



Escuela Superior de Ciencias Experimentales y Tecnología

**GRADO EN INGENIERÍA
DE TECNOLOGÍAS INDUSTRIALES**

Trabajo de Fin de Grado

**BRAZO ROBÓTICO CON CÁMARA ÚNICA
PARA RECOLECTAR FRESAS MEDIANTE
DEEP LEARNING**

David Campoamor Medrano

Director: Julio Vega Pérez

Curso Académico 2024/25



Grado en Ingeniería de Tecnologías Industriales

Trabajo de Fin de Grado

El presente trabajo, titulado ***BRAZO ROBÓTICO CON CÁMARA ÚNICA PARA RECOLECTAR FRESAS MEDIANTE DEEP LEARNING***, constituye la memoria correspondiente a la asignatura Trabajo de Fin de Grado que presenta D./D^a. ***DAVID CAMPOAMOR MEDRANO*** como parte de su formación para aspirar al Título de Graduado/a en Ingeniería de Tecnologías Industriales. Este trabajo ha sido realizado en la ***ESCUELA DE INGENIERÍA DE FUENLABRADA*** en el ***DEPARTAMENTO DE TEORÍA DE LA SEÑAL Y COMUNICACIONES Y SISTEMAS TELEMÁTICOS Y COMPUTACIÓN*** bajo la dirección de ***JULIO VEGA PÉREZ***.

Móstoles, 26 de mayo de 2025



©2025 David Campoamor Medrano

Algunos derechos reservados.

Este trabajo se entrega bajo licencia CC-BY-NC-SA 4.0.

Agradecimientos

Nunca es tarea fácil agradecer a tantas personas el apoyo, la ayuda y los consejos que han contribuido en mi beneficio, tanto personal como académico, durante todos estos años.

En primer lugar, me gustaría dar las gracias tanto a la Universidad Rey Juan Carlos como a todos los profesores de los que he tenido el privilegio de ser alumno, por haber sido capaces de transmitir la dedicación, pasión, disciplina y el esfuerzo tan imprescindible como necesarios para la praxis de una profesión como lo es la de ingeniero, y más concretamente en mi caso, la de ingeniero industrial.

Quisiera expresar mi gratitud a mi tutor, Julio Vega, por guiarme, acompañarme y ayudarme durante estos meses de trabajo, para mí fue todo un honor saber que finalmente había aceptado dirigir este trabajo final de grado, y de este modo cerrar un bonito círculo que empezó con él como profesor mío de informática en el colegio, donde nos enseñó, entre otras muchas cosas, que más allá de los editores de texto convencionales, existen otros sistemas para la preparación de documentos, por esto, este trabajo también es en parte suyo, ya que tanto estas líneas como el resto del documento están basados en sus enseñanzas.

Asimismo, me gustaría agradecer a Robotplus, por cumplimentar mi formación académica y darme mi primera oportunidad laboral en el ámbito industrial, y más concretamente a mis compañeros del departamento de servicio técnico y a los del departamento de I+D+i, ya que gracias a ellos hoy por hoy he podido entender y experimentar más en profundidad muchos de los principios teóricos y de los problemas que únicamente conocía sobre el papel, pudiendo desarrollarme de una manera más completa como profesional.

Agradecer también a mis amigos y compañeros de clase, los *Hijos de la Ingeniería* y David, por no haber dejado que me rindiera incluso en los peores momentos y con todo en contra, y por haber sido un gran apoyo tanto dentro como fuera de la universidad.

A mis amigos del equipo de baloncesto en Alcorcón, en especial a Rober y a Adri, por haber confiado siempre en que este momento llegaría, antes o después, y haber formado parte de este proceso del que desde antes de empezar la universidad ya formaban parte, al igual que mis amigos de Móstoles del colegio, el *Cártel de La Manga*. Y sobre todo, gracias a Sandra, por ser para mí el claro ejemplo de que la dedicación y el trabajo duro merecen la pena, pero más allá de todo esto, por estar a mi lado día a día y ser mi compañera de vida, sin ella no habría podido soñar con finalmente llegar hasta aquí.

No querría concluir los agradecimientos sin hacer partícipe a toda mi familia, y en especial a mis padres y mi hermano, la paciencia que han tenido todo este tiempo conmigo, sobre todo en época de entregas y de exámenes, pero sobre todo y más importante, la confianza depositada en mí, que mediante palabras y gestos de apoyo incondicional han demostrado. Ha sido gracias a este amor y apoyo que solo la familia sabe darte cuando más lo necesitas, por lo que sido más fácil poder alcanzar esta meta. Gracias a mis tíos y a mis primos mayores, por hacer que me interesase en el mundo de las ciencias, y más concretamente en la ingeniería y la construcción, faceta en la que ya desde pequeño había fijado mi atención jugando con aquellos bloques fabricados en plástico ABS y de colorines, ya que sin duda, fue gracias a ellos por lo que terminé de decidir embarcarme, ya desde el colegio, en las materias que guardaban mayor similitud con estos aspectos antes que en otras, puesto que veía en ellos una referencia a seguir. Pero sobre todo, gracias a mis abuelos, que como suele decirse, deberían ser eternos. Si antes hablaba de referencias, sin duda ellos han sido el máximo exponente en esto, puesto que sin sus enseñanzas y consejos, y no solo en aspectos académicos, no podría haber llegado hasta aquí. Todos ellos siempre formarán parte de mi y estarán presentes en cada una de las tomas de decisiones importantes que tenga que llevar a cabo, en las desilusiones y en los malos ratos, pero también en la consecución de mis éxitos y logros, como es el caso, aunque algunos de ellos ya no se encuentren entre nosotros o no puedan recordarlo. Espero haber podido aprender y retener algo de la sabiduría que me habéis mostrado y trasmitido.

A todas aquellas personas que, con trabajo y esfuerzo, terminan consiguiendo todo aquello que se proponen.

Madrid, 26 de mayo de 2025
David Campoamor Medrano

Resumen

La robótica y la visión artifical han revolucionado numerosos sectores, incluida la agricultura, en la que, a pesar de los avances tecnológicos, la recolección manual de las frutas y verduras sigue siendo un proceso laborioso, exigente y sujeto a tareas repetitivas susceptibles de derivar en errores humanos.

Uno de los mayores desafíos en este campo es la recolección de frutas pequeñas y delicadas, como lo son en particular las fresas dada su gran variabilidad en tamaño, forma y grado de maduración; ya que requieren gran precisión y un alto consumo de tiempo y esfuerzo físico por quienes lo realizan. Es por esto que la automatización de su recolección se ha convertido en una alternativa para poder mejorar y optimizar su eficiencia, reduciendo la dependencia de la mano de obra humana mediante el uso de la robótica y la inteligencia y visión artificial para identificar, seleccionar y recoger los frutos en el momento óptimo.

El presente trabajo pretende solucionar este problema mediante el desarrollo de un sistema de visión artificial para detectar el estado de maduración de las fresas y facilitar su recolección de forma automatizada, siempre y cuando el estado de maduración de la fresa sea el adecuado, con un brazo robótico y utilizando el modelo YOLOv3 en tiempo real. Mediante el procesamiento de las imágenes capturadas por una cámara web, el sistema identifica la posición y calcula la distancia de cada fresa con respecto a la cámara para poder transmitir esta información a un brazo robótico de Universal Robots a través del protocolo XML-RPC, permitiendo que el robot ejecute esta recolección de forma autónoma y precisa.

Los experimentos realizados han demostrado que el sistema puede identificar fresas maduras con alta precisión en distintas condiciones de iluminación. Además, la integración con el brazo robótico ha permitido validar la eficacia del sistema en la recolección autónoma, logrando resultados satisfactorios en términos de exactitud. Estos avances confirman la viabilidad de la propuesta y sientan las bases para futuras mejoras en rendimiento, velocidad y adaptabilidad a otros cultivos.

Abstract

Artificial intelligence and robotics have revolutionised numerous sectors, including agriculture, where, despite technological advances, the manual harvesting of fruits and vegetables remains a labour-intensive, demanding process, prone to repetitive tasks and human error.

One of the greatest challenges in this field is the harvesting of small and delicate fruits, such as strawberries, which exhibit high variability in size, shape, and ripeness level. These fruits require great precision and significant physical effort and time from those who harvest them. For this reason, the automation of harvesting has become an alternative to enhance and optimise efficiency, reducing dependence on human labour through the use of robotics and artificial intelligence and vision to identify, select, and harvest the fruits at the optimal moment.

This project aims to address this problem by developing a computer vision system capable of detecting the ripeness stage of strawberries and facilitating their automated harvesting, provided that the fruit is at the appropriate stage. The system employs a robotic arm and uses the YOLOv3 model in real time. By processing images captured by a webcamera, the system identifies the position and calculates the distance of each strawberry from the camera in order to transmit this information to an Universal Robots robotic arm via XML-RPC protocol, allowing the robot to perform harvesting in an autonomous and precise manner.

The experiments conducted have demonstrated that the system can identify ripe strawberries with high accuracy under varying lighting conditions. Furthermore, integration with the robotic arm has validated the system's effectiveness in autonomous harvesting, yielding satisfactory results in terms of precision. These advances confirm the feasibility of the proposed approach and lay the foundation for future improvements in yield, scalability, and adaptability to other crops.

Acrónimos

ABB *Asea Brown Boveri*

AER *Asociación Española de Robótica*

AERO *Autonomous Exploration Rover*

AGV *Automated Guided Vehicle*

AI *Artificial Intelligence*

AMR *Autonomous Mobile Robot*

ANN *Artificial Neural Network*

API *Application Programming Interface*

CMI *Cirugía Mínimamente Invasiva*

CPU *Central Processing Unit*

dFoV *diagonal Field of View*

DL *Deep Learning*

DLR *Centro Aeroespacial Alemán (Deutsches Zentrum für Luft - und Raumfahrt e. V.)*

DNN *Deep Neural Network*

DOF *Degree of Freedom*

EKF *Extended Kalman Filter*

EPFL *Escuela Politécnica Federal de Lausana*

FDA *Administración de Alimentos y Medicamentos de EE.UU.*

FOA *Focus of Attention*

FPS *Fotogramas por Segundo*

GA *Genetic Algorithm*

GPIO *General Purpose Input/Output*

GPS *Global Positioning System*

HCI *Human-Computer Interaction*

HRI *Human-Robot Interaction*

Hz *Hercio*

IA *Inteligencia Artificial*

IBM *International Business Machines*

IFR *International Federation of Robots*

IGR *Interfaz Gráfica del Robot*

IMTS *International Manufacturing Technology Show*

IP *Internet Protocol*

ISO *Internacional Organization for Standardization*

LTS *Long Term Support*

LWR *Lightweight Robot*

Mb *Megabit*

ML *Machine Learning*

NN *Neural Network*

OSRF *Open Source Robotics Foundation*

PE *Process Element*

PUMA *Programmable Universal Machine for Assembly*

RNA *Redes Neuronales Artificiales*

ROS *Robot Operating System*

ROS-I *Robot Operating System-Industrial*

RPC *Remote Procedure Call*

RPY *Roll Pitch Yaw*

SAIL *Stanford Artificial Intelligence Laboratory*

SCARA *Selective Compliance Assembly Robot Arm*

SCB *Safety Control Board*

SML *Shallow Machine Learning*

SRI *Stanford Research Institute*

TC *Technical Committee*

UR *Universal Robots*

UWB *Ultra-Wideband*

VA *Visión Artificial*

YOLO *You Only Look Once*

Índice general

1. Introducción	1
1.1. Los robots y la robótica	2
1.1.1. Robots industriales	2
1.1.2. Robots de servicio	6
1.1.3. Robots en medicina	7
1.2. Inteligencia Artificial	9
1.3. Visión Artificial	9
1.4. Machine Learning	10
1.5. Deep Learning	11
2. Estado del arte	14
3. Objetivos	22
3.1. Descripción del problema	22
3.2. Plan de trabajo	23
4. Plataforma de desarrollo	25
4.1. Hardware	25
4.1.1. Cámara Logitech C270 HD	25
4.1.2. Soporte de brazo articulado	26
4.1.3. Soporte de impresión 3D	26

4.1.4. Ordenador principal	27
4.1.5. Robot de Universal Robots de la gama e-series	27
4.1.6. Comunicaciones	28
4.2. Software	29
4.2.1. Ubuntu	29
4.2.2. Polyscope	29
4.2.3. Anaconda	31
4.2.4. Python	31
4.2.5. PyTorch	32
4.2.6. NumPy	32
4.2.7. OpenCV	32
4.2.8. OpenGL	33
4.2.9. SocketTest	33
4.2.10. XML-RPC	34
4.2.11. YOLOv3	35
5. Descripción del sistema	36
5.1. Hipótesis suelo adaptada al plano vertical	37
5.2. Detección de fresas mediante Deep Learning	40
5.3. Arquitectura del sistema	42
6. Conclusiones	51
6.1. Objetivos y requisitos cumplidos	51
6.1.1. Objetivos	51
6.1.2. Requisitos	52
6.2. Líneas Futuras	53

I. Metodología	54
I.1. Requisitos	54
I.2. Competencias	54
I.3. Metodología	57
I.4. Habilidades desarrolladas	58
II. Experimentos	60
II.1. Detección con YOLOv3 y TensorFlow	60
II.1.1. Pruebas con imágenes	60
II.1.2. Pruebas con vídeo en tiempo real	64
II.2. Detección con YOLOv3 y PyTorch	70
II.2.1. Pruebas con modelos preentrenados	70
II.2.2. Entrenamiento del modelo	71
II.2.3. Calibrado de la cámara	73
II.2.4. Pruebas detección de fresas en tiempo real	74
II.3. Pruebas con el robot real	93
Bibliografía	104

Índice de figuras

1.1.	Primer robot industrial	3
1.2.	Standford Arm	4
1.3.	Robots utilizados para el desarrollo de ROS	5
1.4.	UR5 con su controladora	5
1.5.	Robots de servicio	6
1.6.	Robots en medicina	8
1.7.	Diagrama de Venn de la relación entre distintas áreas de la IA	11
1.8.	Redes Neuronales	12
2.1.	Representación de varios tipos de robots agrícolas	14
2.2.	Ilustración global del rendimiento general de los robots revisados	15
2.3.	Agrobot	16
2.4.	Diseño conceptual del robot de recogida con sus componentes	17
2.5.	Robot agrícola Dogtooth	18
2.6.	Montaje del hardware en una explotación de fresas	19
2.7.	Robot recolector de tomates	21
4.1.	Sistema de visión	26
4.2.	Soporte de impresión 3D en PLA	27
4.3.	Gama e-series de Universal Robots ²³	28
4.4.	Pantalla principal de la interfaz de Polyscope 5	30

4.5. Pruebas realizadas con SocketTest para verificar la comunicación robot-servidor externo	34
5.1. Montaje del sistema final con un UR5e	36
5.2. La hipótesis suelo asume que todos los objetos están en el suelo	37
5.3. Plano pared generado en el UR	40
5.4. Detección de fresas maduras mediante deep learning	41
5.5. Disposición de las detecciones para un plano horizontal con un UR5e .	42
5.6. Ajustes de comunicación XMLRPC en el robot y ejecución de movimientos definidos en el programa	45
5.7. Ajustes de red en el robot	46
5.8. Definición de la Instalación del efector final en el robot	47
5.9. Programa <i>xmlrpc_deteccion_fresas_vertical_pinza.urp</i> para uso del sistema con efector final	49
5.10. Pruebas en el plano vertical de detección con garra integrada	50
I.1. Ciclo de la metodología DMADV	58
II.1. Resultado de la detección en imágenes con TensorFlow	61
II.2. Resultado del reentrenamiento de la detección en imágenes con TensorFlow	62
II.3. Pruebas de detección de fresas en imágenes con TensorFlow	63
II.4. Modelo <i>ssd_mobilenet_v2_320x320_coco17_tpu-8</i>	64
II.5. Modelo <i>efficientdet_d4_coco17_tpu-32</i>	64
II.6. Modelo <i>faster_rcnn_resnet50_v1_640x640_coco17_tpu-8</i>	65
II.7. Detección de fresas en webcam con TensorFlow con modelos no preentrenados (<i>ssd mobilenet v2 320x320</i>)	66
II.8. Gráficas de la confianza de detección obtenida en las pruebas según la luminosidad para el modelo <i>ssd mobilenet v2</i>	67

II.9. Gráficas de la media de los porcentajes de confianza obtenidos en las pruebas de detección según la luminosidad para el modelo ssd mobilenet v2	68
II.10.Detección de fresas en Jupyter Notebook	68
II.11.Detección con Pytorch	71
II.12.Etiquetado de las imágenes con labelImg	72
II.13.Calibración de la cámara C270 de Logitech	73
II.14.Medición del ángulo de rotación de la cámara mediante la aplicación de ERGONAUTAS RULER	74
II.15.Primeras pruebas de detección con PyTorch y Python	75
II.16.Primeras pruebas de la estimación de las coordenadas y la distancia de la detección a la cámara	77
II.17.Geometría basada en el modelo de cámara estenopeica	78
II.18.Pruebas para determinar la configuración del sistema de coordenadas de la cámara	79
II.19.Esquema de la rotación de la cámara	81
II.20.Detección y cálculo de las coordenadas y la distancia a la detección . .	82
II.21.Detección múltiple simultánea de post-it	83
II.22.Representación de las detecciones con OpenGL	84
II.23.Detección de fresas e integración con el sistema de cálculo de coordenadas y distancias	85
II.24.Proyección con OpenGL de las detecciones y el campo visual de la cámara	89
II.25.Representación de los sistemas de coordenadas que se había supuesto en un principio y del real obtenido	91
II.26.Representación del montaje y los sistemas de coordenadas obtenidos para las pruebas con la cámara perpendicular al plano de la mesa . . .	92
II.27.Representación básica de un programa de un sistema de visión externo	93
II.28.Programa recibir_cadena_socket.urp	94

II.29.Programa prueba_visionsimple.urp	95
II.30.Programa xmlrpc_example.urp	98
II.31.Pruebas funcionales con el programa pinhole.py y el robot real	99
II.32.Primeras pruebas detección de fresas y envío de las posiciones al UR	100
II.33.Disposición de las detecciones para un plano horizontal con un UR3e	101
II.34.Ejecución del programa en la terminal para un plano horizontal con un UR3e	102
II.35.Disposición de las detecciones para un plano vertical con un UR3e	103

Listado de códigos

5.1.	Fragmento del código que permite que el modelo de <i>deep learning</i> detecte las fresas maduras y genere sus posiciones en píxeles	43
5.2.	Fragmento del código que realiza la retroproyección desde 2D a 3D . . .	43
5.3.	Función <code>getInterseccionZ()</code>	44
5.4.	Envío de la última posición tridimensional detectada al robot mediante XML-RPC	44
II.1.	Función <code>positions_are_similar()</code>	75
II.2.	Fragmento del código donde se comparan posiciones para evitar que se dupliquen	76

Listado de ecuaciones

5.1.	Relación de fórmulas del modelo de cámara estenopeico	38
5.2.	Matriz de parámetros intrínsecos de la cámara	38
5.3.	Matrices de rotación $R(\theta)$ según el eje de rotación	38
5.4.	Matriz de traslación de la cámara en coordenadas tridimensionales . . .	39
5.5.	Descomposición e inversión parcial del modelo de cámara pinhole para estimar las coordenadas tridimensionales de un punto conocido en imagen bajo la hipótesis suelo $Z = Z_0$	39
5.6.	Fórmula del cálculo de la distancia de la cámara a las detecciones . . .	40
II.1.	Equivalencia entre las coordenadas supuestas y obtenidas	91

Índice de cuadros

5.1. Definición de los parámetros de posición y orientación de la cámara pinhole	37
II.1. Distribución de las imágenes utilizadas para el entrenamiento del modelo	61
II.2. Comparación entre coordenadas reales y obtenidas (en mm)	78
II.3. Resultados del programa pinhole.py con valores de Z positivos	80
II.4. Resultados del programa pinhole.py con valores de Z negativos	80
II.5. Resultados del programa pinhole.py con el valor ajustado de rotación de la cámara	81
II.6. Resultados del programa xmlrpc_deteccionfresas.py con la cámara situada a 145 mm de la mesa y la cámara rotada 59 grados	86
II.7. Resultados del programa xmlrpc_deteccionfresas.py con la cámara situada a 125 mm de la mesa y la cámara rotada 59 grados	86
II.8. Resultados del programa xmlrpc_deteccionfresas.py con la cámara situada a 225 mm de la mesa y la cámara rotada 69 grados	87
II.9. Resultados del programa xmlrpc_deteccionfresas.py con la cámara situada a 180 mm de la mesa y la cámara rotada 58 grados	88
II.10. Resultados del programa xmlrpc_deteccionfresas.py con la cámara situada a 225 mm de la mesa y la cámara perpendicular al plano	90
II.11. Resultados del programa xmlrpc_deteccionfresas.py con la cámara situada a 343 mm de la mesa y la cámara perpendicular al plano	92

Capítulo 1

Introducción

Desde sus inicios, la robótica ha proporcionado un sinfín de posibilidades y alternativas ante problemas que anteriormente carecían de las soluciones adecuadas, pero, ¿qué es realmente la robótica?

Se podría definir robótica como el proceso mediante el cual una máquina intercambia energía e información con su entorno, con el propósito de alcanzar una serie de objetivos específicos. Este campo tecnológico en expansión es el resultado de décadas de colaboración continua entre biólogos, informáticos e ingenieros [Koditschek, 2021]. Dada esta multidisciplina, la robótica abarca una amplia gama de aplicaciones, desde la industria hasta la medicina, pasando por la exploración espacial, la domótica o la conducción autónoma, entre otras. Es un campo en constante evolución, impulsado por la búsqueda de soluciones innovadoras para mejorar la calidad de vida y permitir superar desafíos de manera más eficiente y segura.

La industria agrícola no es una excepción, ya que ha contemplado históricamente tareas que requieren una dedicación laboral considerable. No obstante, gracias a la robótica y a los sistemas de visión artificial, surge la oportunidad de transformar una serie de procesos, como puede ser la recolección de cultivos a través de la detección automatizada.

En las siguientes secciones se describen brevemente algunas de las aplicaciones más importantes de la robótica en la sociedad actual, así como los distintos conceptos en los cuales se basa la investigación y el desarrollo llevado a cabo para la realización de este Trabajo Fin de Grado.

1.1. Los robots y la robótica

Según la *Federación Internacional de Robots* (IFR) se define robot según el vocabulario establecido por la *International Organization for Standardization* (ISO), y esto es como *mecanismo accionado programado con cierto grado de autonomía para realizar tareas de locomoción, manipulación o posicionamiento* [ISO/TC299, 2021].

El término robot fue utilizado por primera vez por Karel Čapek en su obra de teatro *Rossum's Universal Robots*, publicada en 1920. Esta palabra viene del vocablo checo *robota* que significa trabajo, en el sentido de la obligatoriedad, entendido como servidumbre, trabajo forzado o esclavitud [Sánchez Martín et al., 2007a]. Aunque esta definición es un punto de partida, es cierto que es posible diferir en aspectos como si un robot debe controlarse automáticamente o podría ser autónomo o si un robot debe ser reprogramable. A un nivel más amplio, cualquier máquina que pueda utilizarse para llevar a cabo acciones o tareas complejas de forma automática puede considerarse un robot [Raj and Seamans, 2019].

Isaac Asimov (1920-1992) utilizó por primera vez el término robótica y postuló las tres leyes de la robótica en su libro *I Robot*, publicado en 1950, coincidiendo con el apogeo de la robótica moderna. Asimov consideró necesario añadir una cuarta ley, antepuesta a las demás, la número cero, que afirma que un robot no debe actuar simplemente para satisfacer intereses individuales, sino que sus acciones deben preservar el beneficio común de toda la humanidad [Sánchez Martín et al., 2007b].

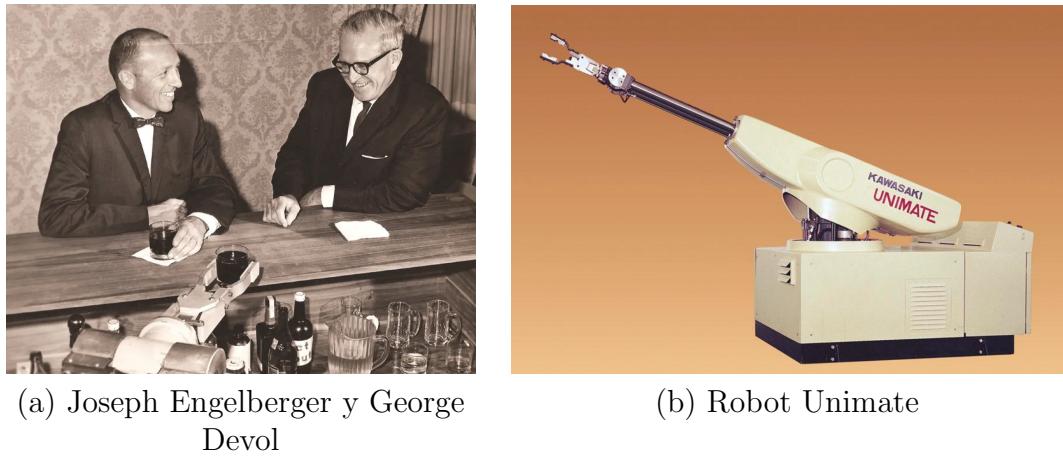
Partiendo de todos estos avances y del interés por automatizar las tareas de producción, la robótica va adquiriendo un gran desarrollo [Sánchez Martín et al., 2007b]. Es debido a este desarrollo que, atendiendo al propósito y al contexto en el que se utilicen estos robots, se fueron creando varios grupos en función de los que clasificarlos. Estos tres grandes grupos fueron, en función de una serie de criterios generales: robots industriales, robots de servicio y robots en medicina.

1.1.1. Robots industriales

Se define robot industrial como un manipulador polivalente, reprogramable y controlado automáticamente, programable en tres o más ejes, que puede ser fijo o móvil para su uso en aplicaciones de automatización industrial [ISO/TC299, 2021].

La evolución de los robots industriales puede subdividirse en cuatro categorías: las tres primeras abarcan el período comprendido entre los años cincuenta y finales de los noventa, mientras que la cuarta generación abarca desde 2000 hasta nuestros días [Gasparetto and Scalera, 2019].

La primera generación, o primeros manipuladores (1950-1967), eran básicamente máquinas programables que no tenían comunicación con el entorno externo y con algoritmos de control sencillos (punto a punto). En cuanto al hardware, contaban con equipos de baja tecnología, sin servo-controladores. Sin embargo, en 1954, George Devol y Joseph Engelberger formaron la empresa Unimation, empresa que desarrollaría Unimate (ver en Figura 1.1), considerado el primer robot industrial de la historia, fabricado en 1961 [Zamalloa et al., 2017].



(a) Joseph Engelberger y George Devol

(b) Robot Unimate

Figura 1.1: Primer robot industrial

La segunda generación, o robots sensorizados (1968-1977), eran máquinas programables básicas con posibilidades limitadas de comportamiento autoadaptativo y capacidades elementales para reconocer el entorno externo, poseían sistemas sensoriales avanzados y eran robots de gran volumen que se utilizaban principalmente en automoción [Zamalloa et al., 2017].

En 1968, en el Stanford Artificial Intelligence Laboratory (SAIL) se confecciona el WAVE, el primer lenguaje de programación para robots. En 1969, Víctor Scheinman, un estudiante de ingeniería mecánica de la Universidad de Standford, diseñó y construyó el primer prototipo de brazo robótico (Figura 1.2), cuya cinemática inversa podía resolverse de manera analíticamente cerrada, permitiendo una rápida ejecución de la trayectoria [Gasparetto and Scalera, 2019].

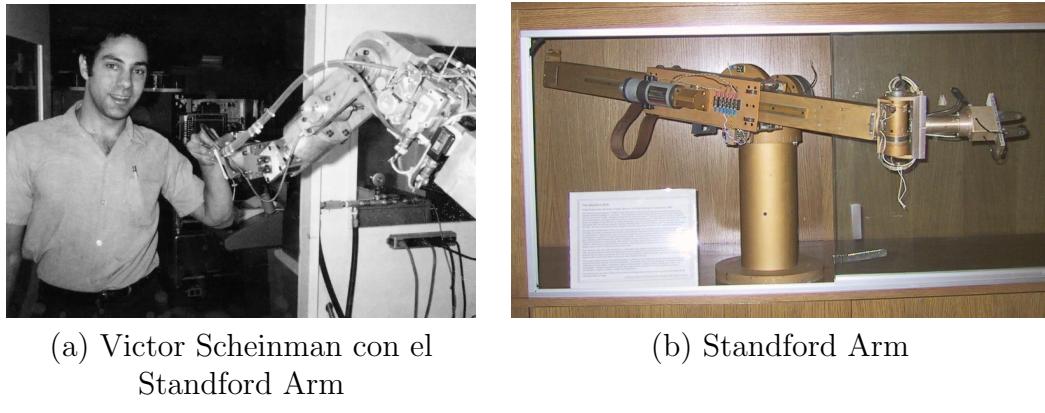


Figura 1.2: Stanford Arm

La tercera generación, o robots industriales (1978-1999), disponían de controladores específicos (ordenadores), siendo un punto clave en la caracterización de esta generación, además del surgimiento de nuevos lenguajes de programación para el control de los robots, la posibilidad de reprogramarlos y la inclusión parcial de la visión artificial [Zamalloa et al., 2017].

A partir del año 2000, aparece la cuarta generación o robots inteligentes (2000-Actualidad), que se caracteriza por la inclusión de capacidades informáticas avanzadas, ya que los ordenadores no sólo trabajan con datos, si no también pueden realizar razonamientos lógicos y aprender, puesto que la Inteligencia Artificial comienza a ser incluida parcial y experimentalmente en estos robots. Los sensores son más sofisticados, y envían información al controlador y la analizan mediante estrategias de control complejas para que el robot pueda basar sus acciones en información sólida y fiable. Es en esta generación cuando se introducen los robots colaborativos [Zamalloa et al., 2017].

A principios de 2007, dos estudiantes de doctorado de la Universidad de Standford, Keenan Wyrobek y Eric Bergerlas, pusieron las primeras piezas de lo que eventualmente se convertiría en ROS (Robot Operating System). Uno de los preceptos principales que se tuvo en cuenta para la creación de este sistema operativo para robots fue el de crear un sistema que permitiese al máximo posible la reutilización de código, dando soporte a distintos tipos de robots y de aplicaciones.

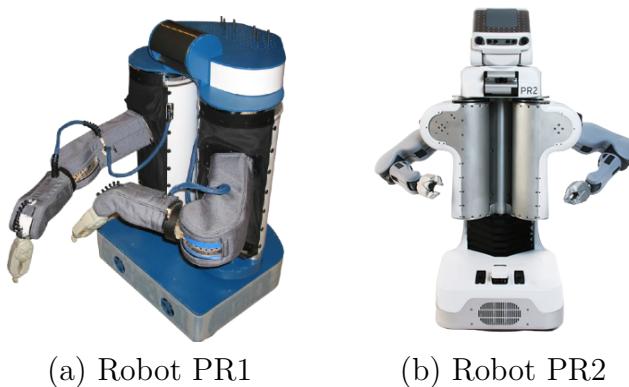


Figura 1.3: Robots utilizados para el desarrollo de ROS

En el año 2008 se entrega el primer robot colaborativo o cobot, el UR5 de Universal Robots¹ (Figura 1.4), considerado como uno de los logros tecnológicos más significativos de la década en la comunidad robótica.[Cusano, 2022]



Figura 1.4: UR5 con su controladora

Más tarde, en 2018, Universal Robots presenta los robots colaborativos e-Series, que se pueden ver en la Figura 4.3, que incluían avances tecnológicos que permitían un desarrollo más rápido para una mayor variedad de aplicaciones, ofrecía una programación más sencilla y seguía las normas de seguridad ISO más actuales y recientes².

El futuro de la robótica industrial promete seguir transformando radicalmente nuestros métodos de trabajo y producción, abriendo así nuevas oportunidades y desafiando constantemente los límites de lo que podemos lograr en la automatización industrial, así como en los otros dos grandes grupos de la robótica, como la robótica de servicio y la robótica médica.

¹<https://www.universal-robots.com/es/>

²<https://www.universal-robots.com/es/acerca-de-universal-robots/nuestra-historia/>

1.1.2. Robots de servicio

Se define robot de servicio como un robot que realiza tareas útiles para las personas o los equipos, incluyendo en esta la manipulación o el servicio de artículos, el transporte, el apoyo físico, la orientación o información, el aseo personal, la cocina y la manipulación de alimentos y la limpieza en el ámbito personal; y la inspección, vigilancia, manipulación de objetos, transporte de personas, orientación o información, cocina y manipulación de alimentos y limpieza en el ámbito profesional [ISO/TC299, 2021].

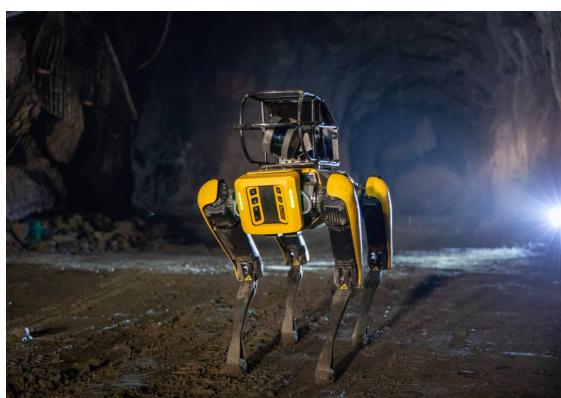
En la práctica, las actuales y potenciales aplicaciones no industriales de los robots son tan variadas y diferentes que se dificulta su catalogación [Barrientos, 2002]; sin embargo, tratando de establecer una división de los robots de servicio, la norma ISO 8373:2012, así como la Federación Internacional de Robótica o IFR, propuso clasificarlos en diferentes categorías según su función y aplicación en robots para uso doméstico y personal y robots de servicio destinados a un uso profesional [Gonzalez-Aguirre et al., 2021], siendo las aplicaciones más importantes las siguientes: limpieza, inspección y mantenimiento, educación, logística y entretenimiento (Figura 1.5).



(a) Roomba de iRobot



(b) AMRs de MiR



(c) Spot de Boston Dynamics



(d) Turtlebot 4

Figura 1.5: Robots de servicio

Estos robots de servicio, y más concretamente los dedicados a la logística, desempeñan un papel fundamental en la optimización de la cadena de suministro, mejorando la eficiencia y la precisión en la manipulación de productos. Un ejemplo del posible uso de estos robots en el sector agrícola, es la primera granja vertical de interior del mundo en Estados Unidos, que producirá 18 millones de kilogramos de fresas al año, marcando un hito en la agricultura moderna, y demostrando que la automatización e integración de la robótica en este tipo de granjas verticales puede transformar la producción y recolección de alimentos a gran escala [EcoInventos.com, 2024].

La robótica de servicio representa una revolución en la asistencia y el apoyo a diversas industrias, desde la logística hasta la atención al cliente en el comercio minorista. Sin embargo, su impacto va más allá, extendiéndose hasta la atención médica. En este contexto, la robótica médica emerge como una vanguardia tecnológica que fusiona la innovación robótica con la medicina moderna para ofrecer soluciones innovadoras en diagnóstico, tratamiento y rehabilitación, demostrando su potencial para revolucionar la forma en que brindamos y recibimos atención médica.

1.1.3. Robots en medicina

Se define *robot médico* como aquellos dispositivos electromecánicos que desempeñan parcial o totalmente algunas funciones de los seres humanos o de sus órganos al resolver problemas médicos, ayudando a mejorar la asistencia al paciente y los resultados, a la vez que aumenta la eficiencia operativa [Kraevsky and Rogatkin, 2010].

Los robots en medicina se desarrollaron por primera vez hace poco más de tres décadas para permitir a los cirujanos operar a sus pacientes a distancia o con mayor precisión. A finales de los años noventa, había 2 tipos de telemanipuladores quirúrgicos aprobados por la Administración de Alimentos y Medicamentos de los Estados Unidos (FDA): el Zeus y el Da Vinci (Figura 1.6(a)), introducido en 1998-1999, que permitía aumentar la precisión de las cirugías mínimamente invasivas [Romero-Tamarit et al., 2020].

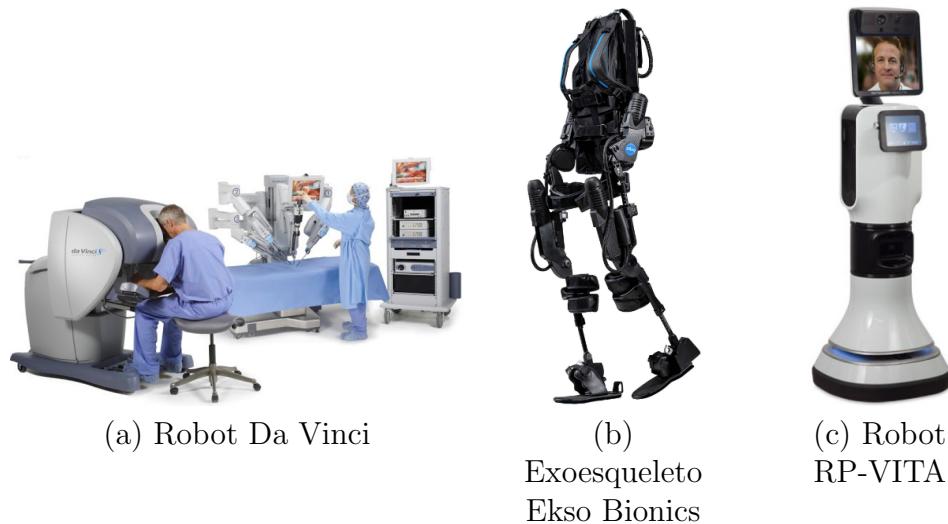


Figura 1.6: Robots en medicina

Las primeras aplicaciones fueron en los campos de neurocirugía y cirugía ortopédica, siendo la cirugía donde mayor impacto han tenido los robots en la medicina, sin embargo, se están investigando otras áreas de la medicina, como los robots para realizar rehabilitación física con pacientes con discapacidades motores, como el exoesqueleto Ekso Bionics (Figura 1.6(b)), robots de telepresencia para la interacción del paciente con el personal sanitario externo, como el robot RP-VITA (Figura 1.6(c)), automatización de farmacias, robots para desinfectar clínicas, etc. [Dupont et al., 2021]

El rápido crecimiento de la robótica médica se debe a una combinación de mejoras tecnológicas (motores, materiales y teoría de control), los avances en imagen médica (mayor resolución, resonancia magnética y ecografía 3D) y una mayor aceptación por cirujanos y pacientes de los procedimientos laparoscópicos y la asistencia robótica [Beasley, 2012], convirtiéndose en un campo interdisciplinario que abarca desde cirugía asistida por robots hasta sistemas de diagnóstico de vanguardia.

A continuación, explicaremos el impacto que la inteligencia y la visión artificial están teniendo en la robótica, y las capacidades y oportunidades que estas presentan en una inmensa variedad de aplicaciones.

1.2. Inteligencia Artificial

La Inteligencia Artificial (IA) es un área multidisciplinaria de la ciencia donde se realizan sistemas que tratan de hacer tareas y resolver problemas como lo hace un humano; así mismo, trata de simular de manera artificial las formas de pensamiento y de trabajar del cerebro para la toma de decisiones [Ponce Gallegos et al., 2014].

El origen del concepto y de los criterios de desarrollo de la IA se remontan al año 1936, con el matemático inglés Alan Turing, quien definió qué era una máquina abstracta [Hardy, 2001], definición que sirvió de base para la noción de algoritmo, y quien intuyó la importancia que jugaría el aprendizaje automático en el desarrollo de la IA al afirmar que, en lugar de intentar emular mediante una máquina la mente de un adulto, sería más factible intentar emular la mente de un niño y luego someter a la máquina a un proceso de aprendizaje que diera lugar a un desarrollo cognitivo de dicha mente hasta alcanzar el equivalente de una mente adulta, lo que actualmente se conoce como robótica de desarrollo [González and de Mántaras Badia, 2017]. Por otro lado, sería John McCarthy a quien se debe el apelativo Inteligencia Artificial, ya que organizó una conferencia en el Darmouth College (Estados Unidos) en agosto de 1956, para discutir sobre la posibilidad de construir máquinas inteligentes. Como resultado de esta reunión, se establecieron las primeras bases sobre la inteligencia de los computadores [Ponce Gallegos et al., 2014].

Una de las ramas más fascinantes y prometedoras de la inteligencia artificial es la visión artificial, que busca dotar a las máquinas de la capacidad de interpretar y comprender el mundo visual que les rodea. La siguiente sección se centra en la importancia de la IA y su intersección con la visión artificial, explorando cómo estas disciplinas se fusionan para mejorar la percepción y la comprensión de imágenes y videos.

1.3. Visión Artificial

La visión artificial se define como la ciencia de programar un ordenador para procesar imágenes o videos e incluso entenderlos [Culjak et al., 2012].

En [Bradski and Kaehler, 2008] se explica cómo es la transformación de datos desde un fotograma o video cámara hasta lo que puede ser una decisión o una nueva representación [Alvear-Puertas et al., 2017]. Para ello, la imagen percibida pasa por los procesos de obtención, caracterización e interpretación de información de imágenes; y estos pro-

cesos pueden ser subdivididos a su vez en [García Santillán and Caranqui Sánchez, 2015], [Martínez Madruga, 2022]: captura, pre-procesamiento, segmentación, descripción, reconocimiento (clasificación) e interpretación.

Estas fases son las empleadas bajo el paradigma de lo que se conoce como Visión Artificial Clásica, enfocada a la utilización de algoritmos específicos para procesar imágenes y reconocer en ellas características básicas [Martínez Madruga, 2022]. Sin embargo, para mejorar aún más la eficacia de los sistemas de visión artificial, se recurre al aprendizaje automático o *machine learning* (ML). A continuación, profundizaremos en el papel del *machine learning* en la visión artificial y su importancia en la creación de sistemas inteligentes de procesamiento de imágenes.

1.4. Machine Learning

El Machine Learning (Aprendizaje Automático) es una rama en evolución de la Inteligencia Artificial que se encarga de generar algoritmos que tienen la capacidad de aprender del entorno circundante y no tener que programarlos de manera explícita, teniendo en cuenta todos los escenarios posibles, a partir de la construcción de modelos analíticos [Sandoval Serrano et al., 2018].

Dependiendo de la tarea de aprendizaje, existen varias clases de algoritmos de ML, cada uno de ellos con múltiples especificaciones y variantes, que pueden englobarse o bien en el Shallow Machine Learning (aprendizaje superficial), que se centra en algoritmos más simples para realizar tareas específicas, o en Deep Learning (aprendizaje profundo), que utiliza la construcción y entrenamiento de Redes Neuronales Artificiales (RNA), un tipo de modelo inspirado en la estructura y funcionamiento del cerebro humano, tal y como se puede apreciar en el diagrama de la Figura 1.7 [Janiesch et al., 2021].



Figura 1.7: Diagrama de Venn de la relación entre distintas áreas de la IA

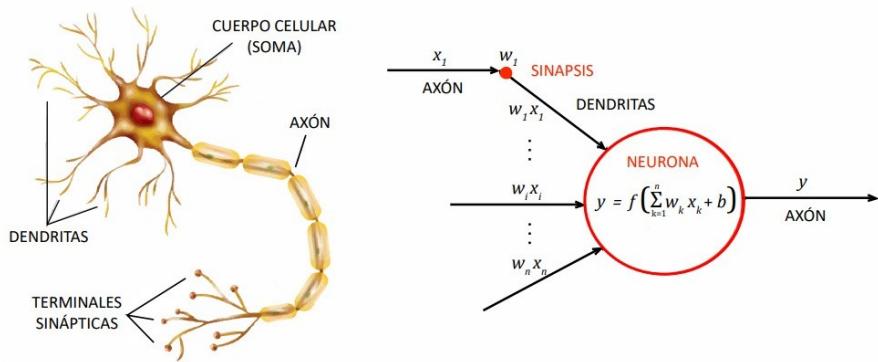
El Machine Learning, abarca desde enfoques más superficiales hasta técnicas más avanzadas, tal y como se ha podido observar; sin embargo, es el Deep Learning lo que realmente potencia la capacidad de las máquinas para aprender y generalizar patrones complejos de manera excepcional. En la próxima sección, se hablará sobre el Deep Learning, centrándose en las RNA y en cómo estas posibilitan abordar tareas más complejas, como el reconocimiento de patrones en imágenes o el procesamiento de lenguaje natural.

1.5. Deep Learning

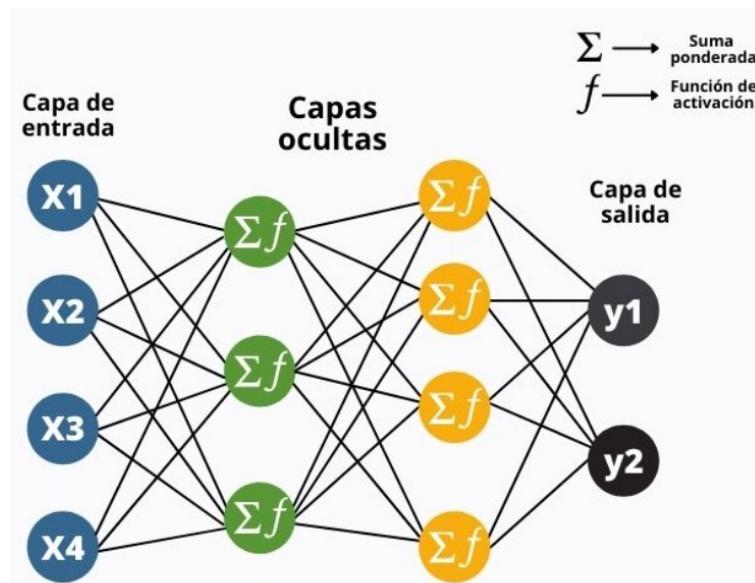
El Deep Learning o aprendizaje profundo, constituye una rama de la IA, incluida dentro del Machine Learning, cuyos modelos computacionales se inspiran en el funcionamiento del cerebro humano y se diseñan con el propósito de adquirir conocimientos y llevar a cabo tareas específicas mediante el procesamiento de datos.

Estas Redes Neuronales Artificiales (RNA) o Artificial Neural Networks (ANN) en inglés, están inspiradas en las redes neuronales biológicas del cerebro humano, tal y como muestra la Figura 1.8(a), presentando características del mismo, ya que estas aprenden de la experiencia, generalizan de ejemplos previos a ejemplos nuevos, y abstraen las características principales de una serie de datos. En las RNA, la unidad

análoga a la neurona biológica es el elemento procesador, PE (Process Element). Un PE tiene varias entradas y las combina, normalmente, con una suma básica. La suma de las entradas es modificada por una función de transferencia y el valor de la salida de esta función de transferencia se pasa directamente a la salida del elemento procesador. Existen dos capas con conexiones con el mundo exterior, una capa de entrada o *buffer* de entrada, donde se presentan los datos a la red, y una capa o *buffer* de salida que mantiene la respuesta de la red a una entrada, mientras que el resto de las capas reciben el nombre de capas ocultas [Basogain, 2008], Figura 1.8(b). Distintos pesos dan como resultado diferentes respuestas a una entrada. De esta manera, se puede decir que el aprendizaje es el ajuste de los pesos en respuesta a un estímulo [Dinamarca, 2018].



(a) Modelo biológico de una neurona genérica y modelo matemático



(b) Arquitectura de una red neuronal

Figura 1.8: Redes Neuronales

En este capítulo se ha introducido el nacimiento y la historia de los robots y la robótica tal y como los conocemos hoy en día, dentro de cuya rama encontramos uno de los tres grandes grupos en los cuales puede dividirse esta, la Robótica de Servicio, y para la que la Inteligencia Artificial, y más concretamente el campo de la Visión Artificial junto con el del Deep Learning, juegan un papel fundamental en el desarrollo de nuevas aplicaciones.

En este proyecto se presenta un sistema que, mediante Visión Artificial y Machine Learning, es capaz de reconocer la maduración de frutos, más concretamente de fresas, con el objetivo de poder ayudar así a mejorar su proceso de recolección en un huerto vertical, gracias al algoritmo desarrollado para esto y su integración con un brazo robótico de la marca Universal Robots, que se encargará de llevar a cabo este proceso.

En los siguientes capítulos se hará una breve exposición del estado del arte en el que se enmarca este trabajo, se detallarán los objetivos del mismo, delineando claramente las metas; se expondrá la plataforma de desarrollo, detallando las herramientas seleccionadas para la elaboración del proyecto; se presentará el diseño y la arquitectura del proyecto; y, finalmente, llegaremos a las conclusiones, donde tendrá lugar una breve recopilación de información sobre los resultados obtenidos y las posibles direcciones futuras.

Capítulo 2

Estado del arte

En el presente capítulo se van a describir algunos de los prototipos y soluciones más destacables aplicadas a la detección y recolección de fresas usando inteligencia artificial y técnicas robóticas.

En los sistemas de producción de cultivos hay operaciones de campo que requieren mucha mano de obra, ya sea por su complejidad, o por el hecho de que están relacionadas con una interacción sensible entre plantas y productos comestibles, o por la repetitividad que requieren a lo largo de un ciclo de producción de cultivos. Estos son los factores clave para el desarrollo de robots agrícola (Figura 2.1), tal y como se relata en el artículo [Fountas et al., 2020], donde se realiza una revisión sistemática de la bibliografía existente sobre la investigación y la robótica agrícola comercial utilizada en las operaciones de los campos de cultivo en función de las principales operaciones de campo, siendo estas: deshierbe, siembra, detección de enfermedades e insectos, exploración de cultivos (seguimiento de plantas y fenotipado), pulverización o rociado, cosecha, robots de gestión de plantas y sistemas robóticos polivalentes.



Figura 2.1: Representación de varios tipos de robots agrícolas

Entre estas operaciones se escogió centrarse en la recolección, que es una de las tareas más laboriosas y repetitivas, a la vez que forma parte de todos los ciclos de producción en la agricultura, existiendo dos tipos de cosechadoras robotizadas: a granel (se recogen todas las frutas/verduras) y selectivas (sólo se recogen las frutas maduras/listas para ser cosechadas) [Fountas et al., 2020], dentro de las cuales se engloba este trabajo final de grado.

La mayoría de los robots de recolección se centran en las fresas, un cultivo de alto valor, que sufre un alto coste de producción debido principalmente al coste de la mano de obra, sobre todo durante la recolección.

Sin embargo, no basta con tener una alta velocidad de recogida, ya que también es importante una alta tasa de recogida. Para evaluar el rendimiento de los robots en la recolección, se ha mostrado un rango en las tasas de éxito de recolección, desde el 0 % hasta el 64 %, para varias categorías, tal y como se muestra en la Figura 2.2, mereciendo la pena mencionar que, para la recolección de fresas, parece haber un equilibrio entre la velocidad y la tasa de recolección, ya que las velocidades de recolección rápidas van acompañadas de tasas de recolección bajas y viceversa [Fountas et al., 2020].

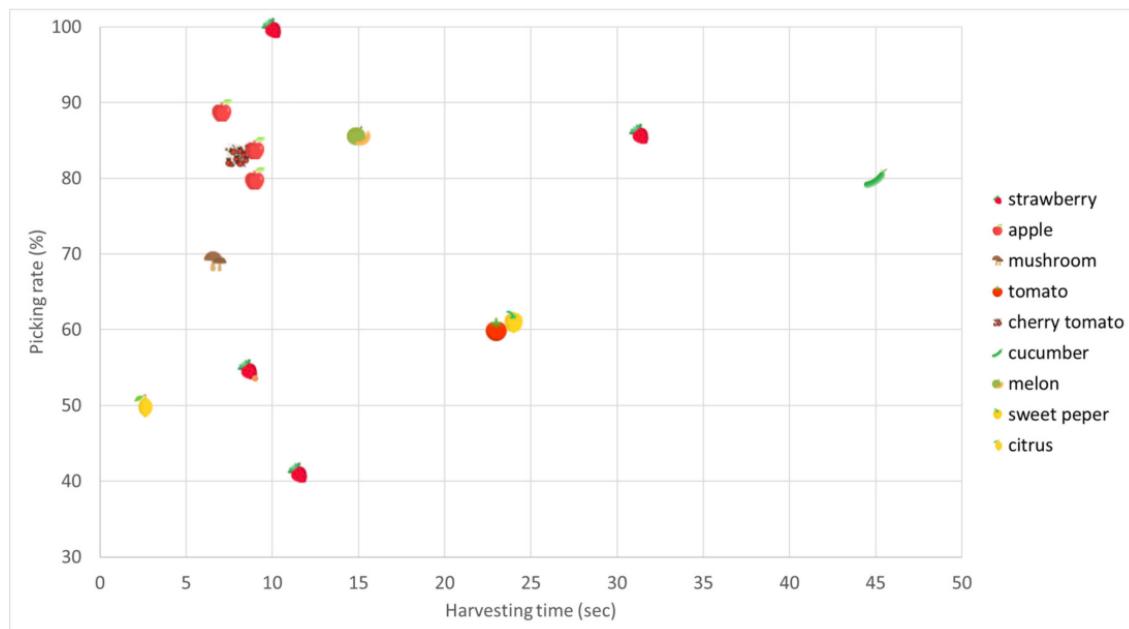


Figura 2.2: Ilustración global del rendimiento general de los robots revisados

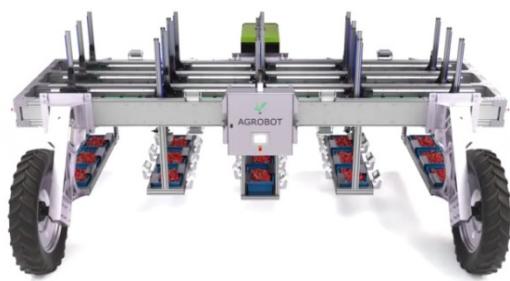
En 2009, después de más de 30 años de comercializar fresas en la provincia de Huelva, y de algunos intentos infructuosos, la empresa AGROBOT⁸, con sede en el Centro de Innovación y Tecnología que la Consejería de Innovación tiene en Lepe (Huelva), consiguió poner en marcha un prototipo que identificaba los frutos maduros y los recogía sin dañarlos, siendo capaz de clasificarlas y colocarlas en los envases que recorren las cintas transportadoras, gracias a un grupo de ingenieros liderado por Juan Bravo [Cabanillas, 2009].

Tras la presentación del prototipo y las primeras pruebas con éxito, la empresa onubense desarrolló el prototipo final, el Agrobot SW 6010, una cosechadora de fresas que es capaz de recoger de la mata solo la fruta que está madura mediante inteligencia artificial con 30 brazos robóticos que incorporan una cámara con visión artificial que detecta el grado de madurez de la fruta en tiempo real, y si la fresa cumple con los parámetros marcados de tamaño, grosor y color.

Este modelo se volvería a mejorar para obtener el Agrobot E-Series⁹ (Figura 2.3), permitiendo adaptarse a cualquier configuración agrícola, y que, fabricado en acero inoxidable y aluminio de calidad militar, puede funcionar de forma robusta con un alto grado de precisión, ya que los sensores de profundidad infrarrojos y en color integrados de corto alcance a bordo, ayudan a evaluar el grado de madurez de la fruta, incorporando a su vez, sensores LiDAR que se encargan de la seguridad de los trabajadores del campo circundantes.



(a) Agrobot E-Series



(b) Representación 3D del Agrobot E-Series

Figura 2.3: Agrobot

⁸<https://www.agrobot.com/?lang=es>

⁹<https://www.agrobot.com/e-series>

La empresa belga de I+D agrícola Octinion¹⁰ desarrolló un prototipo de robot recolector de fresas en 2017, que recoge los frutos de forma totalmente autónoma basándose en el método de cultivo habitual (sobre mesa), con el fin de resolver el obstáculo emergente de la agricultura occidental: la falta de mano de obra asequible que pone en peligro la sostenibilidad y conservación del negocio [De Preter et al., 2018].

Este robot, cuyo diseño está representado en la Figura 2.4, está formado por un vehículo eléctrico consistente en una plataforma eléctrica con una batería recargable; un sistema de localización constituido por codificadores de rueda, un giroscopio y un sistema de posicionamiento en interiores de banda ultra ancha (UWB); tres cámaras RGB utilizadas para la detección de las fresas por cámara mediante visión artificial, un brazo robótico diseñado a medida, la pinza que se acopla al extremo del brazo robótico y agarra con sus dedos la fresa detectada, un módulo de gestión o manipulación logística consistente en varias cestas que se transportan en la plataforma eléctrica y que están preparadas por el robot para que estén inmediatamente listas para el envasado final y el transporte; y un módulo de control de calidad que clasifica las fresas detectadas en función de su madurez, forma, tamaño y dulzor [De Preter et al., 2018].

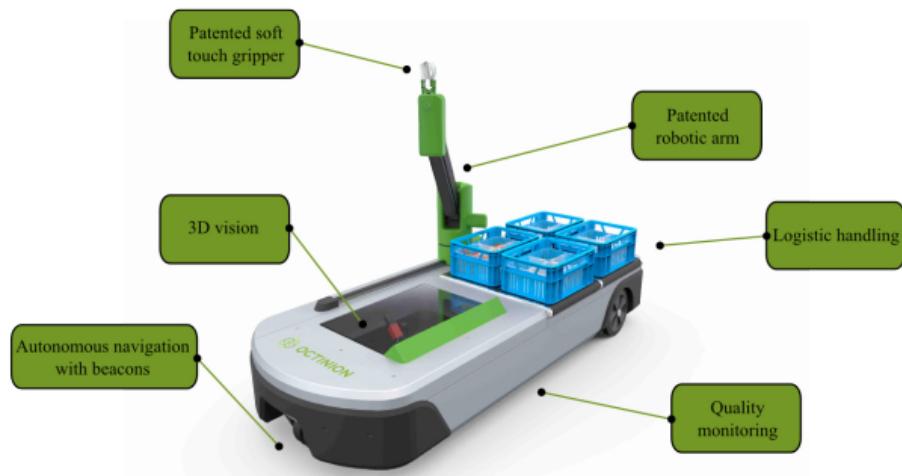


Figura 2.4: Diseño conceptual del robot de recogida con sus componentes

Todo esto le permite al robot recoger al menos el 70 % de las fresas maduras, siempre sin dañarlas, ya que sólo decide recoger la fruta si su acción no va a dañar otras fresas, siendo el tiempo necesario para desplazarse hasta la fresa, cogerla y depositarla en una cesta (caja en la que se colocan las fresas), siendo la calidad y velocidad de recolección comparables a las de un recolector humano ideal [De Preter et al., 2018].

¹⁰<http://octinion.com/products/agricultural-robotics/rubion>

El sistema Dogtooth¹¹ de Cambridge (Reino Unido) (Figura 2.5) fue desarrollado para mejorar las operaciones de recogida de frutos rojos siendo capaz de navegar por hileras de fresas y frambuesas, detectar y localizar las maduras, así como recoger y comprobar la fruta antes de colocarla en un cesto, permite a las empresas de recolección de fruta sustituir el método de recogida manual y ahorrar tiempo. Sin embargo, al cortar el tallo produce una pequeña herida, que permite que las enfermedades entren fácilmente en la planta, y la parte restante del pedúnculo puede magullar otras fresas durante el transporte, cuando las frutas se agitan en sus cajas. Así mismo, Dogtooth parte de un robot industrial caro cuya cinemática no está optimizada para la recogida de fresas, suponiendo un inconveniente frente a sus competidores que, a pesar de encontrarse en fase de prototipo, sus conceptos exigen cambios drásticos en la infraestructura ya que las fresas recolectadas no cumplen las especificaciones exigidas por el mercado.



(a) Robot Dogtooth



(b) Brazo robótico del Dogtooth

Figura 2.5: Robot agrícola Dogtooth

¹¹<https://dogtooth.tech/>

El sistema explicado en [Xiong et al., 2019], presenta el desarrollo y la evaluación de un robot para la recolección de fresas cultivadas en invernaderos (Figura 2.6). El robot encargado de realizar esta tarea está compuesto por una pinza montada en un brazo industrial que a su vez está montado en una base móvil junto con una cámara RGB-D. Este usa el sistema de visión que se basa en el umbral de color combinado con el cribado del área del objeto y el rango de profundidad para seleccionar las fresas maduras y alcanzables. La novedosa pinza está diseñada para apuntar a la fruta y no al tallo, por lo que sólo requiere la ubicación de la fruta para la recolección. Además, está equipada con sensores internos, por lo que la pinza puede detectar y corregir errores de posición, y es resistente a los errores de localización introducidos por el módulo de visión.

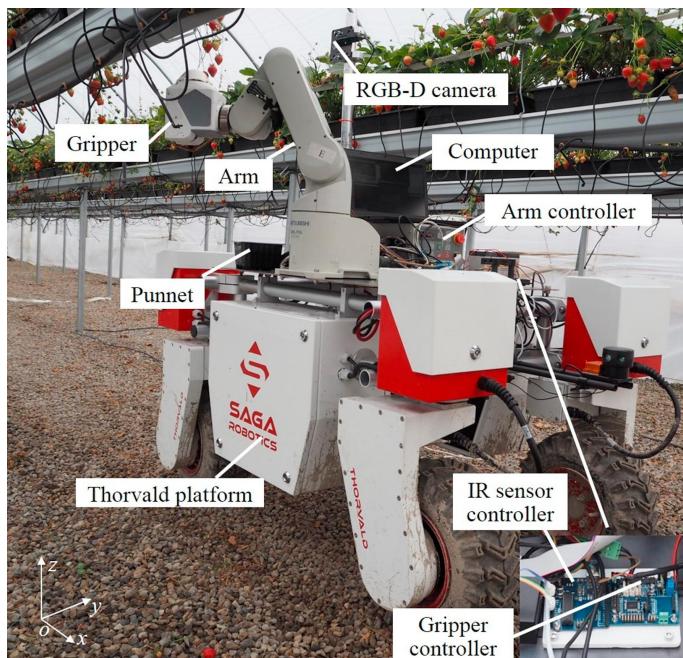


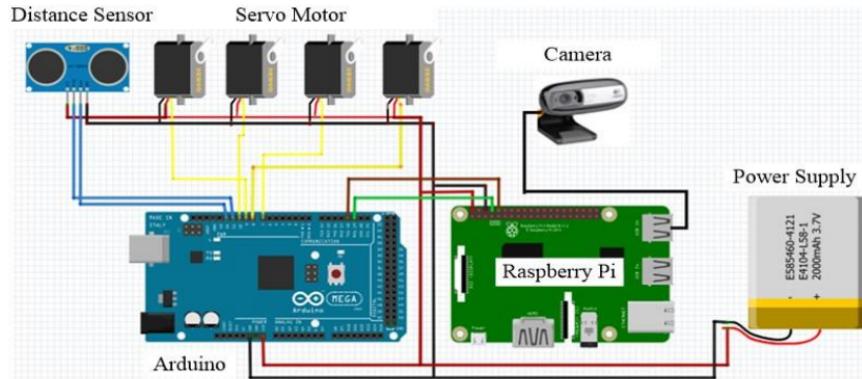
Figura 2.6: Montaje del hardware en una explotación de fresas

Con todo esto, los experimentos de campo muestran este descenso de la tasa de éxito, debido a los siguientes factores:

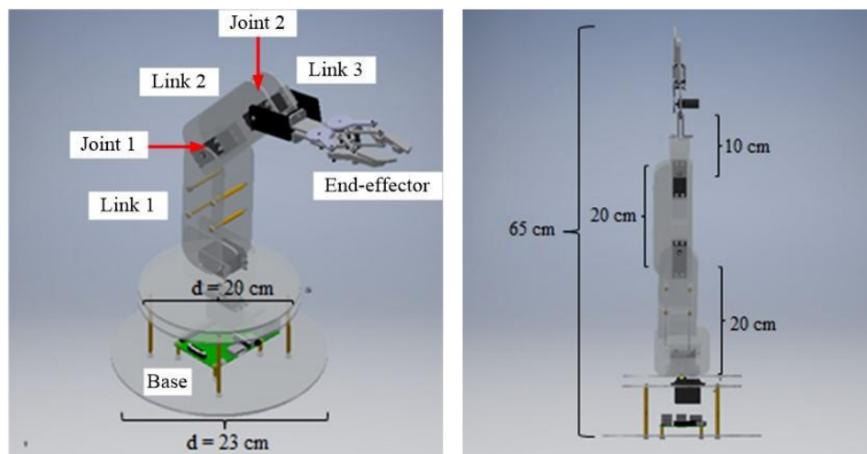
- Oclusión de las fresas, lo que deriva en una detección fallida y una no recolección de las mismas.
- Posibles detecciones duplicadas, debido a las agrupaciones de fresas que se tocan entre sí.

- Errores de localización para la que la pinza pueda coger la fresa, producidos por localizaciones imprecisas y/o fallos de segmentación del sistema.
- Perturbaciones que produce la pinza cuando se encuentran fresas por debajo del objetivo o dentro del área de búsqueda de la pinza, por lo que la pinza detecta las fresas molestas y las considera objetivos. También debido a los toques que pueda tener el brazo robótico durante el proceso de recolección con las plantas, ya que estos toques afectan a la ubicación de objetivos.
- Región de alcance del robot reducida, ya que el espacio de trabajo con el que cuenta el brazo robótico para llevar a cabo la tarea de recolección es limitado.
- Fallos de comunicación del brazo o la pinza.

En 2020, la Universidad Politécnica Negeri Sriwijaya de Indonesia, analizó el empleo de un robot recolector como proyecto piloto, tal y como se cuenta en el artículo [Oktarina et al., 2020], en el cual la fruta a recolectar serían tomates rojos y verdes en lugar de fresas. A pesar de la diferencia en el tipo de cultivo, el estudio resulta relevante para este trabajo debido a la similitud en los principios de detección y recolección de frutos, y al tratarse de un prototipo básico lo convierte en un punto de partida útil para proyectos de investigación enfocados en la automatización agrícola aplicada a fresas (Figura 2.7).



(a) Conexión eléctrica entre los componentes del robot recolector



(b) Diseño tridimensional del robot recolector

Figura 2.7: Robot recolector de tomates

En este capítulo se han revisado algunos de los avances más relevantes en el campo de la robótica agrícola, más concretamente en los sistemas aplicados a la recolección de fresas, donde se ha evidenciado cómo el uso de la inteligencia artificial y la visión artificial ha impulsado mejoras significativas tanto en la velocidad como en la precisión de los procesos de recolección. Sin embargo, aún queda trabajo por hacer para optimizar y mejorar la eficiencia y la viabilidad económica de estos sistemas.

Capítulo 3

Objetivos

Una vez presentado el contexto general en el cual se enmarca el presente trabajo de fin de grado, en este capítulo se describen los objetivos y requisitos de este, así como la metodología y el plan de trabajo llevados a cabo.

3.1. Descripción del problema

La necesidad de implementar soluciones tecnológicas que automaticen y optimicen las tareas de recolección incrementando la eficiencia en la recolección, mejorando la calidad del producto y disminuyendo los costes asociados, surge debido a la situación actual de la agricultura en la que, uno de los mayores desafíos que enfrenta es la recolección de frutas y hortalizas, problema que deriva de la escasa mano de obra disponible y el proceso manual que esto conlleva.

La solución propuesta en este trabajo busca ayudar a mejorar esta situación, proporcionando un robot de bajo coste y accesible a cualquier persona, que sirva para poder mejorar el proceso de reconocimiento por visión de la maduración de frutos, más concretamente fresas, para su posterior recolección. Por lo tanto, este proyecto pretende, como objetivo principal, utilizar un robot colaborativo que, gracias a su interfaz intuitiva sea accesible a cualquier persona y, junto con el sistema de detección elaborado con materiales de bajo coste, sea capaz de reconocer las fresas maduras de un sistema de cultivo agrícola vertical, para su posterior recolección por el brazo robótico, gracias a la comunicación establecida entre el sistema de visión y el robot.

Con el fin de alcanzar este objetivo principal, se han establecido los siguientes subobjetivos:

1. Investigar las soluciones actuales que cumplen con las características y objetivos establecidos.

2. Seleccionar la técnica de inteligencia artificial de reconocimiento de frutas y seleccionar los componentes hardware necesarios para desarrollar el sistema de visión de bajo coste más eficiente.
3. Optimizar la técnica escogida y adaptarla de tal manera que sea capaz de funcionar en nuestra plataforma. Al ser una técnica basada en Machine Learning, se deberá crear un dataset con imágenes de fresas y, por lo tanto, hacer un correcto tratamiento de los datos para conseguir un resultado preciso en el posterior entrenamiento.
4. Realizar el entrenamiento con varios algoritmos de Machine Learning de clasificación. Estudiar el rendimiento y precisión de cada uno de ellos a través de pruebas con el sistema de visión y fresas reales.
5. Seleccionar el protocolo de comunicación entre el sistema de visión y el robot y llevar a cabo pruebas; tanto simuladas, a través del simulador que facilita el fabricante del robot, como reales, para establecer esta comunicación.
6. Dar soporte software al robot mediante un sistema de reconocimiento de fresas, que guarde las posiciones y la distancia de estas a la posición de la cámara, para su posterior envío al brazo robótico.
7. Realizar pruebas de la aplicación final, tanto en entornos simulados como reales.

3.2. Plan de trabajo

El desarrollo y seguimiento que el proyecto ha seguido es una planificación en base a reuniones semanales con el tutor, en las cuales se revisaron los avances, se fijaron nuevos objetivos y se discutieron y propusieron posibles mejoras, mientras que el trabajo se organizó en varias fases clave:

1. *Investigación inicial:* En esta fase, se investigó el estado del arte relacionado con sistemas de visión artificial y técnicas de reconocimiento de objetos, especialmente aplicadas a la maduración de frutas y hortalizas, y utilizando para ello artículos científicos, capítulos de libros y proyectos previos.
2. *Diseño y desarrollo del sistema de visión artificial:* Esta fase se centró en el diseño y la implementación del sistema de visión artificial, abarcando tanto el desarrollo del software como la integración del hardware, e incluyendo la calibración y

obtención de los parámetros intrínsecos a la cámara y las diversas pruebas realizadas con distintos sistemas y códigos, hasta seleccionar el *software* funcional con el que se llevó a cabo el proyecto finalmente.

3. *Pruebas en entorno simulado:* Durante esta fase se realizaron múltiples pruebas y ajustes para optimizar el funcionamiento del sistema y comprobar su funcionamiento en diferentes escenarios, simulando de manera separada la programación del robot, para el que se utilizó un simulador en una máquina virtual, y la detección y funcionamiento del sistema de visión, cuyos algoritmos se afinaron para mejorar la precisión en la detección y se ajustaron los parámetros relacionados con la cámara en los códigos para poder obtener las coordenadas y distancia real de las detecciones respecto a la cámara y poder transmitírselas al brazo robótico. Finalmente, también se llevaron a cabo pruebas de comunicación entre el sistema de visión y el robot, poniendo a prueba su programación, para que este alcance el punto de la detección.
4. *Pruebas en entorno real:* Una vez desarrollado el prototipo inicial, el sistema completo fue sometido a pruebas en un entorno real de lo que sería la aplicación final.
5. *Escritura de la memoria:* Con el sistema ya afinado y probado, se procedió a la redacción de la memoria del proyecto. En esta etapa, se documentó detalladamente todo el proceso seguido, desde la investigación inicial hasta los resultados finales obtenidos durante las pruebas reales.

Todo el contenido del proyecto se puede encontrar en un repositorio público de GitHub¹³, en cuya wiki¹⁴ se puede ver el desarrollo del trabajo en semanas a lo largo de los meses, durante el trascurso del proyecto. Las requisitos necesarios para la consecución de los objetivos planteados, las competencias desarrolladas y la metodología empleada pueden encontrarse descritos en el Anexo I.

¹³<https://github.com/RoboticsURJC/tfg-dcampoamor>

¹⁴<https://github.com/RoboticsURJC/tfg-dcampoamor/wiki>

Capítulo 4

Plataforma de desarrollo

Con los objetivos del proyecto definidos, en este capítulo se abordarán las distintas plataformas de desarrollo, tanto *hardware* como *software*, que han facilitado el logro de esos objetivos.

4.1. Hardware

Este apartado recoge la descripción de los componentes *hardware* utilizados en este proyecto, para los cuales se ha buscado priorizar la reducción de costes en cada elección y utilizar aquellos elementos a los que se tenía acceso al desarrollar el proyecto.

4.1.1. Cámara Logitech C270 HD

Esta cámara (Figura 4.1(a)), de dimensiones 72,91 x 31,91 x 66,64 mm, corrige la iluminación de manera automática, produciendo colores reales y naturales y ajustándose a las condiciones de iluminación del entorno, lo que facilita la detección de fresas. Ofrece una resolución HD 720p a una velocidad de 30 fotogramas por segundo (fps), con una lente que cuenta con enfoque fijo y un campo visual diagonal (dFoV) de 55 grados. Su coste aproximado es de entre 30-40€.



Figura 4.1: Sistema de visión

4.1.2. Soporte de brazo articulado

Para poder ubicar la cámara en una posición fija desde la cual visualizar las fresas, se utilizó un soporte de brazo articulado (Figura 4.1(b)), cuya parte fija en la parte inferior se ancla a la mesa. Este soporte articulado tiene un ajuste de 360 grados, con su extremo más largo de 75 cm, mientras que la carga máxima que permite es de 560 gramos cuando se coloca de manera horizontal, siendo el precio de este soporte 23€.

4.1.3. Soporte de impresión 3D

Para las pruebas finales del sistema, se diseñó e imprimió en 3D un soporte específico para la cámara¹⁸ utilizando PLA de color blanco como material de impresión (Figura 4.2), Ultimaker Cura¹⁹ como software de impresión y la impresora CREALITY CR-10 Smart Pro²⁰ para ello con el fin de fijarla posteriormente al brazo articulado con el objetivo de asegurar una sujeción estable y precisa de la cámara durante los ensayos, de forma que permitiera colocar la cámara en una posición perpendicular tanto al plano de la mesa como a la pared, cumpliendo así con los requisitos geométricos necesarios para el correcto funcionamiento del sistema de visión artificial.

¹⁶<https://www.logitech.com/es-es/products/webcams/c270-hd-webcam.960-001063.html?srstid=AfmB0or4HptUTCGrxE-4SzxKR-ARw-ykNeagHSEzXUvT1Xkx8qLfY4lG>

¹⁷https://www.amazon.es/dp/B08JCG4V5S?ref=ppx_pop_mob_ap_share&th=1

¹⁸https://github.com/dcampaomor/tfg-dcampoamor/blob/dcampaomor_main/aux/CCR10S_camer%20holder%20v2.gcode

¹⁹<https://ultimaker.com/es/software/ultimaker-cura/>

²⁰<https://www.creality.com/es/products/creality-cr-10-smart-pro-3d-printer>

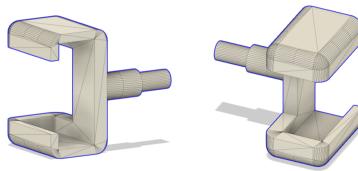


Figura 4.2: Soporte de impresión 3D en PLA

4.1.4. Ordenador principal

El equipo que se ha configurado como entorno de trabajo para este proyecto es un Lenovo Legion 5 IMH05²¹ con procesador Intel Core i7 y tarjeta gráfica dedicada NVIDIA GeForce GTX 1650 Ti Mobile. Ha servido como base para el desarrollo de la programación y las pruebas de la visión artificial, y utilizándose igualmente como servidor para poder llevar a cabo la comunicación con el brazo robótico mediante el protocolo XML-RPC basado en HTTP, y posteriormente el envío de posiciones detectadas en tiempo real a este.

4.1.5. Robot de Universal Robots de la gama e-series

Los robots de Universal Robots, también conocidos como robots colaborativos o cobots, están diseñados para trabajar junto a los humanos de manera segura, eficiente y flexible tanto en aplicaciones industriales como no industriales, destacando por su facilidad de uso, versatilidad y capacidad para automatizar tareas repetitivas o peligrosas [Universal Robots A/S, 2018].

Son fabricados en aluminio, junto con otros materiales de bajo peso. Cada brazo robótico tiene seis ejes, otorgando al robot seis grados de libertad (Degree Of Freedom o DOF), que permiten movimientos precisos y fluidos, y en cuyas articulaciones están equipadas, a su vez, encoders absolutos y reductoras armónicas, que reducen la velocidad de rotación de los engranajes en las juntas, y aumentan el par del eje, ofreciendo una alta precisión y eficiencia; y, en aquellos brazos robóticos pertenecientes a la gama e-series, sensores de fuerza y torque, encontrándose integrados en el efecto o tool flange del propio brazo robótico²².

²¹<https://www.pcccomponentes.com/lenovo-legion-5-15imh05-intel-core-i7-10750h-16gb-1tb-ssd-gtx-1650-156?srsltid=AfmBOopJpvGSHUyQU696jgG7-6orSKMOEWZe2ZvvYtA0NGtJ9Ms2xJFp>

²²<https://www.universal-robots.com/mx/acerca-de-universal-robots/noticias/meet-the-next-generation-of-collaborative-ur-robots-at-robobusiness>

Para el desarrollo de este proyecto se han utilizado diferentes modelos de robots de la gama e-series, todos ellos representados en la Figura 4.3; desde el UR3e [Universal Robots A/S, 2025b] hasta el UR10e [Universal Robots A/S, 2025a], pasando por el UR5e [Universal Robots A/S, 2025c]. A pesar de que tanto la gama CB-series como la gama e-series permiten la comunicación mediante el protocolo XML-RPC, la interfaz más intuitiva y simplificada que facilita la programación y reduce la sobrecarga de información, poseer sensor de fuerza integrado, permitiendo un control más preciso y sensible haciendo que mejore la interacción con el entorno, así como una mayor precisión en la repetición de movimientos, entre otros factores, constituyeron que se eligiera la gama e-series para este trabajo [Universal Robots A/S, 2024].



Figura 4.3: Gama e-series de Universal Robots²³

4.1.6. Comunicaciones

En este proyecto, la comunicación entre el robot y el ordenador que ejecuta el servidor XML-RPC se establece vía Ethernet, permitiendo una conexión rápida y confiable. La infraestructura de red juega un papel fundamental en esta comunicación, ya que el robot y el servidor deben estar correctamente configurados dentro de la misma red, por lo que se emplea un switch Ethernet, en este caso el TP-Link LS105G²³, para gestionar las conexiones entre los diferentes dispositivos y asegurar una comunicación fluida y sin interferencias.

Este switch de escritorio, cuyo precio es de 15€, posee cinco puertos Ethernet RJ45 a 10/100/1000 Megabits por segundo (Mbps), permitiendo la transferencia instantánea de archivos y paquetes, con gran ancho de banda y sin interferencias, y no necesita configuración manual previa.

²²<https://www.universal-robots.com/es/productos/>

²³<https://www.tp-link.com/es/business-networking/litewave-switch/ls105g/>

4.2. Software

Este apartado está dedicado a detallar las plataformas software, librerías y entornos de trabajo que han sido fundamentales para alcanzar los objetivos definidos en el Capítulo 3, desde el sistema operativo utilizado hasta las tecnologías específicas para el procesamiento de imágenes y aprendizaje profundo.

4.2.1. Ubuntu

Ubuntu²⁴ es un sistema operativo de código abierto basado en Linux y desarrollado por la empresa británica Canonical Ltd. Este sistema operativo está diseñado para ser utilizado en una gran variedad de dispositivos, y es reconocido por su facilidad de uso, estabilidad y seguridad, contando con una amplia comunidad de desarrolladores y usuarios que contribuyen activamente a su desarrollo y soporte. La versión utilizada para la realización de este proyecto, de entre todas las versiones disponibles, es Ubuntu 22.04 Long Term Support (LTS) (Jammy Jellyfish), ya que era la última versión disponible de Ubuntu en el momento en el que se empezó a elaborar el proyecto.

4.2.2. Polyscope

Polyscope²⁵ es la interfaz de usuario gráfica desarrollada por Universal Robots para poder programar y utilizar sus robots colaborativos. Está diseñada para ser intuitiva y accesible, ya que permite a los usuarios crear programas de robot sin necesidad de conocimientos avanzados en programación.

Construido sobre una plataforma basada en Linux, PolyScope está basado en una arquitectura de software que combina una interfaz gráfica amigable con un lenguaje de programación propio llamado URScript, permitiendo tanto a usuarios sin experiencia en programación como a programadores avanzados interactuar eficazmente con los robots UR, ya que la interfaz gráfica facilita la creación de programas mediante bloques visuales, mientras que URScript ofrece una mayor flexibilidad para desarrollos más complejos.

²⁴<https://ubuntu.com/>

²⁵<https://www.universal-robots.com/es/productos/polyscope/>

A lo largo de los años, UR ha lanzado varias versiones de PolyScope, cada una con mejoras y nuevas funcionalidades, siendo la primera versión PolyScope 3, lanzada en 2012, ya que fue diseñada para la serie CB3 de robots UR. PolyScope 5²⁶ (Figura 4.4) se introdujo en junio de 2018, coincidiendo con el lanzamiento de la gama de robots e-series. Por último, PolyScope X²⁷ es la última evolución del software de Universal Robots, siendo su lanzamiento oficial en noviembre de 2024²⁸, y estando basado en tecnologías como ROS2 y contenedores Docker, centraliza las funciones más importantes y simplifica la programación mediante el uso de plantillas predefinidas.

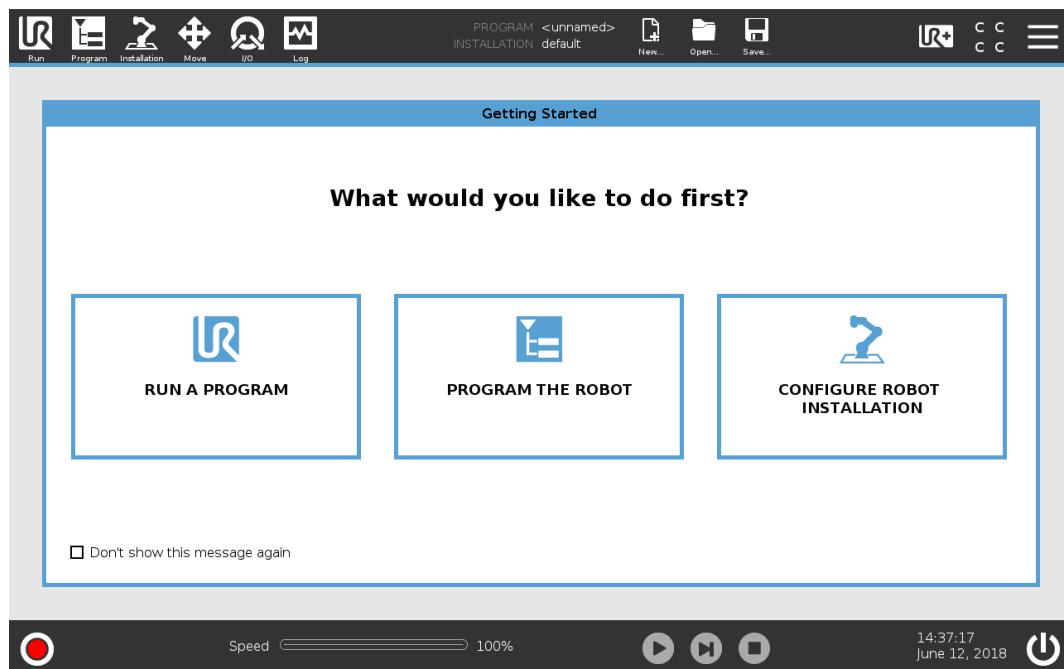


Figura 4.4: Pantalla principal de la interfaz de Polyscope 5

A pesar de que Polyscope X es compatible con los robots de la gama e-series, se deben cumplir una serie de requisitos en cuanto al hardware de la controladora para poder actualizar desde Polyscope 5, por lo que para el desarrollo de este proyecto se ha terminado utilizado la versión 5.16 de Polyscope 5, puesto que no se cumplían todos los requisitos para que se diera esta actualización en los robots utilizados.

²⁶<https://www.universal-robots.com/products/polyscope-5/>

²⁷<https://www.universal-robots.com/products/polyscope-x/>

²⁸<https://www.universal-robots.com/2024q3/polyscope-x-festival/>

4.2.3. Anaconda

Anaconda²⁹ es una distribución de código abierto para los lenguajes de programación Python y R, diseñada para facilitar la gestión de paquetes y entornos, así como el despliegue de aplicaciones de ciencia de datos y aprendizaje automático. Ofrece herramientas como *conda*, un sistema de gestión de paquetes y entornos que funciona en Windows, macOS y Linux; y Anaconda Navigator, una aplicación de escritorio que permite gestionar aplicaciones integradas, paquetes y entornos sin necesidad de utilizar la línea de comandos³⁰.

Para este proyecto se ha hecho uso del programa Conda, ya que, utilizando esta herramienta es posible instalar y actualizar paquetes y dependencias y cambiar entre entornos desde el mismo ordenador local, permitiendo que puedan ser mantenidos y ejecutados independientemente sin archivos, directorios y rutas, para que se pueda trabajar con versiones específicas de librerías y/o el propio Python, sin afectar a otros proyectos Python, es decir, no afectando los cambios de un entorno a otro³¹, estando recogidas las versiones de los paquetes y bibliotecas necesarias del entorno utilizado *deteccionobj* en los documentos *requirements.txt*³² y *environment.yml*³³.

4.2.4. Python

Python³⁴ es un lenguaje de programación de alto nivel, orientado a objetos y de semántica dinámica. La sintaxis de Python, sencilla y fácil de aprender, favorece la legibilidad y, por tanto, reduce el coste de mantenimiento de los programas, admitiendo módulos y paquetes, lo que fomenta la modularidad del programa y la reutilización de código³⁵ para el desarrollo de aplicaciones en diferentes áreas, como sucede en este caso con la inteligencia y visión artificial y el aprendizaje automático.

²⁹<https://www.anaconda.com/>

³⁰<https://docs.anaconda.com/anaconda/>

³¹<https://docs.anaconda.com/reference/glossary/#conda>

³²<https://github.com/RoboticsURJC/tfg-dcampoamor/blob/main/src/deteccion-objetos-video/requirements.txt>

³³<https://github.com/RoboticsURJC/tfg-dcampoamor/blob/main/src/deteccion-objetos-video/environment.yml>

³⁴<https://www.python.org/>

³⁵<https://www.python.org/doc/essays/blurb/>

4.2.5. PyTorch

De los desarrolladores de Facebook AI Research, junto a otros laboratorios, PyTorch³⁶ es un marco de aprendizaje profundo de código abierto conocido por su compatibilidad con Python, siendo un marco de trabajo completo para crear modelos de aprendizaje profundo. Se distingue por su excelente compatibilidad con GPU y su uso de la autodiferenciación en modo inverso, que permite modificar los gráficos de cálculo sobre la marcha, lo que lo convierte en una opción popular para la experimentación rápida y la creación de prototipos.

4.2.6. NumPy

NumPy (Numerical Python)³⁷ es una biblioteca de Python fundamental para el cálculo numérico, que proporciona un objeto de matriz multidimensional, varios objetos derivados (como matrices y matrices enmascaradas) y un surtido de rutinas para realizar operaciones rápidas con matrices³⁸.

En este proyecto, la biblioteca Numpy se ha utilizado para inicializar matrices y vectores de los parámetros intrínsecos y extrínsecos de la cámara, realizar cálculos geométricos y poder obtener las matrices de rotación y traslación de la cámara, llevar a cabo operaciones matriciales como multiplicaciones e inversiones. También se ha usado para poder proyectar las coordenadas en el espacio 2D a coordenadas tridimensionales mediante operaciones matriciales, pudiendo obtener posteriormente las distancias a los objetos detectados.

4.2.7. OpenCV

OpenCV (Open Source Computer Vision Library)³⁹ es una biblioteca de software de código abierto diseñada para su uso en aplicaciones de aprendizaje automático y visión artificial. Desarrollado por Intel⁴⁰ en 1999, cuenta con más de 2.500 algoritmos optimizados, que incluyen un amplio conjunto de algoritmos de visión por ordenador y aprendizaje automático.

³⁶<https://pytorch.org/>

³⁷<https://numpy.org/>

³⁸<https://numpy.org/doc/stable/user/whatisnumpy.html>

³⁹<https://opencv.org/>

⁴⁰<https://www.intel.es>

En este proyecto, OpenCV se ha usado importado como `cv2` para poder capturar los frames desde la cámara en tiempo real y mostrarlo en una ventana para poder controlar y verificar que se realizan las detecciones correctamente, para convertir estos frames del formato BGR a RGB, dibujar un rectángulo alrededor de las fresas detectadas, añadiendo la etiqueta de texto correspondiente que indica la clase detectada y la confianza de esta detección. Permite detener el bucle principal del programa, terminarlo y salir de este si se presiona la tecla configurada para ello, asegurando que la cámara o el archivo de vídeo no permanezcan bloqueados por el programa y cerrando todas las ventanas de visualización creadas.

4.2.8. OpenGL

OpenGL (Open Graphics Library)⁴¹ es una especificación de gráficos 2D y 3D de código abierto que define una API multiplataforma ampliamente utilizada para desarrollar aplicaciones gráficas interactivas desarrollado inicialmente por Silicon Graphics Inc. (SGI) en 1992.

Gracias a su rendimiento y compatibilidad con GPU, OpenGL permite una visualización fluida y personalizada de las detecciones, que ha permitido que en este proyecto, se haya integrado con otras bibliotecas como OpenCV para superponer elementos visuales, como rectángulos, etiquetas o coordenadas, sobre el entorno, utilizándose para representar gráficamente la información capturada por la cámara en una ventana emergente, facilitando así las pruebas relacionadas con el cálculo de la distancia a las detecciones. Esto ha permitido verificar si los resultados obtenidos eran válidos y coherentes, así como comprobar que el sistema respondía correctamente a la distorsión geométrica derivada de la perspectiva de la cámara para el modelo pinhole.

4.2.9. SocketTest

SocketTest⁴² es una herramienta de software utilizada para probar conexiones de red a través de sockets, ya sea como cliente o como servidor, que permite enviar y recibir datos de manera manual o automatizada mediante protocolos como TCP o UDP.

⁴¹<https://opengl.org/>

⁴²<https://sockettest.sourceforge.net/>

En este trabajo, se ha utilizado la versión v3.0.0, tal y como se puede comprobar en la Figura 4.5, para probar y verificar en una etapa temprana del proyecto, la comunicación entre el robot y un servidor externo, permitiendo comprobar que los puertos estaban abierto y que la conexión era posible, ya que los datos enviados desde el servidor llegaban correctamente al robot, de igual manera que los datos que se enviaron del robot al servidor.

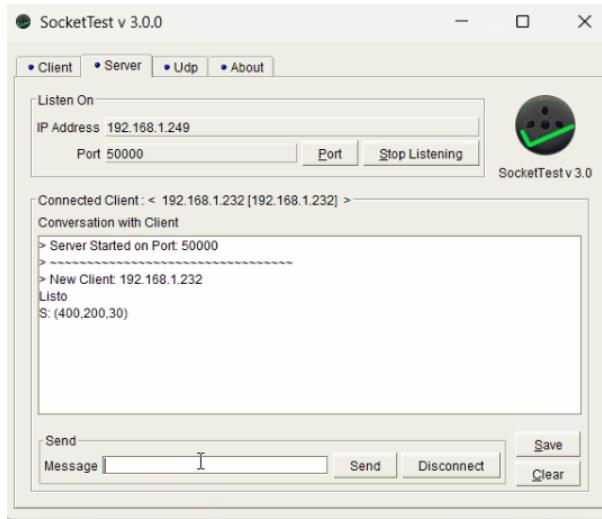


Figura 4.5: Pruebas realizadas con SocketTest para verificar la comunicación robot-servidor externo

4.2.10. XML-RPC

XML-RPC⁴³ es un método de llamada a procedimiento remoto (RPC) que usa XML para codificar y transferir datos entre programas a través de sockets y HTTP como protocolo de transporte. Para muchos lenguajes de programación existen servidores XML-RPC gratuitos, entre otros para: Python, Java, C++ y C.

Debido al uso de Python en el proyecto, se utilizó el paquete *xmlrpc*, que agrupa los módulos tanto de cliente como de servidor que implementan XML-RPC. Con este paquete, el controlador del robot UR puede llamar a métodos o funciones (con parámetros) en un programa/servidor remoto y obtener de vuelta datos estructurados, pudiendo realizar un cálculo complejo mediante su uso, que no está disponible en el lenguaje propio de programación del robot.

⁴³<https://docs.python.org/es/3.8/library/xmlrpc.html>

4.2.11. YOLOv3

YOLOv3 (You Only Look Once versión 3)⁴⁴ es un algoritmo de detección de objetos en tiempo real que identifica y localiza múltiples objetos dentro de una imagen o vídeo. Desarrollado por Joseph Redmon y Ali Farhadi en 2018, YOLOv3 es la tercera iteración de la serie YOLO, conocida por su capacidad para realizar detecciones rápidas y precisas [Redmon and Farhadi, 2018]. La serie YOLOv3, está diseñada específicamente para tareas de detección de objetos, además, YOLOv3 añadió funciones como predicciones multietiqueta para cada cuadro delimitador y una red extractora de características mejorada. Estos modelos son famosos por su eficacia en diversos escenarios del mundo real, equilibrando precisión y velocidad, lo que los hace adecuados para una amplia gama de aplicaciones⁴⁵.

Esta herramienta ha sido elegida para el entrenamiento del modelo de detección de fresas debido a su eficiencia en el uso de recursos computacionales, lo que lo hace más accesible para sistemas con capacidades limitadas; a la amplia documentación y la comunidad activa existente en torno a YOLOv3, que proporciona recursos valiosos para la implementación y resolución de problemas, lo que es esencial en proyectos académicos con plazos definidos⁴⁶; a la competencia en cuanto a precisión y velocidad de YOLOv3 frente a versiones superiores a pesar de presentar estas ciertas mejoras; y a que YOLOv3 es compatible con entornos de desarrollo ampliamente utilizados en proyectos académicos, como Python y bibliotecas estándar de aprendizaje automático, facilitando su integración en el flujo de trabajo del proyecto.

Una vez analizadas las plataformas de software y hardware utilizadas en este trabajo de fin de grado, se procederá a detallar el proceso completo de diseño y desarrollo del sistema, lo cual será explicado con detalle en el Capítulo 5, pudiéndose encontrar, además, diversos experimentos en el Anexo II.

⁴⁴<https://docs.ultralytics.com/es/models/yolov3/>

⁴⁵<https://docs.ultralytics.com/es/models/yolov3/#supported-tasks-and-modes>

⁴⁶<https://github.com/ultralytics/yolov3/>

Capítulo 5

Descripción del sistema

Una vez expuestos el contexto, el estado del arte, los objetivos del proyecto y las plataformas de desarrollo empleadas, en este capítulo se describe de forma detallada el prototipo desarrollado (Figura 5.1), y se abordan tanto los fundamentos teóricos que lo sustentan como el proceso seguido para su diseño e implementación, con el objetivo de ofrecer una visión completa del funcionamiento del sistema.



Figura 5.1: Montaje del sistema final con un UR5e

5.1. Hipótesis suelo adaptada al plano vertical

En el sistema propuesto, la estimación de coordenadas tridimensionales de las fresas a recolectar se basa en la hipótesis suelo, y se realiza a partir de una única cámara estenopeica RGB fija para poder realizar esta estimación sin recurrir a sensores adicionales. Esta técnica de simplificación empleada en visión por ordenador asume que los objetos de interés se encuentran sobre un plano conocido y fijo respecto a la cámara.

La hipótesis suelo ([Vega and Cañas, 2021], Figura 5.2) supone que todos los objetos del mundo en 3D están apoyados sobre el suelo, por lo tanto, asumiendo que el suelo está en el plano $Z = 0$, nos permitirá estimar, con el uso de la cámara, la posición 3D de los puntos que estén en el suelo.

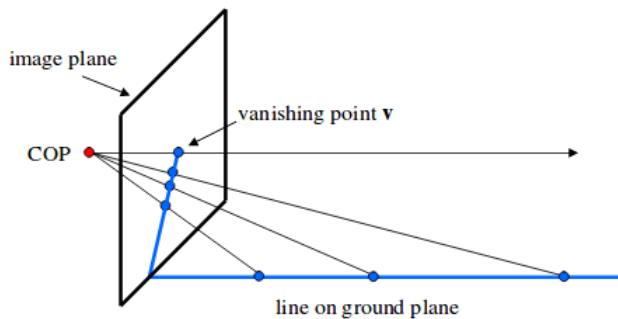


Figura 5.2: La hipótesis suelo asume que todos los objetos están en el suelo

El hecho de que la cámara siga el modelo *pinhole* o estenopeico, implica que la proyección de un punto en el espacio bidimensional de una imagen al tridimensional se realiza usando los parámetros intrínsecos de la propia cámara y parámetros extrínsecos como la rotación o la traslación de la misma (Cuadro 5.1). Al asumir que los puntos están sobre el plano $Z=0$ mediante la hipótesis suelo, este modelo estenopeico permite estimar las coordenadas X e Y reales a partir de la imagen captada por la cámara mediante una serie de fórmulas matemáticas (Ecuación 5.1).

Parámetros de cámara	Definición
$K (3 \times 3)$	Parámetros intrínsecos
$R (3 \times 3)$	Rotación de la cámara
$T (3 \times 1)$	Traslación de la cámara

Cuadro 5.1: Definición de los parámetros de posición y orientación de la cámara pinhole

$$w \cdot \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \cdot [R | T] \cdot \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Ecuación 5.1: Relación de fórmulas del modelo de cámara estenopeico

En esta ecuación se observa la relación entre un punto tridimensional (X, Y, Z) y su proyección bidimensional (u, v) multiplicada por un factor de escala w que representa la profundidad del punto proyectado en la imagen captada por la cámara, donde la matriz de parámetros intrínsecos K (Ecuación 5.2) junto con las matrices de rotación R (Ecuación 5.3) y la matriz de translación T de la cámara (Ecuación 5.4), constituyen la matriz de proyección. Esta matriz de proyección transforma los puntos tridimensionales al plano imagen.

$$K = \begin{pmatrix} F_x & 0 & C_x \\ 0 & F_y & C_y \\ 0 & 0 & 1 \end{pmatrix}$$

Ecuación 5.2: Matriz de parámetros intrínsecos de la cámara

$$R(\theta)_X = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & \sin(\theta) \\ 0 & -\sin(\theta) & \cos(\theta) \end{pmatrix}$$

$$R(\theta)_Y = \begin{pmatrix} \cos(\theta) & 0 & -\sin(\theta) \\ 0 & 1 & 0 \\ \sin(\theta) & 0 & \cos(\theta) \end{pmatrix}$$

$$R(\theta)_Z = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Ecuación 5.3: Matrices de rotación $R(\theta)$ según el eje de rotación

$$T = \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix}$$

Ecuación 5.4: Matriz de translación de la cámara en coordenadas tridimensionales

Para poder obtener la proyección tridimensional a partir de la bidimensional habría que operar en la Ecuación 5.1 para despejar las incógnitas (X, Y, Z) , siendo la distancia a la cámara constante y conocida ($Z = Z_0$) debido a la hipótesis suelo (Ecuación 5.5), ya que se dispone de los parámetros intrínsecos de la cámara y la geometría del plano.

$$\begin{aligned} K^{-1} \cdot w \cdot \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} &= R \cdot \begin{bmatrix} X \\ Y \\ Z_0 \end{bmatrix} + T \\ \begin{bmatrix} X \\ Y \\ Z_0 \end{bmatrix} &= R^{-1} \cdot \left(K^{-1} \cdot w \cdot \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} - T \right) \end{aligned}$$

Ecuación 5.5: Descomposición e inversión parcial del modelo de cámara pinhole para estimar las coordenadas tridimensionales de un punto conocido en imagen bajo la hipótesis suelo $Z = Z_0$.

En este caso, y debido a la orientación del escenario, la hipótesis suelo ha sido adaptada al plano vertical, lo que significa que se asume que las fresas están dispuestas sobre una superficie vertical conocida, como es el caso en los huertos verticales. En este plano de trabajo vertical, se ha establecido un sistema de coordenadas 3D, cuyo origen coincide con el punto central del campo de visión de la cámara, actuando como referencia para crear el plano de trabajo en el robot a través de su interfaz y utilizando las mismas orientaciones de los ejes de coordenadas de la cámara (Figura 5.3). Esto permite que una vez realizado el cálculo de las coordenadas de las fresas detectadas y el de las distancias a cada detección (Ecuación 5.6), estas coincidan con las coordenadas en este plano vertical, simplificando de esta manera el proceso de transformación de las mismas.

$$\text{distancia} = \sqrt{X^2 + Y^2 + Z^2}$$

Ecuación 5.6: Fórmula del cálculo de la distancia de la cámara a las detecciones

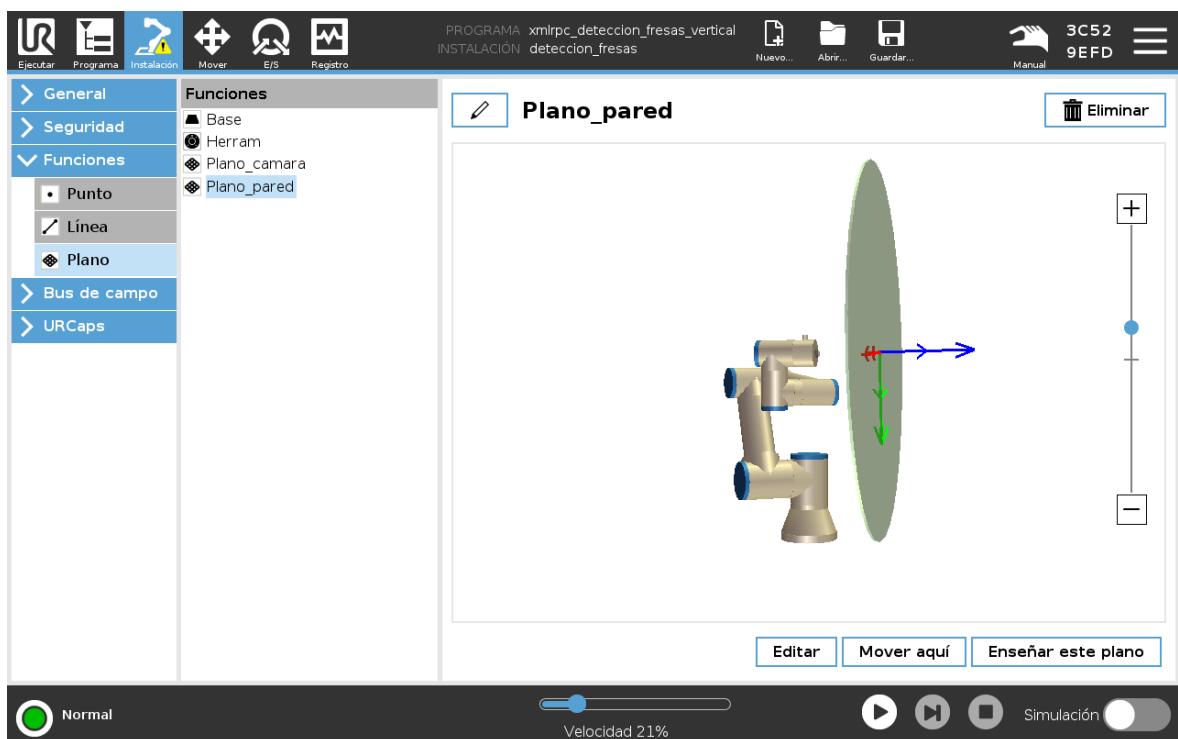


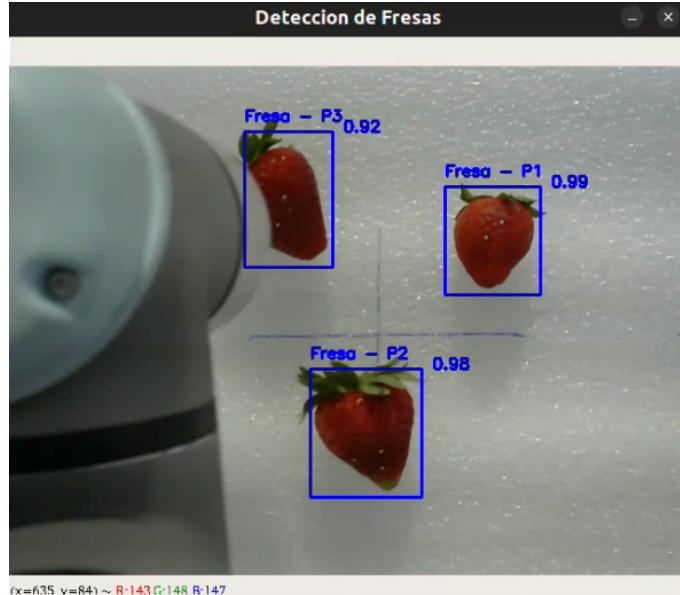
Figura 5.3: Plano pared generado en el UR

5.2. Detección de fresas mediante Deep Learning

La detección de las fresas maduras se ha resuelto mediante el uso de técnicas de visión artificial basadas en deep learning, concretamente, se ha utilizado el modelo YOLOv3 (You Only Look Once), un detector de objetos en tiempo real ampliamente utilizado por su equilibrio entre precisión y velocidad, tal y como se define en la Sección 4.2.11.

Este modelo ha sido entrenado específicamente para reconocer fresas maduras en las condiciones del entorno de trabajo y, una vez entrenado, se ha integrado en un script que analiza el flujo de vídeo en tiempo real y devuelve las coordenadas y la distancia a la cámara de cada fresa detectada en la imagen, junto con la clase del objeto y la confianza de detección (Figura 5.4).

Para mejorar la precisión a la hora de estimar la posición de la fresa, se ha utilizado el centro del *bounding box* de la fresa detectada como punto de referencia para la proyección sobre el plano vertical, conforme a la hipótesis descrita en la Sección 5.1.



(a) Ventana donde se muestran las detecciones

```
[INFO] Iniciando programa...
[INFO] Iniciando detección de fresas en el frame actual...
[DEBUG] Matriz RT de la cámara:
[[ 1.   0.   0.   0.]
 [ 0.   1.   0.   0.]
 [ 0.   0.   1. -355.]
 [ 0.   0.   0.   1.]]
[INFO] Parámetros de la cámara cargados correctamente.
Servidor XML-RPC corriendo en el puerto 50000...
Punto P1 - Coordenadas 3D: X=60.77, Y=-31.44, Z=355.00
Punto P1 - Distancia al punto: 349.76 milímetros
Punto P2 - Coordenadas 3D: X=8.43, Y=47.15, Z=355.00
Punto P2 - Distancia al punto: 346.33 milímetros
Punto P3 - Coordenadas 3D: X=-28.20, Y=-48.77, Z=355.00
Punto P3 - Distancia al punto: 347.60 milímetros
[DEBUG] Enviando última posición detectada: (-28.20182882193582, -48.771758071747826, 355.0)
192.168.23.214 - - [05/May/2025 19:01:15] "POST / HTTP/1.1" 200 -
Punto P1 - Coordenadas 3D: X=60.93, Y=-31.53, Z=355.00
Punto P1 - Distancia al punto: 349.79 milímetros
Punto P2 - Coordenadas 3D: X=8.96, Y=47.30, Z=355.00
Punto P2 - Distancia al punto: 346.36 milímetros
Punto P1 - Coordenadas 3D: X=60.77, Y=-31.40, Z=355.00
Punto P1 - Distancia al punto: 349.75 milímetros
Punto P2 - Coordenadas 3D: X=7.73, Y=47.31, Z=355.00
Punto P2 - Distancia al punto: 346.33 milímetros
Punto P3 - Coordenadas 3D: X=-24.18, Y=-47.87, Z=355.00
Punto P3 - Distancia al punto: 347.17 milímetros
```

(b) Mensajes mostrados por la terminal al ejecutar el programa de detección

Figura 5.4: Detección de fresas maduras mediante deep learning

5.3. Arquitectura del sistema

El prototipo final se ha implementado utilizando una combinación de componentes hardware y software, descritos con detalle en el Capítulo 4, específicamente seleccionados y configurados para responder a los requisitos del sistema de recolección automatizada. Esta integración ha sido diseñada para garantizar la detección precisa de fresas, el cálculo de su localización espacial en coordenadas tridimensionales y la ejecución del movimiento del robot sobre estas posiciones (Figura 5.5).

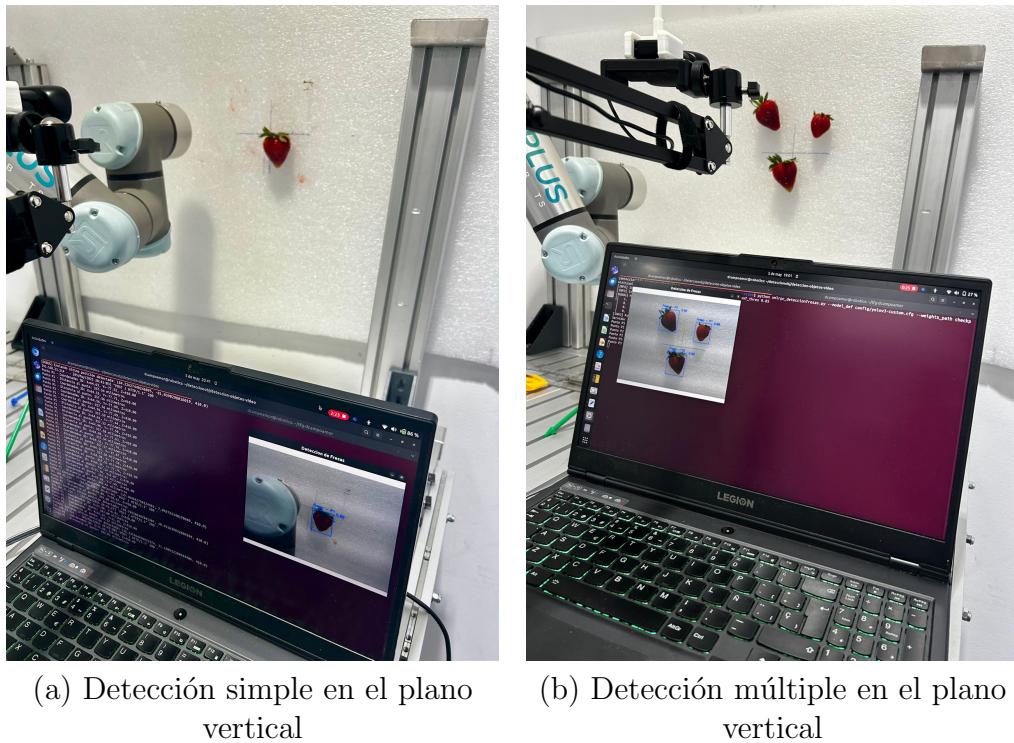


Figura 5.5: Disposición de las detecciones para un plano horizontal con un UR5e

A partir de esta base, este apartado expone el proceso completo de integración y funcionamiento coordinado de los distintos módulos que conforman la solución propuesta, siendo este el siguiente:

1. La cámara captura imágenes de la escena en tiempo real.
2. El modelo de deep learning detecta las fresas maduras y genera sus posiciones en píxeles gracias al fragmento de código del script en Python 5.1.

```

with torch.no_grad():
    detections = model(imgTensor)
    detections = non_max_suppression(detections, opt.conf_thres,
                                      opt.nms_thres)

    positions = []
    if detections:
        for detection in detections:
            if detection is not None:
                detection = rescale_boxes(detection, opt.img_size,
                                           frame.shape[:2])
                for i, det in enumerate(detection):
                    if len(det) == 7:
                        x1, y1, x2, y2, cls_conf, conf, cls_pred = det
                        box_w = x2 - x1
                        box_h = y2 - y1
                        center_x = x1 + box_w // 2
                        center_y = y1 + box_h // 2
                        color = (255, 0, 0)
                        cv2.rectangle(frame, (int(x1), int(y1)), (int(x2),
                            int(y2)), color, 2)
                        label = f"Fresa - P{i+1}"
                        cv2.putText(frame, label, (int(x1), int(y1) - 10),
                                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)
                        confidence_text = f"{float(cls_conf):.2f}"
                        cv2.putText(frame, confidence_text, (int(x2) + 10,
                            int(y1)),
                                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)
                        p2d = np.array([center_x, center_y])

```

Código 5.1: Fragmento del código que permite que el modelo de *deep learning* detecte las fresas maduras y genere sus posiciones en píxeles

3. Estas coordenadas se proyectan al espacio tridimensional sobre el plano vertical utilizando la función `getIntersectionZ(p2d)` (Código 5.3), que calcula la intersección del rayo proyectado con el plano Z definido por la posición de la cámara, utilizando como argumento de entrada `p2d`, es decir, las coordenadas en píxeles de la detección (Código 5.2).

```

pixelOnGround3D = getIntersectionZ(p2d)
x_punto = float(pixelOnGround3D[0])
y_punto = float(pixelOnGround3D[1])
z_punto = float(pixelOnGround3D[2])
positions.append((x_punto, y_punto, z_punto))

```

Código 5.2: Fragmento del código que realiza la retroproyección desde 2D a 3D

```

def getIntersectionZ(p2d):
    p2d_h = np.array([p2d[0], p2d[1], 1])
    inv_K = np.linalg.inv(myCamera.k[:, :3])
    inv_RT = np.linalg.inv(myCamera.rt[:3, :3])
    p3d_h = np.dot(inv_K, p2d_h)
    p3d_h = np.dot(inv_RT, p3d_h)
    if np.abs(p3d_h[2]) > 1e-6:
        escala = -myCamera.position[2] / p3d_h[2]
        p3d_h *= escala
    return np.array(p3d_h)

```

Código 5.3: Función `getIntersectionZ()`

4. El servidor XML-RPC se crea en la dirección IP y puerto especificado, se transmiten las coordenadas al robot a través de la función `get_next_pose()`, y lanza en un hilo independiente que queda a la espera de nuevas peticiones (Código 5.4).

```

# Conexion con el robot usando XML-RPC
server = SimpleXMLRPCServer(("192.168.23.107", 50000))
server.RequestHandlerClass.protocol_version = "HTTP/1.1"
print("Servidor XML-RPC corriendo en el puerto 50000...")

def get_next_pose():
    if detected_points:
        pose = detected_points[-1]
        print(f"[DEBUG] Enviando ultima posicion detectada: {pose}")
        return [pose[0], pose[1], pose[2], 0, 0, 0]
    else:
        print(f"[DEBUG] No se detectaron puntos, enviando posicion de
              inicio")
        return [0, 0, 0, 0, 0, 0]

server.register_function(get_next_pose, "get_next_pose")
import threading
server_thread = threading.Thread(target=server.serve_forever)
server_thread.daemon = True
server_thread.start()

```

Código 5.4: Envío de la última posición tridimensional detectada al robot mediante XML-RPC

5. El robot interpreta la posición, calcula una trayectoria y actúa sobre la fruta detectada (Figura 5.6).



The figure consists of two screenshots of a software interface. The top screenshot shows a hierarchical tree of configuration variables. The root node is 'Configuración de variables'. It has three children: 'AntesDeIniciar', 'Configuración XMLRPC', and 'camera:=rpc_factory("xmlrpc","http://192.168.23.107:50000")'. The bottom screenshot shows a detailed view of movement instructions. The code starts with 'next_pose:=camera.get_next_pose()' and continues with several conditional blocks ('If', 'Else') that involve 'Movimiento UR' (UR movement) and 'Mover' (Move) actions. The code ends with 'Esperar: 1.0'.

```

1 X Configuración de variables
2 ▾ AntesDeIniciar
3 ♀ Configuración XMLRPC
4   ⚡ 'Configuración del XMLRPC'
5     camera:=rpc_factory("xmlrpc","http://192.168.23.107:50000")

```

(a) Configuración de la comunicación XMLRPC en el robot

```

20   ⚡ next_pose:=camera.get_next_pose()
21   ⚡ 'Se hace el cambio de unidades mm a m'
22   ⚡ next_pose_list:=p[next_pose[0]/1000,next_pose[1]/1000,0,0,0,0]
23 ♀ Movimiento UR
24   ⚡ If next_pose_list[0]≠0 and next_pose_list[1]≠0
25     ⚡ NO_Deteccion
26       ⚡ MoverJ
27         ⚡ Casa
28   ⚡ Else
29     ⚡ Deteccion_OK
30       ⚡ If next_pose_aux ≠ next_pose_list
31         ⚡ 'Hay detecciones que recoger'
32         ⚡ 'Posicion de la recogida'
33       ⚡ MoverL
34         ⚡ next_pose_list
35           ⚡ 'Variable aux para comparar listas '
36           ⚡ next_pose_aux:=p[next_pose_list[0],next_pose_list[1],next_pose_list[2],0,0,0]
37     ⚡ Else
38       ⚡ 'No hay detecciones que recoger'
39       ⚡ MoverJ
40         ⚡ Casa
41   ⚡ Esperar: 1.0

```

(b) Instrucciones de movimiento del robot según la casuística

Figura 5.6: Ajustes de comunicación XMLRPC en el robot y ejecución de movimientos definidos en el programa

Esta arquitectura permite un sistema modular, robusto y fácilmente escalable a escenarios reales más complejos, sentando las bases para su posible aplicación futura en entornos agrícolas reales o semiautomatizados. Para hacer funcionar el sistema completo y conseguir una correcta integración entre el sistema de visión y el brazo robótico UR, deben seguirse los siguientes pasos de forma ordenada.

1. Conexión de red: Es fundamental asegurar que, tanto el robot como el ordenador que ejecuta el sistema de visión, están conectados a la misma red local y que ambos dispositivos pertenezcan a la misma subred, recomendándose el uso de cables ethernet para una mayor estabilidad. Por este motivo, es necesario comprobar que hay conectividad entre ambos dispositivos (por ejemplo, mediante el uso del

comando ping a la IP del robot desde el ordenador), ya que la comunicación se establece a través del protocolo XML-RPC sobre una dirección IP y puerto determinados.

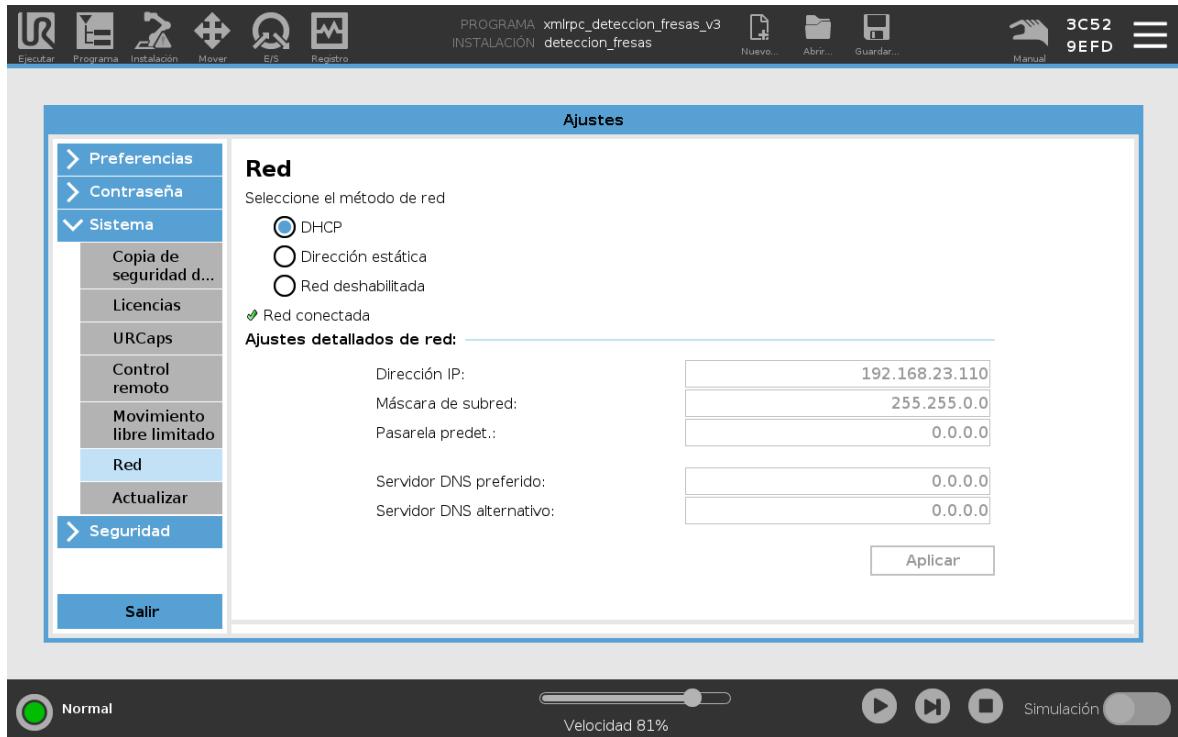
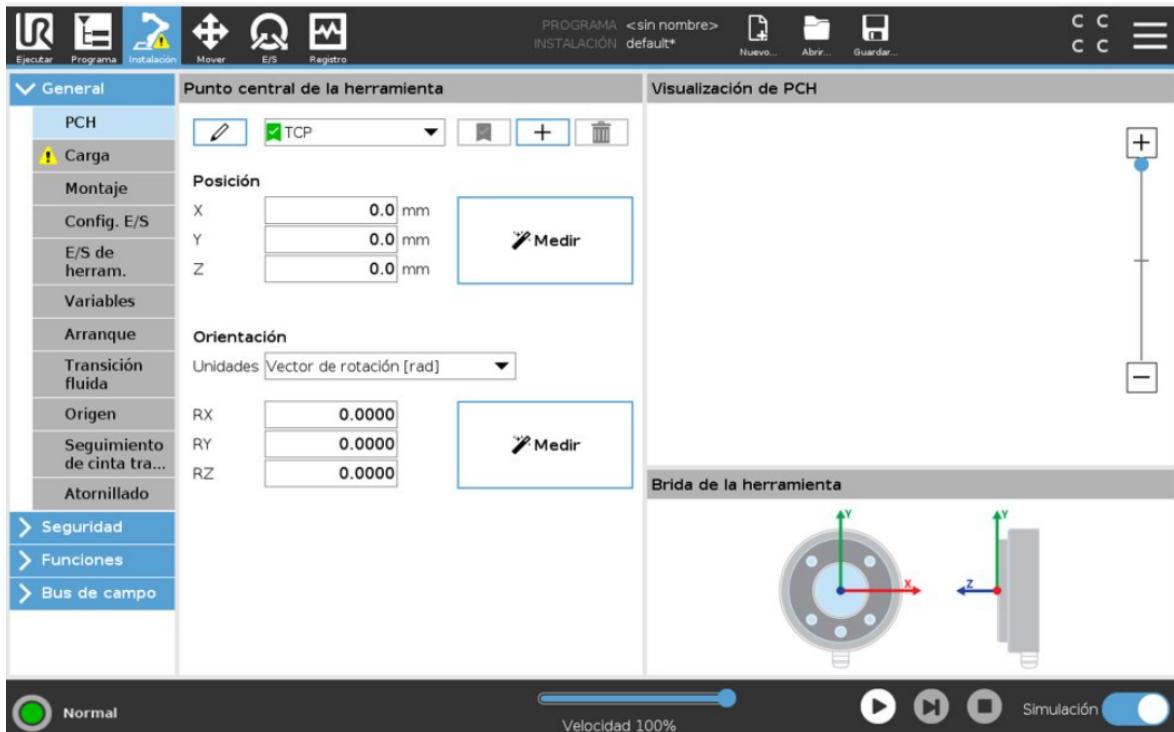
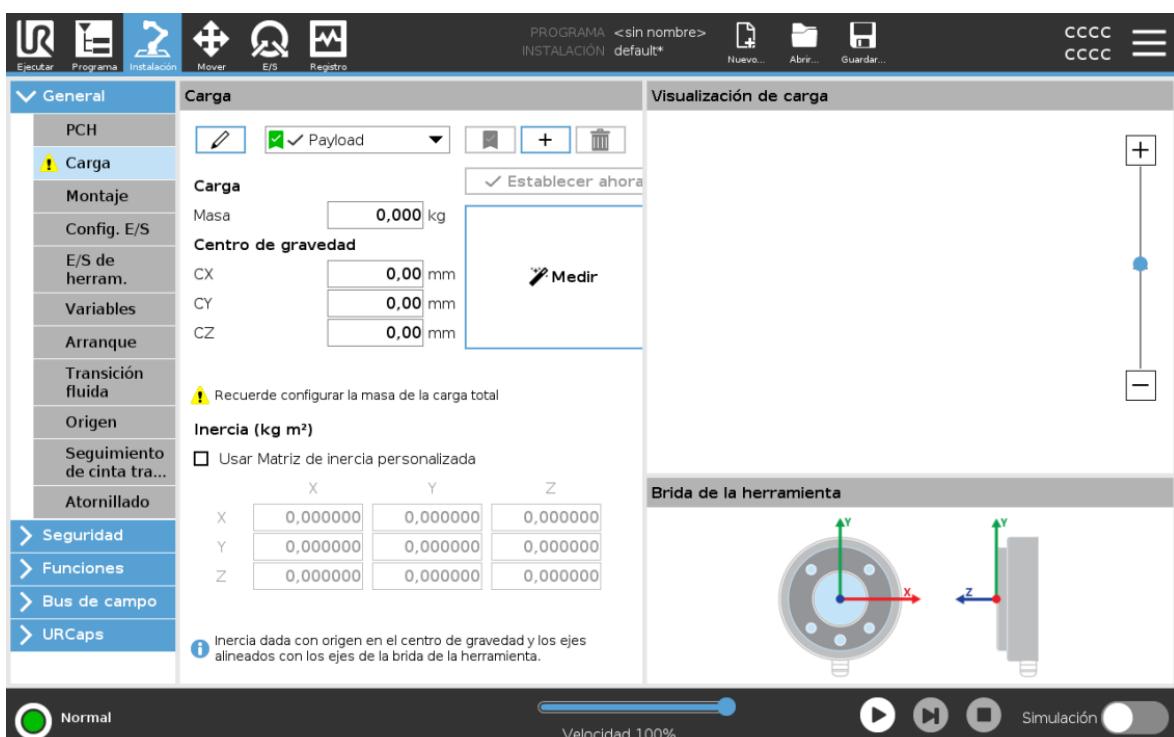


Figura 5.7: Ajustes de red en el robot

2. Definición de la Instalación del robot: Si el robot lleva instalado un efecto final, como una garra o un actuador, es importante configurar correctamente la carga útil, el peso y el centro de gravedad en el software del UR, ya que esto garantiza un funcionamiento seguro y preciso, ajustando los parámetros de control de movimiento y compensación de la cinemática.



(a) Configuración del Punto Central de la Herramienta (PCH)



(b) Configuración de la carga

Figura 5.8: Definición de la Instalación del efecto final en el robot

3. Creación del plano de trabajo: A continuación, se debe definir el plano de trabajo sobre el que se va a operar, en este caso un plano vertical, como se detalló en la Sección 5.1.
4. Lanzamiento del sistema de comunicación: En este punto, se debe ejecutar el script en Python mediante la línea de comando que se muestra a continuación. Para este caso, es el programa *xmlrpc_deteccionfresas.py*¹ el que ejerce de servidor, y tiene que ejecutarse desde la terminal del ordenador para poder ejecutar posteriormente mediante el botón de *play* de la interfaz del robot, el programa *xmlrpc_deteccion_fresas_vertical.urp*², que realiza la solicitud de datos al servidor remoto a través del protocolo XML-RPC, una vez que el servidor esté en marcha.

```
python xmlrpc_deteccionfresas.py --model_def config/yolov3-custom.cfg
--weights_path checkpoints/yolov3_ckpt_99.pth
--class_path data/custom/classes.names --conf_thres 0.85
```

5. Ejecución: Una vez en marcha tanto el programa en el ordenador como en el robot, el sistema detectará automáticamente las fresas presentes en la escena, calculará sus coordenadas tridimensionales y las enviará al robot, que ejecutará el movimiento correspondiente según el programa *xmlrpc_deteccion_fresas_vertical.urp*³, para acercarse a la fresa y situarse encima de esta para una futura recolección.

Para completar la evaluación del prototipo, a pesar de que no se llegó a implementar la acción de cierre o agarre de la pinza dentro del flujo automatizado del sistema, se realizaron pruebas complementarias utilizando una pinza industrial acoplada al robot en el programa de robot *xmlrpc_deteccion_fresas_vertical_pinza.urp*⁴ (Figura 5.9) con el objetivo de verificar la compatibilidad y robustez del programa desarrollado. Estas pruebas permitieron demostrar que el sistema era plenamente funcional y adaptable, y que se podía integrar sin inconvenientes cualquier herramienta de agarre que fuera adecuada para el manejo de fresas, dada su delicadeza. De este modo, se confirma que el sistema de detección, proyección y comunicación con el robot puede extenderse fácilmente para incluir un actuador final destinado a la recolección efectiva del fruto en un entorno real (Figura 5.10).

¹https://github.com/RoboticsURJC/tfg-dcampoamor/blob/main/src/robot/xmlrpc_deteccionfresas.py

²https://github.com/RoboticsURJC/tfg-dcampoamor/blob/main/src/robot/xmlrpc_deteccion_fresas_vertical.urp

³https://github.com/RoboticsURJC/tfg-dcampoamor/blob/main/src/robot/xmlrpc_deteccion_fresas_vertical.urp

⁴https://github.com/RoboticsURJC/tfg-dcampoamor/blob/main/src/robot/xmlrpc_deteccion_fresas_vertical_pinza.urp

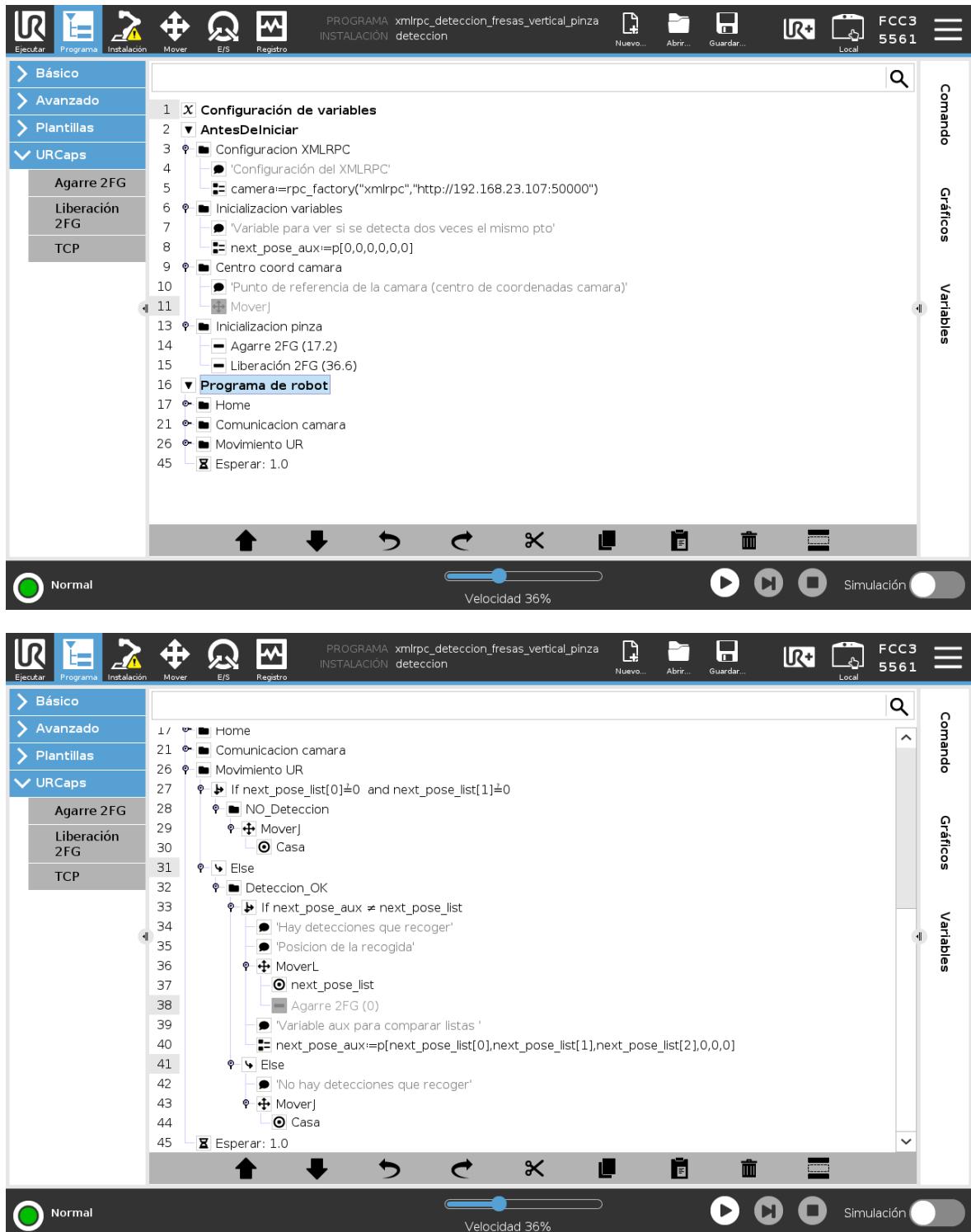


Figura 5.9: Programa `xmlrpc_deteccion_fresas_vertical_pinza.urt` para uso del sistema con efector final

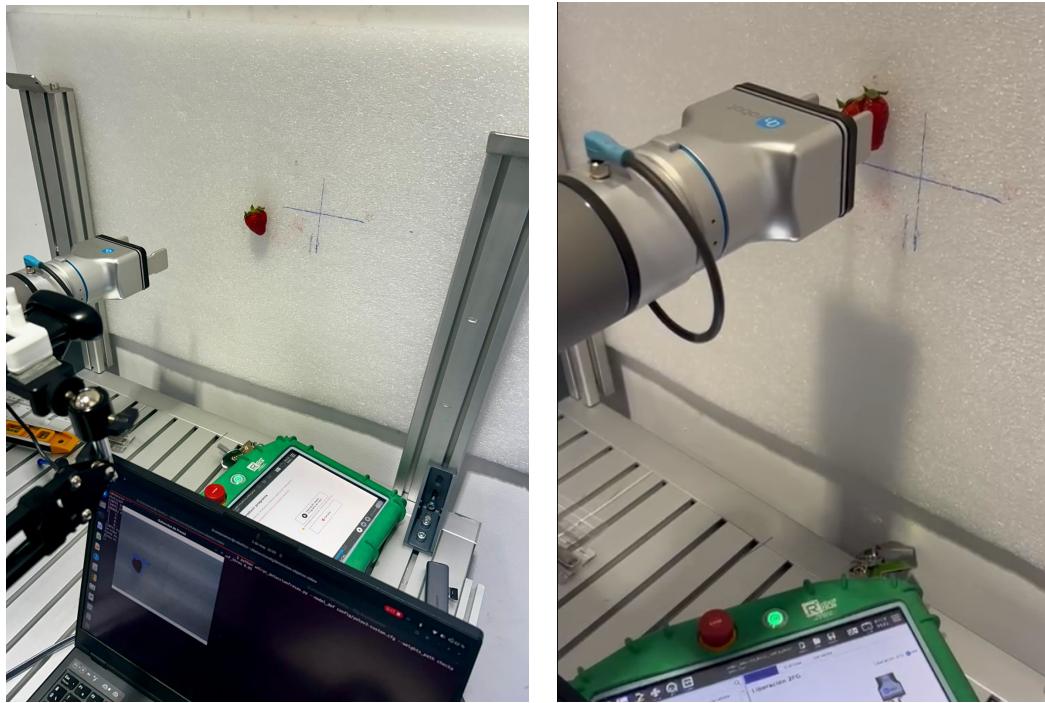


Figura 5.10: Pruebas en el plano vertical de detección con garra integrada

En este capítulo, se ha descrito en detalle el sistema desarrollado, incluyendo los fundamentos técnicos, las herramientas de visión artificial empleadas, y el proceso de integración con el robot colaborativo UR, siendo un sistema diseñado con el objetivo de ser modular, reproducible y eficaz en tareas de detección y recolección de frutos maduros en entornos controlados. En el Anexo II, se presentan los experimentos realizados, donde se detallan las diferentes pruebas llevadas a cabo, los ajustes realizados durante el desarrollo y los resultados obtenidos hasta alcanzar el estado funcional final del proyecto. De igual manera, puede encontrarse mayor documentación gráfica en el repositorio público de GitHub⁵ dedicado al proyecto; como por ejemplo, vídeos del funcionamiento del sistema conjunto, tanto sin efecto final, como el vídeo *Pruebas plano vertical detección múltiple UR5e.mp4*⁶, como vídeos del sistema conjunto con efecto final, tal cual se muestra en *Pruebas plano vertical detección simple garra UR5e.mp4*⁷, entre otros.

⁵<https://github.com/RoboticsURJC/tfg-dcampoamor>

⁶<https://github.com/RoboticsURJC/tfg-dcampoamor/blob/main/img/code/UR/Pruebas%20plano%20vertical%20deteccion%20multiple%20UR5e.mp4>

⁷<https://github.com/RoboticsURJC/tfg-dcampoamor/blob/main/img/code/UR/Pruebas%20plano%20vertical%20deteccion%20simple%20garra%20UR5e.mp4>

Capítulo 6

Conclusiones

En este último capítulo se exponen las conclusiones generales del trabajo realizado, detallando el grado de cumplimiento de los objetivos y requisitos planteados al inicio del proyecto. Asimismo, se presentan algunas posibles líneas de mejora y evolución que podrían ser exploradas en futuros trabajos, con el fin de dar continuidad y ampliar el alcance del sistema propuesto.

6.1. Objetivos y requisitos cumplidos

A continuación, se van a explicar todos los objetivos y requisitos cumplidos en la realización del presente trabajo fin de grado.

6.1.1. Objetivos

Se ha conseguido cumplir con el objetivo principal de este Trabajo Fin de Grado: desarrollar un sistema de visión artificial de bajo coste, basado en técnicas de inteligencia artificial, capaz de detectar fresas maduras y comunicar su posición y distancia a un brazo robótico para su recolección automatizada. Todo ello ha sido probado tanto en entornos simulados como en condiciones reales, demostrando la viabilidad del sistema diseñado.

A su vez, se han cumplido todos los objetivos definidos en la Sección 3.1:

1. Se han investigado las soluciones actuales relacionadas con la detección de frutos mediante visión artificial, encontrándose una gran variedad de propuestas tanto académicas como comerciales, muchas de ellas aún en desarrollo o centradas en otros tipos de cultivos.

2. Se ha seleccionado la técnica de inteligencia artificial más adecuada para el reconocimiento de fresas, optando por el uso de redes neuronales convolucionales (CNN) a través del modelo YOLOv3, debido a su eficiencia y precisión en tareas de detección en tiempo real. Asimismo, se han elegido los componentes hardware necesarios para implementar un sistema de visión robusto y de bajo coste.
3. La técnica escogida se ha optimizado y adaptado para funcionar en la plataforma de trabajo, lo que ha requerido la creación de un dataset específico de imágenes de fresas. Este conjunto de datos fue tratado adecuadamente para garantizar la calidad del entrenamiento y mejorar la precisión del modelo final.
4. Se ha realizado el entrenamiento del sistema con distintos algoritmos de clasificación basados en Machine Learning, evaluando su rendimiento mediante pruebas con imágenes reales. El modelo YOLOv3 ha ofrecido el mejor equilibrio entre velocidad y precisión.
5. Se ha seleccionado el protocolo de comunicación entre el sistema de visión y el robot, implementando un servidor XML-RPC para transmitir de forma efectiva la información de las detecciones. Este protocolo ha sido validado mediante pruebas tanto en simulador como en el entorno real del robot.
6. Se ha dotado al sistema de software capaz de reconocer fresas maduras, calcular su posición en coordenadas del mundo real y estimar su distancia a la cámara, información que se guarda y se transmite al brazo robótico para su uso operativo.
7. Finalmente, se ha probado el sistema completo en situaciones tanto simuladas como reales, comprobando su funcionamiento y eficiencia, y sentando las bases para posibles mejoras y aplicaciones futuras.

6.1.2. Requisitos

También cabe destacar que se han satisfecho todos los requisitos planteados en la Sección I.1:

1. Se ha utilizado como sistema operativo la distribución Ubuntu 22.04 LTS sobre GNU/Linux, cumpliendo así con el requisito de utilizar software libre y con soporte a largo plazo para la ejecución del programa del sistema de visión.

2. Los modelos entrenados han sido optimizados para ajustarse a las limitaciones del hardware utilizado, asegurando su correcto funcionamiento sin necesidad de recursos computacionales de alto rendimiento.
3. El sistema ha demostrado ser capaz de operar en tiempo real, realizando la detección de fresas, el cálculo de distancias y la transmisión de datos al robot con una latencia reducida, adecuada para su uso práctico.
4. Todo el hardware empleado en el desarrollo del sistema de visión ha sido seleccionado con un criterio de bajo coste, haciendo que sea accesible para estudiantes o centros con recursos limitados.
5. La aplicación final es fácilmente reproducible y desplegable tanto en un entorno simulado como en condiciones reales, permitiendo su integración en contextos educativos o de laboratorio sin dificultades técnicas significativas.

6.2. Líneas Futuras

A partir de los resultados obtenidos en este proyecto, se identifican diversas líneas de trabajo que podrían abordarse en el futuro para mejorar y ampliar el sistema desarrollado:

- Entrenamiento con datasets más amplios y variados: Ampliar el conjunto de imágenes utilizado para entrenar el modelo de detección, incluyendo diferentes niveles de maduración, con el fin de mejorar la robustez y generalización del sistema.
- Optimización del rendimiento en hardware embebido: Adaptar el sistema para funcionar en dispositivos aún más limitados (como Raspberry Pi Zero o NVIDIA Jetson Nano), buscando reducir el consumo energético y el coste del sistema.
- Implementación en entornos agrícolas reales: Validar el sistema en condiciones reales de campo, frente a variables como viento, sombra o vegetación densa para comprobar su fiabilidad y utilidad práctica.
- Diseño o integración de una herramienta de recolección versátil: Investigar o desarrollar una pinza robótica compatible con el sistema de visión y adecuada para manipular distintos tipos de frutos con cuidado y precisión, ampliando así el alcance del sistema a otros cultivos más allá de la fresa.

Apéndice I

Metodología

I.1. Requisitos

Para dar respuestas a los objetivos planteados, este trabajo deberá cumplir los siguientes requisitos:

1. Se utilizará *GNU/Linux*, con la distribución *Ubuntu 22.04 LTS*, como sistema operativo, en la plataforma hardware que se encargará de ejecutar el programa del sistema de visión.
2. Los modelos entrenados se deben ajustar a las limitaciones del hardware que ejecutará el programa del sistema de visión.
3. El sistema deberá poder ser utilizado en tiempo real.
4. El *hardware* utilizado para el desarrollo del sistema de visión deberá ser lo suficientemente económico como para ser adquirido por cualquier estudiante.
5. La aplicación debe ser fácilmente reproducible y desplegable tanto en un entorno simulado como en un ambiente educativo real o de laboratorio.

I.2. Competencias

Las competencias adquiridas en el Grado de Ingeniería de Tecnologías Industriales que han sido utilizadas para la realización de este proyecto, se dividen tanto en generales como específicas, y son las siguientes:

1. *Capacidad de organización y planificación: CG02*. Esta competencia ha sido empleada en la consecución de todo el trabajo de fin de grado, y queda reflejada

tanto en las reuniones semanales o quincenales con el tutor responsable de este trabajo como en la wiki de GitHub¹ dedicada al proyecto, que refleja los avances y la organizacización llevada a cabo.

2. *Conocimiento de una lengua extranjera: CG04.* Esta competencia ha sido utilizada a la hora de buscar toda clase de información para poder elaborar este proyecto, ya que se han utilizado documentos publicados en, al menos, una lengua extranjera, como lo es el inglés.
3. *Resolución de problemas: CG06.* Dado el nivel de conocimiento necesario en ciertas materias como la inteligencia artificial o la visión artificial, ha sido empleada para poder resolver los diversos inconvenientes que afrontar durante las pruebas prácticas relacionadas con estas áreas del trabajo.
4. *Uso de internet como medio de comunicación y como fuente de información: CG21.* Para poder elaborar este trabajo de fin de grado, ha sido necesario emplear esta competencia para buscar la información necesaria y poder completar principalmente los capítulos 1 y 2.
5. *Conocimientos de informática relativos al ámbito de estudio: CG24.* Esta competencia ha sido empleada a la hora de desarrollar y programar la aplicación sobre la que trata este trabajo.
6. *Conocimientos básicos sobre el uso y programación de los ordenadores, sistemas operativos, bases de datos y programas informáticos con aplicación en ingeniería: CE3.* Para poder desarrollar la aplicación se tuvo que emplear esta competencia a la hora de llevar a cabo la partición del disco duro en el ordenador y poder instalar la versión de *Ubuntu 22.04.5 LTS (Jammy Jellyfish)*.
7. *Conocimientos sobre los fundamentos de automatismos y métodos de control: CE13.* Esta competencia se refleja en la programación de la toma de decisiones automática, donde el sistema ajusta las operaciones en función de los resultados obtenidos del sistema de visión, así como el trabajo sincronizado de varios dispositivos (cámara y robot) y la comunicación entre estos.
8. *Conocimiento de los principios de regulación automática y su aplicación a la automatización industrial: CE32.* Esta competencia se emplea una vez que el sistema de visión identifica el grado de maduración de la fresa, ya que el sistema

¹<https://github.com/RoboticsURJC/tfg-dcampoamor/wiki>

toma la decisión de recolectar o no el fruto, ajustando los actuadores o robots para realizar la tarea de forma precisa, dependiendo de las condiciones detectadas. Así mismo, se emplea la regulación automática en la optimización del proceso ajustando el comportamiento del robot en función del estado de la maduración de la fresa, maximizando la velocidad de la aplicación y su precisión.

9. *Capacidad para diseñar sistemas de control y automatización industrial: CE33.*

Gracias a esta competencia se ha podido estructurar el sistema completo, incluyendo la parte de visión artificial, el procesamiento de imágenes, la toma de decisiones y el control del brazo robótico, integrando todos los datos en tiempo real y llevando a cabo operaciones de monitoreo y seguimiento de la aplicación y sus operaciones en remoto, tal y como se realizó en la etapa de pruebas.

Por otro lado, las competencias adquiridas con el desarrollo de este trabajo fin de grado, y que aparecen descritas en la guía docente de la propia asignatura, son las siguientes:

1. *Capacidad de análisis y síntesis: CG01.* Esta competencia se adquiere debido a la necesidad de la búsqueda y recopilación de información necesaria para el desarrollo del proyecto.

2. *Razonamiento crítico: CG11.* En relación con la competencia adquirida CG01, el razonamiento crítico se emplea a la hora de filtrar, seleccionar y decidir qué de toda la información obtenida es válido, cómo se puede utilizar y cómo ajustarlo y añadirlo al contenido del proyecto.

3. *Aprendizaje autónomo: CG13.* Esta competencia ha sido adquirida dada la necesidad de adaptarse al marco técnico en cuál se desarrolla este proyecto, su complejidad, y la constante actualización y mejoras de las técnicas que pueden ser empleadas para el desarrollo del mismo.

4. *Adaptación a nuevas situaciones: CG14.* Esta competencia se adquiere gracias a la aplicación de la competencia adquirida *CG13 Aprendizaje autónomo*, justificada anteriormente, y que se puede ver reflejada en el proyecto.

5. *Capacidad de aplicar los conocimientos teóricos en la práctica: CG20.* La parte empírica y práctica del proyecto refleja la adquisición de esta competencia.

6. *Capacidad para entender el lenguaje y propuestas de otros especialistas: CG22.*

Esta competencia se ha visto reflejadas dado que, en relación con el resto de competencias adquiridas, sobre todo con las competencias *CG01 Capacidad de análisis y síntesis* y *CG11 Razonamiento crítico*, se ha tenido la necesidad de consulta, recopilación y filtrado de información y metodologías científicas existentes para poder desarrollar la aplicación del proyecto.

I.3. Metodología

Para lograr los objetivos descritos anteriormente, se optó por emplear el método DMADV (Definir, Medir, Analizar, Diseñar, Verificar), perteneciente a la metodología *Six Sigma* (Figura I.1).

Se decidió optar por esta metodología dado que la aplicación desarrollada en este trabajo está basada en otros proyectos y estudios similares ya existentes, y esta metodología está diseñada para desarrollar procesos o productos nuevos o significativamente mejorados, independientemente de si partes de cero o si te basas en conocimientos previos, y esta información recopilada fue utilizada como punto de partida en las fases iniciales de la metodología. A continuación, se detallan las fases del método DMADV y cómo han sido aplicadas en el desarrollo de este proyecto.

- Definir: En esta fase se establecen como objetivos del proyecto la identificación de fresas maduras mediante un sistema de visión artificial y su integración con el brazo robótico encargado de su recolección.
- Medir: Tomando como referencia el estado del arte existente, se llevó a cabo la selección de la tecnología (hardware y software necesarios) y las métricas existentes para poder considerar que las detecciones son aceptables y suficientes, basándose así el trabajo en proyectos y estudios que han demostrado ser eficaces, constituyendo un proyecto robusto y eficiente y ayudando a evitar problemas comunes ya que se sostiene sobre conocimientos adquiridos de proyectos similares testados.
- Analizar: En esta etapa se llevaron a cabo pruebas con diferentes algoritmos y sistemas para detección de objetos, tanto en imágenes como en vídeo a tiempo real, para poder determinar cuál ofrecía un mejor rendimiento en precisión y

velocidad y se ajustaba a los requisitos que presentaba el hardware disponible para el desarrollo del proyecto.

- Diseñar: En base a los resultados obtenidos en la fase de análisis, se configuró el sistema de reconocimiento por visión incluyendo el hardware, previamente seleccionado, y el software, compuesto por los códigos y entornos necesarios para poder desarrollarlo y ejecutarlo, y su integración con el robot.
- Verificar: Finalmente, se llevaron a cabo pruebas para verificar que el sistema funcionaba y cumplía con los requisitos definidos, validándose la precisión en la identificación de las fresas maduras con distintas intensidades de luz ambiente y la comunicación y el correcto funcionamiento del brazo robótico.

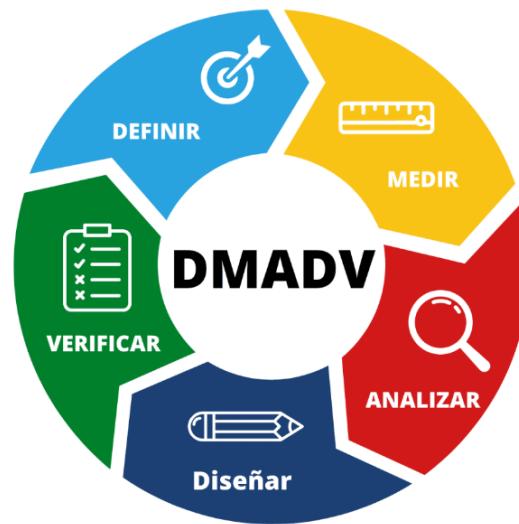


Figura I.1: Ciclo de la metodología DMADV

I.4. Habilidades desarrolladas

Además de todas las competencias descritas en la Sección I.2, a lo largo del desarrollo de este Trabajo Fin de Grado se han adquirido y reforzado numerosas habilidades y conocimientos, entre los cuales cabe destacar:

- Se ha adquirido una sólida capacidad de organización y planificación, debida a la estructuración del proyecto, el seguimiento mediante reuniones periódicas con el tutor y la documentación detallada de los avances en la plataforma GitHub.

- Se ha mejorado notablemente la capacidad de búsqueda, análisis y síntesis de información técnica en inglés, utilizando documentación científica y recursos especializados en visión artificial, inteligencia artificial y robótica.
- Se ha fortalecido la competencia en resolución de problemas, especialmente al afrontar desafíos técnicos relacionados con el entrenamiento de modelos de Machine Learning, la calibración del sistema de visión o la integración del sistema con el robot.
- Se ha desarrollado la habilidad de utilizar Internet como fuente de información académica y técnica, contrastando distintas metodologías y enfoques para fundamentar las decisiones adoptadas en el diseño del sistema.
- Se han adquirido conocimientos avanzados en programación en Python, además de conocimientos básicos en el uso de bibliotecas específicas como OpenCV, OpenGL, PyTorch y en herramientas de comunicación como XML-RPC.
- Se ha reforzado el uso y conocimiento sobre el sistema operativo GNU/Linux, ya que, partiendo desde la instalación y configuración de Ubuntu 22.04 LTS y la gestión de particiones del disco duro del ordenador utilizado en la elaboración del proyecto, se ha hecho uso de la terminal y de conexiones remotas vía SSH a partir de esta.
- Se han desarrollado competencias en la recogida, tratamiento y etiquetado de datos para el entrenamiento de modelos de inteligencia artificial, construyendo un dataset propio de imágenes de fresas.
- Se ha adquirido la habilidad de entrenar y optimizar modelos de aprendizaje supervisado adaptados a plataformas con recursos limitados, asegurando su funcionamiento en tiempo real.
- Se ha aprendido a integrar un sistema de visión con un brazo robótico, automatizando la toma de decisiones a partir de los datos recogidos por la cámara.
- Se ha reforzado el conocimiento de los fundamentos de la automatización industrial y la regulación automática, aplicados a la ejecución precisa de tareas por parte del robot en función del estado de maduración detectado en cada fresa.
- Finalmente, se ha adquirido experiencia en la redacción técnica y científica, mediante el uso de LaTeX para la elaboración de la memoria del proyecto, cuidando la estructura, el lenguaje y el formato exigido en un entorno académico.

Apéndice II

Experimentos

En este apéndice se recogen las distintos experimentos que se han llevado a cabo durante el desarrollo del proyecto. Estas pruebas han sido fundamentales para verificar el correcto funcionamiento del sistema de reconocimiento de fresas maduras y su comunicación con el brazo robótico, permitiendo así alcanzar los objetivos definidos en fases anteriores del trabajo.

II.1. Detección con YOLOv3 y TensorFlow

Dada la finalidad del proyecto, se requería que la detección de objetos se diera en tiempo real, por lo que se buscó información sobre YOLO, un sistema de código abierto que permitía esto a partir de una red neuronal convolucional para detectar objetos en imágenes y vídeo. De este modo se iniciaron las pruebas pertinentes para la selección del algoritmo de detección y de las bibliotecas a utilizar.

II.1.1. Pruebas con imágenes

En primer lugar, se creó un entorno de Anaconda para poder probar la detección de objetos en imágenes utilizando Tensorflow mediante el repositorio *deteccion_objetos*¹, que estaba basado en la configuración *faster rcnn resnet101 coco* como modelo de detección de objetos. Se llevó a cabo el etiquetado de imágenes, en este caso de tigres, mediante la herramienta labelImg², y se prepararon las carpetas y archivos de configuración correspondientes para poder llevar a cabo el entrenamiento del modelo, siguiendo los pasos indicados en el repositorio; y utilizando una distribución de las imágenes utilizadas para el aprendizaje del modelo y su uso en la detección aproxima-

¹https://github.com/puigalex/deteccion_objetos

²<https://github.com/HumanSignal/labelImg>

damente del 70:30 (73 % datos de entrenamiento y 27 % datos de prueba)(Cuadro II.1), a partir de los cuales se entrenó ese 70 % con uno de los algoritmos y los respectivos parámetros escogidos, y medimos su rendimiento usando el 30 % restante de los datos.

Imágenes usadas en entrenamiento	Imágenes usadas en test	Número total de imágenes
594	218	812

Cuadro II.1: Distribución de las imágenes utilizadas para el entrenamiento del modelo

Se entrenó este modelo hasta que se observó que la pérdida estaba por debajo de 1, considerando que esta pérdida no era alta, y que no existían demasiadas fluctuaciones, deteniendo este entrenamiento a los 1400 pasos, a pesar de que este entrenamiento estaba programado para llegar hasta los 20000. Esto supuso que se tuviera que utilizar el último checkpoint disponible, en este caso el del paso 1337, para convertirlo en un modelo final y de esta manera poder generar predicciones, utilizando imágenes de diferentes tamaños.

Una vez convertido el checkpoint en un modelo final, se procedió a realizar las primeras pruebas de detección en imágenes de este modelo, comprobando su capacidad para detectar correctamente los tigres en este caso, y se evaluó visualmente los resultados obtenidos en algunos ejemplos mostrados en la Figura II.1.

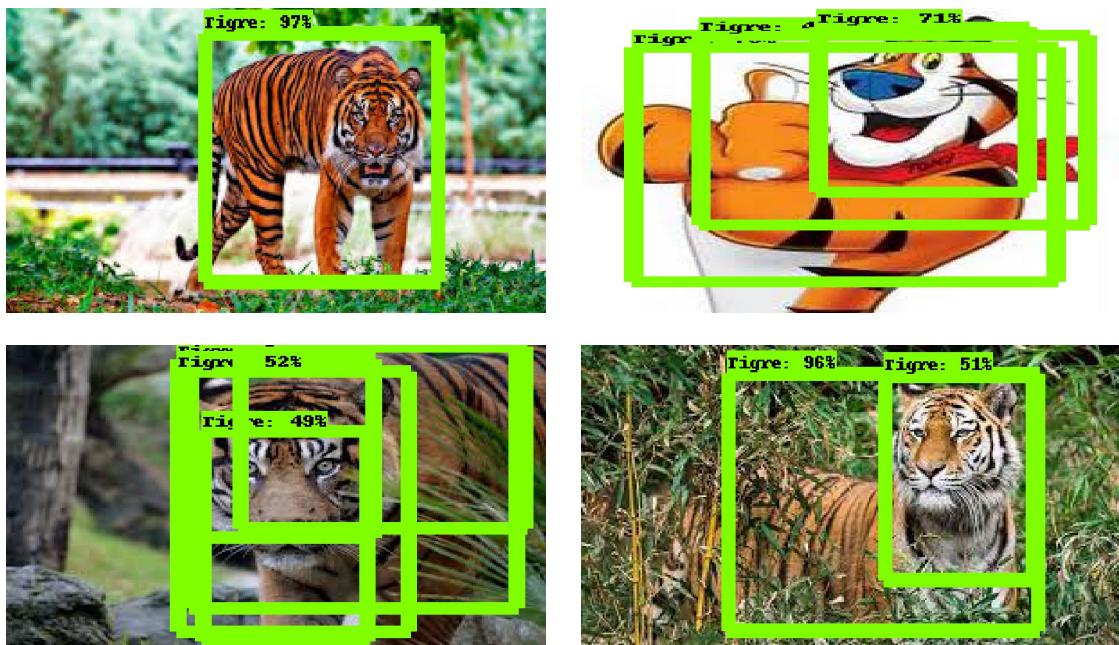


Figura II.1: Resultado de la detección en imágenes con TensorFlow

Tras los resultados obtenidos en las imágenes utilizadas para esta primera prueba, se decidió llevar a cabo un nuevo proceso de entrenamiento a partir del último checkpoint disponible, con el objetivo principal de comprobar si, aumentando el número de pasos de entrenamiento, se lograba una mejora significativa tanto en la disminución del valor de pérdida, como en el incremento del porcentaje de confianza en las detecciones realizadas. Así, se retomó el entrenamiento desde el checkpoint del paso 1337, extendiéndose en esta segunda ocasión hasta el paso 2945, momento en el cual se optó por detener manualmente el proceso al observarse una estabilización progresiva en los valores de pérdida, y siendo el último checkpoint generado el correspondiente al paso 2877, obteniéndose en este punto un valor de pérdida de tan solo 0.222, notablemente inferior al registrado en el primer intento.

A continuación, se procedió a ejecutar nuevamente el programa sobre las mismas imágenes de prueba de tigres empleadas en la primera serie de tests, lo que permitió realizar una comparación directa entre ambos modelos, y observar que en esta segunda ejecución existía una clara mejora en la calidad de las detecciones, tanto en términos de mayor porcentaje de confianza como en la precisión de los cuadros delimitadores sobre los objetos detectados (ver Figura II.2).

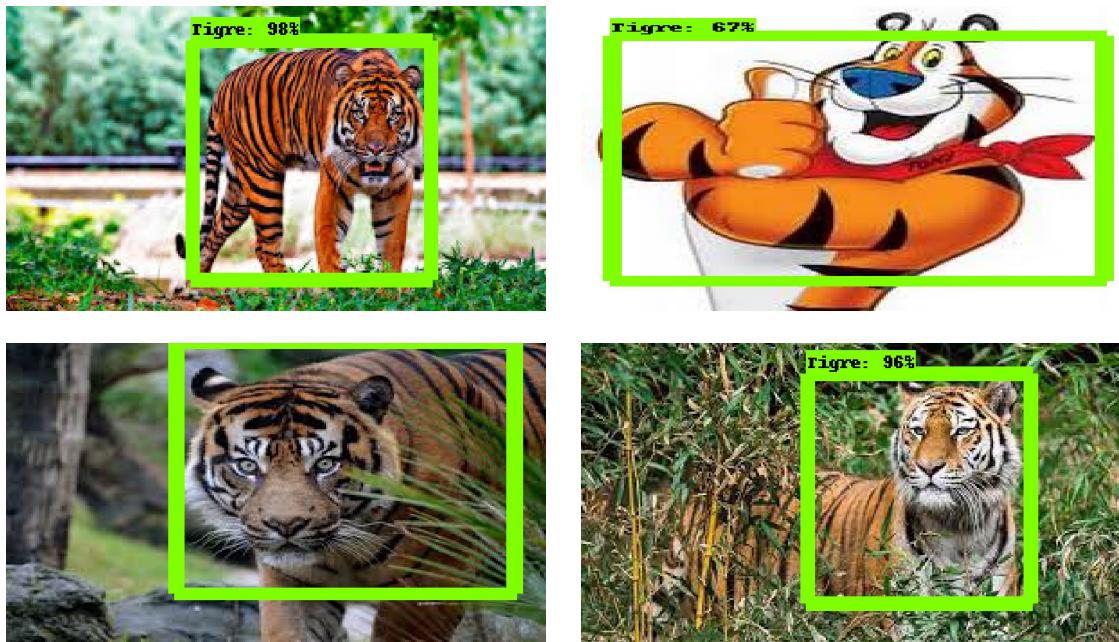


Figura II.2: Resultado del reentrenamiento de la detección en imágenes con TensorFlow

Después de llevar a cabo estas pruebas con imágenes de tigres, se comprobó que el modelo funcionase también con fresas, por lo que, a través de la página Kaggle, se obtuvo un dataset de 262 frutas³, de las cuales únicamente se utilizó el archivo de las fresas, que contenía 1002 imágenes.

Una vez descargado el archivo, se comenzó a etiquetar una a una las imágenes mediante la herramienta *labelImg* para obtener los archivos xml, tal y como se había hecho con el ejemplo anterior de los tigres, y antes de terminar de etiquetar el dataset entero, se probó este modelo utilizando las primeras 405 imágenes etiquetadas siguiendo una distribución de estas del 80:20 para su entrenamiento y usando el checkpoint guardado en el paso 3490 para congelar el modelo, y así poder utilizar varias imágenes aún por etiquetar para probarlo, obteniendo un resultado satisfactorio en cuanto a la detección y su confianza, tal y como se puede observar en la Figura II.3.

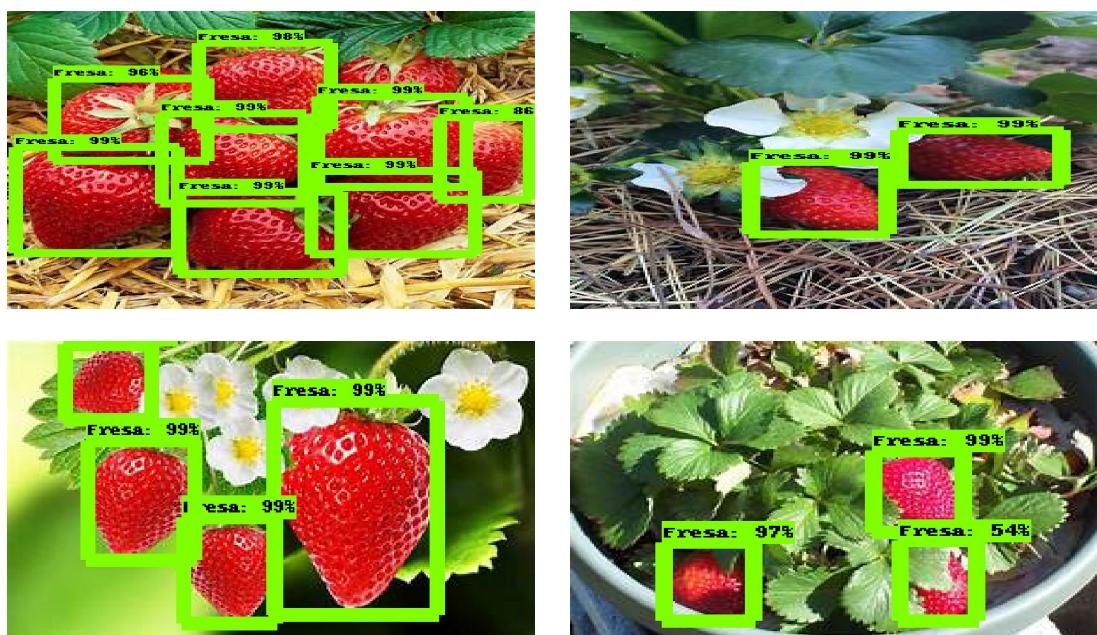


Figura II.3: Pruebas de detección de fresas en imágenes con TensorFlow

Tras haber conseguido la detección de fresas en imágenes estáticas utilizando TensorFlow, el siguiente paso dentro del desarrollo del sistema consistió en extender las pruebas a la detección en vídeo en tiempo real, por lo que, se procedió a evaluar distintos modelos de detección de objetos ya preentrenados sobre conjuntos de datos de referencia, lo que permitió llevar a cabo una comparación de estos diferentes modelos o sistemas bajo las mismas condiciones iniciales sin necesidad de realizar un nuevo entrenamiento desde cero.

³<https://www.kaggle.com/datasets/aelchimminut/fruits262>

II.1.2. Pruebas con vídeo en tiempo real

Para la realización de estas pruebas, se utilizó tanto la cámara web integrada del ordenador portátil como una imagen previamente seleccionada, para que, de esta manera pudieran observarse las diferencias entre los modelos tanto en la detección en vídeo como en la detección en imágenes, y poder valorar qué modelo de los tres distintos probados ofrecería mejores prestaciones en términos de precisión, velocidad de procesamiento y robustez frente a las condiciones reales de trabajo (ver figuras II.4, II.5 y II.6).

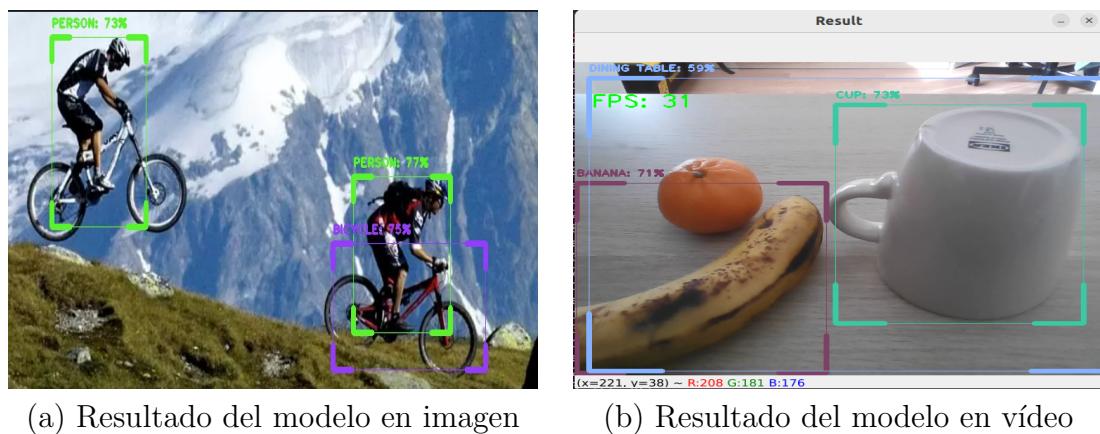


Figura II.4: Modelo ssd_mobilenet_v2_320x320_coco17_tpu-8

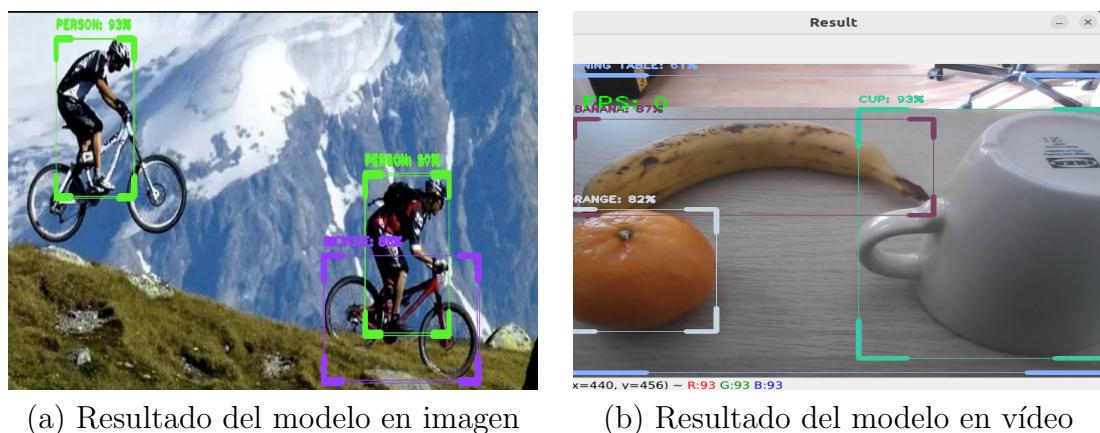


Figura II.5: Modelo efficientdet_d4_coco17_tpu-32



(a) Resultado del modelo en imagen

(b) Resultado del modelo en vídeo

Figura II.6: Modelo faster_rcnn_resnet50_v1_640x640_coco17_tpu-8

Después de haber llevado a cabo estas pruebas con los modelos de detección de objetos `ssd_mobilenet_v2_320x320_coco17_tpu-8`, `efficientdet_d4_coco17_tpu-32` y `faster_rcnn_resnet50_v1_640x640_coco17_tpu-8`, y tras valorar que, el principal uso del modelo en la aplicación final sería la de llevar a cabo detecciones en tiempo real con una cámara, se escogió el modelo `ssd_mobilenet_v2` para proseguir con los experimentos.

La elección de este modelo, incluso por delante de cualquiera de los otros dos, se llevó a cabo a pesar de tener menor precisión y calidad de detección, puesto que destacaba principalmente por su elevada velocidad de procesamiento y su bajo consumo de recursos, gracias a su arquitectura ligera basada en MobileNetV2 y su tamaño de entrada reducido, haciéndolo especialmente adecuado para aplicaciones en tiempo real sobre hardware con capacidades limitadas como el implementado.

Una vez escogido el modelo, para poder llevar a cabo la detección de fresas, era necesario entrenarlo desde cero, para lo que se utilizó de guía el repositorio *real_time_object_detection_cpu*⁴, creando y activando un nuevo entorno de Anaconda, en el cual se instalaron los paquetes y librerías necesarios para ello, junto al Object Detection API de TensorFlow y junto con Jupyter Notebook⁵, un entorno computacional interactivo basado en web para crear cuadernos que contienen código en vivo, ecuaciones, visualizaciones y texto narrativo.

⁴https://github.com/haroonshakeel/real_time_object_detection_cpu/blob/main

⁵<https://jupyter.org>

Completada la configuración del entorno, la instalación de todos los componentes, y el entrenamiento del modelo, se realizó una primera prueba de detección utilizando el modelo entrenado para comprobar si funcionaba, obteniendo las primeras predicciones en tiempo real sobre vídeo con fresas reales. Estas primeras detecciones sirvieron de base para la batería de pruebas en las cuales se variaba tanto el número de fresas como las condiciones de luz, para poder conocer en qué condiciones se obtenía un mayor porcentaje de confianza en la detección (Figura II.7).

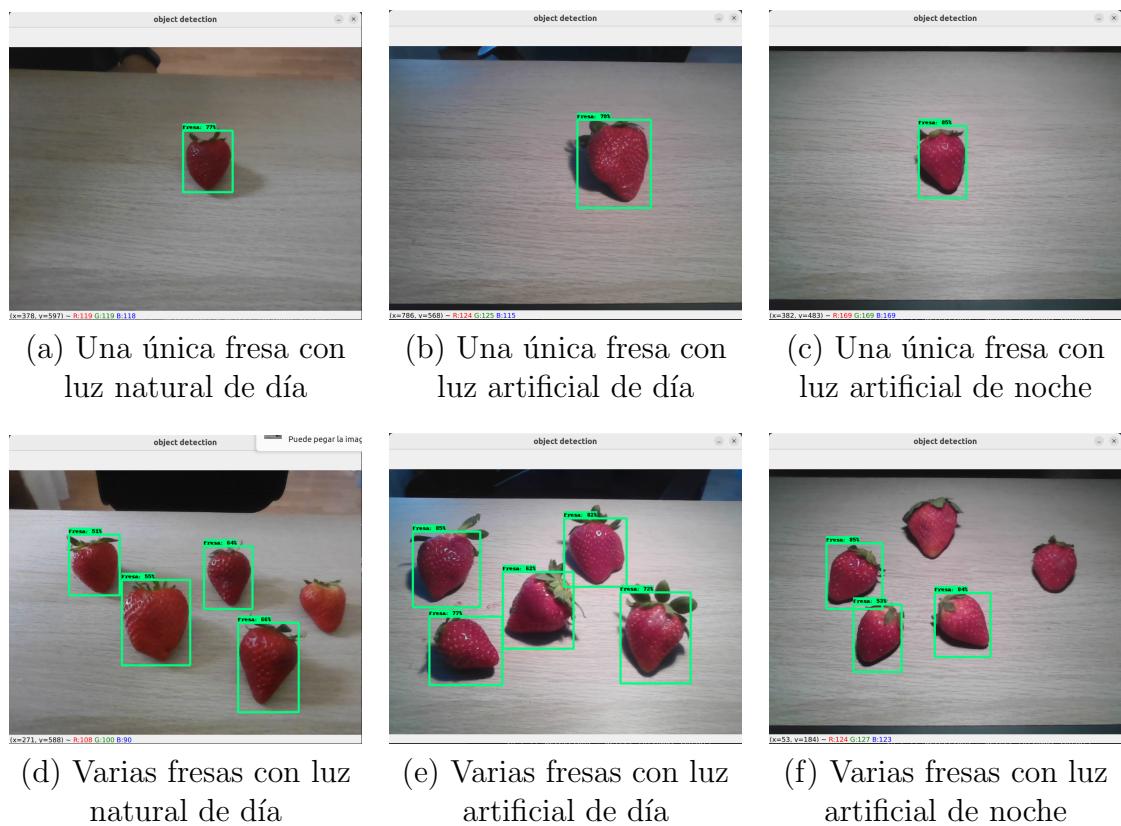
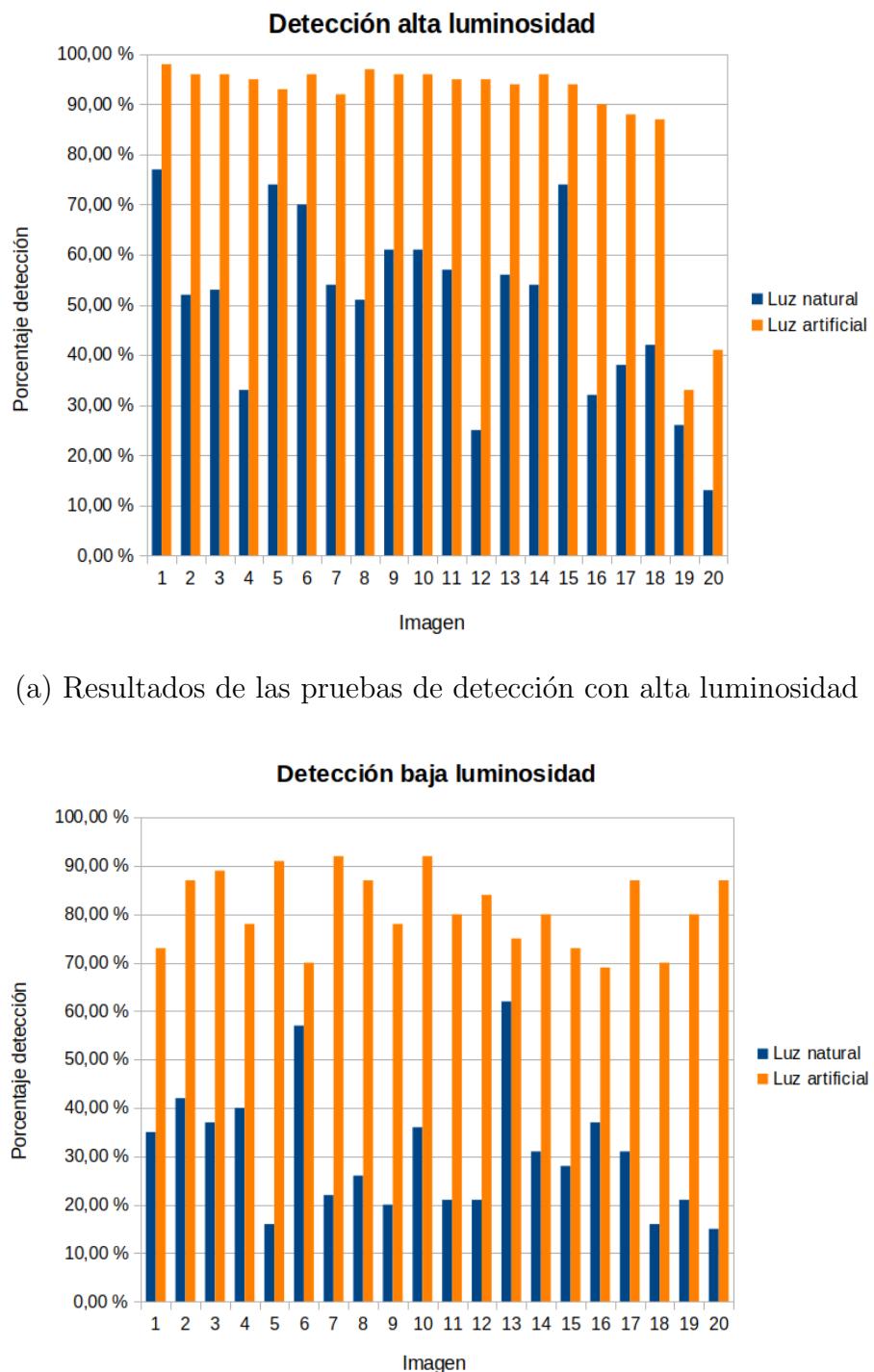
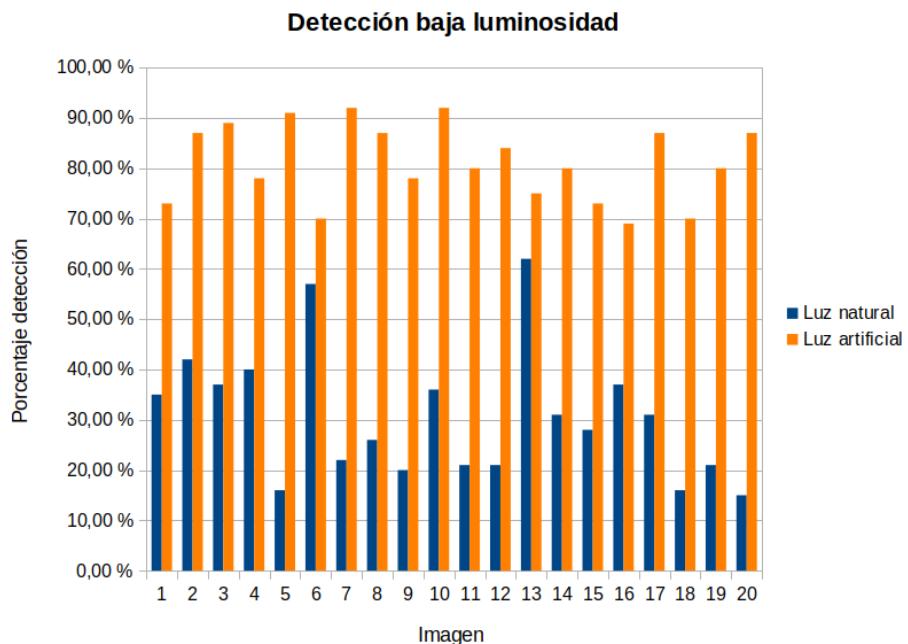


Figura II.7: Detección de fresas en webcam con TensorFlow con modelos no preentrenados (ssd mobilenet v2 320x320)

Después de verificar la viabilidad y funcionamiento de estas pruebas, y de detectar en los resultados que, con luz artificial en condiciones de alta luminosidad existía un mayor porcentaje de confianza en las detecciones que con luz natural y baja luminosidad, tal y como se puede apreciar en las figuras II.8 y II.9, se modificó el programa de detección para obtener más datos sobre estas detecciones y dotar al programa de nuevas funcionalidades.



(a) Resultados de las pruebas de detección con alta luminosidad



(b) Resultados de las pruebas de detección con baja luminosidad

Figura II.8: Gráficas de la confianza de detección obtenida en las pruebas según la luminosidad para el modelo ssd mobilenet v2

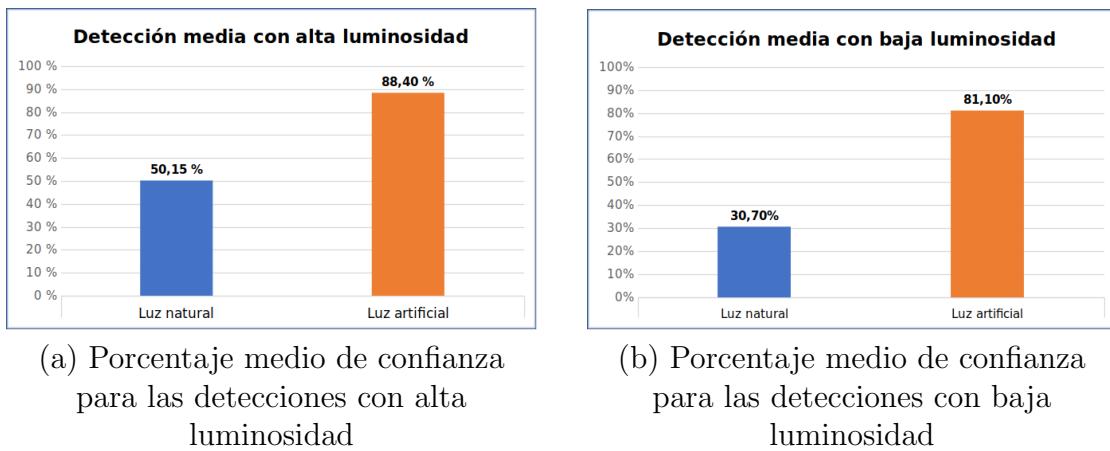


Figura II.9: Gráficas de la media de los porcentajes de confianza obtenidos en las pruebas de detección según la luminosidad para el modelo ssd mobilenet v2

Estas modificaciones incluían la instrucción mediante la cual se dejase de captar lo que se podía ver por la cámara del ordenador y se cerrase la ventana emergente correspondiente al finalizar la ejecución el programa, a la que también se le cambió el nombre por *strawberry detection*. También se calcularon las coordenadas del punto central del recuadro de la detección, y se añadió el cálculo de los FPS (fotogramas por segundo) en tiempo real en la ventana (ver Figura II.10); midiendo la velocidad de procesamiento de los cuadros, lo cuál era útil para comparar entre las distintas condiciones de detección, ya que un FPS más alto indicaba que se estaban procesando más cuadros por segundo, lo que es deseable para aplicaciones en tiempo real.

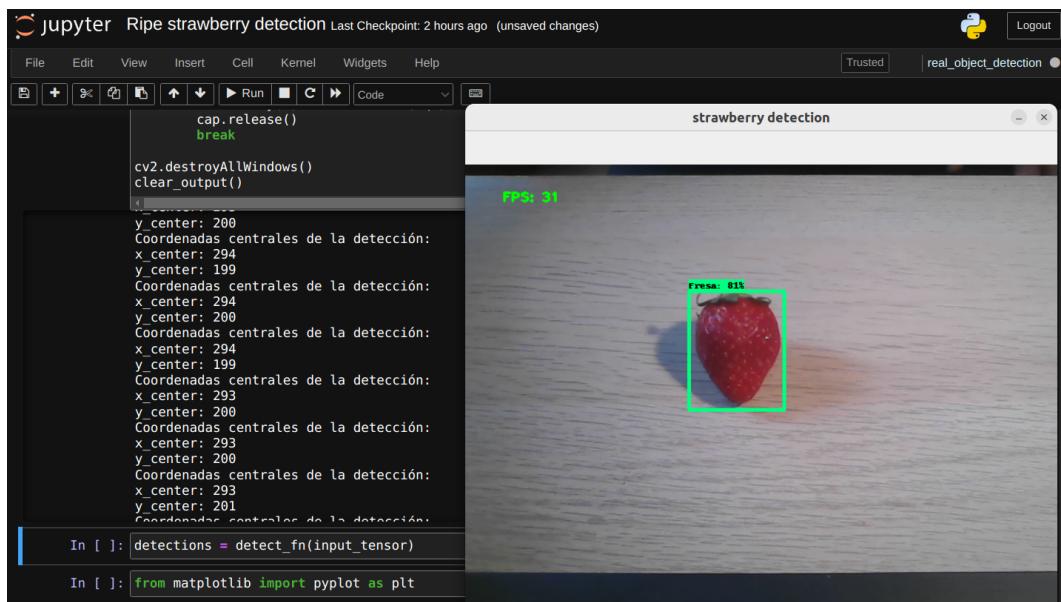


Figura II.10: Detección de fresas en Jupyter Notebook

A pesar de que Jupyter Notebook ofrecía un entorno interactivo y muy útil, y de haber llevado a cabo todas las pruebas anteriormente mencionadas, no era recomendable utilizarlo como entorno de ejecución para aplicaciones estables conectadas a robots, puesto que su diseño está orientado principalmente a tareas de análisis, visualización y prototipado, donde el usuario interactúa continuamente con el entorno mediante la ejecución manual de celdas, suponiendo una limitación importante para sistemas robóticos.

Una de las principales desventajas de Jupyter en este contexto es su modelo de ejecución no lineal, ya que, a diferencia de un script en Python, donde el flujo de ejecución es siempre secuencial y controlado, en un *notebook* es posible ejecutar fragmentos de código en cualquier orden, pudiendo provocar desincronización en las variables del programa y errores difíciles de detectar, especialmente críticos en aplicaciones donde se controla hardware, se toman decisiones en tiempo real o se actúa sobre el entorno físico.

Además, a pesar de que Jupyter Notebook utiliza el lenguaje Python y puede ejecutar cualquier código compatible, su arquitectura está basada en un servidor web local que muestra la interfaz en un navegador, lo que implica que, aunque no necesita conexión a Internet, sí requiere iniciar un servidor HTTP en el sistema local, por lo que sería necesario implementar manualmente un servidor adicional dentro del propio *notebook*. Esto introduce una complejidad innecesaria y un entorno frágil, ya que tanto el servidor adicional como el entorno Jupyter deben mantenerse activos, y cualquier error o bloqueo en una celda puede interrumpir toda la operación.

Por todas estas razones, aunque Jupyter Notebook puede ser muy útil durante las fases iniciales del desarrollo para validar algoritmos de visión o procesado de datos, la implementación definitiva del sistema se realizó mediante scripts de Python, permitiendo un mayor control sobre el flujo de ejecución, una integración más sencilla en sistemas de control y producción, y una mayor robustez operativa, aspectos esenciales en el desarrollo de aplicaciones robóticas fiables.

II.2. Detección con YOLOv3 y PyTorch

Para poder comprobar las diferencias en un ejemplo práctico a la hora de detectar objetos entre PyTorch y TensorFlow, y de esta manera poder escoger una de las dos bibliotecas para el desarrollo del modelo de aprendizaje automático y aprendizaje profundo en este proyecto, se decidió crear de nuevo un entorno de Anaconda y probar a detectar objetos en imágenes utilizando PyTorch.

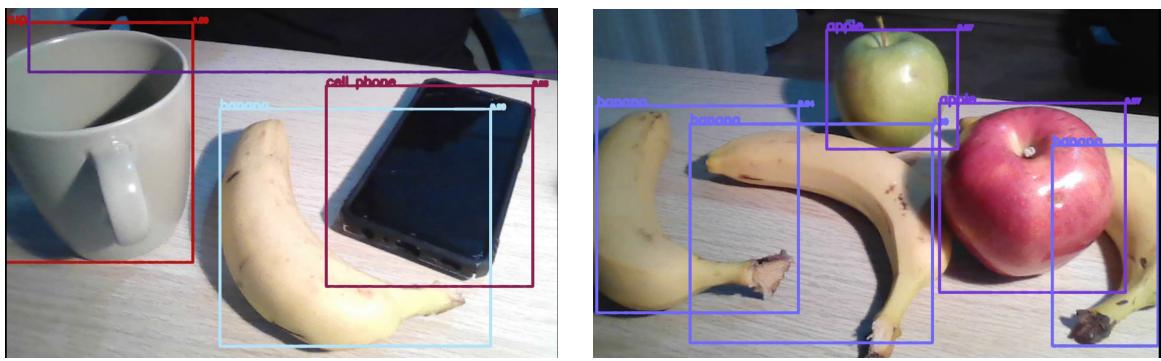
II.2.1. Pruebas con modelos preentrenados

Después de realizar la lectura *You Only Look Once: Unified, Real-Time Object Detection*[Redmon et al., 2016], se replicó lo que se exponía en dicho artículo con la cámara integrada del ordenador portátil, mediante un programa en Python y usando la librería OpenCV mediante la biblioteca Pytorch. Este programa, partiendo del *feed* de la propia webcam, descomponía el vídeo en imágenes o cuadros, alimentando a la red neuronal (en este caso YOLOv3), que recibía esta detección y se procesaba con OpenCV, dibujando los recuadros o *bounding box* alrededor de los objetos que se detectaban en vivo.

Para ello, se clonó el repositorio *deteccion-objetos-video*⁶ basado en el proyecto *PyTorch-YOLOv3*⁷ para correr detección de objetos sobre vídeo y se siguieron los pasos detallados en el archivo README. Una vez instalado todo, se probó a utilizar con varios objetos, y posteriormente con varias frutas simultáneamente, para verificar que el modelo las diferenciaba correctamente y las detectaba, tal y como se muestra en la Figura II.11.

⁶<https://github.com/puigalex/deteccion-objetos-video>

⁷<https://github.com/eriklindernoren/PyTorch-YOLOv3>



(a) Prueba detección de objetos con Pytorch

(b) Prueba detección de frutas con Pytorch

Figura II.11: Detección con Pytorch

II.2.2. Entrenamiento del modelo

Conociendo las limitaciones de Jupyter Notebook, habiendo escogido llevar a cabo la elaboración del programa de detección en Python, y una vez comprobada la diferencia entre TensorFlow y Pytorch, se sustituyó TensorFlow por PyTorch como biblioteca de desarrollo del modelo de visión, basándose en que PyTorch ofrece una sintaxis más intuitiva y cercana a la programación en Python puro, lo que facilita su integración con scripts que deben ejecutarse en tiempo real junto con otros módulos, como los encargados de la comunicación con el robot, además de que PyTorch presenta una curva de aprendizaje más suave para depuración y prototipado rápido, y proporciona una mayor facilidad a la hora de exportar modelos, optimizarlos o ajustarlos dinámicamente durante la ejecución, resultando ser más adecuado para un sistema unificado, local y modular que debe ejecutarse de forma autónoma, sin depender de interfaces gráficas ni entornos web.

Tras esta decisión, se tomaron como referencia y ayuda los repositorios *Real Time Emotion Detection for Low Cost Robot in ROS*⁸ y *Detección de objetos en vídeo*⁹ y se creó un entorno de trabajo nuevo en Anaconda llamado *deteccionobj* para llevar a cabo el entrenamiento del modelo de detección de fresas. Durante este entrenamiento, surgieron varios códigos de error relacionados con el etiquetado de las imágenes utilizadas, por lo que se decidió etiquetarlas de nuevo mediante el programa labelImg (Figura II.12), tal y como se había hecho anteriormente con TensorFlow.

⁸https://github.com/jamarma/emotion_detection_ros

⁹<https://github.com/puigalex/deteccion-objetos-video>



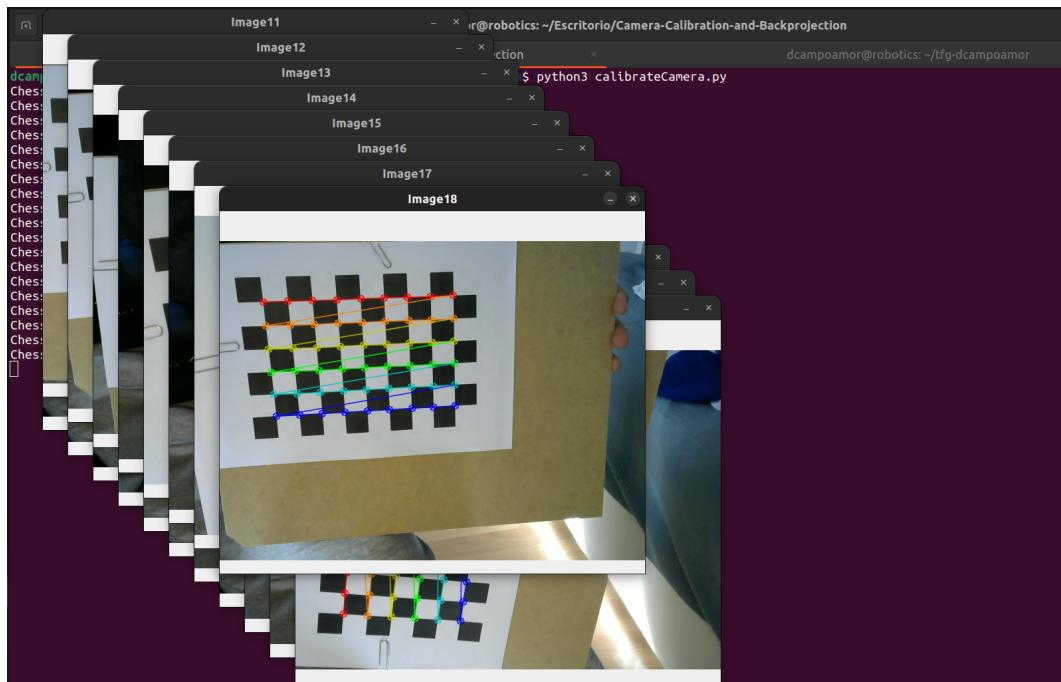
Figura II.12: Etiquetado de las imágenes con labelImg

Finalizado el proceso de etiquetado de 432 imágenes, se almacenaron en la carpeta *labels* los archivos que incluían tanto el número de clase, identificado con el valor 0, correspondiente a la única clase considerada, "Fresa", como las coordenadas que delimitaban la ubicación del objeto dentro de cada imagen. Con esta información organizada, se procedió al entrenamiento del modelo utilizando la arquitectura Darknet-53, implementada en el framework Darknet, más concretamente el archivo darknet53.conv.74, correspondiente a las primeras 74 capas de la red preentrenadas con pesos convolucionales, lo cual permitió una inicialización eficiente y evitó entrenar el modelo YOLO desde cero.

Para el entrenamiento, se configuró el parámetro *batch_size* con un valor de 2, debido a las limitaciones de capacidad de la tarjeta gráfica empleada, lo que implicó que las imágenes se procesaran de dos en dos por iteración. Además, al finalizar cada época del entrenamiento, entendida como el momento en que la red ha procesado y actualizado todos los ejemplos del conjunto de entrenamiento, se generaba un checkpoint con los pesos actuales del modelo, almacenado en la carpeta correspondiente. Al concluir el proceso, el entrenamiento había dado lugar a un total de 100 checkpoints, dado que el entrenamiento fue configurado por defecto para ejecutarse durante 100 épocas.

II.2.3. Calibrado de la cámara

De manera paralela al entrenamiento del modelo, con el fin de optimizar la detección de objetos mediante YOLOv3 y PyTorch, se procedió a la calibración de la cámara empleada, la Logitech C270, determinando sus parámetros intrínsecos y la transformación de su sistema de coordenadas respecto al entorno. Para ello, se utilizaron 20 imágenes de un patrón de tablero de ajedrez o *chess board* en diferentes posiciones, tomadas con la cámara a calibrar, mediante las cuales, y a través del uso del programa *PiCam-Calibrator.py*¹⁰ (tomado del artículo [Vega and Cañas, 2021]), como muestra la Figura II.13, se obtenían estos valores de la matriz K (Ecuación 5.2).



(a) Chess board

```
(deteccionobj) dcampoamor@robotics:~/Escritorio$ python3 PiCamCalibration.py
camera matrix:
[[820.30364609   0.          325.55229469]
 [ 0.          820.24892564 231.76974106]
 [ 0.          0.          1.        ]]
=====
Parametros interesantes de la matriz:
=====
Distancia focal [Fx, Fy] = [ 820.3036460891856 ,  820.2489256369695 ]
Centro optico [Cx, Cy] o [u0, v0] = [ 325.5522946910438 ,  231.7697410628658 ]
Coeficientes de distorsion = [[-4.18308124e-02  1.23163350e+00 -3.22083612e-04 -2.06418599e-04
 -5.12835368e+00]]
```

(b) Parámetros intrínsecos de la cámara

Figura II.13: Calibración de la cámara C270 de Logitech

¹⁰<https://github.com/RoboticsURJC/tfg-dcampoamor/blob/main/src/piCamCalibrator/PiCamCalibration.py>

Para corroborar que la calibración de la cámara C270 de Logitech fuera buena, se llevaron a cabo diez calibraciones para comprobar los resultados entre sí, siendo la media aritmética entre todas las mediciones los valores tomados para la programación.

Para poder llevar a cabo las primeras pruebas después de la calibración, se instaló la cámara en un trípode, cuya altura al plano mesa conocíamos, con una inclinación de la cámara medida mediante la aplicación de ERGONAUTAS RULER - Medición de ángulos en fotografías y vídeos¹¹ de la Universidad Politécnica de Valencia, como se aprecia en la Figura II.14.

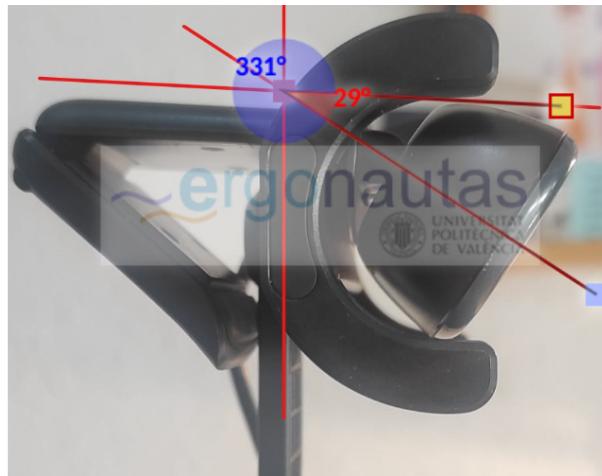


Figura II.14: Medición del ángulo de rotación de la cámara mediante la aplicación de ERGONAUTAS RULER

II.2.4. Pruebas detección de fresas en tiempo real

Una vez con el modelo entrenado y la cámara calibrada, se realizaron las primeras pruebas de detección de fresas en tiempo real utilizando Python, para las que se utilizó la cámara integrada del ordenador portátil e imágenes de fresas en el móvil mediante el programa *deteccion_video.py*¹², tal y como se muestra en la Figura II.15. A partir de estas primeras pruebas, se modificaron tanto el grosor del nombre de la clase a detectar como el *threshold* de la detección mostrado en la ventana emergente con OpenCV, ajustándolo de tal manera que el argumento pudiera ser más legible.

¹¹<https://www.ergonautas.upv.es/herramientas/ruler/ruler.php>

¹²https://github.com/RoboticsURJC/tfg-dcampoamor/blob/main/src/deteccion-objetos-video/deteccion_video.py

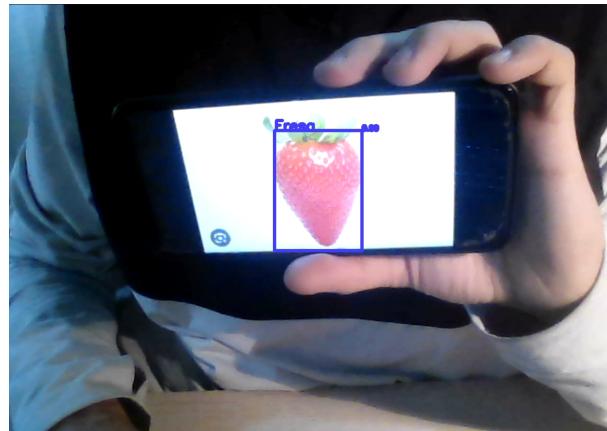


Figura II.15: Primeras pruebas de detección con PyTorch y Python

Sobre esta primera versión se fue modificando el código para poder añadir más funcionalidades al sistema, como la incorporación de un fragmento diseñado para almacenar dinámicamente las coordenadas centrales de los recuadros de las fresas detectadas por el modelo en una lista, y, dado que la versión inicial del script no consideraba la posible redundancia en la detección, es decir, la identificación múltiple de un mismo objeto debido a ligeras variaciones en la posición, también se incorporó un criterio de tolerancia espacial, que permitiera verificar si una nueva detección se encontraba a una distancia euclídea inferior al umbral respecto a alguna de las posiciones ya registradas, y en tal caso, la nueva posición no se añadiría a la lista, evitando así duplicidades en las detecciones (códigos II.1 y II.2).

```

threshold = 10 # Se mantiene para filtrar detecciones cercanas
...
# Esta función compara dos listas de posiciones usando un umbral
def positions_are_similar(list1, list2, threshold):
    if len(list1) != len(list2):
        return False
    for pos in list1:
        found = False
        for pos2 in list2:
            if calcular_distancia_3d(pos[0], pos[1], pos[2], pos2[0],
                                      pos2[1], pos2[2]) < threshold:
                found = True
                break
        if not found:
            return False
    return True

```

Código II.1: Función `positions_are_similar()`

```

# Solo se imprimen si las posiciones actuales difieren de las impresas
# anteriormente, usando threshold para comparar
if not positions_are_similar(filtered_positions, last_printed_positions,
    threshold):
    for idx, (x, y, z) in enumerate(filtered_positions, start=1):
        print(f"Punto P{idx} - Coordenadas 3D: X={x:.2f}, Y={y:.2f},
              Z=410.00")
        distancia = calcular_distancia_3d(0, 0, 0, x, y, z)
        print(f"Punto P{idx} - Distancia al punto: {distancia:.2f}
              milímetros")
    try:
        detected_points.append((x, y, z))
    except Exception as e:
        print(f"[ERROR] No se pudo enviar la posición al robot: {e}")
last_printed_positions = filtered_positions.copy()

```

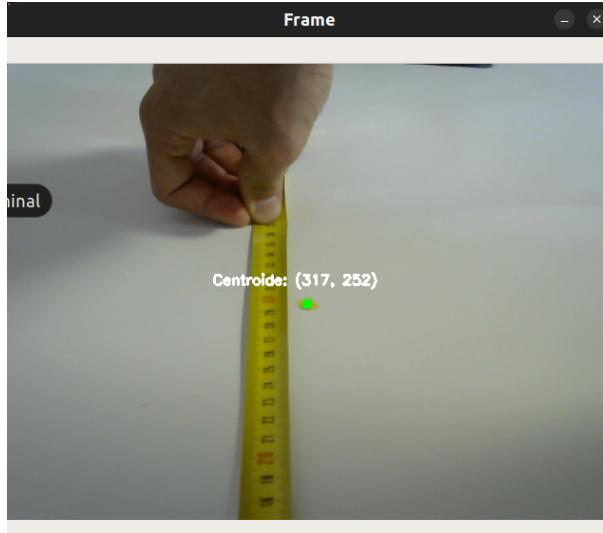
Código II.2: Fragmento del código donde se comparan posiciones para evitar que se dupliquen

Paralelamente a esto, con el fin de verificar los cálculos de las transformaciones entre sistemas de coordenadas, y poder obtener las distancias a las que se encontraban las detecciones de la cámara, se desarrolló el programa *pos_centroide.py*¹³ en Python. Este script utilizaba la biblioteca OpenCV para realizar la detección de un objeto a partir de un filtro por color, calculando su centroide y devolviendo el resultado en píxeles. A partir de estas coordenadas en píxeles, era posible obtener las coordenadas en el sistema óptico de la cámara y finalmente a partir de estas coordenadas bidimensionales, proyectar el punto en el espacio tridimensional utilizando los parámetros intrínsecos y extrínsecos de la cámara, permitiendo así obtener las coordenadas espaciales X,Y,Z que se mostraban en la terminal como salida del programa.

Por este motivo, se utilizó un cuadrado de un post-it subrayado con color amarillo fosforecente sobre una cartulina blanca para poder minimizar el error cometido por el programa al aplicar el filtro de color y poder calcular así de mejor manera las distancias y coordenadas del centroide de ese cuadrado. Para estas pruebas, se supuso un sistema de referencia cartesiano, donde el eje Z es perpendicular al plano de la mesa, es decir, la altura a la que se encontraba instalada la cámara a la hora de realizar las pruebas para poder seguir el principio de la hipótesis suelo, mientras que en un principio, se supuso el eje X como el eje longitudinal de la mesa y el eje Y el transversal de la misma. No obstante, se realizaron más mediciones desplazando el post-it en distintas

¹³https://github.com/RoboticsURJC/tfg-dcampoamor/blob/main/src/camera/pos_centroide.py

direcciones para poder validar estas suposiciones sobre la configuración del sistema de coordenadas y ajustar su orientación a fin de asegurar la correspondencia entre los resultados estimados por el sistema y las coordenadas reales del entorno (Figura II.16).



(a) Ventana donde se muestran las detecciones mediante filtro de color

```
Coordenadas ópticas: X=0.008737500000000153, Y=0.1883375000000001
Coordenadas 3D: X=[-164778.1382016], Y=[-42783.87764831], Z=[-181.22113765]
Coordenadas ópticas: X=0.008737500000000153, Y=0.1883375000000001
Coordenadas 3D: X=[-164778.1382016], Y=[-42783.87764831], Z=[-181.22113765]
Coordenadas ópticas: X=0.018112500000000153, Y=0.1883375000000001
Coordenadas 3D: X=[-164773.69002305], Y=[-42785.18374148], Z=[-181.22665015]
Coordenadas ópticas: X=-0.5162624999999998, Y=-0.2241624999999993
Coordenadas 3D: X=[-165027.23620005], Y=[-43046.69416871], Z=[-180.91243765]
Coordenadas ópticas: X=0.018112500000000153, Y=0.1883375000000001
Coordenadas 3D: X=[-164773.69002305], Y=[-42785.18374148], Z=[-181.22665015]
Coordenadas ópticas: X=0.008737500000000153, Y=0.1883375000000001
Coordenadas 3D: X=[-164778.1382016], Y=[-42783.87764831], Z=[-181.22113765]
Coordenadas ópticas: X=0.008737500000000153, Y=0.1883375000000001
Coordenadas 3D: X=[-164778.1382016], Y=[-42783.87764831], Z=[-181.22113765]
Coordenadas ópticas: X=0.008737500000000153, Y=0.1883375000000001
Coordenadas 3D: X=[-164778.1382016], Y=[-42783.87764831], Z=[-181.22113765]
Coordenadas ópticas: X=0.008737500000000153, Y=0.1883375000000001
```

(b) Mensajes mostrados por la terminal al ejecutar el programa

Figura II.16: Primeras pruebas de la estimación de las coordenadas y la distancia de la detección a la cámara

De los resultados de estas mediciones, recogidos en el Cuadro II.2, se observó que existía algún tipo de error en el proceso de cálculo, posiblemente en las transformaciones aplicadas o en la omisión de algún factor relevante, ya fuera geométrico o de calibración, ya que se manifestaba la falta de concordancia entre las coordenadas espaciales reales y las estimadas por el sistema, a pesar de que se pudo comprobar que los valores obtenidos presentan una coherencia relativa, al variar conforme estos desplazamientos.

	Coordenadas reales (mm)			Coordenadas obtenidas (mm)		
	X	Y	Z	X	Y	Z
Punto inicial	300	0	-225	-164778,14	-42783,88	-181,22
Punto desplazado en +X	350	0	-225	-164702,52	-43447,45	-181,31
Punto desplazado en -X	250	0	-225	-164693,62	-41943,35	-181,33
Punto desplazado en -Y	300	-50	-225	-164199,87	-42923,13	-181,94
Punto desplazado en +Y	300	50	-225	-165182,92	-429695,56	-180,72

Cuadro II.2: Comparación entre coordenadas reales y obtenidas (en mm)

Ante estas discrepancias, se procedió a consultar bibliografía útil para este ámbito, concretamente el documento *Real World Coordinate from Image Coordinate Using Single Calibrated Camera*[Siswantoro et al., 2013], donde se analizaba la geometría del modelo de cámara basado en el modelo estenopeico o modelo pinhole (Figura II.17), cuya figura se empleó como referencia para verificar la correcta definición y orientación de los ejes del sistema de coordenadas. Esta revisión permitió contrastar las hipótesis iniciales relativas a los ejes sobre los que se aplican las rotaciones de la cámara, así como los ejes en los que se realizaron las mediciones experimentales, con el objetivo de detectar posibles inconsistencias en la configuración del sistema de referencia adoptado.

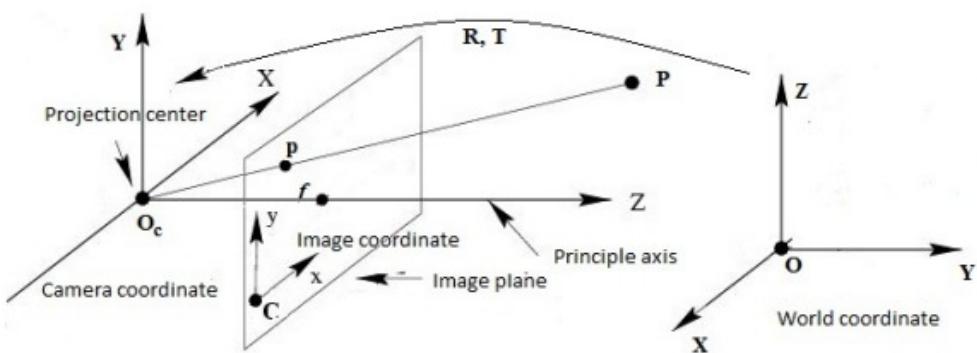


Figura II.17: Geometría basada en el modelo de cámara estenopeica

Dado que los resultados experimentales mostraban una desviación considerable respecto a las distancias reales y no se lograba establecer una relación clara y precisa, se optó por adoptar un enfoque alternativo basado en un script preexistente denominado *pinhole.py*¹⁴, que implementaba el modelo de cámara estenopeica para estimar coordenadas espaciales a partir de coordenadas de un imagen o vídeo.

¹⁴<https://github.com/RoboticsURJC/tfg-dcampoamor/blob/main/src/camera/cameraPibot/pinhole.py>

Se ajustaron diversos parámetros globales fundamentales, tales como el ancho y alto de la imagen, la distancia focal y la posición del centro óptico de la cámara, datos obtenidos anteriormente en la calibración de la cámara, así como variables asociadas a la rotación de la misma, como el ángulo de inclinación de la cámara, y a la translación o altura de la cámara respecto a la superficie de trabajo, con el fin de calcular adecuadamente las matrices de rotación y translación necesarias para la proyección de puntos desde el espacio imagen al espacio real.

Además, se adaptó el rango de detección cromática para que, al ejecutar el script, pudiera detectarse el color amarillo característico del post-it sobre la cartulina blanca dispuesta sobre la mesa, facilitando así la identificación automática del objeto en las pruebas, tal y como se había hecho con scripts anteriores.

Se estableció como origen del sistema de coordenadas del mundo la proyección vertical del centro óptico de la cámara sobre la superficie de la mesa, y se llevaron a cabo distintas pruebas experimentales, modificando tanto el signo del ángulo de rotación como el signo de la distancia de la cámara a la mesa, con el objetivo de determinar la orientación correcta de los ejes espaciales definidos en el script y verificar si los resultados estimados se correspondían con las coordenadas reales. Para la recogida de datos, se empleó una regla graduada colocada sobre la superficie de la mesa, y se fue desplazando de manera progresiva el post-it en distintas direcciones para poder validar las suposiciones sobre la configuración del sistema de coordenadas (Figura II.18).

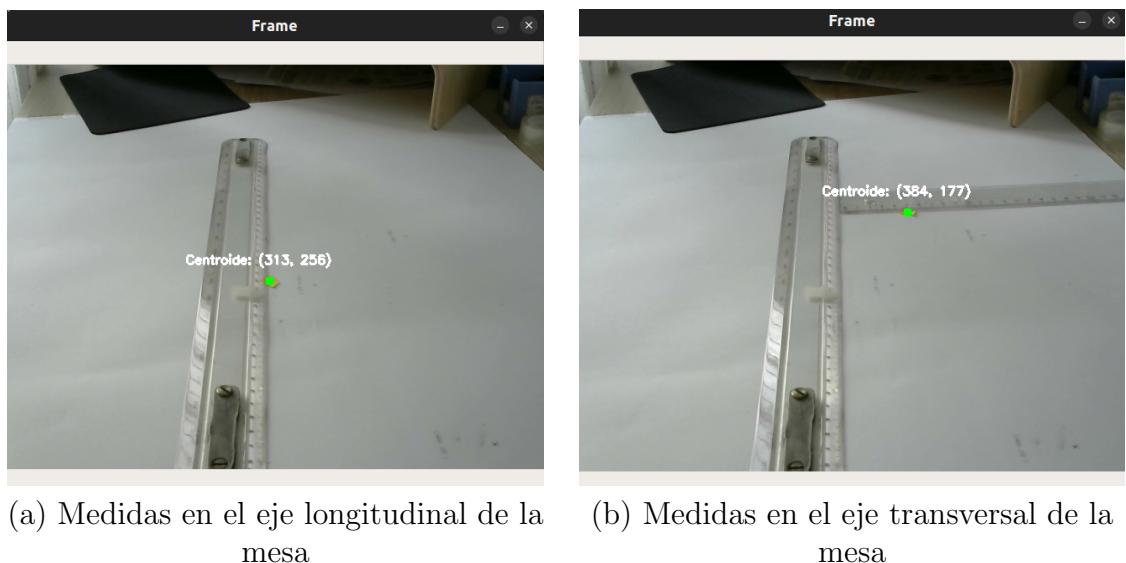


Figura II.18: Pruebas para determinar la configuración del sistema de coordenadas de la cámara

A partir de los resultados obtenidos en estas pruebas, recogidos en los cuadros II.3 y II.4, se pudo comprobar que, para que las estimaciones del sistema fueran coherentes con las distancias reales, era necesario introducir la altura de la cámara respecto al plano suelo, en este caso, la mesa, como un valor negativo, mientras que el ángulo de rotación de la cámara debía establecerse en valor positivo.

pinhole.py con posición Z de la cámara positiva								
Coordenadas reales (mm)			Coordenadas mundo (mm) (22º Rotación)			Coordenadas mundo (mm) (-22º Rotación)		
X	Y	Z	X	Y	Z	X	Y	Z
300	0	265	-35,51	-24,83	0	195,14	-30,56	0
400	0	265	-72,43	-26,49	0	144,33	-28,73	0
500	0	265	-101,42	-27,54	0	112,42	-27,23	0
500	50	265	-101,79	-51,6	0	112,41	-51,62	0
500	100	265	-102,91	-76,81	0	111,27	-77,3	0
500	150	265	-104,41	-102,14	0	110,5	-102,96	0
500	200	265	-105,16	-128,84	0	108,98	-129,49	0
600	0	265	-124,82	-27,31	0	90,1	-26,44	0

Cuadro II.3: Resultados del programa pinhole.py con valores de Z positivos

pinhole.py con posición Z de la cámara negativa								
Coordenadas reales (mm)			Coordenadas mundo (mm) (22º Rotación)			Coordenadas mundo (mm) (-22º Rotación)		
X	Y	Z	X	Y	Z	X	Y	Z
300	0	265	35,51	24,51	0	-196,08	30,59	0
400	0	265	73,47	25,52	0	-144,74	28,01	0
500	0	265	102,16	26,52	0	-112,8	27,6	0
500	50	265	101,79	51,25	0	-112,4	51,97	0
500	100	265	102,91	76,81	0	-111,27	77,65	0
500	150	265	104,41	102,84	0	-110,13	102,55	0
500	200	265	105,91	127,57	0	-108,23	127,96	0
600	0	265	124,42	26,94	0	-89,74	26,09	0

Cuadro II.4: Resultados del programa pinhole.py con valores de Z negativos

No obstante, a pesar de haber definido un sistema de coordenadas inicial, los resultados continuaban sin ajustarse adecuadamente a las distancias reales. Por ello, se decidió revisar con mayor detalle la lógica del script *pinhole.py*, prestando especial atención a los comentarios incluidos en el código, en los que se indicaba que los ángulos utilizados para calcular la matriz de rotación se definían bajo la suposición de que la cámara, en orientación vertical, había sido sometida a una rotación previa de 90º sobre el eje Y. A partir de esta observación, se procedió a representar y analizar el sistema de coordenadas original de la cámara antes de aplicar dicha transformación, con el objetivo de comprender mejor la correspondencia entre los ejes del sistema imagen y del sistema mundo, y así ajustar de forma más precisa las transformaciones necesarias (Figura II.19).

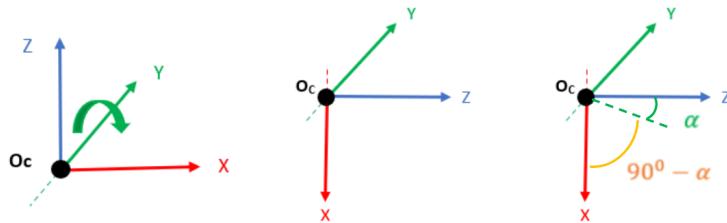


Figura II.19: Esquema de la rotación de la cámara

Se consideró entonces una rotación sobre el eje Y de la cámara, y se utilizó como ángulo de entrada para el código, el valor resultante de restar los 90° de la rotación inicial asumida en el script menos el ángulo previamente utilizado (definido como el ángulo entre la horizontal y la dirección de la lente). El valor resultante en esta ocasión fue de aproximadamente 65° , con el cual se volvió a ejecutar el script, a fin de evaluar si esta nueva configuración ofrecía una mayor coherencia entre las coordenadas estimadas y las medidas reales, repitiendo las mismas pruebas definidas anteriormente y obteniendo los resultados del Cuadro II.5.

pinhole.py con posición Z de la cámara negativa					
Coordenadas reales (mm)			Coordenadas mundo (mm) (65° Rotación)		
X	Y	Z	X	Y	Z
240	0	0	260,47	29,09	0
300	0	0	325,28	33,49	0
400	0	0	438,29	41,31	0
500	0	0	553,93	49,05	0
500	50	0	541,95	36,06	0
500	100	0	545,32	148,92	0
500	150	0	550,46	202,57	0
500	200	0	552,19	261,74	0
600	0	0	675,18	59,61	0

Cuadro II.5: Resultados del programa pinhole.py con el valor ajustado de rotación de la cámara

Una vez verificada y establecida la forma correcta de introducir los parámetros relativos al ángulo de inclinación y a la altura de la cámara respecto al plano suelo, ya que pudieron considerarse satisfactorios debido a su similitud con las mediciones reales, se procedió a incorporar en el propio script el cálculo de la distancia a partir de las coordenadas tridimensionales (X,Y,Z) del punto detectado, desarrollando para ello la función `calcular_distancia_3d(x_cam, y_cam, z_cam, x_punto, y_punto, z_punto)`, que permitía estimar la distancia euclídea desde el origen del sistema hasta el punto proyectado (Figura II.20).

Figura II.20: Detección y cálculo de las coordenadas y la distancia a la detección

Dado que el programa *pinhole.py* permitía inicialmente la detección de un único punto amarillo dentro de la escena, en línea con el enfoque de las pruebas preliminares, centradas en validar el funcionamiento del sistema y la precisión del cálculo de la posición y la distancia del objeto respecto a la cámara, surgía una limitación evidente al poder darse la posibilidad de trabajar con múltiples detecciones. Para permitir la detección simultánea, se modificó este programa, generando uno nuevo denominado *pinhole_deteccionmultiple.py*¹⁵, y se procedió en primer lugar a ajustar los filtros de color del sistema, ya que en el entorno de pruebas existían varios elementos de madera cuyas tonalidades se confundían con el amarillo claro bajo determinadas condiciones de iluminación, generando falsas detecciones, por lo que, para mitigar este problema, se optó por reemplazar el color amarillo por el verde, lo que implicó también pintar los fragmentos de post-it utilizados en las pruebas con este nuevo color. Una vez realizada esta modificación, se implementaron mejoras adicionales, como la numeración de los objetos detectados para permitir identificar de forma unívoca cada detección dentro del mismo escenario, lo cual resultaba esencial para poder asociar correctamente las coordenadas espaciales y las distancias estimadas con cada objeto individual, ampliando la capacidad del sistema para trabajar en escenarios más complejos con múltiples puntos de interés (Figura II.21).

¹⁵https://github.com/RoboticsURJC/tfg-dcampoamor/blob/main/src/camera/cameraPibot/pinhole_deteccionmultiple.py



(a) Ventana donde se muestran las detecciones múltiples de manera simultánea mediante filtro de color

```

Punto P1 - Coordenadas 3D: X=269.64831540978133, Y=-16.160880912610924, Z=0.0
Punto P1 - Distancia al punto: 378.41 milímetros
Punto P2 - Coordenadas 3D: X=306.2312816037335, Y=33.0335204979437, Z=0.0
Punto P2 - Distancia al punto: 406.32 milímetros
Punto P3 - Coordenadas 3D: X=346.09337326614883, Y=-27.142293030539225, Z=0.0
Punto P3 - Distancia al punto: 436.74 milímetros
Punto P1 - Coordenadas 3D: X=269.64831540978133, Y=-16.160880912610924, Z=0.0
Punto P1 - Distancia al punto: 378.41 milímetros
Punto P2 - Coordenadas 3D: X=306.2312816037335, Y=33.0335204979437, Z=0.0
Punto P2 - Distancia al punto: 406.32 milímetros
Punto P3 - Coordenadas 3D: X=346.09337326614883, Y=-27.142293030539225, Z=0.0
Punto P3 - Distancia al punto: 436.74 milímetros
Punto P1 - Coordenadas 3D: X=269.64831540978133, Y=-16.160880912610924, Z=0.0
Punto P1 - Distancia al punto: 378.41 milímetros
Punto P2 - Coordenadas 3D: X=306.9339449684114, Y=33.08752536730562, Z=0.0
Punto P2 - Distancia al punto: 406.85 milímetros
Punto P3 - Coordenadas 3D: X=346.09337326614883, Y=-26.619652454594455, Z=0.0
Punto P3 - Distancia al punto: 436.71 milímetros
Punto P1 - Coordenadas 3D: X=269.64831540978133, Y=-16.160880912610924, Z=0.0
Punto P1 - Distancia al punto: 378.41 milímetros
Punto P2 - Coordenadas 3D: X=305.5309082505877, Y=32.97969163288177, Z=0.0
Punto P2 - Distancia al punto: 405.79 milímetros
Punto P3 - Coordenadas 3D: X=346.09337326614883, Y=-27.142293030539225, Z=0.0
Punto P3 - Distancia al punto: 436.74 milímetros

```

(b) Mensajes mostrados por la terminal al ejecutar el programa

Figura II.21: Detección múltiple simultánea de post-it

Una vez resueltas las limitaciones iniciales del programa *pinhole.py* en cuanto a la detección de múltiples objetos, y tras haber conseguido proyectar correctamente los centroides de los objetos detectados junto con el cálculo de sus coordenadas espaciales y distancias respectivas, se avanzó en la representación visual de estos resultados modificando el script *pinhole_deteccionmultiple.py* para que, además de realizar la detección y proyección de múltiples detecciones de manera simultánea, se capturaran y representaran estos datos en una ventana adicional mediante OpenGL. Para ello, se reutilizaron y adaptaron las funcionalidades de navegación y visualización incluidas en el

script *scene_navigation.py*¹⁶, que permitía representar escenas 3D de forma interactiva mediante el uso de *multithreading*, una técnica de programación que permite ejecutar múltiples tareas concurrentemente dentro de un mismo proceso.

Tras varias pruebas y ajustes en la integración de ambos scripts, se consiguió que los puntos detectados por la cámara se representaran correctamente en una ventana paralela con fondo negro utilizando OpenGL, mostrándose cada punto como una marca roja en el espacio tridimensional, pero para mejorar la visibilidad y el contraste de estos puntos, se modificó tanto el tamaño de los puntos como la selección del color para que fuera más perceptible frente al fondo y se añadieron líneas entre los puntos detectados, tal y como se muestra en la Figura II.22. Esta representación permitió verificar visualmente la coherencia de las coordenadas 3D calculadas a partir de la cámara, facilitando así la validación y comprensión de los resultados espaciales obtenidos durante las pruebas anteriores.

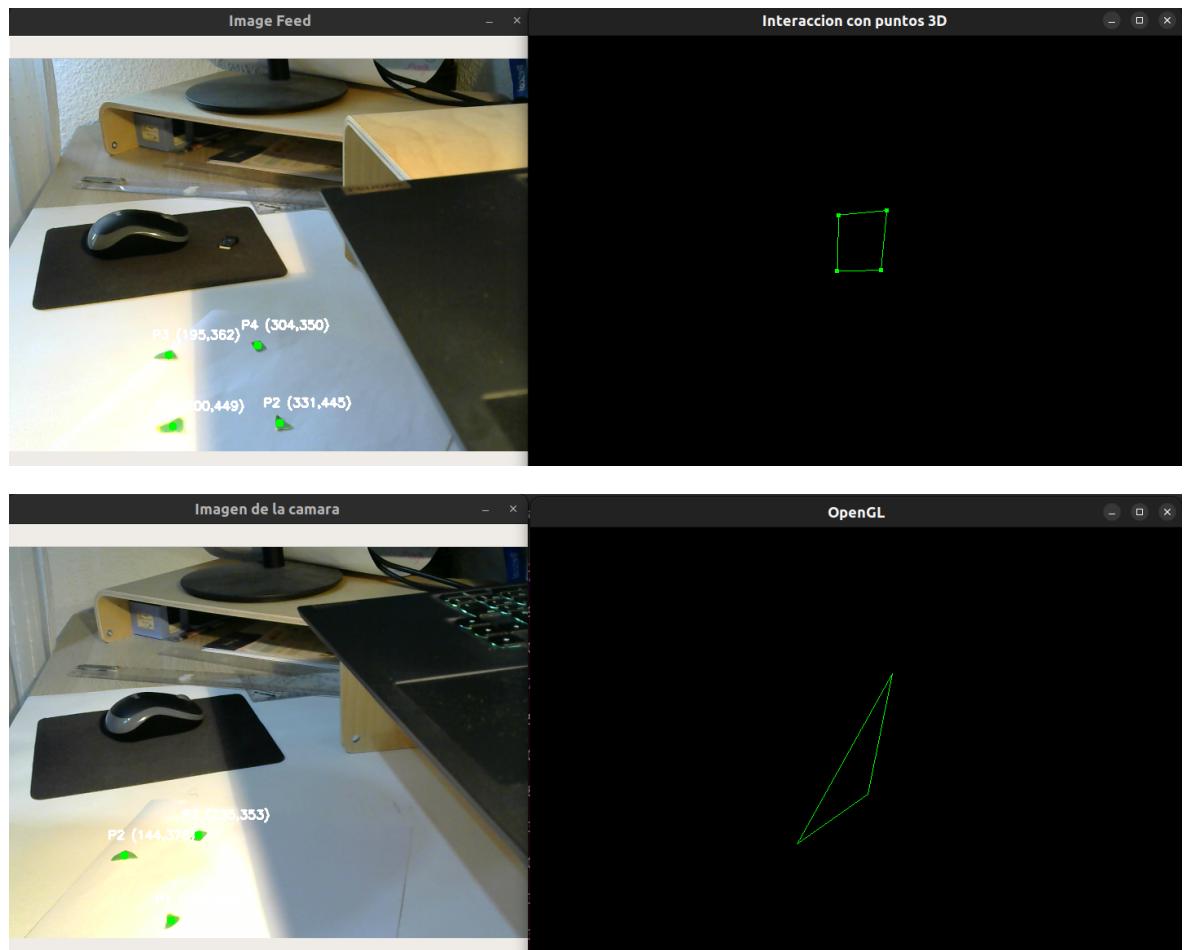


Figura II.22: Representación de las detecciones con OpenGL

¹⁶https://github.com/RoboticsURJC/tfg-dcampoamor/blob/main/src/camera/openglhw/scene_navigation.py

Una vez validado el funcionamiento del sistema con fragmentos de post-it pintados, y comprobada la capacidad del programa para detectar múltiples puntos y calcular sus coordenadas espaciales, se consideró que el sistema estaba preparado para abordar el siguiente paso del proyecto: sustituir los objetos de prueba por el objeto real de estudio, las fresas, por lo que se inició una nueva fase centrada en la detección, localización y estimación de la distancia a estas detecciones de fresas en condiciones más próximas a las del entorno operativo final, recogida en el programa *xmlrpc_deteccionfresas.py*¹⁷. Para ello, fue necesario modificar algunos parámetros del sistema y sustituir el método de detección basado en filtros de color, empleado durante las pruebas iniciales, por el modelo de detección entrenado específicamente para identificar fresas, que ya se encontraba integrado en el script *deteccion_video.py*, por lo que se reutilizó dicha lógica adaptándola al flujo de trabajo del sistema basado en *pinhole.py*, obteniendo los resultados de la Figura II.23.

```
[INFO] Parámetros de la cámara cargados correctamente.
Punto P1 - Coordenadas 3D: X=1.69, Y=0.56, Z=0.00
Punto P1 - Distancia al punto: 265.01 milímetros
Punto P1 - Coordenadas 3D: X=1.76, Y=0.62, Z=0.00
Punto P1 - Distancia al punto: 265.01 milímetros
Punto P1 - Coordenadas 3D: X=1.91, Y=0.55, Z=0.00
Punto P1 - Distancia al punto: 265.01 milímetros
Punto P1 - Coordenadas 3D: X=2.02, Y=0.57, Z=0.00
Punto P1 - Distancia al punto: 265.01 milímetros
Punto P1 - Coordenadas 3D: X=2.19, Y=0.65, Z=0.00
Punto P1 - Distancia al punto: 265.01 milímetros
Punto P1 - Coordenadas 3D: X=2.28, Y=0.68, Z=0.00
Punto P1 - Distancia al punto: 265.01 milímetros
Punto P1 - Coordenadas 3D: X=2.12, Y=0.57, Z=0.00
Punto P1 - Distancia al punto: 265.01 milímetros
Punto P1 - Coordenadas 3D: X=2.00, Y=0.44, Z=0.00
Punto P1 - Distancia al punto: 265.01 milímetros
Punto P1 - Coordenadas 3D: X=1.68, Y=0.29, Z=0.00
Punto P1 - Distancia al punto: 265.01 milímetros
Punto P1 - Coordenadas 3D: X=1.61, Y=0.22, Z=0.00
Punto P1 - Distancia al punto: 265.00 milímetros
Punto P1 - Coordenadas 3D: X=1.74, Y=0.19, Z=0.00
Punto P1 - Distancia al punto: 265.01 milímetros
Punto P1 - Coordenadas 3D: X=1.66, Y=0.44, Z=0.00
Punto P1 - Distancia al punto: 265.01 milímetros
Punto P1 - Coordenadas 3D: X=1.60, Y=0.10, Z=0.00
Punto P1 - Distancia al punto: 265.00 milímetros
```

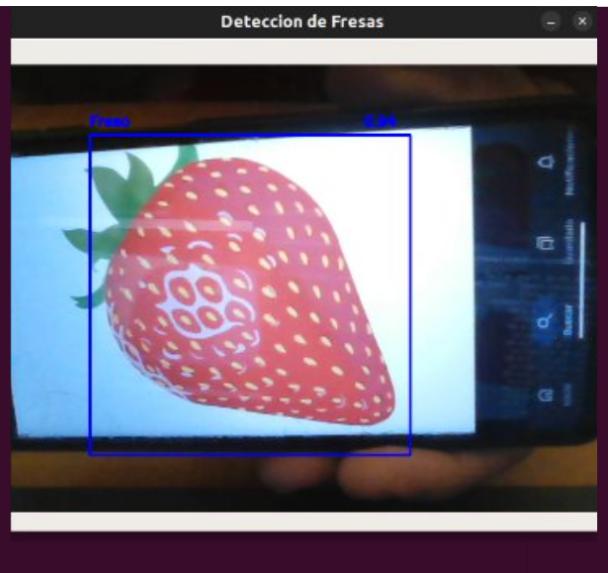


Figura II.23: Detección de fresas e integración con el sistema de cálculo de coordenadas y distancias

Sin embargo, la integración del modelo de detección de fresas no fue un proceso directo, ya que el cambio de un método de detección basado en filtrado de color a uno basado en inteligencia artificial implicó importantes ajustes, ya que las características de la detección variaban considerablemente. Debido a esta modificación, fue necesario repetir las pruebas de estimación de coordenadas y distancias, con el fin de verificar nuevamente la orientación del sistema de coordenadas y asegurar que las nuevas de-

¹⁷https://github.com/RoboticsURJC/tfg-dcampoamor/blob/main/src/deteccion-objetos-video/xmlrpc_deteccionfresas.py

tecciones proporcionadas por el modelo se proyectaran correctamente en el espacio tridimensional. Estas pruebas permitieron recalibrar el sistema en función del nuevo flujo de trabajo (ver cuadros II.6, II.7 II.8 y II.10).

Parámetros de la cámara		
Altura (mm)	Rotación (º)	Rotación (rad)
145	59	1,0297

POSICIÓN	Coordenadas reales (mm)				Coordenadas obtenidas (mm)			
	X	Y	Z	Distancia real (mm)	X	Y	Z	Distancia obtenida (mm)
P1	280	-25	145	316,31	273,24	-24,65	145	310,31
P2	200	0	145	247,03	342,97	20,24	145	372,91
P3	185	-75	145	246,73	166,39	20,1	145	221,62
P4	360	85	145	397,30	634,66	-87,32	145	656,84
P5	235	45	145	279,78	480,32	8,74	145	501,81
P6	255	75	145	302,78	191,22	-11,9	145	240,27
P7	255	0	145	293,34	250,75	-12,19	145	289,91
P8	340	-50	145	372,99	451,91	-63,67	145	478,85
P9	160	0	145	215,93	345,53	53,99	145	378,59
P10	220	105	145	283,64	137,04	-2,71	145	199,53

Cuadro II.6: Resultados del programa xmlrpc_deteccionfresas.py con la cámara situada a 145 mm de la mesa y la cámara rotada 59 grados

Parámetros de la cámara		
Altura (mm)	Rotación (º)	Rotación (rad)
125	59	1,0297

POSICIÓN	Coordenadas reales (mm)				Coordenadas obtenidas (mm)			
	X	Y	Z	Distancia real (mm)	X	Y	Z	Distancia obtenida (mm)
P1	280	-25	125	307,65	409,94	-12,35	125	428,75
P2	200	0	125	235,85	257,34	36,56	125	288,42
P3	185	-75	125	235,53	134,88	23,6	125	185,40
P4	360	85	125	390,45	580,85	-68,12	125	598,04
P5	235	45	125	269,95	514,59	35,56	125	530,75
P6	255	75	125	293,73	505,49	15,6	125	520,95
P7	255	0	125	283,99	261,32	-1,88	125	289,68
P8	340	-50	125	365,68	206,22	-30,37	125	243,05
P9	160	0	125	203,04	246,03	56,44	125	281,68
P10	220	105	125	273,95	510,73	92,88	125	533,94

Cuadro II.7: Resultados del programa xmlrpc_deteccionfresas.py con la cámara situada a 125 mm de la mesa y la cámara rotada 59 grados

Parámetros de la cámara		
Altura (mm)	Rotación (º)	Rotación (rad)
225	69	1,2043

POSICIÓN	Coordenadas reales (mm)				Coordenadas obtenidas (mm)			
	X	Y	Z	Distancia real (mm)	X	Y	Z	Distancia obtenida (mm)
P1	0	0	225	225,00	0,26	-1,64	225	225,01
P2	0	20	225	225,89	28,09	1,69	225	226,75
P3	0	50	225	230,49	51,31	-0,33	225	230,78
P4	10	0	225	225,22	-3,35	-9,69	225	225,23
P5	10	70	225	235,85	74,54	-5,77	225	237,10
P6	10	-70	225	235,85	-70,86	-18,43	225	236,61
P7	20	0	225	225,89	-1,7	-19,21	225	225,82
P8	20	60	225	233,72	63,57	-15,74	225	234,34
P9	20	-90	225	243,16	-72,89	-27,81	225	238,14
P10	30	0	225	226,99	-0,13	-31,5	225	227,19
P11	30	80	225	240,68	77,38	-15,14	225	238,42
P12	30	-80	225	240,68	-70,83	-37,88	225	238,91
P13	40	0	225	228,53	4,46	-41,76	225	228,89
P14	40	10	225	228,75	23,23	-42,44	225	230,14
P15	40	-10	225	228,75	-15,39	-44,21	225	229,82
P16	50	0	225	230,49	2,04	-50,43	225	230,59
P17	50	40	225	233,93	50,13	-47,55	225	235,37
P18	50	-40	225	233,93	-30,09	-50,87	225	232,63
P19	-10	0	225	225,22	-1,17	8,24	225	225,15
P20	-10	20	225	226,11	16,58	9,84	225	225,82
P21	-10	-20	225	226,11	-24,21	21,4	225	227,31
P22	-20	0	225	225,89	0,2	25,28	225	226,42
P23	-20	50	225	231,35	51,77	17,49	225	231,54
P24	-20	-80	225	239,64	-73,12	15,76	225	237,11
P25	-30	0	225	226,99	2,14	30,06	225	227,01
P26	-30	15	225	227,49	23,67	34,31	225	228,83
P27	-30	-15	225	227,49	-19,09	31,75	225	228,03
P28	-40	0	225	228,53	5,71	44,66	225	229,46
P29	-40	40	225	232,00	48,34	38,55	225	233,34
P30	-40	-40	225	232,00	-37,19	41,19	225	231,74
P31	-50	0	225	230,49	-5,96	48,37	225	230,22
P32	-50	35	225	233,13	43,18	46,08	225	233,69
P33	-50	-65	225	239,48	-63,22	49,21	225	238,84

Cuadro II.8: Resultados del programa `xmlrpc_deteccionfresas.py` con la cámara situada a 225 mm de la mesa y la cámara rotada 69 grados

Parametros de la cámara		
Altura (mm)	Rotación (º)	Rotación (rad)
180	58	1,0123

POSICIÓN	Coordenadas reales (mm)				Coordenadas obtenidas (mm)			
	X	Y	Z	Distancia real (mm)	X	Y	Z	Distancia obtenida (mm)
P1	190	0	180	261,73	262,24	58,83	180	323,47
P2	190	50	180	266,46	466,78	101,58	180	510,49
P3	190	-50	180	266,46	175,89	45,5	180	255,75
P4	220	0	180	284,25	248,64	45,58	180	310,32
P5	220	70	180	292,75	544,92	63,8	180	577,42
P6	220	-70	180	292,75	177,44	25,41	180	254,03
P7	250	0	180	308,06	250,7	21,85	180	309,40
P8	250	90	180	320,94	602,52	43,22	180	630,32
P9	250	-90	180	320,94	162,81	14,27	180	243,13
P10	280	0	180	332,87	253,89	5,38	180	311,27
P11	280	100	180	347,56	589,71	5,32	180	616,59
P12	280	-100	180	347,56	157,87	3,38	180	239,45
P13	300	0	180	349,86	261,1	-3,52	180	317,15
P14	300	10	180	350,00	322,72	-12,42	180	369,73
P15	300	-10	180	350,00	236,74	-3,42	180	297,42
P16	350	0	180	393,57	261,91	-27,14	180	318,96
P17	350	40	180	395,60	369,51	-33,83	180	412,41
P18	350	-40	180	395,60	212,74	-24,87	180	279,78
P19	370	0	180	411,46	259,89	-31,71	180	317,72
P20	370	20	180	411,95	320,27	-41,32	180	369,70
P21	370	-20	180	411,95	236,49	-30,17	180	298,73
P22	400	0	180	438,63	270,09	-44,29	180	327,58
P23	400	80	180	445,87	430,15	-65,36	180	470,85
P24	400	-80	180	445,87	185,39	-35,32	180	260,80
P25	450	0	180	484,66	259,87	-55,65	180	320,98

Cuadro II.9: Resultados del programa `xmlrpc_deteccionfresas.py` con la cámara situada a 180 mm de la mesa y la cámara rotada 58 grados

En todas estas mediciones, en las que se realizaron variaciones tanto en la altura de la cámara como en su ángulo de inclinación de la misma, se analizó el impacto de estos ajustes sobre la precisión del sistema de proyección de coordenadas, observando un patrón consistente en el que las coordenadas obtenidas mediante el programa de las fresas situadas en los extremos del campo de visión (FoV) de la cámara presentaban valores anómalos o significativamente desviados con respecto a las posiciones reales y a la distancia total, y más concretamente, en aquellas fresas detectadas cerca de los bordes superior, inferior o laterales del encuadre. En contraste, los objetos ubicados en la zona central del campo de visión ofrecían una mayor precisión en los resultados obtenidos respecto a las medidas reales, siendo esta región la más fiable tanto para las coordenadas proyectadas en los ejes X e Y, asumidos como el eje longitudinal y transversal de la mesa respectivamente, como para la distancia al plano de referencia.

A pesar de haber identificado una zona dentro del campo de visión de la cámara en la que los resultados obtenidos se aproximaban considerablemente a las coordenadas reales, persistían errores significativos, especialmente en el eje que se había supuesto como eje Y del sistema, donde se concentraban las mayores desviaciones entre las coordenadas supuestas y las estimadas. Con el fin de analizar más a fondo este problema

y validar visualmente el comportamiento del sistema, se llevó a cabo la proyección en OpenGL de las detecciones junto con la representación del campo visual de la cámara en una ventana emergente paralela en la ejecución del programa *xmlrpc_deteccionfresas_OpenGL.py*¹⁸, permitiendo comprobar de manera más intuitiva y precisa si la transformación de puntos desde el espacio 2D al espacio 3D se estaba realizando correctamente, tal y como muestra la Figura II.24.

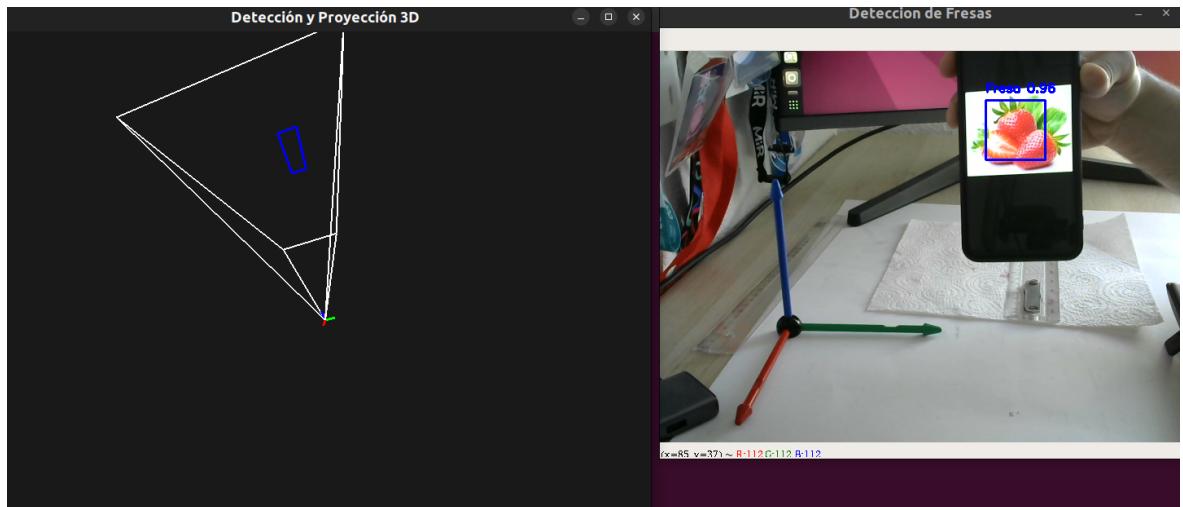


Figura II.24: Proyección con OpenGL de las detecciones y el campo visual de la cámara

Mediante esta representación, se pudo verificar que la proyección desde coordenadas 2D a 3D se realizaba de forma adecuada, lo que reforzaba la validez del modelo de transformación empleado, por lo que se llevaron a cabo nuevas pruebas experimentales, esta vez situando la cámara en una posición completamente perpendicular a la superficie del plano suelo (plano de la mesa) gracias al soporte impreso en 3D para poder colocar y fijar la cámara de manera mas fiable, lo que correspondía a un ángulo de rotación de 0 grados.

El objetivo de esta configuración era eliminar posibles efectos de inclinación en las mediciones y facilitar la comprobación directa de la correspondencia entre los ejes del sistema real y los ejes definidos teóricamente, tomando nuevas medidas que permitieron evaluar si las coordenadas estimadas mantenían una relación coherente con la realidad física del entorno y terminar de validar la orientación definitiva de los ejes espaciales del sistema.

¹⁸https://github.com/RoboticsURJC/tfg-dcampoamor/blob/main/src/deteccion-objetos-video/xmlrpc_deteccionfresas_OpenGL.py

Parámetros de la cámara		
Altura (mm)	Rotación (º)	Rotación (rad)
225	0	0

POSICIÓN	Coordenadas reales (mm)				Coordenadas obtenidas (mm)			
	X	Y	Z	Distancia real (mm)	X	Y	Z	Distancia obtenida (mm)
P1	0	0	225	225,00	0,26	-1,64	225	225,01
P2	0	20	225	225,89	28,09	1,69	225	226,75
P3	0	50	225	230,49	51,31	-0,33	225	230,78
P4	10	0	225	225,22	-3,35	-9,69	225	225,23
P5	10	70	225	235,85	74,54	-5,77	225	237,10
P6	10	-70	225	235,85	-70,86	-18,43	225	236,61
P7	20	0	225	225,89	-1,7	-19,21	225	225,82
P8	20	60	225	233,72	63,57	-15,74	225	234,34
P9	20	-90	225	243,16	-72,89	-27,81	225	238,14
P10	30	0	225	226,99	-0,13	-31,5	225	227,19
P11	30	80	225	240,68	77,38	-15,14	225	238,42
P12	30	-80	225	240,68	-70,83	-37,88	225	238,91
P13	40	0	225	228,53	4,46	-41,76	225	228,89
P14	40	10	225	228,75	23,23	-42,44	225	230,14
P15	40	-10	225	228,75	-15,39	-44,21	225	229,82
P16	50	0	225	230,49	2,04	-50,43	225	230,59
P17	50	40	225	233,93	50,13	-47,55	225	235,37
P18	50	-40	225	233,93	-30,09	-50,87	225	232,63
P19	-10	0	225	225,22	-1,17	8,24	225	225,15
P20	-10	20	225	226,11	16,58	9,84	225	225,82
P21	-10	-20	225	226,11	-24,21	21,4	225	227,31
P22	-20	0	225	225,89	0,2	25,28	225	226,42
P23	-20	50	225	231,35	51,77	17,49	225	231,54
P24	-20	-80	225	239,64	-73,12	15,76	225	237,11
P25	-30	0	225	226,99	2,14	30,06	225	227,01
P26	-30	15	225	227,49	23,67	34,31	225	228,83
P27	-30	-15	225	227,49	-19,09	31,75	225	228,03
P28	-40	0	225	228,53	5,71	44,66	225	229,46
P29	-40	40	225	232,00	48,34	38,55	225	233,34
P30	-40	-40	225	232,00	-37,19	41,19	225	231,74
P31	-50	0	225	230,49	-5,96	48,37	225	230,22
P32	-50	35	225	233,13	43,18	46,08	225	233,69
P33	-50	-65	225	239,48	-63,22	49,21	225	238,84

Cuadro II.10: Resultados del programa `xmlrpc_deteccionfresas.py` con la cámara situada a 225 mm de la mesa y la cámara perpendicular al plano

A partir de estas mediciones, se puede observar que existe una discrepancia notable entre los ejes de referencia inicialmente supuestos y los que realmente se corresponden con el sistema físico, ya que los resultados obtenidos no guardan una relación directa ni coherente con las coordenadas esperadas según la hipótesis inicial del sistema de ejes, y si se analizan con detenimiento las mediciones, se aprecia que la correspondencia aparente entre los ejes obtenidos y los reales podría responder a una rotación o permutación de los mismo, lo que indica una posible equivalencia o relación entre estos, evidenciando que el único error existente era la consideración de los ejes para realizar las medidas reales, y quedando estos definidos como se representa en la Figura II.25. Esta equivalencia se detalla en la Ecuación II.1.

$$X_{\text{obtenido}} = Y_{\text{supuesto}}$$

$$Y_{\text{obtenido}} = -X_{\text{supuesto}}$$

Ecuación II.1: Equivalencia entre las coordenadas supuestas y obtenidas

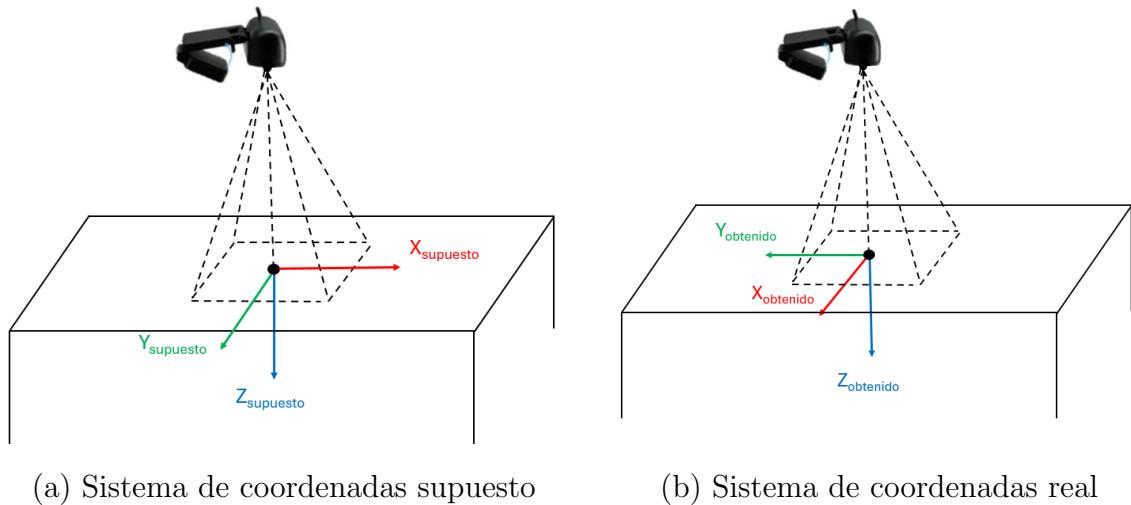


Figura II.25: Representación de los sistemas de coordenadas que se había supuesto en un principio y del real obtenido

Después de verificar que existía un problema en cuanto a la elección de los ejes de coordenadas, se volvieron a realizar pruebas de medición para la toma de datos y su análisis y verificación posterior en el caso de la cámara perpendicular al plano mesa (Figura II.26), siendo esta vez para la serie de puntos que se encuentran en el Cuadro II.11, y dando de esta manera por terminadas este tipo de pruebas dada la exactitud obtenida en estos últimos resultados.

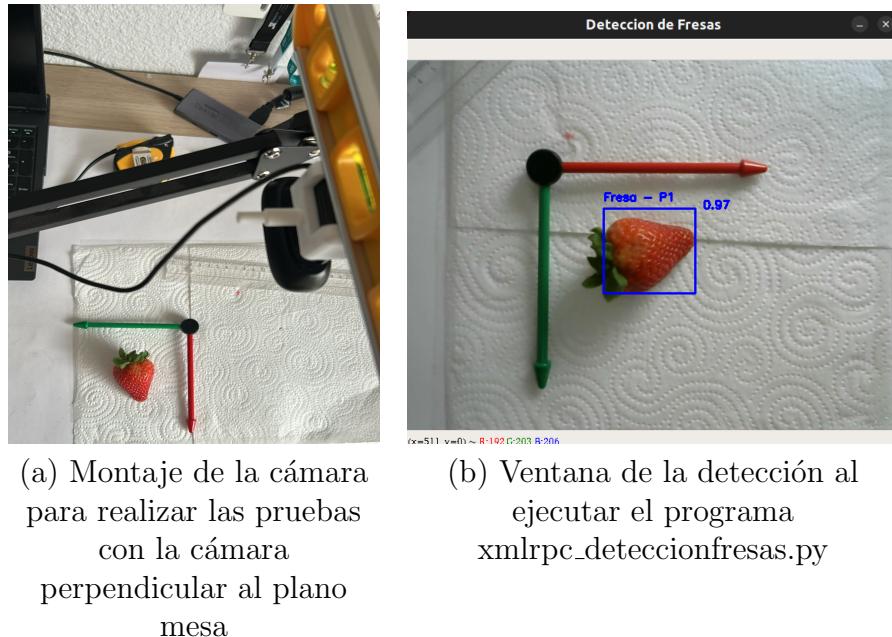


Figura II.26: Representación del montaje y los sistemas de coordenadas obtenidos para las pruebas con la cámara perpendicular al plano de la mesa

Parámetros de la cámara		
Altura (mm)	Rotación (º)	Rotación (rad)
343	0	0

POSICIÓN	Coordenadas reales (mm)				Coordenadas obtenidas (mm)			
	X	Y	Z	Distancia real (mm)	X	Y	Z	Distancia obtenida (mm)
P1	0	0	0	0,00	-0,43	-3,07	0	3,10
P2	0	20	0	20,00	-3,03	24,87	0	25,05
P3	0	-20	0	20,00	-2,24	-33,47	0	33,54
P4	0	100	0	100,00	-2,69	85,78	0	85,82
P5	0	-100	0	100,00	-0,9	-84,27	0	84,27
P6	20	0	0	20,00	27,53	-9,43	0	29,10
P7	20	70	0	72,80	26,24	64,84	0	69,95
P8	20	-70	0	72,80	14,61	-71,50	0	72,98
P9	-20	70	0	72,80	-24,13	71,7	0	75,65
P10	-20	-70	0	72,80	-27,22	-75,87	0	80,61
P11	40	0	0	40,00	36,9	-1,44	0	36,93
P12	40	90	0	98,49	46,09	85,38	0	97,03
P13	40	-90	0	98,49	44,23	-81,31	0	92,56
P14	50	0	0	50,00	51,86	2,94	0	51,94
P15	50	50	0	70,71	50,1	50,14	0	70,88
P16	50	-50	0	70,71	46,23	-47,3	0	66,14
P17	-50	50	0	70,71	-62,27	53,34	0	81,99
P18	-50	-50	0	70,71	-51,39	-61,66	0	80,27
P19	70	0	0	70,00	73,07	6,19	0	73,33
P20	70	35	0	78,26	67,49	43,13	0	80,09
P21	70	-35	0	78,26	75,32	-41,53	0	86,01
P22	90	0	0	90,00	93,93	3,97	0	94,01
P23	90	40	0	98,49	89,84	46,62	0	101,22
P24	90	-40	0	98,49	88	-41,83	0	97,44
P25	-90	40	0	98,49	-92,32	41,56	0	101,24
P26	-90	-40	0	98,49	-92,61	-41,87	0	101,64

Cuadro II.11: Resultados del programa xmlrpc_deteccionfresas.py con la cámara situada a 343 mm de la mesa y la cámara perpendicular al plano

II.3. Pruebas con el robot real

Una vez decidido que la mejor opción para la detección de fresas era utilizar el modelo YOLOv3, implementado con PyTorch en Python, se procedió a realizar las primeras pruebas con el brazo robótico de Universal Robots. El objetivo principal de estas pruebas fue comprobar la correcta comunicación entre el sistema de detección y el robot y evaluar el funcionamiento del sistema en un entorno controlado antes de su aplicación conjunta.

Para ello, en primer lugar, se programó en la Interfaz Gráfica de Usuario (IGR) del robot el programa *visionsimple.urp*¹⁹ para ir a una posición fija que simulaba una posición en el espacio determinada por un sistema de visión externo, tal y como se muestra en la Figura II.27, siendo esto la base del programa final del propio robot.

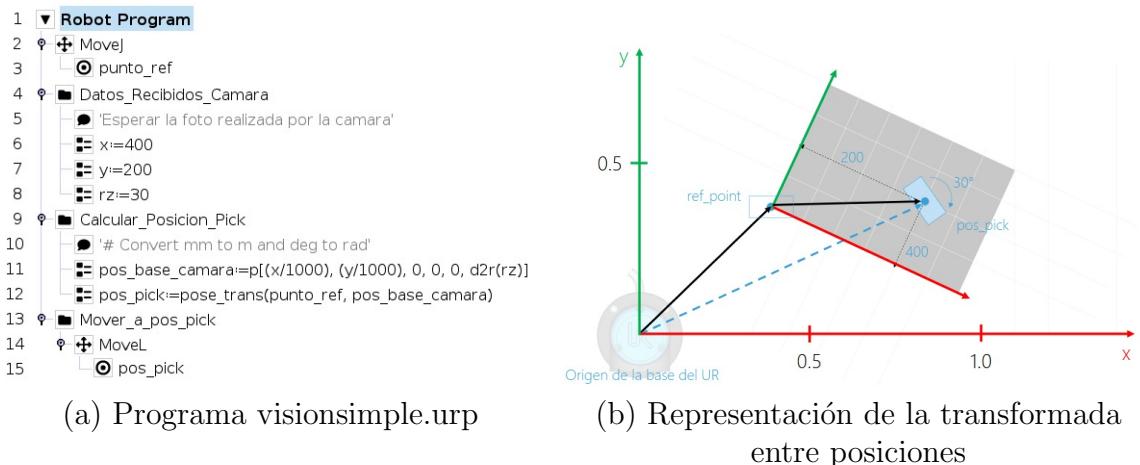


Figura II.27: Representación básica de un programa de un sistema de visión externo

Después de este inicio, se desarrolló el programa *recibir_cadena_socket*²⁰ con el objetivo de establecer una comunicación mediante sockets entre el robot y un servidor externo, donde el robot abría una conexión socket con el servidor y enviaba una cadena de caracteres tipo string con el contenido "listo", como señal de que la comunicación había sido establecida correctamente, para posteriormente esperar tres valores de tipo float de este servidor externo, que eran almacenados en variables internas antes de cerrar la conexión (Figura II.28), todo esto, asegurando que ambos dispositivos se encontraban conectados a la misma red local.

¹⁹<https://github.com/RoboticsURJC/tfg-dcampoamor/blob/main/src/robot/visionsimple.urp>

²⁰https://github.com/RoboticsURJC/tfg-dcampoamor/blob/main/src/robot/recibir_cadena_socket.urp

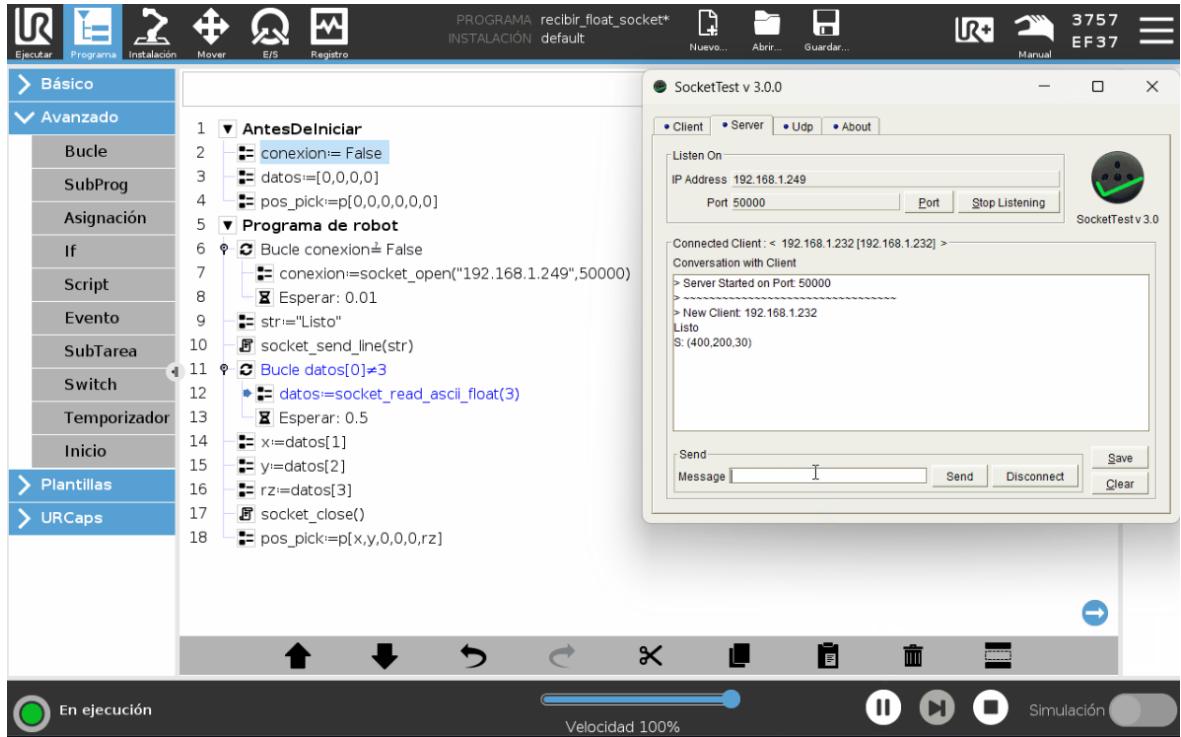


Figura II.28: Programa recibir_cadena_socket.urp

Continuando con las pruebas iniciales del programa de robot *visionsimple.urp* orientadas a enviar un brazo robótico a una posición en el espacio determinada por un sistema de visión externo, se desarrolló y amplió en el programa *prueba_visionsimple.urp*²¹, añadiendo las líneas necesarias para establecer la comunicación con el servidor mediante sockets, leer los datos recibidos, introducidos manualmente desde el programa SocketTest, calcular la posición de objetivo *pos_pick* correspondiente a la posición de recogida de la detección, y enviar estos datos al robot para que se desplazase a dicha posición (ver Figura II.29). Todo esto, en el modo de simulación que permite la interfaz del propio robot, ya que no se estaba seguro de hacia qué posición ni en qué dirección se iba a desplazar, pudiendo ocasionar alguna colisión de realizarse directamente.

²¹https://github.com/RoboticsURJC/tfg-dcampoamor/blob/main/src/robot/prueba_visio nsimple.urp

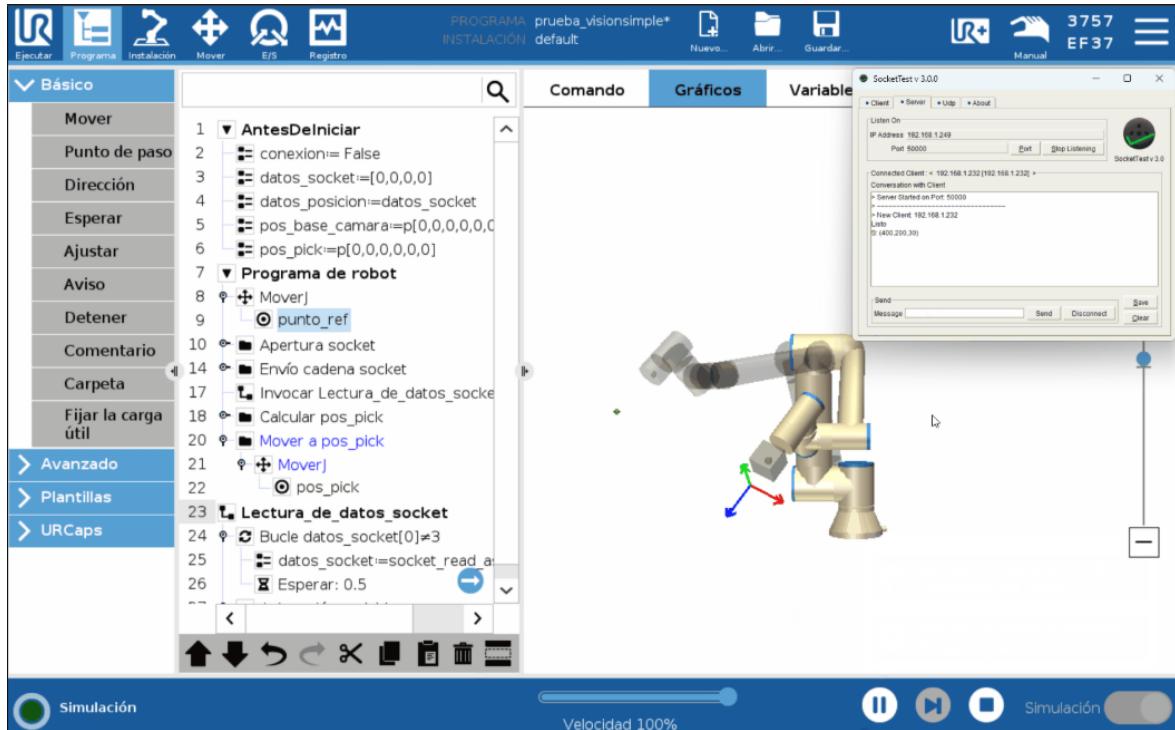


Figura II.29: Programa prueba_visionsimple.urp

Paralelamente a estas pruebas, para conseguir alinear el robot de manera paralela respecto al plano base o cualquier otro plano, se realizó el cálculo de la posición de destino aplicando las rotaciones adecuadas definidas en el sistema de referencia RPY (Roll-Pitch-Yaw), puesto que el brazo robótico por defecto utiliza este sistema como vector de rotación, y se implementó en el programa *Alinear_en_base.urp*²², donde se hizo uso del script *alinearRobot.script*²³. Puesto que este programa era posible usarlo con cualquier plano, se modificaron determinados parámetros en el script para que pudiera utilizarse para alinear el robot con un plano vertical como paso previo a la fase de recolección de fresas.

Una vez establecida la capacidad del robot para alinearse correctamente con distintos planos de trabajo, el siguiente paso consistió en definir el mecanismo de comunicación más adecuado para permitir el envío de posiciones desde un sistema de visión externo y lograr que el robot pudiera recibir en tiempo real las coordenadas de las fresas detectados y desplazarse a la posición de recogida. Para ello, se llevó a cabo un estudio general de las interfaces de cliente y protocolos de comunicación que ofrece

²²<https://github.com/RoboticsURJC/tfg-dcampoamor/blob/main/src/robot/Alinear%20en%20base.urp>

²³<https://github.com/RoboticsURJC/tfg-dcampoamor/blob/main/src/robot/alinearrobot.script>

UR, con el fin de determinar cuál de ellas se adaptaba mejor a los requerimientos del sistema, centrándose en las siguientes:

- Sockets (TCP/IP): El robot UR puede comunicarse con sistemas externos a través del protocolo TCP/IP mediante sockets, tal y como se había probado anteriormente en las pruebas de código para enviar una cadena a un servidor y recibir por socket una lista de enteros y poder realizar la integración de estos datos en el cuerpo del programa principal del robot. En este tipo de conexión, el robot actúa como cliente, mientras que el otro dispositivo actúa como servidor, y donde URScript proporciona instrucciones específicas para abrir y cerrar conexiones, así como para enviar y recibir datos en distintos formatos.
- XML-RPC: Es un protocolo de llamada a procedimiento remoto que utiliza XML para la codificación de datos y los transfiere a través de sockets, permitiendo que la controladora del UR pueda ejecutar funciones remotas con parámetros definidos y recibir respuestas estructuradas, siendo su principal ventaja la capacidad de delegar cálculos complejos a programas externos, superando así las limitaciones de URScript. Además, permite la integración de paquetes de software adicionales en URScript para ampliar la funcionalidad del sistema.
- RTDE (Real-Time Data Exchange): RTDE permite el intercambio de datos en tiempo real entre el robot y aplicaciones externas mediante una conexión TCP/IP estándar, sin comprometer la integridad del sistema en tiempo real de la controladora del UR, siendo útil para sincronizar aplicaciones externas con el robot, interactuar con buses de campo (como Ethernet/IP), manipular entradas/salidas del sistema y monitorizar el estado del robot, incluyendo sus trayectorias y parámetros de seguridad. La funcionalidad RTDE se organiza en dos fases:
 1. Configuración: consiste en la definición de los datos a intercambiar.
 2. Bucle de sincronización: se trata de la transmisión periódica de datos, ya que entre los datos de salida posibles se encuentran el estado del robot, la posición articular, E/S analógicas y digitales, y los registros de propósito general, y en cuanto a datos de entrada, permite modificar E/S digitales y analógicas, así como registros de entrada de propósito general.

- Buses de campo (MODBUS, PROFINET, Ethernet/IP, PROFIsafe): Los buses de campo son protocolos de comunicación ampliamente utilizados en la automatización industrial para la conexión de dispositivos como PLCs, sensores, actuadores y robots. Además, los buses de campo están diseñados para ofrecer una comunicación determinista, es decir, aquella en la que se garantiza que los datos se transmitirán y recibirán en un tiempo exacto y predecible, sin variaciones ni retrasos inesperados, y que sea fiable y con baja latencia, lo que los convierte en una opción adecuada para entornos industriales donde se requiere sincronización precisa y robustez. Estos buses suelen requerir una fase de configuración previa en el software del robot, así como en el sistema externo, y están orientados principalmente a la integración en arquitecturas de automatización industrial, más que al prototipado o desarrollo de aplicaciones personalizadas.

En el contexto de este proyecto, se optó por utilizar la interfaz XML-RPC como método de comunicación entre el sistema de visión, encargado de detectar fresas y calcular sus coordenadas, y el robot UR basándose en varias ventajas que ofrece XML-RPC frente a otras alternativas como los sockets tradicionales o la interfaz RTDE. En primer lugar, permite la llamada directa a funciones remotas definidas en el servidor, lo que simplifica notablemente la integración, ya que el robot puede solicitar de forma estructurada las posiciones objetivo con una única instrucción, además, XML-RPC facilita el intercambio de datos complejos, como listas o vectores, y cuenta con soporte nativo tanto en URScript como en Python, el lenguaje utilizado en el sistema de visión, lo que reduce considerablemente la complejidad de la implementación y mejora la mantenibilidad del sistema. Asimismo, permite concentrar la lógica de decisión en el servidor externo, lo que resulta útil en sistemas escalables donde se pueden añadir más criterios o funcionalidades sin modificar el programa del robot, siendo la opción más adecuada para garantizar una comunicación robusta, eficiente y fácilmente ampliable entre ambos sistemas.

Una vez seleccionado el procedimiento XML-RPC como método de comunicación más adecuado para el sistema, se llevaron a cabo las primeras pruebas básicas para validar su funcionamiento, consistentes en establecer la comunicación entre el robot y un servidor remoto simulado encargado de proporcionar las posiciones objetivo, donde el servidor, que actuaba como una cámara remota ficticia, fue implementado en Python

mediante los script *xmlrpc_server.py*²⁴, *xmlrpc_server_severalpos.py*²⁵ y *xmlrpc_server_3positions.py*²⁶, en función de pequeñas variaciones aplicadas, mientras que el robot ejecutó el programa *xmlrpc_example.urp*²⁷ para conectarse, solicitar la posición y moverse en consecuencia (Figura II.30). Esto permitió verificar que el intercambio de datos y la ejecución remota de funciones se realizaban correctamente, sentando así las bases para la integración futura del sistema de visión real.

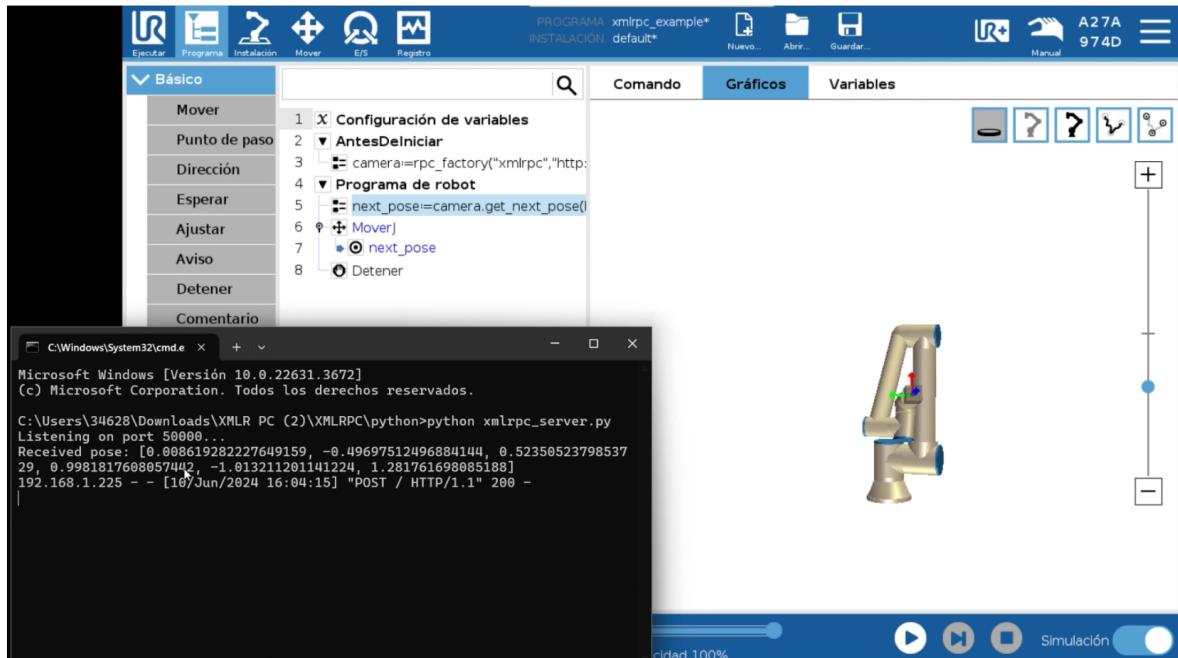


Figura II.30: Programa *xmlrpc_example.urp*

Después de lograr que el robot se desplazara a tres posiciones distintas definidas en una lista, se planteó una mejora en el comportamiento del sistema mediante una lista dinámica para evitar que estas posiciones se repitieran continuamente en bucle, de modo que el robot se desplazase a cada posición una única vez, en orden secuencial, y descartara las posiciones ya procesadas. Para ello, se modificó tanto el script del servidor, generando el programa *xmlrpc_server_listadinamica.py*²⁸ como el programa del robot, adaptando la lógica para gestionar la lista de forma dinámica.

²⁴https://github.com/RoboticsURJC/tfg-dcampoamor/blob/main/src/robot/XMLR%20PC/XMLRPC/python/xmlrpc_server.py

²⁵https://github.com/RoboticsURJC/tfg-dcampoamor/blob/main/src/robot/XMLR%20PC/XMLRPC/python/xmlrpc_server_severalpos.py

²⁶https://github.com/RoboticsURJC/tfg-dcampoamor/blob/main/src/robot/XMLR%20PC/XMLRPC/python/xmlrpc_server_3positions.py

²⁷https://github.com/RoboticsURJC/tfg-dcampoamor/blob/main/src/robot/XMLR%20PC/XMLRPC/xmlrpc_example.urp

²⁸https://github.com/RoboticsURJC/tfg-dcampoamor/blob/main/src/robot/XMLR%20PC/XMLRPC/python/xmlrpc_server_listadinamica.py

Tras estas pruebas iniciales, se continuó empleando el script en Python encargado de detectar puntos de color verde, añadiendo en paralelo la tarea de ejecutar el servidor XML-RPC mediante el programa *xmlrpc_server.py* desde el terminal del ordenador, mientras que, de manera simultánea en el robot se ejecutaba el programa *xmlrpc-example.urp*, que había sido previamente modificado respecto a su versión original utilizada en las pruebas iniciales descritas anteriormente, incluyendo nuevas variables en el script en Python, y obteniendo una respuesta adecuada del sistema, puesto que el programa enviaba correctamente las coordenadas del punto detectado al robot, y, una vez alcanzada la posición, al detectar un nuevo punto en una ubicación distinta, el robot se desplazaba automáticamente hacia un nuevo destino (Figura II.31).

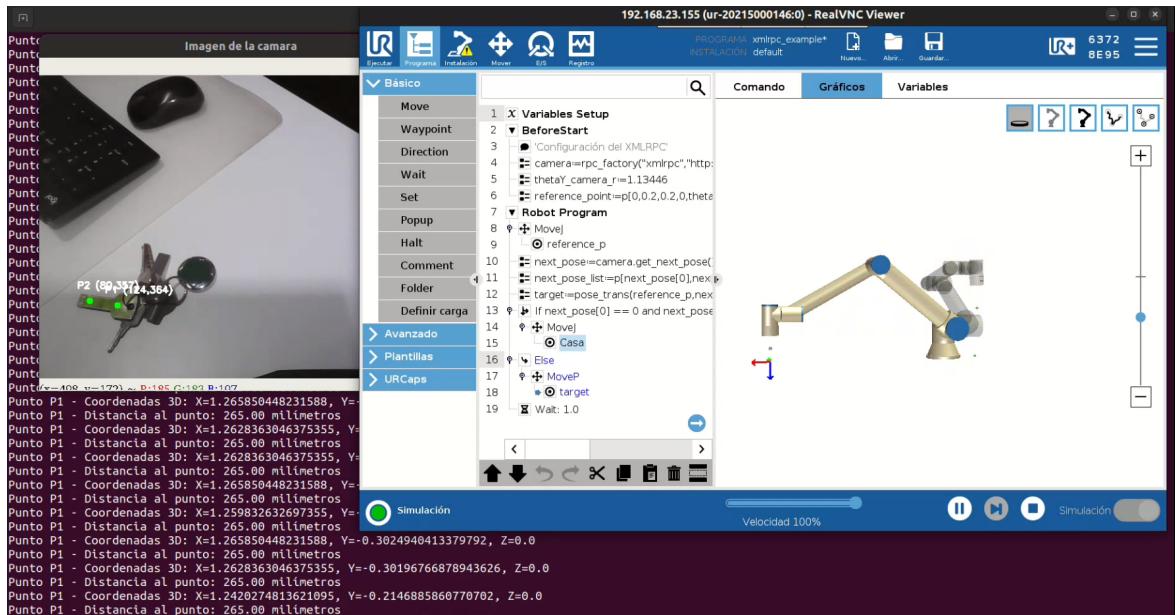


Figura II.31: Pruebas funcionales con el programa pinhole.py y el robot real

A continuación, se iniciaron las pruebas de envío de las coordenadas de detección de fresas al robot real, utilizando como base el programa *xmlrpc_deteccionfresas.py*²⁹, y logrando establecer con éxito la detección de fresas mediante visión artificial, la transmisión de las coordenadas al robot y la ejecución del movimiento correspondiente a través de la variable *next_pose* en el programa del robot, como se observa en la Figura II.32.

²⁹https://github.com/RoboticsURJC/tfg-dcampoamor/blob/main/src/robot/xmlrpc_deteccionfresas.py

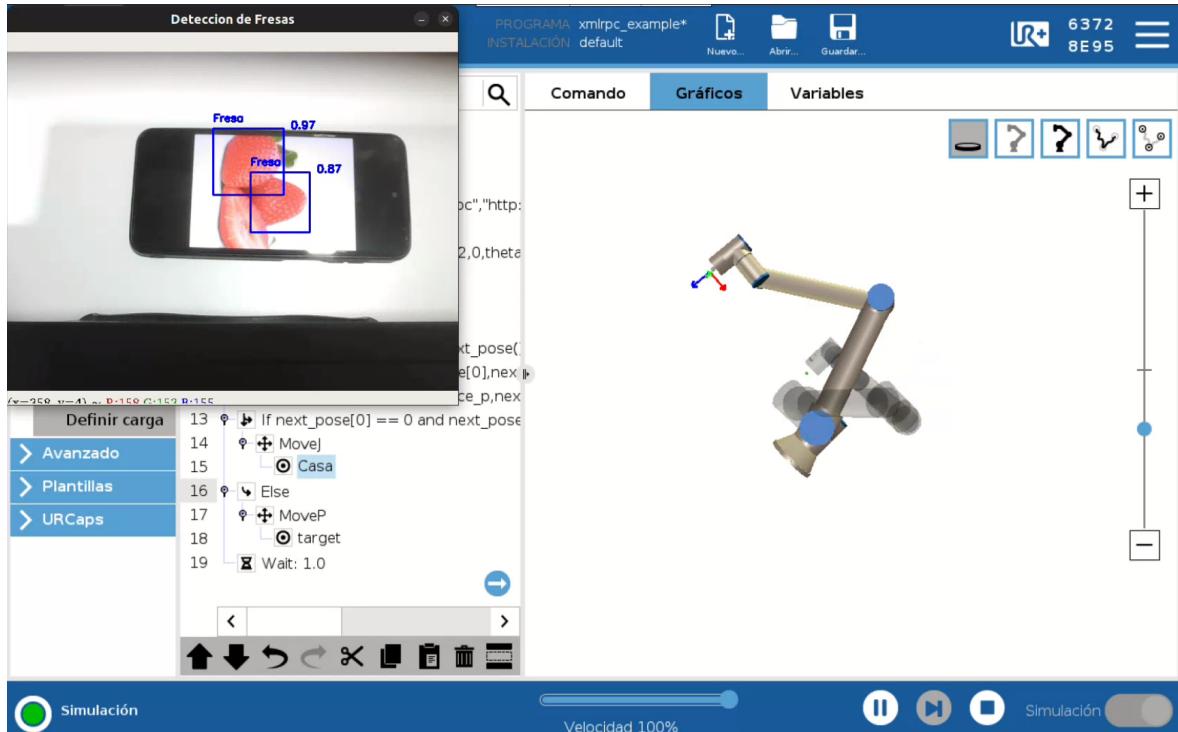


Figura II.32: Primeras pruebas detección de fresas y envío de las posiciones al UR

Una vez verificado que el programa en Python *xmlrpc_deteccionfresas.py*, encargado de gestionar la detección de fresas mediante inteligencia artificial funcionaba correctamente, incluyendo el cálculo de las coordenadas y distancias a partir de la imagen de cámara, y habiendo confirmado previamente la validez de dichas coordenadas a través del análisis detallado del sistema de referencia y la correcta identificación del eje real de coordenadas, conociendo de igual manera que la comunicación con el robot UR a través de XML-RPC se realizaba de forma satisfactoria, se procedió a realizar pruebas en el entorno definitivo del sistema con un robot de Universal Robots modelo UR3e. Para ello se utilizó el programa de robot *xmlrpc_deteccion_fresas_v3.urp*³⁰, programado de tal manera que si no recibía ninguna detección que variase en un umbral definido previamente de la última detección, no se movía de la posición de *Casa*. Estas pruebas comenzaron considerando el plano de la mesa como hipótesis suelo, y consistieron inicialmente en la detección y seguimiento de una única fresa para verificar la estabilidad del sistema en condiciones reales y, posteriormente, se avanzó hacia la detección múltiple de fresas, evaluando el comportamiento del sistema al identificar y gestionar varias coordenadas objetivo de forma secuencial (figuras II.33 y II.34).

³⁰https://github.com/RoboticsURJC/tfg-dcampoamor/blob/main/src/robot/xmlrpc_deteccion_fresas_v3.urp



(a) Detección simple en el plano horizontal



(b) Detección múltiple en el plano horizontal

Figura II.33: Disposición de las detecciones para un plano horizontal con un UR3e

```

dcampoamor@robotics:~/deteccionobj/deteccion-objetos-video$ python xmlrpc_deteccionfresas.py --model_def config/yolov3-custom.cfg --weights_path checkpoints/ovov3_ckpt_99.pth --class_path data/custom/classes.names --conf_thres 0.85
[INFO] Iniciando programa...
[INFO] Iniciando detección de fresas en el frame actual...
[DEBUG] Matriz RT de la cámara:
[[ 1.  0.  0.  0.]
 [ 0.  1.  0.  0.]
 [ 0.  0.  1. -400.]
 [ 0.  0.  0.  1.]]
[INFO] Parámetros de la cámara cargados correctamente.
Servidor XML-RPC corriendo en el puerto 50000...
Punto P1 - Coordenadas 3D: X=69.43, Y=-17.89, Z=400.00
Punto P1 - Distancia al punto: 350.41 milímetros
Punto P1 - Coordenadas 3D: X=51.37, Y=-4.53, Z=400.00
Punto P1 - Distancia al punto: 346.86 milímetros
Punto P1 - Coordenadas 3D: X=39.84, Y=1.11, Z=400.00
Punto P1 - Distancia al punto: 345.32 milímetros
Punto P1 - Coordenadas 3D: X=44.44, Y=3.13, Z=400.00
Punto P1 - Distancia al punto: 345.88 milímetros
Punto P1 - Coordenadas 3D: X=36.55, Y=3.41, Z=400.00
Punto P1 - Distancia al punto: 344.96 milímetros
[DEBUG] Envío de la última posición detectada: (36.5510234063158, 3.40836755150453
192.168.23.110 - - [01/May/2025 15:04:09] "POST / HTTP/1.1" 200 -
Punto P1 - Coordenadas 3D: X=38.21, Y=-1.48, Z=400.00
Punto P1 - Distancia al punto: 345.13 milímetros
[DEBUG] Envío de la última posición detectada: (38.21327870491555, -1.484004419853
192.168.23.110 - - [01/May/2025 15:04:09] "POST / HTTP/1.1" 200 -
Punto P1 - Coordenadas 3D: X=44.08, Y=1.15, Z=400.00
Punto P1 - Distancia al punto: 345.82 milímetros
[DEBUG] Envío de la última posición detectada: (44.08837579015321, 1.1546974483251038, 400.0)
192.168.23.110 - - [01/May/2025 15:04:10] "POST / HTTP/1.1" 200 -
Punto P1 - Coordenadas 3D: X=37.87, Y=-8.02, Z=400.00
Punto P1 - Distancia al punto: 352.42 milímetros
Punto P1 - Coordenadas 3D: X=-8.87, Y=-80.41, Z=400.00
Punto P1 - Distancia al punto: 352.41 milímetros
[DEBUG] Envío de la última posición detectada: (-8.86635699749483, -80.4073364367676, 400.0)
192.168.23.110 - - [01/May/2025 15:04:17] "POST / HTTP/1.1" 200 -
Punto P1 - Coordenadas 3D: X=76.57, Y=44.43, Z=400.00
Punto P1 - Distancia al punto: 354.24 milímetros
[DEBUG] Envío de la última posición detectada: (76.57311070024342, 44.43314314487785, 400.0)
192.168.23.110 - - [01/May/2025 15:04:21] "POST / HTTP/1.1" 200 -

```

(a) Ejecución del programa en la terminal para la detección simple

```

dcampoamor@robotics:~/deteccionobj/deteccion-objetos-video$ python xmlrpc_deteccionfresas.py --model_def config/yolov3-custom.cfg --weights_path checkpoints/ovov3_ckpt_99.pth --class_path data/custom/classes.names --conf_thres 0.85
[INFO] Iniciando programa...
[INFO] Iniciando detección de fresas en el frame actual...
[DEBUG] Matriz RT de la cámara:
[[ 1.  0.  0.  0.]
 [ 0.  1.  0.  0.]
 [ 0.  0.  1. -400.]
 [ 0.  0.  0.  1.]]
[INFO] Parámetros de la cámara cargados correctamente.
Servidor XML-RPC corriendo en el puerto 50000...
Punto P1 - Coordenadas 3D: X=34.86, Y=84.50, Z=400.00
Punto P1 - Distancia al punto: 354.97 milímetros
Punto P2 - Coordenadas 3D: X=74.04, Y=77.79, Z=400.00
Punto P2 - Distancia al punto: 359.42 milímetros
Punto P3 - Coordenadas 3D: X=-3.31, Y=-13.71, Z=400.00
Punto P3 - Distancia al punto: 343.29 milímetros
Punto P1 - Coordenadas 3D: X=-35.01, Y=84.78, Z=400.00
Punto P1 - Distancia al punto: 355.05 milímetros
Punto P2 - Coordenadas 3D: X=74.71, Y=80.03, Z=400.00
Punto P2 - Distancia al punto: 360.05 milímetros
Punto P3 - Coordenadas 3D: X=-3.06, Y=-14.48, Z=400.00
Punto P3 - Distancia al punto: 345.32 milímetros
Punto P4 - Coordenadas 3D: X=-23.65, Y=-76.09, Z=400.00
Punto P4 - Distancia al punto: 352.13 milímetros
[DEBUG] Envío de la última posición detectada: (-23.65269112675015, -76.08659926961778
192.168.1.110 - - [04/may/2025 12:08:43] "POST / HTTP/1.1" 200 -

```

(b) Ejecución del programa en la terminal para la detección múltiple

Figura II.34: Ejecución del programa en la terminal para un plano horizontal con un UR3e

Tras verificar la precisión, exactitud y el correcto funcionamiento del sistema en el plano horizontal, considerado inicialmente como hipótesis suelo, se procedió a realizar pruebas en el plano vertical, en línea con el objetivo final del TFG para desarrollar un sistema de detección y recolección de fresas adaptado a un entorno de cultivo en huerto vertical (Figura II.35), generando el plano pared común a todos los elementos.

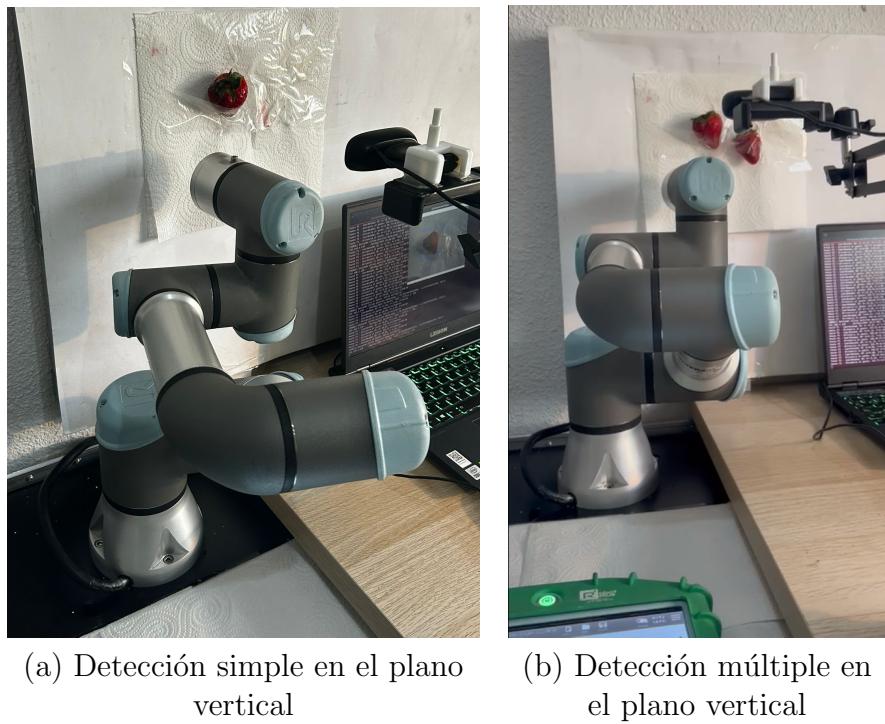


Figura II.35: Disposición de las detecciones para un plano vertical con un UR3e

Una vez completadas las pruebas en el plano vertical y verificado el correcto funcionamiento del sistema en el entorno de huerto vertical, se repitieron los ensayos en el plano vertical utilizando un modelo distinto de robot con mayor alcance, concretamente un UR5e. Para esto, se creó el programa *xmlrpc_deteccion_fresas_vertical.urp*³¹, ya que durante las pruebas iniciales en disposición vertical con el robot UR3e, este presentaba limitaciones físicas de movimiento que restringían su capacidad para alcanzar determinadas posiciones, como se puede observar en las imágenes correspondientes, por lo que al ampliar el rango de trabajo del brazo robótico, fue posible acceder a un mayor número de puntos dentro del entorno de prueba, lo que permitió validar de forma más completa el sistema y confirmar la fiabilidad de la detección, proyección y ejecución del movimiento, considerándolo válido para su uso destinado. Así mismo, para estas pruebas, se mejoró el método de sujeción de las fresas, pasando de sujetarlas con cinta adhesiva a una cartulina, que a su vez también estaba fijada a la pared con esa misma cinta, a llevarlo a cabo con alfileres que sujetaban las fresas sobre una plancha de espuma, sujetada por la presión que ejercía la propia mesa en la que se encontraba montado el robot contra la pared.

³¹https://github.com/RoboticsURJC/tfg-dcampoamor/blob/main/src/robot/xmlrpc_deteccion_fresas_vertical.urp

Bibliografía

- [Alvear-Puertas et al., 2017] Alvear-Puertas, V., Rosero-Montalvo, P., Peluffo-Ordóñez, D., and Pijal-Rojas, J. (2017). Internet de las Cosas y Visión Artificial, Funcionamiento y Aplicaciones: Revisión de Literatura. *Enfoque UTE*, 8:244–256.
- [Barrientos, 2002] Barrientos, A. (2002). Nuevas aplicaciones de la robótica. robots de servicio. *Avances en robótica y visión por computador. Cuenca, Ediciones Castilla-La Mancha*, 288.
- [Basogain, 2008] Basogain, X. (2008). Redes neuronales artificiales y sus aplicaciones. *Dpto. Ingeniería de Sistemas y Automática, Escuela Superior de Ingeniería Bilbao.*, page 79. Open Course Ware. [En línea] disponible en <http://ocw.ehu.es/ensenanzas-tecnicas/redes-neuronales-artificiales-y-sus-aplicaciones/Course-listing>.
- [Beasley, 2012] Beasley, R. A. (2012). Medical Robots: Current Systems and Research Directions. *Journal of Robotics*.
- [Bradski and Kaehler, 2008] Bradski, G. and Kaehler, A. (2008). *Learning OpenCV*. O'Reilly Media, Inc.
- [Cabanillas, 2009] Cabanillas, F. (2009). Preparan en lepe el prototipo final de un robot para la recogida de fresas.
- [Culjak et al., 2012] Culjak, I., Abram, D., Pribanic, T., Dzapo, H., and Cifrek, M. (2012). A brief introduction to OpenCV. In *2012 proceedings of the 35th international convention MIPRO*, pages 1725–1730. IEEE.
- [Cusano, 2022] Cusano, N. (2022). Cobot and Sobot: For a new Ontology of Collaborative and Social Robots. *Foundations of Science*, pages 1–13.
- [De Preter et al., 2018] De Preter, A., Jan Anthonis, and Josse De Baerdemaeker (2018). Development of a robot for harvesting strawberries. *IFAC-PapersOnLine*, 51. 6th IFAC Conference on Bio-Robotics BIOROBOTICS 2018.

- [Dinamarca, 2018] Dinamarca, A. (2018). Aprendizaje y análisis de redes neuronales artificiales profundas. Tesina de grado, Universidad Nacional de Cuyo. Facultad de Ciencias Exactas y Naturales.
- [Dupont et al., 2021] Dupont, P. E., Nelson, B. J., Goldfarb, M., Hannaford, B., Meniciassi, A., O’Malley, M. K., Simaan, N., Valdastri, P., and Yang, G.-Z. (2021). A decade retrospective of medical robotics research from 2010 to 2020. *Science Robotics*, 6(60).
- [EcoInventos.com, 2024] EcoInventos.com (2024). La primera granja vertical de interior del mundo producirá 1,8 millones de kg de fresas al año.
- [Fountas et al., 2020] Fountas, S., Mylonas, N., Malounas, I., Rodias, E., Hellmann Santos, C., and Pekkeriet, E. (2020). Agricultural robotics for field operations. *Sensors*, 20(9).
- [García Santillán and Caranqui Sánchez, 2015] García Santillán, I. D. and Caranqui Sánchez, V. M. (2015). La visión artificial y los campos de aplicación. *Tierra Infinita*, 1:98–108.
- [Gasparetto and Scalera, 2019] Gasparetto, A. and Scalera, L. (2019). A Brief History of Industrial Robotics in the 20th Century. *Advances in Historical Studies*, 8:24–35.
- [Gonzalez-Aguirre et al., 2021] Gonzalez-Aguirre, J. A., Osorio-Oliveros, R., Rodríguez-Hernández, K. L., Lizárraga-Iturralde, J., Morales Menendez, R., Ramírez-Mendoza, R. A., Ramírez-Moreno, M. A., and Lozoya-Santos, J. d. J. (2021). Service Robots: Trends and Technology. *Applied Sciences*, 11(22):1–22.
- [González and de Mántaras Badia, 2017] González, P. M. and de Mántaras Badia, R. L. (2017). *Inteligencia Artificial*. LOS LIBROS DE LA CATARATA.
- [Hardy, 2001] Hardy, T. (2001). IA (Inteligencia Artificial). *Polis: Revista Latinoamericana*, (2):18.
- [ISO/TC299, 2021] ISO/TC299 (2021). *ISO 8373:2021 Robotics — Vocabulary*, pages 1–22. International Organization for Standardization. Only informative sections of standards are publicly available.
- [Janiesch et al., 2021] Janiesch, C., Zschech, P., and Heinrich, K. (2021). Machine learning and deep learning. *Electronic Markets*, 31:685–695.

- [Koditschek, 2021] Koditschek, D. E. (2021). What Is Robotics? Why Do We Need It and How Can We Get It? *Annual Review of Control, Robotics, and Autonomous Systems*, 4(1):1–33.
- [Kraevsky and Rogatkin, 2010] Kraevsky, S. and Rogatkin, D. (2010). Medical robotics: the first steps of medical robots. *Russian Journal: Technologies of live systems*, 7(4):3–14.
- [Martínez Madruga, 2022] Martínez Madruga, J. (2022). Sistema de detección de emociones faciales mediante técnicas de Machine Learning adaptado a ROS para un robot de bajo coste basado en Raspberry Pi. Trabajo de fin de grado, Universidad Rey Juan Carlos.
- [Oktarina et al., 2020] Oktarina, Y., Dewi, T., Risma, P., and Nawawi, M. (2020). Tomato harvesting arm robot manipulator; a pilot project. *Journal of Physics: Conference Series*, 1500.
- [Ponce Gallegos et al., 2014] Ponce Gallegos, J. C., Torres Soto, A., Quezada Aguilera, F. S., Silva Srock, A., Martínez Flor, E. U., Casali, A., Scheihing, E., Túpac Valdivia, Y. J., Torres Soto, M. D., Ornelas Zapata, F. J., et al. (2014). *Inteligencia artificial*. Iniciativa Latinoamericana de Libros de Texto Abiertos LATIn.
- [Raj and Seamans, 2019] Raj, M. and Seamans, R. (2019). Primer on artificial intelligence and robotics. *Journal of Organization Design*, 8:1–14.
- [Redmon et al., 2016] Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, Real-Time Object Detection. *Computer Vision and Pattern Recognition (CVPR)*.
- [Redmon and Farhadi, 2018] Redmon, J. and Farhadi, A. (2018). Yolov3: An Incremental Improvement. *arXiv preprint*, abs/1804.02767.
- [Romero-Tamarit et al., 2020] Romero-Tamarit, A., Reig-Viader, R., Estrada-Sabadell, M. D., and Espallargues-Carreras, M. (2020). Eficacia, efectividad, seguridad y eficiencia de la cirugía robótica con el sistema quirúrgico Da Vinci.
- [Sandoval Serrano et al., 2018] Sandoval Serrano, L. J. et al. (2018). Algoritmos de aprendizaje automático para análisis y predicción de datos. *Revista Tecnológica*; no. 11.

- [Siswantoro et al., 2013] Siswantoro, J., Prabuwono, A. S., and Abdullah, A. (2013). Real World Coordinate from Image Coordinate Using Single Calibrated Camera Based on Analytic Geometry. In *Soft Computing Applications and Intelligent Systems*, pages 1–11. Springer Berlin Heidelberg.
- [Sánchez Martín et al., 2007a] Sánchez Martín, F., Millán Rodríguez, F., Salvador Bayarri, J., Palou Redorta, J., Rodríguez Escovar, F., Esquena Fernández, S., and Villavicencio Mavrich, H. (2007a). Historia de la robótica: de Arquitas de Tarento al robot Da Vinci (Parte I). *Actas Urológicas Españolas*, 31:69 – 76.
- [Sánchez Martín et al., 2007b] Sánchez Martín, F., Millán Rodríguez, F., Salvador Bayarri, J., Palou Redorta, J., Rodríguez Escovar, F., Esquena Fernández, S., and Villavicencio Mavrich, H. (2007b). Historia de la robótica: de Arquitas de Tarento al Robot Da Vinci. (Parte II). *Actas Urológicas Españolas*, 31:185–196.
- [Universal Robots A/S, 2018] Universal Robots A/S (2018). *UR e-series brochure*.
- [Universal Robots A/S, 2024] Universal Robots A/S (2024). *Service manual - e-Series*. Last modified on Nov 18, 2024.
- [Universal Robots A/S, 2025a] Universal Robots A/S (2025a). *User manual - UR10e e-Series - SW 5.20*. Last modified on Jan 13, 2025.
- [Universal Robots A/S, 2025b] Universal Robots A/S (2025b). *User manual - UR3e e-Series - SW 5.20*. Last modified on Jan 13, 2025.
- [Universal Robots A/S, 2025c] Universal Robots A/S (2025c). *User manual - UR5e e-Series - SW 5.20*. Last modified on Jan 13, 2025.
- [Vega and Cañas, 2021] Vega, J. and Cañas, J. M. (2021). Open Vision System for Low-Cost Robotics Education. *Electronics*, 85.
- [Xiong et al., 2019] Xiong, Y., Peng, C., Grimstad, L., From, P. J., and Isler, V. (2019). Development and field evaluation of a strawberry harvesting robot with a cable-driven gripper. *Computers and Electronics in Agriculture*, 157:392–402.
- [Zamalloa et al., 2017] Zamalloa, I., Kojcev, R., Hernández, A., Muguruza, I., Usategui, L., Bilbao, A., and Mayoral, V. (2017). Dissecting Robotics - historical overview and future perspectives.