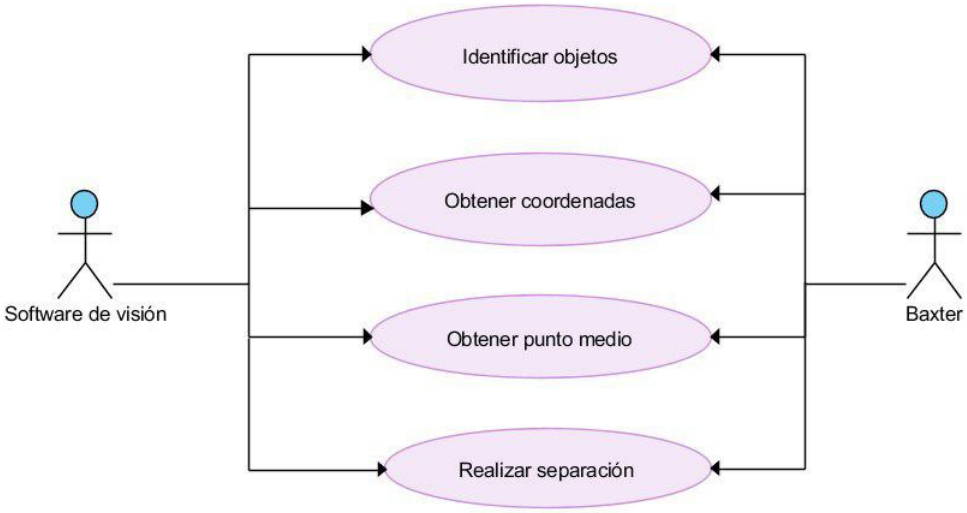

Figura 10: Diagrama de casos de uso.

Caso de Uso	Reconocimiento de objetos	**CU-001**
Actor	Baxter y software de visión	
Tipo	Principal — Real	
Referencias	RF-001 13, RF-002 14, RF-003 15, RF-004 16, RF-007 19, RNF-001 20, RNF-002 21	CU-004 5, CU-003 6, CU-004 7, CU-005 8, CU-006 9, CU-007 10, CU-008 10, CU-008 11, CU-009 12
Precondiciones	Los objetos deben estar en el rango de visión de la cámara de Baxter y la iluminación debe permitir reconocerlos.	
Poscondiciones	Se conocerá el centro de los objetos reconocidos en la imagen.	
Propósito	Conocer los objetos que hay sobre la mesa o cinta transportadora para clasificarlos.	
Resumen	El software de visión se encarga de reconocer y obtener el punto medio de los objetos que le llegan de la cámara de Baxter.	

Cuadro 4: Caso de uso CU-001.

Caso de Uso	Obtención del punto medio	**CU-002**
Actor	Software de visión	
Tipo	Secundario — Real	
Referencias	RF-003 15, RF-004 16, RF-005 17, RF-007 19, RNF-001 20, RNF-002 21	CU-001 4, CU-003 6, CU-004 7, CU-005 8, CU-006 9, CU-007 10, CU-008 10, CU-008 11, CU-009 12
Precondiciones	Se deben haber reconocido objetos en la mesa. El algoritmo k-medias debe haberse ejecutado correctamente.	
Poscondiciones	Se conocerá el punto medio de los dos clústers.	
Propósito	Conocer el punto medio para situar en él el brazo robótico.	
Resumen	El software de visión obtiene a partir del algoritmo k-medias la posición de dos clústers de los que calcula el punto medio.	

Cuadro 5: Caso de uso CU-002.

4.2.1. Entorno estático

Además de los casos de uso 4 y 5, para realizar la clasificación en entorno estático es necesario el siguiente paso.

Caso de Uso	Situar brazo en punto medio en entorno estático	**CU-003**
Actor	Baxter	
Tipo	Primario — Esencial	
Referencias	RF-005 17, RF-006 18, RF-007 19, RNF-001 20, RNF-002 21	CU-001 4, CU-002 5, CU-004 7, CU-005 8
Precondiciones	Haber obtenido el punto medio entre los dos clústers con el algoritmo k-medias.	
Poscondiciones	El brazo robótico se situará en el punto medio entre los clústers con una orientación perpendicular a la que forma la recta que une los dos clústers.	
Propósito	Situar el brazo de Baxter en el punto medio con la orientación necesaria para realizar la clasificación.	
Resumen	Baxter mueve su brazo derecho al punto medio a partir de los datos obtenidos mediante tópicos de ROS.	

Cuadro 6: Caso de uso CU-003.

Caso de Uso	Rotar paleta en entorno estático	**CU-004**
Actor	Baxter	
Tipo	Primario — Esencial	
Referencias	RF-005 17, RF-006 18, RF-007 19, RNF-001 20, RNF-002 21	CU-001 4, CU-002 5, CU-003 6, CU-005 8
Precondiciones	Haber obtenido el punto medio entre los dos clústers y haber situado en él el gripper con la orientación calculada.	
Poscondiciones	El gripper girará hasta que se encuentre paralelo al eje horizontal.	
Propósito	Rotar el gripper de forma que se pueda realizar más fácilmente la separación.	
Resumen	Se rota el gripper de Baxter de forma que los objetos quedan separados por una recta horizontal.	

Cuadro 7: Caso de uso CU-004.

Caso de Uso	Separar objetos en entorno estático	**CU-005**
Actor	Baxter	
Tipo	Primario — Esencial	
Referencias	RF-005 17, RF-006 18, RF-007 19, RNF-001 20, RNF-002 21	CU-001 4, CU-002 5, CU-003 6, CU-004 7
Precondiciones	Tener situado el gripper en el punto medio con una orientación paralela al eje horizontal.	
Poscondiciones	El brazo se desplazará en el eje vertical para separar los objetos.	
Propósito	Separar los objetos mediante movimientos en el eje vertical de Baxter.	
Resumen	Desplazar el brazo arriba y abajo en el eje vertical separando los objetos conforme a color y posición.	

Cuadro 8: Caso de uso CU-005.

4.2.2. Entorno dinámico

Para realizar la clasificación con una cuña como herramienta y una cinta transportadora sobre la que se sitúan los objetos, además del caso de uso 4, es necesario lo siguiente.

Caso de Uso	Cálculo de la velocidad en entorno dinámico	**CU-006**
Actor	Software	
Tipo	Secundario — Real	
Referencias	RF-004 16, RF-005 17, RF-006 18, RF-007 19, RNF-001 20, RNF-002 21	CU-001 4, CU-002 5, CU-007 10, CU-008 11, CU-009 12
Precondiciones	Haber obtenido el punto medio entre los dos clústers en dos ocasiones.	
Poscondiciones	Se conocerá la velocidad media a la que van los objetos sobre la cinta transportadora.	
Propósito	Obtener la velocidad a la que van los objetos para realizar la separación.	
Resumen	Se obtiene la posición del punto medio entre los dos clústers en dos ocasiones, una al principio y otra un tiempo después y se calcula la velocidad que llevan los objetos.	

Cuadro 9: Caso de uso CU-006.

Caso de Uso	Situar brazo en posición cercana al punto medio en entorno dinámico	**CU-007**
Actor	Baxter	
Tipo	Primario — Esencial	
Referencias	RF-005 17, RF-006 18, RF-007 19, RNF-001 20, RNF-002 21	CU-001 4, CU-002 5, CU-006 9, CU-008 11, CU-009 12
Precondiciones	Haber obtenido el último valor de punto medio entre los dos clústers y la velocidad a la que van los objetos.	
Poscondiciones	El brazo robótico se situará en un punto cercano al punto medio aumentando la rotación del gripper.	
Propósito	Mover el brazo a un punto cercano al punto medio para realizar el barrido necesario para clasificar los objetos.	
Resumen	Baxter mueve su brazo derecho al punto calculado con un valor de orientación más alto que el calculado para el punto medio.	

Cuadro 10: Caso de uso CU-007.

Caso de Uso	Situar brazo en punto medio en entorno dinámico	**CU-008**
Actor	Baxter	
Tipo	Primario — Esencial	
Referencias	RF-005 17, RF-006 18, RF-007 19, RNF-001 20, RNF-002 21	CU-001 4, CU-002 5, CU-006 9, CU-007 10, CU-009 12
Precondiciones	El gripper debe estar situado en un punto cercano al punto medio.	
Poscondiciones	Baxter mueve el brazo hacia el último punto medio obtenido con la orientación necesaria para realizar la separación.	
Propósito	Mover el brazo al punto medio para realizar la separación.	
Resumen	Baxter mueve su brazo derecho al punto medio con la orientación adecuada para que los objetos queden separados.	

Cuadro 11: Caso de uso CU-008

Caso de Uso	Sacudida de objetos en entorno dinámico	**CU-009**
Actor	Baxter	
Tipo	Primario — Esencial	
Referencias	RF-005 17, RF-006 18, RF-007 19, RNF-001 20, RNF-002 21	CU-001 4, CU-002 5, CU-006 9, CU-007 10, CU-008 11
Precondiciones	El brazo se encuentra en el punto medio en el que separa los objetos.	
Poscondiciones	Baxter moverá el brazo con la misma orientación que tenía en el eje vertical para separar los objetos más satisfactoriamente.	
Propósito	Realizar movimientos verticales con la posición actual del gripper para separar más rápido los objetos.	
Resumen	Se mueve el brazo derecho en su posición actual en el eje vertical separando más rápidamente los objetos.	

Cuadro 12: Caso de uso CU-009.

4.3. Especificación de requisitos

Los requisitos pueden ser clasificados en funcionales o no funcionales. Los funcionales se refieren a cualquier comportamiento que esté directamente relacionado con el funcionamiento del sistema, mientras que los requisitos no funcionales especifican cómo debe comportarse el sistema.

4.3.1. Requisitos funcionales

REQUISITO DEL SISTEMA					
ID	RF-001		**Fuente**	CU-001	
Nombre	Reconocimiento de objetos				
Descripción	Se obtienen imágenes de los objetos situados en la mesa, visibles a la cámara del brazo derecho de Baxter.				
Prioridad	Alta	**Necesidad**	Esencial	**Estabilidad**	Estable

Cuadro 13: Requisito funcional RF-001.

REQUISITO DEL SISTEMA					
ID	RF-002		**Fuente**	CU-001	
Nombre	Recortar la imagen				
Descripción	Recortar la imagen para asegurar que la visión no reconoce otros objetos distintos a los que hay en la cinta.				
Prioridad	Alta	**Necesidad**	Deseable	**Estabilidad**	Estable

Cuadro 14: Requisito funcional RF-002.

REQUISITO DEL SISTEMA					
ID	RF-003		**Fuente**	CU-001	
Nombre	Identificar objetos				
Descripción	El sistema identifica los objetos de colores situados en la escena.				
Prioridad	Alta	**Necesidad**	Esencial	**Estabilidad**	Estable

Cuadro 15: Requisito funcional RF-003.

REQUISITO DEL SISTEMA					
ID	RF-004		**Fuente**	CU-001	
Nombre	Obtener coordenadas				
Descripción	El sistema obtiene el centro de los objetos situados en la escena.				
Prioridad	Alta	**Necesidad**	Esencial	**Estabilidad**	Inestable

Cuadro 16: Requisito funcional RF-004.

REQUISITO DEL SISTEMA					
ID	RF-005		**Fuente**	CU-001	
Nombre	Obtener punto medio y orientación				
Descripción	El sistema obtiene, a partir del algoritmo k-medias, el centro entre los dos clústers. A partir de los dos puntos de los clústers calcula la recta que forman y la pendiente perpendicular a esta, que será la orientación asociada a la separación.				
Prioridad	Alta	**Necesidad**	Esencial	**Estabilidad**	Inestable

Cuadro 17: Requisito funcional RF-005.

REQUISITO DEL SISTEMA					
ID	RF-006		**Fuente**	CU-001	
Nombre	Cargar posición inicial				
Descripción	Baxter pone sus extremidades en la posición inicial (figura 4).				
Prioridad	Alta	**Necesidad**	Esencial	**Estabilidad**	Estable

Cuadro 18: Requisito funcional RF-006.

REQUISITO DEL SISTEMA					
ID	RF-007		**Fuente**	CU-001	
Nombre	Mostrar escena				
Descripción	Mediante tópicos de ROS se obtiene la escena que Baxter ve en cada momento, delineando los objetos reconocidos.				
Prioridad	Media	**Necesidad**	Deseable	**Estabilidad**	Inestable

Cuadro 19: Requisito funcional RF-007.

4.3.2. Requisitos no funcionales

REQUISITO DEL SISTEMA					
ID	RNF-001		**Fuente**	CU-001	
Nombre	Iluminación adecuada				
Descripción	La luz debe ser lo bastante intensa como para ver los objetos pero para que estos no la reflejen.				
Prioridad	Media	**Necesidad**	Deseable	**Estabilidad**	Inestable

Cuadro 20: Requisito no funcional RNF-001.

REQUISITO DEL SISTEMA					
ID	RNF-002		**Fuente**	CU-001	
Nombre	Utilizar a Baxter				
Descripción	En este proyecto el robot utilizado es Baxter.				
Prioridad	Alta	**Necesidad**	Esencial	**Estabilidad**	Estable

Cuadro 21: Requisito no funcional RNF-002.

5. Diseño

Para especificar la estructura que se ha seguido en el diseño del proyecto, se muestra un diagrama de flujo vertical en el que se incluye tanto el programa encargado de la clasificación en entorno estático como el encargado de la clasificación en entorno dinámico.
Seguidamente, se expondrán los principales módulos responsables de realizar las tareas especificadas.

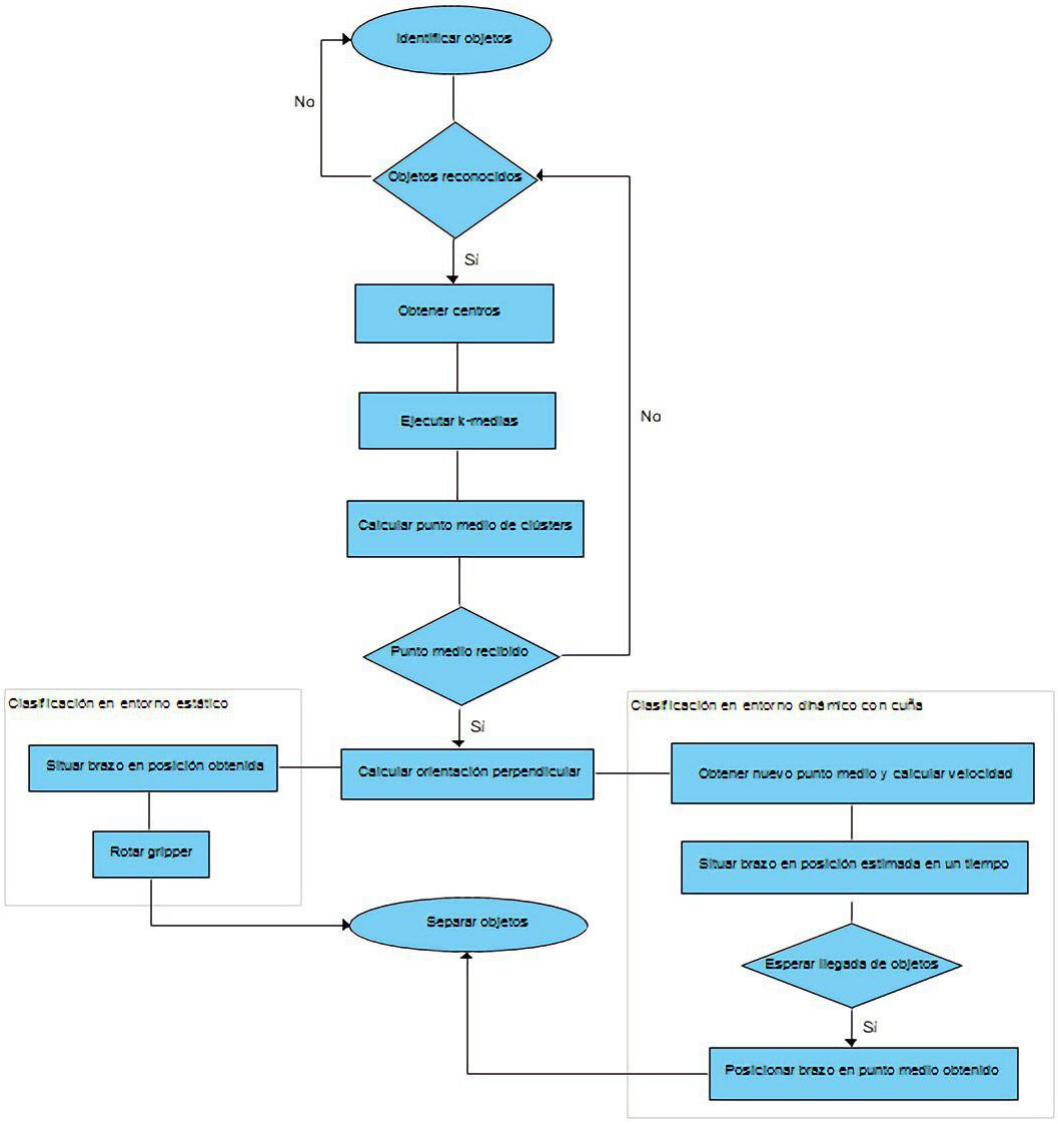

Figura 11: Diagrama de flujo.

- **Módulo de reconocimiento.** Este es el que se encargará de identificar los objetos, procesar los datos y enviarlos al módulo de control de Baxter.

- **Módulo de control de Baxter.** La función de este módulo será realizar los movimientos y cálculos necesarios para separar los objetos. Además se incluye la planificación y ejecución de trayectorias.

5.1. Módulo de reconocimiento

Este módulo incorpora la visión por computador. Lo componen:

- *GetPosition*. Es la función encargada de obtener las coordenadas del centro de los objetos en la imagen.
- *Get3DPos*. Se encarga de obtener las coordenadas en el mundo real a partir de las coordenadas de la imagen.
- *GetRealPos*. A partir de las coordenadas dadas por *Get3DPos*, obtiene el centro de los objetos en el mundo real.
- *ImageCallBack*. Cada vez que se recibe una imagen, esta función se encarga de procesarla y enviar los datos a través de ROS al módulo de control de Baxter.

5.2. Módulo de control de Baxter

Como se dijo anteriormente, este es el módulo que se encargará del control sobre el robot, además de realizar algunos cálculos necesarios para poder clasificar los objetos.

- *baxter_interface*: este nodo es el encargado de definir los brazos y *grippers* de Baxter en Python. Para ello, se utilizan las siguientes órdenes:

  - *right_arm = baxter_interface.limb.Limb('right')*
  - *right_gripper = baxter_interface.Gripper('right')*

  Además, una vez se han definido los *grippers*, estos se pueden calibrar y abrir o cerrar.

- *moveit_commander*: este es el nodo de MoveIt! para Python que se encargará de realizar los movimientos en Baxter y planificar la escena para detectar colisiones o utilizar RViz para moverlo.

  - *p = PlanningSceneInterface("base")*. Esta orden inicializa la escena en MoveIt!.
  - *arms_group = MoveGroupInterface("both_arms", "base")*. Se encarga de inicializar ambos brazos en la escena.
  - *rightarm_group = MoveGroupInterface("right_arm", "base")*. Inicializa el brazo derecho en la escena. Se realiza análogamente para el izquierdo.
  - *rightarm_group.moveToPose (position, "right_gripper", max_velocity_scaling_factor=1, plan_only=False)*. Esta orden es la encargada de realizar los movimientos, en este caso, el brazo derecho se moverá hasta la posición "*position*" con una velocidad máxima de 1 m/s.

- *rotate_pose_by_euler_angles*: es la función encargada de rotar los *grippers* a la orientación establecida.

6. Implementación

Las tareas realizadas pueden clasificarse en tareas de visión por computador y tareas de control. En este apartado se explicará cada una de ellas, incluyendo todos los programas realizados a lo largo del proyecto.

Los tópicos de ROS realizan una tarea importante en este trabajo, ya que gracias a ellos podemos escribir códigos en distintos lenguajes de programación, en este caso, la visión en C++ y el control en Python, y compartir información entre ellos. A continuación se explica cada una de estas tareas.

baxter_img.cpp Este código escrito en lenguaje C++ es el encargado del procesamiento de imágenes utilizando OpenCV y de enviar la posición del punto medio entre los dos clústers al tópico */classification*.

pick_and_place.py Este programa se subscribe al tópico */detected_objects* para obtener la posición de los objetos y construir una torre con ellos.

classif.py Este archivo en Python se subscribe al tópico */classification* y realiza una clasificación de los objetos situados en la mesa.

dyn_classif.py Este otro programa también escrito en Python se subscribe al tópico */classification* para realizar una clasificación en entorno dinámico con una paleta como herramienta.

dyn_wedge.py Este último programa en Python obtiene el punto medio a través del tópico */classification* y realiza una clasificación en entorno dinámico con una cuña como herramienta.

archivo.launch Estos archivos son los encargados de inicializar el nodo de MoveIt! y RViz, la cámara del brazo derecho de Baxter y los programas de visión y control de cada caso definido anteriormente.

CMakeLists.txt Archivo que incluye los paquetes, dependencias y librerías que requiere el proyecto.

package.xml Define las dependencias de compilación y ejecución que requiere el proyecto además de definir algunas variables como el nombre, la versión o la licencia.

6.1. Esquema general del escenario

En este apartado se muestran los componentes utilizados en el escenario y las conexiones que se establecen entre ellos.

- Objetos a clasificar. Se utilizarán varios objetos de distinto tamaño y color. Además se realizarán tests utilizando mayor o menor número de piezas.

- Cinta transportadora. Es la mesa sobre la que se posicionan los objetos. Se utilizará tanto en entorno dinámico como en estático. Para realizar la segmentación en entorno dinámico, se utilizarán las primeras velocidades de la cinta, que se pueden regular mediante la rueda de la que dispone la cinta.

- Baxter. Se utilizará el brazo robótico derecho para el control y la cámara de este mismo brazo para capturar las imágenes.

- Ordenador. Sobre este PC se ejecuta la versión de ROS acorde con la distribución de Ubuntu utilizada, en nuestro caso, ROS Kinetic. Este ordenador se conecta a través de una red Ethernet a Baxter y, mediante línea de comandos en un terminal RSDK, se comunica con él.

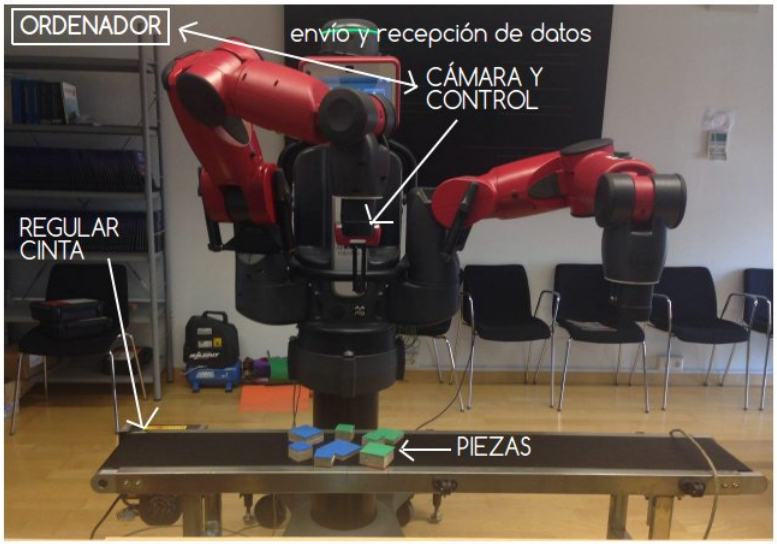

Figura 12: Esquema general del escenario.

6.2. Visión por computador

Como base para realizar la visión se ha utilizado un filtro de color [5], a partir del cual se realiza la clasificación de objetos por color y posición.
En esta sección se explicarán estas funciones, que se pueden dividir en reconocimiento de objetos y procesamiento de imágenes.

6.2.1. Reconocimiento de objetos

Utilizando los tópicos de ROS se obtiene la imagen captada por la cámara del brazo derecho de Baxter, a partir de la cual se puede operar utilizando una función de *callback* si existe, es decir, la función que se ejecutará al suscribirse al tópico de la cámara. En este caso esa función existe y se llama *ImageCallback*.
Una vez se ha suscrito, es necesario conocer el color de los objetos rectangulares para reconocer su posición, tamaño y orientación, ya que sólo detectará los colores predefinidos en el código. Esto se realiza en formato HSV, estableciendo un rango para cada color de forma que estos se puedan filtrar en la imagen dejando el fondo negro; se han elegido colores con valores HSV lo bastante diferentes como para poder reconocerlos fácilmente, el azul y el verde.

Para calcular estos valores HSV, se ha utilizado el archivo hsvThresholder.py [27], al que se le pasa la imagen captada por la cámara de Baxter, que se puede ejecutar con el comando:

python hsvThresholder.py image.jpg

Para obtener estos valores, es recomendable lanzar el programa para más imagenes con distintas posiciones de los objetos y distina iluminación. Se realizan capturas de los objetos con la máxima y la mínima iluminación posible, obteniendo un rango de valores HSV para los que se reconoce ese color.

Una vez se han reconocido los objetos que presentan los colores predefinidos, se obtiene su centro y se realiza un interlineado por sus bordes. En la siguiente figura 6.2.1 se muestran distintas piezas identificadas en la ventana de visualización. Como se dijo anteriormente, estos objetos se han buscado por color en la captura proporcionada por el tópico de la cámara del brazo derecho. Se han obtenido sus centros en coordenadas de imagen, a partir de los cuales se han definido sus perímetros mediante líneas de color negro.

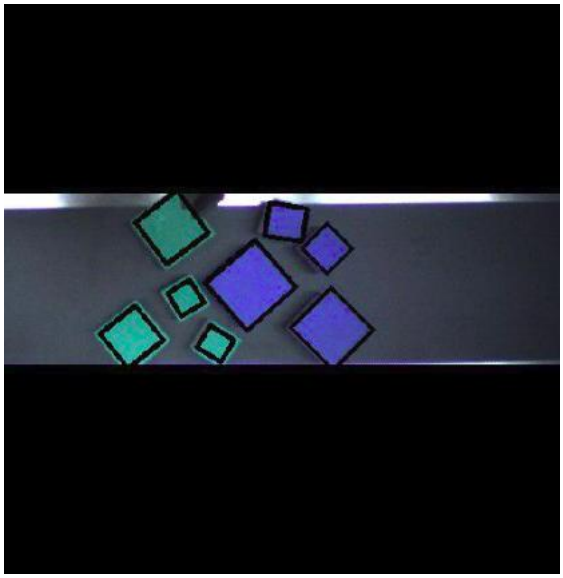

Figura 13: Captura que muestra el reconocimiento de los objetos y la definición de sus perímetros.

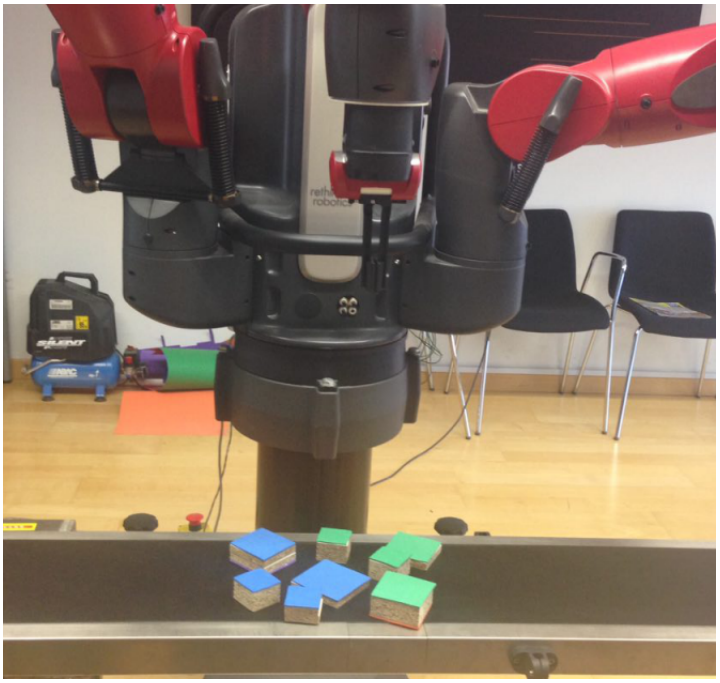

Figura 14: Imagen que muestra la configuración de objetos de la imagen anterior 6.2.1 desde fuera.

6.2.2. Procesamiento de imágenes

Al obtener la captura a través del tópico, se realiza una copia de esta a un tipo de dato correspondiente a una clase de OpenCV para imágenes, y se buscan, color por color, los objetos disponibles en esta imagen.

Para el posible ruido que pueda tener la imagen, se utiliza la función *cvSmooth()* [28] de OpenCV y se realiza un filtrado del rango de color que se está buscando en ese momento, dejando todo lo demás en negro. Se hace uso de la función *cvErode()* [29] de OpenCV para borrar algunos puntitos blancos que pueden quedar tras filtrar la imagen. Tras esto, la imagen se pasa a escala de grises y se obtienen sólo los píxeles del objeto, logrando reconocer el rectángulo que lo define. El resultado de los objetos con el interlineado se muestra en una ventana de visualización (figura 6.2.1), por lo que el usuario puede determinar la calidad de detección de objetos con las características del entorno en el que se encuentra.

Una vez se ha conseguido obtener el centro de todos los objetos visibles en la imagen, se realiza el clustering con el algoritmo K-means (explicado en el anexo 9). Para ello, es imperativo el uso de un tipo de dato de OpenCV llamado *Mat*, a través del que se copia un tipo de dato *PoseArray*, un array de *PoseStamped* que incluye las coordenadas de posición *x, y, z* y las coordenadas de orientación, *x, y, z, w*. Como se explicó en la sección de gestión de problemas (3.4), a través de este tipo de dato se le ha pasado tanto la posición del objeto como el color del mismo.
De la salida resultante del K-means se obtienen las etiquetas de cada uno de los objetos, siendo 0 si pertenece al clúster 0 y 1 si pertenece al 1, además de la posición obtenida en la última iteración de k-medias de los centroides que se han utilizado para obtener esos grupos, a partir de los que podemos calcular el punto medio y la orientación para realizar la clasificación.

Como todos estos cálculos se han realizado a partir de la imagen, las coordenadas que se han obtenido no nos valen para realizar las tareas de control, por ello, para enviar a través de un *publisher* de ROS el valor del punto medio hay que realizar una conversión a coordenadas del mundo real, para lo que es necesario añadir desplazamientos, calculados empíricamente, a la conversión de las coordenadas x e y a coordenadas de la "base" de la escena de MoveIt!, ya que a la hora de realizar cualquier movimiento, este va a estar relacionado con la "base" en esta escena.

6.3. Procedimientos de control

Para la implementación de estas tareas se ha utilizado el lenguaje Python junto con el paquete *moveit_commander*, que permite realizar movimientos rectos con distinta orientación del *gripper* o pinza.
Para realizar las tareas de clasificación, se han utilizado como herramientas una paleta y una cuña fabricadas a partir de cartón y goma eva, ya que las existentes, vistas en la sección 2.2.2, no eran adecuadas para realizar este proyecto, puesto que se está trabajando con más de un objeto a la vez.

6.3.1. *Pick and place* adaptado

Para este programa, se ha adaptado el código de *pick and place* proporcionado por Gazebo. Gazebo [30] es un simulador de robots en 3D de código abierto que incluye una gran variedad de modelos y resulta bastante útil a la hora de realizar pruebas, diseñar o mejorar las implementaciones. Además permite crear un entorno acorde con el real y simular interacciones.
Esta implementación de la que se habla está destinada a ser ejecutada en este simulador. En este proyecto, como ejercicio de adaptación se ha utilizado este programa junto con visión por computador para realizar un *pick and place* en el Baxter real (figura 6.3.1).

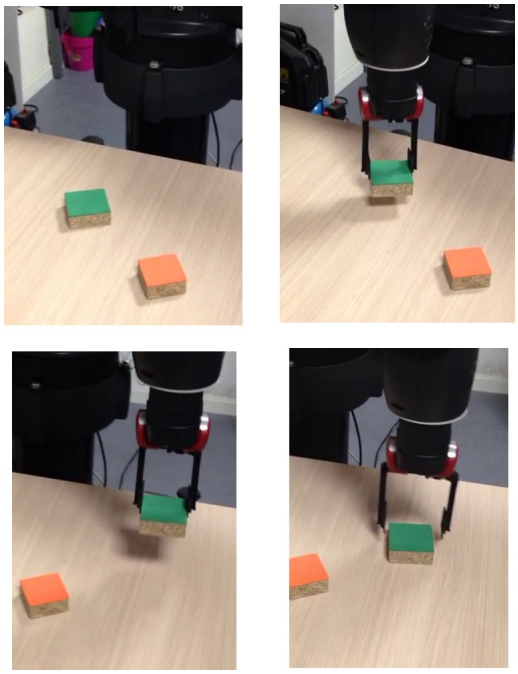

Figura 15: Capturas de la ejecución de *pick and place* adaptado.

6.3.2. Clasificación en entorno estático

Conociendo el punto medio obtenido por el programa de visión, utilizando la paleta (figura 6.3.2), Baxter mueve su brazo hasta esta posición con una orientación que también ha obtenido. Cuando se encuentra en esta posición, gira el *gripper* hasta que forme 90º situándose en un eje horizontal y arrastra los objetos hacia arriba y hacia abajo, separándolos entre sí (figura 6.3.2).

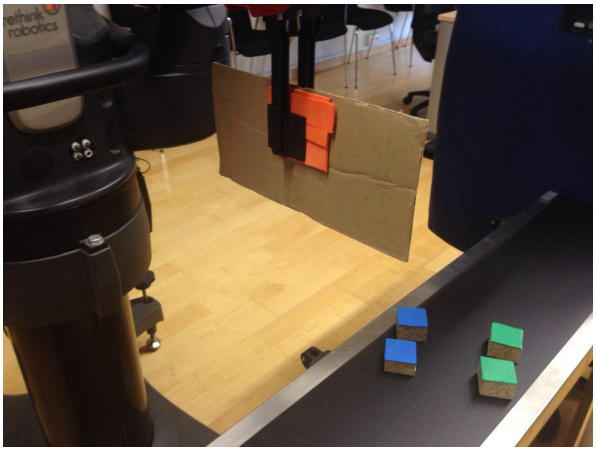

Figura 16: Paleta de cartón.

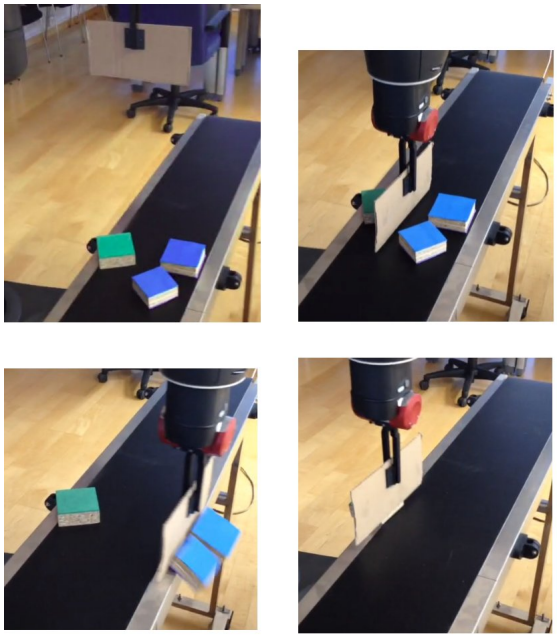

Figura 17: Capturas de la clasificación en entorno estático.

6.3.3. Clasificación en entorno dinámico con cinta transportadora

En este tipo de clasificación se han utilizado dos prototipos de herramienta, una cuña y una paleta hechas de cartón y goma eva para evitar el deslizamiento; sin embargo en una aplicación real, deberían utilizarse materiales más pesados y robustos, puesto que en ocasiones no se obtienen los resultados esperados debido al peso, rigidez o forma de estas sencillas herramientas construidas para la realización del proyecto.

El comienzo de ambas tareas es el mismo, obteniendo la posición en dos ocasiones separadas en el tiempo para conocer la velocidad y predecir en qué posición estarán en el momento de la separación.

1. Clasificación utilizando como herramienta una paleta 6.3.2

   Con esta herramienta se ha diseñado un comportamiento que separe primero los objetos de una clase y después los de la otra, calculando de nuevo la posición en la que se encontrarán entonces.
   Una vez conocemos la posición en la que estarán, el *gripper* se sitúa en ese punto y espera a que los objetos lleguen, después separa en primer lugar los de abajo, apartándolos con la paleta en la dirección y con la orientación adecuadas. En ese momento además obtiene el punto en el que estarán situados los demás y al terminar de separar los anteriores va hacia esa posición, en la que realiza un movimiento análogo al realizado anteriormente pero en la dirección contraria (figura 1).

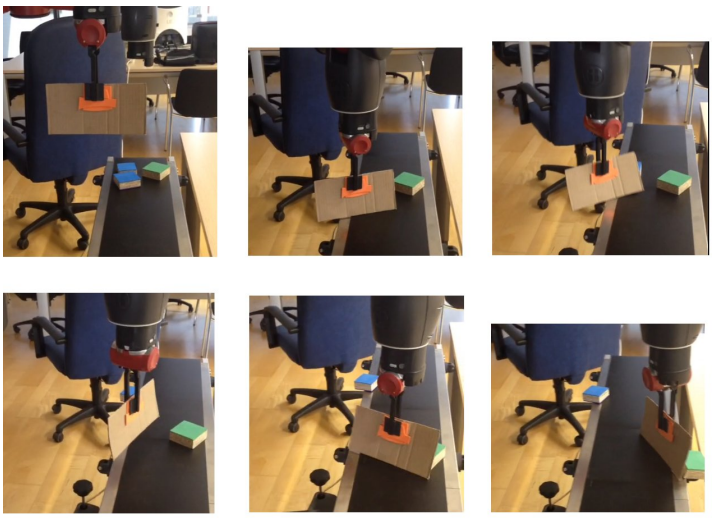

Figura 18: Capturas de la ejecución de la segmentación con paleta en entorno dinámico.

2. Clasificación utilizando como herramienta una cuña

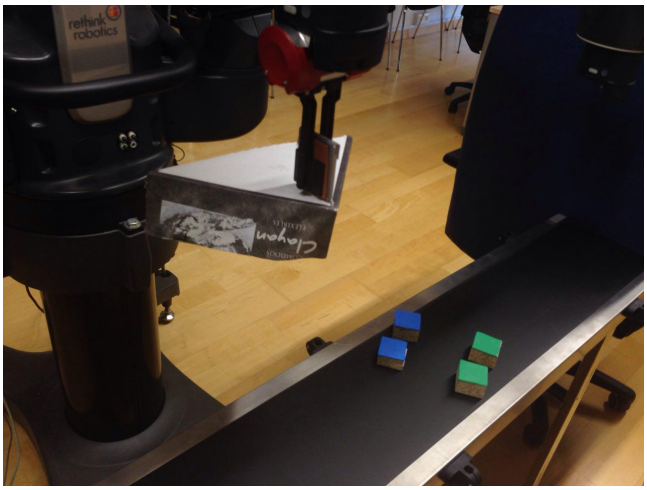

Figura 19: Cuña de cartón.

La cuña 2 es una herramienta que hace mucho más sencillo el control en este tipo de clasificación con cinta transportadora, por lo que las pruebas se realizarán con ella. Una vez se conoce la posición inicial, Baxter sitúa el brazo un poco más alejado de esta y con distinta orientación, de forma que pueda clasificar elementos cuya recta de separación tenga una pendiente más pronunciada. Cuando se detecta que los objetos han llegado a la posición en la que se encuentra la cuña, va hacia el punto que había calculado en un principio y avanza, de forma que los elementos se topen con los lados de la cuña y caigan fuera de la cinta transportadora, ayudándose de una oscilación en el eje vertical. (figura 2)

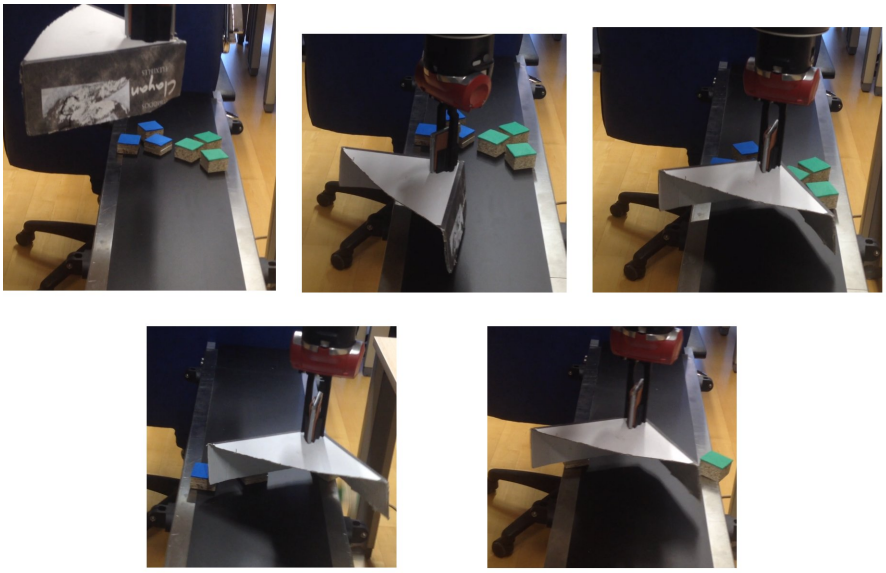

Figura 20: Capturas de la ejecución de la segmentación con cuña en entorno dinámico.

7. Pruebas

7.1. Introducción

Para contrastar el funcionamiento de las tareas realizadas conforme a los objetivos fijados, se van a realizar una serie de pruebas para cada tipo de clasificación.

Métricas

A continuación se exponen los experimentos realizados, así como la característica medida.

- Medidas de rendimiento.
  - Precisión frente a velocidad en clasificación dinámica.
  - Precisión frente a exhaustividad en clasificación estática con misma configuración de objetos.
  - Obtención del valor F para los valores de precisión y exhaustividad nombrados en el punto anterior.
  - Precisión en robótica adaptativa.
- Medidas de error. Estas se realizan tanto en entorno estático como dinámico.
  - Precisión frente a número de objetos.
  - Precisión frente a tamaño de objetos.

Resultados

7.2. Medidas de rendimiento

A continuación se muestran los resultados obtenidos para las pruebas de rendimiento.

7.2.1. Precisión frente a velocidad en clasificación dinámica

Para esta prueba, se han realizado unas 25 ejecuciones con distintas configuraciones (posición que toma cada objeto). Como vemos en la gráfica 7.2.1, cuanto mayor es la velocidad, menor es la precisión que se obtiene.

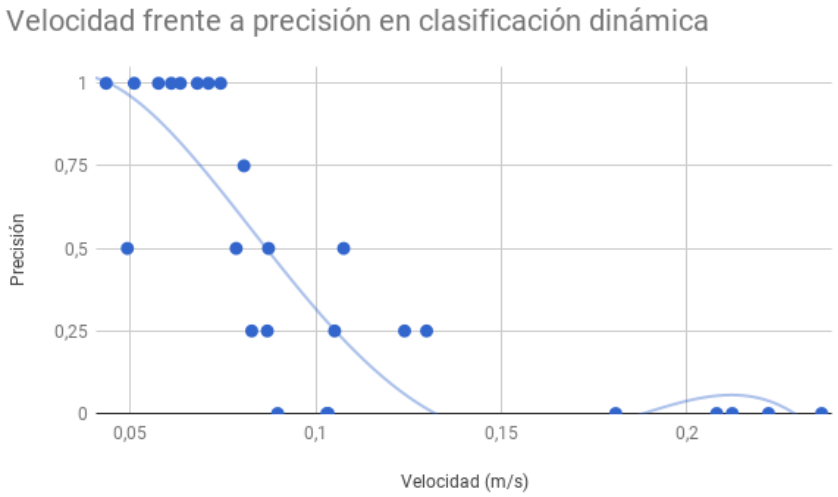

Figura 21: Gráfica que muestra la velocidad frente a la eficiencia en clasificación dinámica con cuña donde la línea sigue un ajuste polinómico de grado 6.

Se ha realizado un ajuste de los datos obtenidos con una línea de tendencia de tipo polinómica de grado 4 y se ha obtenido la figura anterior 7.2.1. Los resultados obtenidos siguen una curva lógica, siendo los valores de precisión más bajos cuanta más velocidad lleva la cinta transportadora. Es por esto por lo que las siguientes pruebas que se realicen en un entorno dinámico se ejecutarán con una velocidad menor a 0.1 m/s. Sin embargo, si nos fijamos en la gráfica 7.2.1, podemos observar algunos puntos algo dispares con respecto a la media; esto puede deberse al algoritmo de planificación de MoveIt! utilizado, como se explicaba en la sección 3.4.

7.2.2. Precisión y exhaustividad

Estos valores, utilizados en medicina como especificidad, *"specificity"* y sensitividad, *"sensitivity"* [31], también pueden ser de ayuda a la hora de comparar un algoritmo de clasificación con otro. Para explicarlos, se va a mostrar un ejemplo tras definir estos conceptos. [32]

A la hora de realizar una clasificación, se denomina al trabajo realizado de 4 posibles maneras fijándonos en el número de objetos verdes:

- **Verdadero positivo**. En esta prueba, son aquellos objetos verdes que Baxter ha separado y que realmente eran verdes.

- **Verdadero negativo**. Son los objetos separados como verdes, pero que realmente son azules.
- **Falso positivo**. Son aquellos objetos azules que no se han separado.
- **Falso negativo**. Son los objetos verdes que han quedado por separar.

El valor de precisión, "precision", se refiere a los objetos clasificados como positivos que realmente son verdaderos positivos, mientras que la exhaustividad o "recall" es el ratio de verdaderos positivos.
Así, si tenemos un conjunto de 4 objetos verdes y 4 azules 7.2.2 y separamos 2 verdes y 1 azul, la precisión será 2/3 y la exhaustividad 2/4. Es decir, habremos separado únicamente 2 objetos que realmente son verdes de 4 objetos verdes que había en total.

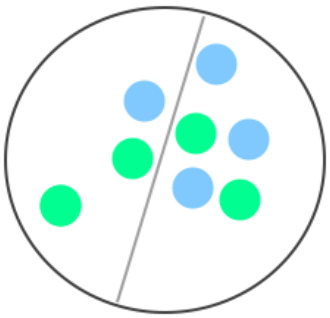

Figura 22: Ejemplo de precisión y exhaustividad en el que se ven los objetos que toma como verdaderos positivos (a la izquierda) y los que toma como negativos (a la derecha).

Para esta prueba, se ha realizado una modificación en el programa de clasificación estática, de forma que se pueda cambiar un parámetro "*precision*" que permita separar en mayor o menor porcentaje el número de objetos de un color, de manera que se pueda medir la precisión y exhaustividad que se obtienen para unos valores predefinidos.

Se han realizado ejecuciones para valores del parámetro "*precision*" de 0.5, 0.7 y 0.9 con 3 objetos de cada color, verde y azul. A continuación se muestra la gráfica de curvas PR, que consiste en una compensación entre precisión y exhaustividad para un diferente umbral, en este caso, el parámetro "*precision*".

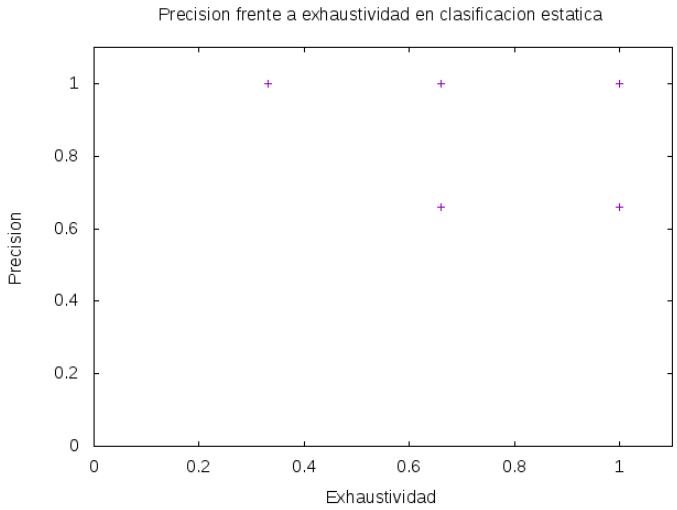

Figura 23: Gráfica que muestra la exhaustividad frente a la precisión en clasificación estática

El número de objetos de los que se dispone es de 3 de cada color, por lo que los valores de precisión y exhaustividad únicamente son 0.33, 0.66 y 0.99, ya que se obtienen en función del número de objetos verdes separados. No se ha realizado ninguna curva de ajuste ya que con los únicos datos que se pueden obtener no existe ninguna función que pueda aproximarse a ellos.

La alta precisión observada en la gráfica indica un bajo ratio de falsos positivos, mientras que un valor alto de exhaustividad indica un bajo ratio de falsos negativos [33]. En nuestro caso, la gráfica presenta valores altos para precisión y exhaustividad, por lo que se puede decir que los resultados son positivos, obteniendo una alta eficacia para los objetos clasificados.

Otra medida recurrente en este tipo de curvas consiste en calcular el valor F [31], equivalente a la media armónica. Este valor une las dos medidas, precisión y exhaustividad, cuando sus valores son cercanos, de forma que podamos realizar comparativas de algoritmos con una sola variable. Su fórmula cuando la importancia que se le da a ambas medidas es la misma, es la siguiente:

$$F_1 = 2 * (precision * exhaustividad)/(precision + exhaustividad)$$

En caso de que no se le dé el mismo peso a ambas medidas se utiliza la siguiente fórmula:

$$F_\beta = (1+\beta^2)*(precision/exhaustividad)/(\beta^2 * precision + exhaustividad)$$

Ya que es una medida ponderada, en esta prueba se utilizará F2, una variante en la que se da más peso a la exhaustividad frente a la precisión, es decir, en nuestro caso, se intenta que se separe el mayor número de objetos verdes, aunque además se clasifique alguno azul con estos. La fórmula para ello es la siguiente:

$$F_2 = 5 * (precision * exhaustividad)/(4 * precision + exhaustividad)$$

Calculándolo para cada uno de los valores obtenidos en precisión y exhaustividad y quedándonos con el valor más alto se obtiene: $F2 = 1$, seguido de $F2 = 0.9066$.

Puesto que este valor es la media de ambas medidas precisión y exhaustividad, podemos decir que, como se explicó anteriormente, las pruebas realizadas para este número de objetos presentan una alta eficacia, separándose el mayor número de verdes posible.

7.2.3. Robótica adaptativa

Para esta implementación, se ha utilizado el código de clasificación en entorno estático. A partir de este, se ha definido una variable booleana que indica cuándo la segmentación se debe enfocar de forma adaptativa. Cuando esta variable se establece como *True*, el programa preguntará la precisión obtenida en la ejecución anterior en un rango de [0:1] y la guardará en una variable. Como caso inicial, esta se debe establecer a -1.

A partir de este parámetro modificado a través de línea de comandos que llamaremos *"precision_value"* y otro valor *alpha* inicializado a 0.1, el programa se encarga de alterar el valor de la coordenada x, que en el caso de Baxter corresponde a la coordenada en el eje de ordenadas. De esta forma, cuanto menor sea el valor de *"precision_value"*, mayor será la modificación de la coordenada x, que hará que el brazo de Baxter se sitúe más arriba, alcanzando más objetos de un mismo color.

Para mostrar el funcionamiento de esta aproximación se expone una tabla a partir de algunos resultados obtenidos, que consta del valor de precisión "*precision_value*" que se le proporciona en cada ejecución y de la precisión que realmente se obtiene. Se han utilizado 6 objetos, 3 de cada color, por lo que los valores de precisión que se pueden obtener son únicamente 0, 0.33, 0.66 y 1.

precision_value	**Precisión obtenida**
-1.0	1.0
-1.0	0.66
0.66	1.0
0.66	0.66
0.33	1.0
0.0	1.0
0.0	0.66

Cuadro 22: Tabla que muestra la precisión que se le indica al programa en cada ejecución y la precisión obtenida realmente.

Al constar de pocos objetos, es difícil ver si realmente se ha mejorado la precisión. En la tabla se han incluido todas las combinaciones de resultados obtenidas en las 20 pruebas realizadas y, como se puede ver, cuanto menor es el valor que se le proporciona, mayor es la alteración de la coordenada x, de manera que se mejore más eficazmente la precisión que se obtenga.

7.3. Medidas de error

Estas pruebas se han realizado tanto en clasificación dinámica como en estática. Es necesario tener en cuenta el tamaño de la cinta transportadora, ya que esta es bastante estrecha y no caben muchos objetos, de esta forma, se han realizado las ejecuciones para 4, 6 y 7 objetos de distinto tamaño.

7.3.1. Clasificación estática

Veremos primero los resultados obtenidos con objetos grandes y a continuación con objetos pequeños.

- **Objetos grandes**
  En primer lugar, vemos la gráfica obtenida para configuraciones de objetos grandes de 4, 6 y 7 unidades.

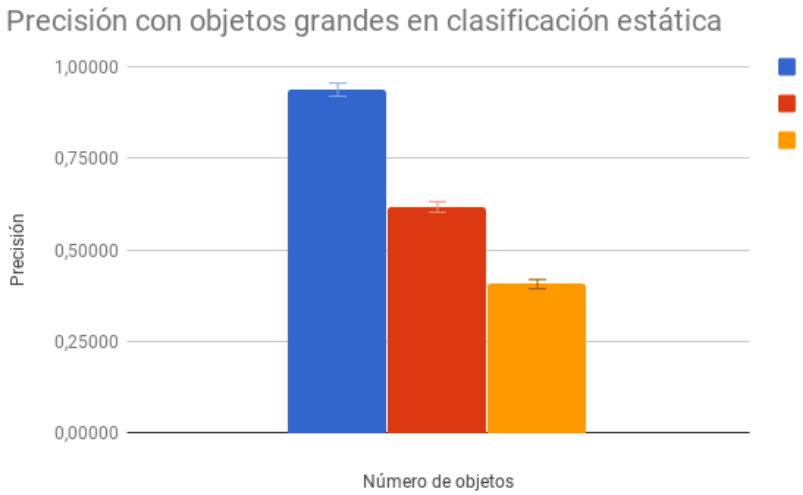

Figura 24: Gráfica que muestra el número de objetos grandes frente a la precisión en clasificación estática, representando la media con una columna y la desviación típica con una barra.

En esta gráfica, se ha representado con un punto la media de los valores de precisión obtenidos en las ejecuciones para 4, 6 y 7 objetos, a la que se ha añadido una barra de error equivalente a la desviación estándar para el conjunto de resultados obtenidos con ese número de objetos. En este caso, no existe una gran variación de precisión en las ejecuciones, por lo que se ajusta bastante a la media. Como era de esperar, cuantos más objetos haya en la mesa, menos precisión se obtiene, puesto que la herramienta de la que se dispone no presenta la longitud adecuada para separar todos los objetos.

- **Objetos pequeños**

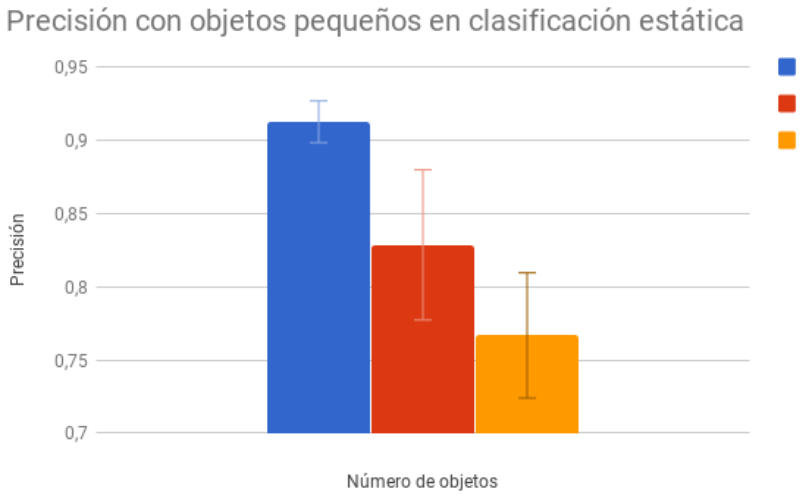

Figura 25: Gráfica que muestra el número de objetos pequeños frente a la precisión en clasificación estática, representando la media con una columna y la desviación típica con una barra.

En este caso, vemos que la media de los valores obtenidos es bastante alta para las 3 configuraciones, ya que al ser más pequeños, la herramienta que en este caso lleva Baxter alcanza a la mayoría de ellos; si nos fijamos en la barra de error, para la configuración de 6 y de 7 objetos es más amplia, por lo que sabemos que ha habido más ejecuciones en las que no se han separado todos los objetos de un mismo color.

7.3.2. Clasificación dinámica

Del mismo modo que en la anterior, veremos primero los resultados con objetos grandes y después con pequeños, esta vez para clasificación en entorno dinámico utilizando una cuña como herramienta.

- **Objetos grandes**

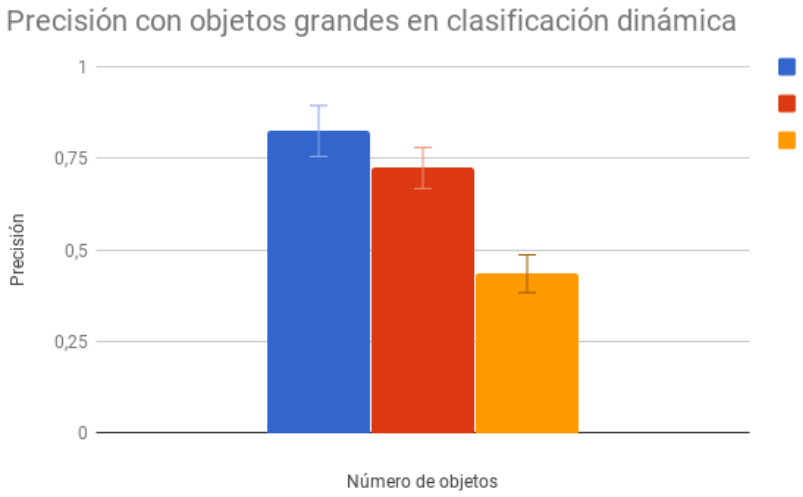

Figura 26: Gráfica que muestra el número de objetos grandes frente a la precisión en clasificación dinámica con cuña, representando la media con una columna y la desviación típica con una barra.

Como muestra la figura, los resultados son bastante parecidos a los que nos encontramos en clasificación estática, aunque en este caso la barra de error o desviación estándar vuelve a ser más amplia. Esto se debe a que los objetos son grandes y de madera, Baxter sujeta la cuña de cartón con el *gripper*, por lo que en algunas ocasiones el peso de los objetos vence la fuerza del *gripper*, del que se resbala la cuña.

- **Objetos pequeños**

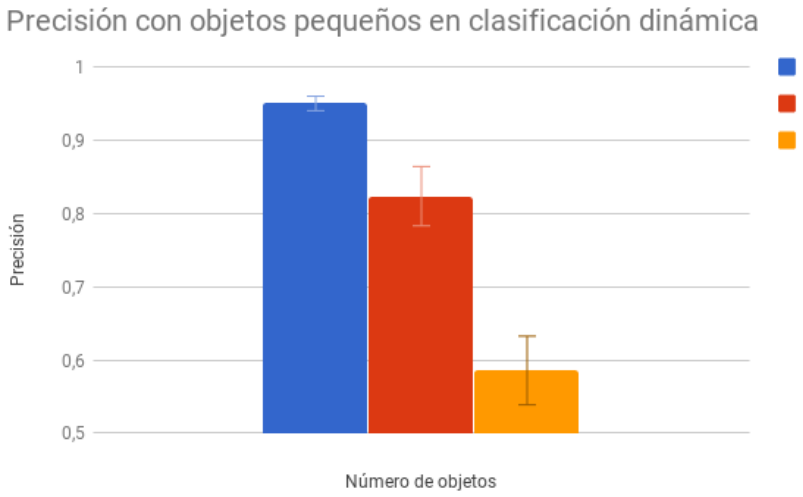

Figura 27: Gráfica que muestra el número de objetos pequeños frente a la precisión en clasificación dinámica con cuña, representando la media con una columna y la desviación típica con una barra.

En la figura 7.3.2 podemos ver que la utilización de objetos pequeños permite obtener mejores resultados en este entorno, ya que se consigue una eficiencia de casi 1 para 4 objetos, y para 6 y 7 no baja de 0.5.

En definitiva, con los objetos pequeños se obtiene una mayor eficiencia en ambas clasificaciones, aunque esto se puede deber a las herramientas utilizadas, ya que no son las comunes que se utilizarían en una fábrica. Con las adecuadas se podrían obtener resultados preferentes a los mostrados en cuanto al peso o el área que ocupan en la mesa los objetos.

8. Conclusión

Para concluir, se va a realizar una comparativa entre los objetivos fijados al principio del proyecto y los resultados logrados a lo largo de este. Finalmente, se describirán las capacidades adquiridas y algunos proyectos futuros que se proponen a partir del desarrollo de este.

La sección de Introducción (1) dio una visión general sobre los objetivos de este trabajo, en él se han utilizado herramientas como MoveIt!, los tópicos y servicios de ROS, y las librerías de OpenCV para implementar la visión por computador.
El objetivo principal de este proyecto (sección 1.3) consistía en hacer que el robot Baxter clasificase objetos por color y posición en un entorno dinámico. Para lograrlo, se ha pasado por distintas fases ya nombradas anteriormente, como la adaptación de un programa de *pick and place* del simulador Gazebo para la familiarización con el entorno, la clasificación de objetos en entorno estático y distintas posibilidades de realizar la segmentación en entorno dinámico. Así, con estos objetivos fijados se ha utilizado un desarrollo en espiral (sección 3) como metodología seguida para la planificación de tareas, lo que ha permitido mejorar continuamente el trabajo realizado.

Todos estos programas se han puesto a prueba, como veíamos en la sección 7, en la que se ha estudiado la precisión, velocidad y capacidad de procesamiento en estas funciones, por parte de Baxter y de las distintas herramientas nombradas. Además, se ha implementado un módulo de robótica adaptativa, que permite a Baxter realizar la mejor aproximación a partir de la precisión adquirida en la ejecución anterior.

En cuanto a los aspectos éticos, aunque en muchas ocasiones se piense que la automatización de tareas acaba con muchos puestos de trabajo, con las ventajas ya vistas en la sección 1.5.1, resulta de gran interés el intentar que esas características idílicas generen beneficios a nuestro favor.

En general, los resultados obtenidos en este proyecto pueden resultar útiles para proyectos futuros; por ello, la implementación y el contenido de este trabajo será abierto y estará disponible en *GitHub* [34] y en la página web del laboratorio [35].

8.1. Competencias adquiridas

A lo largo de este proyecto, se han adquirido las siguientes capacidades:

- **Familiarización con el entorno ROS.** ROS es un sistema operativo distribuido, en el que se ejecutan varios nodos en paralelo, controlados

por tópicos, servicios y mensajes. En este proyecto se ha realizado un estudio de todos sus módulos a través de la documentación que proporciona para conseguir controlar el robot.

- **Librería OpenCV.** Esta es la librería utilizada en el módulo de visión por computador. Para la realización de este proyecto ha sido necesario el estudio de variables y funciones que esta librería proporciona.
- **MoveIt!.** Esta plataforma ha permitido realizar los movimientos correspondientes para separar los objetos. Gracias a su uso se han conseguido movimientos más rectos y simples que los que se conseguían con ROS.

8.2. Trabajos futuros

El resultado de las pruebas de este trabajo no se ha obtenido en su totalidad mediante código; al concluir una ejecución, sería una mejora significativa que Baxter pudiese autoevaluarse a sí mismo, conociendo los objetos que ha separado bien, los que no ha movido y los que ha clasificado mal.

Otra posible extensión de este trabajo consistiría en realizar un *pick and place* en entorno dinámico.

Otra aproximación podría apoyarse en utilizar piezas de otros tamaños, formas y colores, quizás, realizando una clasificación en un entorno real con aceitunas o tomates, entre otras frutas y verduras.

La velocidad de la cinta transportadora podría regularse automáticamente, teniendo en cuenta la velocidad de ejecución de las trayectorias, de forma que se maximice la precisión de la clasificación de las piezas.

9. Anexo. Algoritmos de clustering

En este anexo, se explicarán los algoritmos de clustering y, en concreto el algoritmo K-means [36], que es el algoritmo utilizado en este proyecto.

El clustering es un método de aprendizaje no supervisado. La función de los algoritmos de clustering consiste en agrupar en conjuntos de datos una información. Dado un conjunto de datos, se utiliza un algoritmo de clustering para clasificar cada uno de los puntos en un grupo específico. Cada grupo contiene un grupo de objetos con características similares entre ellos.

Para este proyecto, se utiliza el algoritmo de clustering K-means.

9.1. Algoritmo K-means

El algoritmo K-means (K-Medias en español) consiste en un tipo de aprendizaje no supervisado, es decir, se ajusta a las observaciones sin conocer los datos. Este algoritmo se encarga de encontrar K clústers en los datos proporcionados, siendo K una entrada.

9.1.1. Cómo funciona

Este algoritmo toma dos entradas, K (el número de clústers deseados) y un conjunto de datos. Para empezar, sitúa K centroides en posiciones aleatorias, y, para cada punto de la entrada encuentra el centroide más cercano y lo asigna al clúster perteneciente a ese centroide. Una vez están todos los puntos identificados y clasificados, para cada clúster se calcula un nuevo centroide que consistirá en la media de todos los puntos asignados a él, calculará de nuevo qué puntos pertenecen a esos clústers y parará cuando se mantengan los puntos incluidos en estos.

9.1.2. Por qué elegir K-means

Este algoritmo es de orden O(n). Es bastante rápido, ya que lo único que hace es calcular las distancias entre los puntos y clústers.
Una desventaja aparece a la hora de elegir el parámetro K, pues no siempre es trivial. Además, este algoritmo posiciona inicialmente dos centroides aleatoriamente, y esto puede llevar a obtener diferentes clasificaciones en varias ejecuciones con la misma configuración.

En nuestro caso conocemos el valor de K, puesto que disponemos de dos grupos de objetos. Además, se ha elegido este algoritmo por su simpleza y porque, en contraste con otros algoritmos de clustering, es el más rápido para estas tareas.

Referencias

[1] Statista, "https://www.statista.com/statistics/760207/worldwide-industrial-robotics-market-revenue/," Consultado en junio de 2018.

[2] ResearchAndMarkets, "https://globenewswire.com/news-release/2018/05/01/1493768/0/en/Industrial-Robotics-Market-Global-Forecast-to-2025.html," Consultado en junio de 2018.

[3] P. A. Lasota, G. F. Rossano, and J. A. Shah, "Toward safe close-proximity human-robot interaction with standard industrial robots," in *Automation Science and Engineering (CASE), 2014 IEEE International Conference on*, pp. 339–344, IEEE, 2014.

[4] Z. Ju, C. Yang, and H. Ma, "Kinematics modeling and experimental verification of baxter robot," in *Control Conference (CCC), 2014 33rd Chinese*, pp. 8518–8523, IEEE, 2014.

[5] M. Heufekes, "https://github.com/cvr-lab/baxter_pick_and_place," Consultado en mayo de 2018.

[6] A. Currie, "https://web.archive.org/web/20060718024255/http://www.faculty.ucr.edu/~currie/roboadam.htm," Consultado en Mayo de 2018.

[7] MIT, "https://people.csail.mit.edu/brooks/," Consultado en junio de 2018.

[8] T. S. Tadele, T. de Vries, and S. Stramigioli, "The safety of domestic robotics: A survey of various safety-related publications," *IEEE robotics & automation magazine*, vol. 21, no. 3, pp. 134–142, 2014.

[9] I. Brown, "Human factors: the journal of the human factors and ergonomics society," *Driver Fatigue*, vol. 36, pp. 298–314, 1994.

[10] C. Reardon, H. Tan, B. Kannan, and L. DeRose, "Towards safe robot-human collaboration systems using human pose detection," in *Technologies for Practical Robot Applications (TePRA), 2015 IEEE International Conference on*, pp. 1–6, IEEE, 2015.

[11] RobotWorx, "https://www.robots.com/faq/how-much-do-industrial-robots-cost," Consultado en Mayo de 2018.

[12] "http://wiki.ros.org/ROS/Introduction," Consultado en Mayo de 2018.

[13] "http://www.ros.org/about-ros/," Consultado en Mayo de 2018.

[14] R. Robotics, "https://www.rethinkrobotics.com/baxter/," Consultado en Mayo de 2018.

[15] R. Robotics, "http://sdk.rethinkrobotics.com/wiki/Robot," Consultado en Mayo de 2018.

[16] R. Robotics, "http://sdk.rethinkrobotics.com/wiki/Camera_Control_Tool," Consultado en Mayo de 2018.

[17] R. Robotics, "http://sdk.rethinkrobotics.com/wiki/Arms," Consultado en Mayo de 2018.

[18] C. U. P. Steven M. Lavalle, "http://planning.cs.uiuc.edu/node102.html," Consultado en Mayo de 2018.

[19] "https://moveit.ros.org/," Consultado en Mayo de 2018.

[20] "http://moveit.ros.org/documentation/faqs/," Consultado en Mayo de 2018.

[21] "http://wiki.ros.org/rviz," Consultado en Mayo de 2018.

[22] R. Wiki, "http://wiki.ros.org/urdf," Consultado en Mayo de 2018.

[23] R. Robotics, "http://sdk.rethinkrobotics.com/wiki/URDF," Consultado en Mayo de 2018.

[24] J. Lentin, "Mastering ros for robotics programming," Consultado en mayo de 2018.

[25] B. W. Boehm, "A spiral model of software development and enhancement," *Computer*, vol. 21, no. 5, pp. 61–72, 1988.

[26] M. Vega, "https://lsi.ugr.es/~mvega/docis/casos%20de%20uso.pdf," Consultado en mayo de 2018.

[27] S. Khanduja, "https://github.com/saurabheights/IPExperimentTools/blob/master/AnalyzeHSV/hsvThresholder.py," Consultado en mayo de 2018.

[28] O. Doc, "https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html," Consultado en mayo de 2018.

[29] O. Doc, "https://docs.opencv.org/2.4/doc/tutorials/imgproc/erosion_dilatation/erosion_dilatation.html," Consultado en mayo de 2018.

[30] Gazebo, "http://gazebosim.org/tutorials?cat=guided_b&tut=guided_b1," Consultado en mayo de 2018.

[31] D. M. Powers, "Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation," 2011.

[32] L. Thatcher, "How can precision-recall curves help evaluate your model?," 2018.

[33] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[34] C. M. G. López, "https://github.com/cristgl/TFG," Junio de 2018.

[35] C. Lab, "http://www.ugr.es/~cvrlab/projects/baxter_conveyor/baxter.html," Junio de 2018.

[36] J. A. Hartigan and M. A. Wong, "Algorithm as 136: A k-means clustering algorithm," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 28, no. 1, pp. 100–108, 1979.

*