



GRADO EN INGENIERÍA DE TECNOLOGÍAS INDUSTRIALES

Escuela Superior de Ciencias Experimentales y Tecnología

Curso académico 2024-2025

Trabajo Fin de Grado

Sistema de reconocimiento por visión de maduración de frutos
para su recolección con un brazo robótico

Autor: David Campoamor Medrano

Tutor: Julio Vega Pérez



Este trabajo se distribuye bajo los términos de la licencia internacional CC BY-NC-SA International License (Creative Commons AttributionNonCommercial-ShareAlike 4.0). Usted es libre de *(a) compartir*: copiar y redistribuir el material en cualquier medio o formato; y *(b) adaptar*: remezclar, transformar y crear a partir del material. El licenciador no puede revocar estas libertades mientras cumpla con los términos de la licencia:

- *Atribución.* Usted debe dar crédito de manera adecuada, brindar un enlace a la licencia, e indicar si se han realizado cambios. Puede hacerlo en cualquier forma razonable, pero no de forma tal que sugiera que usted o su uso tienen el apoyo de la licenciatante.
- *No comercial.* Usted no puede hacer uso del material con propósitos comerciales.
- *Compartir igual.* Si remezcla, transforma o crea a partir del material, debe distribuir su contribución bajo la misma licencia del original.

Documento de David Campoamor Medrano.

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, xx de xxxxxxx de 20xx

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. (Food and Drug Administration)*

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*

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	3
1.1.2. Robots de servicio	11
1.1.3. Robots médicos	14
1.2. Inteligencia Artificial	16
1.3. Visión Artificial	17
1.4. Machine Learning	19
1.5. Deep Learning	20
2. Estado del arte	23
3. Objetivos	31
3.1. Descripción del problema	31
3.2. Requisitos	32
3.3. Competencias	33
3.4. Metodología	35
3.5. Plan de trabajo	37
4. Plataforma de desarrollo	39
4.1. Hardware	39
4.1.1. Cámara Logitech C270 HD	39
4.1.2. Soporte de brazo articulado	40
4.1.3. Ordenador principal	40
4.1.4. Robot de <i>Universal Robots</i> de la gama <i>e-series</i>	41
4.1.5. Comunicaciones	42
4.2. Software	42
4.2.1. Ubuntu	42
4.2.2. Polyscope	43

4.2.3. Python	44
4.2.4. PyTorch	45
4.2.5. NumPy	45
4.2.6. OpenCV	45
4.2.7. OpenGL	46
4.2.8. SocketTest	46
4.2.9. XML-RPC	47
4.2.10. Anaconda	47
4.2.11. YOLOv3	48
5. Sistema de reconocimiento por visión de maduración de frutos para su recolección con un brazo robótico	50
5.1. Snippets	50
5.2. Verbatim	51
5.3. Ecuaciones	51
5.4. Tablas o cuadros	52
6. Experimentos	53
6.1. Detección con YOLOv3 y TensorFlow	53
6.1.1. Pruebas con imágenes	53
6.1.2. Pruebas con vídeo en tiempo real	57
6.2. Detección con YOLOv3 y PyTorch	63
6.2.1. Pruebas con modelos preentrenados	63
6.2.2. Entrenamiento del modelo	64
6.2.3. Calibrado de la cámara	66
6.2.4. Pruebas detección de fresas en tiempo real	67
6.3. Pruebas con el robot real	83
7. Conclusiones	86
7.1. Conclusiones	86
7.2. Corrector ortográfico	87

Índice de figuras

1.1.	Primer robot industrial	3
1.2.	Standford Arm	4
1.3.	Robot Cincinnati Milacron T3	5
1.4.	Uno de los primeros prototipos de robot SCARA	6
1.5.	Robot ABB IRB 360 Flexpicker	6
1.6.	Robot Motoman DA-20	7
1.7.	LWR3	8
1.8.	Robots utilizados para el desarrollo de ROS	9
1.9.	UR5 con su controladora	10
1.10.	Universal Robots e-Series	10
1.11.	Robot aspirador Roomba de iRobot	12
1.12.	Robots de inspección y mantenimiento	12
1.13.	Robots de educación	13
1.14.	Robots de logística	13
1.15.	Robots de entretenimiento	14
1.16.	Robot Da Vinci	15
1.17.	Modelos de inteligencia	17
1.18.	Diagrama de Venn de la relación entre distintas áreas de la IA	20
1.19.	Modelo biológico de una neurona genérica (izquierda) y el respectivo modelo matemático (derecha)	21
1.20.	Arquitectura de una red neuronal	22
2.1.	Representación de varios tipos de robots agrícolas	23
2.2.	Ilustración global del rendimiento general de los robots revisados	24
2.3.	Agrobot	25
2.4.	Diseño conceptual del robot de recogida con sus componentes	26
2.5.	Robot agrícola Dogtooth	27
2.6.	Montaje del hardware en una explotación de fresas	28
2.7.	Robot recolector de tomates	30

3.1. Ciclo de la metodología DMADV	36
4.1. Cámara Logitech C270 HD ¹⁶	39
4.2. Soporte de brazo articulado ¹⁷	40
4.3. Gama e-series de Universal Robots ²⁰	41
4.4. Pantalla principal de la interfaz de Polyscope 5	44
4.5. Pruebas realizadas con SocketTest para verificar la comunicación robot-servidor externo	47
6.1. Entrenamiento del algoritmo con TensorFlow	54
6.2. Resultado de la detección en imágenes con TensorFlow	55
6.3. Resultado del reentrenamiento de la detección en imágenes con TensorFlow	56
6.4. Pruebas de detección de fresas en imágenes con TensorFlow	57
6.5. Modelo ssd_mobilenet_v2_320x320_coco17_tpu-8	58
6.6. Modelo efficientdet_d4_coco17_tpu-32	58
6.7. Modelo faster_rcnn_resnet50_v1_640x640_coco17_tpu-8	58
6.8. Detección de fresas en webcam con TensorFlow con modelos no pre-entrenados (ssd mobilenet v2 320x320)	60
6.9. Gráficas de la confianza de detección obtenida en las pruebas según la luminosidad para el modelo ssd mobilenet v2	61
6.10. Gráficas de la media de los porcentajes de confianza obtenidos en las pruebas de detección según la luminosidad para el modelo ssd mobilenet v2	61
6.11. Detección de fresas en Jupyter Notebook	62
6.12. Detección con Pytorch	64
6.13. Etiquetado de las imágenes con labelImg	65
6.14. Calibración de la cámara C270 de Logitech	66
6.15. Medición del ángulo de rotación de la cámara mediante la aplicación de ERGONAUTAS RULER	67
6.16. Primeras pruebas de detección con PyTorch y Python	67
6.17. Primeras pruebas de la estimación de las coordenadas y la distancia de la detección a la cámara	69
6.18. Geometría basada en el modelo de cámara estenopeica	70
6.19. Pruebas para determinar la configuración del sistema de coordenadas de la cámara	71
6.20. Esquema de la rotación de la cámara	72
6.21. Cálculo de la distancia de la cámara a la detección	73

6.22. Detección múltiple simultánea de post-it	74
6.23. Representación de las detecciones con OpenGL	75
6.24. Detección de fresas e integración con el sistema de cálculo de coordenadas y distancias	76
6.25. Proyección con OpenGL de las detecciones y el campo visual de la cámara	79
6.26. Representación de los sistemas de coordenadas que se habia supuesto en un principio y del real obtenido	81
6.27. Representación del montaje y los sistemas de coordenadas obtenidos para las pruebas con la cámara perpendicular al plano de la mesa . . .	82
6.28. Representación básica de un programa de un sistema de visión externo	83

Listado de códigos

5.1. Función para buscar elementos 3D en la imagen	50
5.2. Cómo usar un Slider	51

Listado de ecuaciones

5.1. Ejemplo de ecuación con fracciones	51
5.2. Ejemplo de ecuación con array y letras y símbolos especiales	51
6.1. Equivalencia entre las coordenadas supuestas y obtenidas	81

Índice de cuadros

1.1. Procesos de la visión artificial	18
5.1. Parámetros intrínsecos de la cámara	52
6.1. Distribución de las imágenes utilizadas para el entrenamiento del modelo	54
6.2. Comparacion entre coordenadas reales y obtenidas (en mm)	69
6.3. Resultados del programa pinhole.py con valores de Z positivos	71
6.4. Resultados del programa pinhole.py con valores de Z negativos	72
6.5. Resultados del programa pinhole.py con el valor ajustado de rotación de la cámara	73
6.6. Resultados del programa xmlrpc_deteccionfresas.py con la cámara situada a 145 mm de la mesa y la cámara rotada 59 grados	76
6.7. Resultados del programa xmlrpc_deteccionfresas.py con la cámara situada a 125 mm de la mesa y la cámara rotada 59 grados	77
6.8. Resultados del programa xmlrpc_deteccionfresas.py con la cámara situada a 225 mm de la mesa y la cámara rotada 69 grados	77
6.9. Resultados del programa xmlrpc_deteccionfresas.py con la cámara situada a 180 mm de la mesa y la cámara rotada 58 grados	78
6.10. Resultados del programa xmlrpc_deteccionfresas.py con la cámara situada a 225 mm de la mesa y la cámara perpendicular al plano	80
6.11. Resultados del programa xmlrpc_deteccionfresas.py con la cámara situada a 343 mm de la mesa y la cámara perpendicular al plano	82
6.12. Programa recibir_cadena_socket.urp	84
6.13. Programa prueba_visionsimple.urp	85

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 [?]. 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* [?].

El término robot fue utilizado por primera vez por Karel Capek en su obra de teatro *Rossum's Universal Robots*, publicada en 1920. Esta palabra viene del vocablo checo *robo* que significa trabajo, en el sentido de la obligatoriedad, entendido como servidumbre, trabajo forzado o esclavitud [?]. 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 [?].

Históricamente, las civilizaciones antiguas, como la egipcia y la griega, dieron los primeros pasos en lo que se puede denominar robótica clásica, construyendo autómatas y mecanismos diseñados para imitar acciones humanas, con características mecánicas rudimentarias. Con el paso del tiempo, la ciencia y la ingeniería avanzaron, y los conceptos de la robótica comenzaron a tomar forma más definida hasta que, en el siglo XX, con el desarrollo de la ingeniería en sus diferentes ramas (mecánica, electrónica, informática, telecomunicaciones), 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 [?].

Partiendo de todos estos avances y del interés por automatizar las tareas de producción, la robótica va adquiriendo un gran desarrollo [?]. 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 médicos.

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 [?].

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 [?].

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 [?].



(a) Joseph Engelberger y George Devol



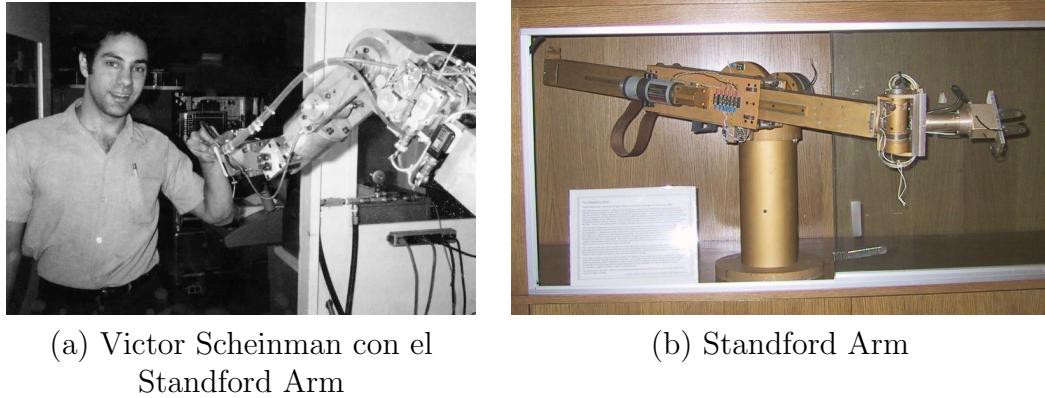
(b) Robot Unimate

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 [?].

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 [?].



(a) Victor Scheinman con el
Stanford Arm

(b) Stanford Arm

Figura 1.2: Stanford Arm

En 1973, KUKA¹ construyó el primer robot industrial con 6 ejes electromecánicos llamado Famulus. Un año más tarde, Cincinnati Milacron introdujo en el mercado el robot T3 (Figura 1.3). Cincinnati Milacron (adquirida por ABB² en 1990). El robot T3 fue el primer robot comercial controlado por un microordenador [?].

¹<https://www.kuka.com/es-es>

²<https://new.abb.com/products/robotics>



Figura 1.3: Robot Cincinnati Milacron T3

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 [?].

En 1978, Unimation diseñó y fabricó el robot PUMA. El PUMA (*Programmable Universal Machine for Assembly*) fue considerado durante muchas décadas el arquetipo de los robots antropomórficos [?]. Ese mismo año, el científico japonés Hiroshi Makino, de la Universidad de Yamanashi, propuso una nueva estructura cinemática. El robot con esta estructura se denominó SCARA (*Selective Compliance Assembly Robot Arm*) (ver Figura 1.4), ya que su conformidad en la dirección horizontal resultó menor que la conformidad en la dirección vertical. Por esta razón, así como por la ligereza de la cadena cinemática (que permitía un controlador más sencillo y rápido), este robot era adecuado para ser empleado en tareas como el ensamblaje de objetos pequeños [?].



Figura 1.4: Uno de los primeros prototipos de robot SCARA

Basado en este tipo de estructura, ABB desarrolló el Flex-Picker (Figura 1.5) en 1998, siendo este el robot de picking más rápido del mundo [?].



Figura 1.5: Robot ABB IRB 360 Flexpicker

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 [?].

Los requisitos de velocidad y peso de un robot han dado lugar a novedosos diseños cinemáticos y de transmisión. Desde el principio, la reducción de la masa y la inercia de las estructuras robóticas ha sido un objetivo primordial en el desarrollo de la robótica. El brazo humano, con una relación peso-carga de 1:1, se consideraba la referencia definitiva [?]. En el año 2004, con motivo de Automática, la mayor exposición de robots del mundo, se presentó por primera vez la combinación entre el robot ligero del DLR y la controladora KUKA, denominado RoboAssistant, donde se permitió a los visitantes mover y programar manualmente el robot, haciendo la visión de un robot que asiste a un trabajador durante los procesos de producción evidente para los visitantes [?].

Es en estos procesos de producción, donde la manipulación a dos manos puede ser crítica para tareas de ensamblaje complejas, manipulación simultánea y procesamiento de piezas de trabajo o para la manipulación de objetos de gran tamaño, por lo que en 2005, MOTOMAN presenta el primer robot comercial para la manipulación sincronizada a dos manos [?] (ver Figura 1.6).



Figura 1.6: Robot Motoman DA-20

Sin embargo, fue en el año 2006 cuando se toma la decisión de producir una primera pequeña serie del robot ligero del KUKA LWR3 [?], tal y como muestra la Figura 1.7.

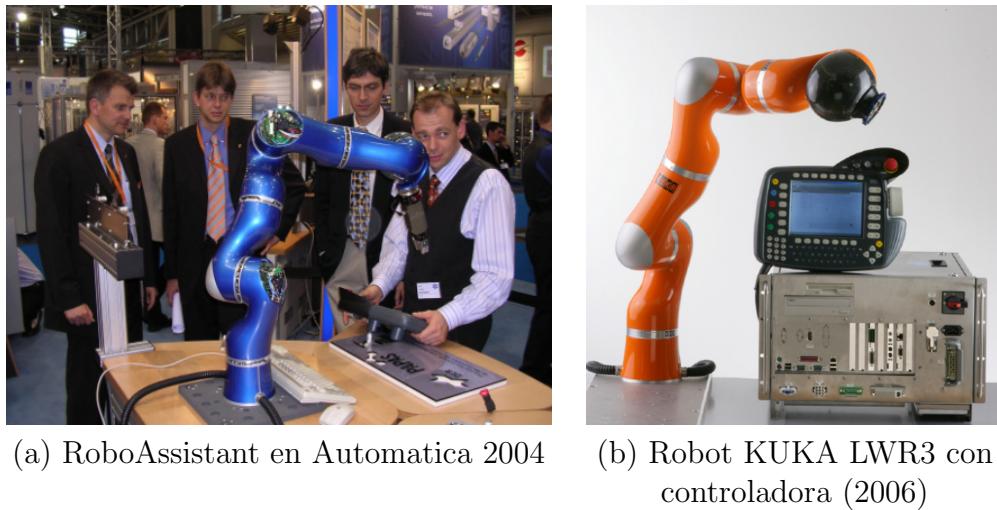


Figura 1.7: LWR3

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. Esto resultó en la incorporación de ROS en una sorprendentemente amplia variedad de robots, extendiéndose incluso a dominios más allá de la comunidad académica de investigación a la que se dirigió inicialmente. Los años siguientes superaron todas las expectativas debido a que los avances en el ámbito de la robótica se compartieron de manera reproducible en ROS, y la Open Source Robotics Foundation (OSRF) se convirtió en el administrador principal de ROS en 2014. Con el objetivo de atender de manera más efectiva las demandas de una comunidad ROS más extensa y abordar sus nuevos escenarios de aplicación, la OSRF se dedicó a desarrollar ROS2 como un conjunto de paquetes paralelos que pudieran ser instalados junto a ROS1 (la versión original de ROS que nació en el año 2010, Figura 1.8) y ser compatibles entre sí.

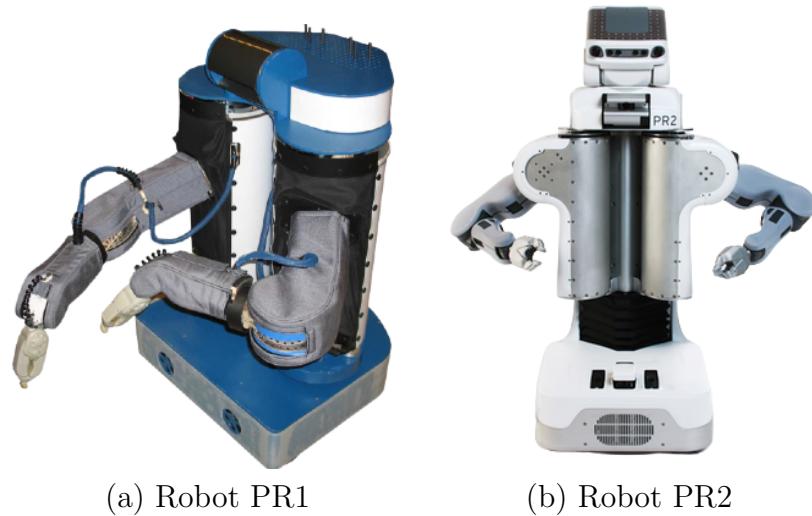


Figura 1.8: Robots utilizados para el desarrollo de ROS

En el año 2008 se entrega el primer robot colaborativo o cobot, el UR5 de Universal Robots³ (Figura 1.9), considerado como uno de los logros tecnológicos más significativos de la década en la comunidad robótica. El brazo robótico es pionero en la programación 3D fácil de usar pero sofisticada, con una interfaz de usuario intuitiva que permite a cualquier persona configurarlo y utilizarlo de forma rápida. Esta empresa, fundada en el año 2005 por Esben Østergaard, Kasper Støy y Kristian Kassow tras conocerse en la Universidad de Dinamarca, surgió con el objetivo de hacer que la robótica sea accesible para las pequeñas y medianas empresas⁴.

Esben H. Østergaard, Director de Tecnología y cofundador de Universal Robots, tomó el trabajo original de Peskin y Colgate, dos investigadores de la empresa automovilística Ford de los años 90, que decidieron crear un nuevo robot industrial, más pequeño y ágil que los tradicionales, que saliera de su jaula para colaborar estrechamente con el ser humano en las tareas de calidad y personalización de los productos, sin embargo, no fueron capaces, puesto que el problema estaba en la relación entre seguridad y rendimiento, ya que el aumento de la primera reducía el de la segunda. Østergaard consiguió diseñar un sistema de seguridad y control para el cobot que lo bloquea en caso de colisión con el operario, siendo capaz de operar en espacios confinados, en estrecho contacto con humanos, y sin instalar costosas barreras de seguridad [?].

³<https://www.universal-robots.com/es/>

⁴<https://www.universal-robots.com/es/acerca-de-universal-robots/nuestra-historia/>

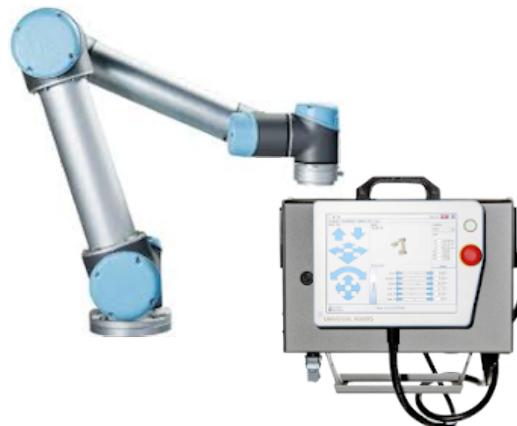
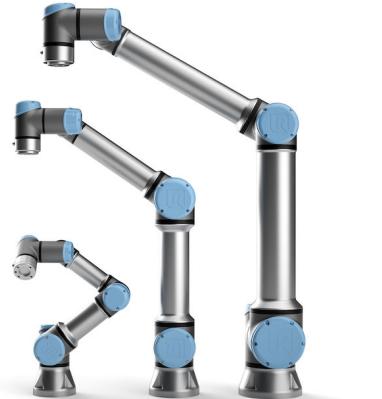


Figura 1.9: UR5 con su controladora

Más tarde, en 2018, Universal Robots presenta los robots colaborativos e-Series, que se pueden ver en la Figura 1.10, 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 recientes⁵.



(a) UR presenta los nuevos e-Series en Automatica 2018



(b) UR e-Series

Figura 1.10: Universal Robots e-Series

⁵<https://www.universal-robots.com/es/acerca-de-universal-robots/nuestra-historia/>

Esta cuarta generación de robótica industrial ha establecido un sólido punto de partida para una continua revolución en el campo de la automatización. Es esencial destacar que varios de los modelos de robots mencionados previamente han seguido evolucionando y mejorando con el tiempo, siendo fruto de estas mejoras, la comercialización de nuevos modelos y series. Debido a que la tecnología se encuentra en constante desarrollo y a la colaboración cada vez más estrecha entre humanos y robots, 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.

1.1.2. 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 [?].

En la práctica, las actuales y potenciales aplicaciones no industriales de los robots son tan variadas y diferentes que se dificulta su catalogación [?]; sin embargo, existen ciertas características especiales en estos robots de servicio que los hacen diferentes de los robots industriales [?], y los caracterizan para llevar a cabo estas tareas para las personas, siendo las principales características estos tres atributos de diseño: representación, antropomorfismo y orientación a la tarea, es decir, los robots de servicio pueden tener una representación física o tener una representación únicamente virtual , diseñarse como humanoides (es decir, antropomorfos) simulando una apariencia humana o como no humanoides , y pueden realizar tareas analíticas o tareas emocionales-sociales (por ejemplo, robots de recepción) [?].

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 [?], siendo las aplicaciones más importantes las siguientes:

- *Limpieza*: Suelen estar equipados con sensores y tecnología de navegación que les permite moverse de manera autónoma por el espacio, detectar obstáculos y llevar a cabo actividades de limpieza de manera eficiente.



Figura 1.11: Robot aspirador Roomba de iRobot

- *Inspección y mantenimiento*: Son máquinas diseñadas para llevar a cabo tareas de supervisión, evaluación y mantenimiento en entornos de infraestructura o áreas de difícil acceso. Estos robots suele ser máquinas autónomas o teleoperadas equipadas con sensores, cámaras y herramientas especializadas que les permiten evaluar, reparar y mantener equipos, estructuras y sistemas en entornos desafiantes o peligrosos.

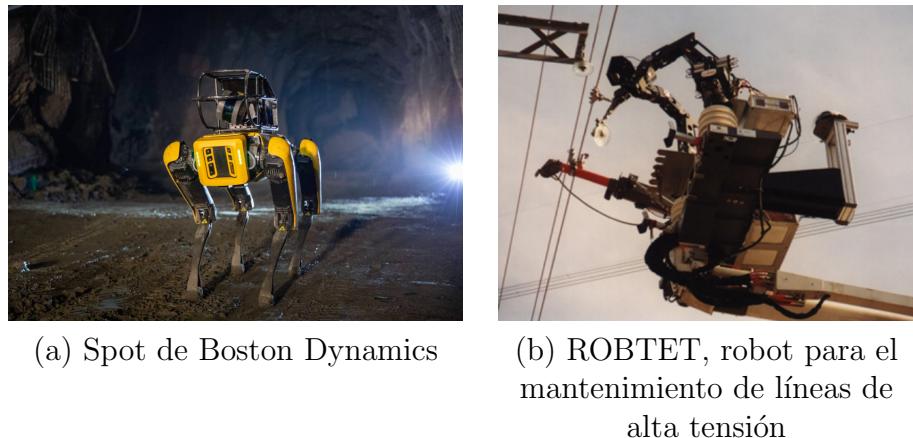
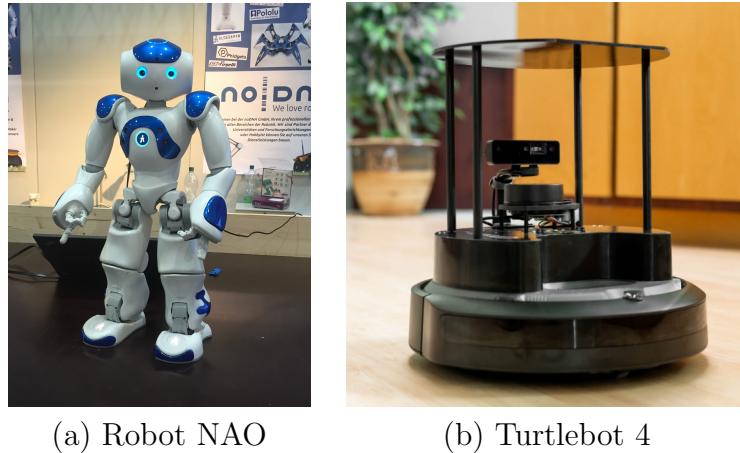


Figura 1.12: Robots de inspección y mantenimiento

- *Educación*: Son robots diseñados para facilitar el aprendizaje y la enseñanza en los diferentes niveles educativos, pudiendo ser utilizados en aulas, bibliotecas y entornos de aprendizaje para ayudar a los estudiantes a adquirir habilidades, fomentar la creatividad y brindar experiencias educativas interactivas.



(a) Robot NAO

(b) Turtlebot 4

Figura 1.13: Robots de educación

- *Logística:* Los robots de servicio utilizados en logística son robots diseñados para llevar a cabo tareas relacionadas con la gestión y el movimiento de mercancías y productos en entornos de almacenamiento, distribución y transporte. Estos robots 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 [?].



(a) AGV Robots Kiva de Amazon

(b) AMRs de MiR

Figura 1.14: Robots de logística

- *Entretenimiento:* Son robots diseñados específicamente para proporcionar experiencias lúdicas y de entretenimiento a las personas. Estos robots se utilizan en una variedad de contextos, siendo máquinas robóticas diseñadas para interactuar con el público.



(a) SONY Aibo



(b) Dron DJI Spark

Figura 1.15: Robots de entretenimiento

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 médicos

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 [?].

Los robots médicos 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 1.16), introducido en 1998-1999, que permitía aumentar la precisión de las cirugías mínimamente invasivas (CMI) [?].



Figura 1.16: Robot Da Vinci

Las primeras aplicaciones fueron en los campos de neurocirugía y cirugía ortopédica, siendo la cirugía donde mayor impacto han tenido los robots médicos, 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, robots de telepresencia para la interacción del paciente con el personal sanitario externo, como el robot RP-VITA, automatización de farmacias, robots para desinfectar clínicas, etc. [?]

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 [?], convirtiéndose en un campo interdisciplinario que abarca desde cirugía asistida por robots hasta sistemas de diagnóstico de vanguardia. Gran parte de su éxito radica en la integración de tecnologías avanzadas, como la inteligencia y la visión artificial. Estas disciplinas están redefiniendo la forma en que los robots médicos pueden interactuar con el entorno, interpretar datos y, en última instancia, mejorar los resultados en la atención médica.

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 [?].

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 una máquina abstracta como ya vimos en la sección 1.1, que sirvió de base de la noción de algoritmo y la definición de clase de problemas deducibles [?], 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, quizá 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 [?], mientras que el apelativo Inteligencia Artificial se debe a John McCarthy, quien organizó una conferencia en el Dartmouth 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 [?].

Dentro de las diversas formas de clasificar la IA, existe una clasificación, como se muestra en la Figura 1.17, que se basa en el objetivo y la forma en que trabaja el sistema: sistemas que piensan como humanos, sistemas que actúan como humanos, sistemas que piensan racionalmente, y sistemas actuantes racionales. Esta clasificación de manera inicial se veía como clases independientes, sin embargo, en la actualidad los sistemas mezclan características de ellas. [?]



Figura 1.17: Modelos de inteligencia

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 vídeos.

1.3. Visión Artificial

La visión artificial se define como la ciencia de programar un ordenador para procesar imágenes o vídeos e incluso entenderlos [?].

En [?] se explica cómo es la transformación de datos desde un fotograma o vídeo cámara hasta lo que puede ser una decisión o una nueva representación [?]. Para ello, la imagen percibida pasa por los procesos de obtención, caracterización e interpretación de información de imágenes; y estos procesos pueden ser subdivididos a su vez en [?] según el Cuadro 1.1.

Procesos	Nivel de Visión	Entrada	Salida	Área
1. Captura 2. Pre-procesamiento	Bajo	Imagen	Imagen	Procesamiento de imágenes
3. Segmentación	Medio	Imagen	Grupo de píxeles en bruto (objetos o regiones)	Análisis de Imágenes
4. Descripción		Objetos o regiones	Información cuantitativa de los objetos o regiones	
5. Reconocimiento (clasificación)		Información cuantitativa	Objetos clasificados en categorías	
6. Interpretación	Alto	Objetos clasificados en categorías	Compresión de la escena	Visión por Computador

Cuadro 1.1: Procesos de la visión artificial

1. *Captura*: Es el proceso en el que se obtiene una imagen digital a partir de una imagen analógica a través de un dispositivo para que pueda ser manipulada por un ordenador. Esta imagen estará representada como una matriz de números (píxeles) [?].
2. *Pre-procesamiento*: En esta fase, se incorporan métodos destinados a restaurar las imágenes capturadas. Esta etapa tiene como objetivo corregir estos problemas mediante procedimientos como la eliminación de ruido o la mejora del contraste y la nitidez.
3. *Segmentación*: Consiste en dividir una imagen en regiones o componentes más pequeños (grupo de píxeles) con el objetivo de identificar y aislar objetos o áreas de interés dentro de la imagen para que sea más fácil de analizar, comprender y procesar por ordenador.
4. *Descripción*: Es el proceso que obtiene características relevantes para poder diferenciar un tipo de objeto de otro, pudiendo ser externas, como la forma, el perímetro o el rectángulo mínimo que contiene la región; o internas, como el área o el centro de gravedad, entre otros [?].
5. *Reconocimiento (clasificación)*: El proceso de reconocimiento implica el uso de algoritmos y técnicas de aprendizaje automático, como redes neuronales artificiales o métodos estadísticos, entre otros, para entrenar un modelo que pueda tomar las características extraídas y realizar predicciones sobre la clase o categoría a la que pertenecen los objetos detectados.

6. *Interpretación:* Esta etapa implica razonamiento, toma de decisiones y puede requerir el procesamiento de lenguaje natural para obtener una comprensión más profunda del contenido visual.

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 [?]. 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 [?].

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 1.18 [?].



Figura 1.18: 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 1.19, 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 [?].

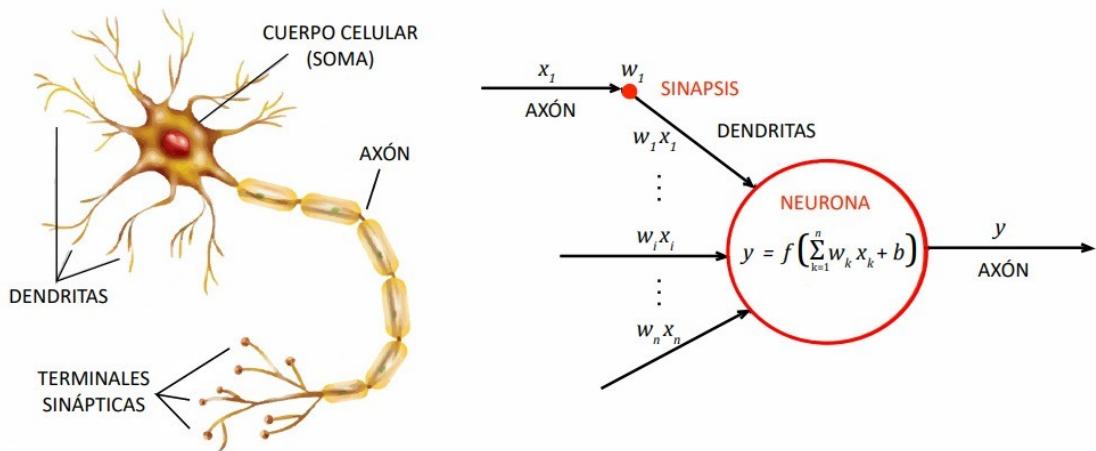


Figura 1.19: Modelo biológico de una neurona genérica (izquierda) y el respectivo modelo matemático (derecha)

En consecuencia, se puede construir una red neuronal artificial mediante un conjunto de neuronas artificiales, es decir, mediante un conjunto de funciones, y conectando comúnmente la salida de cada una a las entradas de otras diferentes, como se representa en la Figura 1.20. Es importante señalar que la característica clave de la sinapsis, el escalar las señales de entrada por factores (pesos), es la manera en la que se cree que el cerebro aprende. Por lo tanto, 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 [?].

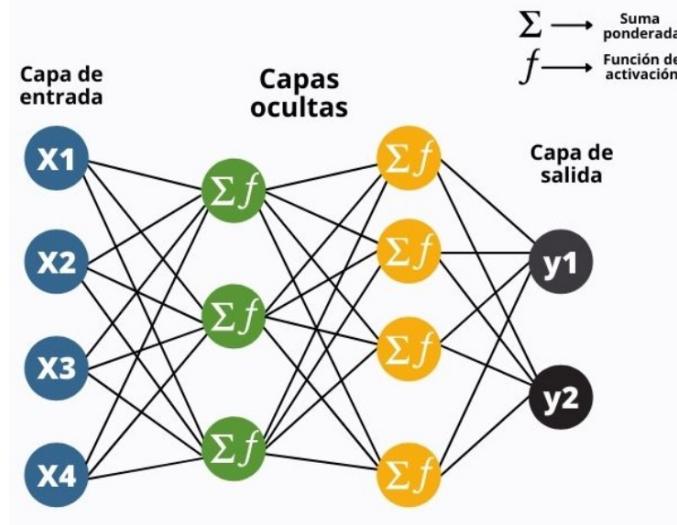


Figura 1.20: Arquitectura de una red neuronal

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, siendo este subcategoría del Machine 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 de 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 [?], 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) [?], 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 [?].

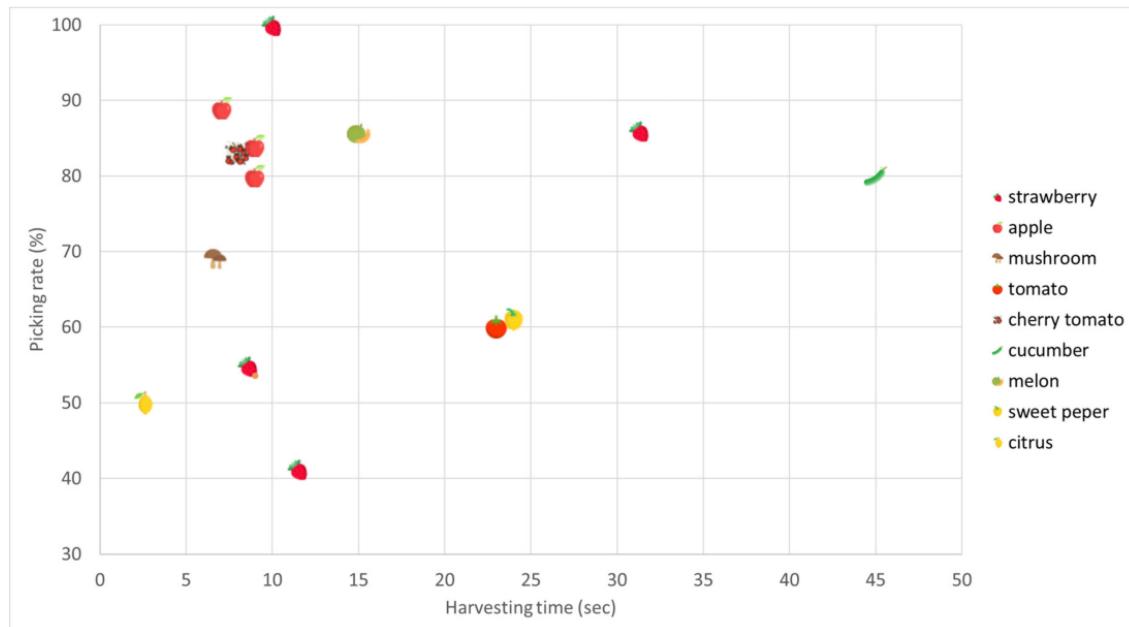
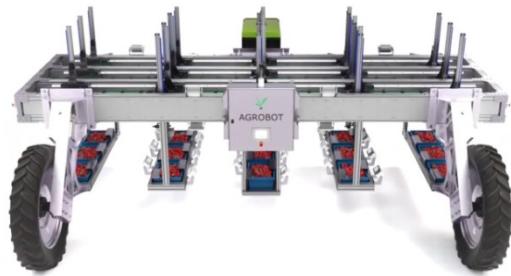


Figura 2.2: Ilustración global del rendimiento general de los robots revisados

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 [?]. 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 [?].

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 [?].

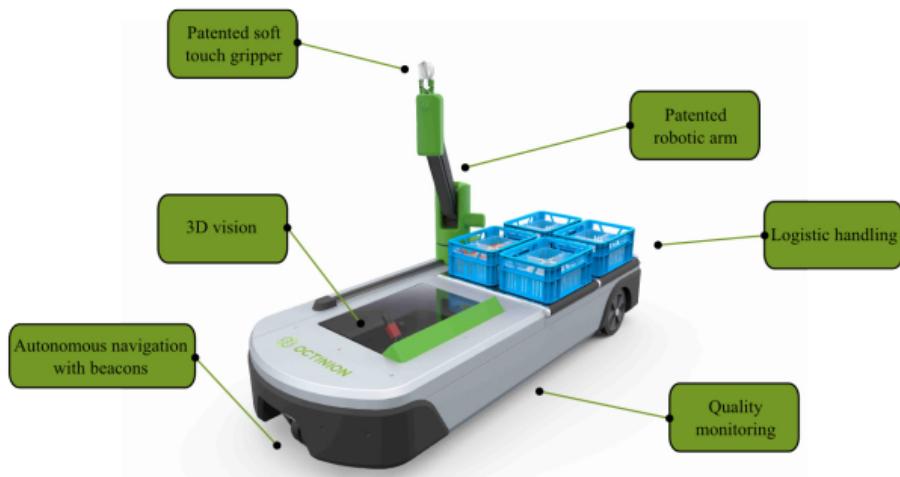


Figura 2.4: Diseño conceptual del robot de recogida con sus componentes

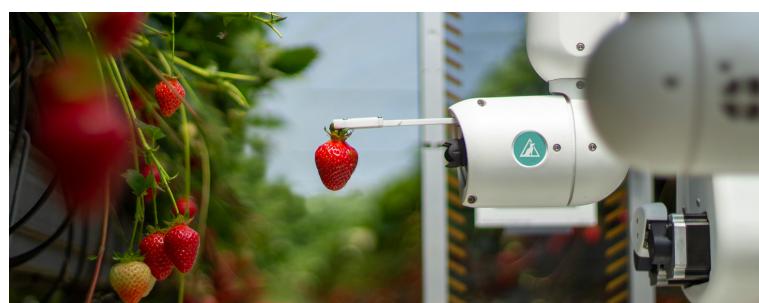
Todo esto le permite al robot recoger al menos el 70 % de las fresas maduras, siempre sin dañarlas, ya que sólo decide recoger la fruta si su acción no va a dañar otras fresas, siendo el tiempo necesario para desplazarse hasta la fresa, cogerla y depositarla en una cesta (caja en la que se colocan las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

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 [?], 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.

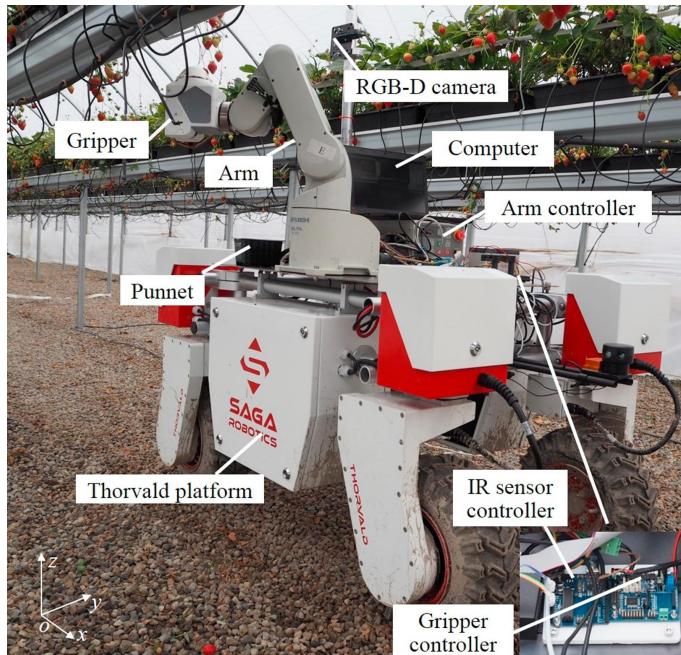


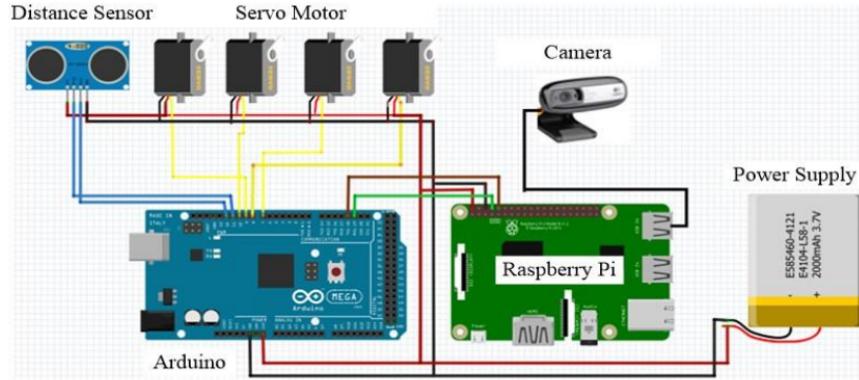
Figura 2.6: Montaje del hardware en una explotación de fresas

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

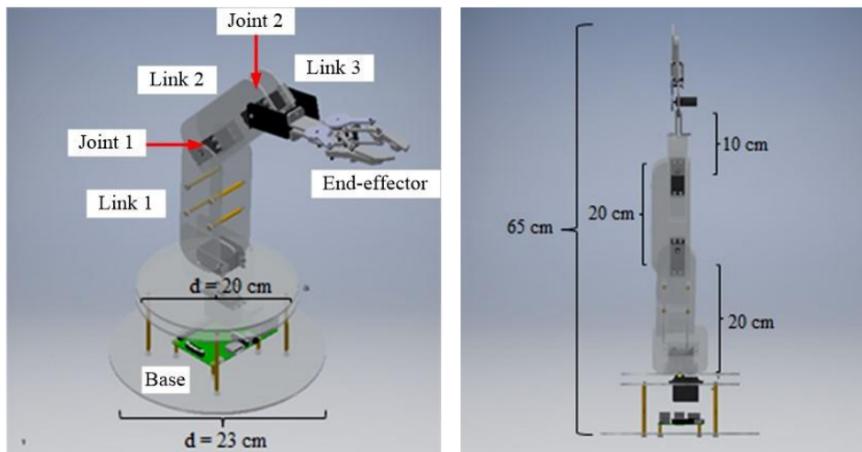
- Oclusión de las fresas, lo que deriva en una detección fallida y una no recolección de las mismas.
- Posibles detecciones duplicadas, debido a las agrupaciones de fresas que se tocan entre sí.
- Errores de localización para la que la pinza pueda coger la fresa, producidos por localizaciones imprecisas 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 [?], 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. El objetivo de este estudio es establecer el proyecto inicial en la creación de una serie de robots aplicados en la agricultura para hacer realidad la idea de la agricultura digital. La novedad de este estudio es que este método es simple, al igual que el procesamiento de imágenes, para adaptarse a los recursos limitados de los procesadores con los que se llevó a cabo este proyecto. El robot diseñado se personaliza en función del tamaño de la tomatera y el color. El robot, representado en la Figura 2.7, consta de una cámara web común instalada en el efecto final y un sensor ultrasónico de proximidad HC-SR04, siendo la cámara quien captura la imagen en bruto de la fruta, mientras que el sensor de proximidad detecta la distancia entre el efecto final, que consiste en una tijera para cortar la rama de tomates, y la fruta objetivo. El resultado del procesamiento de imágenes (los tomates detectados) se envían a la Raspberry Pi 3 Modelo B, donde estos dan el número binario «1» correspondiente a los tomates detectados, mientras que al resto de la imagen, convertida en escala de grises, se considera «0», y se envían al microcontrolador los ángulos de los servomotores que mueven el brazo robótico de 4 grados de libertad, para acercarse a los tomates.



(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.

En este contexto, en el siguiente capítulo se presentan los objetivos, la metodología y el plan de trabajo orientado a desarrollar este trabajo, y que se basa en un sistema de visión artificial capaz de detectar con precisión el estado de maduración de las fresas y determinar su posición en el espacio, facilitando así su recolección mediante un brazo robótico.

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, y de la posibilidad de que existan errores humanos en la identificación de los frutos para su recolección, pudiendo influenciar esto en la calidad del producto, especialmente en la recolección de frutos que requieren un manejo cuidadoso, como las fresas.

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. 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.

3.3. 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 GitHub¹³ dedicada al proyecto, que refleja los avances y la organización 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 las pruebas prácticas relacionadas con estas áreas del trabajo suponían.
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:*

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

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 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 el cual 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 reflejada 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.

3.4. 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 3.1).

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.

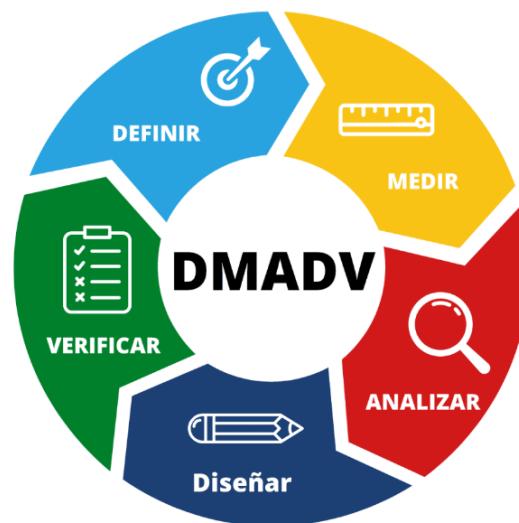


Figura 3.1: Ciclo de la metodología DMADV

3.5. 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 alcanzase 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 GitHub¹⁴, en cuya Wiki¹⁵ se puede ver el desarrollo del trabajo en semanas a lo largo de los meses, durante el trascurso del proyecto.

Después de haber revisado los objetivos, requisitos, competencias, metodología y el plan de trabajo implementado para la realización de este proyecto, en el siguiente capítulo se abordarán las plataformas de desarrollo empleadas.

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

¹⁵<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), 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: Cámara Logitech C270 HD¹⁶

¹⁶<https://www.logitech.com/es-es/products/webcams/c270-hd-webcam.960-001063.html?srsltid=AfmB0or4HptUTcGrxE-4SzxKR-ARw-ykNeagHSEzXUvT1Kx8qLfY41G>

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 4.2), 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€.



Figura 4.2: Soporte de brazo articulado¹⁷

4.1.3. Ordenador principal

El equipo que se ha configurado como entorno de trabajo para este proyecto es un Lenovo Legion 5 IMH05¹⁸ 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.

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

¹⁸<https://www.pcccomponentes.com/lenovo-legion-5-15imh05-intel-core-i7-10750h-16gb-1tb-ssd-gtx-1650-156?srsltid=AfmBOopJpvGSHUyQU696jgG7-6orSKMOEWZe2ZvvYtA0NGtJ9Ms2xJFp>

4.1.4. 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 [?].

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 robótico¹⁹.

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 [?] hasta el UR10e [?], pasando por el UR5e [?]. 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 [?].

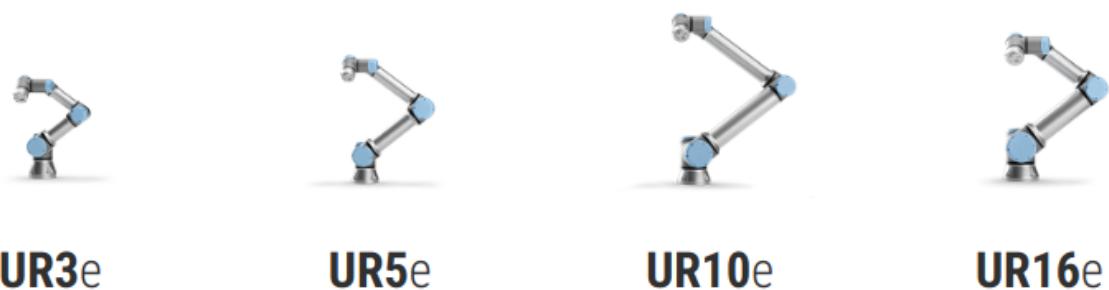


Figura 4.3: Gama e-series de Universal Robots²⁰

¹⁹<https://www.universal-robots.com/mx/acerca-de-universal-robots/noticias/meet-the-next-generation-of-collaborative-ur-robots-at-robobusiness>

4.1.5. 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 LS105G, 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, ya que se trata de un dispositivo *plug and play*, por lo que simplemente se tiene que conectar y empieza a funcionar.

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

Ubuntu²² 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.

²⁰<https://www.universal-robots.com/es/productos/>

²¹<https://www.tp-link.com/es/business-networking/litewave-switch/ls105g/>

²²<https://ubuntu.com/>

4.2.2. Polyscope

Polyscope²³ 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.

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 5²⁴ (Figura 4.4) se introdujo en junio de 2018, coincidiendo con el lanzamiento de la gama de robots e-series. Por último, PolyScope X²⁵ es la última evolución del software de Universal Robots, siendo su lanzamiento oficial en noviembre de 2024²⁶, 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.

²³<https://www.universal-robots.com/es/productos/polyscope/>

²⁴<https://www.universal-robots.com/products/polyscope-5/>

²⁵<https://www.universal-robots.com/products/polyscope-x/>

²⁶<https://www.universal-robots.com/2024q3/polyscope-x-festival/>

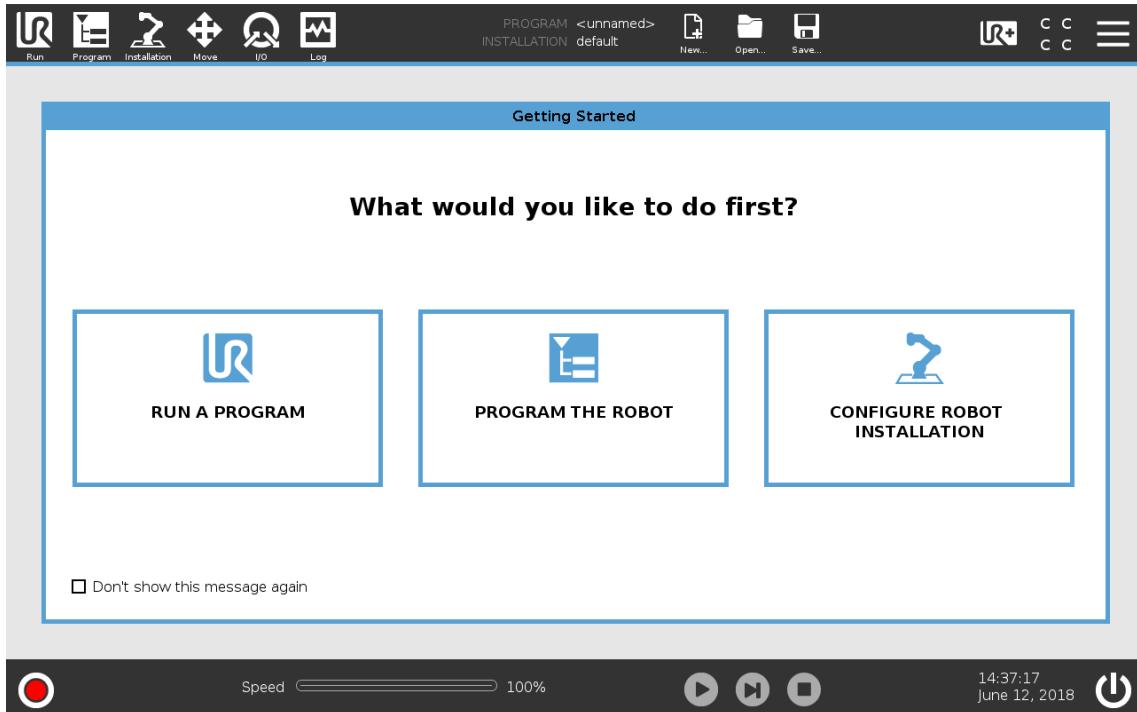


Figura 4.4: Pantalla principal de la interfaz de Polyscope 5

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.

4.2.3. Python

Python²⁷ 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 código²⁸ para el desarrollo de aplicaciones en diferentes áreas, como sucede en este caso con la inteligencia y visión artificial y el aprendizaje automático.

²⁷<https://www.python.org/>

²⁸<https://www.python.org/doc/essays/blurb/>

4.2.4. PyTorch

De los desarrolladores de Facebook AI Research, junto a otros laboratorios, PyTorch²⁹ 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.

4.2.5. NumPy

NumPy (Numerical Python)³⁰ 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 matrices³¹.

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.

4.2.6. OpenCV

OpenCV (Open Source Computer Vision Library)³² es una biblioteca de software de código abierto diseñada para su uso en aplicaciones de aprendizaje automático y visión artificial. Desarrollado por Intel³³ en 1999, cuenta con más de 2.500 algoritmos optimizados, que incluyen un amplio conjunto de algoritmos de visión por ordenador y aprendizaje automático.

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

²⁹<https://pytorch.org/>

³⁰<https://numpy.org/>

³¹<https://numpy.org/doc/stable/user/whatisnumpy.html/>

³²<https://opencv.org/>

³³<https://www.intel.es>

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.

4.2.7. OpenGL

OpenGL (Open Graphics Library)³⁴ 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.

4.2.8. SocketTest

SocketTest³⁵ 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.

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.

³⁴<https://opengl.org/>

³⁵<https://sockettest.sourceforge.net/>

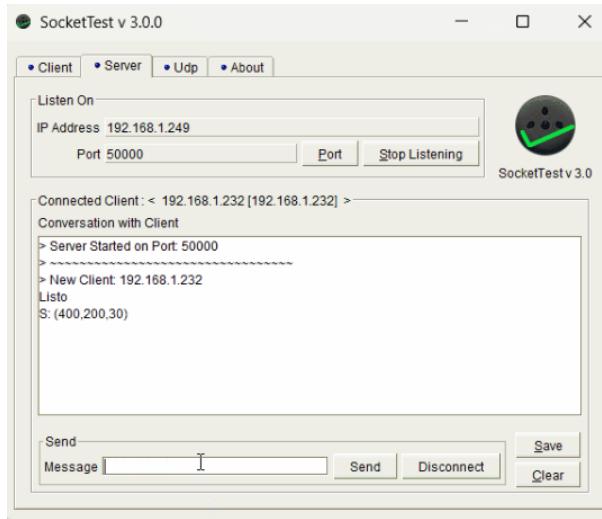


Figura 4.5: Pruebas realizadas con SocketTest para verificar la comunicación robot-servidor externo

4.2.9. XML-RPC

XML-RPC³⁶ 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 *xmlrpclib*, 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.

4.2.10. Anaconda

Anaconda³⁷ 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

³⁶<https://docs.python.org/es/3.8/library/xmlrpc.html>

³⁷<https://www.anaconda.com/>

la línea de comandos³⁸.

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 otro³⁹.

4.2.11. YOLOv3

YOLOv3 (You Only Look Once versión 3)⁴⁰ es un algoritmo de detección de objetos en tiempo real que identifica y localiza múltiples objetos dentro de una imagen o video. 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 [?]. 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 aplicaciones⁴¹.

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 definidos⁴²; 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.

³⁸<https://docs.anaconda.com/anaconda/>

³⁹<https://docs.anaconda.com/reference/glossary/#conda>

⁴⁰<https://docs.ultralytics.com/es/models/yolov3/>

⁴¹<https://docs.ultralytics.com/es/models/yolov3/#supported-tasks-and-modes>

⁴²<https://github.com/ultralytics/yolov3/>

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 los capítulos siguientes.

Capítulo 5

Sistema de reconocimiento por visión de maduración de frutos para su recolección con un brazo robótico

Quizás algún fragmento de libro inspirador...

Autor, Título

Escribe aquí un párrafo explicando brevemente lo que vas a contar en este capítulo. En este capítulo (y quizás alguno más) es donde, por fin, describes detalladamente qué has hecho y qué experimentos has llevado a cabo para validar tus desarrollos.

5.1. Snippets

Puede resultar interesante, para clarificar la descripción, mostrar fragmentos de código (o *snippets*) ilustrativos. En el Código 5.1 vemos un ejemplo escrito en C++.

```
void Memory::hypothesizeParallelograms () {
    for(it1 = this->controller->segmentMemory.begin(); it1++) {
        squareFound = false; it2 = it1; it2++;
        while ((it2 != this->controller->segmentMemory.end()) && (!squareFound))
        {
            if (geometry::haveACommonVertex((*it1), (*it2), &square)) {
                dist1 = geometry::distanceBetweenPoints3D ((*it1).start, (*it1).end);
                dist2 = geometry::distanceBetweenPoints3D ((*it2).start, (*it2).end);
            }
        // [...]
```

Código 5.1: Función para buscar elementos 3D en la imagen

En el Código 5.2 vemos un ejemplo escrito en Python.

```

def mostrarValores():
    print (w1.get(), w2.get())

master = Tk()
w1 = Scale(master, from_=0, to=42)
w1.pack()
w2 = Scale(master, from_=0, to=200, orient=HORIZONTAL)
w2.pack()
Button(master, text='Show', command=mostrarValores).pack()

mainloop()

```

Código 5.2: Cómo usar un Slider

5.2. Verbatim

Para mencionar identificadores usados en el código —como nombres de funciones o variables— en el texto, usa el entorno literal o verbatim `hypothesizeParallelograms()`. También se puede usar este entorno para varias líneas, como se ve a continuación:

```

void Memory::hypothesizeParallelograms () {
    // add your code here
}

```

5.3. Ecuaciones

Si necesitas insertar alguna ecuación, puedes hacerlo. Al igual que las figuras, no te olvides de referenciarlas. A continuación se exponen algunas ecuaciones de ejemplo: Ecuación 5.1 y Ecuación 5.1.

$$H = 1 - \frac{\sum_{i=0}^N \left(\frac{d_{js} + d_{je}}{2} \right)}{M}$$

Ecuación 5.1: Ejemplo de ecuación con fracciones

$$v(\text{entrada}) = \begin{cases} 0 & \text{if } \epsilon_t < 0,1 \\ K_p \cdot (T_t - T) & \text{if } 0,1 \leq \epsilon_t < M_t \\ K_p \cdot M_t & \text{if } M_t < \epsilon_t \end{cases} \quad (5.1)$$

Ecuación 5.2: Ejemplo de ecuación con array y letras y símbolos especiales

5.4. Tablas o cuadros

Si necesitas insertar una tabla, hazlo dignamente usando las propias tablas de L^AT_EX, no usando pantallazos e insertándolas como figuras... En el Cuadro 5.1 vemos un ejemplo.

Parámetros	Valores
Tipo de sensor	Sony IMX219PQ[7] CMOS 8-Mpx
Tamaño del sensor	3.674 x 2.760 mm (1/4"format)
Número de pixels	3280 x 2464 (active pixels)
Tamaño de pixel	1.12 x 1.12 um
Lente	f=3.04 mm, f/2.0
Ángulo de visión	62.2 x 48.8 degrees
Lente SLR equivalente	29 mm

Cuadro 5.1: Parámetros intrínsecos de la cámara

Capítulo 6

Experimentos

En este capítulo 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 maduración de frutos y su comunicación con el brazo robótico, permitiendo así alcanzar los objetivos definidos en fases anteriores del trabajo.

6.1. 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, y se iniciaron las pruebas pertinentes para la selección del algoritmo de detección y de las bibliotecas a utilizar.

6.1.1. 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 basados en la configuración *faster rcnn resnet101 coco* de los modelos de detección de objetos de Tensorflow para poder llevar a cabo la comparación, y etiquetando imágenes, en este caso de tigres, mediante la herramienta *labelImg*², 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 utilizando una distribución de las imágenes utilizadas para el aprendizaje del modelo y su uso en la detecciónn aproximadamente del 70:30 (73 % datos de entrenamiento y 27 % datos de prueba)(Cuadro 6.1), 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

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

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

datos.

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

Cuadro 6.1: Distribución de las imágenes utilizadas para el entrenamiento del modelo

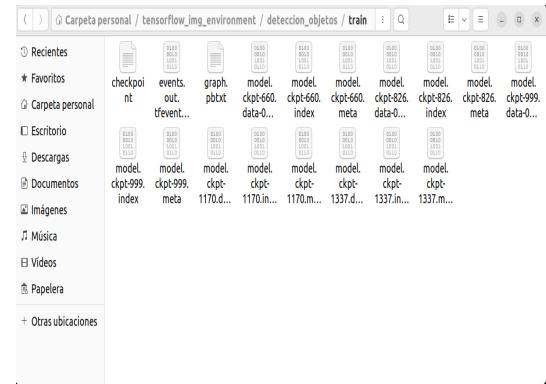
Se entrenó este modelo hasta que se observó que la pérdida estaba por debajo de 1, considerando que esta pérdida no era alta, y que no existían demasiadas fluctuaciones, deteniendo este entrenamiento a los 1400 pasos, a pesar de que este entrenamiento estuviera programado para llegar hasta los 20000, ya que se trataba de una prueba simplemente. Esto supuso que se tuviera que utilizar el último checkpoint disponible, en este caso el del paso 1337 (Figura 6.1), para convertirlo en un modelo final y de esta manera poder generar predicciones, utilizando imágenes de diferentes tamaños.

```

INFO:tensorflow:Recording summary at step 1369.
INFO:tensorflow:global step 1370: loss = 0.3654 (4.243 sec/step)
INFO:tensorflow:global step 1371: loss = 0.6998 (2.566 sec/step)
INFO:tensorflow:global step 1372: loss = 0.4905 (2.495 sec/step)
INFO:tensorflow:global step 1373: loss = 0.2510 (2.445 sec/step)
INFO:tensorflow:global step 1374: loss = 0.2621 (2.534 sec/step)
INFO:tensorflow:global step 1375: loss = 0.2305 (2.504 sec/step)
INFO:tensorflow:global step 1376: loss = 0.1974 (2.093 sec/step)
INFO:tensorflow:global step 1377: loss = 0.0982 (2.442 sec/step)
INFO:tensorflow:global step 1378: loss = 0.1205 (2.168 sec/step)
INFO:tensorflow:global step 1379: loss = 0.0652 (6.610 sec/step)
INFO:tensorflow:global step 1380: loss = 0.1447 (2.004 sec/step)
INFO:tensorflow:global step 1381: loss = 0.2082 (2.731 sec/step)
INFO:tensorflow:global step 1382: loss = 0.3774 (3.576 sec/step)
INFO:tensorflow:global step 1383: loss = 0.1702 (8.328 sec/step)
INFO:tensorflow:global step 1384: loss = 0.1468 (10.973 sec/step)
INFO:tensorflow:global step 1385: loss = 0.1500 (8.590 sec/step)
INFO:tensorflow:global step 1386: loss = 0.5760 (8.619 sec/step)
INFO:tensorflow:global step 1387: loss = 0.0436 (2.195 sec/step)
INFO:tensorflow:global step 1388: loss = 1.3603 (2.574 sec/step)
INFO:tensorflow:global step 1389: loss = 0.1874 (2.546 sec/step)
INFO:tensorflow:global step 1390: loss = 0.2388 (2.462 sec/step)
INFO:tensorflow:global step 1391: loss = 0.1824 (2.217 sec/step)
INFO:tensorflow:global step 1392: loss = 0.4054 (6.584 sec/step)
INFO:tensorflow:global step 1393: loss = 0.2781 (2.549 sec/step)
INFO:tensorflow:global step 1394: loss = 0.1960 (2.458 sec/step)
INFO:tensorflow:global step 1395: loss = 0.5281 (2.579 sec/step)
INFO:tensorflow:global step 1396: loss = 0.0996 (2.433 sec/step)
INFO:tensorflow:global step 1397: loss = 0.5380 (2.562 sec/step)
INFO:tensorflow:global step 1398: loss = 0.1804 (2.570 sec/step)
INFO:tensorflow:global step 1399: loss = 0.1801 (2.570 sec/step)
INFO:tensorflow:global step 1400: loss = 0.9420 (2.661 sec/step)

```

(a) Pasos finales del entrenamiento con TensorFlow



(b) Checkpoints del entrenamiento con TensorFlow

Figura 6.1: Entrenamiento del algoritmo con TensorFlow

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 para este caso, y evaluar visualmente los resultados obtenidos en algunos ejemplos mostrados en la Figura 6.2, de estas primeras detecciones realizadas tras el entrenamiento.

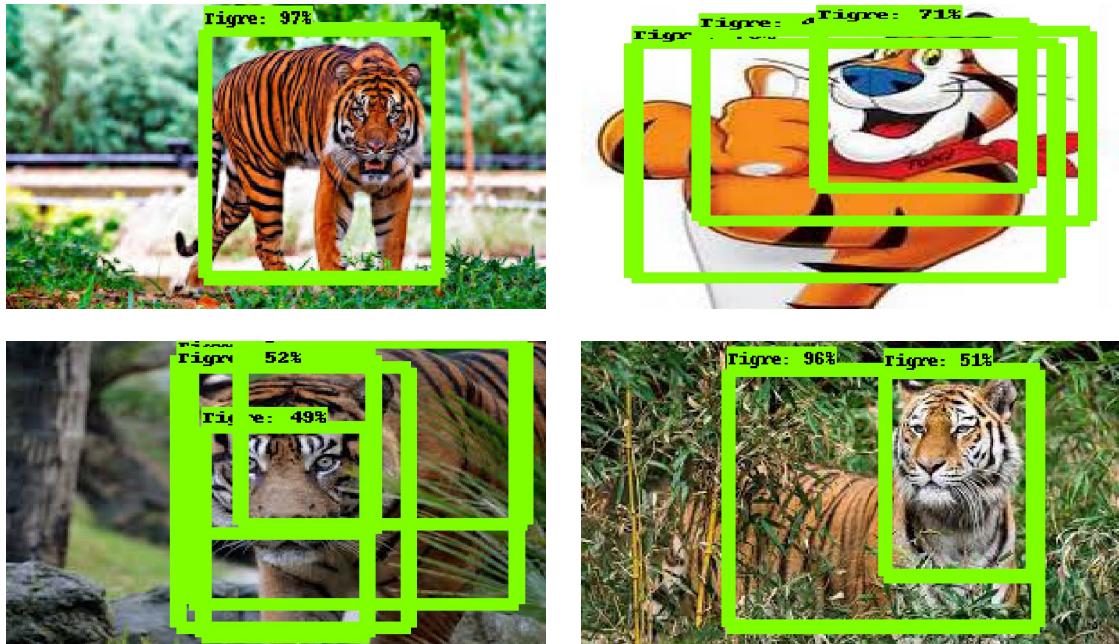


Figura 6.2: Resultado de la detección en imágenes con TensorFlow

Dados 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 este último checkpoint generado fue 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 6.3).

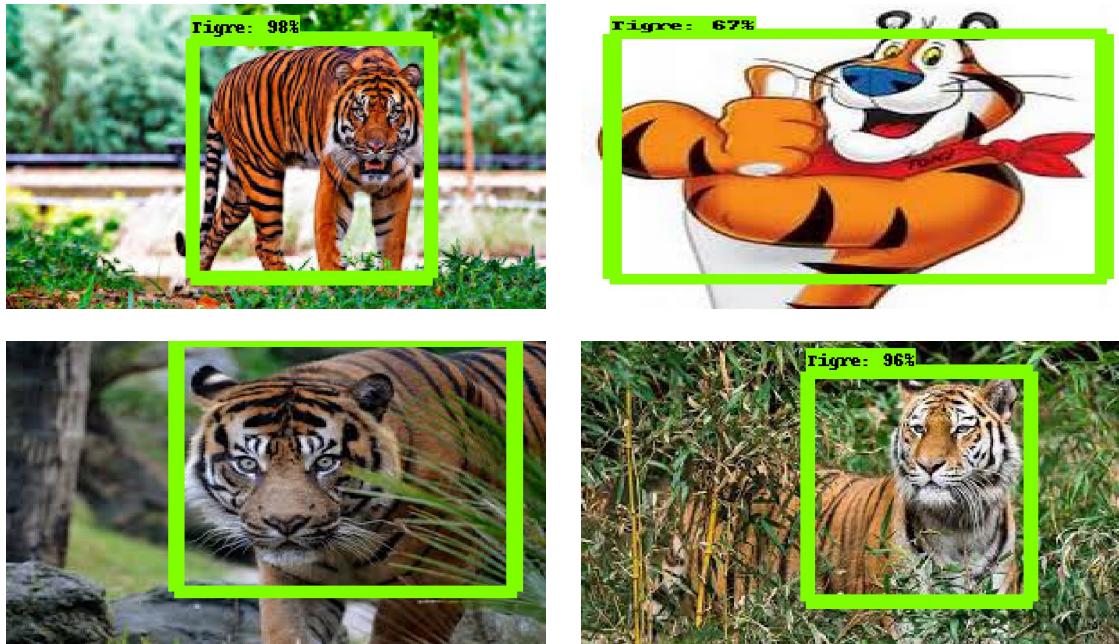


Figura 6.3: Resultado del reentrenamiento de la detección en imágenes con TensorFlow

Después de llevar a cabo estas pruebas con el ejemplo de los tigres, se comprobó si el modelo funcionaría también con el objeto final, en este caso, 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 para llevar a cabo estas pruebas.

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 6.4.

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

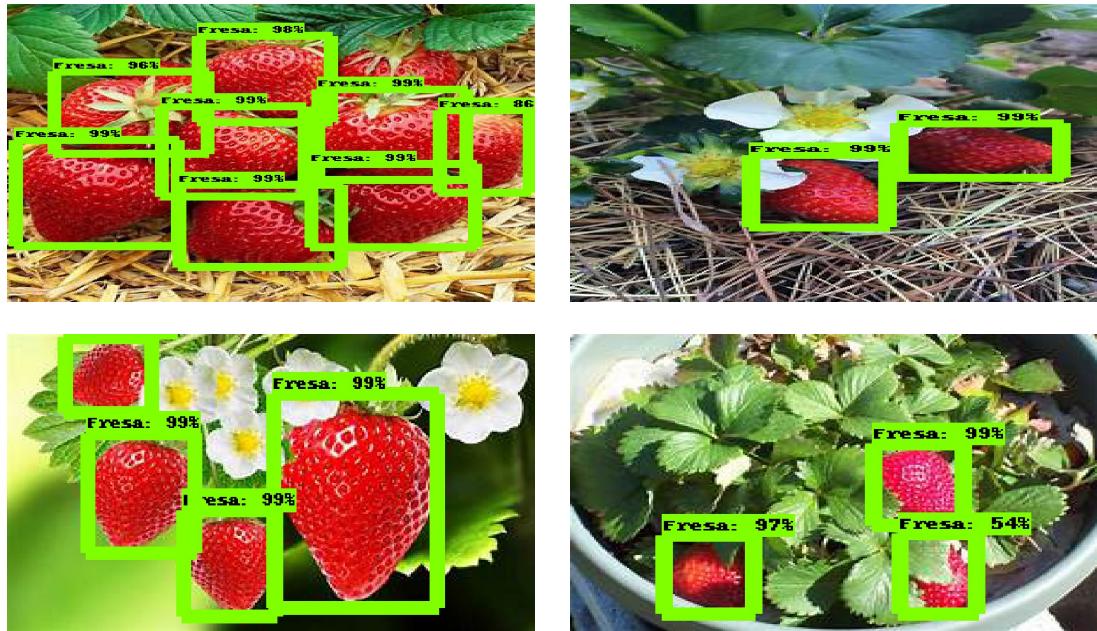
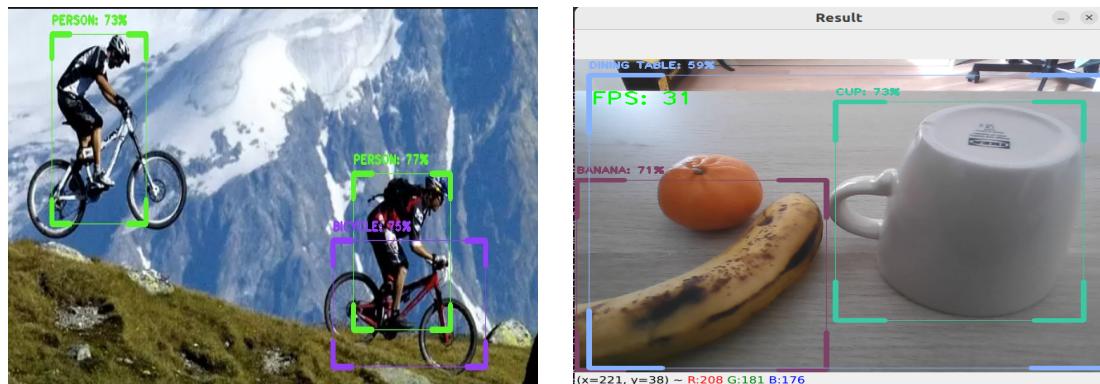


Figura 6.4: Pruebas de detección de fresas en imágenes con TensorFlow

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, de forma análoga a como se había realizado previamente con la librería PyTorch, se procedió a evaluar distintos modelos de detección de objetos pertenecientes al zoo de TensorFlow (TensorFlow 2 Detection Model Zoo), los cuales, al estar ya preentrenados sobre conjuntos de datos de referencia, permitieron 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.

6.1.2. Pruebas con vídeo en tiempo real

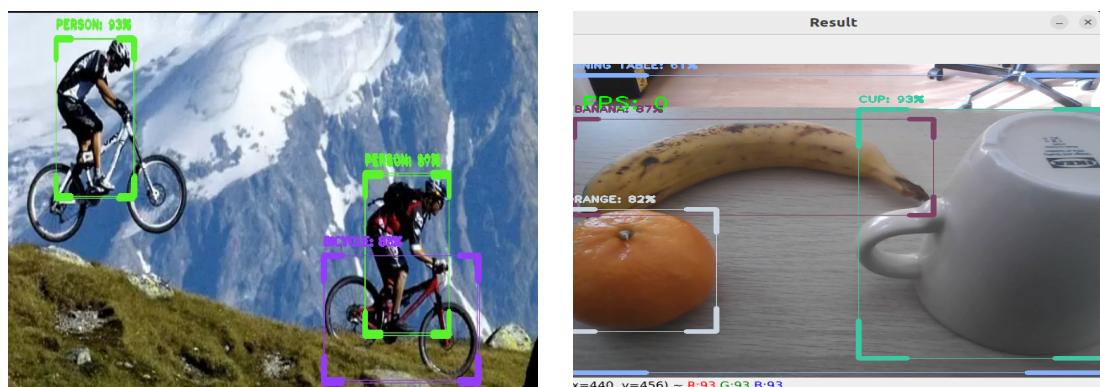
Para la realización de estas pruebas, se utilizó tanto la cámara web integrada del ordenador portátil como una imagen previamente seleccionada, para que, de esta manera pudieran observarse las diferencias entre los modelos tanto en la detección en vídeo como en la detección en imágenes, y poder valorar 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 6.5, 6.6 y 6.7).



(a) Resultado del modelo en imagen

(b) Resultado del modelo en vídeo

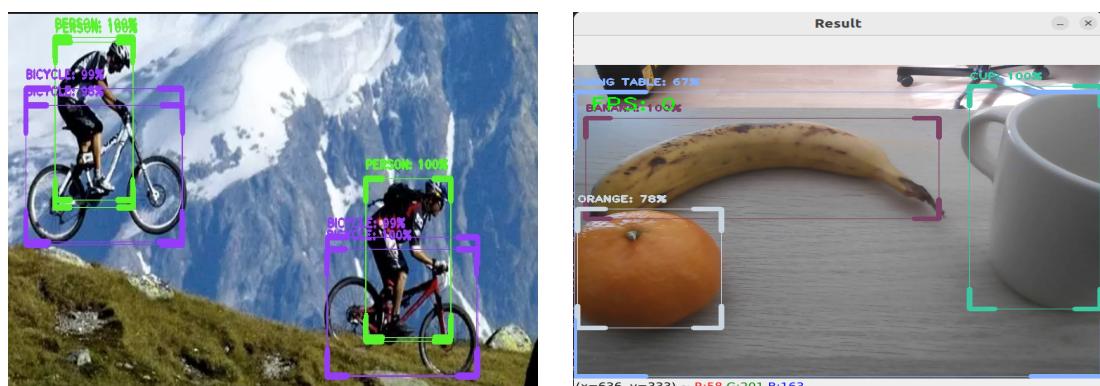
Figura 6.5: Modelo ssd_mobilenet_v2_320x320_coco17_tpu-8



(a) Resultado del modelo en imagen

(b) Resultado del modelo en vídeo

Figura 6.6: Modelo efficientdet_d4_coco17_tpu-32



(a) Resultado del modelo en imagen

(b) Resultado del modelo en vídeo

Figura 6.7: 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 a tiempo real con una cámara, se escogió el modelo `ssd_mobilenet_v2` para proseguir con los experimentos, incluso por delante de cualquiera de los otros dos modelos, ya que, a pesar de tener menor precisión y calidad de detección, 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 puede ser un sistema de visión embarcado en un brazo robótico.

Para ello, y dado que para poder llevar a cabo la detección de fresas era necesario utilizar un modelo entrenado 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, donde se instalaron los paquetes y librerías necesarios para ello junto al Object Detection API de TensorFlow junto con Jupyter Notebook⁵, un entorno computacional interactivo basado en web para crear cuadernos que contienen código en vivo, ecuaciones, visualizaciones y texto narrativo.

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, y de este modo, poder llevar a cabo la batería de pruebas en las cuales se variaba tanto el número de fresas como las condiciones de luz, siendo los resultados de estos escenarios los que se muestran a continuación en la Figura 6.8:

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

⁵<https://jupyter.org>

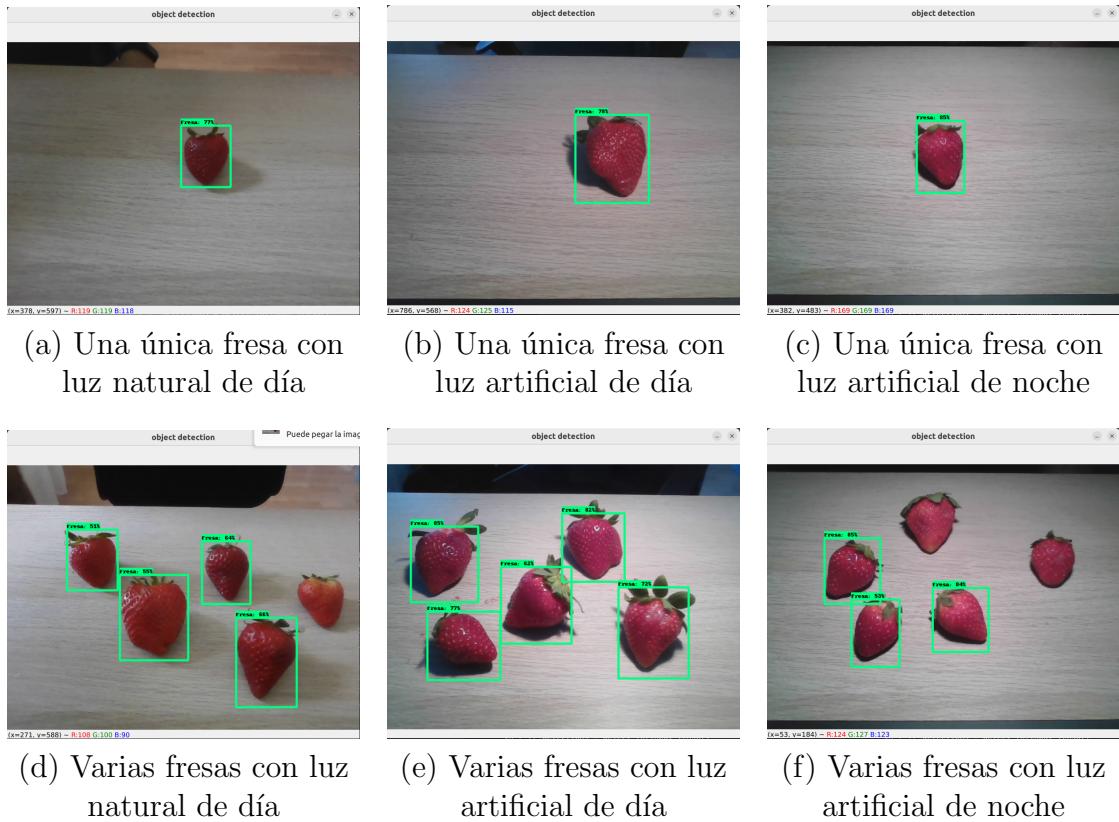


Figura 6.8: Detección de fresas en webcam con TensorFlow con modelos no pre-entrenados (ssd mobilenet v2 320x320)

Después de verificar la viabilidad y funcionamiento de estas pruebas, y de detectar en los resultados de estas 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 6.9 y 6.10, se modificó el programa de detección para obtener más datos sobre estas detecciones y para poder mejorar el programa.

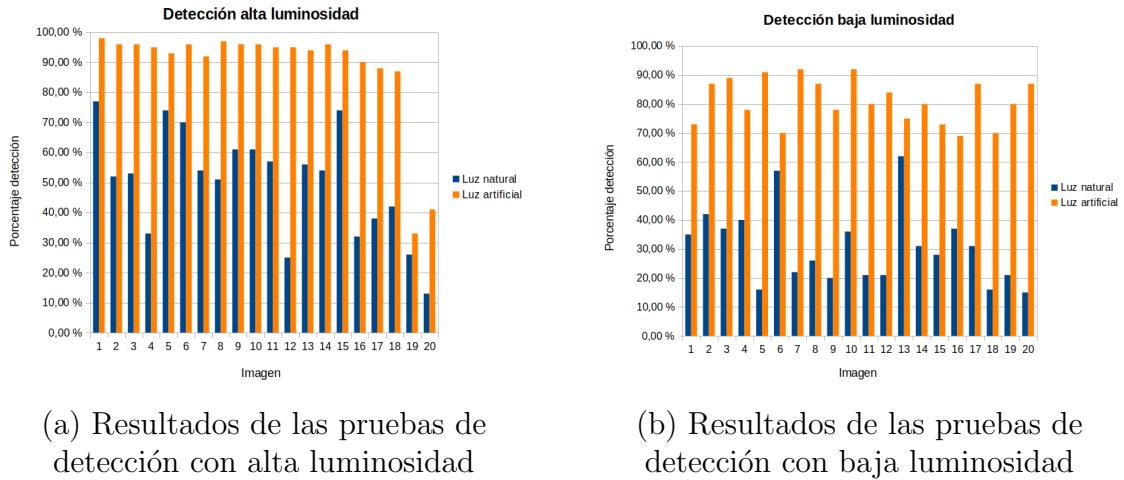


Figura 6.9: Gráficas de la confianza de detección obtenida en las pruebas según la luminosidad para el modelo ssd mobilenet v2

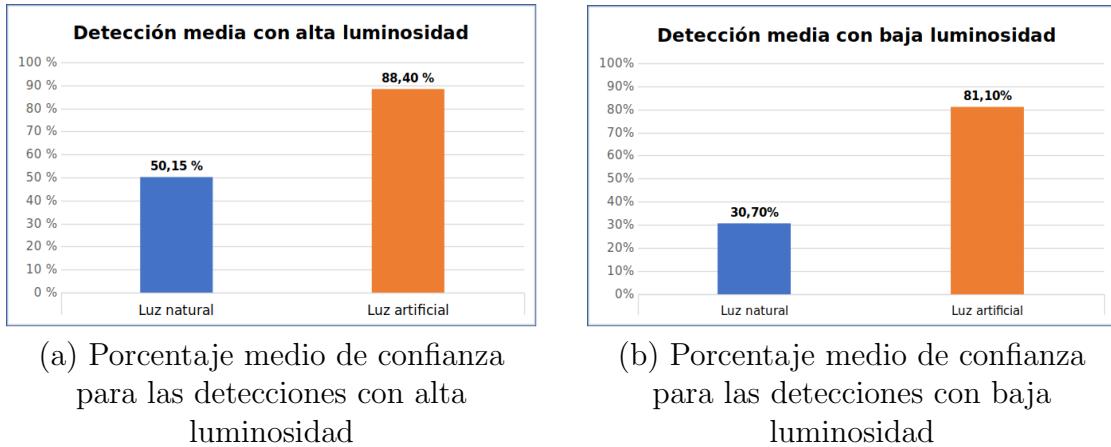


Figura 6.10: Gráficas de la media de los porcentajes de confianza obtenidos en las pruebas de detección según la luminosidad para el modelo ssd mobilenet v2

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, a la que también se le cambió el nombre por *strawberry detection*, calculando las coordenadas del punto central del recuadro de la detección, o añadiendo del cálculo de los FPS (fotogramas por segundo) en la ventana de detección en tiempo real (ver Figura 6.11), es decir, la medida de la frecuencia de cuadros en el vídeo, ya que representa la cantidad de imágenes individuales que se muestran en un segundo, midiendo la velocidad de procesamiento de los cuadros, siendo útil para comparar entre las distintas condiciones de detección, ya que un FPS más alto indica que se están procesando más cuadros por segundo, lo que generalmente se considera deseable para

aplicaciones en tiempo real.

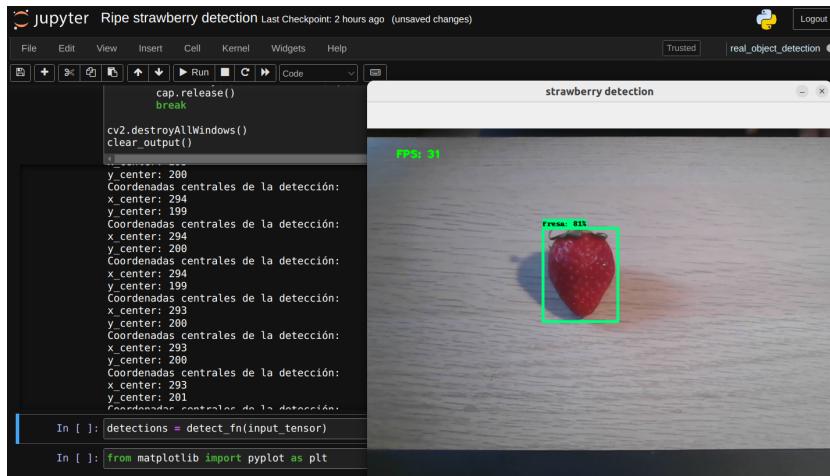


Figura 6.11: Detección de fresas en Jupyter Notebook

Aunque Jupyter Notebook ofrecía un entorno interactivo y muy útil 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, los cuales requieren un comportamiento determinista, autónomo y ejecutable sin supervisión constante. 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 de un sistema conectado a un robot debe realizarse 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.

6.2. 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 Tensorflow.

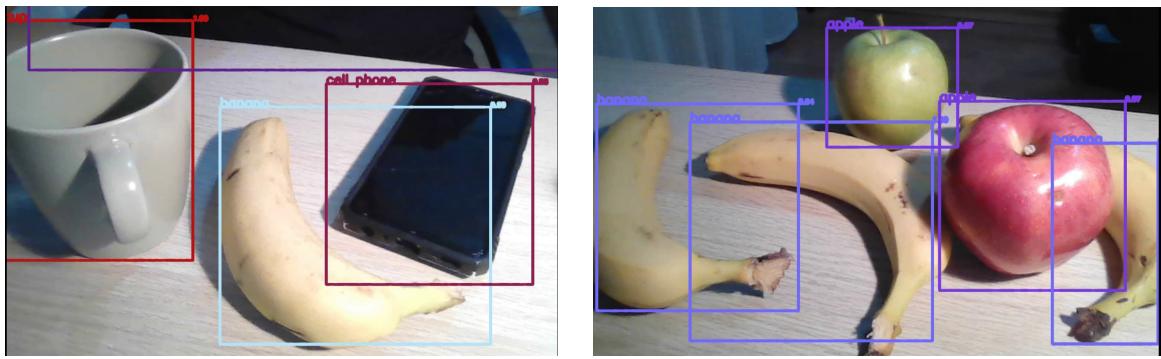
6.2.1. Pruebas con modelos preentrenados

Después de realizar la lectura *You Only Look Once: Unified, Real-Time Object Detection*[?], 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 Open Source Computer Vision Library (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, primero se realizó la instalación de Anaconda para poder crear un ambiente de trabajo independiente y así evitar problemas entre las versiones de los paquetes necesarios para la ejecución de estas pruebas, y posteriormente se instaló OpenCV. Se clonó el repositorio *deteccion-objetos-video*⁶ basado en el proyecto *PyTorch-YOLOv3*⁷ para correr detección de objetos sobre video y se siguieron los pasos detallados en el archivo README. Una vez instalado todo, se probó a utilizar varios objetos y posteriormente varias frutas simultáneamente para verificar que el modelo las diferenciaba correctamente y las detectaba, tal y como se muestra en la Figura 6.12.

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 6.12: Detección con Pytorch

6.2.2. Entrenamiento del modelo

Debido a que, tras la detección de las posiciones de las fresas mediante el programa de Jupyter Notebook, no se puede trabajar directamente con ellas y enviarlas al robot, ya que Jupyter Notebook está basado en web, por lo que haría falta un servidor web intermedio para esto, tal y como se explicó en el apartado anterior, se intentó llevar a cabo la elaboración del programa de detección en Python, para que de este modo se pudieran enviar al brazo robótico directamente las coordenadas, al estar trabajando en local y tener todo el código unificado. Además, 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 como Jupyter.

Para ello se tomó 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 nuevo en Anaconda llamado *deteccionobj*.

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

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

Durante el entrenamiento del modelo, surgieron varios códigos de error relacionados con el etiquetado de las imágenes utilizadas para esto, por lo que se decidió etiquetarlas de nuevo mediante el programa labelImg (Figura 6.13), tal y como se había hecho anteriormente con TensorFlow.

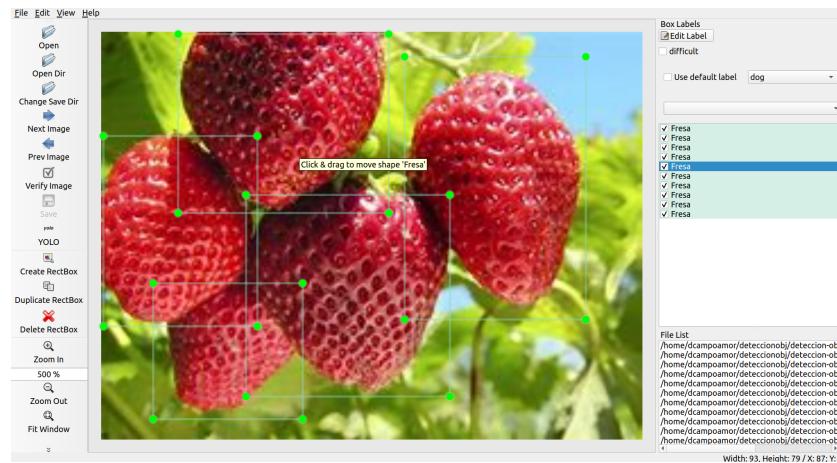


Figura 6.13: Etiquetado de las imágenes con labelImg

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 se empleó 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 generó un checkpoint con los pesos actuales del modelo, almacenado en la carpeta correspondiente, dando lugar a un total de 100 checkpoints al concluir el proceso dado que el entrenamiento fue configurado por defecto para ejecutarse durante 100 épocas.

6.2.3. Calibrado de la cámara

Con el fin de optimizar la detección de objetos mediante YOLOv3 y PyTorch, se procedió a la calibración de la cámara web de 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 en diferentes posiciones, tomadas con la cámara a calibrar, mediante las cuales, y a través del uso del programa *PiCamCalibrator.py*¹⁰, se obtenían estos valores de la matriz K, como muestra la Figura 6.14.



Figura 6.14: Calibración de la cámara C270 de Logitech

Para corroborar si esta primera calibración de la cámara C270 de Logitech había sido buena, se llevaron a cabo diez calibraciones más para comprobar los resultados entre sí con diferentes imágenes tomadas con el mismo *chess board*, siendo los valores tomados para la programación, la media aritmética entre todas las mediciones.

Junto con las matrices de Rotación R y la matriz de Traslación T de la cámara, se realizó el cálculo de la transformación del sistema de coordenadas que viene definido por la multiplicación de las tres matrices, para lo cuál, se instaló la cámara en un trípode, cuya altura al plano mesa conocíamos, y 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 6.15.

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

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



Figura 6.15: Medición del ángulo de rotación de la cámara mediante la aplicación de ERGONAUTAS RULER

6.2.4. Pruebas detección de fresas en tiempo real

Una vez con el modelo entrenado, se realizaron las primeras pruebas utilizando Python de detección de fresas en tiempo real, 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 6.16, y a partir de las cuales se modificaron tanto el grosor del nombre de la clase a detectar como el threshold de la detección mostrado en la ventana emergente en OpenCV, ajustándole tal manera quemo argumento que pudieran ser más legibles en el programa.

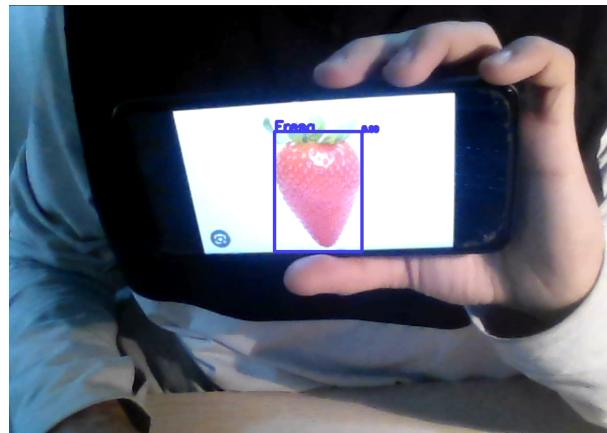


Figura 6.16: Primeras pruebas de detección con PyTorch y Python

Sobre esta primera versión se fue modificando el código para poder añadir más funcionalidades al sistema, como la incorporación de un fragmento diseñado para almace-

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

nar 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 euclíadiana 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.

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 lenguaje 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 pixeles. 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 en 2D, proyectar el punto en el espacio tridimensional (3D) 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.

Se utilizó un cuadrado de un post-it subrayado con color amarillo fosforescente 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 el 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 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 6.17).

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

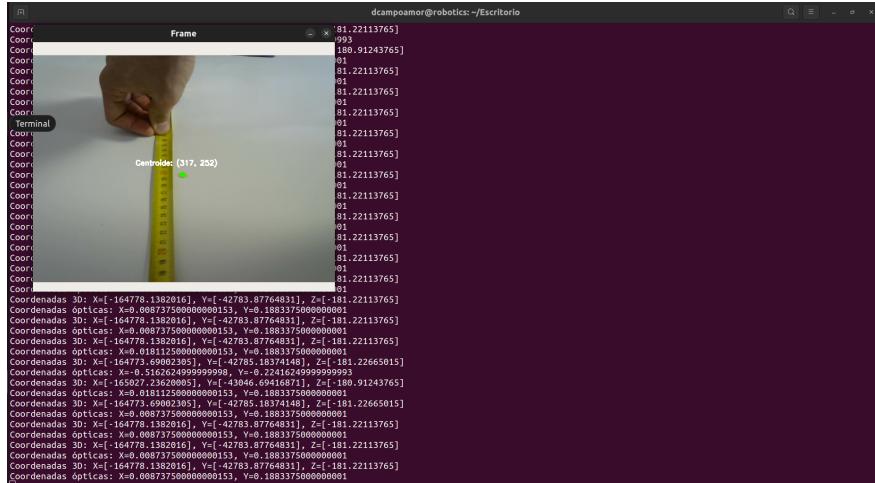


Figura 6.17: Primeras pruebas de la estimación de las coordenadas y la distancia de la detección a la cámara

De los resultados de estas mediciones, recogidos en el Cuadro ??, se observó que existía algún tipo de error en el proceso de cálculo, posiblemente en las transformaciones aplicadas o a la omisión de algún factor relevante, ya fuera geométrico o de calibración, ya que se manifestaba en 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 6.2: Comparacion 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*[?], donde se analizaba la geometría del modelo de cámara basado en el modelo estenopeico o modelo pinhole (Figura 6.18), 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.

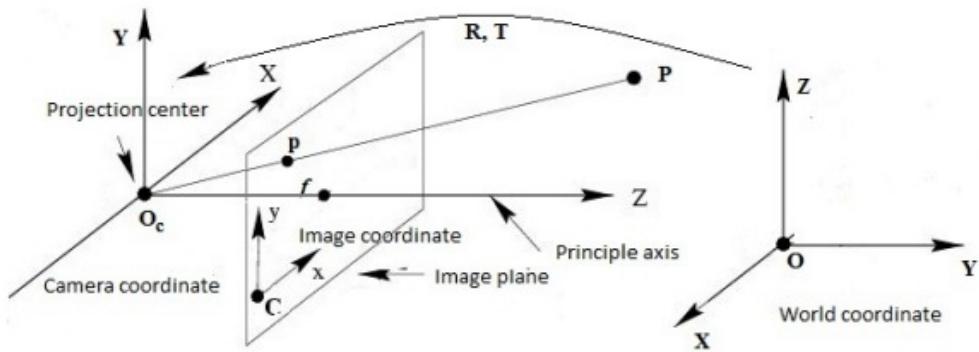


Figura 6.18: Geometría basada en el modelo de cámara estenopeica

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. 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, cuya medida era de 22º para estos experimentos, 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

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

progresiva el post-it en distintas direcciones para poder validar las suposiciones sobre la configuración del sistema de coordenadas (Figura 6.19).

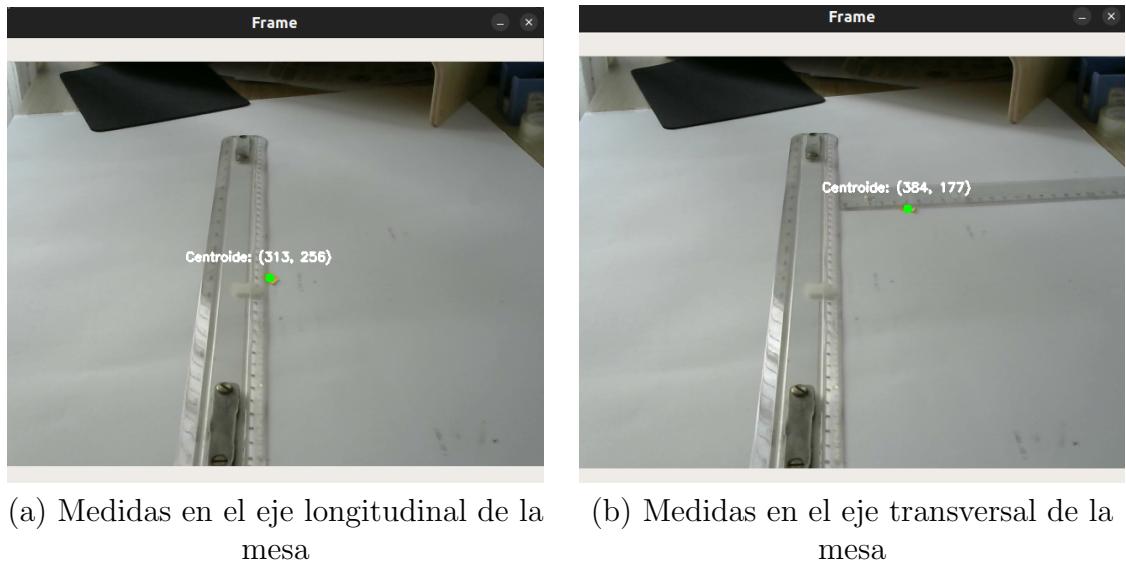


Figura 6.19: Pruebas para determinar la configuración del sistema de coordenadas de la cámara

A partir de los resultados obtenidos en estas pruebas, recogidos en los Cuadros 6.3 y 6.4, 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. Esta configuración permitía que, al alejar el post-it amarillo del origen, la distancia estimada aumentara progresivamente, tal y como era esperable.

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 6.3: 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) (22° Rotación)			Coordenadas mundo (mm) (-22° 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 6.4: 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 en 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 90 grados 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 6.20).

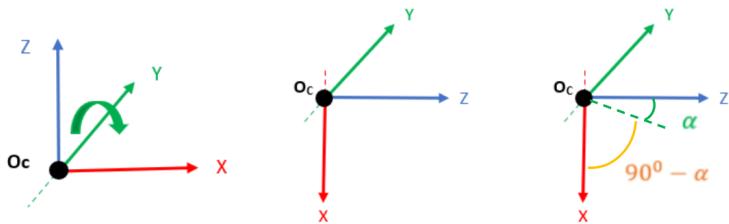


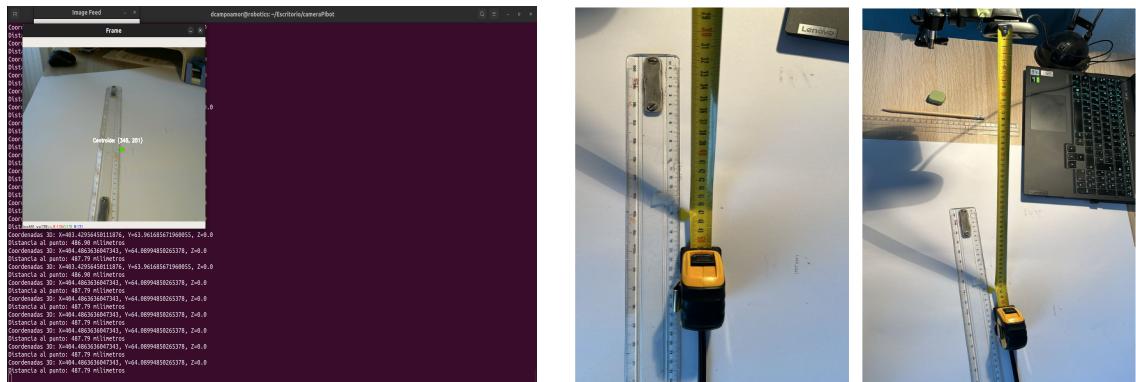
Figura 6.20: Esquema de la rotación de la cámara

Se consideró entonces una rotación sobre el eje vertical 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 6.5.

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 6.5: 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*, que permitía estimar la distancia euclídea desde el origen del sistema hasta el punto proyectado (Figura 6.23).



(a) Detección y cálculo de las coordenadas y la distancia a la detección

(b) Comprobación de la medición de la distancia de la cámara a la detección

Figura 6.21: Cálculo de la distancia de la cámara 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 6.22).

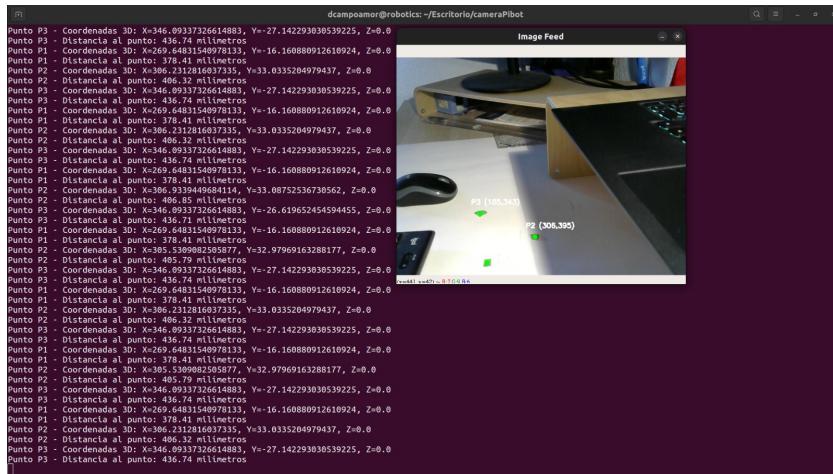


Figura 6.22: Detección múltiple simultánea de post-it

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

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

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 6.23. 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.

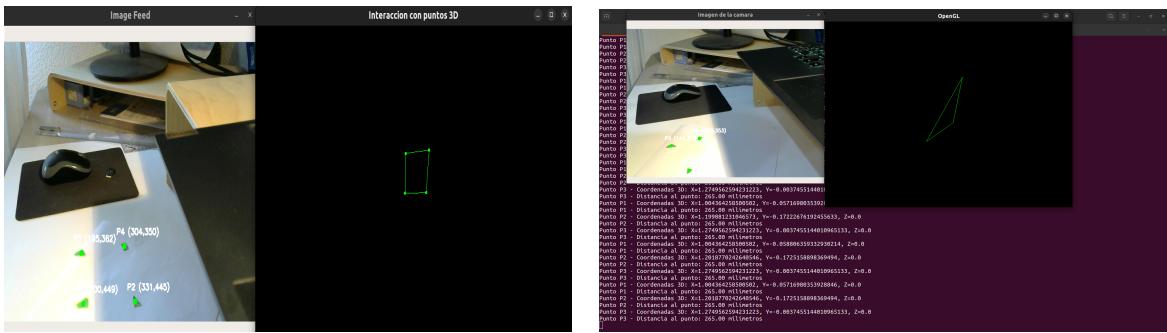


Figura 6.23: Representación de las detecciones con OpenGL

Una vez validado el funcionamiento del sistema con fragmentos de post-it pintados, y comprobada la capacidad del programa para detectar múltiples puntos, 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 específicamente para identificar fresas, que ya

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

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

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 6.24.

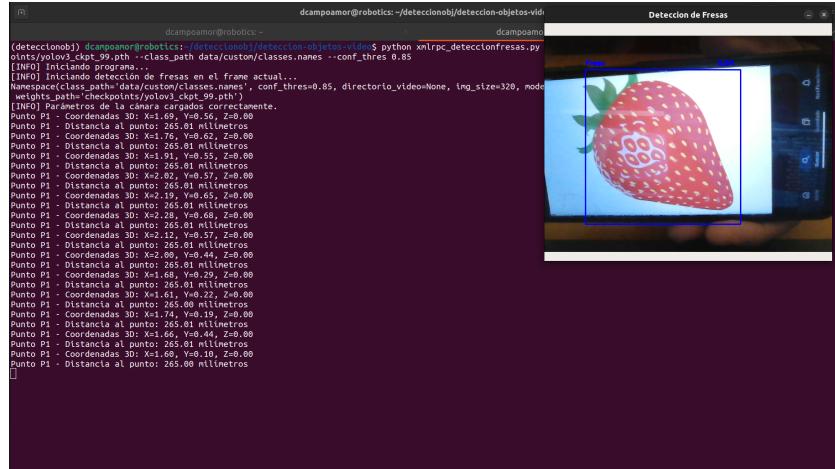


Figura 6.24: Detección de fresas e integración con el sistema de cálculo de coordenadas y distancias

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 detecciones 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 6.6, 6.7 6.8 y 6.10).

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 6.6: 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 6.7: 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	233,89
P33	-50	-65	225	239,48	-63,22	49,21	225	238,84

Cuadro 6.8: 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 6.9: 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 (imagen) al espacio 3D se estaba realizando correctamente, tal y como muestra la Figura 6.25.

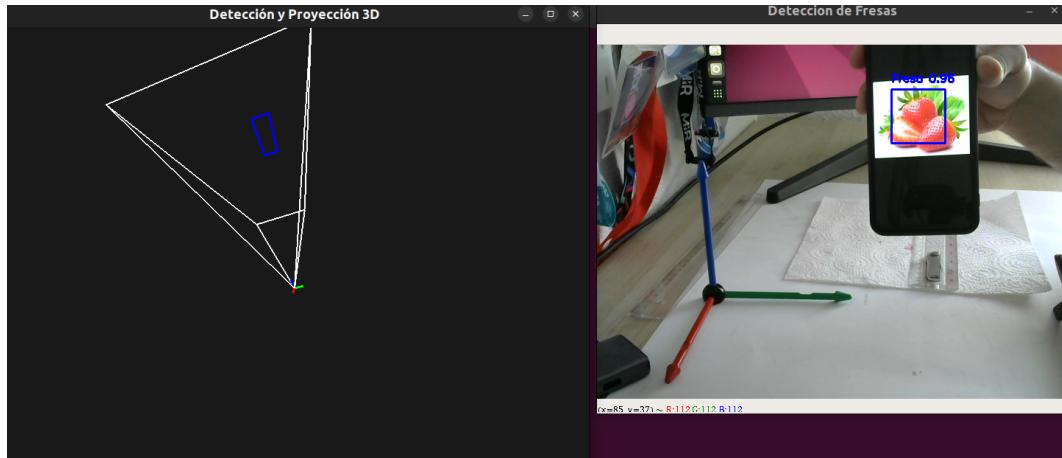


Figura 6.25: Proyección con OpenGL de las detecciones y el campo visual de la cámara

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, como se puede apreciar en la Figura 6.27, 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 6.10: 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 proyectados y los reales podría responder a una rotación o permutación de los mismos, lo que indica una posible equivalencia o relación entre los ejes definidos y los reales, 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 6.26. Esta equivalencia se detalla en la Ecuación 6.1.

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

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

Ecuación 6.1: Equivalencia entre las coordenadas supuestas y obtenidas

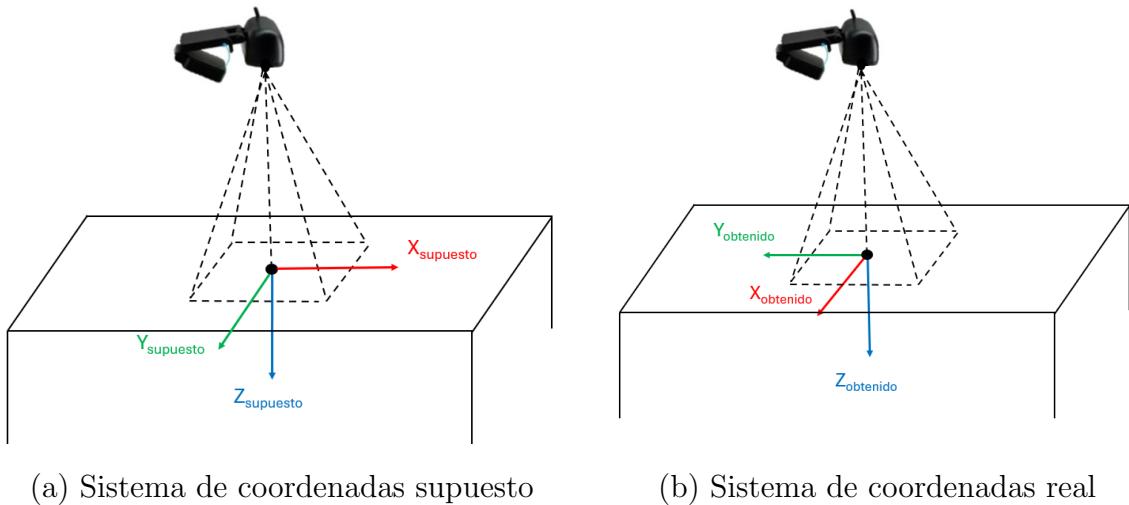


Figura 6.26: Representación de los sistemas de coordenadas que se había supuesto en un principio y del real obtenido

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 6.27), siendo esta vez para la serie de puntos que se encuentran en el Cuadro 6.11, y dando de esta manera por terminadas este tipo de pruebas dada la exactitud obtenida en estos últimos resultados.

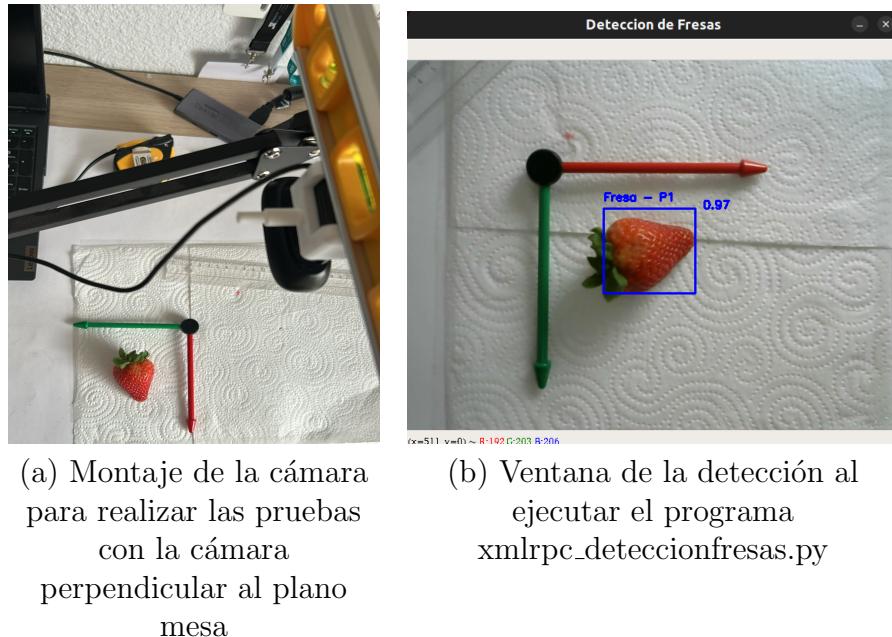


Figura 6.27: Representación del montaje y los sistemas de coordenadas obtenidos para las pruebas con la cámara perpendicular al plano de la mesa

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

6.3. Pruebas con el robot real

Una vez decidido que la mejor opción para la detección de fresas era utilizar el modelo YOLOv3, implementado con TensorFlow 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, así como validar la precisión del movimiento del robot hacia las posiciones detectadas en el plano, permitieron identificar posibles ajustes en la calibración y evaluar el funcionamiento del sistema en un entorno controlado antes de su aplicación conjunta.

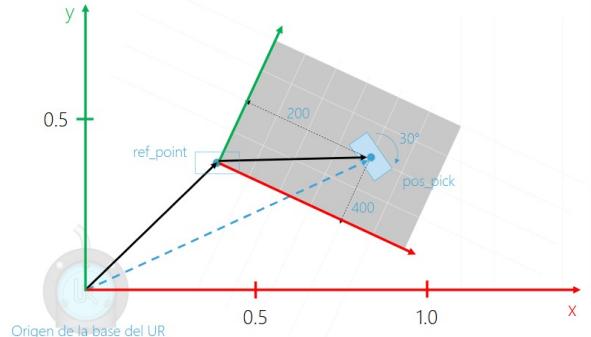
En primer lugar, se programó en la Interfaz Gráfica de Usuario (IGR) del robot el programa *visionsimple.urp*¹⁹ para ir a una posición fija que simulaba una posición en el espacio determinada por un sistema de visión externo, tal y como se muestra en la Figura 6.28, siendo esto la base del programa final del propio robot.

(a) Programa *visionsimple.urp*

```

1  ▾ Robot Program
2  ♀ + MoveJ
3   Ⓛ punto_ref
4   ♀ └ Datos_Recibidos_Camara
5     Ⓛ 'Esperar la foto realizada por la camara'
6     Ⓛ x:=400
7     Ⓛ y:=200
8     Ⓛ rzi:=30
9   ♀ └ Calcular_Posicion_Pick
10    Ⓛ '# Convert mm to m and deg to rad'
11    Ⓛ pos_base_camara=p[(x/1000), (y/1000), 0, 0, d2r(rz)]
12    Ⓛ pos_pick:=pose_trans(punto_ref, pos_base_camara)
13   ♀ └ Mover_a_pos_pick
14   ♀ + MoveL
15   Ⓛ pos_pick

```



(b) Representación de la transformada entre posiciones

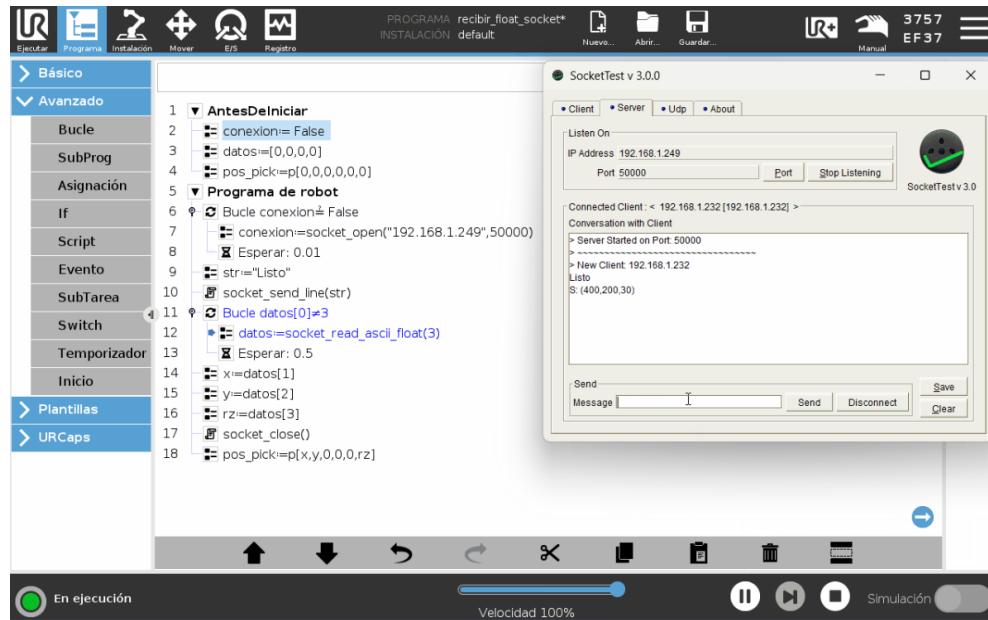
Figura 6.28: Representación básica de un programa de un sistema de visión externo

Se desarrolló el programa *recibir_cadena_socket*²⁰ con el objetivo de establecer una comunicación mediante sockets entre el robot y un servidor externo, como puede ser un sistema de visión. En este programa, 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

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

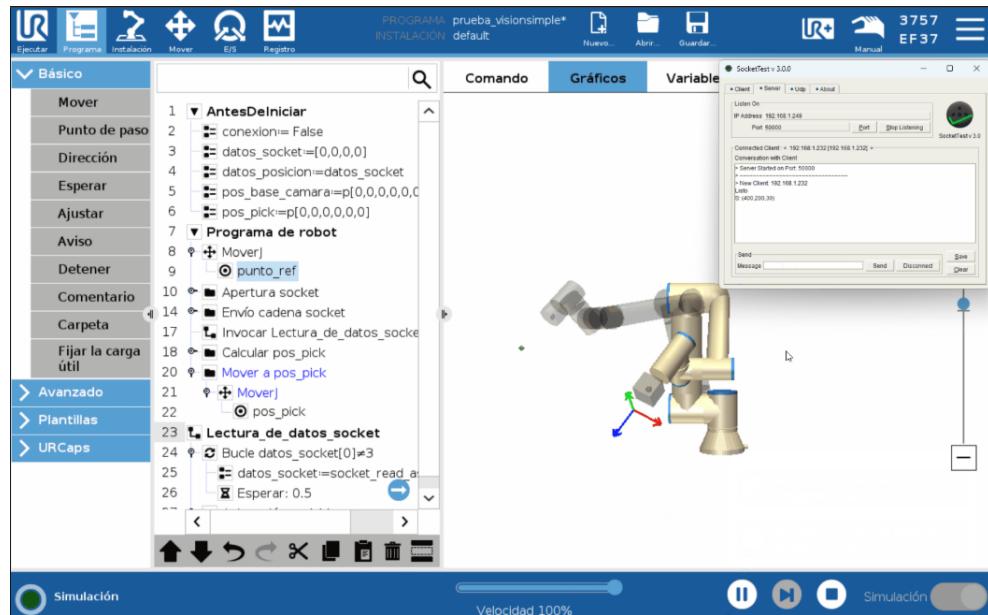
internas antes de cerrar la conexión (Figura 6.12), todo esto, asegurando que ambos dispositivos se encontraban conectados a la misma red local.



Cuadro 6.12: 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 del objeto, y enviar estos datos al robot para que se desplazase a dicha posición (ver Figura 6.13).

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



Cuadro 6.13: Programa prueba_visionsimple.urp

Capítulo 7

Conclusiones

Quizás algún fragmento de libro inspirador...

Autor, Título

Escribe aquí un párrafo explicando brevemente lo que vas a contar en este capítulo, que básicamente será una recapitulación de los problemas que has abordado, las soluciones que has prouesto, así como los experimentos llevados a cabo para validarlos. Y con esto, cierras la memoria.

7.1. Conclusiones

Enumera los objetivos y cómo los has cumplido.

Enumera también los requisitos implícitos en la consecución de esos objetivos, y cómo se han satisfecho.

No olvides dedicar un par de párrafos para hacer un balance global de qué has conseguido, y por qué es un avance respecto a lo que tenías inicialmente. Haz mención expresa de alguna limitación o peculiaridad de tu sistema y por qué es así. Y también, qué has aprendido desarrollando este trabajo.

Por último, añade otro par de párrafos de líneas futuras; esto es, cómo se puede continuar tu trabajo para abarcar una solución más amplia, o qué otras ramas de la investigación podrían seguirse partiendo de este trabajo, o cómo se podría mejorar para conseguir una aplicación real de este desarrollo (si es que no se ha llegado a conseguir).

7.2. Corrector ortográfico

Una vez tengas todo, no olvides pasar el corrector ortográfico de L^AT_EXa todos tus ficheros *.tex*. En Windows, el propio editor TeXworks incluye el corrector. En Linux, usa aspell ejecutando el siguiente comando en tu terminal:

```
aspell --lang=es --mode=tex check capitulo1.tex
```