TRABAJO FIN DE MASTER

PLANIFICACIÓN DE TAREAS PARA LA COSECHA SELECTIVA CON UN SISTEMA ROBÓTICO DE MANIPULACIÓN DUAL

TRABAJO FIN DE MÁSTER
PARA LA OBTENCIÓN DEL
TÍTULO DE **MÁSTER EN AUTOMÁTICA Y ROBÓTICA**

JUNIO **2023**

Carlos Ferreira González

DIRECTOR DEL TRABAJO FIN DE MASTER:
Roemi Emilia Fernández Saavedra

INDUSTRIALES
ETSII | UPM

POLITÉCNICA

"No he fracasado, sino que he descubierto 1000 maneras de no fabricar una bombilla."

- Thomas Edison

AGRADECIMIENTOS

Me gustaría agradecer al Centro de Automática y Robótica, y en especial a mi tutora, Roemi Fernández, por la oportunidad de realizar este trabajo Fin de Máster con ellos, participando activamente en el proyecto PoC-ROBOCROP.

Poder trabajar en un proyecto real dentro del mundo de la investigación ha sido una experiencia muy enriquecedora a nivel profesional. Creo que contribuir a mejorar la calidad de vida de las personas a través de la robótica es la mejor forma posible de fusionar mis gustos y los conocimientos adquiridos durante mis años de estudio, y ha sido en parte posible gracias a este proyecto.

Por último, agradecer a mi círculo cercano de familia y amigos por acompañarme durante los últimos 6 años de formación y estudios. Todos y cada uno de ellos han ayudado y aportado su granito de arena para formar la persona que soy hoy en día, y no sería el mismo sin ellos.

GRACIAS

RESUMEN

El auge de la robótica y la inteligencia artificial en los últimos años ha revolucionado todos los sectores de la industria. Las mejoras tecnológicas de sensores y actuadores junto a los progresos en sistemas inteligentes de toma de decisiones suponen un entorno perfecto para la creación de sistemas autónomos capaces de operar en cualquier ámbito de la vida humana.

La agricultura es uno de los campos con mayor potencial de aplicación para estas tecnologías. Se trata de un entorno muy poco automatizado, pero con un gran impacto en la calidad de vida de cada país. Esta situación genera un gran interés por parte del mundo científico, que se puede ver reflejado en el aumento de líneas de investigación que centran sus esfuerzos dentro de este sector.

En este contexto, surge la prueba de concepto PoC-ROBOCROP. Se trata de un proyecto financiado por la Agencia Estatal de Investigación, cuyo objetivo es desarrollar un prototipo funcional de un robot manipulador de dos brazos para la recolección selectiva. Este Trabajo Fin de Máster se centra en el diseño, desarrollo y validación de un sistema inteligente de toma de decisiones a alto nivel que se implemente en el sistema robótico propuesto.

A través del entorno de trabajo ROS (*Robot Operating System)* y herramientas dedicadas de visión por computador como *OpenCV*, se implementan algoritmos encargados de organizar la recolección selectiva en un entorno no controlado. Estos algoritmos consideran diferentes acciones, como la recolección simultánea con dos brazos o la recolección de frutas parcialmente ocluidas mediante hojas, buscando siempre realizar acciones y movimientos eficientes.

Asimismo, se han desarrollado diferentes algoritmos que utilizan distintos parámetros de control extraídos de la información proporcionada por el sistema multisensorial del robot. De esta forma, se pueden realizar pruebas para comprobar la eficiencia de cada uno de ellos en diferentes situaciones. La última fase de desarrollo se ha orientado a asentar las bases y requisitos necesarios para la futura implementación de un sistema de toma de decisiones basado en *Machine Learning*, mediante aprendizaje por refuerzo.

Al final de la memoria, también se detalla el proceso de integración del planificador con el resto de la arquitectura de software del sistema, así como con otros avances paralelos de la línea de investigación general. En esta fase del desarrollo, cobran especial importancia las características de modularidad y adaptabilidad de la arquitectura de software implementada.

Por último, se realiza un breve análisis sobre el impacto social, económico y medioambiental del proyecto.

Palabras clave: planificación de tareas, robótica agrícola, inteligencia artificial, arquitectura de software, ROS, robótica manipuladora, manipulación dual.

Códigos UNESCO:

- 3304.91 Robótica
- 3304.96 Desarrollo de software
- 3103.93 Visión Computacional aplicada a la Agricultura

ABSTRACT

Recent progress in the fields of robotics and artificial intelligence has revolutionized the industrial sector. The technological evolution of sensors and actuators, alongside the improvement in intelligent decision-making systems, has created a perfect environment for the growth of autonomous robotics systems capable of operating under any environment.

Agriculture represents one of the most promising fields for the application of these technologies and despite the impact it has on human life, it is a sector with very low automatization. This context attracts the attention of scientists who try and provide robotic solutions to common agricultural tasks.

The PoC-ROBOCROP project, funded by the Spanish Research Agency, aims to develop a functional prototype of a dual-armed robot geared towards selective harvesting. This Master's Thesis focuses on the design, development, and validation of an intelligent high-level decision-making system capable of operating the PoC-ROBOCROP prototype.

Using Robot Operating System (ROS) middleware and dedicated computer vision tools such as OpenCV, algorithms are developed to perform organized selective harvesting tasks in unstructured environments. These algorithms study different actions, such as two-arm simultaneous harvesting or partially occluded fruits picking, with the objective of operating under high efficiency levels.

Each algorithm is designed to use a different set of parameters based on the data captured by the robot's sensorial system. Throughout the testing phase of the project, the efficiency of every algorithm was validated under different scenarios. The final phase of development focused on settling the foundations and software requirements for the future implementation of a decision-making system based on Machine Learning's reinforcement methodology.

The project also includes an explanation of the design requirements arising from the need to adapt this work to parallel progress made in the line of research as a whole. During this phase of development, additional emphasis was placed on the modularity and adaptability features of the software architecture to facilitate the expansion of future work.

Lastly, the regulatory framework of the project is analyzed, along with a study of its possible socioeconomic and environmental costs and impact.

Keywords: task planning, agricultural robotics, artificial intelligence, software arquitecture, ROS, robotic manipulator, dual-arm manipulation.

UNESCO Codes:

- 3304.91 Robotics
- 3304.96 Software development
- 3103.93 Computer vision applied to agriculture

LISTA DE ABREVIATURAS

AEI	Agencia Estatal de Investigación
CMOS	Complementary Metal-Oxide-Semiconductor
CSIC	Consejo Superior de Investigaciones Científicas
ECTS	European Credit Transfer and Accumulation System
EOL	End Of Life
FPS	Fotogramas Por Segundo
GPS	Global Positioning System
I+D+i	Investigación, Desarrollo e Innovación
IA	Inteligencia Artificial
IMU	Inertial Measurement Unit
LCD	Loop Closure Detection
LiDAR	Light Detection And Ranging
ODS	Objetivo de Desarrollo Sostenible
ONU	Organización de las Naciones Unidas
OOI	Object Of Interest
PCL	Point cloud Data
RGB	Red, Green, Blue
ROS	Robot Operating System
SLAM	Simultaneous Localization And Mapping
TLR	Technology Readiness Level
TOF	Time Of Flight
URDF	Unified Robot Description Format
XML	Extensible Markup Language
YOLO	You Only Look Once

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

ÍNDICE DE CÓDIGOS

1. INTRODUCCIÓN

1.1. Antecedentes y Justificación

La agricultura ha sufrido numerosas y grandes transformaciones a lo largo de la historia, a medida que se han ido adaptando las innovaciones tecnológicas a sus necesidades. En la actualidad, predomina una agricultura altamente industrializada, aunque existen ciertas diferencias internacionales dependiendo del nivel de desarrollo industrial de cada país. Esta agricultura ha evolucionado hacia la producción a gran escala, para poder satisfacer la alta y creciente demanda de alimento producida por el crecimiento de la población. Sin embargo, este enfoque ha dejado de lado otros aspectos sociales y ecológicos, como la disminución de la calidad del suelo o la utilización masiva de pesticidas.

La adaptación de la robótica a la industria agrícola se realiza desde un enfoque distinto. Aunque también busca aumentar la eficiencia de la productividad, lo hace desde un punto de vista sostenible. La robótica tiene la capacidad de realizar una agricultura de precisión, lo que además permite una mejor colaboración y sincronización con los trabajadores humanos en el campo. Los robots agrícolas buscan cubrir la realización de tareas tales como la recolección de frutos delicados, la inspección de cosechas, la eliminación de malas hierbas o la fumigación de precisión, reduciendo así la carga de trabajo humano en las tareas más arduas.

Todas estas cualidades han provocado que la aplicación de la robótica en la agricultura se convierta en un campo de alto interés y consideración, con un notable aumento en la investigación y el número de publicaciones relacionadas [1]. La robótica, a través de los nuevos sistemas sensoriales y la toma de decisiones inteligente, permite la automatización de tareas complejas que no es posible llevar a cabo con la tecnología agrícola actual. De esta forma, es posible fusionar las últimas tendencias tecnológicas con las actividades más tradicionales de los sectores agrícolas que siguen requiriendo actividades manuales a día de hoy. Este nuevo sistema presenta numerosas ventajas, como la significativa reducción de los costes de producción al disminuir la necesidad de mano de obra, la mejora de la calidad del producto final y la reducción del impacto medioambiental [2].

Sin embargo, a pesar de que la robótica se ha aceptado e integrado por completo en otros sectores de la industria, en el mundo agrícola todavía se encuentra en plena fase de desarrollo y está lejos de alcanzar su máximo potencial. Existen todavía numerosos desafíos como la necesidad de desarrollar tecnologías más precisas y adaptadas a las distintas condiciones del campo y de cultivos especializados, así como la necesidad de educar y capacitar a los trabajadores para que puedan colaborar eficazmente con los robots.

En este contexto, es especialmente importante el desarrollo de sistemas de toma de decisiones inteligentes para robots agrícolas que permitan una recolección selectiva de frutos en función del entorno que percibe el robot en cada momento. Para lograr esto, es necesario una planificación eficiente de tareas a alto nivel. Las nuevas tendencias y capacidades de la inteligencia artificial, así como las últimas mejoras en los sistemas de percepción robóticos, constituyen la base idónea sobre la que centrar el desarrollo de esta línea de investigación.

La motivación principal de este trabajo es utilizar todas estas nuevas herramientas para desarrollar un sistema de toma de decisiones inteligente para un robot de manipulación dual, con capacidad para detectar y recolectar frutos cuando se encuentran en el momento preciso de recolección. En definitiva, este trabajo busca contribuir al avance de la robótica agrícola y a su potencial para mejorar la sostenibilidad, la eficiencia y la calidad de la producción agrícola.

1.2. Objetivos

El objetivo de este trabajo consiste en desarrollar un software encargado de la planificación de tareas a alto nivel de un sistema robótico manipulador dual. El propósito del robot es llevar a cabo la recolección de frutas maduras de una manera eficiente e inteligente, imitando siempre que sea conveniente, la recolección humana. Las frutas seleccionadas para el estudio han sido berenjenas.

Para acometer esta tarea, el sistema robótico cuenta con sensores inteligentes que aprovechan las últimas tecnologías para obtener una gran cantidad de datos del entorno. Estos datos deben servir como entrada al algoritmo de planificación de tareas. El software desarrollado debe llevar a cabo un análisis de los datos, extraer aquellos que se consideren relevantes para realizar una toma de decisiones inteligente y crear las instrucciones que deben ser enviadas a los programas y componentes de bajo nivel del robot, para que sean ejecutados.

Este trabajo forma parte de la línea de investigación que se ha realizado dentro del proyecto ROBOCROP. El proyecto cuenta con financiación de la Agencia Estatal de Investigación y ha sido desarrollado desde el Centro de Automática y Robótica, institución dedicada a la investigación y cuya titularidad se comparte entre el CSIC (Consejo Superior de Investigaciones Científicas) y la Universidad Politécnica de Madrid. Aunque el proyecto original finalizó en 2020, la línea de investigación continúa a través de una prueba de concepto llamada PoC-ROBOCROP [3]. El objetivo de este nuevo proyecto consiste en implementar un prototipo real que permita alcanzar un grado de madurez de la tecnología categorizado como "TRL7", por sus siglas en inglés *Technology Readiness Level*.

Por tanto, debido a la naturaleza y amplitud del proyecto, este trabajo se ha realizado en paralelo con otras investigaciones centradas en el avance de distintas secciones del sistema robótico. Algunas de las que más afectan al progreso de este desarrollo son la mejora y migración del sistema de visión por computador a otro entorno de trabajo o el desarrollo y prueba de planificadores de trayectorias que permitan implementar las tareas conferidas. Esta metodología de trabajo, aunque acelera el avance general del proyecto, condiciona parcialmente el desarrollo realizado individualmente y que ha sido reflejado en esta memoria.

La integración con estos trabajos adquiere, en consecuencia, un gran peso en el progreso realizado, en especial en las fases finales de pruebas donde se requiere comprobar el funcionamiento del sistema en su conjunto. Es necesario dotar al software desarrollado de una consistente modularidad y adaptabilidad, con el objetivo de facilitar y simplificar el proceso de integración. Para ello, es preciso realizar cambios en la arquitectura de software existente, reemplazando aquellos elementos que reduzcan la versatilidad global del robot.

A partir de estos objetivos se pueden considerar los siguientes requerimientos de alto nivel:

- Realizar una migración a ROS (*Robot Operating System*) del sistema de toma de decisiones existente, desarrollado en MATLAB. El software diseñado debe emplear las funcionalidades y opciones ofrecidas por ROS para elaborar un programa modular y escalable.
- Diseñar y desarrollar algoritmos de toma de decisiones adicionales que conformen un planificador de tareas a alto nivel, de forma que el robot sea capaz de realizar la cosecha selectiva de frutas en función de los datos percibidos por su sistema multisensorial.
- Los algoritmos desarrollados deberán realizar las siguientes subtareas:
    - Determinar el orden de recogida de las frutas capturadas por el sistema multisensorial.

    - Determinar el tipo de acción que se debe llevar a cabo para realizar la recolección. Es decir, elegir entre recolección secuencial de uno de los brazos, simultánea o colaborativa entre ambos brazos.
    - Crear una instrucción que pueda ser procesada por el planificador de trayectorias y que contenga toda la información necesaria para realizar la tarea encomendada.
- Elaborar un programa que permita visualizar, mediante una interfaz gráfica, el procedimiento utilizado en los algoritmos. Esta herramienta se usará para comprobar el correcto funcionamiento del software elaborado.
- Integrar el planificador de tareas con el sistema robótico completo. Esto incluye no solamente adaptarlo a la arquitectura de software ya creada, sino fusionarlo con los avances realizados de forma paralela mientras se produzca el desarrollo del trabajo.
- Utilizar tanto herramientas virtuales de simulación, como el robot físico para realizar pruebas y registrar los resultados.

1.3. Estructura del documento

Este documento tiene como objetivo recopilar todo el trabajo desarrollado y reflejar los avances que se han realizado en la línea de investigación. A lo largo del mismo, se muestran los cambios y nuevas funcionalidades que se han realizado para cumplir los objetivos establecidos. La memoria se ha estructurado en los siguientes apartados:

- **Introducción:** conforma este primer capítulo del documento. Incluye una pequeña introducción sobre la robótica en el sector agrícola y su impacto potencial en la sociedad, que sirve como motivación para este trabajo. También se incluyen los objetivos del proyecto.
- **Estado del arte:** el segundo capítulo incluye una breve introducción al concepto de planificación de tareas, así como un estudio y recopilación de algunos de los proyectos más influyentes a nivel europeo relacionados con la robótica agrícola. Uno de estos proyectos es CROPS, que conforma los antecedentes de esta línea de investigación. Aquí se detallan algunos de sus aspectos más importantes y sus objetivos. Finalmente, se explica de forma resumida algunos de los conceptos teóricos que pueden ayudar a comprender el funcionamiento del entorno de trabajo utilizado (ROS).
- **Metodología:** este capítulo comienza con un análisis del estado inicial del proyecto, que sirve como base para comprender el punto de partida del trabajo. A continuación, se comienza a describir todo el desarrollo de software realizado, incluyendo justificaciones del procedimiento utilizado y algunos ejemplos con los que se pretende ayudar a comprender la lógica que hay detrás de cada una de las decisiones tomadas.
- **Pruebas y resultados:** el objetivo de esta parte de la memoria es registrar las pruebas más relevantes que han sido realizadas para comprobar el funcionamiento del sistema robótico. Se incluyen algunas pruebas individuales que afectan únicamente al software desarrollado y otras pruebas en las que se muestra el comportamiento general del sistema.
- **Conclusiones y líneas futuras:** en este penúltimo capítulo se redactan las conclusiones alcanzadas durante el trabajo. Utilizando los resultados obtenidos en el anterior capítulo, se realiza un análisis de los objetivos logrados y se mencionan algunas de las líneas de trabajo futuras más interesantes que pueden ser exploradas.

- **Presupuesto y planificación:** a lo largo de este capítulo se analiza el presupuesto de este trabajo. También se incluye una descripción de la planificación temporal seguida durante el desarrollo del proyecto.
- **Aspectos legales, sociales y económicos:** finalmente, se realiza un breve análisis sobre el marco regulador y estándares vigentes que afectan al proyecto, así como una evaluación del impacto social y medioambiental del mismo. También se tratan temas éticos y se realiza un pequeño estudio económico del proyecto.

Al final del documento, se incluye un apartado de Anexos donde se recopila un diagrama de Gantt en relación con la planificación temporal, pero que no se ha incluido dentro del capítulo correspondiente debido a su extensión y para evitar romper la continuidad de la memoria.

2. ESTADO DEL ARTE

En el comienzo de este capítulo, se realiza una introducción a los conceptos de planificación de tareas y toma de decisiones inteligentes en robótica, relacionando dichos conceptos con los objetivos de este trabajo.

A lo largo del capítulo, también se desarrolla y detalla el nivel actual de robotización dentro del sector agrícola, exponiendo los principales avances que se han producido en los últimos años y las líneas de investigación que se encuentran abiertas a día de hoy. También se exponen varios de los proyectos europeos que más características comparten con la línea de investigación en la que se centra este Trabajo Fin de Máster.

En la parte final del capítulo, se realiza una breve introducción sobre el software y las tecnologías involucradas en el trabajo, de forma que se asienten las bases para comprender el desarrollo que se ha llevado a cabo.

2.1. Planificación de Tareas en Robótica

La planificación de tareas y los algoritmos de toma de decisiones son de gran relevancia dentro de la robótica. El concepto de robot está intrínsecamente asociado con la idea de autonomía, entendiendo este concepto como la capacidad para realizar acciones sin la intervención humana. Para ello, todo robot debe de estar dotado tanto de sensores que le permitan percibir el entorno que le rodea, como de un sistema inteligente que le ayude a decidir qué acción debe realizar para cumplir su tarea.

Dentro del campo de la inteligencia artificial, el concepto de *agente* recibe justo esta definición, un ente que actúa en función de las observaciones o datos que recoge del entorno [4]. Un robot, por tanto, es un *agente* que debe realizar una tarea a través de sus actuadores. La naturaleza de cada tarea está ligada a las características del robot. Así, los robots móviles se centran en desplazarse por su entorno, mientras que las tareas de los manipuladores implican la interacción con otros objetos. La mecánica de observación y actuación con el entorno, en cambio, es común a cualquier robot. Determinar qué acción es la correcta en cada situación es el problema al que se enfrentan los algoritmos de toma de decisiones, y es el campo de estudio central de la inteligencia artificial.

La complejidad de estos algoritmos es muy variada. Se pueden encontrar sistemas que utilizan una única variable para determinar qué acción van a realizar, mientras que otros calculan todos los posibles estados futuros para elegir con certeza cuál es la mejor solución. La elección de la complejidad de estos algoritmos, está ligada a la naturaleza del problema que se pretende resolver. Algoritmos sencillos permiten tiempos de respuesta muy altos, mientras que algoritmos más profundos realizan acciones más óptimas a cambio de tiempos de respuesta mayores. El balance entre estos dos o más parámetros debe ser analizado y estudiado para cada situación.

No obstante, cuando el problema al que debe hacer frente un sistema robótico es complejo, las acciones que debe realizar para resolverlo son, a su vez, complejas, llegando a requerir en muchas ocasiones, una secuencia larga de acciones para lograr el objetivo marcado. En este contexto, aparece el concepto de planificación de tareas. Un planificador de tareas es el encargado de convertir las especificaciones de una tarea expresadas en alto nivel, en especificaciones concretas que sean interpretadas por actuadores y manipuladores [5].

Véase el ejemplo de un robot móvil. Se puede definir una tarea de forma tan sencilla como "moverse del punto A al punto B". El planificador tomará esta tarea y la convertirá en una secuencia de instrucciones que serán enviadas a los controladores de bajo nivel para llevarla a cabo. Por ejemplo, "avanza 2 metros en el eje X". La complejidad de la tarea a realizar y, por tanto, la cantidad de instrucciones que serán necesarias para cumplir la tarea están ligadas al número de restricciones que se impongan a la misma. Si se añaden restricciones a la tarea considerada en el ejemplo, de forma que resulte en "moverse del punto A al punto B sin chocar con ningún obstáculo, pasando previamente por los puntos C y D y en el menor tiempo posible", se puede observar que aumenta la dificultad de la resolución del problema.

De la misma forma, si se aplica este concepto a robots manipuladores, se pueden diseñar tareas sencillas como "recolecta una fruta". O tareas mucho más complejas como "recolecta todas las frutas maduras de la escena minimizando el tiempo de recolección". Para resolver este problema, el planificador de tareas debe aplicar criterios de valoración para saber qué es una fruta "madura" y, además, considerar diferentes opciones para calcular cuál de todas ellas es la que resuelve la tarea en el menor tiempo posible. Y seguidamente, transformar la solución en un conjunto de acciones que pueda interpretar el robot manipulador.

Es intuitivo pensar que, los algoritmos utilizados en la toma de decisiones de la primera tarea pueden ser mucho más sencillos que los necesarios para encontrar una solución óptima en la segunda tarea. A medida que aumenta el número de restricciones y, por tanto, la complejidad del problema a resolver, se van requiriendo planificadores de tareas más complejos o "inteligentes".

En este punto es conveniente hablar de los conceptos de Inteligencia Artificial. Un algoritmo de inteligencia artificial es todo aquel que permite tomar decisiones y realizar acciones a partir de datos del entorno, es decir, cualquier algoritmo que se utilice en un robot autónomo. La tarea "recolecta una fruta" que se ha utilizado anteriormente de ejemplo, puede ser resuelta por algoritmos jerárquicos sencillos de inteligencia artificial. Estos algoritmos siguen el mismo procedimiento secuencial todas las veces que se utilizan, y son útiles en la aplicación de tareas relativamente simples que utilizan pocas variables de control. En cambio, cuando se requiere tener muchos parámetros en cuenta, estos algoritmos no son suficiente.

Ante este tipo de tareas más complejas, es más conveniente "enseñar" al programa cómo actuar que diseñar algoritmos jerárquicos que tengan en cuenta una gran cantidad de parámetros simultáneamente. En concreto, el aprendizaje por refuerzo aborda el problema de "cómo un agente autónomo, capaz de percibir y actuar sobre su entorno, puede aprender a elegir acciones óptimas para alcanzar su objetivo" [6]. Existen otros algoritmos populares de *Machine Learning*, como el aprendizaje supervisado y no supervisado, pero su aplicación es distinta al contexto de este trabajo.

Mediante técnicas de aprendizaje por refuerzo, se puede "enseñar" a un programa cuál es la mejor forma de actuar. El procedimiento consiste en iterar una gran cantidad de veces sobre la misma situación, realizando acciones ligeramente distintas que en el experimento anterior. A continuación, se valora la efectividad de las acciones realizadas mediante funciones de recompensa y evaluación. De esta forma, el sistema puede ir comprobando qué acciones producen un mayor beneficio y cuáles no. La configuración de estos programas, no obstante, es compleja. El diseño de una mala función de evaluación o recompensa puede provocar que el robot realice acciones no deseadas para conseguir su objetivo. En el equilibrio de estas funciones de evaluación, está la clave para conseguir robots eficientes y correctamente entrenados.

En resumen, la planificación de tareas y toma de decisiones tiene una importancia extremadamente alta en el mundo de la robótica. Se puede considerar estos sistemas como el "cerebro" de un robot, ya que son los encargados de determinar cómo va a actuar ante cada situación.

Además, con el auge y rápido crecimiento de la Inteligencia Artificial, el potencial de estas áreas se ha disparado en los últimos años, permitiendo el desarrollo de robots cada vez más complejos capaces de resolver algunas tareas de manera más óptima y eficiente que los humanos.

2.2. Robótica en el sector agrícola

Como se ha mencionado en la Introducción, la robótica dentro del sector agrícola se encuentra en sus primeras fases de desarrollo, dada la complejidad tecnológica asociada y la rentabilidad aún limitada de su adopción e implementación. No obstante, debido a su gran perspectiva de futuro y sus beneficios económicos, sociales y medioambientales, existen una gran cantidad de proyectos enfocados a optimizar y perfeccionar las funcionalidades de la robótica para que pueda ser implementada en la agricultura cuanto antes.

La Unión Europea es uno de los principales promotores dentro del sector, liderando y financiando numerosas líneas de investigación. A continuación, se describen algunos de los principales proyectos europeos que se están llevando a cabo de forma internacional, y donde trabajan conjuntamente centros de investigación, universidades y empresas privadas.

2.2.1. Robs4Crops

Este proyecto, coordinado por la Universidad de Wageningen en los Países Bajos, y en el que participan hasta seis países diferentes, entre ellos España, es uno de los más grandes a nivel europeo. Comenzó en 2021 y tiene una duración estimada de 4 años. Su principal característica es que no solamente aborda la adopción de la agricultura robotizada desde un punto de vista tecnológico, si no que utiliza normas y prácticas agrícolas para enfrentarse también al reto organizativo que supone la integración de la robótica en un sector nuevo. Su objetivo es "crear y proporcionar un sistema flexible y modular, totalmente autónomo, listo para pruebas comerciales a gran escala" [7].

Robs4Crops atribuye la falta de integración de la robótica en la agricultura a motivos principalmente técnicos. Los robots agrícolas no alcanzan su potencial debido a que son utilizados como unidades independientes, en lugar de formar parte de un sistema robótico completo e innovador, capaz de gestionar y coordinar múltiples unidades robóticas de forma simultánea. Pero también destaca que los robots agrónomos no pueden encajar con las prácticas y estándares agrícolas vigentes. Por tanto, propone una solución en la que las herramientas y vehículos actuales sean mejorados y actualizados, para que puedan funcionar con los robots agrónomos existentes como parte de un único sistema global.

Este nuevo sistema agrícola ofrece la oportunidad de desarrollar nuevos modelos de negocio, a la vez que se desarrollan las mejoras tecnológicas que lo harían posible. El objetivo final de este proyecto es disponer de sistemas robóticos terminados para la protección de cultivos, y que puedan ser probados a escala en 4 países diferentes de la Unión Europea.

2.2.2. Canopies

Canopies es un proyecto europeo cuyo objetivo principal es establecer un entorno colaborativo entre robots y trabajadores humanos en operaciones de vendimia. Para ello, se pretende desarrollar un equipo de robots agrónomos y logísticos que sean capaces de integrar su trabajo con

los agricultores [8].

El proyecto cuenta con la participación de numerosas universidades y empresas internacionales. En particular, Pal Robotics, una de las empresas españolas más grandes de robótica de servicio, es uno de los colaboradores activos del proyecto. Es la encargada de fabricar uno de los robots duales adaptado para prácticas agrícolas utilizando su modelo comercial, el robot TIAGo, con nuevos efectores terminales y herramientas especializadas.

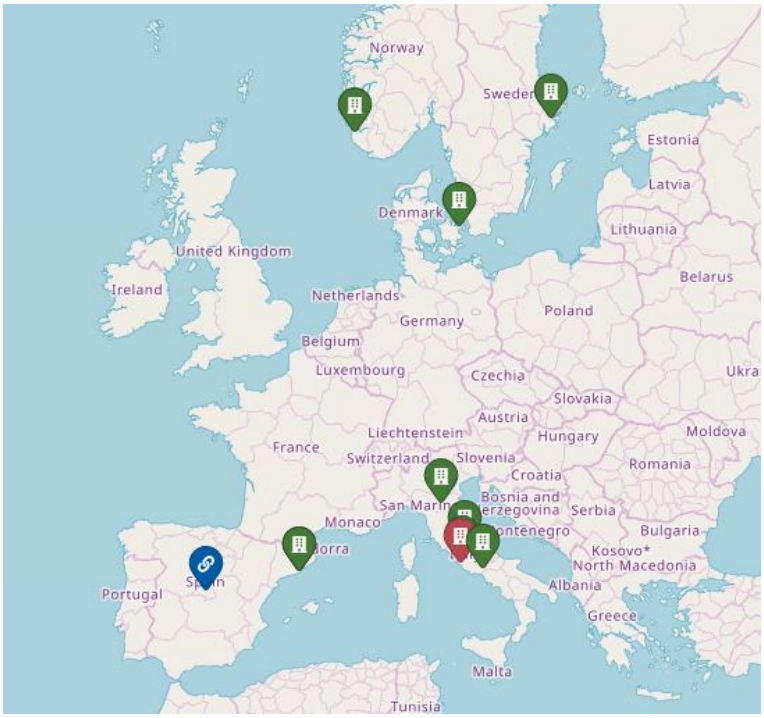


Figura 2.1: Mapa de colaboración en el proyecto Canopies (Las universidades colaboradoras en verde. La universidad coordinadora en rojo. Entidades asociadas en azul) [8].

El proyecto Canopies representa un hito en la agricultura agrónoma, ya que es el primer intento de implementar un modelo colaborativo en la agricultura de precisión para cultivos permanentes [9]. Este proyecto supone un gran desafío de integración, ya que para su correcto funcionamiento, se requiere de un complejo sistema de percepción y comunicación a bajo nivel, sumado a una planificación grupal y compartida a alto nivel, que pueda interpretar las actividades e intenciones humanas y responder en consecuencia.

A todo esto, hay que añadirle las dificultades de un entorno no controlado y cambiante, con situaciones muy variadas en cuanto a iluminación y accesibilidad. A diferencia de los cultivos de campo, como pueden ser los cereales, la recolección de los cultivos permanentes implica un sistema inteligente que sea capaz de obtener el fruto sin dañar el resto de la planta, ya que ésta continúa produciendo después de la primera recolección. Como consecuencia, la producción agrícola en cultivos permanentes requiere de un enfoque más minucioso. Sin embargo, el mayor reto consiste en implementar robots que dispongan de las medidas y protocolos necesarios para garantizar que la colaboración con humanos se produzca siempre en situaciones de máxima seguridad.

Dentro del desarrollo de este proyecto se han realizado grandes avances en numerosos ámbitos, como en el de la manipulación robótica de objetos deformables. La investigación en este ámbito se ha orientado hacia la planificación de acciones de manipulación mediante mapas de camino de espacio latente. Esta técnica, conocida como *Latent Space Roadmaps* en inglés, se basa en

un modelo matemático que permite la representación del espacio de todas las posibles configuraciones del objeto y del robot en un espacio abstracto de menor dimensión, llamado "espacio latente". El espacio latente incluye las características más relevantes de los sensores del robot, normalmente las imágenes captadas por las cámaras. A partir de éste, se puede generar un mapa que sirva para guiar al robot en la planificación de sus movimientos, reduciendo el número de cálculos necesarios. Canopies, realiza esta tarea a través de dos redes neuronales entrenadas conjuntamente que actúan como codificador y decodificador [10]. Este tipo de redes neuronales se denominan *autoencoders*.

Figura 2.2: El robot TIAGo++ en una prueba del proyecto Canopies [11].

Por otro lado, en el control adaptativo para robots cooperativos se han estudiado y considerado las opciones clásicas, que incluyen los controladores pasivos o soluciones basadas en la estimación de la trayectoria humana para anticiparse y reducir el esfuerzo humano. Sin embargo, ambas soluciones por sí solas no son suficientes ya que, o bien no proporcionan la precisión requerida para la tarea, o requieren esfuerzos computacionales demasiado altos.

La solución propuesta es una compleja estrategia de control que se centra en realizar un seguimiento de las cargas compartidas por humanos y robots mientras analiza de forma paralela los comportamientos humanos que se consideran ambiguos, es decir, que no ayudan a la tarea pero tampoco la perjudican [12].

En conclusión, Canopies es uno de los proyectos internacionales con más avances y publicaciones dentro del sector de la robótica agrícola de manipulación dual. La colaboración internacional de importantes universidades y empresas dentro del sector, lo constituyen como uno de los principales proyectos que se encuentran actualmente activos.

2.2.3. Bacchus

Bacchus es otro de los grandes proyectos europeos de robótica agrícola. En este caso, las empresas Robotnik, especializada en la robótica móvil, y Universal Robots, especializada en brazos robóticos de manipulación, trabajan conjuntamente con otras universidades para diseñar, desarrollar y probar un sistema robótico inteligente cuyo principal objetivo es la inspección y recolección selectiva de cosechas con gran valor [13]. Al igual que Canopies, este proyecto se ha centrado en la vendimia, pero pretende imitar las actividades humanas que son necesarias para la recolección de uvas. Es decir, Bacchus ofrece un sistema robótico completamente automatizado que es capaz de operar sin la intervención humana.

Aunque comparte muchos de los desafíos y dificultades que presenta el proyecto Canopies, la forma de afrontarlos es ligeramente distinta. Bacchus apuesta por un sistema de recolección dual, utilizando plataformas robóticas modulares. Para navegar y operar en el entorno hace uso de un avanzado sistema cognitivo que alimenta a un inteligente sistema de toma de decisiones. Además, utiliza herramientas y pinzas fabricadas mediante procesos de fabricación aditiva para ajustarse a la geometría de cada cultivo [14]. Estas características convierten a este proyecto en la línea de investigación activa más similar a la seguida a lo largo de este trabajo.

Figura 2.3: Robot diseñado para el proyecto Bacchus [15].

El problema de la navegación en viñedos representa una gran dificultad para el proyecto. Muchos de los sistemas de navegación utilizados se basan en la captura de imágenes mediante cámaras RGB o la utilización de precisos sistemas de GPS. No obstante, estas alternativas deben descartarse ya que los viñedos son cultivos caracterizados por formar una techo natural con sus ramas. Esto provoca zonas con malas condiciones lumínicas y genera un gran obstáculo para recibir una buena señal GPS que sea capaz de posicionar al robot con una alta precisión. Estas condiciones refuerzan la utilización del SLAM como principal herramienta de navegación y localización. Aunque, esta técnica tampoco está libre de inconvenientes.

Los viñedos son terrenos de gran extensión y con muchas similitudes entre ellos. Los algoritmos de SLAM para grandes extensiones suelen utilizar un sistema de retroalimentación para corregir la estimación de la posición dentro del mapa, que recibe el nombre de *Loop Closure Detection*

en inglés. La retroalimentación utilizada por el SLAM se suele basar en la colocación de balizas cuya posición es conocida o en las mediciones del GPS. La primera opción presenta una ejecución muy costosa y complicada debido a las grandes extensiones del terreno, y la segunda opción ya había sido descartada por mala cobertura en la señal. Por lo tanto, Bacchus presenta una nueva estrategia para la técnica de LCD SLAM que se basa en la segmentación de los troncos de las vides para encontrar rasgos únicos y rastreables que puedan ser utilizados como retroalimentación al sistema. Todo esto es posible gracias a una red neuronal profunda que trabaja junto a funciones de filtrado y funciones probabilísticas [16]. De esta forma, se ofrece una nueva estrategia de SLAM que puede ser utilizada dentro de la robótica agrícola.

Bacchus, también ha realizado grandes avances en el control de brazos robóticos y en la gestión de la identificación de Objetos de Interés (OOI) que se encuentran parcialmente ocluidos. Por una parte, destaca el desarrollo de un sistema de asistencia a la programación por guiado o demostración. Esta técnica de programación requiere que se guíe físicamente al robot, moviendo sus brazos, para realizar la tarea que se desea programar (existen también otras técnicas de demostración por guiado en las que un humano realiza la tarea deseada a modo de ejemplo, pero requieren de complejos sistemas de percepción para capturar el movimiento). Sin embargo, a la hora de guiar dos brazos robóticos de forma simultánea en un entorno complejo, la tarea de programación por guiado adquiere cierta complejidad. Para simplificar el proceso y obtener una duración de enseñanza reducida y una carga física menor, se ha desarrollado un controlador pionero y compartido que utiliza la geometría de la pieza de trabajo, la optimización de la inercia y el espacio de trabajo del robot para asistir en la enseñanza de tareas bimanuales [17].

Por otra parte, han desarrollado un esquema de control para aproximarse a un OOI que se encuentre parcialmente ocluido. Para ello, utilizan una cámara unida al efector final del robot y un algoritmo capaz de identificar qué puntos de la nube de puntos obtenida por la cámara pertenecen al OOI y cuáles no. A continuación, comienza un proceso en el que el efector final del robot se desplaza lentamente, utilizando una trayectoria que le permita revelar las dimensiones del OOI, para después poder realizar la planificación de tareas que sea precisa una vez se ha obtenido toda la información posible. Este método es aplicable a otro tipo de robots humanoides [18].

2.2.4. CROPS

Aunque todos los proyectos mencionados hasta ahora están en fase de desarrollo, los primeros programas de investigación en robótica agrícola se empezaron a desarrollar hace más de una década. CROPS fue un proyecto que se inició en 2010 y tuvo una duración de 4 años. Su objetivo era el desarrollo de una plataforma móvil modular que permitiera instalar en ella diversos sensores y herramientas inteligentes para efectuar tareas agrícolas según las condiciones del entorno [19].

Este proyecto supuso uno de los primeros prototipos europeos que fue diseñado, fabricado y probado en un entorno real controlado. El ambicioso proyecto tuvo un gran alcance y realizó grandes avances en los campos de la detección de frutas y verduras en condiciones óptimas de madurez y en el desarrollo tanto de herramientas especializadas, como de algoritmos capaces de operar en entornos rurales. Utilizó *Robot Operating System* o ROS como *middleware* para desarrollar la arquitectura de software del programa.

Los manipuladores creados se basan en robots de 9 grados de libertad, que pueden ser equipados con herramientas de agarre específicas para manzanas, pimientos o uvas. También se desarrollaron y probaron herramientas de fumigación. El robot fue equipado con sensores extra que se añadieron a los efectores finales para poder realizar detección de corto rango, que permitiera mejorar la

precisión del robot a través de un sistema de control visual.

Los resultados y conclusiones del proyecto se encuentran publicados, y tuvieron gran importancia a la hora de marcar las bases de otras líneas de investigación. En cuanto a los resultados de los algoritmos y sensores utilizados para la detección de objetos de interés, se obtuvieron unos resultados muy prometedores, en especial en las pruebas realizadas con manzanas, donde se obtuvo un porcentaje de éxito de detección del 100 %. Por otro lado, a pesar de los diversos modelos que se desarrollaron para el agarre, los resultados de la cosecha de frutas y verduras no tuvieron una tasa de éxito tan alta. La recolección de pimientos se quedó en tan sólo un 33 % y la de la manzanas en un 72 % [20].

Estos resultados, no obstante, son muy alentadores. Aunque aún queda un gran margen de mejora y mucho trabajo por realizar en el desarrollo de software y hardware especializado, los resultados garantizan que se trata de un campo con un gran potencial. La recolección selectiva de frutos es factible a gran escala, y tiene capacidad para revolucionar el sector agrícola.

2.2.5. ROBOCROP

Finalmente, es necesario hablar del proyecto ROBOCROP. Este proyecto de nivel nacional tenía como objetivo diseñar y evaluar un sistema robótico que estuviese dotado, tanto a nivel de hardware como de software, de las herramientas necesarias para operar de forma completamente autónoma en entornos no estructurados. Para ello, el proyecto proponía "un cambio significativo en las áreas de percepción y manipulación robótica en entornos no estructurados, diseñando e implementado algoritmos de detección y localización robustos en entornos agrícolas y algoritmos de planificación y control basados en el comportamiento y el aprendizaje" [21]. De esta forma, se buscaba replicar las operaciones de recolección que realizan los seres humanos.

El sistema propuesto por ROBOCROP consiste en una unidad robótica que fusiona un robot de manipulación de doble brazo con un robot móvil que le permite desplazarse por el entorno, aunque sólo se encuentra dentro del alcance del proyecto el diseño y desarrollo del robot de manipulación, pero no el control y la navegación del robot móvil. Las pruebas se realizaron con berenjenas como fruto de recolección.

La unidad robótica empleada está integrada por dos brazos robóticos MICO2^{TM}, comercializados por Kinova Technology (Montreal, QC, Canadá). Estos brazos, con una capacidad de carga de 2.1 kg en operación continua, tienen las capacidades suficientes para proporcionar un método fiable de manipulación de las berenjenas [22]. Proporcionan 6 grados de libertad y cuentan con un efector final que consiste en una pinza con tres dedos que permite un agarre máximo de 40 N de fuerza. Sin embargo, estos agarres se utilizan como solución alternativa, mientras se diseña en paralelo un conjunto de efectores específicos para el fruto que se desea recoger. La idea detrás de estos efectores personalizados es que se trate de una herramienta modular, que sea fácilmente intercambiable para que el robot pueda adaptarse a cada tipo de cultivo.

El sistema sensorial se basa en dos cámaras. La primera es una cámara RGB, que proporciona imágenes a color de alta resolución. La segunda es una cámara de tiempo de vuelo o TOF, por su nombre en inglés *time-of-flight*. Esta segunda cámara opera a una mayor tasa de fotogramas por segundo y es la encargada de obtener el mapa de profundidad que se utiliza para generar una nube de puntos del entorno de trabajo.

En cuanto a la arquitectura de software, tal y como se describe en [23], se implementó una arquitectura basada en ROS, y consta de 4 módulos que se encargan de:

- La obtención de las las imágenes de las cámaras.
- La detección y localización las berenjenas en un sistema de coordenadas conocido por el robot.
- La planificación del movimiento.
- El control de los manipuladores.

El módulo encargado de la detección y localización de los frutos se ejecuta en un nodo de MATLAB integrado con ROS. Su principal función es estimar los centroides de posición de las berenjenas en el sistema de coordenadas de los manipuladores y calcular la cinemática inversa de los brazos para obtener las configuraciones de las articulaciones necesarias y comprobar la viabilidad del movimiento. Para ello, cuenta con algoritmos de recolección dual simultánea y un algoritmo encargado de procesar las berenjenas que se encuentran parcialmente ocluidas.

Figura 2.4: Brazos robóticos MICO2^{TM} recolectando berenjenas.

Antes del final de proyecto en 2020, ROBOCROP realizó pruebas en el laboratorio para analizar la efectividad del sistema propuesto. El sistema robótico era capaz de recolectar el 91.67 % de las berenjenas en los escenarios propuestos [23], lo que demuestra un alto grado de validez. El proyecto ha evolucionado ahora hacia PoC-ROBOCROP, que consiste en la prueba de concepto del sistema. Forma parte del programa de "I+D+i" de la Agencia Estatal de Investigación de España, y el objetivo es validar el prototipo en un entorno real, en vez de en un laboratorio. Así, esta prueba serviría para pasar del grado de madurez de la tecnología 4, TRL4, a un grado de madurez TRL7.

El desarrollo realizado a lo largo de este trabajo forma parte de esta prueba de concepto. En este trabajo se pretende añadir un módulo adicional entre la detección y localización y la generación de trayectorias de movimiento, para implementar una planificación de tareas a alto nivel que permita comparar diferentes métodos y elegir aquel que consiga aumentar la eficiencia del sistema. Actualmente, el algoritmo de planificación se ejecuta en el segundo módulo a través de un sistema de toma de decisiones jerárquica. En el siguiente capítulo se realiza un análisis más detallado del sistema para entender el punto de partida del trabajo.

Figura 2.5: El proyecto PoC-ROBOCROP en las Jornadas Nacionales de Robótica y Bioingeniería en la Universidad Politécnica de Madrid.

2.3. Software Utilizado

Durante este epígrafe se realiza una breve introducción acerca del software utilizado durante el proyecto. El objetivo es explicar de forma concisa las bases de funcionamiento de ROS e introducir las librerías necesarias para cumplir con los objetivos del trabajo, a la vez que se estudian los usos y aplicaciones que reciben estas herramientas en los campos de la robótica. De esta forma, se pretende que este apartado sirva a modo de introducción para aquellos que no dispongan de conocimiento previo sobre las herramientas empleadas.

2.3.1. Robot Operating System (ROS)

Se puede definir ROS como un *middleware*, o un entorno de trabajo, de código abierto, diseñado específicamente para la programación y control de robots. Desde su lanzamiento en 2007, se ha convertido en la herramienta principal y más extendida en la investigación y desarrollo de robots a nivel internacional, utilizada en todos y cada uno de los campos de la robótica.

La principal ventaja de ROS reside en su alta modularidad y su amplia biblioteca de paquetes y librerías, lo que ofrece la posibilidad de reutilizar código ya probado y realizar proyectos colaborativos. Además, está dotado de estándares que permiten integrar una gran variedad de sensores y actuadores del mercado. Esto aporta una gran versatilidad y flexibilidad a cualquier arquitectura de software basada en ROS.

Su funcionamiento se basa en una arquitectura *peer-to-peer*, estableciendo una red de conexiones capaz de conectar múltiples sistemas. La información se procesa en los Nodos, que se comunican

entre ellos a través de los *Topics*, mediante un sistema de mensajes propio, simulando así, una arquitectura de grafos.

El funcionamiento principal y más elemental de ROS se puede explicar mediante los siguientes cuatro elementos:

- Nodos: son la unidad básica de procesamiento de información de ROS. Son procesos que realizan una o varias tareas de computación. Los nodos se comunican entre sí enviando mensajes a través de los *topics*. Un programa complejo está constituido por una gran cantidad de nodos diferentes, todos ellos comunicándose entre sí. Esta característica es la que aporta la modularidad. Cada nodo se ejecuta de forma independiente al resto, por lo que pueden combinarse nodos de distintos paquetes instalados con nodos personalizados.
- *Topics*: estos elementos son los buses de información de ROS. Son canales encargados de controlar el intercambio de datos entre los nodos. La comunicación se basa en un sistema de publicador y subscriptor. Un nodo puede subscribirse a tantos *topics* como requiera, recibiendo información de distintas fuentes, al igual que puede publicar mensajes en uno o más *topics*. No existen restricciones respecto al número de nodos que pueden interactuar con cada *topic*. Cada *topic* sólo trabaja con un tipo de mensaje, y no se pueden mezclar distintos formatos de mensajes en el mismo canal de información. Este sistema permite independizar la producción de mensajes de su consumición.
- Mensajes: son simplemente estructuras de datos propias de ROS. Un mensaje puede estar formado por todos los campos o tipos de variable que se requieran, y se pueden incluir mensajes dentro de otros mensajes, creando una estructura encadenada. Hay una gran cantidad de mensajes existentes, pero se pueden crear mensajes personalizados para adaptarse a las necesidades de cada aplicación.
- Servicios: constituyen la segunda estructura de comunicación entre nodos. Los servicios se basan en un mecanismo de petición y respuesta. Cuando un nodo realiza una petición a otro a través de un servicio, el nodo que provee el servicio recibe la solicitud, procesa la información y envía la respuesta de vuelta al nodo que realizó la petición. Los servicios se definen a través de archivos en los que especifican los formatos de los mensajes de solicitud y respuesta. Este mecanismo es útil cuando se requieren realizar tareas concretas que requieren la intervención de otros nodos.
- ROS Master: por último, el ROS Master es el elemento encargado de gestionar el funcionamiento general de ROS. Para ello lleva un registro de los nodos y los *topics* activos, de forma que cada nodo sea capaz de localizar o saber de la existencia del resto de nodos.

Existen más herramientas y opciones de comunicación en ROS, pero el funcionamiento básico de cualquier programa se basa en estos conceptos.

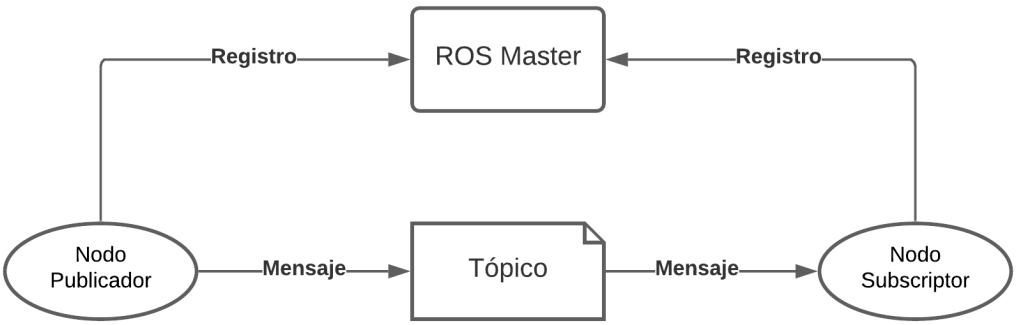


Figura 2.6: Arquitectura de publicador y subscriptor de ROS.

Por último, faltaría explicar el concepto de paquete. Los paquetes son la forma que tiene ROS de estructurar y organizar el software. Cada paquete puede contar con sus propios nodos, mensajes, archivos de configuración, etc. Así, se puede distribuir el software de forma sencilla, ya que para utilizar paquetes de otros desarrolladores, basta con clonarlos y compilarlos o con instalarlos desde el gestor de paquetes del sistema Unix.

2.3.1.1. ROS en la industria

El auge de ROS como entorno de trabajo en el mundo de la robótica es innegable. Su crecimiento en cuanto a funcionalidades y comunidad han convertido este *middleware* en, probablemente, la mejor herramienta para programar y controlar sistemas robóticos. Su modularidad y versatilidad lo convierten en la mejor opción dentro del mundo de la investigación, ya que facilita la adaptación del sistema a cualquier tipo de aplicación y a la gran mayoría del hardware utilizado. Observando en la Figura 2.7 la evolución del número de publicaciones relacionadas con ROS en los últimos años, se puede ver reflejado la importancia que está teniendo este *middleware* en la comunidad científica. Sin embargo, ROS no solamente ha tenido impacto dentro de la investigación, si no que también es uno de los principales entornos de trabajo utilizados por las empresas privadas.

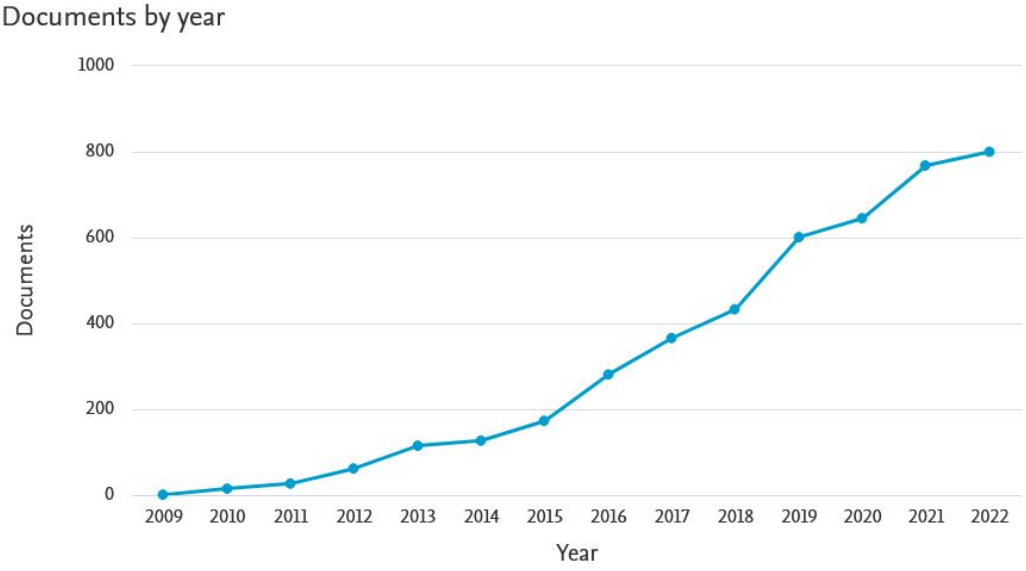


Figura 2.7: Evolución del número de publicaciones relacionadas con ROS en Scopus [24].

Existen más de 600 empresas activas en la actualidad que utilizan ROS dentro de sus actividades, ya sea para el desarrollo o creación de sus productos u ofreciendo servicios en los que esté involucrado ROS [25]. Algunos de los paquetes y herramientas más utilizados por estas empresas incluyen paquetes de navegación, como *Nav2*, paquetes de manipulación de objetos, como *MoveIt*, o paquetes de percepción del entorno a través de una amplia gama de sensores. Existe, además, un proyecto de código abierto llamado "ROS-Industrial" cuyo objetivo es la extensión de las funcionalidades de ROS a la robótica industrial. En este proyecto colaboran algunas de las empresas más grandes de robótica a nivel mundial, como por ejemplo ABB, Universal Robots, Yamaha, Siemens, Yaskawa, Mitsubishi o BMW, entre muchas otras.

2.3.1.2. ROS 2

Existe actualmente una nueva versión de ROS, denominada ROS 2. Aunque todavía se utilizan numerosas distribuciones de ROS, y la última versión, Noetic, tiene previsto su mantenimiento por parte de los desarrolladores hasta 2025, se decidió realizar un cambio sustancial en el programa para adaptarse a los cambios que han ido surgiendo desde el lanzamiento oficial de ROS en 2007. ROS 2 plantea una estructura de funcionamiento y programación diferente en el que mejora ciertos aspectos de ROS, aunque mantiene sus bases y puntos fuertes.

No obstante, a pesar de que existen herramientas para migrar programas de ROS a ROS 2, esta migración no es trivial. Durante el proceso, pueden surgir ciertas dificultades en cuanto a compatibilidades entre programas o actualización de librerías. Como esta línea de investigación lleva abierta varios años, se ha decidido realizar el desarrollo de este trabajo en ROS, evitando así tener que migrar el resto de programas y software ya desarrollados.

2.4. OpenCV

Esta librería de código libre está centrada en el manejo de imágenes de visión artificial. Dispone de herramientas para gestionar las imágenes obtenidas por diversas cámaras o sensores, realizar operaciones con ellas y aplicar algoritmos y funciones de inteligencia artificial sobre ellas. De esta forma, se pueden identificar objetos específicos en imágenes, extraer modelos en 3D, producir nubes de puntos o clasificar acciones humanas. Además, existe un paquete de ROS encargado de compatibilizar OpenCV con las librerías y características de ROS.

OpenCV dispone de las herramientas necesarias para analizar las imágenes que sirven como *input* al algoritmo de toma de decisiones que se diseña a lo largo de este proyecto. La mayoría de la información que se utiliza como punto de partida se basa en datos concretos de una imagen, como coordenadas de posición o profundidad de puntos clave en una nube de puntos. Con los algoritmos de la librería de OpenCV se pueden procesar estos datos e integrarlos en ROS de una manera eficiente y sencilla.

El paquete encargado de integrar OpenCV con ROS se denomina *cv_bridge*. Esta librería actúa a modo de interfaz entre ambos marcos de trabajo, permitiendo fusionar el procesamiento de imágenes de OpenCV con las librerías de sensores incluidas en ROS. Esta herramienta incluye, entre otras cosas, estructuras de datos y conversiones entre los mensajes más utilizados por los sensores convencionales del mundo de la robótica (como sensores LiDAR, cámaras RGB o de tiempo de vuelo) y las funciones y algoritmos empleados en el procesamiento y visualización de imágenes.

3. METODOLOGÍA

Este capítulo incluye el diseño y desarrollo de software elaborado durante este trabajo. En él se detallan las propuestas que se han planteado para alcanzar los objetivos establecidos, explicando cuáles son las principales ventajas e inconvenientes de cada uno, así como los motivos por los que se ha terminado eligiendo una propuesta frente al resto. No obstante, antes de comenzar con el desarrollo, es necesario realizar un análisis y descripción del estado inicial del proyecto, en especial del hardware utilizado y de la arquitectura de software existente, para poder comprender las mejoras que se han introducido.

3.1. Análisis del estado inicial

Tal y como se ha descrito en la sección 2.2.5, esta línea de investigación forma parte del proyecto ROBOCROP, financiado por la Agencia Estatal de Investigación. Los buenos resultados que se obtuvieron han motivado la continuación del trabajo realizado a través de la prueba de concepto PoC-ROBOCROP, financiada también por la Agencia Estatal de Investigación, y cuyo objetivo es probar las técnicas implementadas en el primer proyecto en un entorno real, a través de un prototipo. Aunque se han mantenido la mayoría de equipos y componentes, así como la arquitectura diseñada en ROBOCROP, se han llevado a cabo algunas actualizaciones del equipo y el software con el objetivo de mantener su compatibilidad con las últimas mejoras tecnológicas y las prestaciones que éstas ofrecen, buscando mejorar el rendimiento y obtener una funcionalidad avanzada en consonancia con los estándares actuales del sector. Una descripción detallada del proyecto ROBOCROP se puede ver en [23].

3.1.1. Descripción del Hardware

Respecto al hardware utilizado, se han mantenido los manipuladores originales, por lo que se dispone de unos brazos robóticos MICO2^{TM}, comercializados por Kinova Technology (Montreal, QC, Canadá). Las características de los brazos se han descrito en la sección 2.2.5. Los brazos robóticos se encuentran acoplados en un soporte rígido para este prototipo, aunque la perspectiva a futuro es instalarlo sobre una base robótica móvil que permita su desplazamiento por el campo y las plantaciones.

El sistema de percepción inteligente se realiza a través de dos cámaras. Los dos modelos originales han sido sustituidos por versiones más recientes para mejorar las prestaciones del sistema en su conjunto. Estos modelos son:

- Cámara "TRI016S", que consiste en una cámara a color Triton. Dispone de 1.6 MegaPíxeles y un sensor Sony "IMX392 CMOS" de 1/2.9". Es capaz de proporcionar una velocidad de captura de 71.6 fps, mejorando la tasa de fotogramas por segundo considerablemente, ya que el anterior equipo únicamente era capaz de alcanzar los 15 fps. La resolución obtenida es 1440 x 1080 píxeles. Aquí, se ha producido una reducción respecto al modelo anterior, que proporcionaba una resolución de 2448 x 2050 píxeles. Esta disminución en la resolución es el factor que permite el aumento de fotogramas por segundo ya mencionado. De esta forma, se consigue aumentar la tasa de refresco del sistema de percepción sin que la resolución de la imagen caiga drásticamente, ya que la resolución proporcionada por la cámara Triton es suficiente para la identificación y localización de las frutas con las que se trabaja.

- Cámara de tiempo de vuelo "Helios2" con un sensor Sony "DepthSense IMX556PLR CMOS" de 8 mm. Proporciona 640 x 480 píxeles con una precisión de punto flotante de 16 bits. La tasa de refresco alcanzada es de 30 fps. El objetivo de esta cámara consiste en obtener un mapa de profundidad del entorno y una imagen en escala de grises que se corresponde con la respuesta en amplitud. El mapa de profundidad permite extraer la distancia de un punto a la cámara, a través de la coordenada Z del mismo. La coordenada Z aumenta a medida que incrementa la distancia de un punto a la cámara, la coordenada X incrementa horizontalmente hacia la derecha de la cámara y la coordenada Y incrementa verticalmente hacia abajo. El resto de propiedades ópticas de interés se describen en la Tabla 3.1.

Los dos sensores Sony disponen de obturador global, lo que permite capturar imágenes de objetos en movimiento sin que aparezcan distorsiones o deformaciones. Esto aumenta la precisión y nitidez de la imagen de trabajo. Los parámetros de calibración intrínsecos y extrínsecos de ambas cámaras, así como la calibración de distancias para la cámara de tiempo de vuelo se detallan en [23].

Las propiedades ópticas de la cámara de tiempo de vuelo son importantes para el cálculo del área y de la profundidad de las berenjenas, que se realiza en el algoritmo de procesamiento de datos previo a la toma de decisiones. Estas propiedades se han incluido dentro de una clase de C++ específica para la cámara. En caso de modificar alguno de sus parámetros o llevar a cabo una recalibración, habría que actualizar esta clase para utilizar los nuevos datos.

Característica	Valor
Resolución	640 x 480 píxeles
Precisión de punto flotante	16 bits
Tasa de refresco	30 fps
Distancia de detección mínima	0.3 m
Distancia de detección máxima	8.3 m
Campo de visión Horizontal	69°
Campo de visión Vertical	51°
Tamaño de píxel	10.0 μm (H) x 10.0 μm (V)

Tabla 3.1: Propiedades de la cámara TOF "Helios2" con sensor Sony "DepthSense IMX556PLR".

Las dos cámaras se encuentran alineadas verticalmente y están instaladas conjuntamente en un soporte fijo respecto a los brazos manipuladores, de modo que se puede conocer la relación entre los sistemas de coordenadas de los sensores y el manipulador bimanual midiendo la distancia que separa ambos equipos, sobre cada eje de coordenadas. No obstante, se prevé acoplar el sistema multisensorial a una unidad de *pan-tilt*. Este dispositivo mecánico consiste en un componente de dos grados de libertad que permite girar los sensores sobre los ejes horizontal y vertical, dotando al sistema de capacidad de giro de cabeceo y guiñada. La instalación de este equipo viene acompañada de los servomotores encargados de realizar el movimiento y de algún sensor que permita conocer el giro actual. Comúnmente, se utiliza un potenciómetro integrado con el servomotor. Este tipo de sensores permiten conocer el estado actual del servomotor, para poder conocer la orientación de la cámara y emplear estos datos en los algoritmos de detección.

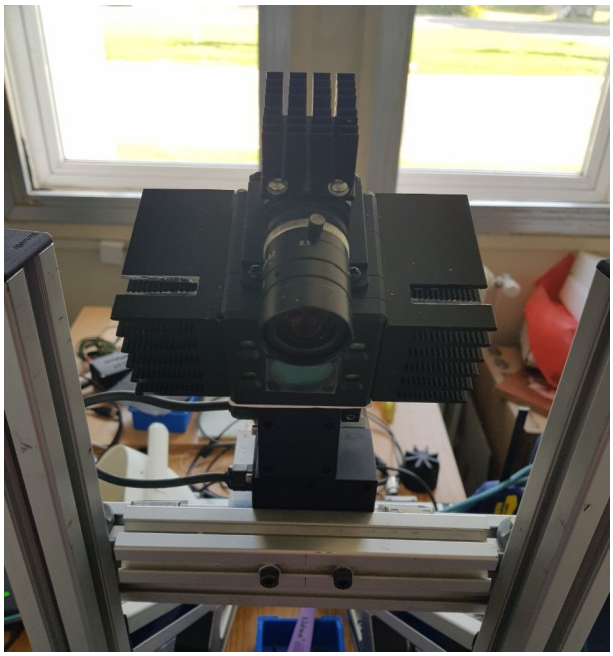


Figura 3.1: Instalación de las dos cámaras utilizadas por el robot.

3.1.2. Arquitectura de Software

La arquitectura de software desarrollada a lo largo del proyecto ROBOCROP utiliza como entorno de trabajo ROS. La primera versión que se hizo del proyecto estaba implementada con ROS Indigo, pero esta distribución terminó su periodo de mantenimiento oficial o EOL (por sus siglas en ingles *End of Life*) en 2019. En ese momento, se migró la arquitectura de software a la distribución Melodic. No obstante, con el objetivo de poder implementar las últimas herramientas y algoritmos de inteligencia artificial, y con el fin de alargar al máximo la vida de la arquitectura diseñada, se ha optado por migrar nuevamente el desarrollo realizado a la distribución de ROS Noetic. Esta distribución es la más moderna de ROS, y se trata de la última distribución que se lanzará oficialmente, ya que una vez se termine la vida de Noetic, ROS dejará de recibir mantenimiento y los esfuerzos de los desarrolladores se centrarán exclusivamente en ROS 2.

La arquitectura se diseñó con el objetivo de utilizar la modularidad e independencia de nodos que facilita este *middleware* de trabajo. Para ello, se separó el funcionamiento del programa en cuatro módulos principales, conectados entre ellos mediante un flujo de información prácticamente lineal. Es decir, la salida de un módulo son las entradas del siguiente en la mayoría de los casos. Este sistema es muy similar a una arquitectura por capas. Sin embargo, la arquitectura de grafos de ROS permite tener un flujo de información que sea no lineal, y la información publicada por cualquiera de estos módulos puede ser utilizada por cualquier nodo del sistema. Es preciso tener esta característica en cuenta a la hora de diseñar cada nuevo nodo de la arquitectura, ya que limitar el funcionamiento del programa a un flujo lineal sería restringir las capacidades del mismo innecesariamente. Esta arquitectura presenta numerosas ventajas a la hora de diseñar, desarrollar y probar la arquitectura implementada.

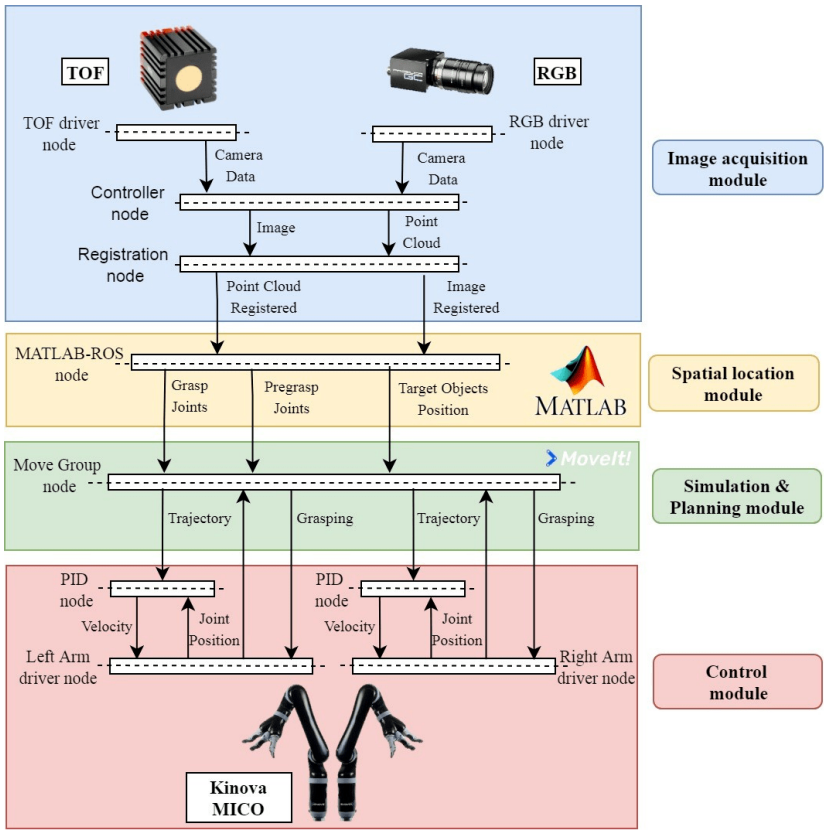


Figura 3.2: Esquema de la arquitectura de software existente [23].

En primer lugar, diseñar una arquitectura de forma modular permite dividir las tareas de programación en otras más pequeñas, convirtiendo el diseño del software en una integración de pequeñas funciones. Este mecanismo facilita la depuración del código y la detección de errores, ya que se pueden identificar los fallos de una manera más rápida y precisa.

En segundo lugar, una arquitectura modular aumenta considerablemente la escalabilidad del programa. Es decir, se pueden añadir mejoras y ampliaciones fácilmente, sin tener que modificar el comportamiento global del software. Conociendo las entradas y salidas de cada módulo, se pueden incorporar nodos intermedios que mejoren el procesamiento de datos sin tener que modificar el resto del código.

Por último, el sistema de nodos incrementa la adaptabilidad del programa. Los nodos pueden ser intercambiados por otros siempre y cuando se mantenga las mismas estructuras de datos para las entradas y las salidas. Esto permite modificar el funcionamiento completo del programa realizando pequeñas modificaciones en uno de sus módulos. Estas dos últimas ventajas han tenido gran repercusión a la hora de implementar un sistema de planificación de tareas. Por ejemplo, la arquitectura desarrollada permite implementar diversos algoritmos de toma de decisiones sin alterar el resto del código del programa. Únicamente se intercambia el nodo dónde se ejecutan los algoritmos por otro distinto, pero manteniendo las mismas salidas y entradas. De esta forma, se puede modificar la toma de decisiones realizada por el programa sin tener que modificar todos los archivos que lo componen.

Cada uno de los cuatro módulos que componen la arquitectura de software está formado, a su vez, por uno o más nodos de ROS. Los módulos realizan las siguientes funciones:

- Obtención de la información e imágenes de cada una de las cámaras.
- Detección y localización de las berenjenas en un sistema de coordenadas conocido por el robot. También se realiza un primer cálculo de la cinemática inversa a partir de los datos proporcionados por el algoritmo de localización.
- Planificación del movimiento.
- Control de los manipuladores.

El primer módulo de adquisición de imágenes contiene los nodos encargados de ejecutar los drivers de cada una de las cámaras de forma síncrona y de registrar la información espacial proporcionada por la cámara de tiempo de vuelo con las imágenes RGB. Estos nodos se encargan de realizar el control del equipo multisensorial a bajo nivel, liberando al programador de esta tarea. Esta particularidad es muy propia de los sistema desarrollados en ROS. La mayoría de proveedores de hardware, tanto de sensores como de actuadores, proporcionan paquetes propios que contienen las herramientas para gestionar, calibrar y operar sus equipos.

Estos paquetes han sido desarrollados por profesionales muy familiarizados con el equipo y suelen contar, además, con un personal de mantenimiento que ayuda a resolver problemas de incompatibilidad o errores en el funcionamiento del paquete. La utilización de estos paquetes es un paso fundamental en el desarrollo de arquitecturas de software de equipos robóticos, y es otra de las grandes ventajas brindadas por la selección de un *middleware* tan extendido y con una comunidad tan grande. La información registrada por las cámaras es publicada en *topics* y utilizada por el segundo módulo mediante la técnica de publicador y subscriptor ya explicada.

El segundo módulo se basa en un único nodo de ROS, pero ejecutado en MATLAB. Este programa dispone de una extensión llamada *ROS Toolbox* que permite utilizar funcionalidades de ROS e integrar la interfaz de MATLAB como si se tratase de un nodo más, de forma que se pueda comunicar e interactuar con el resto de nodos en ejecución. El razonamiento detrás de la utilización de MATLAB es tener acceso al conjunto de herramientas que proporciona este programa para el tratamiento de imágenes, como puede ser el *Image Processing Toolbox*^{TM}. El nodo de MATLAB tiene como objetivo utilizar la información en crudo, o *raw data*, proporcionada por el primer módulo para localizar cada una de las berenjenas objetivo y sus contornos, así como todas las hojas que se consideren obstáculos en la recolección.

Esta tarea, aunque trivial para un ser humano, es uno de los mayores retos del proyecto. Una detección correcta es vital para obtener un funcionamiento adecuado del robot, siendo esta información el punto de partida para el resto del programa. Aunque las diferentes pruebas, análisis y comparaciones realizadas entre los distintos métodos de segmentación y clasificación de imágenes quedan fuera del alcance de este trabajo, es conveniente realizar una breve explicación del funcionamiento general del algoritmo de detección, explicado detalladamente en [23].

Las imágenes en crudo son tratadas mediante diversas transformaciones de color que permiten reducir la sensibilidad del algoritmo a las condiciones de iluminación, volviéndolo más robusto. A continuación, se procesan las imágenes mediante un procedimiento que funciona en tres etapas. En la primera se aplica un clasificador basado en píxeles que utiliza un algoritmo de aprendizaje supervisado denominado *Support Vector Machine* o SVM. Este método clasifica los píxeles en cuatro posibles clases, que corresponden a berenjenas, hojas, ramas y fondo. Después, para poder identificar unidades de berenjenas a partir de los píxeles clasificados como tal, se realizan varios procesos basados en la transformada *Watershed*. El resultado de este procedimiento, son varios conjuntos de píxeles que se corresponden con cada una de las berenjenas de la imagen, por lo que solamente es necesario realizar algunos cálculos trigonómetricos sencillos para calcular el

centroide de cada berenjena con la ayuda de la nube de puntos proporcionada por la cámara de tiempo de vuelo.

De esta forma, la información que se dispone acerca de cada una de las berenjenas de la imagen incluye el número de píxeles que pertenecen a la berenjena, los píxeles que delimitan el contorno de la misma y el punto correspondiente al centroide de la berenjena. A partir de estos datos, es posible obtener otros parámetros geométricos como una estimación del área de cada berenjena o la orientación de la misma, dada como un ángulo respecto al eje horizontal. Por otro lado, también se dispone de los píxeles que componen los contornos de cada hoja. No obstante, el proceso de obtención de contornos de hojas es menos importante y se trata con menos técnicas de visión por computador, por lo que es posible que algunas hojas que se encuentran juntas sean tratadas como una sola por el programa, o incluso que se clasifique alguna pequeña rama como hoja. Todos estos datos constituyen la entrada que utiliza el algoritmo de planificación de tareas a alto nivel como punto de partida.

La planificación de tareas existente consiste en un algoritmo jerárquico simple que sigue el procedimiento descrito a continuación. En un primer lugar, procesa los datos obtenidos y realiza una clasificación entre berenjenas ocluidas y no ocluidas. Para ello, utiliza los parámetros del área o número de píxeles de cada berenjena y los datos de profundidad proporcionados por la nube de puntos. En el caso de las berenjenas no ocluidas, es decir, que no están tapadas por alguna hoja, asigna a cada brazo un número de berenjenas a recoger basándose en la posición de cada una de ellas. Es decir, si la berenjena se encuentra en la mitad izquierda de la imagen, se asigna al brazo izquierdo, si se encuentra en la mitad derecha, se asigna al brazo derecho.

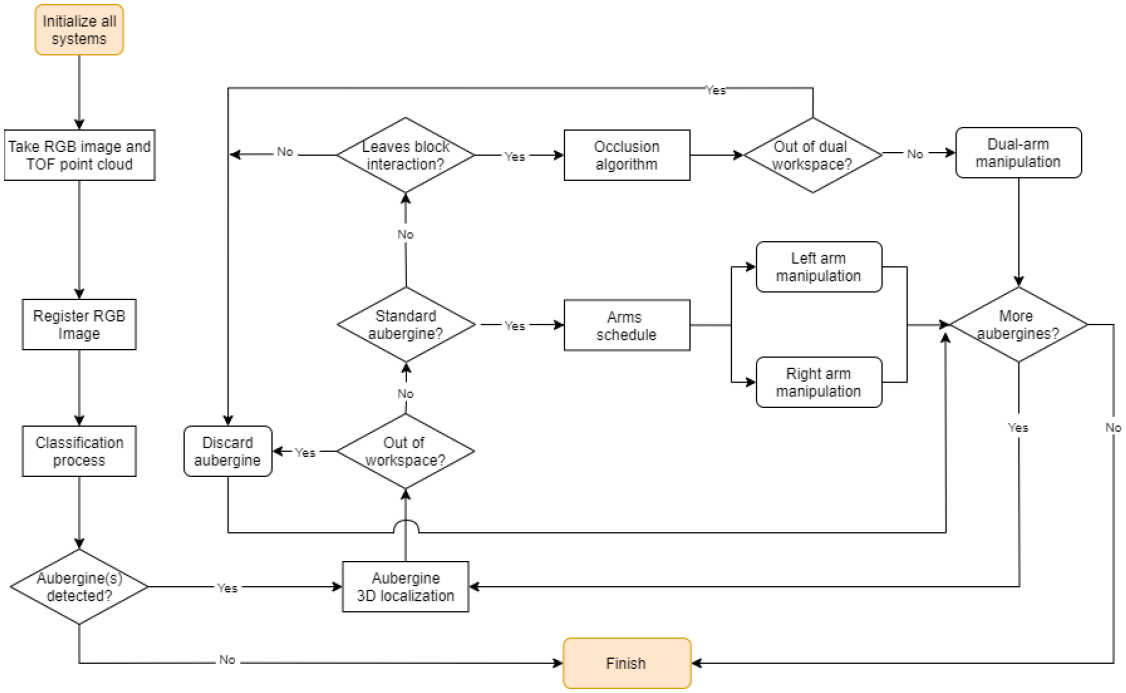


Figura 3.3: Flujograma del comportamiento existente [23].

Por último, se realiza un primer cálculo de la cinemática inversa de cada brazo necesaria para llegar hasta el centroide de cada berenjena detectada, a través del *Robotics System Toolbox* de MATLAB. Este paso sirve como primera comprobación de la viabilidad del movimiento propuesto. Es preciso recordar en este momento que, el espacio de trabajo de cualquier robot manipulador es limitado y está sujeto a ciertas restricciones intrínsecas a su morfología. Aunque la planifica-

ción de movimiento no se realiza en este módulo, el cálculo de la cinemática inversa se puede utilizar como verificación de la tarea propuesta. En caso de que esta cinemática devuelva una configuración no factible, se puede intentar plantear otra estrategia antes de clasificar la berenjena como inalcanzable, que sería el resultado obtenido al enviar una posición no válida al planificador de trayectorias.

En caso de que una berenjena sea clasificada como ocluida, se inicia un algoritmo específico para tratar con este tipo de berenjenas. Éste consiste en identificar qué hoja está ocluyendo a la berenjena, para a continuación calcular una trayectoria óptima que permita apartar la hoja. Esta trayectoria sería realizada por uno de los dos brazos, mientras el otro brazo libre se encarga de recoger la berenjena, que debería haber quedado liberada.

Como se ha podido observar, el segundo módulo de la arquitectura de software, desarrollado en MATLAB, dispone de un algoritmo de toma de decisiones que permite realizar una asignación de las berenjenas que debe recoger cada brazo. Este algoritmo es la primera versión del planificador de tareas a alto nivel que se diseña a lo largo de esta memoria. Dentro de los objetivos del trabajo, se ha propuesto mejorar este programa y realizar nuevos algoritmos que permitan realizar una planificación de tareas a alto nivel más eficiente, utilizando otros parámetros que no han sido tenidos en cuenta en la primera versión del módulo.

El tercer módulo de la arquitectura se encarga de la planificación del movimiento. Una vez se ha decidido qué berenjenas deben ser recogidas, es necesario realizar una simulación de las trayectorias que se deben realizar para asegurar que se tratan de movimientos libres de colisiones. Esta tarea se realiza mediante el paquete *MoveIt!*. Éste es el paquete de ROS más extendido para robots manipuladores. Incluye herramientas para la planificación de trayectorias y detección de colisiones, numerosas herramientas para calcular el modelo cinemático inverso e interfaces preparadas para la visualización de los datos de profundidad y mapas de nubes de puntos. En este caso particular, es utilizado para realizar una simulación en un entorno virtual fiel a la realidad, que permite comprobar que las trayectorias que se van a realizar son válidas. Una vez se dispone de una trayectoria validada, es posible almacenar la configuración de cada articulación de los brazos robóticos y publicarlas para que sean utilizadas por el último módulo de la arquitectura.

Finalmente, el módulo encargado del control de los manipuladores ejecuta los comandos o posiciones, que han sido validados, por medio de unos controladores encargados de comunicarse con los procesadores de señales digitales de los brazos robóticos. Al igual que el primer módulo que se ha explicado, éste último está compuesto por los nodos desarrollados por los proveedores del hardware. En este caso, los paquetes gestionan la comunicación directa y el control a bajo nivel con los brazos robóticos MICO2^{TM}. El módulo también contiene dos nodos PID (uno para cada brazo), que actúan sobre la señal que se envía al driver de los brazos robóticos. De esta forma se obtiene un control más seguro y regulable.

3.2. Estructura del paquete

Todo el trabajo recogido en esta memoria se ha incluido en un único paquete de ROS. Tal y como se ha explicado en la sección 2.3.1, un paquete es la forma que tiene ROS de estructurar el software, y contiene nodos, mensajes y archivos de configuración, entre otros. Aunque no hay muchas restricciones a la hora de organizar los archivos y directorios del paquete, existen buenas prácticas de programación y algunas directrices que ayudan a entender cada paquete y simplifican el proceso de búsqueda de archivos. Con el objetivo de cumplir estas directrices, el paquete creado incluye las siguientes carpetas:

- *include*: contiene los ficheros de encabezado de todas las clases definidas en el paquete. El sistema de compilación de ROS, llamado *catkin*, busca dentro de esta carpeta los ficheros que se instalan en sus librerías globales para que puedan ser utilizadas por otros paquetes.
- *src*: contiene los ficheros correspondientes a los nodos. Existe un fichero ".cpp" por cada nodo definido en el paquete. Además, dentro del directorio "src" hay otra carpeta anidada, llamada *implementation*. En esta carpeta se incluyen las implementaciones de las clases utilizadas, que han sido definidas en los archivos de encabezado que se encuentran dentro de la carpeta "include".
- *msg*: contiene la definición de los mensajes propios creados.
- *launch*: contiene los ficheros ".launch". Estos archivos se utilizan para ejecutar todos los nodos requeridos por un programa mediante una única línea de comando. También permite la ejecución de *rosbags* y otras opciones como la declaración de parámetros.
- *rosbags*: no forma parte de la estructura convencional de paquetes de ROS, pero se ha añadido para incluir dentro los archivos ".bag" que se han utilizado para algunas pruebas. Éstos incluyen los datos ya convertidos por el conversor de MATLAB, de forma que no es necesario ejecutarlo para comprobar el funcionamiento de alguno de los nodos.
- *matlab*: este directorio tampoco forma parte de la estructura convencional. Se utiliza para incluir un código de MATLAB de forma conjunta con el paquete y que su distribución sea más rápida, pero no es utilizado por ningún archivo de ROS ni su presencia es necesaria para el funcionamiento del paquete.

De forma adicional, el paquete incluye los archivos "CMakeLists.txt" y "package.xml". Estos archivos se incluyen en todos los paquetes de ROS. El primero comprende las instrucciones de compilación del paquete, la vinculación de librerías, las dependencias del paquete o la generación de ejecutables. El segundo incluye metadatos acerca del paquete, como descripción, autor o dependencias. Deben modificarse cada vez que se añada algún archivo nuevo al paquete.

En ROS Noetic existen diferentes métodos para crear y configurar un nodo, y se puede realizar tanto en Python como en C++. Este trabajo se ha realizado mayoritariamente en C++, anteponiendo la estructuración de paquetes que proporciona este lenguaje frente al amplio abanico de opciones y herramientas que ofrece Python. Además, aunque se puede tener un paquete que incluya código en ambos lenguajes, es recomendable mantener los archivos separados, y crear un paquete independiente para cada lenguaje.

Los ficheros utilizados en el paquete para definir los nodos son códigos muy sencillos. Solamente incluyen las funciones necesarias para instanciar el nodo, comunicarlo con el ROS Máster y ejecutar sus funciones básicas. El desarrollo de los algoritmos implementados se ha realizado en clases de C++. De esta forma, al iniciar el nodo, se instancia un objeto de la clase que incluye las funciones, subscriptores y publicadores que se necesitan para el funcionamiento del nodo.

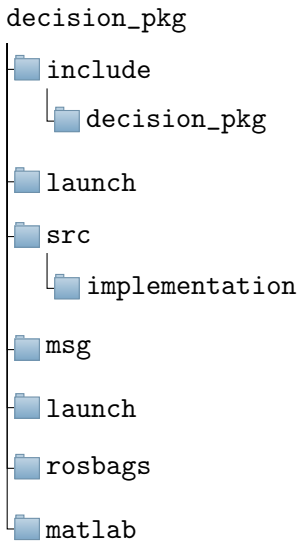


Figura 3.4: Árbol de directorios del paquete.

Tal y como se puede observar en el árbol de directorios del paquete, se ha decidido separar la declaración de las clases de C++ utilizadas de su implementación. Esta es una práctica muy común en los lenguajes de programación de C y C++, debido al comportamiento del compilador. No obstante, en ROS también aporta otros beneficios. El principal es que la herramienta de compilación, *catkin*, permite añadir los archivos de cabecera a la librería general de ROS, de forma que se puedan utilizar las clases y funciones definidas en estos archivos desde otros paquetes. Este sistema ayuda a la reutilización de código, y aporta una gran modularidad. Para utilizar librerías de otros paquetes, únicamente hay que declarar el paquete que se quiera utilizar como dependencia, a través de los archivos "CMakeLists.txt" y "package.xml".

Todas las clases definidas contienen en sus primeras líneas del encabezado unos comentarios para indicar la utilidad de la clase, sus subscriptores y sus publicadores. El objetivo de estos comentarios es facilitar el entendimiento del código, para que se pueda conocer de manera rápida y sencilla el funcionamiento de cada clase y archivo. Esto pretende simplificar todavía más la reutilización de código. A parte de estos primeros comentarios, el código se ha comentado de manera tradicional para explicar los comandos y funciones utilizados.

3.3. Migración de MATLAB a ROS

Tal y como se ha explicado en la Sección 3.1.2, el segundo módulo de la arquitectura de software, encargado de la localización de las berenjenas y la planificación de tareas a alto nivel, se realiza exclusivamente en MATLAB. La comunicación con el resto de capas de la arquitectura se realiza mediante el *ROS Toolbox*. Esta estructura, basada en la combinación de dos software diferentes, ofrece ciertas ventajas al permitir el uso de las herramientas ofrecidas por ambos sistemas. Sin embargo, también presenta inconvenientes asociados. En un primer lugar, se encuentran limitaciones en cuanto al número de funcionalidades de ROS que pueden ser implementadas en MATLAB. Además, se deben considerar las restricciones propias de la programación debido a la necesidad de utilizar dos software distintos cada vez que se quiera modificar o mejorar el código desarrollado.

Por otro lado, es preciso destacar que MATLAB es un software con un alto consumo de recursos y ejecución lenta, lo que se traduce en tiempos prolongados para iniciar su funcionamiento. Esta

característica puede impactar negativamente en el rendimiento general del software utilizado en el sistema, perjudicando los tiempos de arranque y computación del robot.

Adicionalmente, a pesar de que MATLAB ofrece una gran variedad de opciones y funciones integradas en sus *add-ons*, la última distribución de ROS (Noetic) también dispone de la misma oferta de librerías y herramientas. Es por todo esto, por lo que se ha decidido realizar una migración completa del código desarrollado en MATLAB a ROS. Tras esta migración se dispone de todo el software del robot operando en el mismo entorno de trabajo, lo cual agiliza la programación y la ejecución del software, eliminando las restricciones que pudieran existir debido a la necesidad de integrar dos entornos de trabajo distintos.

El proceso de migración se ha dividido en dos líneas de trabajo que avanzan en paralelo, una encargada del área de visión por computador y otra enfocada en la parte de toma de decisiones. El desarrollo relacionado con la visión por computador se realiza en otro trabajo, mientras que la migración del software encargado de la planificación de tareas a alto nivel se ha realizado en este, y es uno de los objetivos definidos en la memoria. Esta mecánica de trabajo, no obstante, presenta un inconveniente principal, ya que la toma de decisiones necesita utilizar los datos proporcionados por los algoritmos de detección por computador, que estaban siendo desarrollados simultáneamente. Por lo tanto, con el fin de que una línea de trabajo no frenara a la otra, se decidió utilizar las salidas generadas en el algoritmo ya existente en MATLAB, de manera temporal, hasta que se completara la migración.

De esta forma, se ha diseñado un código en MATLAB que utiliza las funciones proporcionadas por el *ROS Toolbox* y el sistema de publicación y subscripción de ROS para convertir y enviar todos los datos útiles que se obtienen con los algoritmos de detección, ya programados, al entorno de trabajo de ROS, donde se desarrollan los algoritmos de toma de decisiones. Se trata, por tanto, de un código de uso temporal, ya que cuando se termine la migración completa de esta parte del software dejará de ser necesario.

El código de MATLAB, al que de ahora en adelante se nombrará como conversor, ya que es ésta su funcionalidad última, genera un *topic* por cada conjunto de datos que se necesite enviar a ROS. Los datos publicados son aquellos que se han considerado útiles para realizar la toma de decisiones, y se describen en la siguiente lista:

- **Datos sobre las berenjenas:** se trata de una estructura de datos de MATLAB que contiene la información básica de cada berenjena. Se publica en el *topic /matlab_statsAub*. La estructura está formada por 6 campos.
  - Área: un *integer* sin signo que almacena el número de píxeles que componen la berenjena.
  - Centroide: un *array* de dos *floats* que contienen las coordenadas X e Y del centroide de la berenjena. Este dato hace referencia a píxeles, es decir, la información almacenada corresponde con el píxel de la imagen registrada en el que se encuentra el centroide. Aunque se utilicen variables de punto flotante, cuando se quiera acceder a la información de este píxel habrá que redondear al número entero más cercano. Esta pérdida de precisión es inherente a la discretización de la imagen y, por tanto, inevitable.
  - Eje mayor: un *float* que indica la longitud del eje mayor de la berenjena, en píxeles.
  - Eje menor: un *float* que indica la longitud del eje menor de la berenjena, en píxeles.
  - Solidez: un *float* que indica la relación entre tamaño de área y número de píxeles que la componen. Este parámetro se puede utilizar como elemento diferenciador para clasificar berenjenas. Tiene mayor importancia en la parte dedicada a la visión por computador.

    - Orientación: un *float* que indica la rotación de la berenjena respecto el eje horizontal positivo.

- **Contornos de las berenjenas:** esta estructura de datos consiste en tres arreglos de celdas encadenados. El tamaño del primer arreglo se corresponde con el número de berenjenas que se detecta en la imagen. Cada celda de este arreglo se compone de otro *array* que contiene todos los píxeles que forman el contorno de la berenjena. El último se trata siempre de un *array* de dos *floats* que contienen las posiciones X e Y de cada píxel. Se publica en el *topic /matlab_boundAubergines*.
- **Contornos de las hojas:** esta estructura de datos es igual que la descrita para el contorno de las berenjenas, pero para las hojas. El tamaño del primer arreglo corresponde con el número de hojas detectadas. En general, este *array* se diferencia del anterior en que contiene muchas más hojas, pero el contorno de cada hoja suele ser más pequeño que el de las berenjenas. Esto quiere decir que el tamaño del primer arreglo será mayor, pero el tamaño del siguiente *array* tiende a ser más pequeño. Se publica en el *topic /matlab_boundLeaves*.
- **Imagen RGB:** se trata de la imagen a color que se capta con la cámara RGB. Se publica en el *topic /matlab_imgRGB*.
- **Imagen Procesada:** esta segunda imagen se corresponde con la imagen generada tras aplicar los algoritmos de visión por computador y realizar la clasificación por píxeles descrita en el apartado anterior. Se publica en el *topic /matlab_img*.

Con el objetivo de agilizar la comunicación y simplificar el proceso de publicación de datos, se ha optado por crear un mensaje de ROS personalizado que incluya toda la información contenida en la estructura de datos de MATLAB para las berenjenas. El mensaje creado recibe el nombre de *statsAub* y contiene los mismos campos que la estructura de datos de MATLAB. *ROS Toolbox* proporciona un sistema para añadir mensajes personalizados a sus librerías y poder usarlos como los mensajes convencionales de ROS. Así, es posible publicar un único mensaje que contenga toda la información de cada berenjena, lo que permite agilizar el proceso de transferencia de datos entre los dos entornos de trabajo.

Para los contornos de las berenjenas y de las hojas se ha utilizado otro mensaje personalizado llamado *boundAub*. Existen diversas configuraciones posibles para transmitir esta información a ROS. Por un lado, se puede optar por crear un mensaje que imite la estructura de datos que se dispone en MATLAB, de tres arreglos encadenados. No obstante, aunque se puedan crear mensajes personalizados con *arrays* encadenados, ROS no dispone de un mecanismo sencillo para acceder a ellos una vez se han publicado. Además, añadir este tipo de mensajes a las librerías de MATLAB puede generar algunos errores.

Otra posible configuración sería publicar un mensaje por cada berenjena u hoja, que contenga un *array* con todas las posiciones de los píxeles que lo componen. Sin embargo, esta configuración sigue necesitando de un sistema encadenado al igual que la primera opción. Por lo tanto, se ha decidido utilizar un mensaje compuesto por un único *array* de *floats* para almacenar las posiciones X e Y y un entero sin signo para almacenar el tamaño original del *array*. El arreglo de *floats* almacena primero las coordenadas en X y luego las coordenadas en Y, de forma conjunta. El entero que indica el tamaño se utiliza para que cuando el mensaje sea leído en el nodo de destino, se pueda reconstruir fácilmente la matriz original.

$$A = \begin{pmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \end{pmatrix} \quad ; \quad B = \begin{pmatrix} x_1 & x_2 & x_3 & y_1 & y_2 & y_3 \end{pmatrix} \tag{3.1}$$

De esta forma, cada elemento del arreglo en MATLAB correspondiente a los contornos de hojas o berenjenas será una matriz similar a A, donde la primera columna corresponde con las coordenadas en X de un punto, y la segunda columna con las coordenadas en Y. Cada fila corresponde a un punto distinto, así que se puede visualizar el contorno uniendo con una línea los puntos almacenados en la matriz. Cuando se convierte al mensaje *boundAub*, la matriz A se transforma en la matriz B, donde la segunda columna se concatena con la primera y se almacena en una única fila, como si fuera un vector. Cuando el mensaje sea recibido, se realizará la transformación inversa a partir del dato del tamaño original para volver a obtener A. Este procedimiento se realiza tanto para los contornos de las berenjenas, como para los contornos de las hojas, ya que la estructura de datos es la misma.

(a) Imagen RGB con codificación "64fc3".

(b) Imagen procesada con codificación "rgb8".

Figura 3.5: Imágenes publicadas en ROS por el conversor.

Los mensajes utilizados para las imágenes son proporcionados por el paquete *sensor_ msgs*. Se trata de una librería predeterminada que incluye mensajes para gestionar la información capturada por los sensores más usados dentro del mundo de la robótica. Incluye mensajes para sensores LiDAR, GPS, IMU y nubes de puntos, entre otros. En este caso particular, se han utilizado los mensajes *sensor_ msgs/Image*. Para la imagen RGB se ha utilizado una codificación tipo "64fc3" y para la imagen procesada una codificación tipo "rgb8". El primer tipo representa cada píxel mediante 8 bytes de precisión de coma flotante en tres canales de color: rojo, verde y azul. La segunda codificación por otro lado, representa cada píxel mediante un único byte por cada canal de color. Es decir, la segunda codificación tiene menos precisión y mucho menor rango dinámico de color.

Finalmente, el conversor convierte la nube de puntos obtenida por la cámara de tiempo de vuelo a un archivo ".pcd". Este tipo de archivo, cuyo nombre proviene de las siglas en inglés *Point Cloud Data*, es un formato específico para nubes de puntos. MATLAB dispone de funciones específicas para crear este archivo sin perder información, y puede ser leída fácilmente desde cualquier nodo de ROS utilizando las librerías correspondientes: *pcl_ conversion* y *pcl_ ros*.

El conversor diseñado, consiste en la transformación de los datos disponibles en MATLAB a

formatos compatibles con los mensajes de ROS, para que después puedan ser publicados con la ayuda del *ROS Toolbox*. El conversor no tiene por qué ser utilizado cada vez que se quiera hacer una prueba. Es suficiente con ejecutarlo una única vez con los datos que se quieren utilizar y almacenar los mensajes publicados en un archivo *bag* de ROS. Una vez se han almacenado los datos en un *bag*, al reproducir este archivo desde una terminal a través de las herramientas de ROS se replicará el funcionamiento del conversor. Esta característica resulta extremadamente útil a la hora de comprobar el funcionamiento del programa desarrollado, ya que evita la utilización de MATLAB y permite trabajar con un único entorno de trabajo.

A modo de resumen, en la siguiente Tabla 3.2 se muestran los datos transformados por el conversor, junto a la información acerca de los *topics* y mensajes utilizados.

Dato	*Topic*	Mensaje
Berenjenas	/matlab_statsAub	statsAub.msg
Contorno de Berenjenas	/matlab_boundAubergines	boundAub.msg
Contorno de Hojas	/matlab_boundLeaves	boundAub.msg
Imagen RGB	/matlab_imgRGB	sensors_msgs/Image
Imagen Procesada	/matlab_img	sensors_msgs/Image

Tabla 3.2: Resumen sobre *topics* y mensajes utilizados en el conversor.

Hay que recordar, que el objetivo de este código no es incluirse en la versión final del software, sino que tiene un uso temporal hasta que se puedan integrar las dos líneas de trabajo en una sola. Es posible que cuando esto ocurra, haya que realizar ciertas adaptaciones al código o la estructura para posibilitar la integración.

3.4. Componentes del módulo de planificación de tareas

Una vez se han convertido todos los datos a un formato compatible con ROS mediante el conversor descrito, es posible empezar a operar con estos datos en crudo para obtener características útiles para la toma de decisiones. La arquitectura diseñada durante este trabajo se basa en dos nodos de ROS conectados secuencialmente entre ellos. El primero de ellos se encarga de realizar las operaciones necesarias sobre los datos para extraer información útil, mientras que el segundo es donde se realizan los algoritmos de toma de decisión.

La arquitectura de software general del sistema, por tanto, se ve modificada únicamente en su segundo módulo, encargado de la detección, localización de las berenjenas y la aplicación de los algoritmos de planificación de tareas. Los datos obtenidos por el software de visión por computador son transformados e introducidos en ROS mediante el conversor de MATLAB, a continuación son procesados por un nodo de tratamiento de datos denominado *data_management* y, finalmente, son enviados al nodo encargado de la toma de decisiones, cuyo nombre dependerá del algoritmo que se esté utilizando.

La salida del nodo de la toma de decisiones debe ser un mensaje que contenga toda la información necesaria para que el tercer módulo (encargado de la planificación del movimiento) conozca la secuencia de tareas que se debe realizar, y los puntos a los que debe ir el robot. Para enviar esta información, se ha desarrollado un nuevo mensaje de ROS, denominado *decision_msg*. El mensaje contiene los siguientes campos:

- *arms*: variable entera sin signo de 8 bits que toma valores de 0 a 3. Un valor de 0 corresponde

a una instrucción en la que únicamente debe moverse el brazo izquierdo, mientras que el derecho se mantiene inmóvil. Un valor de 1 corresponde a la instrucción inversa, en la que solamente se mueve el brazo derecho. Un valor de 2 equivale a una instrucción en la que ambos brazos se mueven simultáneamente para recoger frutas. Y por último, un valor de 3 corresponde a una instrucción de recogida de una berenjena ocluida, en la que deben utilizarse los dos brazos, y cuyo funcionamiento se detalla más adelante en la sección 3.6.4.

- *centroids*: se trata de un arreglo de mensajes del tipo *geometry_ msgs/Point*. Cada mensaje representa las coordenadas X, Y y Z del punto al que debe ir el robot para realizar la acción. Se utiliza un arreglo ya que cuando el campo *arms* tome valores de 2 o 3, se enviarán varios puntos en el mismo mensaje. En una instrucción que utilice los dos brazos simultáneamente, se enviarán dos puntos, el primero corresponde al centroide objetivo del brazo izquierdo, mientras que el segundo al del derecho. En el caso de que la instrucción sea para recoger una berenjena ocluida, el primer punto será el correspondiente a la berenjena, el segundo el del centroide de la intersección entre la hoja que lo ocluye y la berenjena, y el tercero es el punto al que debe desplazarse la hoja para dejar de ocluir la berenjena. Nuevamente, el comportamiento de este algoritmo se explica en la sección 3.6.4.
- *orientation*: un arreglo de variables de tipo *float* que contiene la orientación de cada berenjena. El arreglo únicamente contiene más de un valor cuando se trata de una instrucción correspondiente a dos brazos, ya que indica que se deben recoger dos berenjenas.

Este mensaje se crea al final del nodo encargado de la planificación de tareas, a partir de la información obtenida por los diferentes algoritmos implementados. Se publica un mensaje por cada tarea que se requiera realizar. Por ejemplo, si se quisiera realizar una recolección con los dos brazos de forma simultánea, seguida de una recolección con el brazo izquierdo y una recolección de una berenjena ocluida, se publicarían tres mensajes. El primero de ellos tendría un valor para el campo *arms* de 2, el siguiente de 0 y por el último un valor de 3. De esta forma, el planificador de movimientos puede distinguir de forma sencilla cada instrucción y realizarlas una detrás de otra, utilizando las funciones *callback* asociadas a cada subscriptor de ROS. El *topic* utilizado para estos mensajes se ha llamado *grasp_ decision*.

```
# Message used to define an action for KINOVA ARMS MICO2

uint8 ONE_ARM_LEFT = 0
uint8 ONE_ARM_RIGHT = 1
uint8 TWO_ARMS = 2
uint8 OCCLUDED = 3

uint8 arms

# Centroids of the aubs or leaves. Size of centroids depends on arms field
geometry_msgs/Point[] centroids

# Orientation of the aubergines selected
float64[] orientation
```

Código 3.1: Mensaje *decision_ msg*.

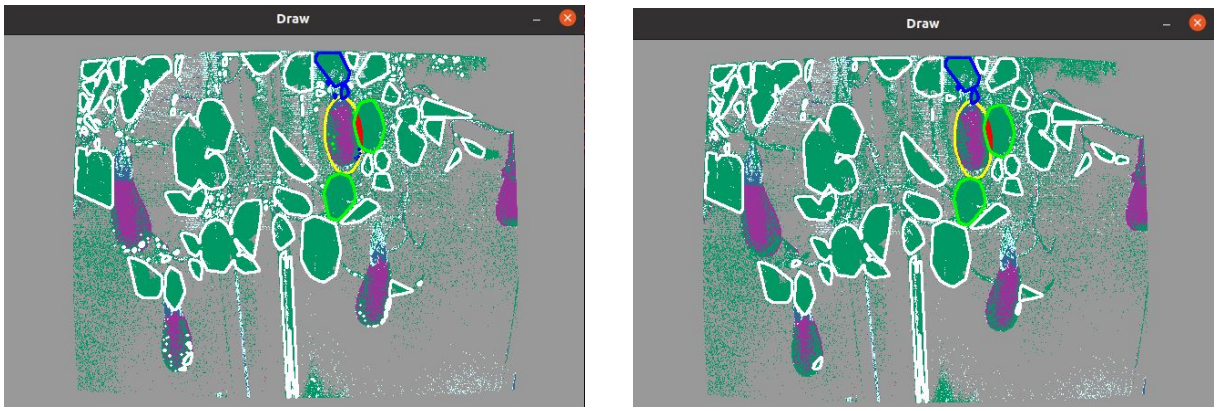
3.5. Nodo de procesamiento de datos

El nodo *data_ management* contiene un subscriptor por cada *topic* creado en el conversor de MATLAB, con la excepción de las imágenes, que no requieren ser tratadas en este nodo. Los

topics se pueden visualizar en la Tabla 3.2. En ROS, cada subscriptor tiene una función *callback* asociada. Esta función es ejecutada cada vez que se recibe un mensaje, y es donde se realizan las operaciones necesarias con los datos, antes de publicarlas en un nuevo *topic* para que sean leídos por el siguiente nodo.

Los subscriptores han sido configurados con una cola de 100 elementos. El tamaño de la cola indica el número de mensajes que se almacenan en memoria en caso de que se reciban mensajes a una velocidad superior a la de ejecución de la función *callback* asociada. El criterio tanto para procesar mensajes, como para desecharlos es, en ambos casos, el de una cola FIFO. Se procesan primeros los mensajes más antiguos, o primeros en llegar, pero también son los mensajes más antiguos los primeros en desecharse. Se ha elegido un tamaño de 100 ya que la magnitud de este número asegura que nunca se desecharán mensajes en ninguno de los tres *topics*.

La función *callback* asociada a los contornos de las hojas (*topic /matlab_ boundLeaves*) realiza un filtro de los mensajes recibidos que tengan menos de 10 puntos almacenados. El programa de visión por computador devuelve un gran número de elementos que se consideran hojas, pero muchos de ellos son ruido o puntos demasiado pequeños como para constituir una hoja por sí mismos. El propósito de este filtro es eliminar estos elementos para que el algoritmo de toma de decisiones únicamente tenga que procesar las hojas reales. El límite se ha establecido en 10 puntos observando los resultados que producía el algoritmo, y determinando que los contornos despreciables solían tener tamaños inferiores a este número. Los mensajes que superan el filtro, son publicados nuevamente en el *topic /boundLeaves*.



(a) Contornos de hojas sin filtrar.

(b) Contornos de hojas filtrados.

Figura 3.6: Comparación entre los contornos de hojas sin filtrar y filtrados.

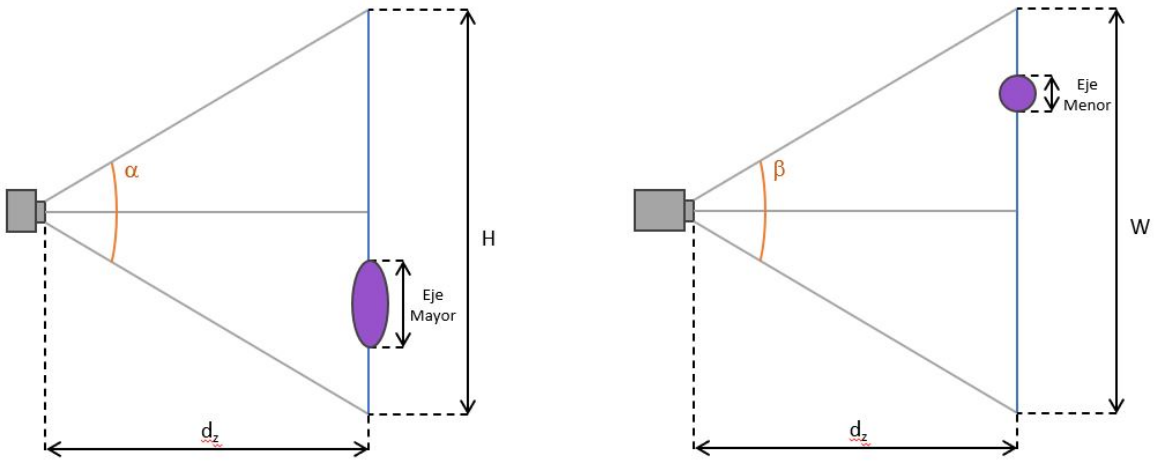
En la Figura 3.6 se puede observar la diferencia producida por el efecto del filtro. Los contornos de las hojas se han marcado en blanco. En la imagen (a) se pueden apreciar una gran cantidad de puntos blancos que representan el ruido obtenido con el nodo de detección y localización. En la imagen filtrada en cambio, estos pequeños puntos desaparecen, restando únicamente los contornos grandes. En el ejemplo mostrado, gracias al filtro, el número de contornos de hojas procesado se reduce de 263 elementos a tan solo 47. Esto aumenta la rapidez de cálculo y además genera imágenes más limpias.

Por otro lado, la función asociada a los contornos de las berenjenas no realiza ninguna operación adicional, ya que el contorno de las berenjenas no necesita ser tratado por el momento. El algoritmo de visión por computador obtiene un dato fiable y preciso. Por lo tanto, la función *callback* de este subscriptor simplemente publica el mensaje según lo recibe en el *topic /boundAubergines*.

Aunque se podría prescindir de ese subscriptor y el funcionamiento del programa no se vería alterado, se ha decidido conservar por dos razones. La primera es para mantener una estructura y un flujo de información equitativo en todos los datos, que simplifique el entendimiento del software desarrollado. La segunda es que el nodo se ha programado con la idea de que el contorno se utilice para segmentar la nube de puntos, y poder así reconstruir tridimensionalmente la superficie visible de las frutas. Esta información tridimensional será de gran ayuda en la planificación del agarre de las frutas. De esta forma, si se quiere incluir esta funcionalidad, simplemente habría que añadir las operaciones necesarias a la función *callback* ya existente, y no sería necesario crearla de nuevo ni modificar la estructura general del nodo.

Por último, la función *callback* asociada a los datos de las berenjenas (*topic /matlab_statsAub*) es donde se realizan la mayor parte de las operaciones. El mensaje *statsAub* contiene la posición de los píxeles que representan el centroide de la berenjena. Sin embargo, estos datos no aportan información útil. En la planificación de tareas resulta más interesante conocer las posiciones de los centroides de las berenjenas en un sistema de coordenadas que permita conocer las distancias reales entre los brazos del robot y las berenjenas. Por lo tanto, es preciso convertir los píxeles de los centroides en coordenadas de posición.

Para realizar esta tarea, es necesario utilizar la información de profundidad proporcionada por la cámara de tiempo de vuelo "Helios2". También es necesario conocer la posición exacta de la cámara y su orientación, para poder construir de forma correcta las matrices de rotación que permitan realizar las conversiones al sistema de coordenadas de la base del robot. Estos últimos datos de la cámara, se han obtenido manualmente de la disposición física del robot, y se encuentran almacenados en una clase de C++ llamada "C". Esta clase contiene toda la información correspondiente a la cámara, como el ángulo de visión, sus ángulos de orientación dados en el sistema de *Roll, Yaw, Pitch*, o alabeo, guiñada y cabeceo, y otros datos referentes a las dimensiones típicas del fruto a recoger, en este caso berenjenas.



(a) Vista de perfil de la cámara.

(b) Vista de planta de la cámara.

Figura 3.7: Vistas de la cámara para calcular el área real.

A partir de la nube de puntos y conociendo los píxeles en los que se encuentra el centroide, se puede obtener su profundidad y obtener sus coordenadas espaciales en el sistema de referencia situado en la cámara. Estas coordenadas se almacenan en una variable definida en una librería de ROS que se llama *geometry_msgs/Point*, y se ha denominado *centerCam*. A continuación, a partir de la orientación guardada en la clase C, se generan las matrices de rotación y se convierte

al sistema de referencia de la base del robot. Las coordenadas obtenidas se almacenan nuevamente en una variable del tipo *geometry_ msgs/Point*, pero llamada en este caso *centerRoot*.

El último dato que se quiere conocer es el área real de la berenjena. El mensaje *statsAub* tiene un campo llamado área, pero hace referencia al número de píxeles que conforman la berenjena. Sin embargo, este número varía en función de la posición y la orientación de la berenjena en la imagen, ya que las berenjenas más próximas a la cámara ocupan un mayor número de píxeles. Por lo tanto, es necesario realizar una transformación para conocer el área real. El cálculo del área se realiza aproximando la forma de la berenjena a una elipse, y aplicando la fórmula del área de esta figura geométrica (Ecuación 3.2).

$$\text{Área} = \frac{\text{Eje Mayor}}{2} \cdot \frac{\text{Eje Menor}}{2} \cdot \pi \tag{3.2}$$

Es necesario, por tanto, conocer la longitud real de los ejes de la berenjena. El nodo de detección y localización proporciona la longitud de los ejes de la berenjena en píxeles. La conversión de este valor a una magnitud de distancia se consigue aplicando un factor de corrección proporcional con respecto a la altura y anchura total de la imagen. Estos valores son desconocidos en un principio, pero pueden ser obtenidos de la relación trigonométrica del ángulo de visión de la cámara y la distancia a la que se encuentra la berenjena. En las Ecuaciones 3.3 y 3.4 se observan estas relaciones, donde H y W son, respectivamente, la altura y el ancho en metros de la imagen y d_z es la distancia entre la cámara y la berenjena. Los ángulos α y β se corresponden con los ángulos de apertura de la cámara y se dan en grados.

$$\tan\left(\frac{\alpha}{2} \cdot \frac{\pi}{180}\right) = \frac{H}{2 \cdot d_z} \tag{3.3}$$

$$\tan\left(\frac{\beta}{2} \cdot \frac{\pi}{180}\right) = \frac{W}{2 \cdot d_z} \tag{3.4}$$

Una vez despejados la altura y el ancho y conociendo la resolución de la imagen (640x480), se obtiene la longitud real de los ejes aplicando el factor de corrección proporcional, tal y como se muestra en las Ecuaciones 3.5 y 3.6. En la Figura 3.7 se puede observar una representación de las vistas de la cámara de donde se han obtenido las relaciones trigonómetricas.

$$\text{Eje Mayor (m)} = \frac{\text{Eje Mayor (píxeles)}}{480 \text{ (píxeles)}} \cdot H(\text{m}) \tag{3.5}$$

$$\text{Eje Menor (m)} = \frac{\text{Eje Menor (píxeles)}}{640 \text{ (píxeles)}} \cdot W(\text{m}) \tag{3.6}$$

Cuando se han obtenido las coordenadas espaciales en referencia al sistema de coordenadas de la cámara y de la base, junto al área real, es necesario publicar esta información en ROS. Para ello, se ha creado un nuevo mensaje personalizado denominado *statsAub_ ext*. Este mensaje, tal y como indica su nombre, es una extensión del mensaje *statsAub*. Incluye los mismos campos con la adición de otros tres nuevos, correspondientes a los datos que se han calculado. La estructura del mensaje resultante se puede observar en el siguiente Código 3.2.

```
uint16 area
float64[] centroid
float64 major_axis_length
float64 minor_axis_length
float64 orientation
float64 solidity
geometry_msgs/Point centerCam
geometry_msgs/Point centerRoot
float64 real_area
```

Código 3.2: Mensaje *statsAub_ext*.

Una vez se han realizado todos los cálculos, el nuevo mensaje extendido es publicado en el *topic* */statsAub*. En la Figura 3.8 se pueden observar de forma gráfica tanto los *topics* a los que se subscribe, como los *topics* que publica el nodo descrito. Los nodos se muestran en elipses, mientras que los *topics* se muestran en rectángulos. El nodo situado a la izquierda, con el nombre de *player* es un *bag* de ROS simulando el comportamiento del conversor de MATLAB.

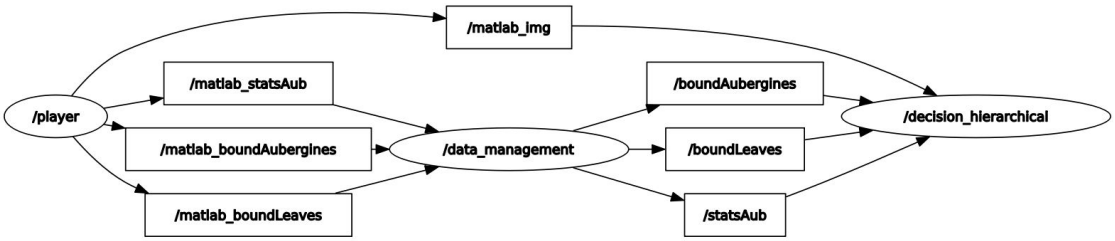


Figura 3.8: Diagrama RQT del nodo *data_management*.

Por último, aunque el conversor desarrollado en MATLAB convierte la nube de puntos a un archivo ".pcd" para que pueda ser leído por cualquier nodo con acceso a la ruta del archivo, el nodo *data_management* también se encarga de publicarla en el *topic* */pointCloud_aub*. Se realiza publicando un mensaje del tipo *sensor_msgs::PointCloud2* a una frecuencia de 10 Hz. De esta forma, para acceder a la nube de puntos, los nodos pueden leer el archivo ".pcd" o subscribirse a este *topic* para acceder a la información. Este *topic* también sirve para visualizar la nube de puntos en herramientas gráficas como "Rviz", que requieren de un publicador periódico para mostrar la información por pantalla.

3.6. Nodo de toma de decisiones

El nodo destinado a la toma de decisiones, o planificación de tareas a alto nivel, es donde se ejecutan los algoritmos que determinan el tipo de recolección que se debe realizar con cada berenjena detectada y el orden en el que se procesarán los comandos de acción. Por tanto, el objetivo de este nodo es crear la secuencia de mensajes *decision_msg* (descrito en la Sección 3.4) que describe los comandos que debe ejecutar el robot para llevar a cabo la recolección.

El nodo se subscribe a los *topics* creados por el nodo *data_management* para utilizar toda la información proporcionada por éste. Una vez dispone de toda la información, comienzan los algoritmos de planificación de tareas. Estos algoritmos se han definido en la función *callback* asociada a un *topic* denominado */calc_Flag*. Este *topic* está configurando para mensajes *std_msgs/Bool*,

que contienen una variable booleana, de "Verdadero" o "Falso". De esta forma, cuando se quiera comenzar la planificación de tareas, simplemente hay que publicar un mensaje en este *topic* con el valor "Verdadero". Antes de comenzar la planificación, el código comprueba que se han recibido correctamente todos los datos. En caso de que no sea así, informa a través de la consola mediante las funciones de error de ROS.

El *callback* asociado al *topic /calc_Flag* llama a dos funciones, primero a *arms_grabDecision()* y, a continuación, a *arms_Occluded()*. En la primera función se define el algoritmo que se utiliza para realizar la toma de decisiones. Estos algoritmos se explican detalladamente en las próximas secciones, pero todos se basan en la clasificación y en el ordenamiento de las berenjenas que deben ser recogidas. Por otro lado, la segunda función ejecuta el algoritmo encargado de manipular las berenjenas ocluidas.

Aunque el robot debe operar de forma completamente autónoma, en el estado actual la publicación de este mensaje que activa la toma de decisiones se realiza de forma manual, a través de la consola del sistema. Esto se debe principalmente al tipo de pruebas que se quieren llevar a cabo con el prototipo en este Trabajo Fin de Máster, donde se pretende realizar experimentos concretos y supervisados por un humano, para evaluar los distintos algoritmos propuestos. El flujo de información es, por tanto, discontinuo.

Con el objetivo de poder implementar y probar el funcionamiento de los distintos algoritmos propuestos, se han diseñado numerosos nodos, que son intercambiables entre sí. De esta forma, para cambiar de planificador de tareas es suficiente con ejecutar un nodo u otro. Todos ellos comparten los mismos subscriptores y publicadores, ya que sus entradas y salidas son las mismas.

Para acortar el código elaborado, se ha implementado un sistema de herencia de C++. La clase madre se llama *Decision*. En esta clase se definen todos los subscriptores, publicadores, variables y funciones que deben tener en común cada uno de los nodos para funcionar correctamente. Así, cuando una clase hereda de ésta, obtiene inmediatamente la estructura que necesita el nodo para operar en esta "posición", y simplemente debe reescribir la función donde se define el algoritmo utilizado, que es *arms_grabDecision*. Ésta se ha declarado como una función virtual pura en la clase madre, para que sea reemplazada por las clases derivadas.

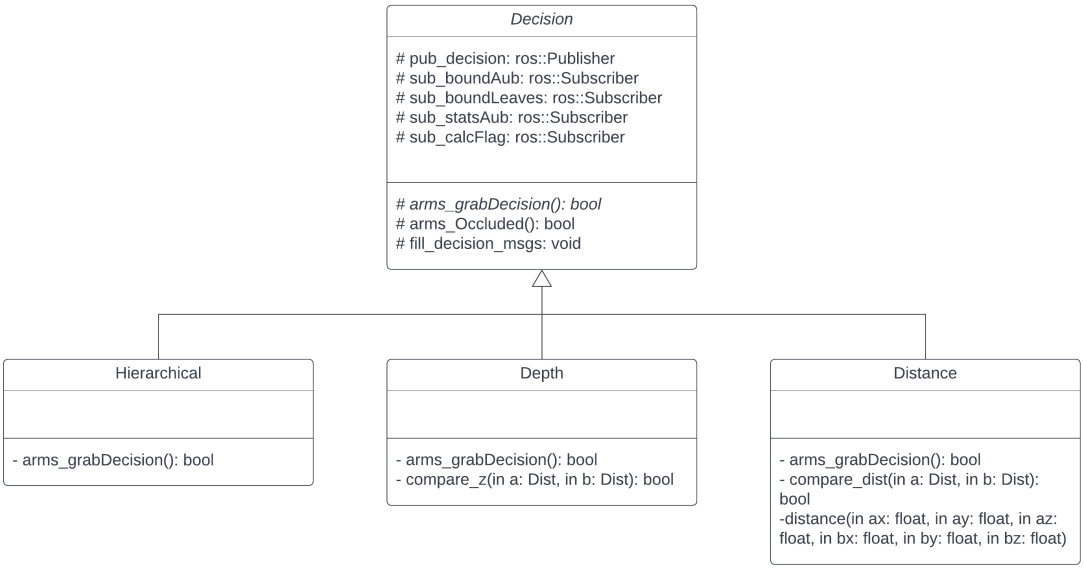


Figura 3.9: Diagrama de clases conceptual del nodo de toma de decisiones.

Declarar una función como virtual pura permite no realizar su implementación en la clase base, y obliga a todas las clases derivadas a definir esta función. En este caso, la clase madre se convierte en una clase abstracta. Este tipo de clases se caracterizan por tener al menos una función virtual pura, y el efecto producido es que no se pueden instanciar objetos de esta clase, ya que el compilador devolvería un error al intentar crear un objeto de una clase que tiene una función que no está definida. Las clases derivadas deben implementar esta función para dejar de ser clases abstractas, como la clase de la que heredan. Este comportamiento es justo el deseado para la clase *Decision*, ya que no se busca instanciar objetos de esta clase, si no que sirva de base para otras.

Si se requiere implementar un nuevo algoritmo de toma de decisiones, se puede proceder creando una nueva clase en el paquete que herede de la clase *Decision*. A continuación, solamente hay que implementar el algoritmo deseado en la función *arms_ grabDecision*, ya que los demás elementos de la clase ya han sido definidos en la clase principal.

Los principales atributos y métodos de la clase *Decision* son los siguientes:

- Subscriptores y Publicadores del nodo.
- Funciones *callbacks*: cada subscriptor tiene una función asociada para almacenar el mensaje recibido en una variable local de la clase, que puede ser utilizada por cualquiera de sus funciones miembro. En las funciones asociadas a los contornos de las berenjenas y de las hojas es donde se realiza la reconstrucción del mensaje *boundAub* a una matriz de coordenadas, tal y como se ejemplificaba en la Ecuación 3.3.
- Vectores de datos y variables: son las estructuras donde se almacena la información recibida mediante los subscriptores. Estos atributos de la clase se reinician después de realizar la planificación de tareas, para que el dato almacenado no vuelva a ser utilizado.
- Funciones destinadas a la creación del mensaje que se utiliza como salida del nodo.
- Función *arms_ Occluded()*: se trata del algoritmo utilizado para la recolección de berenjenas ocluidas. Su funcionamiento se explica en la Sección 3.6.4.

El paquete cuenta con tres nodos de toma de decisiones, cada uno con su propia clase derivada de *Decision*. El primero de ellos se denomina *decision_ hier*, y se implementa a través de la clase *Hierarchical*. El segundo nodo tiene el nombre de *decision_ depth* y utiliza la clase *Depth*. Por último, el tercer nodo, denominado *decision_ distance*, instancia un objeto de la clase *Distance*. Las tres clases tienen sus propios archivos de encabezado (.h) y archivos de implementación (.cpp), manteniendo la estructura del paquete.

En la Figura 3.9 se puede observar un diagrama de clases conceptual de la clase abstracta *Decision* y las tres clases que heredan de ella. Se muestran únicamente los atributos y funciones más significativos, incluyendo la función virtual pura. Además, en la Tabla 3.3 se muestra un resumen de los *topics* a los que se subscriben los nodos encargados de la planificación de tareas. No se incluye el *topic* donde se publica la nube de puntos ya que ésta puede ser leída desde un archivo ".pcd".

Dato	*Topic*	Mensaje
Datos de Berenjenas	/statsAub	statsAub.msg
Contorno de Berenjenas	/boundAubergines	boundAub.msg
Contorno de Hojas	/boundLeaves	boundAub.msg
Imagen Procesada	/matlab_img	sensors_msgs/Image
Topic de control	/calcFlag	std_msgs/Bool

Tabla 3.3: Resumen sobre *topics* y mensajes utilizados en el nodo de toma de decisiones.

3.6.1. Decisión jerárquica

La clase *Hierarchical* es la clase más básica que se ha desarrollado. Es el nodo que menos datos utiliza para realizar la planificación de tareas y, por tanto, es el que debería proporcionar los resultados menos óptimos. El algoritmo implementado es el que utilizaba la arquitectura de software previa a este trabajo en su nodo en MATLAB para realizar la toma de decisiones.

Este algoritmo, definido en la función *arms_grabDecision()* de la clase, comienza clasificando todas las berenjenas que se visualizan en la imagen, y de las cuales se disponen datos, entre berenjenas ocluidas y no ocluidas. Para realizar esta clasificación, se compara el valor del área real, que ha sido calculado en el nodo *data_management*, con una constante que ha sido determinada empíricamente. En algunos frutos, el área desarrollada puede ser un indicador del grado de madurez de la fruta de estudio, como es el caso de las berenjenas. Conociendo este valor a partir del cual se considera que la fruta ha alcanzado un grado de madurez óptimo, se puede obtener una constante que ayude a determinar el área que debería tener una berenjena. Por tanto, si el área real de la berenjena es mayor que la constante definida, se considera como berenjena no ocluida, mientras que si el área real es menor, se considera como berenjena ocluida. El funcionamiento del modelo se basa en comparar la visibilidad del fruto captada por la cámara, con un modelo preestablecido.

A continuación, las berenjenas no ocluidas se reparten en dos zonas. La primera zona corresponde al sector izquierdo del robot, y la segunda al sector derecho. La división se obtiene directamente con las coordenadas del centroide de la berenjena en el sistema de coordenadas del robot. Es decir, mediante el dato *centerRoot*.

Esta división, aunque simple, es vital para realizar la recolecta de frutas. Aunque el robot disponga de dos brazos robóticos, estos son controlados de forma independientes, por lo que a efectos prácticos se pueden considerar como dos robots separados que comparten información entre ellos. Dividir el área de trabajo total del robot en dos mitades, una para cada brazo, disminuye las probabilidades de que se produzcan colisiones entre los brazos. Además, este método permite la implementación de una recolecta dual y simultánea, en la que cada brazo actúa de forma independiente, ya que trabajan en áreas distintas. Únicamente hay que tener especial cuidado, en la frontera entre ambas zonas de trabajo, donde se pueden producir colisiones.

En este punto del proceso, se dispone de una clasificación de las berenjenas en:

- No ocluidas del lado izquierdo.
- No ocluidas del lado derecho.
- Ocluidas.

El único paso restante para completar la planificación de tareas es asignar un comando u orden a cada brazo, a partir de esta clasificación. Para optimizar la recolecta, siempre se intenta implementar la recolección simultánea de los dos brazos en primer lugar. Para ello, se realiza una comprobación del número de berenjenas no ocluidas en cada sector, y siempre que sea factible se asigna una orden de recogida de los dos brazos al mismo tiempo. Esto sólo es posible si hay berenjenas en los dos sectores. Si no es así, se utilizarán órdenes de recolección de un único brazo.

Por ejemplo, véase el caso de disponer de 3 berenjenas no ocluidas en el lado izquierdo y 2 berenjenas no ocluidas en el derecho. Tras la comprobación del número de berenjenas en cada sector, se crearán dos órdenes de recolección simultánea, ya que hay al menos dos frutas en cada sector. Sin embargo, la tercera orden deberá ser de un solo brazo, ya que una vez se ejecuten las dos primeras, únicamente restará una berenjena en el lado izquierdo, por lo que la recolección simultánea no es posible.

Finalmente, se ejecuta un algoritmo encargado de asignar los comandos asociados a la recolección de las berenjenas ocluidas. Este algoritmo se ha implementado mediante la función *arms_ Occluded()*, que se ha definido en la clase *Decision*, ya que es común a cualquier nodo de toma de decisiones. Debido a la complejidad del algoritmo, su explicación se ha realizado en una sección aparte (3.6.4).

Como se ha podido observar, el algoritmo es relativamente sencillo, pero también completamente lógico y eficaz. Para acelerar el proceso de recolección se asigna preferencia a las órdenes de recolección simultánea y se divide el área de trabajo en dos mitades iguales, una para cada brazo. Las berenjenas ocluidas se recogen al final ya que su proceso de recolección requiere del uso de ambos brazos. Mejorar este sistema es complicado, ya que hay que recordar que la base del robot se encuentra inmóvil durante las pruebas.

El único parámetro que puede ser modificado y crear alteraciones en el comportamiento del proceso, es el orden de recolección asignado. La clasificación de berenjenas se realiza recorriendo un vector de datos, que pertenece a la clase, en orden ascendente. Este vector ha sido creado a partir de la información proporcionada por el módulo de detección y localización, por lo que el resultado es un vector con las berenjenas ordenadas de izquierda a derecha de la imagen. Esto puede provocar situaciones en que el orden de recogida no sea óptimo o, por ejemplo, que dos frutas consecutivas en el vector de datos estén situadas en extremos verticales opuestos.

Los otros algoritmos propuestos en este documento, se basan en ordenar el vector de berenjenas de diversas formas según distintos parámetros, para buscar el criterio de ordenamiento más eficiente. Sin embargo, el mecanismo de clasificación entre berenjenas ocluidas y no ocluidas, y su posterior división en el sector izquierdo y el derecho, tal y como se han descrito aquí, se mantienen prácticamente igual.

3.6.2. Decisión en función de la profundidad

Este segundo algoritmo busca ordenar el vector de datos que incluye la información sobre las berenjenas en función de su profundidad en la nube de puntos proporcionada por la cámara de tiempo de vuelo.

El vector de datos que contiene cada clase se ha implementado a través de una estructura *std::vector* de la librería *vector* de C++, y se define en la clase abstracta *Decision*. Se trata de un vector que almacena variables de tipo *decision_ pkg::statsAub_ ext*, y es un atributo privado de la clase. Cada vez que se recibe un mensaje en el *topic /statsAub*, la función *callback* asociada

a este subscriptor, añade el mensaje recibido como un nuevo elemento del vector. De esta forma, el orden del vector viene determinado por el orden de publicación del anterior nodo, que utiliza, a su vez, el orden proporcionado por el módulo de detección y localización. El nombre asociado a este vector de datos es *aubergines*.

Cuando se realiza la clasificación de las berenjenas descrita en la sección anterior (3.6.1), se crean tres vectores adicionales, *leftArm*, *rightArm* y *occludedAubs*, uno por cada grupo. Con el objetivo de ahorrar espacio y evitar duplicar variables, en estos vectores únicamente se almacena el índice que indica la posición de la berenjena a la que hace referencia. Es decir, si la primera berenjena del vector *aubergines* es una berenjena no ocluida que se encuentra en el lado derecho, al vector *rightArm* se le añadirá el elemento "0". Si la segunda berenjena estuviera ocluida, al vector *occludedAubs* se le añadiría el elemento "1", y así sucesivamente.

Los elementos añadidos a estos vectores son los índices que se usan para acceder al vector de datos *aubergines*. Para simplificar el código y evitar errores de acceso a elementos inexistentes, los índices comienzan en cero. Así, si se quisiera acceder a todas las berenjenas de un grupo, solamente habría que utilizar un bucle *for* que iterara sobre el vector del grupo, y utilizar los elementos de este vector como los índices para acceder al vector de datos. Se muestra un ejemplo en el siguiente Código 3.3.

```
for (uint i=0; i< leftArm.size(); i++) { //Itera por el vector
    //Se accede al area real de la berenjena
    real_area = aubergines[leftArm[i]].real_area;
}
```

Código 3.3: Ejemplo de acceso a los datos de una berenjena.

De esta forma, se puede realizar la clasificación de las berenjenas en la imagen sin crear duplicados de los datos originales, solamente almacenando índices en vectores distintos. Gracias a este diseño, se puede alterar el comportamiento del nodo variando el orden original del vector *aubergines*, que es lo que realiza el nodo *decision_ depth* a través de la clase *Depth*.

Antes de aplicar la clasificación de berenjenas, se reordena el vector de datos según la profundidad de cada berenjena, en orden creciente. Esto se realiza a través de las herramientas proporcionadas por la librería de C++ *vector*, junto a una estructura de datos auxiliar y una función creadas para este propósito. Una vez se ha ordenado, se puede aplicar la misma clasificación que se realiza en la clase *Hierarchical*, pero el resultado será distinto, ya que en este segundo caso el índice "0" no hará referencia a la primera berenjena detectada por el módulo encargado de la detección y localización, sino que se referirá a la berenjena con menor profundidad.

El valor utilizado para ordenar las berenjenas por profundidad se encuentra en el sistema de coordenadas de la base del robot. Este dato se ha calculado en el nodo anterior *data_ management* y se almacena en la variable *centerRoot*. Se debe tener en cuenta, no obstante, que en este sistema de coordenadas la profundidad se da en el eje Y, en vez de en el Z. Es decir, los sistemas de coordenadas de la cámara y de la base del robot no tienen la misma orientación. Aunque las coordenadas transformadas se calculen automáticamente mediante las matrices de rotación, hay que tener en cuenta esta peculiaridad a la hora de acceder a la variable para no utilizar otro parámetro que no corresponde con el real. En la Figura 3.10 se puede observar la relación entre los ejes de coordenadas de cada sistema.

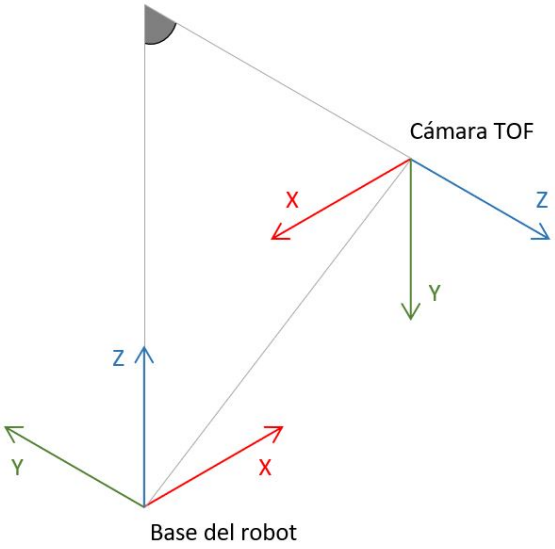


Figura 3.10: Sistemas de referencia de la base del robot y de la cámara TOF.

El criterio para utilizar la profundidad como un parámetro de control se basa en la idea de que recolectar las berenjenas más cercanas primero, simplifica el proceso de recolecta. Esta metodología es similar a la que utilizaría un humano ante la misma tarea. Primero, se recogen las berenjenas más cercanas, que serán las más sencillas de recoger. A continuación, se recogen las más profundas, cuando las berenjenas más próximas ya han sido recolectada y no pueden obstaculizar. Este algoritmo puede ser útil cuando se planteen situaciones donde haya muchas berenjenas en una misma planta, y las oclusiones no se produzcan por hojas, si no entre una o más berenjenas.

3.6.3. Decisión por distancia euclídea

El nodo *decision_distance* implementa, a través de la clase *Distance*, un algoritmo similar al anterior, solo que en esta ocasión el parámetro utilizado es la distancia euclídea entre frutas. La distancia euclidiana es la distancia "convencional" entre dos puntos, es decir, la longitud del segmento que conecta estos dos puntos en el espacio. Su cálculo se realiza a través del teorema de Pitágoras. Aplicado a un espacio tridimensional, la distancia entre dos puntos A = (a_x, a_y, a_z) y B = (b_x, b_y, b_z) viene determinada por la siguiente ecuación:

$$D = \sqrt{(a_x - b_x)^2 + (a_y - b_y)^2 + (a_z - b_z)^2} \tag{3.7}$$

El objetivo de este nodo es implementar un sistema de recolección de frutas donde el orden de recogida se determine por la fruta más cercana a la última recogida, buscando un tiempo de recolección total inferior al que se pudiera conseguir con otras metodologías. El primer elemento de recolección es determinado a partir de la distancia más corta a la posición de reposo del robot, de forma que se minimice el recorrido realizado por el efector final.

No obstante, hay dos puntos que pueden afectar al desempeño del algoritmo, y que por lo tanto deben ser considerados. El primero de ellos es que en robótica manipuladora, la distancia más corta no siempre corresponde con el movimiento más simple. El movimiento de un robot manipulador de 6 grados de libertad se produce mediante la rotación de sus articulaciones. El

movimiento que genera cada una de estas articulaciones por sí misma es no lineal. De hecho, para conseguir un movimiento lineal respecto a los ejes cartesianos, es necesario la activación de numerosas articulaciones de forma simultánea. Por tanto, es posible que un movimiento cuya distancia euclidiana sea más corta, corresponda a un movimiento que requiera un mayor tiempo de ejecución, al producir un mayor cambio en la rotación de sus articulaciones.

En segundo lugar, hay que tener en cuenta que después de la recogida de una fruta, ésta debe ser depositada. Este proceso se incluye en el módulo de planificador de trayectorias, pero tiene un efecto directo en el tiempo necesario para completar la recolección de todas las frutas de una escena.

La motivación, por tanto, de este algoritmo, no es buscar un tiempo de ejecución menor, aunque siempre es un parámetro que conviene reducir. Si no aplicar una lógica de recolección similar a la que utilizaría un ser humano, escogiendo sus objetivos en grupos, o por la proximidad entre ellos. Este comportamiento presenta una gran ventaja a largo plazo, cuando se produzca la integración de los brazos manipuladores con la base móvil. Recoger berenjenas que se encuentran próximas entre sí facilita la integración entre el robot recolector y el robot encargado de la navegación. Así, cuando se recolecten las berenjenas de una zona en concreto, el robot podría moverse buscando nuevos objetivos. Este comportamiento también presenta menores probabilidades de ser obstaculizado por otras berenjenas, ya que a medida que se van recogiendo, liberan un espacio que puede ser utilizado por el brazo para recolectar otras frutas.

Aunque la metodología utilizada es similar a la descrita en la clase *Depth*, la clase *Distance* no puede utilizar el mismo procedimiento. Esto se debe a que el anterior nodo ordenaba el vector de datos *aubergines* en función de la profundidad, y a continuación realizaba la clasificación de berenjenas según su posición y su área. Este procedimiento no es válido en este nuevo caso, ya que se pretende ordenar las berenjenas que va a recoger cada brazo en función de la proximidad entre ellas, pero no todo el conjunto de las berenjenas sin clasificar. Es decir, hay que ordenar el conjunto de berenjenas del sector izquierdo y el derecho por separado. Si no se hiciera así, se pudiera dar el caso de que dos berenjenas situadas en el centro estén muy próximas entre sí, pero fueran recogidas por brazos distintos. Aplicar el mismo método, por tanto, resultaría en unas órdenes erróneas.

Por este motivo, en este nodo es necesario realizar primero la clasificación de berenjenas entre ocluidas y no ocluidas de cada sector, y a continuación ordenar los vectores de clasificación *leftArm* y *rightArm*. Es preciso destacar la diferencia de comportamiento entre ambos algoritmos. El algoritmo basado en la profundidad ordena el vector de datos *aubergines*, de manera que al aplicar los índices que se encuentran en los vectores de clasificación (que tendrán un orden secuencial), se tratan primero las berenjenas con menor profundidad. Este segundo algoritmo, en cambio, ordena los vectores de clasificación, pero mantiene intacto el vector de datos *aubergines*, de forma que al recorrer el vector de clasificación mediante un código similar al mostrado en Código 3.3, los índices que se utilizarán no estarán en orden secuencial.

En la Figura 3.11 se muestra de forma esquemática el orden de las operaciones realizadas en los tres nodos descritos, junto al resultado que tiene cada operación en los datos de la clase asociada al nodo. Las berenjenas se han nombrado utilizando letras del alfabeto, y los índices de los vectores de clasificación son números. En el bloque de "Resultado" se muestra la berenjena a la que hace referencia cada vector de clasificación, pero hay que recordar que estos vectores almacenan índices realmente. Para mayor claridad, se ha omitido el vector destinado a las berenjenas ocluidas.

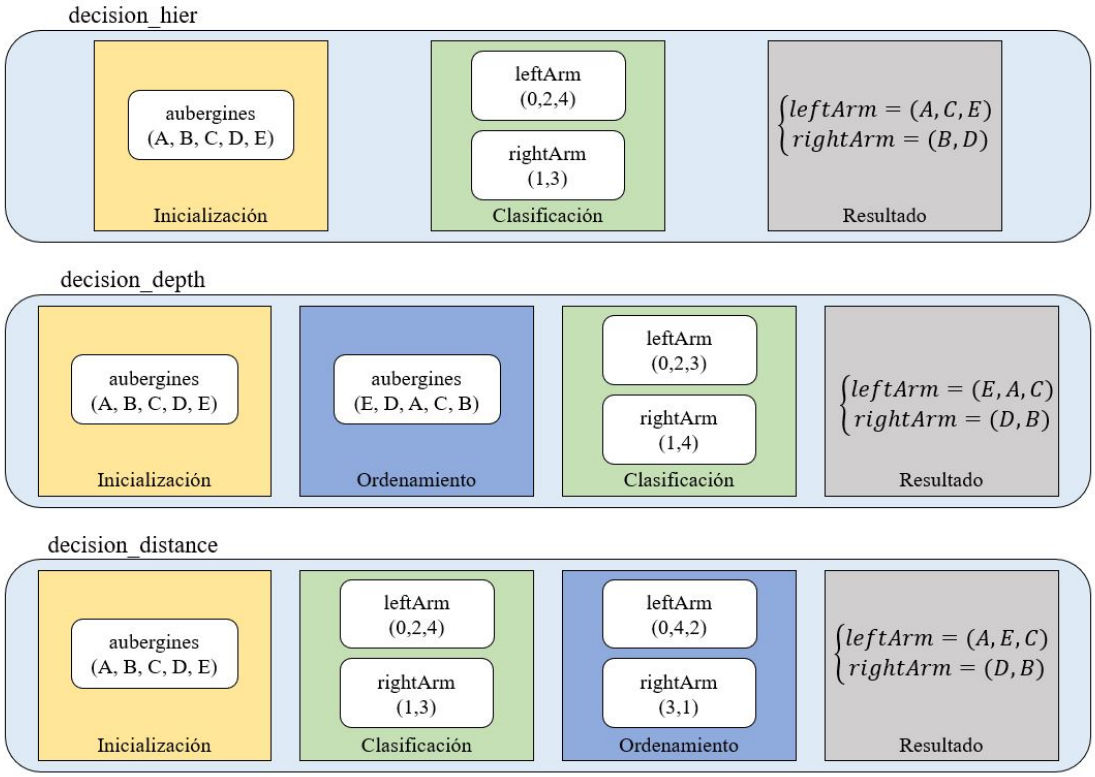


Figura 3.11: Flujo de datos en los algoritmos propuestos.

Se puede ver como en el primer nodo no se produce ordenamiento de ningún tipo, por lo que las berenjenas en "Resultado" tienen el mismo orden que *aubergines*. En el segundo nodo en cambio, se produce un ordenamiento del vector de datos según la profundidad. La clasificación se realiza acorde al nuevo vector de datos (se mantiene el orden secuencial de los índices) y por tanto el resultado obtenido es distinto. Por último, en el nodo *decision_ distance* se realiza primero la clasificación y, a continuación, se ordenan los vectores de clasificación (pierden el orden secuencial). En esta ocasión, las berenjenas a las que hace referencia cada vector tienen otro orden distinto. Hay que destacar que aunque cada algoritmo obtiene un orden diferente, es imposible que se entremezclen berenjenas de un vector de clasificación con otro. Esta clasificación siempre se realiza con los datos almacenados en el vector *aubergines*, y siempre tendrá el mismo resultado.

3.6.4. Algoritmo para berenjenas ocluidas

Como se ha explicado al comienzo de la sección 3.6, la toma de decisiones comienza cuando se publica un mensaje con un valor "Verdadero" en el *topic /calc_ Flag*. En este momento, el *callback* de este subscriptor realiza las llamadas a las funciones *arms_ grabDecision()* y *arms_ Occluded()*.

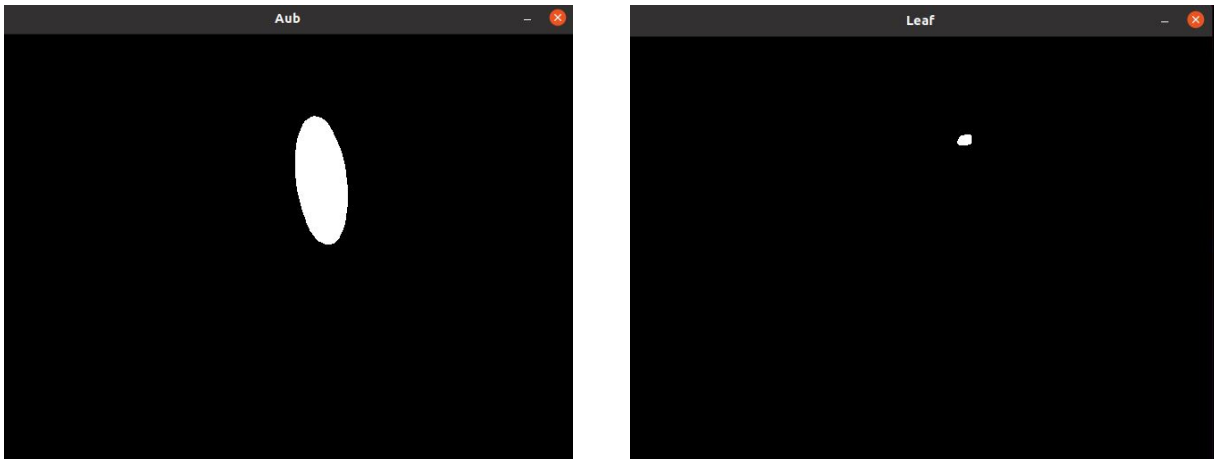
El comportamiento de la primera función se ha descrito en las secciones anteriores. Esta función abarca la clasificación de berenjenas en tres grupos: "no ocluidas del sector izquierdo", "no ocluidas del sector derecho" y "ocluidas". Después, se realiza el ordenamiento de los vectores encargados de asociar las berenjenas a cada grupo. La segunda función, por otro lado, tiene como único objetivo implementar el algoritmo necesario para llevar a cabo la recolección de las berenjenas ocluidas. El procedimiento utilizado es es el que se describe a lo largo de esta sección.

El algoritmo se emplea con las berenjenas que han sido catalogadas como "ocluidas" por la función anterior, ya que no cumplían con los requisitos mínimos de área. Una berenjena puede no cumplir este requisito por dos razones. O bien no ha alcanzado el grado de madurez suficiente, y por tanto no debe ser recogida, o en el caso contrario, se encuentra parcialmente ocluida y sí debe ser recogida. Para comprobar qué situación corresponde a cada berenjena, es necesario comprobar si existen hojas próximas a la berenjena que estén provocando una oclusión.

Para comprobarlo, se dibuja una elipse auxiliar a partir del centroide de la berenjena ocluida. Como el tamaño que pudiera tener la berenjena es desconocido, para modelar la elipse se utilizan los datos geométricos de otras berenjenas en la escena o, en caso de que no existan, unos valores predeterminados. Estos valores se han obtenido empíricamente. Una vez se dispone de esta elipse, se verifica si existen intersecciones entre ésta y las hojas identificadas por el módulo de detección y localización. En caso de que existan intersecciones, se tratará de una berenjena ocluida. Si no es así, se descartará la berenjena al no cumplir los requisitos de madurez establecidos.

Para realizar las intersecciones entre hojas y la elipse auxiliar, se han utilizado las librerías *OpenCV* y *cv_bridge*. La primera consiste en un software dedicado a la visión por computador, y la segunda es una librería propia de ROS que se utiliza como interfaz entre los dos marcos de trabajo. Para verificar si existe alguna intersección se realiza una operación binaria "AND" sobre dos imágenes auxiliares binarias, es decir, cuyos píxeles pueden tener los valores lógicos de cero o uno. En la primera imagen se representa la elipse de la berenjena y en la segunda imagen se representa la hoja que se quiere estudiar. Es decir, estas imágenes tienen todos sus píxeles con un valor lógico de cero, excepto aquellos que conformen el área de la elipse o el área de la hoja. Esta operación se repite con todas las hojas almacenadas en el vector *boundLeaves*.

La operación binaria "AND" devuelve una tercera imagen que es el resultado de aplicar una operación lógica convencional píxel por píxel entre ambas imágenes. Solamente tendrán un valor lógico de 1, aquellos píxeles que no sean nulos en ambas imágenes, es decir, aquellos píxeles en los que se produce una intersección entre la hoja y la elipse auxiliar. De esta forma, realizando un conteo de los píxeles con valor lógico de 1 en la imagen devuelta por la operación, se puede saber si existe intersección, y el tamaño de dicha intersección, dado en número de píxeles. En la siguiente Figura 3.12 se muestran dos ejemplos de las imágenes utilizadas durante este proceso.



(a) Elipse auxiliar.

(b) Una de las hojas de la escena.

Figura 3.12: Ejemplos de imágenes binarias utilizadas para el cálculo de la intersección.

La construcción de las imágenes auxiliares a partir de los datos de *boundLeaves* puede ser realizada con distintas configuraciones ofrecidas por *OpenCV*. Por ejemplo, uno de los parámetros

configurable es el modelado del contorno de la hoja. Se pueden utilizar píxeles con aristas adyacentes entre sí o píxeles que compartan al menos un vértice en común. La diferencia entre estos tipos de línea reside en que con la primera configuración, se considera que un píxel solamente puede tener 4 píxeles adyacentes (uno por cada arista), mientras que con la segunda puede tener hasta 8 píxeles adyacentes (uno por cada arista y otro por cada vértice). Estos parámetros ayudan a caracterizar el comportamiento del algoritmo, configurando cada proceso de forma que proporcione resultados más precisos.

En este punto del algoritmo, se conoce si la berenjena está siendo realmente ocluida por una hoja o, si por el contrario, debe ser descartada. Si se trata del primer caso, es decir, se ha detectado alguna intersección, se procede a calcular los puntos necesarios para su recolección, mientras que si se trata del segundo, finaliza el algoritmo y se repite para la siguiente berenjena ocluida.

Para el primer caso, se comprueba si existe más de una oclusión, ya que es probable que una berenjena sea ocluida por múltiples hojas. A continuación, se descartan las intersecciones producidas por hojas que tienen mayor profundidad que el centroide de la berenjena, es decir, que se encuentran más alejadas de la cámara. Esta situación es factible debido a la creación de una elipse auxiliar alrededor de la berenjena ocluida. Esta elipse puede intersectar con hojas que realmente no están produciendo ninguna oclusión y que se encuentran al fondo de la escena. Realizando este filtro, se evita utilizar datos erróneos. Por último, se selecciona la hoja que produce una mayor intersección, ya que será la hoja que genere una mayor oclusión.

La recolección de una berenjena ocluida requiere la intervención de los dos brazos del robot. Uno de ellos se encargará de apartar la hoja de la escena, y el otro tendrá la tarea de agarrar y recolectar la berenjena. Para determinar qué brazo debe realizar cada tarea, se calcula un vector que indica la dirección en la que debe apartarse la hoja. Si el vector indica un movimiento hacia la izquierda, será el brazo de este sector el encargado de apartar la hoja. En el caso contrario, será el brazo del sector derecho.

El vector se calcula a partir del centroide de la berenjena ocluida y del centroide de la intersección entre la hoja y la elipse auxiliar. Estos dos puntos trazan una recta que indica la dirección en la que se apartará la hoja. La posición final del vector se determina a partir de un parámetro configurable que indica la distancia que debe desplazarse la hoja. De esta forma, se obtienen tres puntos de interés:

- El centroide de la berenjena ocluida.
- El centroide de la intersección entre la hoja y la elipse auxiliar alrededor de la berenjena.
- El punto al que debe moverse la hoja para que desaparezca la oclusión.

Estos tres puntos se convierten a continuación al sistema de coordenadas de la base del robot y se envían dentro del campo *centroids* del mensaje *decision_ msg*. En la Figura 3.13 se muestra un ejemplo del mensaje publicado que recibe el planificador de trayectorias. En él, se pueden observar los tres puntos que se utilizan para llevar a cabo la recolección de una berenjena ocluida.

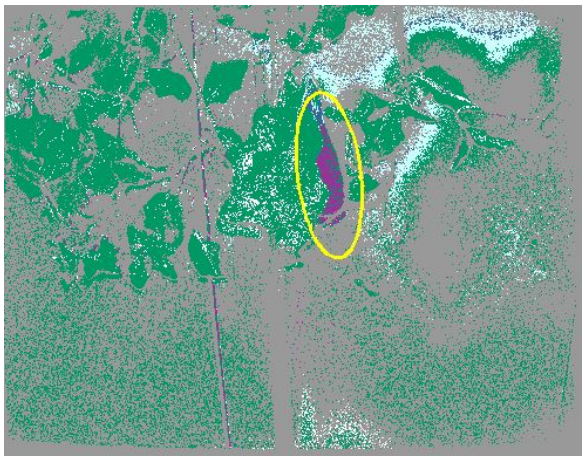
Figura 3.13: Mensaje publicado con el formato definido en *decision_ msg*.

El segundo punto, correspondiente a la intersección, será el objetivo inicial del brazo encargado de apartar la hoja. Cuando haya agarrado la hoja que se encuentra en esta localización, se desplazará hasta el tercer punto. El brazo recolector, acudirá entonces al primer punto para recolectar la berenjena.

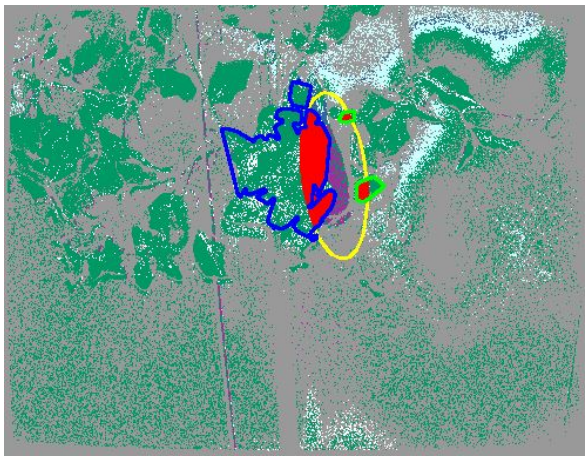
En la Figura 3.14 se muestra todo el proceso por pasos de forma visual. Estas imágenes han sido obtenidas a través de la librería *OpenCV*, que ofrece herramientas para poder visualizar todo el proceso a medida que se van detectando y procesando las hojas. No obstante, aunque estas herramientas son necesarias durante la fase de pruebas y *debugging*, han sido eliminadas del código final. Mostrar imágenes por pantalla consume recursos del ordenador y no son necesarias para el funcionamiento del programa.

En la imagen (b) de la Figura 3.14 se pueden observar tres hojas que intersectan con la elipse. La primera de ellas, que es la más grande, tiene su contorno dibujado en azul, mientras que las otras dos más pequeñas, tienen su contorno dibujado en verde. Estos colores se utilizan para distinguir las hojas que se encuentran por delante y por detrás de la berenjena. Es decir, las hojas que tienen menor y mayor profundidad que la berenjena, respectivamente. El color azul se utiliza para aquellas que se encuentran por delante, y por lo tanto pueden generar oclusión. Mientras que el color verde se utiliza para las hojas que no pueden generar oclusión. El color rojo, por otro lado, representa el área de intersección.

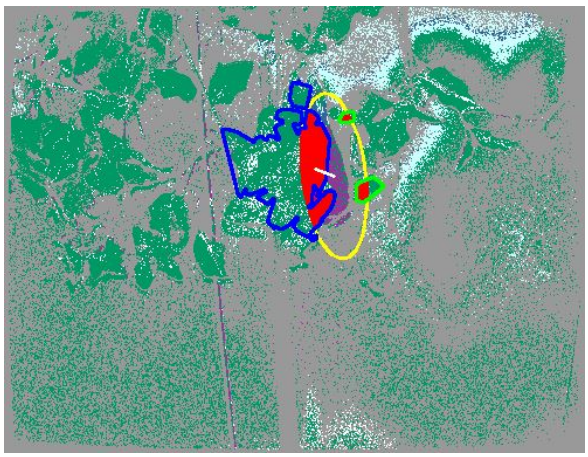
En este ejemplo se puede observar lo acertado de la decisión de descartar las intersecciones de hojas con mayor profundidad que el centroide de la berenjena. Las dos hojas más pequeñas se encuentran situadas por detrás de la berenjena, pero al generar una elipse auxiliar se produce una intersección con ésta, aunque no generan ningún tipo de oclusión. Es necesario, por tanto, descartarlas a la hora de seleccionar la mayor intersección para evitar resultados incorrectos.



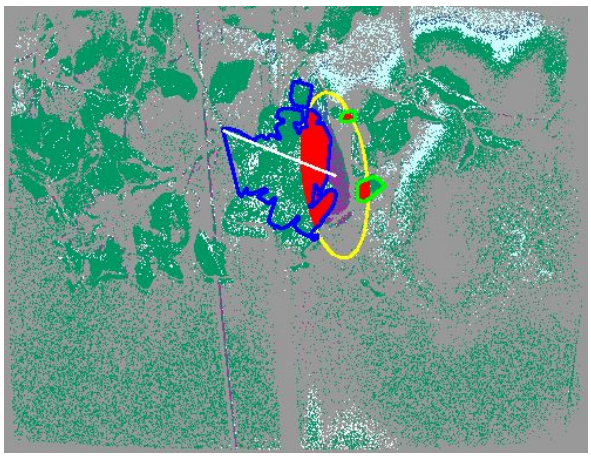
(a) Elipse auxiliar de la berenjena ocluida.



(b) Intersección en rojo entre hojas y elipse.



(c) Segmento entre el centroide de la berenjena y de la mayor intersección.



(d) Extensión del segmento hasta el punto objetivo para apartar la hoja.

Figura 3.14: Procedimiento paso a paso del algoritmo de oclusión.

3.7. Integración con la arquitectura existente

En la fase final del proyecto, fue necesario fusionar el trabajo desarrollado con la arquitectura de software existente y los avances que se estaban realizando en paralelo en otros campos del sistema robótico. Para abordar esta tarea, se realizaron algunas modificaciones en la arquitectura diseñada, de forma que se minimizara el esfuerzo requerido para llevar a cabo la integración del software con el sistema.

En primer lugar, se decidió prescindir del nodo de procesamiento de datos, y sus funciones se incluyeron en el nodo encargado de la detección y localización. Los nuevos algoritmos encargados de identificar las berenjenas y proporcionar los datos de entrada al sistema de planificación de tareas se ejecutan en ROS. De esta forma, se prescinde por completo del uso de MATLAB y, como se había previsto al comienzo del trabajo, el uso del conversor deja de ser necesario.

Las funciones de detección se realizan ahora en dos nodos distintos. El primero de ellos se encarga de ejecutar una red neuronal con arquitectura YOLO (estas siglas hacen referencia al algoritmo *You Only Look Once*). Este tipo de red neuronal se utiliza en la identificación y clasificación de objetos. A través de una única red neuronal convolucional se predicen marcos de contorno

que engloban a los objetos y se estiman probabilidades de clases en una única evaluación [26]. La mayor ventaja de este tipo de red no es solamente su eficiencia, si no su gran velocidad de procesamiento.

El segundo nodo, llamado *detection*, consiste en un algoritmo de procesamiento de datos que a partir de los marcos de contorno generados por la red neuronal, extrae los datos relevantes, como por ejemplo, los píxeles que conforman el contorno de la berenjena. Como se puede apreciar, la función de este nodo es similar al nodo *data_ management*. Aunque se realizan cálculos distintos, ambos procesan la información en crudo que reciben. Por lo tanto, con el objetivo de reducir el número de nodos y la sobrecarga de información en ROS, los dos nodos se han fusionado y las funciones desarrolladas en el nodo de procesamiento de datos se han implementado como una extensión del código del nodo *detection*.

De esta forma, el flujo de información solamente recorre dos nodos antes de llegar al nodo de planificación de tareas. Las imágenes en crudo captadas por las cámaras son procesadas por el primer nodo, donde se ejecuta la red neuronal. A continuación, se publica información sobre los marcos de contorno y el segundo nodo calcula toda la información necesaria para rellenar los campos del mensaje *statsAub_ ext*, que se ha mostrado en el Código 3.2.

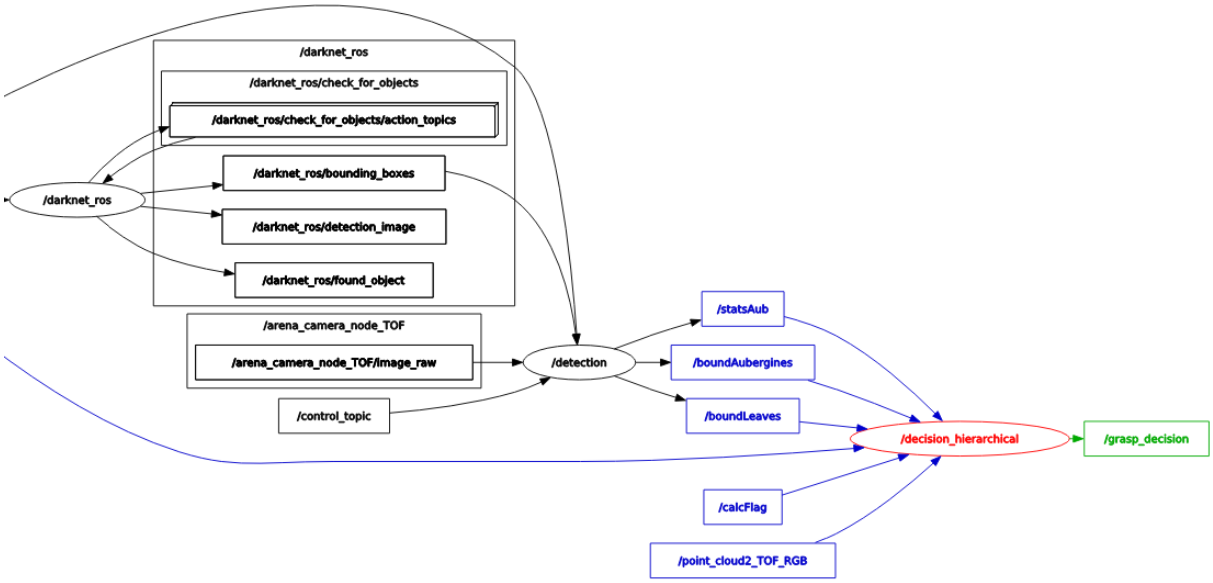


Figura 3.15: Diagrama RQT del módulo de detección y toma de decisiones.

No obstante, para simplificar el paquete, se ha decidido mantener un único mensaje de datos de berenjenas. Así, el mensaje *statsAub_ ext* ha pasado a llamarse *statsAub*, sustituyendo al antiguo mensaje con el mismo nombre.

A modo de resumen, se ha eliminado el nodo *data_ management* de la arquitectura y sus funciones se han migrado al nodo *detection*. En la Figura 3.15 se pueden observar los *topics* utilizados. En azul se muestran los *topics* a los que se subscribe el nodo encargado de la planificación de tareas y, en verde, el *topic* en el que publica los comandos generados. Es preciso destacar que los nombres de los *topics* donde se publican la nube de puntos y la imagen sobre la que se realiza la representación gráfica también han sido modificados. En este caso, para utilizar los *topics* donde se está publicando la información en vivo.

Adicionalmente, también fue necesario adaptar los nodos del tercer módulo, encargado de la

planificación del movimiento, ya que la incorporación de un nodo de planificación de tareas al sistema requiere el diseño de un planificador capaz de interpretar los comandos generados.

Para ello, se ha desarrollado el nodo *IK*. En este nodo se interpretan los 4 tipos de órdenes que han sido considerados en la planificación de tareas y se realiza el cálculo del modelo cinemático inverso. Este modelo genera la configuración de articulaciones que es necesaria aplicar para que el robot alcance una posición determinada.

El cálculo del modelo cinemático inverso se realiza una vez por cada mensaje recibido en el *topic /grasp_decision*. Para ello, se han implementado las funciones correspondientes dentro de la *callback* asociada al subscriptor. Cuando se recibe un mensaje del tipo *decision_msg*, se comprueba el tipo de comando mediante el valor almaceando en el campo *arms* (ver Código 3.1). Dependiendo de este valor, se asignan los centroides de las berenjenas objetivo a un brazo o a otro.

A continuación, es necesario realizar diversas transformaciones entre los sistemas de referencia del robot. El modelo de cinemática inversa debe encontrar una pose del robot en la que su efector final se sitúe en el centroide de la berenjena. No obstante, este dato se encuentra en el sistema de referencia de la base del robot, mientras que el solucionador utiliza la base del brazo como referencia.

Para compensar esta diferencia, se realizan las transformaciones necesarias mediante matrices de rotación a partir de los datos proporcionados por el paquete "tf" de ROS. Este paquete registra la información de cada sistema de coordenadas del robot, y es capaz de calcular cuaternios de orientación o vectores de traslación entre dos sistemas de referencia.

Cuando se obtienen los centroides objetivos en el sistema de referencia apropiado, se itera sobre el solucionador del modelo de la cinématica inversa un número fijo de veces hasta que encuentre una solución viable. En caso de que no se encuentre una solución en el número de iteraciones establecido, se considera la posición como inalcanzable. Sin embargo, esto no debería ocurrir ya que, en nodos anteriores del sistema, se han descartado las berenjenas fuera de rango. El solucionador utilizado ha sido desarrollado por la empresa TRACLABS [27].

Es preciso destacar, que el punto utilizado como objetivo no corresponde exactamente con el centroide calculado en módulos anteriores. Cuando se realizan las transformaciones entre sistemas de referencia también se aplica un desplazamiento horizontal al centroide, para que el brazo se aproxime al punto de recogida, pero sin llegar alcanzarlo. Este punto se denomina *pre-grasp*. El objetivo detrás de este procedimiento es realizar un movimiento rápido y con menores restricciones hasta el *pre-grasp* y, una vez alcanzado, realizar un movimiento cartesiano más lento hasta el punto de recogida. Sin embargo, la configuración de este sistema pertenece al diseño del módulo de planificador de movimientos y no entra dentro del alcance de este Trabajo Fin de Máster, por lo que no se explica detalladamente.

Una vez se ha resuelto el modelo de cinemática inversa para uno o los dos brazos, dependiendo de la orden recibida, se publican las posiciones de las articulaciones para que otro nodo las ejecute y realice la recogida de la berenjena.

De esta forma, mediante el nodo *decision* y el nodo *IK*, el sistema de planificación de tareas desarrollado queda completamente integrado en la arquitectura de software. Es preciso destacar que cualquiera de los tres nodos de toma de decisiones desarrollados puede utilizarse en esta situación, produciendo resultados distintos dependiendo de la escena. En este contexto, se pueden observar los beneficios de una arquitectura modular y del sistema de herencia desarrollado. Además, tal y como se explica en la Sección 3.6, la elaboración de un nuevo nodo para la planificación

de tareas únicamente requiere la creación de una nueva clase que herede de *Decision* y la implementación del algoritmo de ordenación y clasificación que quiera utilizarse. Esta característica dota al sistema de una gran adaptabilidad y escalabilidad para desarrollos futuros.

3.8. Aprendizaje por refuerzo

La planificación de tareas desarrollada a lo largo de este trabajo es capaz de tomar decisiones en función del entorno que perciben los sensores del robot. Cada uno de los tres nodos desarrollados utiliza distintos parámetros para realizar la toma de decisiones. Estos parámetros han sido seleccionados a partir de los razonamientos lógicos que seguiría un ser humano si se enfrentara al mismo problema. Sin embargo, es fácil deducir que la naturaleza del problema a resolver no sigue unas reglas fijas y, por lo tanto, no se conoce a priori la decisión o solución óptima.

Si se realizara un experimento con distintas personas en el que se les pidiera que recogieran frutas, es muy probable que los resultados mostraran que cada persona ha seguido un patrón de recogida distinto. A diferencia de otros problemas típicos dentro del paradigma de la toma de decisiones inteligentes, la recolección de frutas no está sujeto a reglas que determinen cuándo una acción está bien o mal. No se trata de un juego o situación con reglas fijas, en el que se gana, se pierde o se empata. Incluso analizando cada situación detenidamente, es complicado determinar cuando una acción se puede catalogar como "buena" o "mala". Claramente, se pueden buscar parámetros que ayuden a definir cada acción, como el tiempo de recolección, el número total de unidades recogidas o el estado de cada fruta tras su manipulación, pero aun así, sería complicado determinar el mejor sistema de recolección, ya que habría que evaluar que parámetro tiene más importancia frente al resto.

Se trata de un problema complejo sin una solución óptima clara. El orden de recolección seguido por un humano se basa en su experiencia o intuición, pero nunca en un análisis matemático del que se extraigan variables que dicten decisiones claras. Este problema, incita a implementar no solamente algoritmos de inteligencia artificial simples como los que se han desarrollado durante este Trabajo Fin de Máster, sino programas basados en otras técnicas que permitan un aprendizaje basado en experiencias previas.

Tal y como se ha explicado en la Sección 2.1 en el Estado del Arte, el aprendizaje por refuerzo es el problema que mejor aborda esta situación. El sistema robótico estudiado consiste en un robot capaz de percibir e interactuar con su entorno y que debe realizar una serie de acciones para conseguir su objetivo final: recolectar todas las berenjenas. La implementación de un sistema de aprendizaje por refuerzo es, por tanto, la línea de trabajo más lógica.

La consecución de esta tarea, no obstante, es un problema complejo y cuya realización no entra dentro de los límites de tiempo establecidos para la ejecución de este Trabajo Fin de Máster. Sin embargo, con el objetivo de facilitar una futura implementación, se han realizado los primeros pasos hacia el desarrollo de un sistema de aprendizaje por refuerzo.

Existen una gran cantidad de configuraciones posibles y variables que es necesario analizar para implementar de forma efectiva algoritmos de aprendizaje por refuerzo. No obstante, a parte del agente que realiza la acción (el robot) y el entorno (el espacio físico donde se encuentran las berenjenas), se pueden identificar cuatro elementos comunes a todos los problemas: una política, una señal de recompensa, una función de evaluación y un modelo del entorno [28].

La definición de cada uno de estos elementos determinará el comportamiento del software implementado y, a largo plazo, su rendimiento y resultados. Aunque no entra dentro del alcance de

este trabajo explicar estos conceptos debido a su profundidad, es preciso destacar que casi todo el desarrollo realizado puede ser utilizado para la implementación de un sistema de estas características. Los tres nodos desarrollados, se pueden considerar políticas básicas de un sistema de aprendizaje por refuerzo, ya que determinan distintas formas de actuar ante el mismo entorno. El desarrollo completo de una política debe extenderse y considerar otros parámetros en una implementación real, pero ya se ha desarrollado una base sobre la que empezar a trabajar. Aquí surgen los conceptos de exploración y explotación, que determinan el grado en el que el robot utiliza acciones que ya sabe que son beneficiosas o explora acciones nuevas para comprobar si producen más o menos beneficio. La optimalidad de cada estrategia, deberá ser valorada por las señales de recompensa y las funciones de evaluación.

La declaración de estos dos conceptos, afectará directamente al tipo de planificador que se entrene. Existe mucha teoría y el estado del arte relacionado con estos dos elementos es extenso, ya que se pueden considerar los componentes centrales de todo sistema de aprendizaje por refuerzo. La determinación de la señal de recompensa se puede implementar a partir del éxito de la recolección, o analizando el estado de cada fruta recogida. La función de evaluación, por otro lado, deberá tener en consideración otros parámetros que le permitan evaluar el desempeño global del robot, para comprobar si ha cumplido su objetivo o no. En este caso, algunas de las variables a considerar más intuitivas son las mencionadas anteriormente, como el tiempo total o la efectividad de la recolección.

Por último, el modelo del entorno ha sido implementado al realizar la integración del sistema de decisiones con el resto de la arquitectura de software. Al disponer de un sistema funcional, se dispone de un modelo que permite iterar distintos experimentos para conseguir entrenar al robot. La iteración para el aprendizaje se podría realizar sobre el robot físico, pero serían experimentos demasiado costosos y lentos. Es mucho más práctico e interesante utilizar la simulación virtual que se ha utilizado durante la integración completa del sistema. Esta simulación se describe más detalladamente cuando se explican las pruebas realizadas (Sección 4.3).

En definitiva, aunque la implementación de un sistema completo de aprendizaje por refuerzo y su validación está todavía lejos de alcanzarse, a lo largo de este trabajo se han realizado los primeros pasos necesarios para disponer de un modelo capacitado sobre el que desarrollar los algoritmos de *Machine Learning*.

4. PRUEBAS Y RESULTADOS

El siguiente capítulo realiza una recopilación de las pruebas más relevantes del proyecto. Se pueden diferenciar dos secuencias de pruebas distintas. La primera incluye las pruebas realizadas a la arquitectura de forma aislada, sin integrar el software con el resto de la arquitectura del sistema robótico. La segunda secuencia es la realizada una vez integrado el trabajo desarrollado con el resto de componentes de la arquitectura de software. Durante estas pruebas se han tenido en cuenta los cambios descritos en la Sección 3.7

4.1. Pruebas del nodo *decision_hier*

Para las primeras pruebas de funcionamiento se utilizaron los datos de dos escenas distintas. La primera de ellas disponía de 5 berenjenas y la segunda una única berenjena que estaba parcialmente ocluida. Ambas escenas se encontraban almacenadas como estructuras de datos de MATLAB.

El primer paso, por tanto, consiste en utilizar el conversor creado en MATLAB para convertir los datos a formatos compatibles con ROS. Además, estos datos fueron grabados en *bags* para no tener que ejecutar el conversor cada vez que se necesitara realizar una prueba.

```
carlos@ubuntu:~/catkin_ws/src/tfm_eggplants/rosbags$ rosbag info bag1_5aubs.bag
path:        bag1_5aubs.bag
version:     2.0
duration:    53.0s
start:       Feb 14 2023 10:13:48.13 (1676398428.13)
end:         Feb 14 2023 10:14:41.18 (1676398481.18)
size:        8.0 MB
messages:    294
compression: none [3/3 chunks]
types:       rosgraph_msgs/Log         [acffd30cd6b6de30f120938c17c593fb]
             sensor_msgs/Image         [060021388200f6f0f447d0fcd9c64743]
             std_msgs/UInt16MultiArray [52f264f1c973c4b73790d384c6cb4484]
             tfm_eggplants/boundAub    [35aabfa16de8859359b7b3d7552858e6]
             tfm_eggplants/statsAub    [65dfd836f8f3a9c3c7128808405b16dc]
topics:      /matlab_boundAubergines     5 msgs    : tfm_eggplants/boundAub
             /matlab_boundLeaves       264 msgs    : tfm_eggplants/boundAub
             /matlab_img                 1 msg     : sensor_msgs/Image
             /matlab_imgDim              1 msg     : std_msgs/UInt16MultiArray
             /matlab_imgRGB              1 msg     : sensor_msgs/Image
             /matlab_statsAub            5 msgs    : tfm_eggplants/statsAub
             /rosout                     9 msgs    : rosgraph_msgs/Log
             /rosout_agg                 8 msgs    : rosgraph_msgs/Log
```

Figura 4.1: *Bag* creado con los datos publicados por el conversor.

En la Figura 4.1 se muestra un ejemplo de los datos registrados en el archivo ".bag". Se puede observar como se crean de forma correcta todos los *topics* de interés que se describen en la Tabla 3.2.

4.1.1. Escena con 5 berenjenas

Una vez se dispone de los datos en el entorno de trabajo adecuado, es suficiente con ejecutar los nodos correspondientes de ROS mediante los comandos indicados. En una terminal es necesario ejecutar el "ROS Master" y dejarlo en segundo plano. A continuación, se lanzan los nodos mediante el comando mostrado en el Código 4.1. Es necesario iniciar el nodo *data_management* en una terminal y el nodo que se quiera probar en otra. Por último, hay que reproducir el archivo ".bag" en una nueva terminal para introducir los datos que han sido almacenados en el entorno de trabajo.

```
$ rosrun <nombre_paquete> <nombre_nodo>
```

Código 4.1: Comando para iniciar un nodo en ROS.

A medida que los datos del *bag* se vayan reproduciendo, las terminales donde se ejecutan los nodos mostrarán mensajes informativos sobre los datos que reciben. Estos mensajes no son necesarios para el funcionamiento del programa, pero ayudan a conocer el estado actual del nodo y a reconocer visualmente el flujo de información. Son especialmente útiles como herramienta de *debugging* cuando se está desarrollando el código.

Cuando se termina de reproducir el *bag*, todos los datos han sido almacenados en los atributos de clase de cada nodo. Estos nodos permanecen en este estado de forma indefinida, hasta que se detienen manualmente o hasta que se ejecuta el comando encargado de activar la toma de decisiones. Este comando consiste en la publicación de un mensaje del tipo *std_msgs/Bool* con valor "Verdadero". Se puede ver un ejemplo en la Figura 4.2.

```
$ rostopic pub /calcFlag std_msgs/Bool "data: True"
```

Código 4.2: Comando para activar la toma de decisiones.

Cuando el subscriptor del nodo de la toma de decisiones recibe el mensaje publicado, llama a las funciones para la planificación de tareas a alto nivel desde su función *callback* asociada. Aquí se ejecutan los algoritmos detallados en la Metodología (sección 3.6). Un ejemplo de los mensajes informativos del nodo de toma de decisiones se puede observar en las Figuras 4.2 y 4.3. En la primera se muestra el estado de las terminales antes de publicar el mensaje en el *topic /calc_Flag*, y en la segunda el estado después del mensaje. En estas pantallas se incluyen el orden de los vectores utilizados para clasificar las berenjenas, las características de cada intersección en el caso de que exista una berenjena ocluida y el tipo de mensaje que ha sido publicado con el comando que corresponda. Todos los mensajes son configurables y pueden ser eliminados en caso de que no se requiera mostrar la información por pantalla.

También se incluye un parámetro adicional denominado "Center Aside" para cada berenjena ocluida. Este parámetro, compuesto por dos valores, se refiere al *offset* que hay aplicar a cada coordenada del centroide de la berenjena para ajustar la desviación que existe en las berenjenas ocluidas. Esta corrección es necesaria ya que el centroide proporcionado por el software del módulo de detección y localización se calcula a partir del área visible de la berenjena ocluida. Por lo tanto, estas coordenadas no hacen referencia al centroide real de la berenjena, sino al centroide del área visible. El dato "Center Aside" proporciona, en el sistema de coordenadas de la base del robot, la corrección necesaria para aproximar el centroide de una manera más precisa, teniendo en cuenta la desviación producida por la oclusión.

El proceso se basa en aplicar unas pequeñas desviaciones al centroide original en función del tamaño de la oclusión y en la dirección proporcionada por el vector utilizado para apartar la hoja. Cuando la oclusión de una berenjena es muy grande, es decir, el área de intersección entre una hoja y la berenjena es elevada, se considera que el centroide se encuentra muy desplazado y, por tanto, se aplica una corrección mayor que en un caso donde la oclusión fuera más pequeña.

En la terminal del nodo de toma de decisiones se muestra un mensaje por cada comando publicado. En este ejemplo en concreto, los mensajes se pueden observar en la terminal de la esquina inferior derecha de la Figura 4.3. Las cuatro últimas líneas indican la publicación de dos mensajes correspondientes a recolecciones simultáneas y un mensaje de recolección de una berenjena

ocluida, que es exactamente lo requerido por la escena, ya que hay 2 berenjenas asignadas a cada brazo y una ocluida. El último mensaje indica que una vez utilizados estos datos para realizar la toma de decisiones, se eliminan de los atributos de la clase, para no ser reutilizados de nuevo más adelante.

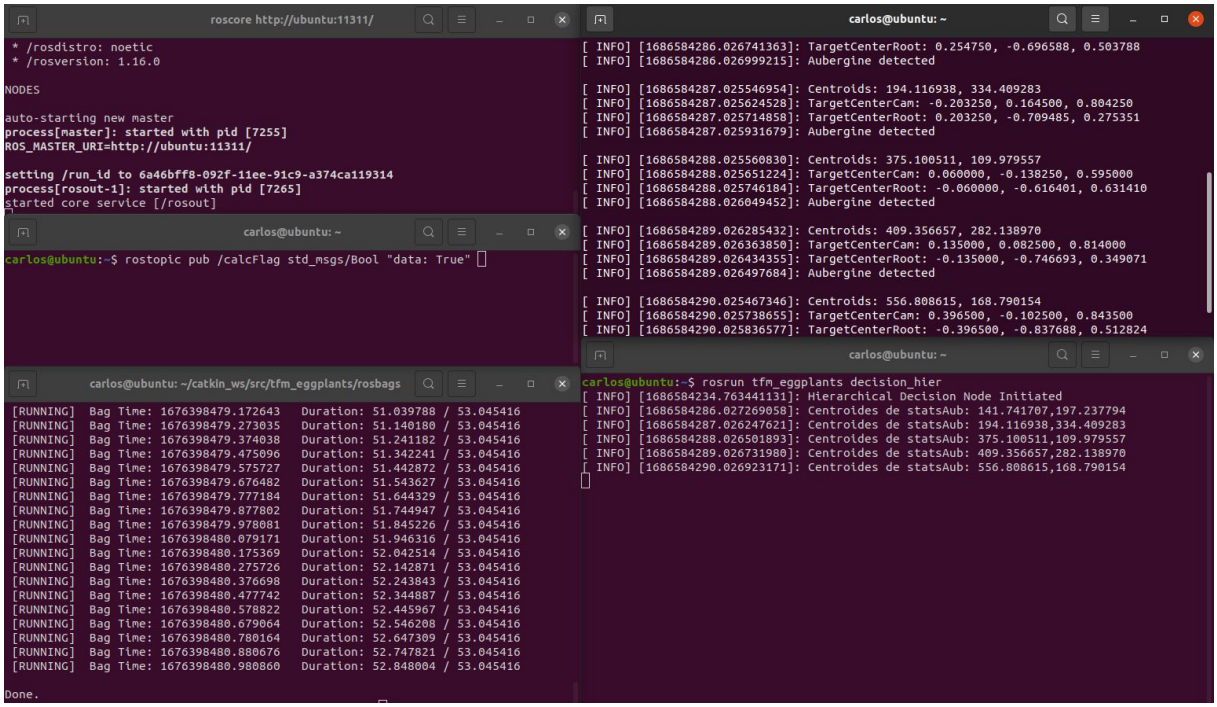


Figura 4.2: Vista de las terminales antes de iniciar la toma de decisiones.

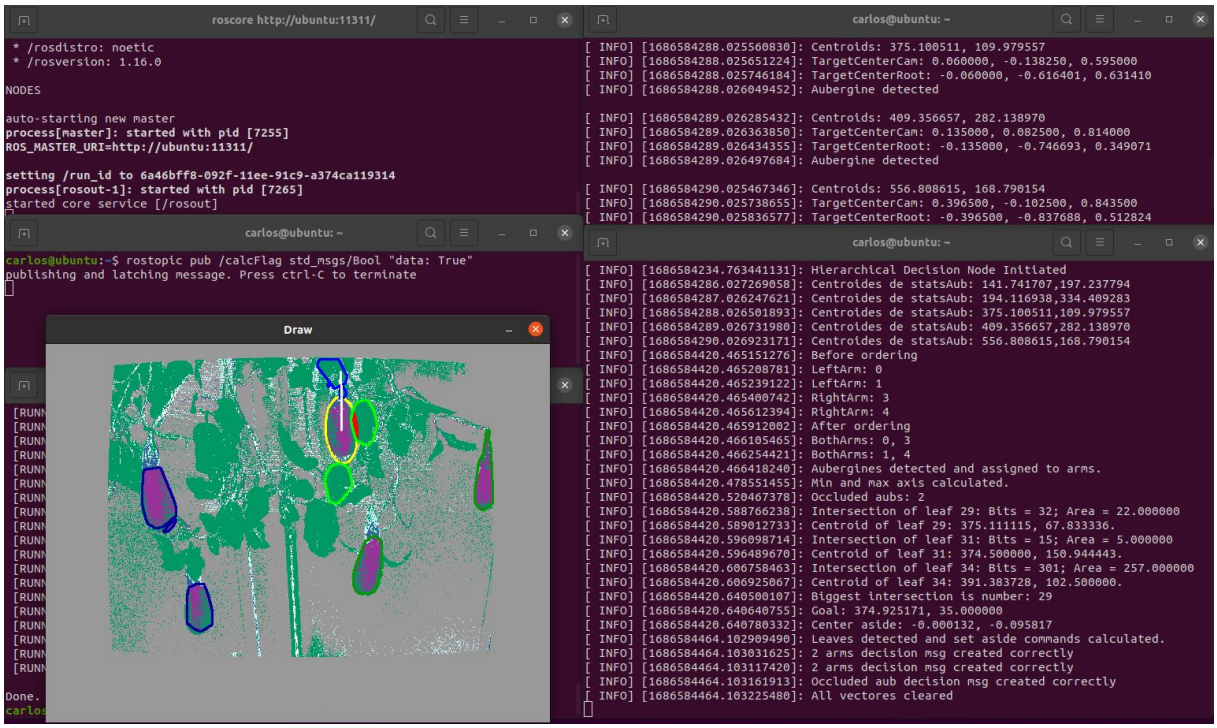


Figura 4.3: Vista de las terminales después de iniciar la toma de decisiones.

De forma adicional, en estas pruebas se ha habilitado la representación gráfica de los algoritmos,

aunque ésta no se encuentra en la versión final del código. En la Figura 4.4 se puede observar la escena después de aplicar los algoritmos de toma de decisiones. Las berenjenas asociadas al brazo izquierdo del robot se han marcado en azul oscuro, mientras que las berenjenas asociadas al brazo derecho están marcadas en verde oscuro. También se muestran las hojas que intersectan con la elipse auxiliar de la berenjena ocluida, y el vector calculado que indica la trayectoria utilizada para apartar la hoja.

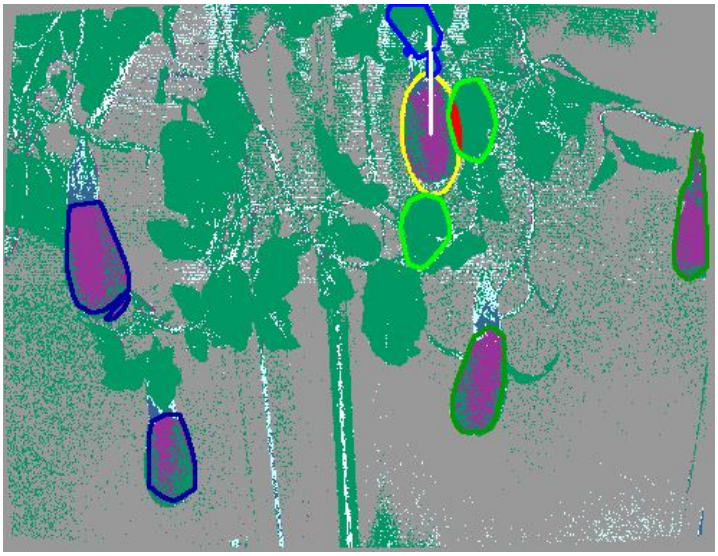


Figura 4.4: Representación gráfica del algoritmo de toma de decisiones del nodo *decision_hier*.

Es necesario mencionar que en la prueba realizada con estos datos se detecta la berenjena superior como ocluida, cuando realmente no presenta ninguna oclusión relevante. El motivo de esta clasificación, errónea a primera vista, se debe al área calculada por el algoritmo del módulo de detección y localización. Si se representa el contorno proporcionado por dicho algoritmo, se puede observar que no se captura todo el área de la berenjena (Figura 4.5). Esto provoca que se utilicen datos imprecisos desde un principio, y por tanto el algoritmo de clasificación asuma que se trata de una berenjena ocluida al considerar un área más pequeña que la que corresponde.

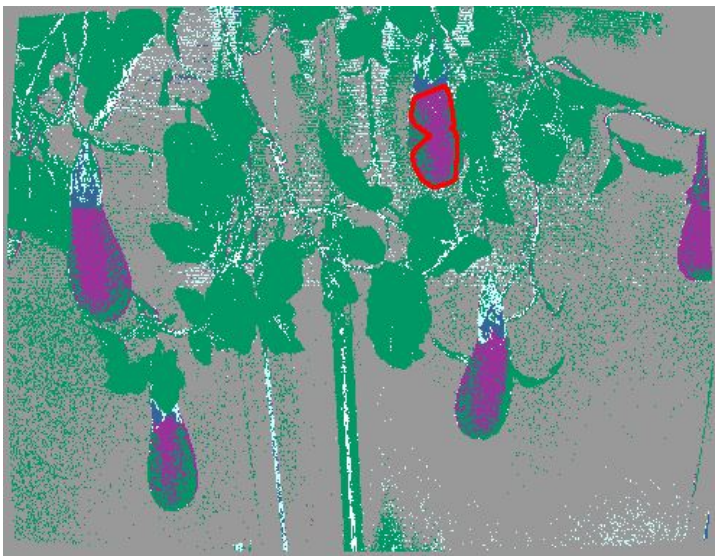


Figura 4.5: Contorno impreciso de la berenjena “ocluida”.

4.1.2. Escena con una berenjena ocluida

La segunda escena que se utilizó contenía una única berenjena ocluida. El procedimiento necesario para poner en funcionamiento los nodos es siempre el mismo, independientemente de los datos que se utilicen. El único cambio que es necesario realizar es la selección de los datos que se quieran utilizar, es decir, seleccionar el archivo ".bag" con el que se quiera experimentar, junto a la nube de puntos almacenada en el archivo ".pcd" correspondiente.

En este segundo caso, tras clasificar la única berenjena de la escena correctamente como ocluida, se inicia el algoritmo de oclusiones directamente, ya que no existen más berenjenas. Esto se muestra en los mensajes de la terminal, ya que después de los mensajes "Before Ordering" y "After Ordering" no aparece ningún dato. Hay que destacar que el mensaje "Occluded aubs: 0" no indica que no haya berenjenas ocluidas, si no que hace referencia al índice de la berenjena ocluida. En este caso, se trata del único elemento del vector, por lo que al tratarse de un código en C++, el índice correspondiente es el cero.

En la Figura 4.6 se pueden observar las terminales tras publicar el mensaje que da comienzo a la toma de decisiones y que se detecte la berenjena ocluida correctamente. Las intersecciones se calculan de manera apropiada y el vector de trayectoria para apartar la hoja muestra una dirección adecuada. Esta escena es la misma que se ha utilizado en la Figura 3.14 para explicar los pasos realizados en el algoritmo de berenjenas ocluidas.

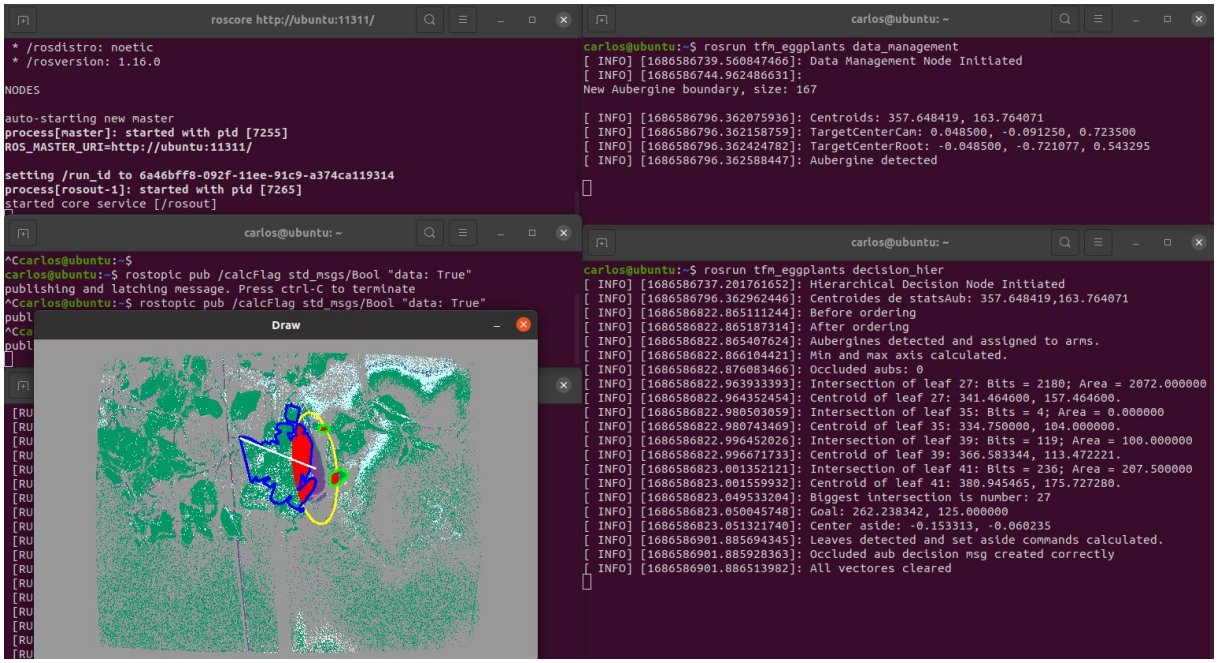


Figura 4.6: Vista de las terminales después de iniciar la toma de decisiones.

Como se ha podido observar, el proceso para utilizar el programa requiere la apertura de muchas terminales simultáneas debido al comportamiento intrínseco de ROS. Esto convierte el código de toma de decisiones en un programa tedioso, a pesar de incluir únicamente dos nodos. Para solucionar este inconveniente, existen los archivos con extensión ".launch".

Los archivos *launch* son ficheros en XML que se utilizan para ejecutar múltiples nodos o herramientas de ROS mediante un único comando. Se utilizan para agrupar la ejecución de nodos y, de esta forma, simplificar el arranque de los programas. Son especialmente útiles para utili-

zar paquetes externos, que requieren del uso de múltiples nodos, como por ejemplo el paquete *MoveIt!*.

Estos ficheros tienen multitud de opciones, como la declaración de parámetros. Los parámetros son una herramienta que tiene ROS para variar el funcionamiento de los nodos sin modificar su código fuente. Por ejemplo, se puede programar un nodo para que lea la ruta a un fichero desde un parámetro, en vez de utilizar una cadena de caracteres constante en el código. De esta forma, si se quisiera modificar la ruta, sería suficiente con modificar el parámetro desde una terminal al lanzar el nodo o, modificar el parámetro desde el archivo ".launch".

En cualquiera de los dos casos, se consigue producir un cambio en el código y el comportamiento del programa sin la necesidad de compilar el paquete de nuevo. Cuando se trabaja con pocos paquetes el proceso de compilación puede resultar rápido, pero a medida que aumenta el número de paquetes en el espacio de trabajo el tiempo de compilación crece considerablemente. Esta característica convierte a este tipo de archivos en opciones mucho más rápidas a la hora de ajustar variables de los programas o realizar distintas pruebas. Asimismo, al ejecutar un archivo ".launch" no es necesario inicializar el "ROS Master" en una terminal paralela, lo que simplifica todavía más el proceso de ejecución de programas.

Para este trabajo, se ha desarrollado un archivo de estas características que ejecuta los dos nodos necesarios para el funcionamiento del programa. También declara la ruta al *bag* utilizado como un parámetro y, a continuación, inicia la reproducción del archivo indicado. Además, se ha configurado el archivo para que el nodo de toma de decisiones se inicie en una terminal nueva de forma automática. Así, se puede ejecutar todo el programa descrito anteriormente mediante el comando en terminal que se muestra a continuación (4.3.

```
$ roslaunch <nombre_paquete> <nombre_launch_file>
```

Código 4.3: Comando para ejecutar un archivo ".launch" en ROS.

En el Código 4.4 se muestra el archivo *launch* utilizado para ejecutar el programa mediante un único comando.

```
<launch>

  <!-- data management node -->
  <node pkg="tfm_eggplants" type="data_management" name="data_management"/>

  <!-- decision node -->
  <node pkg="tfm_eggplants" type="decision_depth" name="decision_depth"
        ↪ output="screen" launch-prefix="gnome-terminal --command"/>

  <!-- Rosbag play -->
  <arg name="path" default="$(find tfm_eggplants)/rosbags/"/>
  <arg name="bag" default="bag1_5aubs"/>
  <node pkg="rosbag" type="play" name="player" output="screen" args="--clock
        ↪ $(arg path)$(arg bag).bag"/>

</launch>
```

Código 4.4: Comando para ejecutar un archivo ".launch" en ROS.

4.2. Pruebas del nodo *decision_depth* y *decision_distance*

Una vez comprobado el correcto funcionamiento del nodo más básico (*decision_hier*), se puede proceder a realizar pruebas con el resto de nodos.

Conviene recordar que los nodos se han implementado mediante un sistema de herencia en C++, y que las funciones básicas de ROS del nodo, junto al algoritmo para berenjenas ocluidas, se han implementado en la clase madre de la que heredan todos los nodos. De esta forma, no hace falta repetir las pruebas de funcionamiento básicas con los nuevos nodos, ya que todos comparten el mismo código fuente. Únicamente es necesario verificar que las funciones nuevas que implementan las clases derivadas funcionan como cabría esperar.

Las pruebas se han realizado con la misma escena de 5 berenjenas que se ha utilizado con el nodo anterior. De esta forma, se pueden visualizar los cambios que produce cada nodo.

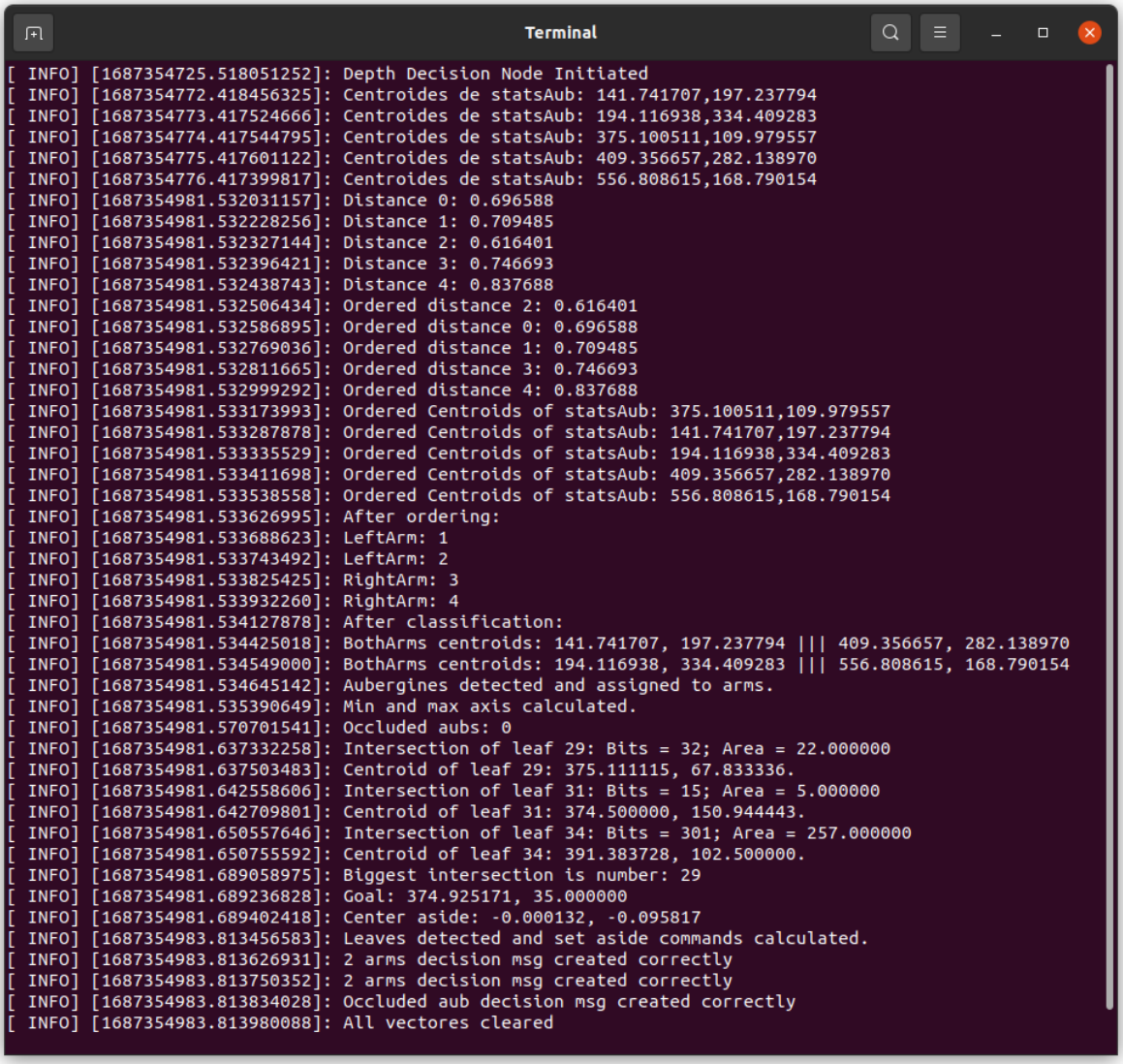


Figura 4.7: Vista de la terminal después de iniciar la toma de decisiones con el nodo *decision_depth*.

En la Figura 4.7 se pueden observar los mensajes informativos de ROS que detallan el el proce-

dimiento seguido por el nodo *decision_ depth*. Inicialmente se muestran los centroides en píxeles de cada berenjena, en el orden recibido desde el nodo anterior. A continuación, se muestra la profundidad de cada berenjena, es decir, la distancia representada en el eje Y de la base del robot. Seguidamente, se ordenan de forma creciente las distancias (o profundidades) y el vector de berenjenas asociado. Finalmente, se muestran los vectores de clasificación después de ordenar y de aplicar la clasificación. En este caso, el vector *bothArms* tiene dos elementos, que representan las 4 berenjenas.

Después del mensaje "After classification" se muestran los centroides de las berenjenas a las que hace referencia el vector *bothArms*, en vez de los índices almacenados en el vector. Se ha decidido mostrarlo de esta manera para saber qué berenjena de la escena se está indicando, con el objetivo de simplificar la compresión del proceso. Hay que recordar que este nodo reordernaba el atributo de clase *aubergines*, donde están contenidos los datos de las berenjenas, de forma que aunque los vectores de clasificación contengan índices secuenciales, acceden a berenjenas ordenadas por su profundidad. El algoritmo de oclusiones que se ejecuta a continuación es el mismo que el descrito con el nodo *decision_ hier*.

Se puede observar en la Figura 4.8 que la asignación de berenjenas a cada brazo es idéntica al anterior nodo (Figura 4.4). Este comportamiento es el adecuado, ya que estos nodos únicamente alteran el orden de recogida de las berenjenas, pero no deben modificar nunca la clasificación que se realiza.

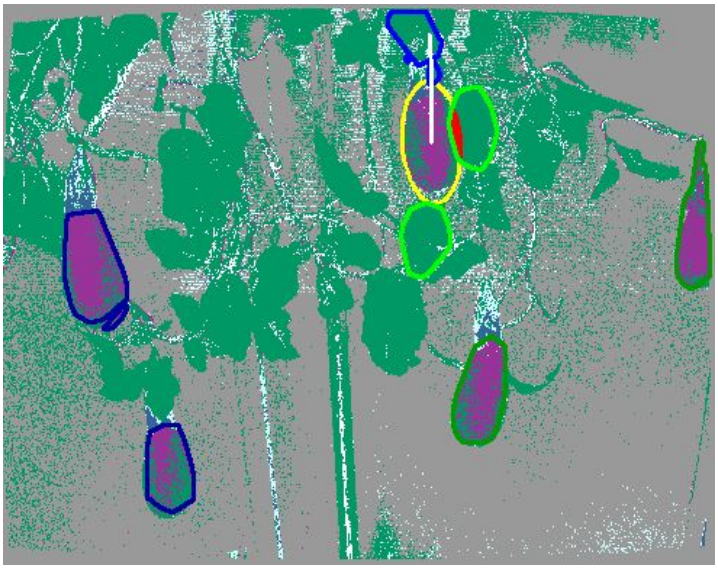


Figura 4.8: Representación gráfica del algoritmo de toma de decisiones del nodo *decision_ depth*.

El nodo *decision_ distance*, por otro lado, produce la clasificación mostrada en la terminal de la Figura 4.9. El nodo clasifica las berenjenas en los sectores izquierdo y derecho y, a continuación, en cada sector, calcula la distancia de cada berenjena a la posición inicial del efector final del robot. Es decir, se calculan las distancias euclidianas entre las berenjenas del sector izquierdo y el efector final del brazo izquierdo por un lado, y las distancias entre las berenjenas del sector derecho y el efector final del brazo derecho por otro. Una vez se ha realizado el cálculo, se ordena el vector de clasificación de cada sector en función de la distancia.

En la terminal se muestran primero las coordenadas en el sistema de referencia del robot, para comprobar en esta primera fase de *debugging* que se están realizando los cálculos correctamente. Después, se muestran las distancias calculadas y cómo queda el vector tras ordenarlo de forma

creciente. En este caso, el vector *leftArm* cambia de orden mientras que el vector *rightArm* no. Por último, se muestran los vectores de clasificación tras asignar la recolección simultánea y el flujo del programa continúa de la misma forma que en los anteriores ejemplos.

Como en esta escena solamente hay dos berenjenas por sector, el cálculo realizado por el nodo es muy sencillo. Simplemente se ordenan las dos berenjenas en función de la distancia más corta al punto de reposo de los efectores finales de los brazos. Si hubiera más berenjenas en alguno de los sectores, el algoritmo calcularía las distancia entre el primer objetivo y el resto de berenjenas, y ordenaría nuevamente el vector.

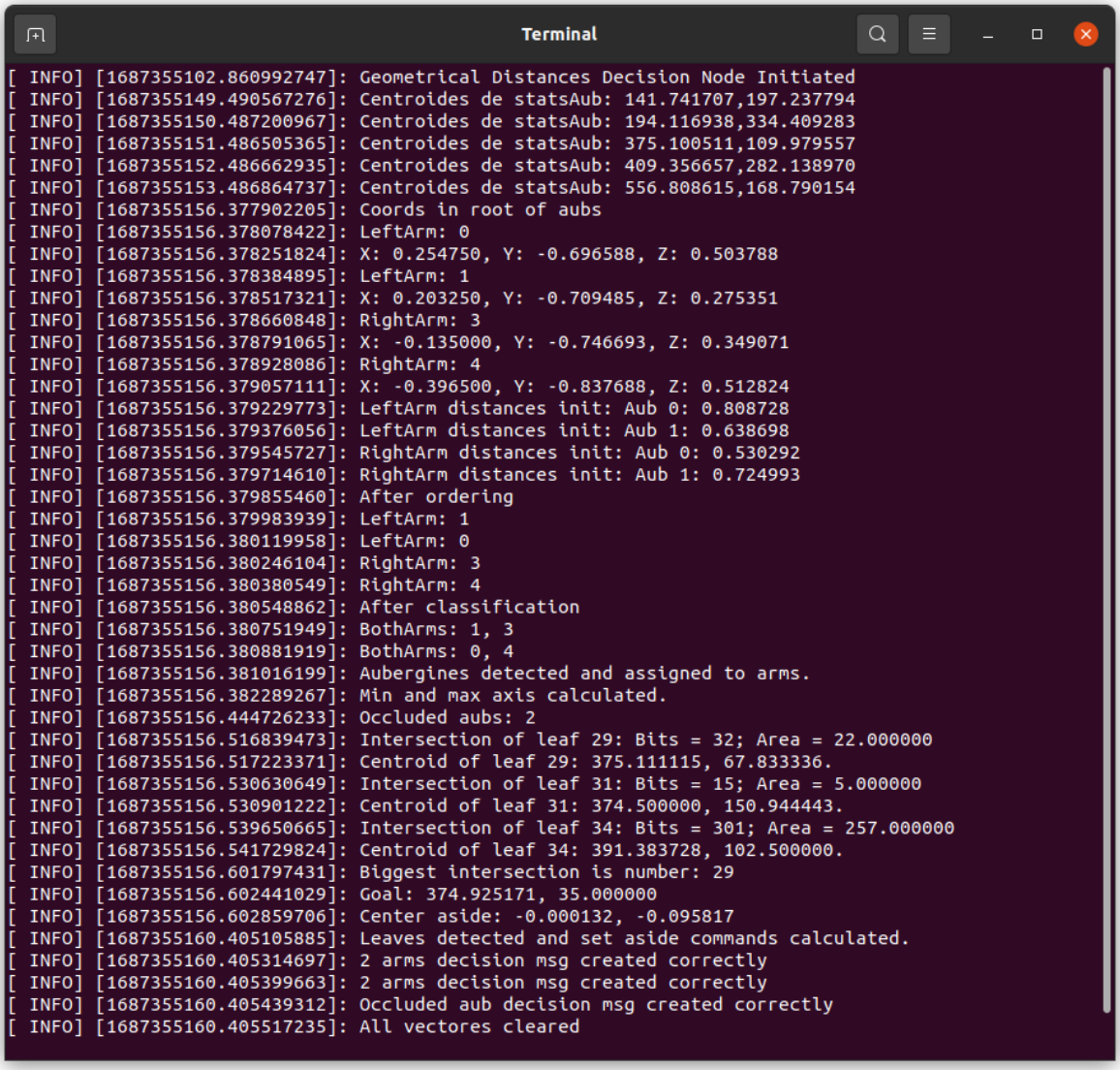


Figura 4.9: Vista de la terminal después de iniciar la toma de decisiones.

El punto de reposo de los efectores finales y, por tanto, la pose inicial de los brazos robóticos, se puede obtener del archivo ".urdf" del robot. Este archivo se utiliza para la simulación virtual del sistema, y consiste en un archivo que describe al robot físicamente. De hecho, la extensión del archivo viene de las siglas URDF en inglés, *Unified Robot Description Format*. El archivo incluye las características geométricas de los brazos y la posición y orientación del sistema de coordenadas asociado a cada eslabón del robot. Se denomina eslabón a cada cuerpo rígido del robot conectado a otros elementos mediante articulaciones. El archivo ".urdf" incluye la posición y orientación de

un sistema de referencia por cada articulación. Además, cada sistema se posiciona en función del anterior, formando así, una cadena.

ROS incluye un paquete dedicado a la gestión de estos sistemas de referencia, que recibe el nombre de "tf". Este paquete realiza las transformaciones entre distintos sistemas de coordenadas de forma automática, y es realmente útil cuando se utilizan robots manipuladores con múltiples eslabones, como en este proyecto. Por lo tanto, para conocer la posición inicial de los efectores finales respecto al sistema de referencia de la base del robot es suficiente con utilizar el comando mostrado en las Figuras 4.10 y 4.11.

```
carlos@ubuntu:~$ rosrun tf tf_echo root left_end_effector
At time 1686667937.004
- Translation: [0.190, -0.248, -0.166]
- Rotation: in Quaternion [0.111, -0.837, -0.063, 0.532]
            in RPY (radian) [2.660, -1.068, -2.591]
            in RPY (degree) [152.408, -61.207, -148.438]
At time 1686667937.605
```

Figura 4.10: Traslación y rotación del efector final izquierdo respecto a la base del robot.

```
carlos@ubuntu:~$ rosrun tf tf_echo root right_end_effector
At time 1686668041.004
- Translation: [-0.189, -0.249, -0.166]
- Rotation: in Quaternion [0.831, -0.149, 0.529, -0.086]
            in RPY (radian) [-2.530, -1.023, -0.704]
            in RPY (degree) [-144.947, -58.598, -40.357]
At time 1686668041.605
```

Figura 4.11: Traslación y rotación del efector final derecho respecto a la base del robot.

4.3. Integración con los trabajos paralelos de la línea de investigación

Tras realizar la integración del módulo de planificación de tareas en la arquitectura de software del sistema, se pudieron realizar algunas simulaciones completas de funcionamiento.

Las simulaciones se realizaron a través de la herramienta gráfica "Rviz". Gracias a los archivos URDF que ya han sido mencionados en el epígrafe anterior, se puede construir un modelo virtual de los brazos robóticos sobre los que comprobar el funcionamiento del programa. Además, se puede representar la nube de puntos captada por la cámara y situarla en la posición exacta en la que se encuentra en la realidad. De esta forma, se consigue un gemelo digital del modelo físico.

En la Figura 4.12 se puede observar el gemelo digital del sistema. En el modelo se representa la jardinera donde se encuentran las berenjenas, junto a la nube de puntos captada por la cámara de tiempo de vuelo combinada con la información de color proporcionada por la cámara RGB. Los brazos robóticos del robot se muestran en un color claro semi-transparente, y están acoplados a la estructura fija donde están instaladas las dos cámaras. Esta estructura está, a su vez, instalada encima de un robot móvil RB-Vogui.

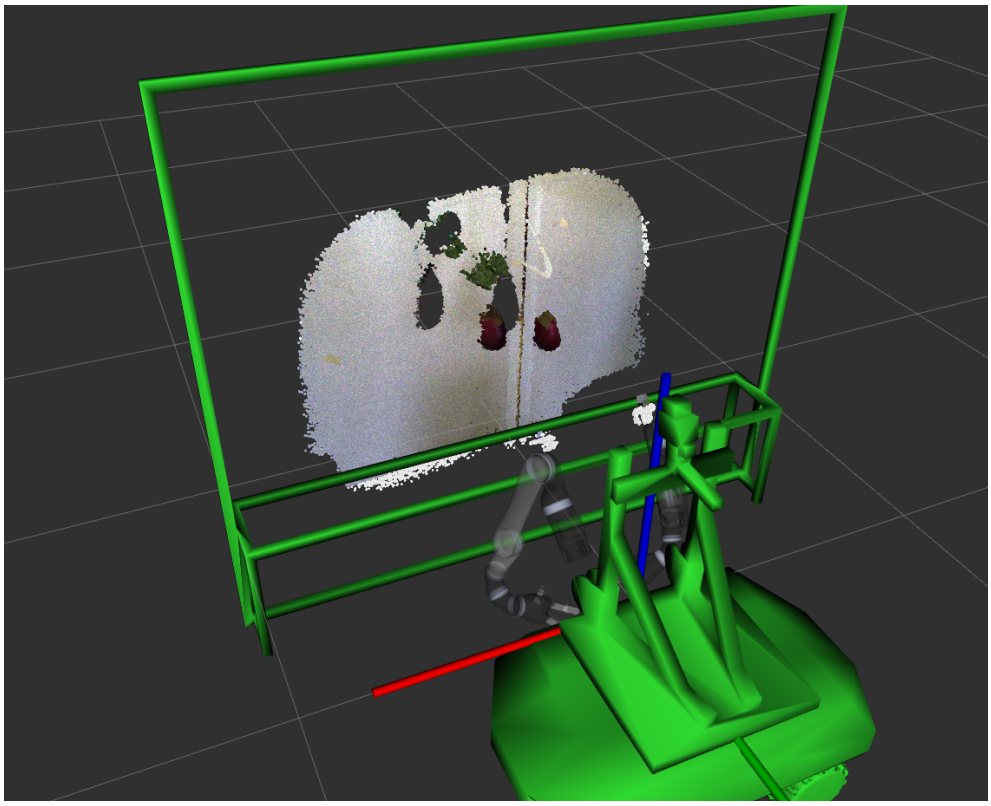


Figura 4.12: Simulación del robot utilizada durante las pruebas.

Una vez se dispone del modelo en la simulación, las pruebas consisten en ejecutar todos los nodos del sistema y comprobar su correcto funcionamiento. A continuación, se va a explicar el desarrollo de una prueba con dos berenjenas.

En primer lugar, se deben ejecutar los dos nodos del módulo de detección y localización. Cuando se ha procesado toda la información de la escena, se publican en los *topics* */statsAubs*, */boundAubergines* y */boundLeaves* los datos de las berenjenas y los contornos de las hojas, tal y como se ha explicado en la Sección 3.5.

En este momento, entra en funcionamiento el nodo de planificación de tareas. Dependiendo de la política de recolección que se quiera utilizar, se debe ejecutar un nodo u otro. En el ejemplo explicado, ya que únicamente se utilizan dos berenjenas, no se pueden apreciar grandes diferencias entre los nodos. Como cada berenjena se encuentra en un lado de la escena, el robot puede alcanzar las dos berenjenas simultáneamente, mediante una acción del tipo "2". Esta acción se procesa en el nodo y se publica mediante un mensaje del tipo *decision_msg* en */grasp_decision*.

El nodo *IK* recibe el mensaje publicado y decodifica la información. En este caso, asigna el primer centroide al brazo izquierdo y el segundo centroide al brazo derecho, ya que se trata de una acción simultánea. Seguidamente, se itera sobre el modelo de cinemática inversa para hallar una pose en la que se alcance el punto de *pre-grasp*. Una vez se ha alcanzado, se publica nuevamente un mensaje con la información de cada articulación y se envía al nodo de planificación de movimiento, que se encarga de generar una trayectoria y ejecutarla. En la Figura 4.13 se puede apreciar el robot en la posición de *pre-grasp*.

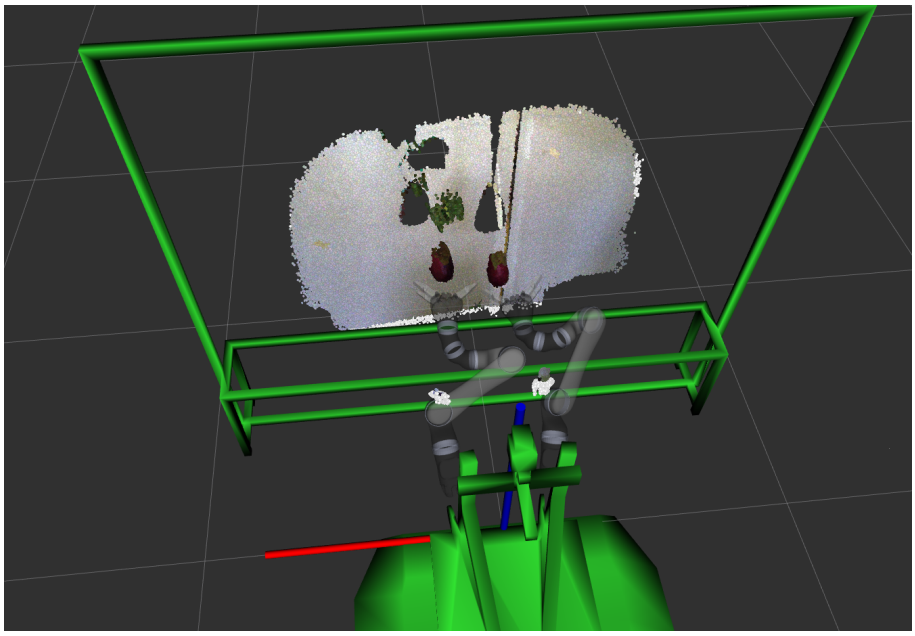


Figura 4.13: Simulación del robot ejecutando la tarea programada de recolección simultánea.

Por último, el planificador de movimiento aproxima los brazos al objetivo para realizar la recolección y tras cerrar los *grippers*, vuelve a la posición de reposo, donde se deberían depositar las berenjenas recogidas.

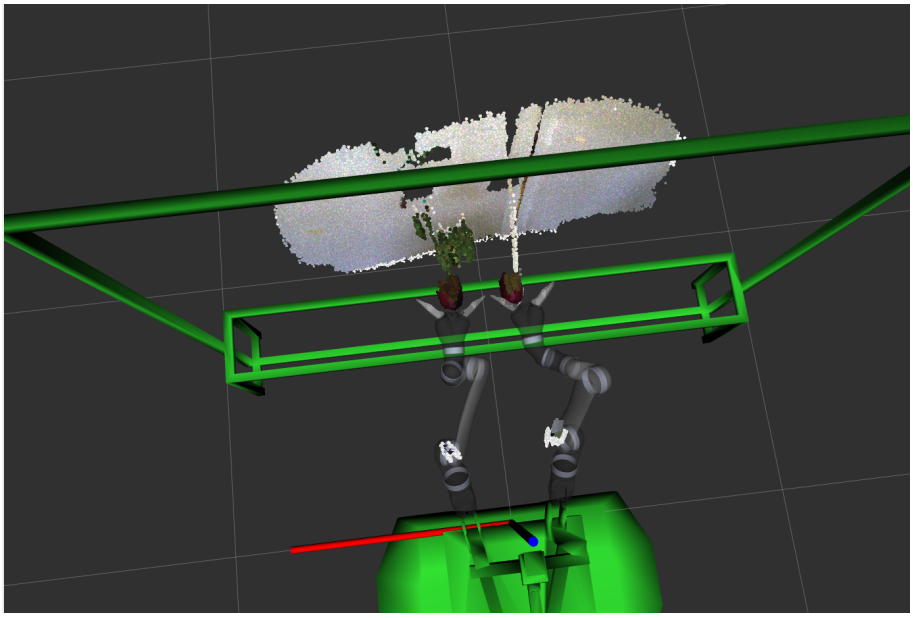


Figura 4.14: Simulación del robot en la posición indicada por los centroides calculados en los nodos de detección y localización y el nodo de planificación de tareas.

Estas pruebas se pueden repetir con cada escena que se considere necesario. Únicamente hay que construir artificialmente la escena en el banco de trabajo, y grabarla con las cámaras. Una vez se dispone de los datos capturados, el proceso se repite siguiendo los pasos descritos.

5. CONCLUSIONES Y LÍNEAS FUTURAS

En este capítulo se analizan los objetivos alcanzados en la ejecución del proyecto, resumiendo el trabajo realizado y destacando los puntos más importantes de la memoria. Adicionalmente, se realizan diversas propuestas de mejora del sistema actual y se mencionan las líneas de trabajo que quedan abiertas y que permitirían obtener un sistema más robusto y eficiente.

5.1. Objetivos alcanzados

A lo largo de este trabajo se ha diseñado y desarrollado un software encargado de la planificación de tareas a alto nivel del sistema robótico bimanual de la prueba de concepto PoC-ROBOCROP, consiguiendo así, dotar al robot de un sistema de toma de decisiones inteligente capacitado para realizar distintas acciones en función de los datos captados a través de su equipo multisensorial.

Analizando el grado de cumplimiento de los requerimientos de alto nivel propuestos al comienzo del trabajo se obtienen las siguientes conclusiones:

- Se ha realizado una migración completa del sistema existente en MATLAB al entorno de trabajo ROS. Además, las nuevas características y funcionalidades se han desarrollado por completo en este *middleware* y se puede integrar con el resto de la arquitectura de software de la línea de investigación. Se dispone, por tanto, de una arquitectura más robusta y unificada.
- Se han desarrollado y probado tres algoritmos de toma de decisiones que utilizan distintos parámetros del entorno para aplicar el criterio de ordenamiento que caracteriza la recolección de frutas. Los tres nodos son intercambiables entre sí debido a la modularidad del software diseñado. Esta característica facilita el uso del robot y dota a la arquitectura global de una gran adaptabilidad y escalabilidad para futuros proyectos.
- Gracias al sistema de herencia implementado, los tres nodos de planificación de tareas utilizan los mismos canales de comunicación para indicar las instrucciones generadas a los nodos del módulo de planificación del movimiento. Estas instrucciones se transmiten mediante un único mensaje que alberga toda la información necesaria para realizar cada tarea, de forma que se aporta robustez al código y se reducen las posibles fuentes de errores.
- El proceso de planificación de tareas puede ser visualizado gracias a la herramienta *OpenCV*. Se utiliza la imagen captada por la cámara para marcar mediante un código de colores las berenjenas que debe recoger cada brazo, y se dibuja el vector de trayectorias que describe el brazo para apartar las hojas en caso de que se encuentre una berenjena ocluida.
- Se han integrado todos los avances realizados en este Trabajo Fin de Máster con los trabajos en paralelo de la línea de investigación. El sistema de planificación de tareas es capaz de comunicarse de forma adecuada con el módulo de detección y localización y, utilizar los datos aportados por éste para realizar la toma de decisiones. Asimismo, las instrucciones generadas por el planificador a alto nivel son procesadas mediante un nodo encargado de decodificar las tareas encomendadas. En este mismo nodo se realiza un cálculo del modelo cinemático inverso y las transformaciones geométricas necesarias para que el módulo de planificador de trayectorias pueda interpretar las coordenadas recibidas y recolectar las berenjenas de forma óptima.

- Se han realizado pruebas de los algoritmos implementados y del sistema robótico completo mediante las herramientas de simulación de ROS. No obstante, no se han realizado suficientes pruebas con el robot físico en un entorno real como para validar todos los algoritmos. Esta tarea queda, por tanto, como un trabajo pendiente a realizar en las líneas futuras.

5.2. Líneas futuras

A pesar de todo el progreso realizado en este Trabajo Fin de Máster, la línea de investigación general abarca un proyecto muy grande y existen multitud de mejoras o implementaciones que se pueden añadir al sistema para seguir perfeccionándolo. A continuación, se listan algunas de estas mejoras, centrándose sobre todo en desarrollos relacionados con el sistema de planificación de tareas desarrollado.

En primer lugar, aunque las pruebas realizadas son suficientes para validar el trabajo realizado, se debería desarrollar un guion de pruebas más completo en el que se probasen todos los nodos y algoritmos en multitud de escenas. El registro y análisis de estas pruebas ayudaría a obtener una visión más precisa de la robustez del sistema propuesto y permitiría detectar posibles fallos que no se han podido detectar en las pruebas realizadas.

Tal y como se ha mencionado en la Sección 3.8, sería conveniente desarrollar un sistema de aprendizaje por refuerzo para realizar la toma de decisiones. Implementar un modelo de *Machine Learning* puede ayudar a obtener resultados óptimos. Además, durante este trabajo se han realizado los primeros pasos como la creación de un modelo sobre el que empezar a trabajar y diseñar el algoritmo de aprendizaje.

La implementación de un sistema de *feedback* avanzado entre el módulo de planificación de movimiento y el de planificación de tareas también se presenta como una línea de trabajo con gran potencial. No solamente se podría utilizar en el modelo de aprendizaje por refuerzo propuesto, sino que esta retroalimentación permitiría utilizar parámetros como el estado de las articulaciones del robot para elegir entre una acción u otra. Esta información se puede utilizar para la creación de nuevos nodos que prevean estados futuros del robot, y se anticipen a situaciones no factibles que se generen al requerir posiciones de los brazos donde se produzcan colisiones entre ellos.

En contraposición a la implementación de un sistema de aprendizaje por refuerzo, se podrían considerar otras opciones también interesantes. Un ejemplo es la aplicación de un planificador de tareas a alto nivel que fusione comportamientos elementales en función de la situación que observe. Dentro de los comportamientos elementales se podrían incluir los tres nodos de toma de decisiones desarrollados a lo largo del trabajo. Así, se dispondría de un planificador capaz de elegir el algoritmo de recolección más eficiente según su entorno, de forma que se obtiene un comportamiento complejo y versátil a partir de la fusión de otros algoritmos más simples.

Por último, para cumplir el objetivo general del proyecto, se tendría que validar el sistema de planificación de tareas implementado en un entorno con jardineras reales, en funcionamiento con el resto de subsistemas del robot, como los módulos de detección y localización o el módulo de planificación de movimiento.

6. PRESUPUESTO Y PLANIFICACIÓN

A lo largo de este capítulo se realiza el estudio económico del trabajo realizado y se explica brevemente la planificación temporal seguida para realizarlo. Es preciso tener en cuenta que únicamente se analiza el presupuesto parcial del trabajo que se refleja en esta memoria. No obstante, este trabajo forma parte de una línea de investigación que agrupa trabajos individuales de más profesionales y estudiantes, que no han sido considerados en este estudio al no disponer de suficiente información. El presupuesto total alcanzado es, por tanto, meramente informativo y su alcance se limita a este trabajo, no representa el coste total de la prueba de concepto PoC-ROBOCROP.

6.1. Planificación temporal

Para facilitar el entendimiento de esta sección, es conveniente utilizar el diagrama de Gantt que se ha incluido en el Anexo A. En dicho diagrama se puede observar de forma gráfica el reparto de tareas a lo largo del proyecto.

La duración total del proyecto es de 9 meses y 18 días. El trabajo comenzó el día 7 de septiembre de 2022 y concluye un día antes de realizar la entrega de esta memoria, el día 24 de junio de 2023. La fecha de inicio se debe tomar como referencia del comienzo del estudio y desarrollo del trabajo, ya que las comunicaciones con el tutor académico para acordar el alcance del proyecto comenzaron en los meses de verano de ese mismo año.

El total de horas invertidas al proyecto es de alrededor de 415 horas. Este tiempo dividido entre los 288 días de duración del mismo, resultan en 1 hora y 26 minutos diarios de media. No obstante, el reparto de horas no ha sido uniforme a lo largo de los meses de trabajo. La redacción de la memoria, por ejemplo, ha requerido una concentración de horas de trabajo en los últimos meses.

El Real Decreto 1125/2003 estable la relación de un crédito ECTS y horas invertidas entre 25 y 30 horas [29]. Esto supone unas horas totales de dedicación al Trabajo Fin de Máster de 360 horas. Existen, por tanto, 45 horas de trabajo extra.

Se ha dividido la duración total del proyecto en 4 módulos:

- **Investigación**: abarca el periodo dedicado a la familiarización con el entorno de trabajo y el estudio del sistema existente, sobre el cual se empezó a trabajar.
- **Desarrollo del software**: en este módulo se ha incluido gran parte del trabajo realizado. Abarca la programación y el diseño de la arquitectura de software, así como el desarrollo de las funciones para implementar todos los algoritmos descritos.
- **Integración**: engloba las tareas orientadas a la integración del trabajo realizado con las líneas de investigación paralelas y que forman parte de la prueba de concepto PoC-ROBOCROP. Este módulo ha tenido un avance más lento que el resto del trabajo, ya que requería la comunicación y coordinación de distintos investigadores, con horarios y lugares de trabajo distintos.
- **Memoria**: hace referencia a las tareas de redacción y revisión de esta memoria.

El flujo de trabajo se puede ver claramente en el diagrama de Gantt ya mencionado, aunque se ha intentado redactar la Sección 3 de Metodología siguiendo los pasos que se realizaron durante

el desarrollo. Inicialmente se realizó un análisis del sistema inicial. A continuación, se empezó a desarrollar el software, comenzando con el conversor de MATLAB. Seguidamente se programaron los nodos de procesamiento de datos y los tres nodos de toma de decisiones. Las pruebas realizadas al código se realizaron de forma simultánea a su programación, para ir verificando poco a poco el correcto funcionamiento del sistema.

Una vez se completó el desarrollo individual, comenzó el proceso de integración. Este proceso requirió un estudio del resto de componentes del sistema, para analizar y diseñar un plan de acción que redujera al mínimo los cambios requeridos en los distintos códigos, buscando siempre el funcionamiento completo del robot.

La redacción de esta memoria comenzó en la fase final del segundo módulo, y se realizó en paralelo a todo el proceso de integración.

6.2. Presupuesto

Se ha decidido dividir el presupuesto total del trabajo en tres apartados: costes directos de componentes, coste salarial y costes indirectos.

- **Costes directos de componentes:** la mayoría de componentes utilizados en el proyecto, como los brazos robóticos utilizados, deben ser comprados por encargo, por lo que no se puede encontrar su precio directamente a través de las páginas web de los proveedores. Para obtener estos datos se ha utilizado la información provista por el equipo y el personal que inició el proyecto hace dos años.

  Además, algunos de los componentes solamente se encuentran disponibles en países fuera de la zona Euro, como Canadá o Estados Unidos. Por tanto, los precios que se han utilizado se han convertido con la equivalencia entre dólar y euro en el momento en el que se escribe la memoria. Esta equivalencia corresponde a 0,92 € por cada dólar americano.

  En la siguiente Tabla 6.1 se muestran los costes de cada componente.

Componente	**Precio (€)**	Unidades	Total (€)
Kinova Arms MICO2	19.450	2	38.900
Kinova KG3 Gripper	4.600	2	9.200
Cámara TRI016S con sensor Sony IMX273 CMOS	443	1	443
Cámara TOF Helios2 con sensor SonyDepthSense IMX556PLR CMOS [30]	1.377	1	1.377
Banco de pruebas	150	1	150
Soportes, alimentación y otros	500	1	500
		Total	50.570

Tabla 6.1: Precios de los componentes utilizados.

  El entorno de trabajo utilizado a lo largo de todo el proyecto (ROS) es un *middleware* de código abierto y, por tanto, completamente gratuito. El único programa utilizado que requiere licencia para su uso es MATLAB. No obstante, como el código desarrollado en MATLAB tiene un uso completamente temporal, y no se incluye en la versión final del software, se ha decidido no incluirlo en la valoración de costes.

- **Costes salariales**: para calcular el coste salarial se han utilizado las horas totales invertidas en el proyecto (ver sección 6.1). La referencia utilizada para el salario del alumno se ha obtenido del portal de transparencia de la Universidad Politécnica de Madrid, en el apartado de PDI o Personal Docente de Investigación. El puesto seleccionado como referencia ha sido el de "Ayudante", ya que se ha realizado un trabajo de investigación y el alumno no posee un doctorado. Este salario es de 25.908,44 € anuales [31]. Dicho salario, suponiendo un pago de 12 meses, 20 días laborables por mes y una jornada de 8 horas, equivale a una tasa de 13,50 €/hora.

  Por otro lado, el salario de la tutora corresponde al salario de científico titular y jefe de proyecto dentro del Centro de Automática y Robótica. La tasa correspondiente a este salario se ha aproximado a 30 €/hora. Además, se calculan unas horas por parte del tutor académico de dedicación exclusiva a este trabajo de aproximadamente 45 horas. Así, el coste salarial del trabajo se obtiene multiplicando las horas invertidas por la tasa de cada trabajador. El resultado se puede ver reflejando en la siguiente Tabla 6.2.

Concepto	**Tasa (€/h)**	Horas	Coste Total (€)
Alumno	13,50	415	5.602,50
Tutor académico	30,00	45	1.350,00
		Total	6.952,50

Tabla 6.2: Coste salarial del trabajo.

- **Costes indirectos**: se refiere a los costes que no se pueden asociar directamente con el trabajo, ya que engloban otros productos o servicios. En este caso, se trata de los costes amortizados de instalaciones y equipo utilizado.

  El trabajo ha sido desarrollado con el ordenador personal del alumno. Por otro lado, las pruebas físicas con el robot, así como alguna jornada de trabajo, se han realizado en las instalaciones del Centro de Automática y Robótica de Arganda del Rey, Madrid. No obstante, la mayoría del desarrollo se ha realizado desde la vivienda personal del alumno, a la que no se le asocia ningún coste.

  El precio original del ordenador del alumno fue de 1000 € aproximadamente, y posee 8 años de antigüedad. El desarrollo del trabajo se comenzó en septiembre de 2022, lo que supone una duración de 10 meses del proyecto. Por tanto, el coste del uso del ordenador teniendo en cuenta una amortización lineal se puede calcular mediante la Ecuación 6.1.

$$\text{Coste (€)} = \frac{\text{Coste inicial}}{\text{Meses de vida}} \cdot \text{Meses utilizado} = \frac{1000}{8 \cdot 12} \cdot 10 = 104,17 \text{ €} \tag{6.1}$$

  Por último, únicamente restaría aproximar el coste estimado de la instalación donde se encuentra el robot y donde se realizan las pruebas. Sin embargo, como el laboratorio se sitúa dentro del edificio del Centro de Automática y Robótica de Arganda del Rey, se trata de un edificio público que no tiene coste particular asociado.

El coste total del trabajo se puede observar en la Tabla 6.3.

Concepto	**Coste (€)**
Componentes	50.570,00
Salarios	6.952,50
Indirectos	104,17
Total	57.626,67

Tabla 6.3: Coste total del trabajo realizado.

Finalmente, es preciso reseñar que todo el trabajo realizado se encuentra dentro de la línea de investigación de la prueba de concepto PoC-ROBOCROP. Este proyecto forma parte del Programa Estatal de I+D+i Orientado a los Retos de la Sociedad «Prueba de Concepto» 2021, financiado por el Ministerio de Ciencia e Innovación, a través de la Agencia Estatal de Investigación (AEI), con fondos del Mecanismo de Recuperación y Resiliencia de la Unión Europea.

7. ASPECTOS LEGALES Y EVALUACIÓN DE IMPACTOS

Este último capítulo tiene como finalidad exponer un análisis sobre los aspectos legales y éticos que afectan al sistema robótico que se ha desarrollado, así como realizar una evaluación sobre el impacto que tendría su aplicación en la agricultura, considerando aspectos sociales, económicos y medioambientales.

7.1. Aspectos legales y éticos

Aunque la introducción de la robótica en la industria comenzó en la segunda mitad del siglo pasado, no ha sido hasta la última década cuando la tecnología ha alcanzado un grado de madurez suficiente como para empezar a implementarse a gran escala en la sociedad. Se trata, por tanto, de una tecnología relativamente joven y que está evolucionando con gran rapidez.

Esta característica dificulta la creación de normativas y leyes que regulen su uso y aplicación, tanto en la industria como en el resto de sectores laborales o domésticos. En 2017, el Parlamento Europeo redactó unas "recomendaciones destinadas a la Comisión sobre normas de Derecho civil sobre robótica" [32]. Este documento, tenía como objetivo redactar unas pautas que sirvieran como guía a la hora de diseñar y desarrollar robots. Se consideran diferentes campos de aplicación y distintos sistemas robóticos. Se proponían medidas como la creación de una agencia europea formada por expertos en robótica e inteligencia artificial para que apoyara las instituciones públicas.

Lo más destacable, sin embargo, es la elaboración de un código de conducta ético para ingenieros en robótica. Aquí se presentan indicaciones acerca de la labor ética del ingeniero y su relación con las leyes y derechos existentes. Se hace especial hincapié en la seguridad, mencionando que "los diseñadores de robots han de tener en cuenta y respetar la integridad física, la seguridad, la salud y los derechos de las personas". También se trata la creación de un código deontológico para comités de ética. No obstante, aunque se marquen pautas a seguir, este documento no supone la elaboración de ninguna ley que regule el uso y diseño de robots.

De hecho, en el momento de redacción de esta memoria, no existe ninguna ley que regule el uso de la inteligencia artificial en el mundo, aunque la Unión Europea está cerca de cambiar esta situación. En 2018 comenzó un proceso para estudiar la elaboración de un marco regulador acerca de las inteligencias artificiales [33]. El primer hito alcanzado se produjo dos años más tarde, cuando se redactó la primera propuesta de reglamento en la que se establecían unas normas armonizadas. El principal objetivo de esta propuesta era "abordar los riesgos de usos específicos de la IA, clasificándolos en 4 niveles diferentes: riesgo inaceptable, alto riesgo, riesgo limitado y riesgo mínimo" [34].

En el mes de mayo de 2023, el futuro texto de la ley fue aprobado por los comités de Mercado Interior y de Libertades Civiles [35]. Esto deja más cerca la redacción de una normativa encargada de regular el uso de la inteligencia artificial y que marque las bases sobre las que se desarrollarán las futuras leyes nacionales. El proyecto en el que se enmarca este trabajo estará altamente influenciado por esta normativa, por lo que el futuro desarrollo del mismo dependerá en gran medida del marco regulador que se termine estableciendo en este proceso.

Sin embargo, hasta que entre en vigor una normativa específica sobre robótica e inteligencia

artificial, el proyecto únicamente se ve afectado por las leyes nacionales vigentes. El sistema robótico propuesto incluye cámaras encargadas de registrar el entorno. Por lo tanto, podría verse afectado por el Reglamento General de Protección de Datos [36]. Aunque en la fase actual del proyecto no es posible que personas sean grabadas debido al tipo de pruebas que se realiza, este factor puede resultar importante en la futura implementación del sistema robótico en un entorno agrícola real.

Algo similar ocurre con la regulación sobre protección y seguridad laboral. No está dentro de los objetivos del proyecto coordinar el robot con agricultores humanos pero, como toda maquinaria, al operar en un entorno no controlado donde pueden coexistir humanos, es necesario cumplir algunos requisitos mínimos de seguridad que regula dicha normativa.

Existe también una gran cantidad de reglamentos acerca de la industria agrícola en España, aunque la mayoría se centran en la protección del suelo y de las aguas frente a sustancias contaminantes, como el Real Decreto 1051/2022 sobre la nutrición de suelos sostenibles [37] o el Real Decreto 47/2022 sobre la protección de las aguas [38]. El proyecto no se ve afectado por este tipo de normativa al no utilizar fertilizantes ni producir ningún tipo de contaminante, pero si se amplían las funciones del sistema robótico propuesto, tendrían que ser tenidos en cuenta. Sin embargo, sí que habría que considerar el impacto del Real Decreto 448/2020 sobre maquinaria agrícola [39] una vez se definan las características del robot móvil que se utilizará como base para el robot manipulador.

7.2. Evaluación de impactos sociales, económicos y medioambientales

El desarrollo y la integración de la robótica en el sector agrícola está en plena fase de desarrollo. Es lógico pensar que a medida que se avance en las numerosas líneas de investigación que se encuentran abiertas, y el uso de la robótica agrícola se vaya extendiendo, este sector se vea beneficiado por el impacto socio-económico que ha traído la robótica a otros sectores industriales.

El trabajo desarrollado colabora en la creación de un sistema robótico inteligente con capacidad para trabajar en entornos agrícolas de forma autónoma. Este sistema tiene la capacidad para mejorar la eficiencia y productividad de cualquier plantación en la que se utilice, en especial de aquellas que todavía requieren de una gran mano de obra. La automatización de tareas repetitivas ayuda a reducir la carga de trabajo humano y puede ser utilizada en las tareas físicas que generan lesiones laborales en los trabajadores, mejorando de esta forma las condiciones laborales en el sector. Además, ofrece una solución a la falta de mano de obra producida por la despoblación de los sectores rurales y la migración de la población a las ciudades.

La mejora en el rendimiento y la reducción de la mano de obra permite reducir los costes salariales de las plantaciones y aumentar la cantidad de producto que puede ser recogido. Un aumento de la disponibilidad de producto con costes laborales menores se traduce en una reducción de los precios y una mayor disponibilidad de mercado, que puede ayudar a combatir la subida de precios generada por la alta inflación. Todos estos factores ayudan a aumentar la rentabilidad y competitividad de los agricultores.

El proyecto también promueve la innovación y el desarrollo tecnológico en la agricultura, que trae asociado la generación de empleo tecnológico de calidad y crecimiento económico. La integración de la robótica en este sector proporciona un nuevo mercado emergente y oportunidades para la creación de empresas, puestos laborales y nuevas colaboraciones de investigación entre universidades, centros de investigación y empresas privadas.

En el ámbito medioambiental, el proyecto propone un nuevo método de recolección selectiva mediante el uso de energía limpia y no contaminante. Al utilizar únicamente componentes electrónicos, es relativamente sencillo implementar sistemas de carga mediante energía renovable o establecer los periodos de carga durante las noches u horas valle de demanda del sistema eléctrico, lo que ayudaría a reducir las emisiones.

Aunque las funcionalidades del sistema robótico propuesto únicamente incluyen acciones de recolección, una vez se disponga de un robot funcional sería posible incluir nuevas funciones como la fumigación selectiva o la aplicación de fertilizantes. Hoy en día, estos procesos se realizan de forma masiva por falta de mano de obra, lo que provoca un gran deterioro en el ecosistema. La adaptación de robots similares al utilizado en este proyecto puede ayudar a disminuir el uso de productos químicos y, consecuentemente, contribuir a reducir la contaminación ambiental.

Además, la mejora del rendimiento en los procesos de recolección provoca una reducción del desperdicio de alimentos. La recolección selectiva busca minimizar los daños producidos en las frutas recogidas, asegurando una manipulación de los alimentos no dañina. Esto, sumado a una mayor capacidad de recolección en el sector agrícola, contribuiría a una reducción de los desperdicios alimentarios.

No obstante, aunque el impacto general de la línea de investigación es extremadamente positivo, también deben ser considerados los cambios culturales y laborales provocados por la implementación de este tipo de sistemas robóticos en el sector agrícola. La reducción de mano de obra, aunque supone una reducción de costes laborales, genera la eliminación de algunos puestos de empleo tradicionales. Sería necesario reacondicionar estos puestos para que puedan adaptarse a las nuevas tecnologías.

La integración de la robótica en el sector agrícola también provoca la necesidad de mano de obra especializada para el mantenimiento y control de los robots. Serían necesarios protocolos de formación de los trabajadores a las nuevas tecnologías para garantizar una correcta adaptación a las novedades del sector.

Por último, es preciso considerar el impacto cultural del proyecto. La adaptación de la robótica a nuevos sectores y los cambios que provoca en la industria puede ser visto como algo negativo por parte de la sociedad. Es necesario dar a conocer y promover este tipo de tecnologías para que se conozcan los beneficios que aportan y facilitar la integración de la robótica en nuevos sectores industriales y de servicios.

7.3. Objetivos de Desarrollo Sostenible

Los Objetivos de Desarrollo Sostenible u ODS, desarrollados por la ONU [40], suponen una gran herramienta para analizar el impacto general del proyecto. La gran cantidad de ODS que son afectados directa o indirectamente por el proyecto, son un claro indicativo de los numerosos beneficios que aporta a la sociedad. Estos objetivos son:

- **ODS 2, Hambre cero**: existe un efecto directo de la aplicación de sistemas robóticos para mejorar el rendimiento y la eficiencia de la agricultura con la producción de alimentos. El aumento de productos agrícolas ayuda a garantizar un suministro de alimentos global, y puede ser extendido a países donde las condiciones agrícolas sean más duras y los alimentos más requeridos.
- **ODS 7, Energía asequible y no contaminante**: la robótica agrícola puede ayudar a

reducir el uso de combustibles fósiles en la agricultura. Además, se pueden diseñar sistemas de recarga de robots a partir de energía completamente renovable.

- **ODS 8, Trabajo decente y crecimiento económico**: la robótica agrícola se encuentra en plena fase de desarrollo. Este proyecto contribuye a la aparición de un nuevo mercado emergente con nuevas oportunidades laborales, crecimiento económico y generación de empleo de calidad.
- **ODS 9, Industria, innovación e infraestructura**: estrechamente relacionado con el anterior ODS, la creación de un nuevo mercado a través del uso de nuevas tecnologías fomenta la innovación y aparición de una nueva industria, con posibilidades de aplicar los últimos avances de campos de investigación como la inteligencia artificial.
- **ODS 12, Producción y crecimiento responsable**: la recolección selectiva ayuda a reducir el desperdicio de alimento, a través de un agricultura sostenible. La utilización de estos sistemas inteligentes puede ser un factor decisivo en la lucha contra el desperdicio generado por las granjas dedicadas a la producción en masa.
- **ODS 13, Acción por el clima**: al igual que el ODS 7, la utilización de sistema robóticos con la capacidad de utilizar energía completamente renovable puede ayudar a reducir las altas emisiones de la industria agrícola, producidas por el uso de maquinaria alimentada por combustibles fósiles.
- **ODS 15, Vida de ecosistemas terrestres**: el sistema robótico propuesto supone un método sostenible que puede ayudar a proteger los ecosistemas terrestres, reduciendo el uso de la maquinaria extensiva que resulta dañina para el medio ambiente. Los robots pueden ser programados para respetar la vida animal y vegetal de los ecosistemas, de forma que realicen su trabajo sin modificar ni dañar el entorno en el que trabajan.
- **ODS 17, Alianza para lograr los objetivos**: la naturaleza de la investigación promueve la colaboración entre universidades, empresas y centros de investigación a nivel nacional e internacional. Las últimas mejoras tecnológicas pueden ser aplicadas en otros proyectos que se realizan dentro de Europa, lo que fomentaría la creación de alianzas para desarrollar sistemas sostenibles complejos.

DATOS DEL PROYECTO

Este Trabajo Fin de Máster forma parte del proyecto de I+D+i PDC2021-121578-I00 (PoC-ROBOCROP – Prueba de Concepto: Robot Autónomo de Doble-Brazo para Cosecha Selectiva), financiado por MCIN/AEI/10.13039/501100011033/ y por la Unión Europea NextGenerationEU/PRTR.

BIBLIOGRAFÍA

[1] Leonidas Droukas et al. «A Survey of Robotic Harvesting Systems and Enabling Technologies». En: *Journal of Intelligent & Robotic Systems* 107.21 (ene. de 2023). DOI: 10.1007/s10846-022-01793-z.

[2] Ebrahim Navid Sadjadi y Roemi Fernández. «Challenges and Opportunities of Agriculture Digitalization in Spain». En: *Agronomy* 13.1 (2023). ISSN: 2073-4395. DOI: 10.3390/agronomy13010259.

[3] Gobierno de España Agencia Estatal de Investigación. *Proyecto I+D+i «Prueba de Concepto» 2021: Prueba de concepto: robot autónomo de doble-brazo para cosecha selectiva (PoCROBOCROP)*. URL: https://www.aei.gob.es/ayudas-concedidas/ayudas-destacadas/proyecto-idi-prueba-concepto-2021-prueba-concepto-robot (visitado 14 de jun. de 2023).

[4] Mykel J Kochenderfer, Tim A Wheeler y Kyle H Wray. *Algorithms for decision making*. Cambridge, Massachusetts: The MIT press, 2022.

[5] Tiehua Cao y Arthur C Sanderson. *Intelligent task planning using fuzzy Petri nets*. Vol. 3. Singapur: World Scientific, 1996.

[6] Tom M Mitchell. *Machine learning*. Vol. 1. 9. New York: McGraw-Hill, 1997.

[7] Comisión Europea. *Robots for protecting crops*. Cordis. URL: https://cordis.europa.eu/project/id/101016807/es (visitado 6 de abr. de 2023).

[8] Comisión Europea. *A Collaborative Paradigm for Human Workers and Multi-Robot Teams in Precision Agriculture Systems*. Cordis. URL: https://cordis.europa.eu/project/id/101016906/es (visitado 6 de abr. de 2023).

[9] Pal Robotics. *CANOPIES*. URL: https://pal-robotics.com/es/proyectos-colaborativos/canopies/ (visitado 6 de abr. de 2023).

[10] Martina Lippi et al. «Enabling Visual Action Planning for Object Manipulation through Latent Space Roadmap». En: *CoRR* abs/2103.02554 (2021). arXiv: 2103.02554. URL: https://arxiv.org/abs/2103.02554.

[11] Pal Robotics. *Project CANOPIES: TIAGo++ starts work in an Italian vineyard*. URL: https://blog.pal-robotics.com/tiago-work-in-italian-vineyards-for-project-canopies/ (visitado 7 de abr. de 2023).

[12] Carlos Rodriguez de Cos y Dimos V. Dimarogonas. «Adaptive Cooperative Control for Human-Robot Load Manipulation». En: *IEEE Robotics and Automation Letters* 7.2 (2022), págs. 5623-5630. DOI: 10.1109/LRA.2022.3158435.

[13] Universal Robots. *Robots para agricultura: hacia un mayor aprovechamiento de los recursos*. URL: https://www.universal-robots.com/es/blog/robots-para-agricultura/ (visitado 7 de abr. de 2023).

[14] Comisión Europea. *MoBile Robotic PlAtforms for ACtive InspeCtion and Harvesting in AgricUltural AreaS*. Cordis. URL: https://cordis.europa.eu/project/id/871704/es (visitado 6 de abr. de 2023).

[15] *BACCHUS. Mobile robotic platforms for active inspection & harvesting in agriculturar areas*. Bacchus. URL: https://bacchus-project.eu/ (visitado 7 de abr. de 2023).

[16] Alexios Papadimitriou et al. «Loop Closure Detection and SLAM in Vineyards with Deep Semantic Cues». En: 2022 International Conference on Robotics and Automation (ICRA) (Philadelphia, PA, USA, 12 de jul. de 2022), págs. 2251-2258. DOI: 10.1109/ICRA46639.2022.9812419.

[17] Sotiris Stavridis, Dimitrios Papageorgiou y Zoe Doulgeri. «Kinesthetic teaching of bimanual tasks with known relative constraints». En: IEEE/RSJ International Conference on Intelligent Robots and Systems 2022 (IROS 2022) (Kyoto, Japan, 22 de oct. de 2022), págs. 11796-11801. DOI: 10.1109/IROS47612.2022.9981196.

[18] Papageorgiou Dimitrios, Koutras Leonidas y Doulgeri Zoe. «A controller for reaching and unveiling a partially occluded object of interest with an eye- in-hand robot». En: IEEE-RAS 21st International Conference on Humanoid Robots (Humanoids) (Ginowan, Okinawa, Japan, 28 de sep. de 2022). Zenodo. DOI: 10.5281/zenodo.7128778.

[19] Comisión Europea. *Intelligent sensing and manipulation for sustainable production and harvesting of high value crops, clever robots for crops*. Cordis. URL: https://cordis.europa.eu/project/id/246252/reporting (visitado 26 de abr. de 2023).

[20] J. Bontsema, J. Hemming y E.J. Pekkeriet. «CROPS Clever Robots for Crops». En: *IET Engineering & Technology Reference* 2015 (2015), págs. 1-11. ISSN: 2056-4007. DOI: 10.1049/etr.2015.0015.

[21] R. Fernández, D. Sepúlveda, E. Navas, M. Armada y P. Gonzalez-de-Santos. «Detección Inteligente y Manipulación Robótica Bimanual para la Cosecha Selectiva de Cultivos de Alto Valor». En: Jornadas Nacionales de Robótica (Alicante, España, 13 de jun. de 2019). Universidad de Alicante, Universidad Miguel Hernández de Elche.

[22] Alexandre Campeau-Lecours et al. «Kinova Modular Robot Arms for Service Robotics Applications». En: *Int. J. Robotics Appl. Technol.* 5.2 (2017), págs. 49-71. DOI: 10.4018/IJRAT.2017070104.

[23] Delia SepúLveda, Roemi Fernández, Eduardo Navas, Manuel Armada y Pablo González-De-Santos. «Robotic Aubergine Harvesting Using Dual-Arm Manipulation». En: *IEEE Access* 8 (2020), págs. 121889-121904. DOI: 10.1109/ACCESS.2020.3006919.

[24] *Document Search*. Results obtained with query: "TITLE-ABS-KEY (ros) AND TITLE-ABS-KEY (robot)". Scopus. URL: https://www.scopus.com/ (visitado 9 de abr. de 2023).

[25] *ROS Robotics Companies*. Github. URL: https://github.com/vmayoral/ros-robotics-companies#active-companies (visitado 7 de abr. de 2023).

[26] Joseph Redmon, Santosh Divvala, Ross Girshick y Ali Farhadi. «You only look once: Unified, real-time object detection». En: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, págs. 779-788.

[27] TRACLABS. *TRACLabs' IK solver Public Repository*. URL: https://bitbucket.org/traclabs/trac_ik/src/master/ (visitado 19 de jun. de 2023).

[28] Richard S. Sutton y Andrew G. Barto. *Machine learning*. Vol. 1. 1. Cambridge, Massachusetts: The MIT Press, 2015.

[29] «Real Decreto 1125/2003, de 5 de septiembre, por el que se establece el sistema europeo de créditos y el sistema de calificaciones en las titulaciones universitarias de carácter oficial y validez en todo el territorio nacional.» En: *Boletín Oficial del Estado* 224 (18 de sep. de 2003), págs. 34355-34356. URL: https://www.boe.es/buscar/doc.php?id=BOE-A-2003-17643.

[30] A3. *Helios2 Time-of-Flight 3D Camera*. URL: https://www.automate.org/products/lucid-vision-labs/helios2 (visitado 14 de jun. de 2023).

[31] Universidad Politécnica de Madrid. *Portal de transparencia, Personal PDI*. URL: https://transparencia.upm.es/personal/pdi/bandasSalariales (visitado 14 de jun. de 2023).

[32] Parlamento Europeo. «Resolución del Parlamento Europeo, de 16 de febrero de 2017, con recomendaciones destinadas a la Comisión sobre normas de Derecho civil sobre robótica (2015/2103(INL))». En: *Textos aprobados* (16 de feb. de 2017). URL: https://www.europarl.europa.eu/doceo/document/TA-8-2017-0051_ES.html.

[33] Comisión Europea. *Un enfoque europeo de la inteligencia artificial*. URL: https://digital-strategy.ec.europa.eu/es/policies/european-approach-artificial-intelligence (visitado 10 de jun. de 2023).

[34] Comisión Europea. «Propuesta de Reglamento por el que se establecen normas armonizadas en materia de inteligencia artificial». En: *Policy and legislation* (21 de abr. de 2021). URL: https://digital-strategy.ec.europa.eu/es/library/proposal-regulation-laying-down-harmonised-rules-artificial-intelligence.

[35] Parlamento Europeo. «AI Act: a step closer to the first rules on Artificial Intelligence». En: *Sala de prensa* (11 de mayo de 2023). URL: https://www.europarl.europa.eu/news/en/press-room/20230505IPR84904/ai-act-a-step-closer-to-the-first-rules-on-artificial-intelligence.

[36] «Reglamento (UE) 2016/679 del Parlamento Europeo y del Consejo». En: *Diario Oficial de la Unión Europea* (27 de abr. de 2016). URL: https://www.boe.es/doue/2016/119/L00001-00088.pdf.

[37] «Real Decreto 1051/2022, de 27 de diciembre, por el que se establecen normas para la nutrición sostenible en los suelos agrarios». En: *Boletín Oficial del Estado* 312 (29 de dic. de 2022), págs. 188873-188916. URL: https://www.boe.es/diario_boe/txt.php?id=BOE-A-2022-23052.

[38] «Real Decreto 47/2022, de 18 de enero, sobre protección de las aguas contra la contaminación difusa producida por los nitratos procedentes de fuentes agrarias». En: *Boletín Oficial del Estado, Legislación Consolidada* 17 (20 de ene. de 2022). URL: https://www.boe.es/buscar/pdf/2022/BOE-A-2022-860-consolidado.pdf.

[39] «Real Decreto 448/2020, de 10 de marzo, sobre caracterización y registro de la maquinaria agrícola». En: *Boletín Oficial del Estado* 117 (27 de abr. de 2020), págs. 30365-30391. URL: https://www.boe.es/diario_boe/txt.php?id=BOE-A-2020-4666.

[40] Organización de las Naciones Unidas. *Objetivos de Desarrollo Sostenible*. URL: https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/ (visitado 14 de jun. de 2023).

ANEXOS

A. Diagrama de Gantt

El diagrama de Gantt se muestra en la siguiente página apaisada para mejorar su visualización.

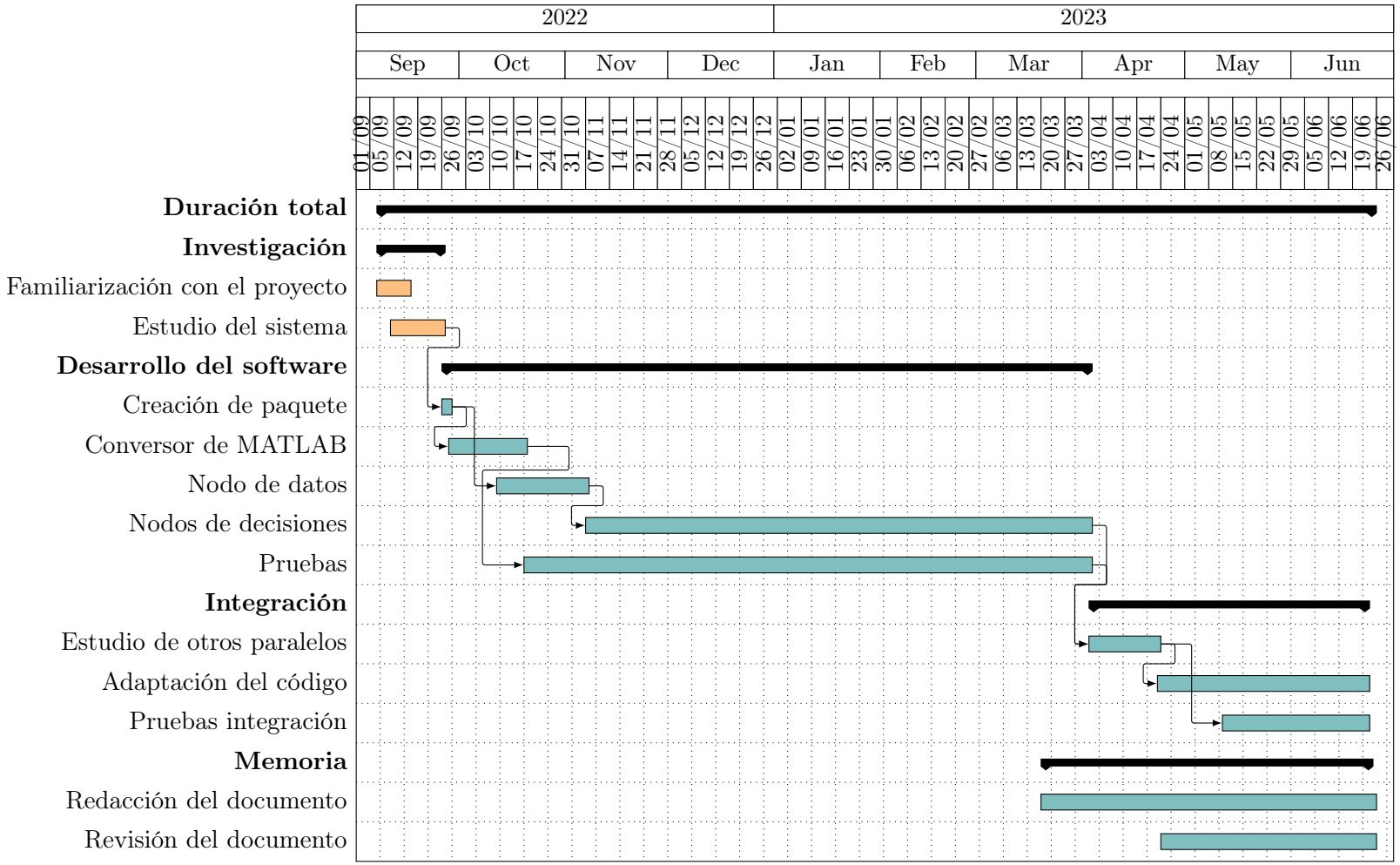


Figura A.1: Ejemplo de diagrama de Gantt con eje temporal introducido manualmente