UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

GRADO EN INGENIERÍA ELECTRÓNICA Y AUTOMÁTICA INDUSTRIAL

"Aplicación de pick and place y de reconocimiento de objetos para el robot IRB140"

TRABAJO FIN DE GRADO

Febrero -2025

AUTOR: Pablo Gómez Mogedas

DIRECTOR/ES: Arturo Gil Aparicio

Luis Miguel Jiménez García

Agradecimientos

Cuando uno se mete en la catedral de conocimiento que es la Universidad, después de realizar el bachillerato, se suele ver como una extensión del periodo educativo. Efectivamente por experiencia se siente como una segunda parte del instituto. Pero, por lo menos estas aprendiendo de un campo que supuestamente te apasiona. O al menos eso es lo que te dices cuando te atascas con esa asignatura que parece que eres incapaz de sacar ni a tiros.

Lo cierto es que la Universidad, puede ser una oficina expendedora de títulos, como mucha gente retracta, pero solo lo es para aquellos que no saben aprovechar el potencial que ofrece. Con un mínimo de interés y movimiento se puede volver en un centro de conocimiento y experiencia sin parangón. Así es que, dentro de la Universidad, he conocido a gente maravillosa, y he vivido experiencias increíbles. Por supuesto que también lleva sacrificio y, por ejemplo, quedarse despierto hasta las cinco de la mañana para acabar un proyecto, pero sinceramente, todo lo que merece la pena en esta vida requiere de esfuerzo y sacrificio pues si fuera fácil, no nos agradaría de la misma manera, algo que se consigue fácilmente no tiene el mismo valor.

Es por eso por lo que me gustaría agradecer primero a todas las personas que vinieron antes de esta etapa de mi vida. A mi familia como no podía ser de otra manera, que siempre me ha apoyado y empujado hacia lo mejor. A mis amigos, gente como Luis, Carlos, Ismail, Alex, Isidro, Jota y Bru que me han ayudado a aguantar el estrés, y me han ofrecido oportunidades y experiencias increíbles, incluyendo mi primer trabajo. Y a mis profesores por poco a poco crear la persona que soy y dadme los conocimientos para prosperar. En especial a Mari Ventura por enseñarme inglés.

Ya dentro de la Universidad me gustaría agradecer a mis compañeros de clase y futuros compañeros de gremio, como todos los Alex que conozco, Andrea, Álvaro y Borja, por la ayuda y camaradería mutua al enfrentarnos con la carrera. A los profesores y catedráticos, incluso aquellos que sus asignaturas se me atascaron, por su paciencia a la hora de enseñar y los conocimientos que nos traspasaron, como Luis Paya, Paco Irles, Juan Manuel y Roberto. Al programa IAESTE y el Observatorio Ocupacional, gente

como Aitor, Blanca, Alba y Javi, por conseguir la oportunidad de trabajar fuera de España como ingeniero. Y finalmente a la Delegación de Estudiantes, a la de la EPSE en particular, Héctor, Jonio, Sergio y Laura, por aguantarme como líder y apoyarme en los proyectos que hicimos, aunque no todos salieran bien, pero por lo menos conseguimos volver a tener representación en la AERRAAITI, y demás congresos de estudiantes. Y a la Delegación General en especial, gente como Miguel, Miriam, Joan y Xenia por mostrarme en profundidad el funcionamiento interno de la Universidad y luchar por los derechos del estudiantado. Además de mostrarme y hacer amistades con gente de otras carreras.

Para finalizar me gustaría agradecer a mi tutor de TFG en particular, Arturo Gil, por ayudarme con el TFG y conseguir que dé más de mí, y no realice otro TFG de una instalación eléctrica número 2537.

Índice

Tabla de Ilustraciones

1.Introducción

1.1 Antecedentes

En la actualidad, la automatización y la robótica han adquirido un papel fundamental en diversos sectores industriales, permitiendo optimizar procesos, mejorar la precisión y reducir costos. Cada vez más, los robots industriales, gracias a su capacidad de realizar tareas repetitivas, precisas y exigentes están abarcando tareas tales como la paletización, el ensamblaje, la soldadura y el embalaje, entre otros. Una de las tareas más comunes y representativas de la robótica es el pick and place, que consiste en la manipulación de objetos desde un punto de origen hasta un destino específico de forma automatizada. Este tipo de operación es esencial en cadenas de montaje, almacenes automatizados y líneas de producción. Pero es una tarea repetitiva y monótona para un trabajador y que además puede conllevar riesgos de lesiones.

Los robots industriales como el IRB140 han transformado la producción al proporcionar una consistencia y calidad inigualables en las operaciones de fabricación. Su capacidad para trabajar incansablemente, manejar cargas pesadas y operar en entornos peligrosos los convierte en activos invaluables para la industria. Además, su programabilidad avanzada permite una adaptación rápida a diferentes tareas y productos, lo que aumenta la flexibilidad de las líneas de producción. En este contexto, la integración de robots industriales no solo mejora la eficiencia y la calidad, sino que también libera a los trabajadores humanos de tareas repetitivas y potencialmente peligrosas, permitiéndoles centrarse en aspectos más creativos y estratégicos de la producción.

Por otra parte, en las últimas décadas la visión por computador ha vivido un auge gracias a los avances en la capacidad computacional del hardware, el desarrollo de nuevos algoritmos y la disponibilidad de enormes conjuntos de datos que facilitan el entrenamiento o “Big Data”. Desde la industria, hasta la salud, pasando por la automoción, el impacto de la visión artificial es cada vez más profundo.
Dentro de la industria podemos ver su aplicación en sistemas de inspección y control de calidad, aplicaciones de “pick and place” inteligentes, soldadura guiada y navegación del entorno, entre otros.

1.2 Objetivos

El objetivo global de este trabajo es el desarrollo de una solución dentro del entorno de simulación de Coppelia Sim para una aplicación de "Pick and Place" usando el robot industrial de ABB, IRB140, y un sistema de visión por computador basado en redes neuronales. La integración de estas tecnologías tiene como objetivo la detección, identificación y manipulación de objetos en un entorno virtual, reproduciendo así escenarios industriales de manera precisa y controlada. Para lograr esto, se plantean los siguientes objetivos específicos de aprendizaje y simulación:

1.2.1 Objetivos de aprendizaje

- Investigar las capacidades del robot IRB140
- Familiarizarse y comprender el software de simulación CoppeliaSim y la API para su funcionamiento en Python, ZeroMQ.
- Investigar y comprender el funcionamiento de la librería PyArte
- Comprender el funcionamiento del algoritmo de detección de objetos YOLO

1.2.2 Objetivos de la simulación

- Generar una simulación del entorno de trabajo en Coppelia Sim
- Ser capaz de controlar el movimiento del robot IRB140
- Añadir una cámara en la simulación y que sea capaz de comunicarse correctamente con la API de Python
- Captar la imagen de la cámara dentro de la simulación y poder preparar la imagen captada para el procesamiento de datos
- Con la información conseguida de la imagen gracias a los algoritmos de reconocimiento de objetos dictar el movimiento del robot
- Añadir modelos y texturas al entorno de Coppelia Sim que puedan ser reconocidos por el algoritmo de detección de objetos
- Encontrar la posición óptima para agarrar un objeto
- Que el robot sea capaz de realizar una ruta y coger y soltar objetos en un punto específico.

El cumplimiento de todos estos subobjetivos debería dar como resultado una aplicación de “pick and place” que utiliza el robot IRB140, e integra el algoritmo de detección de objetos YOLO, mediante el uso de una cámara ubicada en la muñeca del robot. Esta aplicación debería poder detectar varios tipos de objetos y ser capaz de agarrarlos y transportarlos.

1.3 Estado del arte

La robótica, entendida como la creación de máquinas automatizadas, tiene raíces que se remontan a la antigüedad, con ejemplos como el Mecanismo de Anticitera y los autómatas de Heron y Arquímedes; como podemos ver de ejemplo en la Figura 1 . Estos mecanismos adelantados a su tiempo han surgido en múltiples civilizaciones y épocas de la historia, lo que demuestra un ímpetu del ser humano por querer aligerar su carga de trabajo y crear máquinas capaces de repetir las mismas labores que ellos realizan tanto física como mentalmente. Por seguir con el ejemplo el mecanismo de Anticitera se trataba de una computadora mecánica capaz de predecir posiciones astronómicas y eclipses. Y muchos de los autómatas ideados servían para automatizar trabajos físicos. Sin embargo, hay que señalar que la mayoría de los autómatas no tenían una utilidad y servían para entretener a sus dueños e impresionar a las visitas. Este fue el estado de los autómatas durante la mayor parte de su historia, sin embargo, existían autómatas que ayudaban al ser humano a comprender y medir su entorno como podrían ser los relojes mecánicos de los ingenieros árabes o la máquina que señalaba el epicentro de un terremoto inventada por los chinos. Si bien muchos genios diseñaron autómatas que realizaban el trabajo de las clases bajas, muchos de estos diseños no se llegaron a construir por variedad de factores, entre ellos socioeconómicos, pero también, de materiales, energía, conocimiento y demás (Ortiz, 2024).

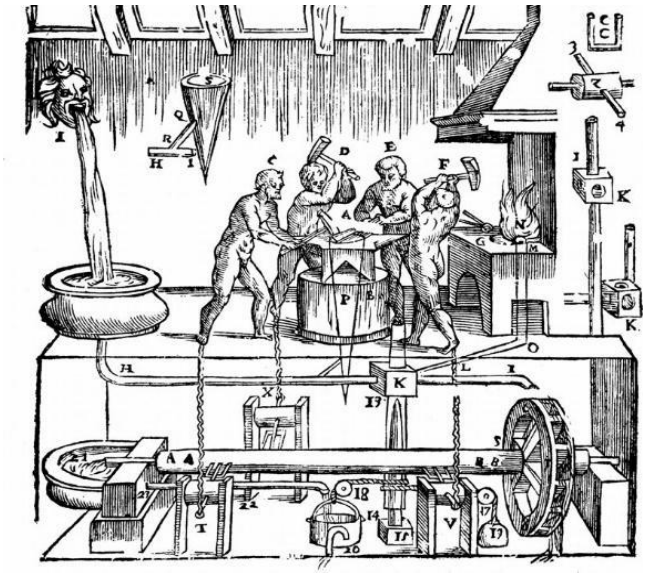


Figura 1. Autómatas accionados por agua corriente en una fragua diseñada por Heron

No fue hasta finales del siglo XVIII y principios del XIX, que se desarrollaron lo que podrían considerarse como los primeros autómatas modernos diseñados para la industria como los telares mecánicos de Cartwrigth en 1785 y el telar de Jacquard de 1801. Este último de hecho fue el primero en aplicar las tarjetas perforadas para programar un trabajo. Eligiendo un conjunto de tarjetas se definía el tipo de tejido a realizar, podemos verlo en la Figura 2.

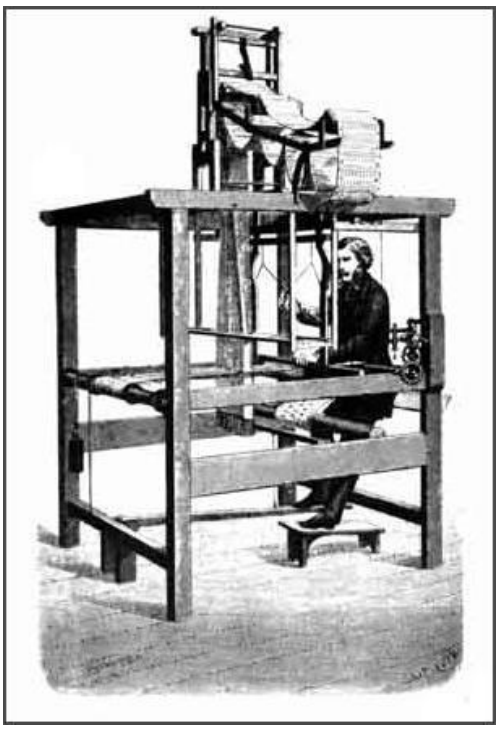


Figura 2.Telar de Jacquard

Es a partir de la revolución industrial que el desarrollo de los automatismos se acelera. La estandarización de las herramientas, los incentivos para mejorar la industria, el acceso de mayor parte de la población al conocimiento, entre otros factores hacen que esta rama del conocimiento crezca de forma explosiva. Lo que hace que la robótica entre en el imaginario colectivo, como una de las tecnologías definitorias de lo que sería el futuro. La propia palabra robot proviene del checo, se traduce como trabajo duro o servidumbre y fue usada en su contexto moderno en la obra de Karel Čapek, R.U.R. Robots Universales Rossum en 1920, si bien en la obra los robots, no son mecánicos, sino que se trata de humanos gestados artificialmente, o lo que la ciencia ficción moderna llamaría clones. Fue esta palabra, robot la que se empezó a usar entre escritores de ciencia ficción como Isaac Asimov para referirse a las máquinas que hoy nos imaginamos. Existía por tanto en el imaginario colectivo la noción de que enseguida se construirían robots humanoides capaces de usar herramientas humanas y realizar todo tipo de trabajos. De ejemplo en la Figura 3 tenemos dos imágenes de 1954, la portada de la revista Galaxy, mostrando una ginoide y el robot humanoide más avanzado de su época, GARCO.

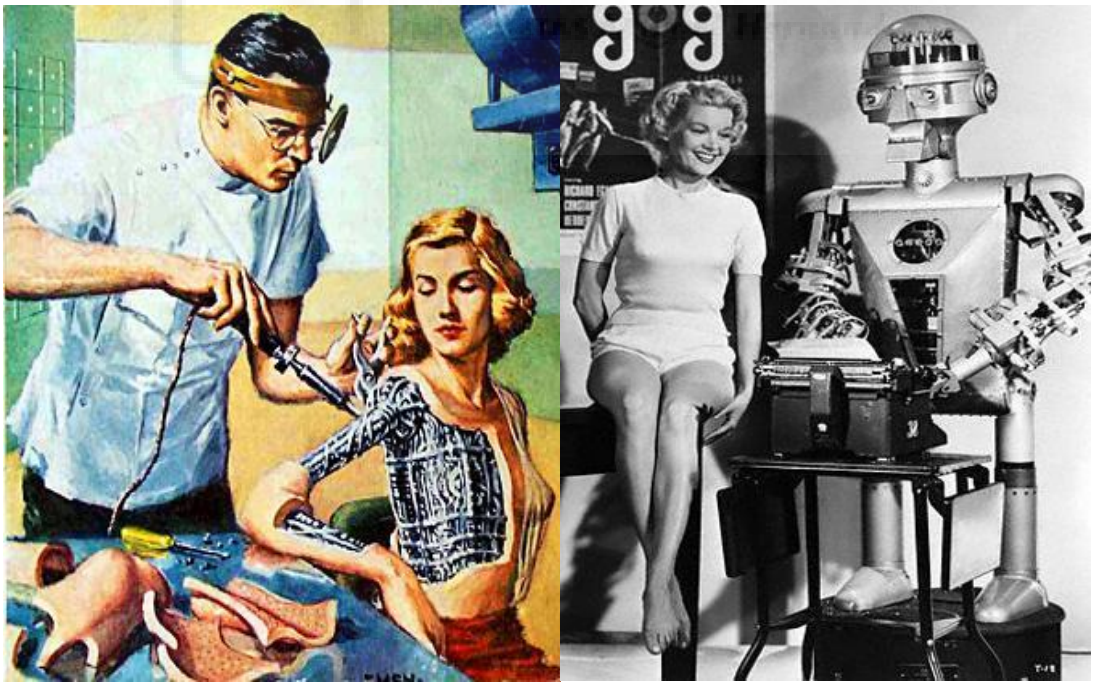


Figura 3 Portada de la revista Galaxy de septiembre de 1954 y foto tomada de GARCO, uno de los primeros intentos en crear un robot humanoide multipropósito, foto de 1954.

Sin embargo, la creación de robots con forma humana que fueran realmente viables es algo que se escapaba a la tecnología de la época, como exponía la paradoja de Moravec,

el pensamiento razonado humano requiere de relativamente poca capacidad computacional mientras que las habilidades sensores y motoras requieren de gran capacidad de computación.

Es por tanto que se podría argumentar que el campo de la robótica evolucionó de querer crear androides capaces de hacer el mismo trabajo que un humano y usar las mismas herramientas, a empezar a robotizar las propias herramientas.

En 1947, Delmar S. Halder que trabajaba para la Ford Motor Company, acuñó el término automatización, y la idea se empezó a expandir por la industria norteamericana. Según Halder, la automatización debería ser un concepto global que abarcara todos los diseños y dispositivos realizados para conseguir una plena automatización de la producción.

Un hito crucial fue la creación de Unimate en 1954 por George Devol, el primer robot industrial programable, su accionamiento era hidráulico, tecnología que dominó el incipiente negocio de los robots industriales en su primera década. Su implementación en General Motors en 1961 marcó el inicio de la automatización industrial a gran escala. Un joven ingeniero noruego, Ole Molaug, desarrolló el primer robot de pintura del mundo con accionamiento hidráulico en 1966. Se diferenciaba del Unimate en que tenía control continuo del recorrido. Se programaba registrando en una cinta magnética los patrones de pulverización de un pintor experto. En 1969, Victor Scheinman creó el Stanford Arm en la Universidad de Stanford, el primer brazo robótico multiprogramable, ligero y

totalmente eléctrico. Este diseño sentó las bases para la producción de brazos comerciales más versátiles. Lo podemos observar en la Figura 4

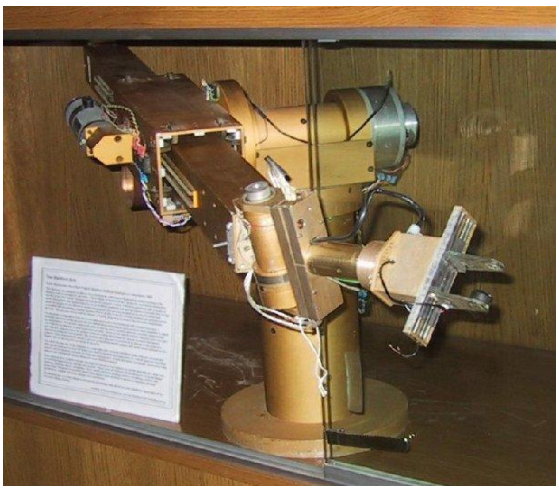


Figura 4. Standford Arm expuesto

Los años 70 vieron una rápida expansión de la robótica industrial. En 1974, ASEA (ahora parte de ABB) desarrolló el ASEA IRB 6, el primer robot totalmente eléctrico controlado por un microprocesador de Intel. La elegancia del diseño del IRB 6 era tal que su cinemática antropomorfa básica, con movimientos giratorios de las articulaciones, se puede ver en los robots actuales (Robots Done Right LLC, s.f.). Lo que ha cambiado a lo largo de los años es la velocidad, la precisión y el ahorro de espacio, con envolventes de trabajo más amplias y dimensiones más reducidas. En 1978, Victor Scheinman, trabajando dentro de Unimation lanzó al mercado el brazo robótico PUMA, que se convirtió en un estándar en la industria (Scalera-, 2019). En la Figura 5 podemos ver ambos.

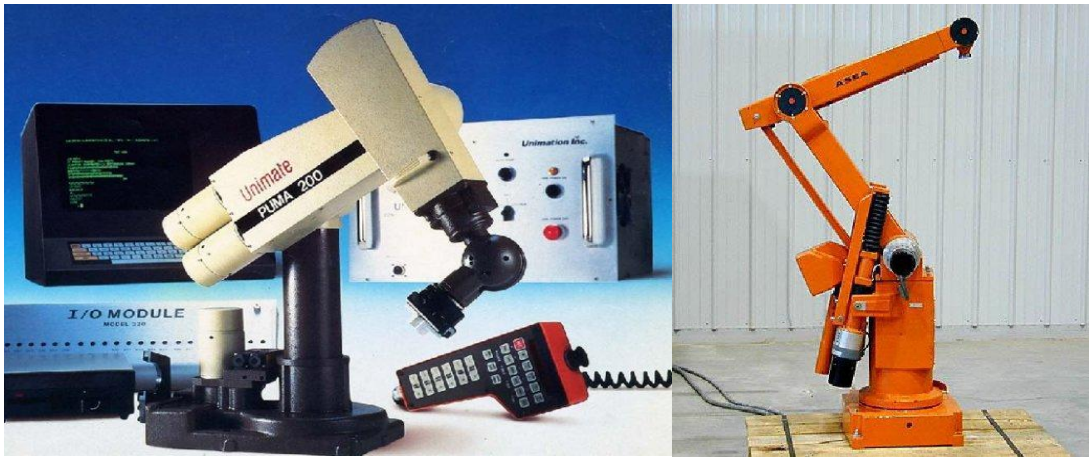


Figura 5. Robot PUMA 200 de Unimate en la izquierda y el robot IRB6 de ESEA a la derecha

No deberíamos olvidarnos de los fallos que se cometieron por el camino, y que como ingenieros debemos recordarlos para no cometer los mismos errores. El primer humano que falleció por culpa de un robot fue Robert Williams, un trabajador de una planta de Ford, el incidente ocurrió en 1979, Robert fue golpeado en la cabeza por el brazo mecánico del robot, falleciendo en el acto. El robot continuó operando durante 30 minutos hasta que otros trabajadores se dieron cuenta de lo ocurrido. Dos años después en Japón ocurrió una tragedia similar. El ingeniero Kenji Urada estaba reparando un robot, cuando lo activó accidentalmente y un brazo hidráulico lo aplasto. Estos casos son terribles, pero, afortunadamente, raros, de hecho, con el desarrollo de la tecnología el número de accidentes en la producción ha disminuido significativamente.

Desde la década de los 80, ha habido un crecimiento exponencial en la producción y uso de robots industriales. Hasta la actualidad, los brazos robóticos industriales han experimentado mejoras continuas en precisión, velocidad, capacidad de carga y, más recientemente, en inteligencia y adaptabilidad. La integración de inteligencia artificial y aprendizaje automático ha llevado a la creación de robots más autónomos y flexibles. La tendencia actual se dirige hacia robots colaborativos o "cobots", diseñados para trabajar de manera segura junto a los humanos, ampliando aún más las posibilidades de la automatización industrial. Podemos ver un ejemplo de uso en la Figura 6.

Figura 6. Robot colaborativo de Universal Robotics con operario

Por otro lado, la visión por computador ha evolucionado considerablemente desde sus inicios en la década de 1960, pasando de simples algoritmos matemáticos a modelos avanzados de inteligencia artificial capaces de interpretar imágenes y videos con una precisión comparable a la humana.

En sus primeras décadas, durante los años 1960 y 1970, la visión artificial era un campo teórico. Investigadores del MIT, como Larry Roberts, exploraron cómo los ordenadores podían extraer información tridimensional a partir de imágenes en 2D. Durante este periodo, surgieron algoritmos básicos para la detección de bordes, como los filtros de Sobel y Canny, y se establecieron las bases matemáticas para la segmentación de imágenes. David Marr, en los años 70, propuso una teoría de la visión basada en tres niveles de procesamiento, lo que ayudó a estructurar el campo. Uno de los grandes hitos de este campo de la ingeniería sería en 1975 cuando la NASA mandó las sondas Viking 1 y 2 a Marte. Se necesitaba realizar un análisis del terreno para elegir el lugar de aterrizaje correcto, una superficie plana y segura, por lo que se aplicaron algoritmos de detección de bordes y segmentación de imágenes para analizar sombras y relieves en la superficie marciana. Además, se utilizó un sistema de visión estereoscópico, combinando imágenes desde distintos ángulos para calcular la topografía del terreno. En la Figura 7 podemos ver la primera imagen de Marte tomada desde su superficie.

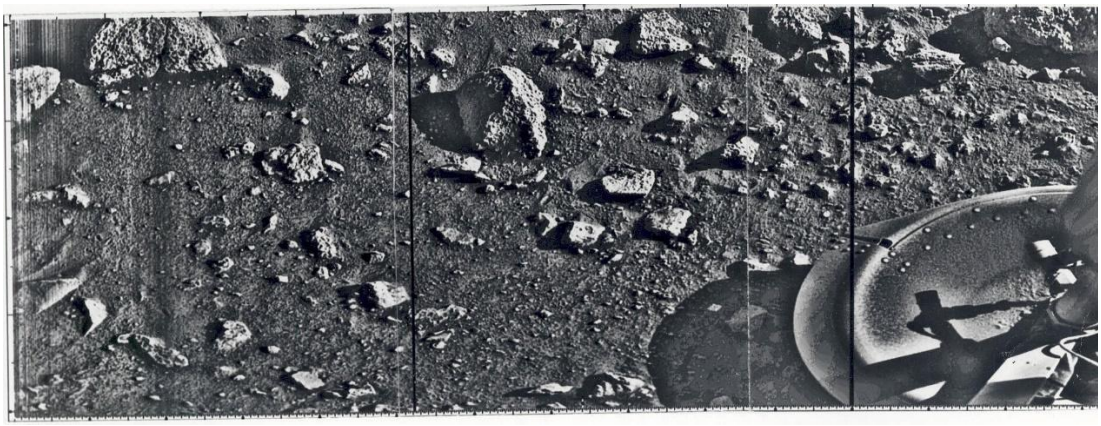


Figura 7. Primera imagen obtenida de la superficie de Marte

Durante los años 80 y 90, los avances en computación permitieron la creación de algoritmos más sofisticados basados en modelos estadísticos y aprendizaje automático. Se desarrollaron métodos como PCA (Análisis de Componentes Principales) para el reconocimiento de patrones y técnicas de extracción de características como SIFT y SURF, que permitían identificar puntos clave en imágenes independientemente de la escala o rotación. También comenzaron a explorarse las primeras redes neuronales artificiales, aunque su implementación era limitada debido a restricciones de hardware.

Con la llegada de los años 2000, la visión por computador dio un gran salto gracias al auge del aprendizaje automático y la creciente disponibilidad de grandes volúmenes de datos. Algoritmos como SVM (Support Vector Machines) se convirtieron en herramientas clave para la clasificación de imágenes. Se crearon bases de datos masivas como MNIST y COCO, lo que permitió entrenar modelos con datos más representativos del mundo real. También se desarrollaron métodos como HOG (Histogram of Oriented Gradients), que mejoraron la detección de objetos, especialmente en aplicaciones como la seguridad vial y el reconocimiento de peatones.

La verdadera revolución llegó en la década de 2010 con la irrupción del aprendizaje profundo (Deep Learning). En 2012, el modelo AlexNet, basado en redes neuronales convolucionales (CNNs), demostró un desempeño superior en el desafío ImageNet, marcando un antes y un después en la visión por computadora. El ImageNet Large Scale Visual Recognition Challenge es una competencia anual que comenzó en 2010 y se ha convertido en uno de los eventos más importantes en el campo de la visión por computador. Su objetivo es evaluar la capacidad de los modelos de inteligencia artificial

para clasificar imágenes en categorías predefinidas. Cada año, los participantes entrenan

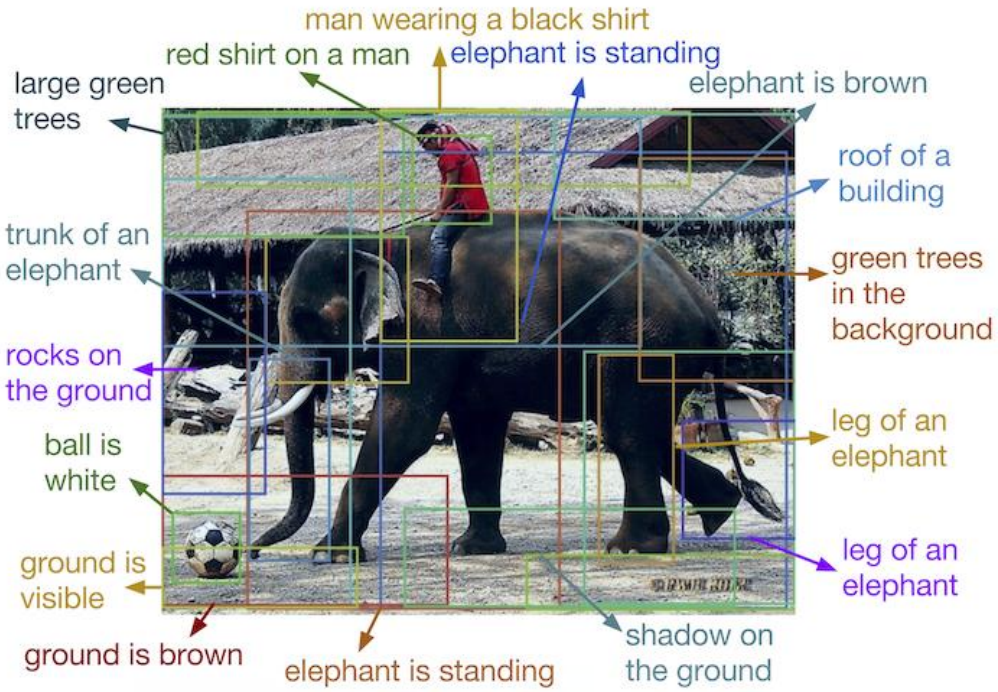


Figura 8. Imagen característica de un dataset. Las imágenes son descritas y todas sus partes importantes señalizadas para poder entrenar a modelos de visión por computador.

sus modelos con el dataset ImageNet, que contiene millones de imágenes etiquetadas en miles de categorías. En 2012, el equipo de Geoffrey Hinton, uno de los pioneros del aprendizaje profundo, participó en el desafío ImageNet con un modelo llamado AlexNet, desarrollado por su estudiante Alex Krizhevsk**y.** El éxito de AlexNet en el desafío no solo demostró la potencia de las redes neuronales profundas, sino que también marcó el comienzo de la era del aprendizaje profundo en visión por computador y otras áreas de la inteligencia artificial (AlexNet and ImageNet: The Birth of Deep Learning, s.f.). Después de este éxito, el interés y la inversión en redes neuronales profundas creció exponencialmente**.** En los años siguientes, arquitecturas como VGGNet, ResNet e Inception mejoraron la precisión y eficiencia de los modelos. Con estos avances, la visión por computadora se aplicó a una gran variedad de industrias, desde la medicina hasta los vehículos autónomos. Para entrenar a los modelos por lo general se requiere de muchas imágenes con anotaciones como podemos ver en la Figura 8

En la actualidad, a partir de 2020, el desarrollo de modelos fundacionales y redes neuronales más avanzadas ha transformado nuevamente el campo. Los Vision

Transformers (ViTs) han introducido una nueva forma de procesar imágenes, reemplazando las convoluciones por mecanismos de autoatención. Modelos como CLIP y DALL·E, que combinan imágenes y texto, han ampliado las capacidades de la visión por computadora en la generación y comprensión de imágenes. Además, técnicas como Segment Anything Model (SAM) han demostrado que es posible segmentar imágenes sin entrenamiento específico.

Me gustaría terminar este apartado sobre la historia de este campo de la ciencia, señalando en que muchas veces los grandes avances no venían de conceptos revolucionarios, sino de una progresión sobre lo que ya existía. En otras palabras, muchas veces como ingenieros, nuestro trabajo no es hacer que lo imposible sea posible, sino hacer que lo posible sea practico. Es con esta mentalidad y este objetivo que he estado trabajando en este proyecto.

2.Técnicas y métodos

En este apartado comprenderemos los fundamentos teóricos de la implementación del programa de “pick and place”. Repasaremos los conceptos de cinemática directa e inversa, el funcionamiento de una cámara y la visión artificial, entre otros, para tener los conocimientos necesarios para entender la implementación del proyecto

2.1 Cinemática del robot

La cinemática es la rama de la ciencia que se centra en el estudio del movimiento de un objeto con respecto de un sistema de coordenadas de referencia. En el caso de un brazo robótico se estudia el movimiento del extremo, y por tanto del resto de eslabones, del robot respecto a un sistema de referencia, que por lo general es la base. De esta manera, se puede describir la cinemática de un brazo robótico como la relación entre la posición y orientación del extremo del robot y los valores angulares de sus articulaciones.

Hay dos problemas fundamentales en la cinemática del robot. El primero es la cinemática directa, que es la determinación de la posición y orientación del extremo del robot, respecto del sistema de referencia de la base, a partir de los ángulos de las articulaciones. El segundo es la cinemática inversa, que es lo contrario, la determinación de los valores angulares de las articulaciones a partir de la posición y orientación del extremo del robot. En la Figura 9 tenemos un esquema con el concepto (Aparicio, 2024).

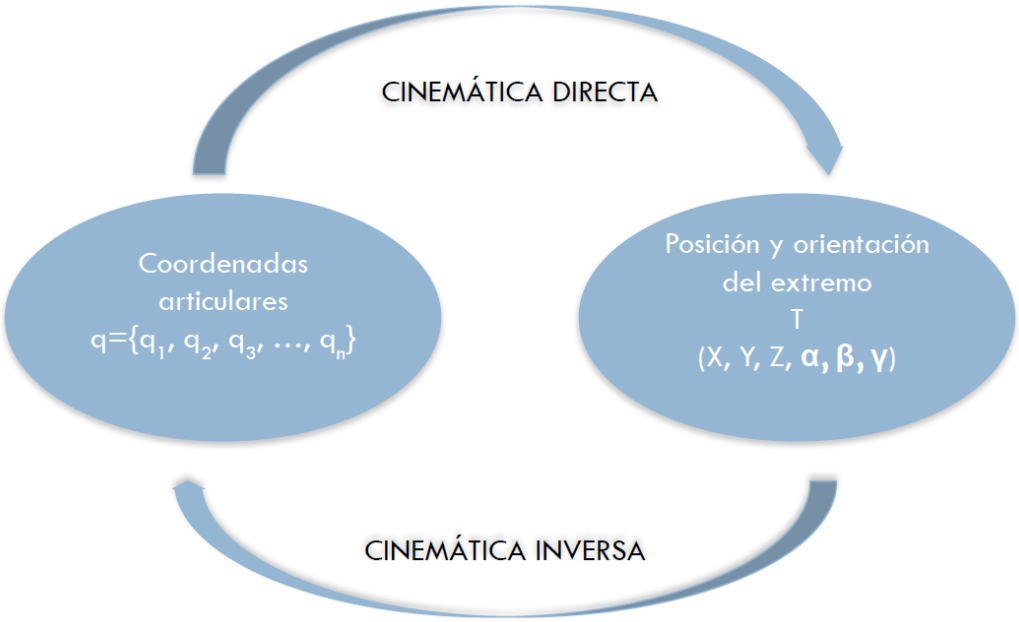


Figura 9. Diagrama de la cinemática directa e inversa de un manipulador

De la misma manera que la velocidad se puede representar como la derivada o el cambio de la posición con respecto al tiempo. En cinemática se puede hallar la relación entre el cambio de las velocidades angulares de las articulaciones y el cambio de velocidad en el extremo final. A esto se le llama matriz jacobiana. Como podemos ver en la Figura 10 (Aparicio, 2024).

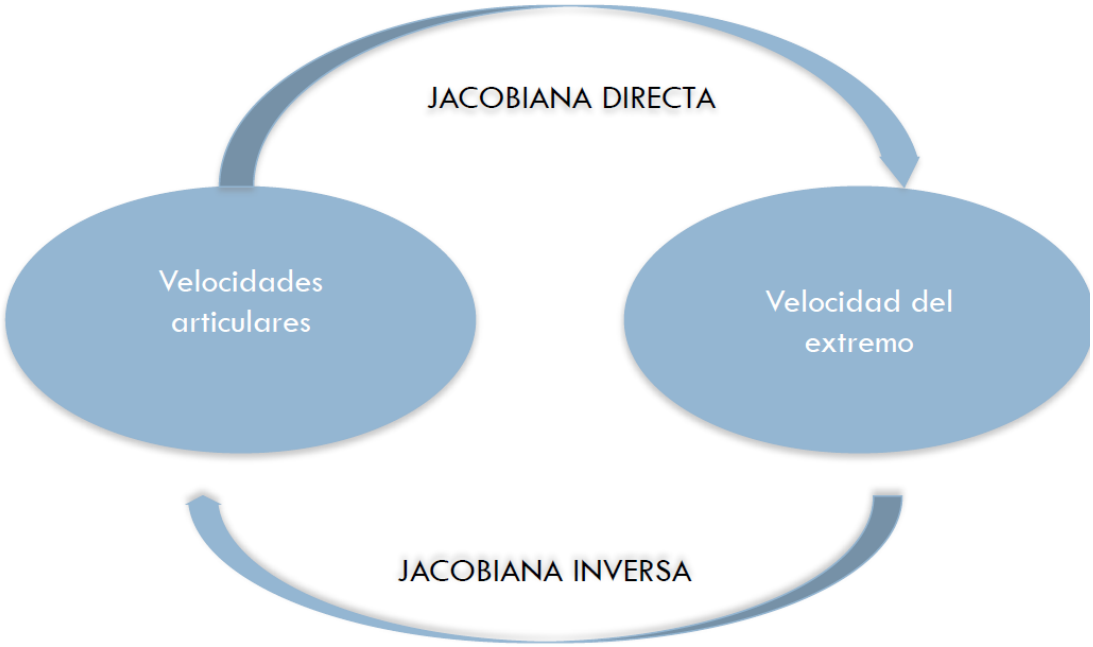


Figura 10. Diagrama de la Jacobiana directa e inversa

2.1.1 Cinemática directa

Jacques Denavit y Richard Hartenberg inventaron un método sistemático matricial en 1955 para calcular la cinemática directa de un robot manipulador. El método de Denavit-Hartenberg (D-H) permite determinar los sistemas de referencia de cada eslabón adjunto, conociendo las características geométricas de estos. El problema de cinemática directa se reduce a encontrar la matriz de transformación homogénea de 4x4 que relaciona la rotación y traslación del eslabón i-1 con el sistema de coordenadas del eslabón i.

La matriz de transformación que relaciona el elemento i-1 con elemento i se compone de cuatro transformaciones que se detallan a continuación:

1. Rotación de un ángulo θ_i alrededor del eje Z_{i-1}
2. Traslación de una distancia d_i a lo largo del eje Z_{i-1} (vector [0,0, di])
3. Traslación de una distancia a_i a lo largo del eje X_i (vector [a_i,0,0])
4. Rotación de un ángulo α_i alrededor del eje X_i

Al multiplicar las rotaciones y traslaciones en el orden correcto se obtiene una matriz de la siguiente forma:

$$^{i-1}A_i = Rot(\theta i) \cdot Tra\ (0,0,\ di) \cdot Tra\ (ai,\ 0,0) \cdot Rot(\alpha i)$$

$$
{}^{i-1}A_i = \begin{pmatrix} \cos\theta i & -\sin\theta i & 0 & 0 \\ \sin\theta i & \cos\theta i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & di \\ 0 & 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} 1 & 0 & 0 & ai \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}
$$
$$
* \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha i & -\sin\alpha i & 0 \\ 0 & \sin\alpha i & \cos\alpha i & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}
$$

Ecuación 1. Matriz de transformación homogénea

Multiplicándolo todo, la forma genérica de una matriz de transformación D-H es:

$$
{}^{i-1}A_i = \begin{bmatrix} \cos(\theta i) & -\sin(\theta i)\cos(\alpha i) & \sin(\theta i)\sin(\alpha i) & ai\cos(\theta i) \\ \sin(\theta i) & \cos(\theta i)\cos(\alpha i) & -\cos(\theta i)\sin(\alpha i) & ai\sin(\theta i) \\ 0 & \sin(\alpha i) & \cos(\alpha i) & di \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

Ecuación 2. Matriz de transformación homogénea simplificada

Con este método se obtendrá un matriz de transformación homogénea para cada grado de libertad del manipulador y la multiplicación de todas las matrices permitirá obtener el modelo cinemático directo del robot.

Para conseguir esta configuración simplificada, hacen falta que se cumplan ciertas reglas:

1. Se numeran los eslabones, asignando el 0 a la base y n para el último eslabón.
2. Se numeran las articulaciones comenzando con 1 para la primera y n para la última, siendo un robot de n grados de libertad.
3. Se colocan los ejes Z_i alineados con el eje de movimiento de la articulación i+1 y marcando su dirección de movimiento positiva. Z_0 se coloca describiendo el movimiento de la articulación 1, Z_1 con la articulación 2 y así progresivamente.
4. El sistema i=n se coloca en el efecto final

5. Comenzamos seleccionando un punto donde colocar el sistema S_0 =[X_0,Y_0,Z_0], y además se debe cumplir con que $Z_0 = X_0 \times Y_0$
6. A continuación, se coloca el vector X_i, comenzando con X_1. La regla general es que X_i debe ser paralelo al vector normal común, calculado como $X_n = Z_i \times Z_{i-1}$, y que, además corte a Z_i.
7. Mostramos esta regla con ciertos casos particulares en la Figura 11. Colocación de

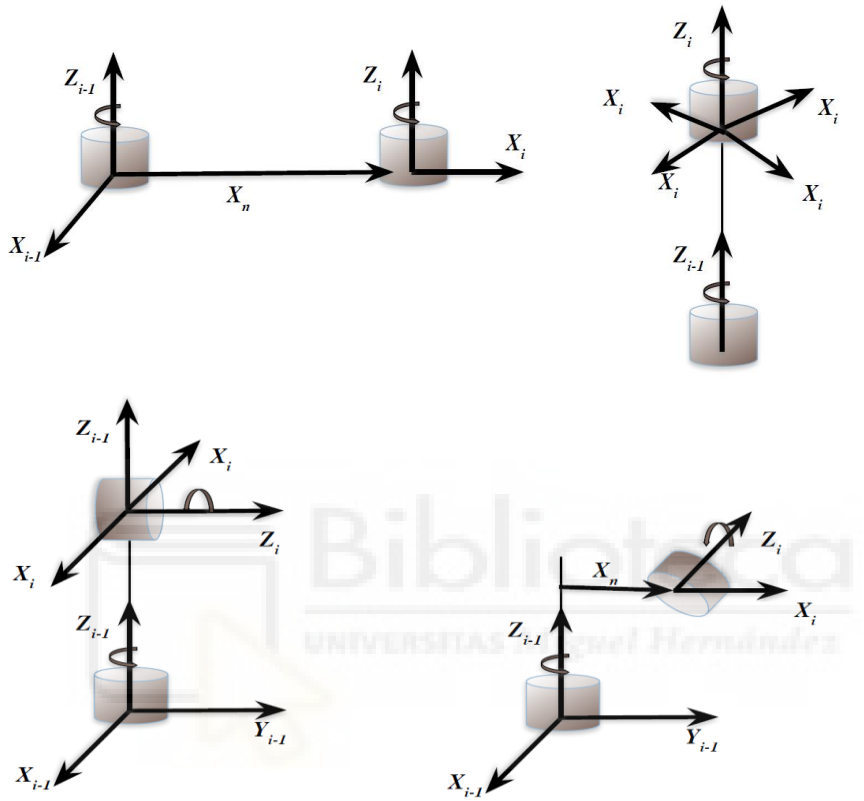


Figura 11. Colocación de Xi para D-H

Xi para D-HFigura 11

8. Por ultimo se realiza la tabla D-H con todos los parámetros que permiten ir del sistema i-1 al i. Sabremos que es correcto si al realizar la tabla podemos pasar de i-1 a i.
9. Se calculan todas las matrices ${}^{i-1}A_i$.
10. Se calcula la matriz de transformación T:

$$T = {}^{0}A_1 \, {}^{1}A_2 \ldots {}^{n-1}A_n$$

Una vez obtenida la matriz T, ya tendríamos desarrollada la cinemática directa de la aplicación de robótica. Con esa matriz es trivial calcular las coordenadas del extremo del robot (Aparicio, 2024).

2.1.2 Cinemática inversa

Una manera de obtener la cinemática inversa es encontrando las relaciones geométricas entre eslabones y las articulaciones del robot. Se podría concluir que manipulando la matriz T se podría obtener la cinemática inversa del robot, pero esta operación resulta ser muy compleja y computacionalmente intensiva para todo robot con más de tres grados de libertad. Además de que trabaja con ecuaciones que pueden tener más de una solución y por tanto nos dará múltiples soluciones.

Es por eso que para resolver la cinemática inversa se buscan relaciones matemáticas que definan la cinemática inversa, y además imponga restricciones a las múltiples soluciones (Aparicio, 2024).

A continuación, se presenta una explicación de cada método:

2.1.2.1 Método algebraico

El método algebraico, como se mencionó anteriormente, consiste en utilizar la matriz de transformación obtenida mediante D-H. Este método es más aplicable a robots de 3GDL o menos, ya que se puede obtener un sistema con más ecuaciones que incógnitas y es difícil seleccionar cuál de las ecuaciones es redundante.

2.1.2.2 Método geométrico

Se basa en el planteamiento de relaciones geométricas y trigonométricas con el fin de definir la posición del efector final con respecto a la base del robot. Se deben encontrar las suficientes relaciones como para poder resolver la cinemática directa. Este método es recomendado para robots de pocos grados de libertad, tres o menos, pero se puede usar para robots con más grados de libertad para hallar los primeros tres eslabones. Para hallar el resto de los eslabones existe el procedimiento del desacoplo cinemático que permite encontrar el resto.

2.1.2.3 Método de desacoplo cinemático

Con los métodos ya mencionados sólo es factible encontrar las coordenadas de las tres primeras articulaciones. Pero la mayoría de robots industriales son de seis o más grados de libertad. Este método se puede aplicar en robots cuyos tres últimos grados corten en

un solo punto, es decir que las tres últimas articulaciones conformen a la muñeca del robot y por tanto los ejes de las articulaciones se cruzan.

Para la implementación de este método primero se omite la muñeca del robot, y se calculan los valores de las tres primeras articulaciones. Con eso, se halla la orientación final que tendrá la muñeca. A partir de ahí se calculan los valores del resto de articulaciones. Como la orientación en donde empieza la muñeca es la misma que en donde termina, solo nos faltara calcular la posición de las tres últimas articulaciones. Con este último cálculo trivial ya tendríamos resuelto el problema de cinemática inversa.

2.1.3 Cinemática de la velocidad

Como ya habíamos comentado al principio del capítulo y mostrado en la Figura 10, hasta ahora hemos visto el comportamiento desde un punto de vista estático. Pero también vamos a necesitar abarcar la relación entre las velocidades articulares $\{\dot{q}_1, \dot{q}_2, \dot{q}_3, \dot{q}_4, \dot{q}_5, \dot{q}_6\}$ y la velocidad lineal y angular del efector final $\{\dot{x}, \dot{y}, \dot{z}, \dot{\alpha}, \dot{\varphi}, \dot{\theta}\}$. Para ello haremos uso de la matriz jacobiana, que tiene la siguiente forma para un robot con n articulaciones:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\alpha} \\ \dot{\varphi} \\ \dot{\theta} \end{bmatrix} = J(q) * \begin{bmatrix} \dot{q_1} \\ \dots \\ \dot{q_n} \end{bmatrix}$$

Ecuación 3. Relación entre las velocidades articulares y la velocidad lineal y angular del efector final para un robot de n articulaciones

J siendo la matriz Jacobiana:

$$J = \begin{bmatrix} \frac{\partial f_x}{\partial q_1} & \cdots & \frac{\partial f_x}{\partial q_n} \\ \dots & \dots & \dots \\ \frac{\partial f_d}{\partial q_1} & \dots & \frac{\partial f_d}{\partial q_n} \end{bmatrix}$$

Ecuación 4. Matriz Jacobiana

Y f_x,f_y,f_d son todas las fórmulas de cinemática directa dadas en función q, los ángulos de las articulaciones.

$x = f_x(q_1, \dots, q_n)$ $y = f_y(q_1, \dots, q_n)$ $z = f_z(q_1, \dots, q_n)$

$a = f_a(q_1, \dots, q_n)$ $b = f_b(q_1, \dots, q_n)$ $c = f_c(q_1, \dots, q_n)$ $d = f_d(q_1, \dots, q_n)$

Como se puede ver toda configuración genera una matriz jacobiana distinta. En el caso que el rango de la matriz sea un numero inferior a los grados de libertad del robot, significa que estamos en una singularidad, donde uno o más grados se libertad se pierden. Lo que se traduce a la práctica es que sería imposible variar la pose del efector final variando los ángulos de las articulaciones. Existen varios puntos que pueden ser singularidades, algunos pueden estar en el límite del área de trabajo del robot, otros pueden ocurrir porque dos o más ejes de las articulaciones se encuentran alineados.

La matriz jacobiana debe ser cuadrada para que sea invertible y poder ser usada en el problema de cinemática inversa. Si la jacobiana no es cuadrada, entonces podemos usar el método de la pseudoinversa (Aparicio, 2024).

El método de la jacobiana pseudoinversa tiene dos formas de calcularse, si el número de columnas es mayor que el de filas se usará la pseudoinversa por la derecha:

$$J_d^+ = J^T[J\,J^T]^{-1}$$

Ecuación 5. Jacobiana pseudoinversa por la derecha

Por el contrario, si tiene más filas que columnas se usará la pseudoinversa por la izquierda:

$$J_i^+ = [J^T J]^{-1} J^T$$

Ecuación 6. Jacobiana pseudoinversa por la izquierda

Para ambos casos se tiene que cumplir que:

$$J\,J_d^+ = I$$

$$J_i^+ J = I$$

2.2 Visión por computador y reconocimiento de objetos en el espacio

Estos campos se basan en una serie de teorías y técnicas que permiten a las máquinas interpretar y comprender el mundo visual. Vamos a mencionar tanto las bases teóricas detrás de la simulación de "pick and place" como las bases teóricas necesarias para trasladar la aplicación a la práctica.

2.2.1 Cámara estenopeica y captación de imágenes dentro de la simulación

La cámara pin-hole (o cámara estenopeica) es un modelo idealizado de cámara que se usa en visión por computador y gráficos 3D para representar cómo la luz entra en una cámara y se proyecta sobre un plano de imagen. Se basa en el principio de la proyección geométrica y es fundamental en la teoría de la formación de imágenes. Podemos ver una

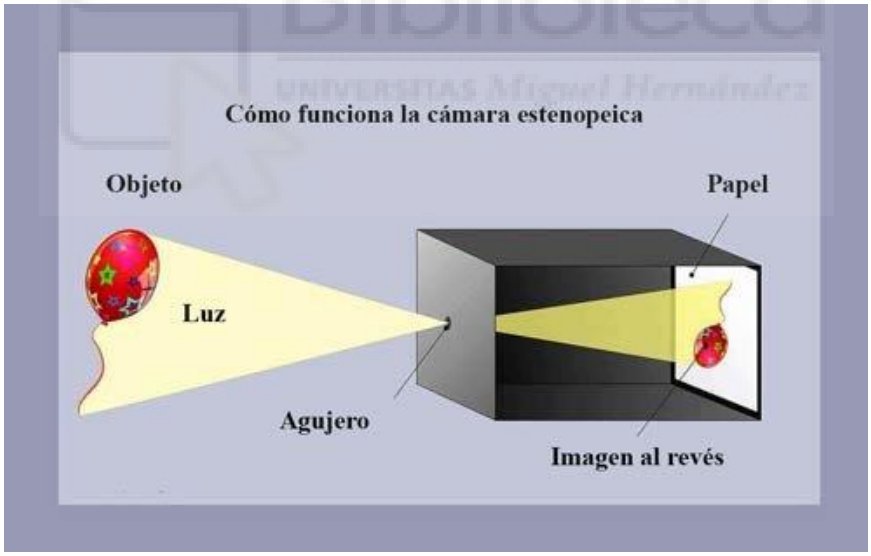


Figura 12. Representación de una cámara estenopeica

representación en la Figura 12. Dado que los sensores de visión en CoppeliaSim se comportan como una cámara pin-hole, haremos una breve explicación.

Se trata de un modelo de cámara sin lentes, por lo que no tiene enfoque ni distorsión óptica. Su modelo es una caja cerrada con un agujero pequeño que deja pasar la luz y en el lado contrario de la caja se proyecta el plano de la imagen. Esta imagen es la que captamos con OpenCV para usarla en nuestra aplicación de visión por computador.

2.2.2 Tratamiento de la señal de video RGB

En este punto veremos los principios básicos detrás de la adquisición, procesamiento y visualización de las imágenes RGB, rojo verde y azul por sus siglas en inglés.

El video es una secuencia de imágenes, también llamados fotogramas, que se presentan a una cierta velocidad, medida en cuadros por segundo

Cada fotograma es una imagen matricial RGB, donde cada píxel tiene tres valores de intensidad que representan la cantidad de rojo, verde y azul. De hecho, se podrían separar y tener tres imágenes por fotograma cada una de un color. La teoría detrás de esto es que se podrán recrear en pantalla la mayoría de colores de esta forma, juntando los valores del rojo, verde y azul.

La razón por la cual se hace así es porque está estandarizado el representar los colores de esta forma. A esto se le denomina espacio de color RGB, un espacio de color es un sistema de interpretación del color en el cual un sistema de coordenadas tridimensionales representa un tono especifico.

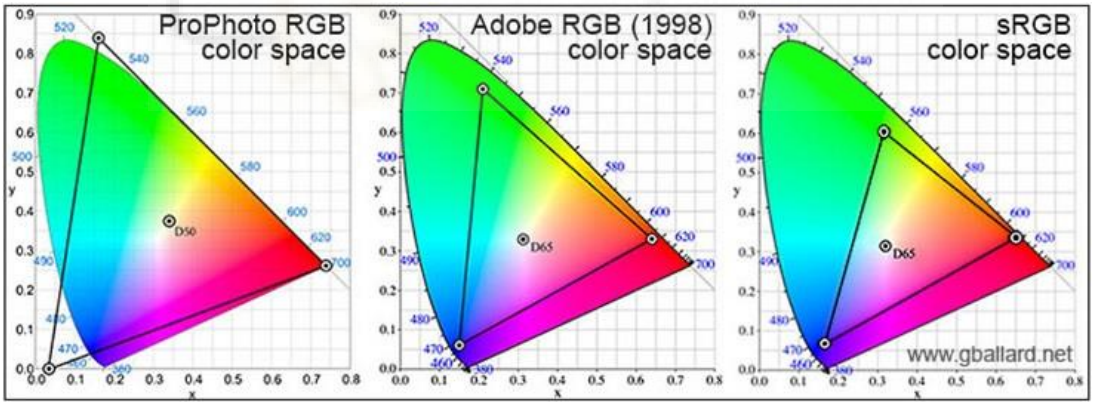


Figura 13. Distintos espacios de color RGB

El modelo RGB es el que se utiliza en todos los sistemas que forman imágenes a través de rayos luminosos ya sea emitiéndolos o recibiéndolos.
En nuestro caso, CoppeliaSim nos devuelve tres tuplas de números, pero en lugar de venir rojo, verde y azul vienen como azul, verde y rojo, por lo que tenemos que ordenarlas para poder usarla. Esto es trivial y OpenCV posee una función para ello.

2.2.3 Reconocimiento de objetos con YOLO

Una vez podemos procesar la señar usaremos YOLO, (You Only Look Once) que es un algoritmo de detección de objetos en tiempo real basado en redes neuronales profundas.

Para lograr su rapidez y notable tasa de aciertos, YOLO usa redes neuronales convolucionales basadas en regiones o R-CNN por sus siglas en ingles.

El proceso funciona de la siguiente manera: Tenemos un modelo que genera propuestas de región (estima en que parte de la imagen está el objeto), otro que extrae un vector de rasgos (en otras palabras, extrae las entidades) y por último un algoritmo de apoyo que clasifica los objetos. Podemos ver un esquema en la Figura 14. Estos tres modelos por lo general se entrenan por separado.

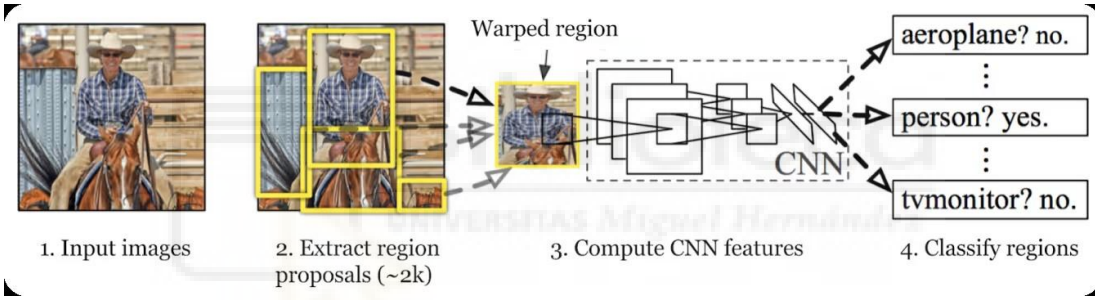


Figura 14. Esquema de funcionamiento de YOLO con sus tres modelos.

La característica esencial de YOLO, es que en lugar de mirar a una imagen varias veces, ya sea para aplicar filtros o para inspeccionar las regiones como hacen otras R-CNN, como dice su nombre, solo mira a la imagen una vez (Kundu, 2023).

2.2.3.1 Modelo de propuestas regionales

Primero se escanea la imagen y se crean propuestas de regiones. Estas propuestas son áreas que contienen potencialmente objetos. Esto se hace superponiendo una cuadricula a la imagen. En cada célula de la cuadricula se crean dos cosas:

Por un lado, una serie de cuadros delimitadores centrados en un punto dentro de la célula y asociado a ello unos valores de confianza de que un objeto exista dentro de esos cuadros delimitadores. Y por otro lado un mapa de probabilidad de clases por cada célula que clasifica que clase de objeto es más probable que este dentro de la célula, siempre y cuando el objeto efectivamente exista dentro de la célula. Para finalizar combina esta

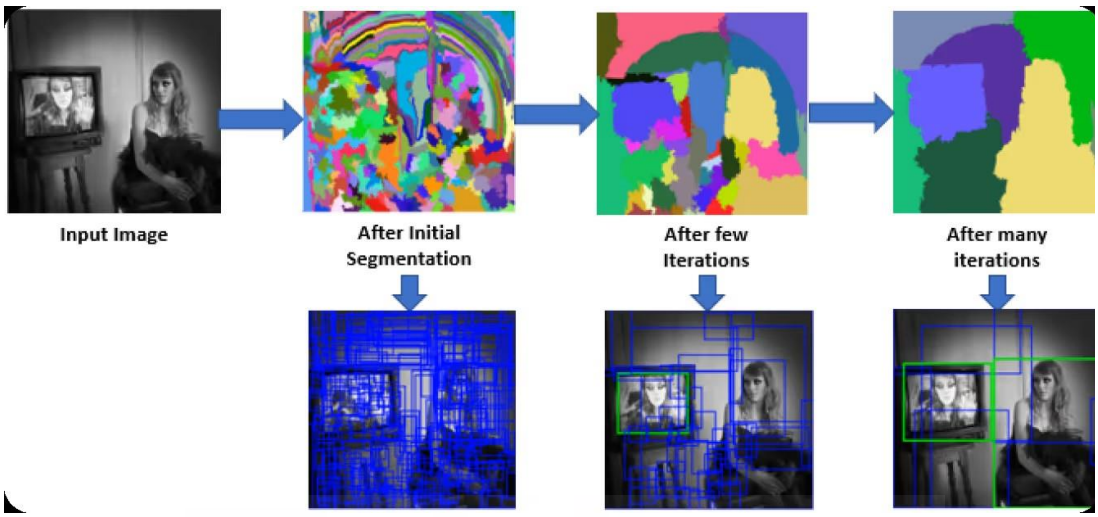


Figura 15.Esquema de funcionamiento del modelo de propuestas regionales

información para dar las detecciones de objeto. Podemos ver una representación de esto en la Figura 15.

2.2.3.2 Modelo de extracción de rasgos

En este paso, se usa una red neuronal convolucional. Esta arquitectura, consiste en un grupo de capas convolucionales que destilan la imagen en una representación más abstracta y viene seguidas de dos capas completamente conectadas que transformar esto en un vector resultado para cada célula. En la entrada tenemos tres canales que

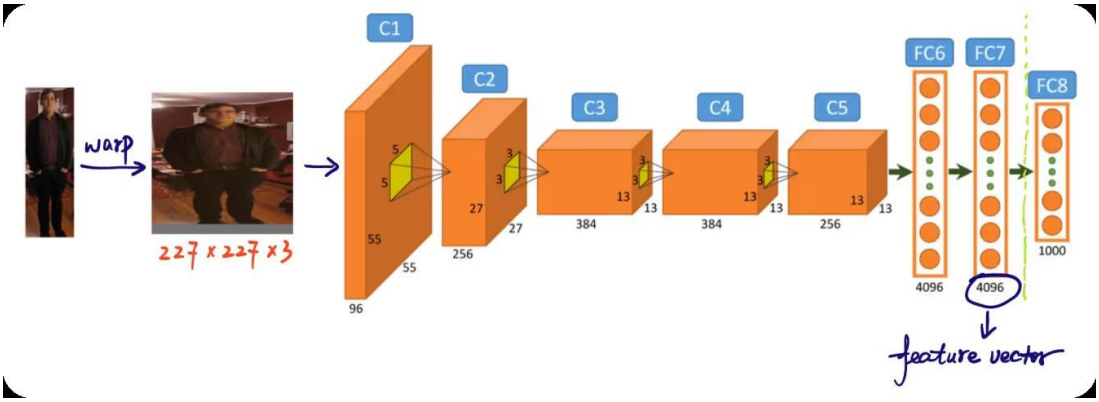


Figura 16. Arquitectura del R-CNN

corresponden a los canales RGB de la imagen. Podemos ver una representación de la estructura en la Figura 16.

Vamos a entrar más en detalle en la arquitectura, vamos a describir las capas que conforman a una red neuronal convolucional:
Las primeras capas, serían las capas convolucionales, este tipo de capas extraen las características relevantes de la imagen, mediante la aplicación de filtros de convolución sobre esta, por lo general se centran en los bordes, esquinas, colores, etc.

Las últimas dos capas se tratan de capas totalmente conectadas, su nombre viene de que cada neurona de esta capa está conectada con todas las neuronas de la capa anterior. Dara como resultado, un vector, que será una transformación lineal de las entradas multiplicadas por los pesos internos y luego se aplica una función de activación (Vina, 2024).

2.2.3.3 Modelo de apoyo clasificador de objetos

El último paso consiste en clasificar los objetos dentro de las regiones. Lo que significa determinar la clase de cada objeto. Los vectores de resultados que vienen de las ultimas capas pasan ahora por máquinas de vectores de soporte (SVM), que son unos clasificadores de aprendizaje automático. Cada SVM esta entrenada para reconocer un objeto en específico mediante el análisis del vector de características y decide si en la región determinada contiene ese objeto.

Durante el entrenamiento, los clasificadores reciben regiones etiquetados que contienen el objeto y regiones sin el objeto. Los clasificadores aprenden a distinguir entre las muestras y con esto ya tendríamos los resultados.

2.2.3.4 Refinamiento de detecciones mediante supresión no máxima

Por el proceso que hemos mencionado anteriormente ocurre que el modelo suele generar varios cuadros delimitadores para el mismo objeto centrado en células distintas. Para arreglar esto, se aplica la supresión no máxima o NMS, que refina las detecciones manteniendo solo las casillas más precisas (Ultralytics, s.f.).

La función de NMS se consigue evaluando la intersección sobre la unión entre cuadros delimitadores y sus puntuaciones de confianza asociadas. A continuación, se explican los pasos:

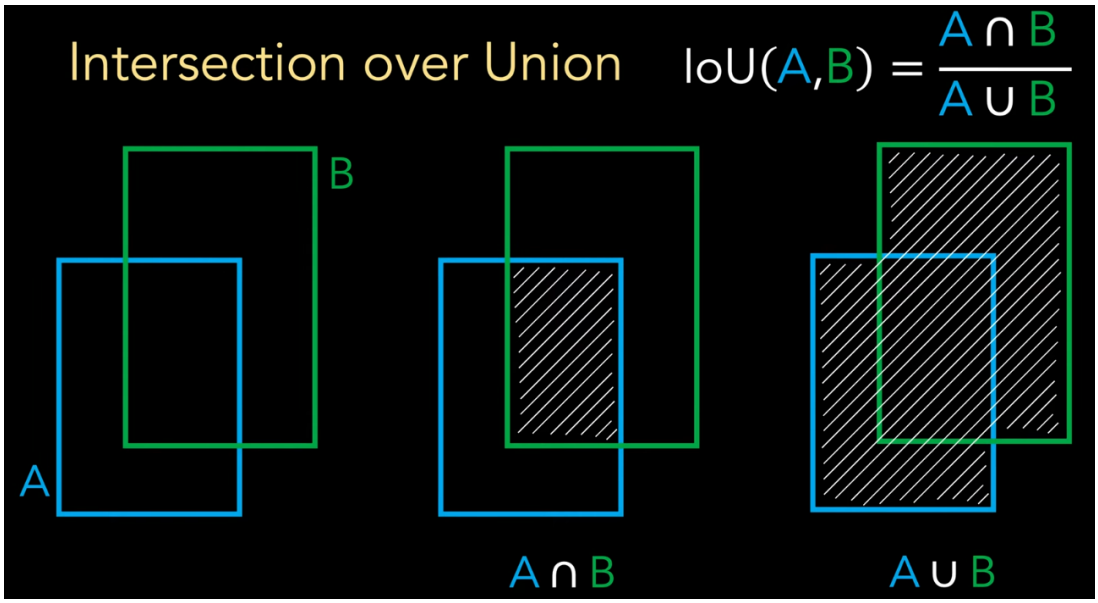


Figura 17. Diagrama que muestra el solapamiento entre dos recuadros

Primero se descartan los cuadros delimitadores con una puntuación de confianza inferior a un umbral determinado. Luego los cuadros restantes se ordenan en orden descendiente según sus puntuaciones de confianza. Solo el cuadro delimitador con la mayor puntuación es seleccionado, el resto que tengan un solapamiento significativo se eliminan, ya que se estima de que probablemente estos recuadros estén detectando el mismo objeto, podemos una representación en la Figura 17. Estos pasos se repiten hasta no queden más cuadros delimitadores que procesar y podemos ver un resultado en la Figura 18.

Figura 18. Antes y después de usar supresión no máxima

2.2.3.5 Funciones de pérdida

La función de pérdida es un elemento determinante durante el entrenamiento de cualquier red neuronal. Se trata de una función que calcula el error cometido por la red en las predicciones de objetos. Dependiendo del error, la red modifica sus pesos internos para evitar pérdidas.

El cálculo de la pérdida consiste primero en la asignación de muestras, se seleccionan las mejores predicciones en función de su puntaje de clasificación y su **IoU (Intersección sobre Unión)** entre la caja predicha y la caja real (*ground truth*). A continuación, se realiza el cálculo de la pérdida que consta de dos partes la clasificación y la regresión. La clasificación usa BCE Loss (Binary Cross Entropy) y la regresión usa Distribution Focal Loss y CloU Loss. Las tres pérdidas se combinan en una relación de pérdida especifica. A continuación, entramos más en detalle:

Binary Cross Entropy Loss

En un problema de clasificación binaria, la salida del modelo es una probabilidad ŷ entre 0 y 1. La BCE calcula qué tan diferente es esta predicción con respecto a la etiqueta real y, que solo puede ser 0 o 1.

La fórmula de Binary Cross Entropy es:

$$L = -\frac{1}{N}\sum_{i01}^{N}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$$

Ecuación 7. Formula de Binary Cross Entropy

Donde N es el número de muestras del lote.
y_i es el valor real (ground truth) que es 0 o 1.
$\hat{y}_i$ es la probabilidad predicha por el modelo.
$\log(\hat{y}_i)$ penaliza cuando la predicción es incorrecta.

En la práctica, si se trata de una clase positiva (y = 1), es decir una célula con un objeto, si la ŷ está cerca de uno, la pérdida es pequeña, por el contrario, si ŷ está cerca de cero, la pérdida es grande.
Y si en la célula no hay objeto (y = 0), es una clase negativa. Por lo que, si ŷ es próxima a cero, la pérdida es pequeña. Y si y ŷ es próxima a uno, la pérdida es grande (Godoy, 2018).

Distribution Focal Loss

Esta función se usa para mejorar la precisión en la predicción de coordenadas de las cajas delimitadoras.

Cuando un modelo predice una caja delimitadora, no siempre predice un número exacto, sino que aproxima valores continuos. Sin embargo, muchos modelos solo generan coordenadas discretas dentro de una rejilla de píxeles. DFL ayuda a mejorar la precisión de estas coordenadas al considerar distribuciones continuas en lugar de solo valores discretos.

En modelos tradicionales, la red predice directamente las coordenadas (x, y, w, h) y la altura y anchura de la caja delimitadora. Sin embargo, en el DFL, el modelo aprende una distribución de probabilidad sobre los valores posibles para cada coordenada. En lugar de predecir solo un número único para una coordenada, predice una distribución en forma de histograma, donde cada recipiente representa una posible posición de la coordenada. Luego, la media ponderada de la distribución se usa para obtener el valor final. El DFL ajusta las pérdidas en función de la confianza en cada recipiente de la distribución. Su

objetivo es dar mayor importancia a los valores más probables y reducir la penalización de errores en valores menos probables. La fórmula que usa es la siguiente:

$$L_{LDF} = \sum_{i} p_i \log(\hat{p}_i)$$

Ecuación 8. Formula de Distribution Focal Loss

En donde p_i es la distribución de la probabilidad de la coordenada real y $\hat{p}_i$ es la distribución de probabilidad predicha por el modelo.

DFL penaliza más las predicciones lejanas a la verdadera coordenada y enfatiza aquellas más cercanas (Marathe, 2020).

CloU Loss

La función Complete Intersection over Union Loss se basa en IoU (Intersección sobre unión), pero agrega términos adicionales para optimizar la convergencia del modelo. Es una función más avanzada que IoU Loss, GIoU Loss y DIoU Loss.

IoU Loss mide la intersección entre dos cajas (predicha y real), pero no considera la distancia entre sus centros ni la relación de aspecto. GIoU Loss mejora IoU al incluir un término de penalización, pero no maneja bien la convergencia en ciertas situaciones. DIoU Loss agrega la distancia entre centros, mejorando la alineación de la caja predicha. Por último CIoU Loss mejora todo lo anterior al incluir la relación de aspecto, permitiendo una convergencia más rápida y precisa. La fórmula que usa es la siguiente [Ecuación 9]:

$$L_{CLoU} = 1 - IoU + \frac{\rho^2(b, b^g)}{c^2} + \alpha v$$

Ecuación 9. Formula de la perdida de la intersección completa sobre la unión

Sus componentes son:

IoU que significa Intersection sobre Union entre la caja predicha y la real.

ρ^2 (b,b^g) es la distancia euclidiana entre los **centros** de ambas cajas.

c es la distancia diagonal máxima que abarca ambas cajas.

v es la medida de diferencia en la relación de aspecto entre ambas cajas.

α es el factor de ajuste dinámico.

Su inclusión hace que se alineen mejor los cuadros delimitadores, corrige problemas de convergencia lenta y reduce errores de cuadros muy estrechos o alargados

2.2.3.6 Entrenamiento de la red, “Deep Learning” y “Transfer Learning”

El “Deep Learning” (Aprendizaje Profundo) es un conjunto de algoritmos basados en iteraciones de transformaciones no lineales de los datos con el fin de “simular” el aprendizaje del cerebro humano, basándose para ello en la arquitectura de este que de forma muy sintetizada y simplista podemos decir que son las neuronas y las conexiones entre ellas, los axones.

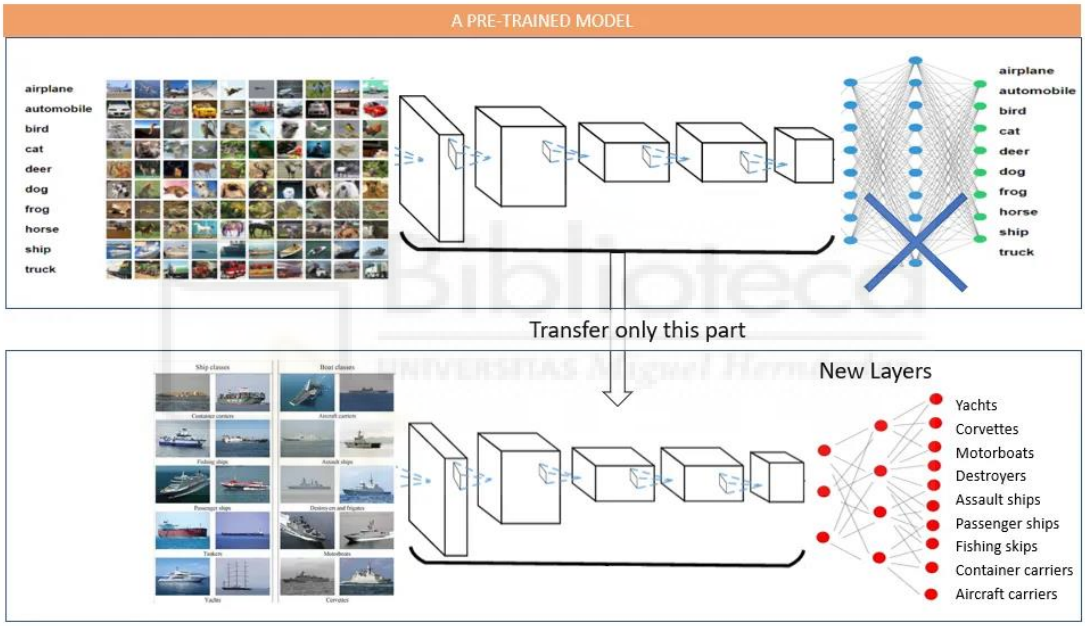


Figura 19. Esquema de Transfer Learning

En concreto las CNN, o Redes Neuronales Convolucionales, mencionadas anteriormente, se basan en realizar convoluciones aleatorias y de forma automatizada.

El “Transfer Learning” se basa en hacer uso de redes pre-entrenadas, es decir que ya han sido desarrolladas y entrenadas con grandes bases de datos y mediante hardware de gran potencia y que son de libre acceso.

En nuestro caso, hemos usado un modelo pre-entrenado de YOLO, y lo hemos reentrenado en un conjunto de datos propio para que se adapte a nuestra aplicación, lo

que hace que reutilice las características aprendidas previamente. Podemos ver un esquema de esto en la Figura 19.

2.2.4 Percepción de la profundidad y visión 3D

Como ya hemos mencionado antes CoppeliaSim usa un modelo de cámara pin-hole o estenopeica, con este tipo de cámara podemos simular digitalmente la mayoría de otros tipos de cámara. Además, tenemos una ventaja y es que CoppeliaSim nos puede dar las coordenadas en metros de cada píxel que nos pasa de una imagen captada. En este apartado vamos a tratar sobre las técnicas y métodos para que, a partir de una imagen con coordenadas de los pixeles, averiguar cuáles son las coordenadas del objeto. Hay que señalar que, en el caso práctico, la cámara obviamente no nos va a devolver la distancia como lo hace CoppeliaSim por lo que dedicaremos un apartado a hablar sobre la tecnología que podría llevar la cámara en el caso práctico y que más se parecería a lo que estamos usando.

Si bien existen muchas técnicas de recuperación de la estructura 3D de la escena, todos los métodos se basan en el principio de la triangulación espacial, que nos permite establecer una estimación de la profundidad de los objetos a partir de un conocimiento parcial de la posición relativa de los objetos respecto a la escena. Como vemos en la Figura 20, lo que nos dice este principio es que, si conocemos 3 incógnitas, ya sean

$$L1\cos(a)+L2\cos(b)=D$$

$$L1\,\mathrm{sen}(a)=L2\,\mathrm{sen}(b)$$

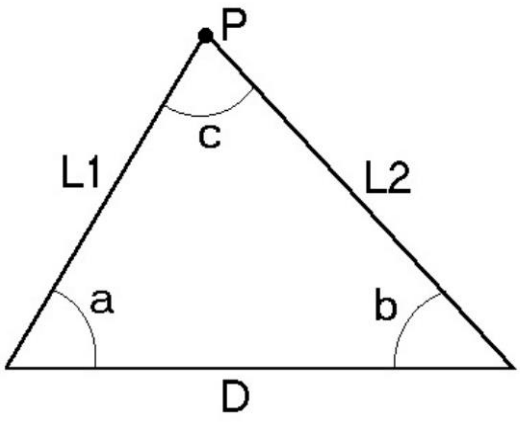


Figura 20. Principio de la triangulación espacial

ángulos, lados o combinaciones de ambas podemos averiguar el resto de los parámetros (García, 2024).

2.2.4.1 Modelo proyectivo de cámara

Como podemos ver en la Figura 21, la proyección se produce uniendo el centro óptico con el punto del espacio 3D, M con coordenadas del mundo (X_w, Y_w, Z_w), eso nos da el

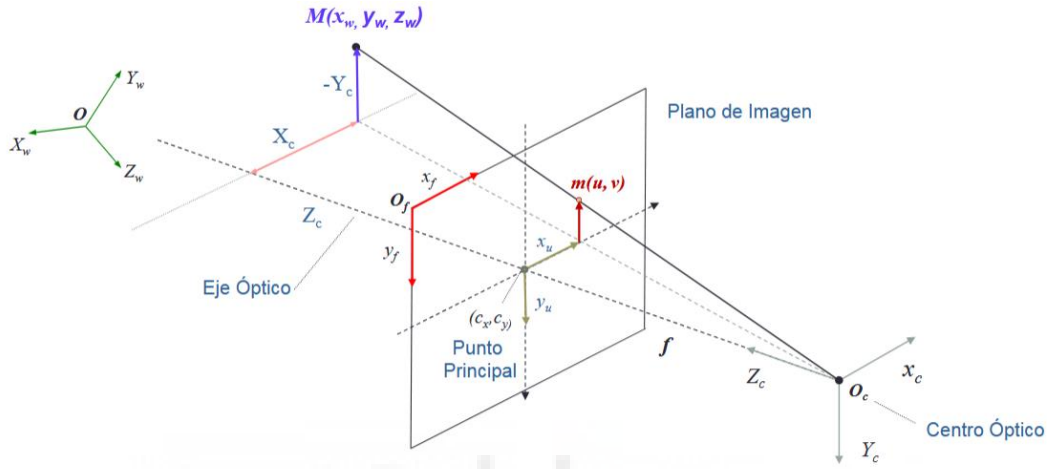


Figura 21. Modelo proyectivo de una lente pin-hole

punto proyectado m en coordenadas (u,v). Vamos a tener que trabajar con varios sistemas de coordenadas además, por un lado las coordenadas del centro óptico (Xc,Yc,Zc), las coordenadas del centro de la imagen (Cx,Cy) que tiene asociado (Xu, Yu) que son paralelas a las coordenadas del centro óptico. Adicionalmente la imagen se puede representar con otro sistema de coordenadas que empieza en la esquina superior izquierda (Xf, Yf) y son las coordenadas de frame, y nos da los datos en pixeles (García, 2024).

Aplicando el teorema de triángulos semejantes la distancia de Xc deberá ser equivalente a Xu, y lo mismo pasaría en el eje y.

$$\frac{X_u}{f} = \frac{X_c}{Z_x}$$

$$\frac{Y_u}{f} = \frac{Y_c}{Z_x}$$

Ecuación 10. Ecuaciones proyectivas para una cámara Pin-hole

Debemos trasladar esas coordenadas a ecuaciones proyectivas, ya que vamos a trabajar con coordenadas proyectivas de la imagen.

$$\begin{bmatrix} \lambda Xu \\ \lambda Yu \\ \lambda \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{f} & 0 \end{bmatrix} \begin{bmatrix} Xc \\ Yc \\ Zc \\ 1 \end{bmatrix}$$

Ecuación 11. Relación entre las coordenadas centrales de la imagen y las coordenadas de la cámara

Tenemos las coordenadas proyectivas del punto en coordenadas 3D más un 1 para el termino adicional a la izquierda. Una matriz de proyección al centro y las coordenadas proyectivas bidimensionales a la derecha. Como estamos trabajando en 2D tenemos tres coordenadas, λ siendo el factor de escala. Después vamos a multiplicar toda la ecuación por f, la distancia focal, para despejar la fracción. Como las coordenadas están definidas por un factor de escala, se mantendrán igual, pero cambiara el factor de escala, lo llamaremos n.

$$\begin{bmatrix} nXu \\ nYu \\ n \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} Xc \\ Yc \\ Zc \\ 1 \end{bmatrix}$$

Ecuación 12. Relación entre las coordenadas centrales de la imagen y las coordenadas de la cámara

La matriz de proyección Ao, que está en el centro, tiene dos componentes una matriz cuadrada y una columna de ceros. Por tanto, será la matriz cuadrada la que usaremos

$$Ao = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Ecuación 13. Matriz de proyección

Si normalizamos la distancia focal y consideramos que f es igual a 1. Las coordenadas del centro de la imagen serían así [Ecuación 14]:

$$Xn = \frac{Xc}{Zc}$$

$$Yn = \frac{Yc}{Zc}$$

$$\begin{bmatrix} nx_n \\ ny_n \\ n \end{bmatrix} = [I \quad 0] \begin{bmatrix} Xc \\ Yc \\ Zc \\ 1 \end{bmatrix} = \begin{bmatrix} Xc \\ Yc \\ Zc \end{bmatrix}$$

Ecuación 14. Coordenadas centrales para imágenes normalizadas

Se pueden usar las coordenadas normalizadas para modelar la distorsión de la lente. Pero en nuestro caso no será necesario ya que nuestra lente no tendrá distorsión. Aun así, mostraremos el planteamiento teórico detrás de esto porque podría ser usado en el caso práctico. Matemáticamente la distorsión se puede modelar de varios métodos, yo usare de ejemplo el modelo usado en la librería de OpenCV y se puede ver en la Figura 22.

Radial Tangencial Prismática

$$D_x(x_n, y_n) = k_1 r^2 x_n + k_2 r^4 x_n + k_3 r^6 x_n + 2p_1 x_n y_n + p_2(2x_n^2 + r^2) + s_1 r^2 + s_2 r^4$$
$$D_y(x_n, y_n) = k_1 r^2 y_n + k_2 r^4 y_n + k_3 r^6 y_n + 2p_2 x_n y_n + p_1(2y_n^2 + r^2) + s_1 r^2 + s_2 r^4$$

Figura 22. Modelo de distorsión de lente de OpenCV

Para hallar las coordenadas centrales de la imagen distorsionada, añadiremos la distorsión a las coordenadas del modelo [Ecuación 15].

$$x_d = x_u + f * D_x(x_n, y_n)$$
$$y_d = y_u + f * D_y(x_n, y_n)$$

Ecuación 15. Ecuaciones de las coordenadas centrales distorsión de imagen distorsionada

La relación entre las coordenadas del mundo y las de la cámara vendría dada por [Ecuación 16]:

$$\begin{bmatrix} Xc \\ Yc \\ Zc \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} = \begin{bmatrix} R_x^w & t_c^w \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ X_w \\ X_w \\ 1 \end{bmatrix}$$

Ecuación 16. Ecuación que describe la relación entre las coordenadas captadas por la cámara y las reales.

Pasamos las coordenadas de cámara a coordenadas centrales de la imagen, que es algo que hemos hecho con anterioridad, por lo que mostraré el resultado directamente [Ecuación 12]:

$$\begin{bmatrix} nXu \\ nYu \\ n \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} Xc \\ Yc \\ Zc \\ 1 \end{bmatrix}$$

Ecuación 17. Relación entre las coordenadas de la cámara y las coordenadas centrales de la imagen

A esto se le añade la distorsión de la lente. Primero calculamos las coordenadas centrales de la imagen distorsionada, y luego transformamos las coordenadas centrales en milímetros a coordenadas del "frame" en pixeles.

$$x_f = K_x x_d + C_x$$

$$y_f = K_y y_d + C_y$$

$$\begin{bmatrix} nx_f \\ ny_f \\ n \end{bmatrix} = \begin{bmatrix} K_x & 0 & C_x \\ 0 & K_y & C_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} nx_d \\ ny_d \\ n \end{bmatrix}$$

Ecuación 18. Relación entre las coordenadas centrales de la imagen y las laterales de la imagen.

Con esto tendríamos todas las ecuaciones proyectivas, y podemos crear la relación matricial entre las coordenadas del mundo y las coordenadas laterales de la cámara.

$$\begin{bmatrix} nx_f \\ ny_f \\ n \end{bmatrix} = \begin{bmatrix} K_x f & 0 & C_x \\ 0 & K_y f & C_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} X_w \\ X_w \\ 0 \\ 1 \end{bmatrix}$$

Ecuación 19. Relación entre las coordenadas del mundo y las laterales de la imagen.

Esta ecuación nos permite pasar de coordenadas de la imagen a coordenadas del mundo de forma proyectiva.

Simplificando la ecuación nos quedaríamos con:

$$\begin{bmatrix} nx_f \\ ny_f \\ n \end{bmatrix} = \begin{bmatrix} K_x f r_1 + C_x r_3 & K_x f t_x + C_x t_z \\ K_y f r_2 + C_y r_3 & K_y f t_y + C_y t_z \\ r_3 & t_z \end{bmatrix} \begin{bmatrix} X_w \\ X_w \\ X_w \\ 1 \end{bmatrix}$$

Ecuación 20. Relación simplificada entre las coordenadas del mundo y las laterales de la imagen

Ahora añadimos la distorsión al modelo. Si tenemos q, que son las coordenadas 2D normalizadas sin distorsión:

$$q = A^{-1} * m_u = \begin{bmatrix} x_n \\ y_n \\ 1 \end{bmatrix}$$

Ecuación 21. Coordenadas centrales de la imagen normalizadas.

Siendo las ecuaciones de distorsión la Figura 23:

$$x_n^d = x_n(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + 2p_1 x_n y_n + p_2(2x_n^2 + r^2) + s_1 r^2 + s_2 r^4$$
$$y_n^d = y_n\ (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + 2p_2 x_n y_n + p_1(2y_n^2 + r^2) + s_1 r^2 + s_2 r^4$$
$$r^2 = x_n^2 + y_n^2$$

Figura 23. Ecuaciones de distorsión

Nos daría unas coordenadas 2D normalizadas con distorsión tal que:

$$q^d = \begin{bmatrix} x_n^d \\ y_n^d \\ 1 \end{bmatrix}$$

Ecuación 22. Coordenadas 2D normalizadas

Y con esto conseguiríamos la relación con las coordenadas 2D distorsionadas (García, 2024):

$$m = A * q^d$$

Ecuación 23. Relación con las coordenadas 2D distorsionadas

2.2.4.2 Active IR Stereo

El Active IR Stereo es una técnica de visión por computadora que utiliza luz infrarroja (IR) y estereovisión para generar mapas de profundidad en 3D.

Se usa un proyector de luz infrarroja para emitir un patrón sobre la escena, podemos ver ejemplos en la fig. Por otro lado, se usan dos cámaras infrarrojas separadas a una distancia fija. Cada cámara captura la imagen de la escena desde un ángulo distinto, y como el patrón proyectado es conocido, se pueden detectar las diferencias en la deformación del patrón entre ambas imágenes. Se usa el principio de disparidad estéreo para medir la diferencia entre las imágenes captadas por ambas cámaras y con esto se genera un mapa de profundidad donde cada píxel indica cómo de lejos está cada punto del sensor. Una

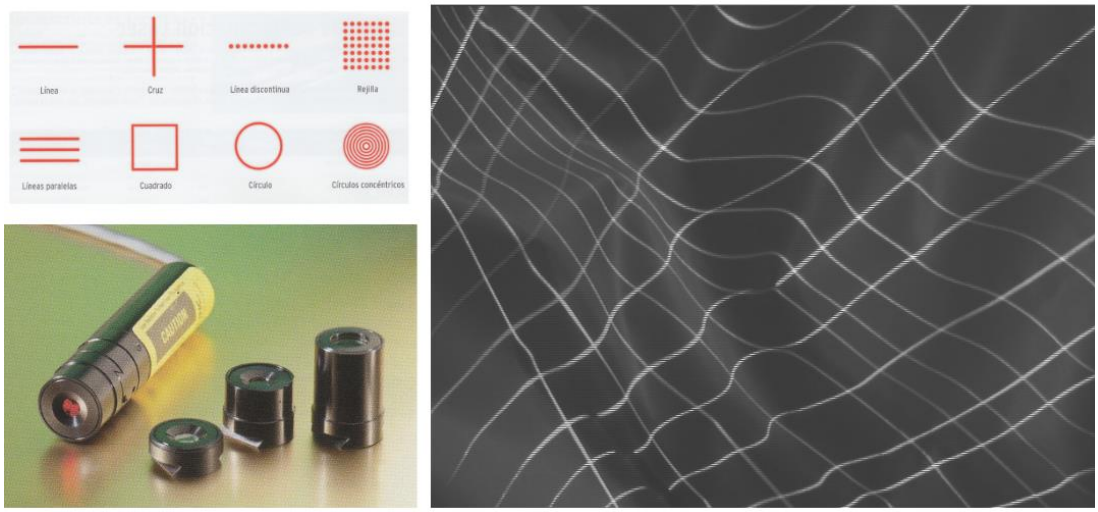


Figura 24. En la imagen podemos ver distintos patrones de proyección, un emisor laser y una rejilla de luz estructurada sobre una superficie irregular

posible ventaja de este sistema es que al estar la cámara situada en el efector final del

robot podríamos recibir estimaciones más robustas al realizar barridos cuando movemos la cámara hacia el objeto para agarrarlo.

2.2.5 Ordenamiento de puntos en el espacio RANSAC

Como ya hemos comentado anteriormente la simulación nos devuelve una imagen en donde cada píxel viene asociado un valor en metros de distancia. Como sabemos la posición del objeto en coordenadas del mundo, se podría decir que lo que tenemos es una nube de puntos, similar a como funcionaria un LIDAR. Para sacar datos útiles de esta nube de puntos primero cribamos los datos, primero solo trabajaremos con los puntos que están dentro del cuadro delimitador del objeto, luego eliminamos lo que consideramos datos que pueden provocar errores en el cálculo como vendría a ser puntos de la imagen que si bien están dentro del cuadro limitador están en otra parte de la escena. Para ello suponemos que todo lo que esté excesivamente cerca o excesivamente lejos no forma parte del objeto. A continuación, ya tenemos lo que consideramos datos útiles y nos aseguramos de que tengamos los suficientes datos para que funcione el algoritmo.

RANSAC o RANdom SAmple Consensous es un algoritmo utilizado para ajustar modelos a datos con ruido o valores atípicos (Derpanis, 2010). Es ampliamente usado en visión por computadora para tareas como detección de líneas, estimación de homografías y correspondencias de puntos en imágenes. Nosotros lo vamos a usar para detección de líneas, vamos a suponer que el objeto que vamos a intentar agarrar es un objeto rectangular. A continuación, explicaremos el funcionamiento de RANSAC:

Selecciona un subconjunto de datos, los puntos, s de forma aleatoria, como indica el nombre, del conjunto de datos total S y calcula el modelo de este subconjunto. Después determina el subconjunto de puntos S_i que están dentro de un umbral t de distancia al modelo, este subconjunto de datos S_i es el conjunto consenso y define los “inliers” de S. Si el conjunto S_i es mayor que un umbral T, vuelve a estimar el modelo usando todos los puntos de S_i y termina. Si el tamaño de S_i es menor que T, selecciona un nuevo subconjunto y repite el paso anterior. Después de un número de intentos N selecciona el subconjunto S_i con mayor consenso, y vuelve a estimar el modelo utilizando todos los puntos del subconjunto S_i. De forma simplificada podemos verlo en la Figura 25.

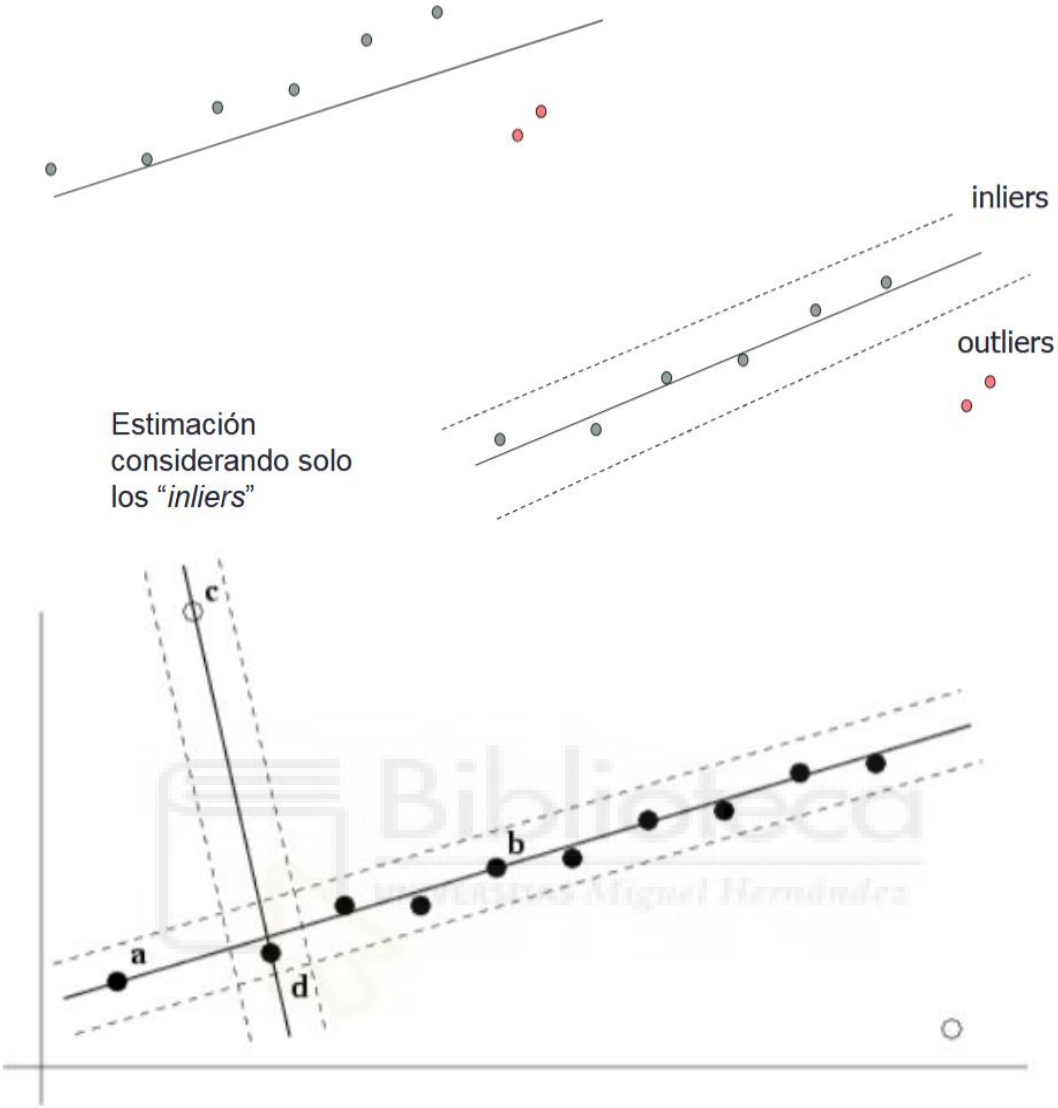


Figura 25. Ejemplo de estimación de una recta. Podemos ver que arriba, la recta no queda bien centrada por los "outliers". Abajo con los "outliers" siendo excluidos se consigue una mayor precisión.

Para escoger el tamaño aceptable del conjunto como el número de "inliers" estimado en el conjunto total, se usa:

$$T = (1 - e) * n$$

Ecuación 24. Ecuación que calcula el tamaño aceptable del conjunto

Siendo T el número de puntos que deben cumplir el consenso para finalizar el algoritmo, e siendo la probabilidad de errores "outliers" y n el tamaño del conjunto total S.

Por otra parte, el valor de e se puede calcular de forma adaptativa en cada iteración del algoritmo:

$$e^{(i)} = 1 - \frac{size(S_i)}{n}$$

Ecuación 25. Ecuación para calcular la probabilidad de errores

Para escoger N, si tenemos la probabilidad p de que el algoritmo solo seleccione "inliers". Queremos que al menos una muestra aleatoria esté libre de "outliers".

Siendo e la probabilidad de errores.

La probabilidad de que exista algún "outlier" en s: $1 - (1 - e)^s$

La probabilidad de que exista algún "outlier" en s en N iteraciones: 1-p

$$(1 - (1 - e)^s)^N = \ 1 - p$$
$$N = {\log(1 - p)}/{\log(1 - (1 - e)^s)}$$

Ecuación 26. Ecuación habiendo despejado en ambas partes

3. Entorno de simulación y herramientas empleadas

Dedicare este capítulo principalmente para dos cosas, explicaré el material y los métodos usados dentro de la simulación, incluido el propio simulador, pero, también expondré como lo planteado en la simulación podría realizarse en la vida real, y pondré ejemplos de materiales y métodos para lograrlo.

3.1 Entorno de simulación y software

3.1.1 CoppeliaSim

CoppeliaSim es un simulador de robótica desarrollado actualmente por **Coppelia Robotics** que permite la simulación de entornos robóticos en tiempo real. Es ampliamente utilizado en investigación, educación y desarrollo de aplicaciones en robótica. Podemos verlo en la Figura 26.

Se usa principalmente para el diseño y puesta a prueba de algoritmos de control de robots. La simulación de todo tipo de robots, tales como drones, brazos robóticos, vehículos autónomos, entre otros. Y la validación de sistemas antes de hacer pruebas en el hardware real.

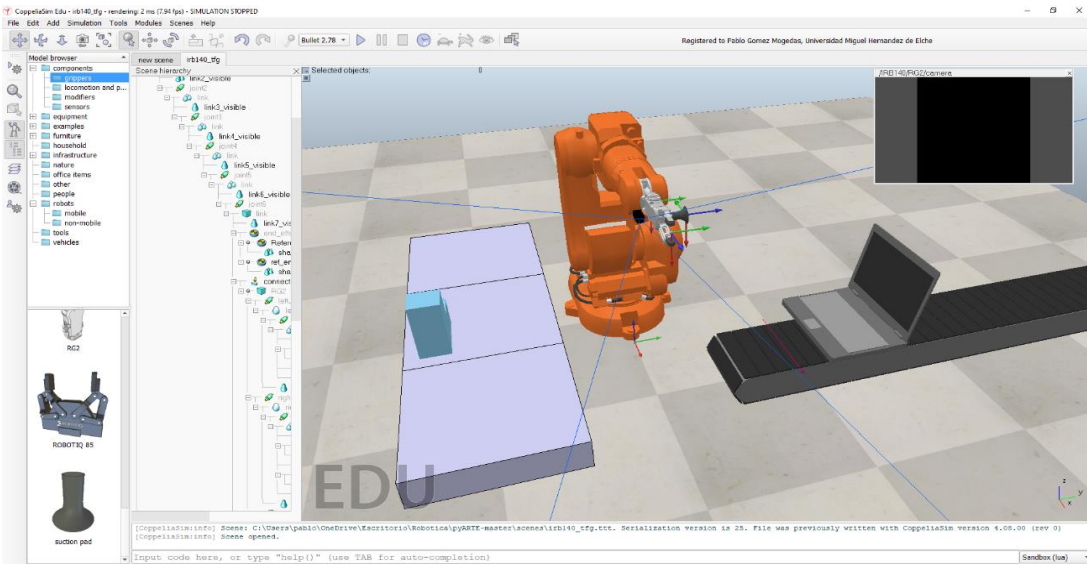


Figura 26. Entorno de simulación de CoppeliaSim

Se caracteriza por su motor de física avanzado el cual es compatible con varias bibliotecas de simulación física como Bullet, ODE, Vortex, Newton y MuJoCo. Su arquitectura lo

hace muy versátil ya que soporta varios lenguajes de programación como Python, Lua, C++, Java y MATLAB, a través de unos controladores asíncronos, en nuestro caso usaremos ZeroMQ para conectarlo a nuestro proyecto de Python en el que estaremos usando la librería PyARTE.

Hemos elegido este software de simulación por nuestra experiencia anterior con este programa en la asignatura de Robótica, además dispone de una versión para estudiantes completamente gratuita que mantiene la mayoría de las funcionalidades de la versión de pago.

A continuación, entraremos más en detalle con los distintos motores físicos y gráficos de simulación.

3.1.1.1 Motores gráficos de CoppeliaSim

CoppeliaSim usa dos motores gráficos para renderizar el entorno 3D, que son:

OpenGL

Este es el motor gráfico base de CoppeliaSim, se usa para la visualización 3D en tiempo real de la escena, la iluminación, sombreado, materiales y texturas y para la vista y generación de imágenes de cámaras dentro de la simulación.
OpenGL, u Open Graphics Library, como podemos ver en el logotipo de la Figura 27, es una API grafica multiplataforma (Windows, Linux, macOS) para renderizado 2D y 3D. Y es una de las más usadas en el desarrollo de videojuegos, simuladores, software CAD, etc.

Fue creada en 1992 por Silicon Graphics Inc. Y se basa en una arquitectura cliente-servidor, diseñada para ser ejecutada mediante una GPU, aunque es posible implementar software que funcione en una CPU.

La API, es un documento que describe una serie de funciones y el comportamiento que deben tener. El usuario, o programa define los vértices, colores, texturas, iluminación, y demás, y OpenGL se encarga de renderizar los objetos en la pantalla. Además, las funciones son independientes del lenguaje lo que lo hace compatible con multitud de lenguajes de programación y sistemas operativos.

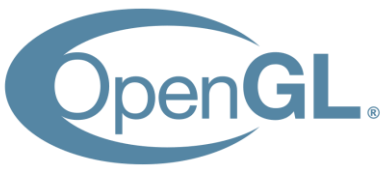


Figura 27. Logotipo de OpenGL

POV-Ray

El Persistance of Vision Raytracer, cuyo logo podemos ver en la Figura 28, es un motor de renderizado para obtener imágenes de alta calidad. Utiliza la tecnología de trazado de rayos, o ray tracing en inglés, y genera sombras suaves, reflejos y transparencias realistas, entre otras cosas. Si bien no se usa en tiempo real, se puede activar para sacar imágenes realistas desde las cámaras virtuales de CoppeliaSim.

El funcionamiento del trazado de rayos consiste en proyectar un rayo desde cada píxel de la cámara. Cada rayo intersecta con objetos y el motor calcula, los rebotes para conseguir el reflejo de la luz, como atraviesa el objeto para las transparencias y refracciones, y como se ilumina, para el cálculo de sombras y colores. Estos cálculos generan efectos de luz muy realistas pero muy intensivos de calcular computacionalmente.

Figura 28. Logotipo de POV-Ray. Podemos observar distintos efectos que podemos conseguir con el motor, como refracción de la luz y reflejos

3.1.1.2 Motores físicos de CoppeliaSim

CoppeliaSim se destaca por su capacidad para trabajar con diferentes motores de simulación física, pudiendo cambiar de una a otro en cualquier momento, dependiendo de las necesidades del usuario. Como la simulación de las leyes de la física es compleja y computacionalmente exigente, se puede seleccionar el motor que más se ajuste a los requerimientos de la simulación, Los resultados dependerán de los algoritmos y rutinas de cálculo, además de las propiedades del motor físico y los parámetros usados como podemos ver en la Figura 30. Las interactuaciones entre los objetos simulados son

similares a las interacciones que pueden ocurrir en la vida real como podemos ver en la Figura 29. Hay que tener en cuenta, sin embargo, que CoppeliaSim no es un simulador de físicas puro y que funciona más como un simulador hibrido que combina cinemática y dinámica para obtener mejores rendimientos a la hora de simular la escena (Coppelia Robotics, s.f.).

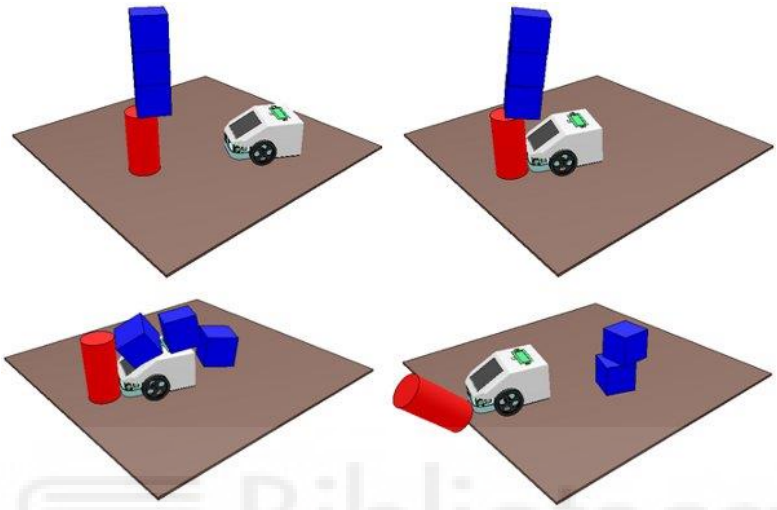


Figura 29.Distintas interactuaciones dinámicas entre objetos simulados.

A continuación, entramos en más detalle con las distintas opciones de motor físico.

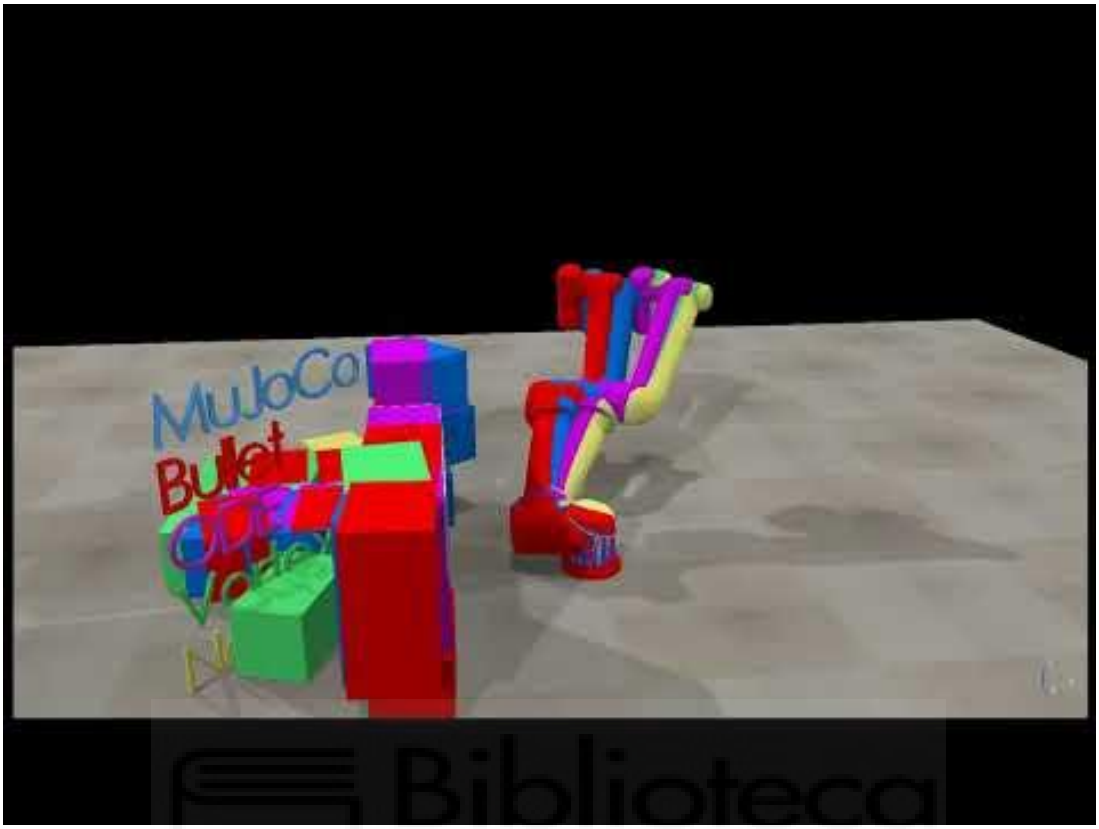


Figura 30. Video que muestra los distintos resultados entre los distintos motores físicos, para la misma escena. Enlace; https://www.youtube.com/watch?v=Y0c35pk7T9M

Bullet physics library

Es un motor de código libre usado principalmente en videojuegos, pero también en efectos visuales de cine. Soporta la detección de colisiones complejas de objetos 3D, dinámica de cuerpos rígidos y blandos, articulaciones y fricción. Muchas veces para facilitar los calcules hay que mantener los valores de la masa y la inercia entre articulaciones conectadas, relativamente bajas. Posee una buena relación entre precisión y rendimiento. Podemos ver su logo en la Figura 31

Figura 31. Logotipo de Bullet physics library.

Open Dynamics Engine (ODE)

Es uno de los primeros motores implementados en CoppeliaSim, y se usa mayoritariamente en videojuegos, pero también en otras aplicaciones. Es sencillo y

eficiente, pero es menos preciso que Bullet. Posee la capacidad de detectar colisiones y dinámicas de cuerpo rígido. Al igual que el Bullet, es recomendable mantener los valores de la masa y la inercia entre articulaciones conectadas, relativamente bajas.
Su logotipo se muestra en la Figura 32

Figura 32. Logotipo de ODE

MuJoCo

Es un motor de física de última generación de código abierto. Tiene una muy alta precisión dinámica, con soporte para contactos suaves como se muestra en la Figura 34 y

Figura 35.Logotipo de MuJoCo

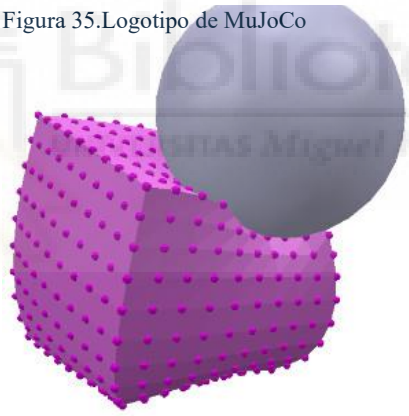


Figura 34. Ejemplo de simulación de cuerpo blando

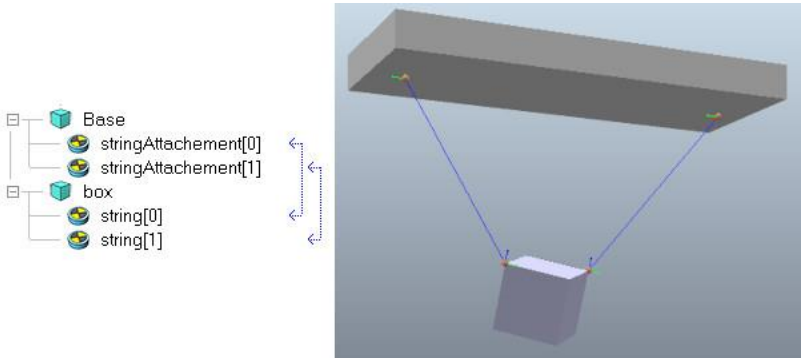


Figura 33. Cuerdas en tensión simuladas por el motor físico MuJoCo

cuerdas, hilos elásticos y tendones como se puede apreciar en la Figura 33. Además, es capaz de simular con mucho realismo articulaciones complejas motorizadas. Es muy usado en simulaciones de control motor y aprendizaje por refuerzo. Su logotipo lo podemos ver en la Figura 35

Vortex Studio

Es un motor comercial, de alta precisión. Es capaz de producir simulaciones físicas de alta fidelidad y ofrece parámetros realistas para una gran cantidad de propiedades físicas, lo que hace que este sea realista y preciso. Además, maneja con mucha estabilidad contactos múltiples, fricción, y grandes masas. Se usa sobre todo para simulaciones industriales de alta precisión y aplicaciones de investigación y desarrollo.
Su mayor inconveniente es que requiere de una licencia externa. En la Figura 36 esta su logotipo.

Figura 36. Logotipo de Vortex Studio

Newton Dynamics

Es una biblioteca de simulación física realista. Implementa un sistema de resolución determinista que consigue físicas muy realistas con simulaciones suaves, y un muy buen comportamiento simulando cuerpos rígidos con formas complejas. Esto hace que sea usado no solo para videojuegos sino también para aplicaciones de simulación física en tiempo real. Su logotipo se muestra en la Figura 37

Figura 37.Logotipo de Newton Dynamics

3.1.1.3 Elementos de CoppeliaSim

Una vez explicados los detalles de los distintos motores gráficos que hacen funcionar la simulación vamos a comentar los distintos elementos que contiene la simulación, así podremos comprender en más detalle el programa.

La columna vertebral de CoppeliaSim son las escenas y los modelos. El modelo siendo un subelemento de la escena. Ambos contienen objetos de escena y scripts de simulación, los cuales explicaremos más adelante. Para entender mejor todos los elementos empezaremos explicando la jerarquía de escena la cual vamos a mencionar varias veces a continuación.

Jerarquía de escena

La jerarquía de escena muestra todo el contenido de una escena. Como los objetos de la escena están construidos en una estructura jerárquica, los elementos se muestran como un árbol en el que se pueden expandir y colapsar sus subelementos y se permite ver la relación entre los distintos elementos de la escena (Coppelia Robotics, s.f.). Se puede observar en detalle en la Figura 38.

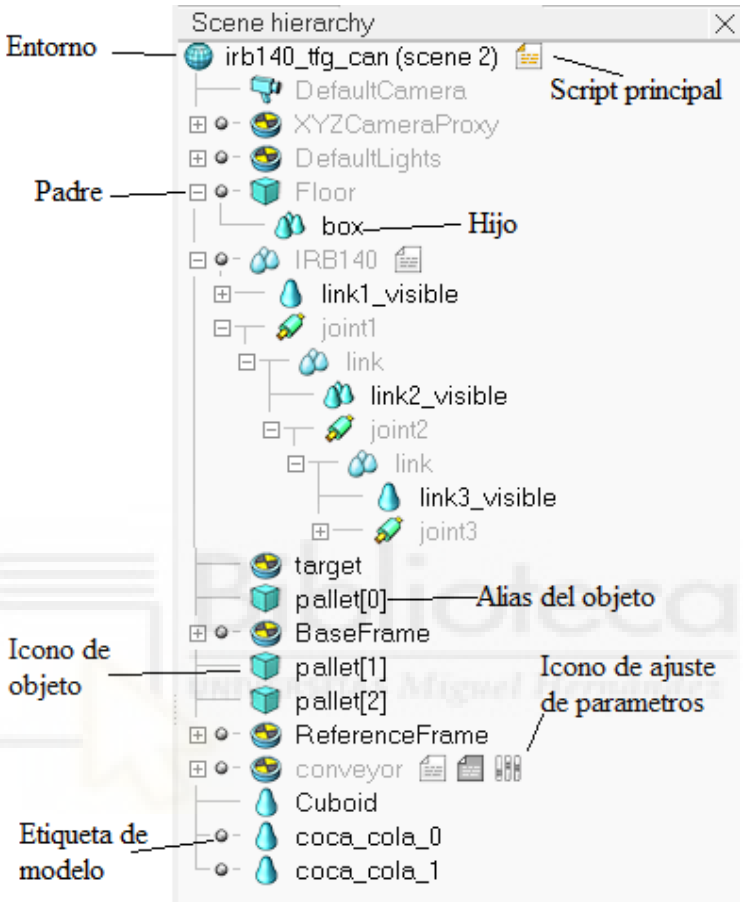


Figura 38. Jerarquía de escena de CoppeliaSim con explicación de los iconos.

Escena

La escena es el entorno virtual completo en el que se lleva a cabo la simulación. Pueden ser tan simples o complejas como la simulación lo requiera, por ejemplo, en la Figura 39 vemos una escena con mucha cantidad de modelos, que simula una vivienda, para programar el recorrido de un robot móvil. Las escenas se guardan en un archivo de tipo ".ttt" y engloban los objetos y los parámetros usados, de forma que cuando se exporta se recrea la misma simulación, con el mismo contenido (Coppelia Robotics, s.f.).

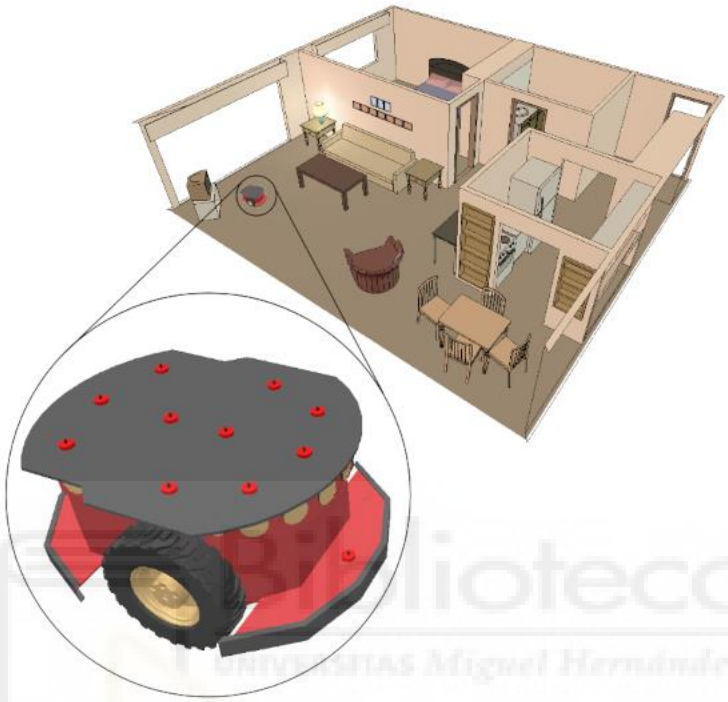


Figura 39. Ejemplo de escena compleja propuesta por CoppeliaSim.

A parte de los elementos que comparte con los modelos, la escena incluye elementos específicos como son:

El entorno, define las propiedades y parámetros que son parte de la escena, pero no son objetos de la escena. Entrando en detalle, se trata de los colores de fondo, los parámetros de niebla que interactúan con las cámaras y los sensores de visión, la luz ambiental, la información que crea la escena y cualquier ajuste adicional.

El script principal, contiene el código básico que permite que se realice la simulación, sin esto la escena permanecería estática. El script principal por defecto está dividido en 4 funciones de devolución de llamada o "callback functions" en inglés: la función de inicialización "sysCall_init", la función de actuación "sysCall_actuation", la función de detección "sysCall_sensing" y la función de restauración "sysCall_cleanup" (Coppelia Robotics, s.f.).

Hay muchas más funciones de devolución de llamada que el script principal puede usar para reaccionar a diversos eventos.

Si se desea modificar el script principal se puede acceder a este pulsando el icono de la Figura 40 en la jerarquía de escena.

Figura 40.Icono del script principal

La Figura 41 muestra lo que sucede en el script principal por defecto cuando un robot móvil con un sensor de proximidad es simulado y se encuentra con un obstáculo.

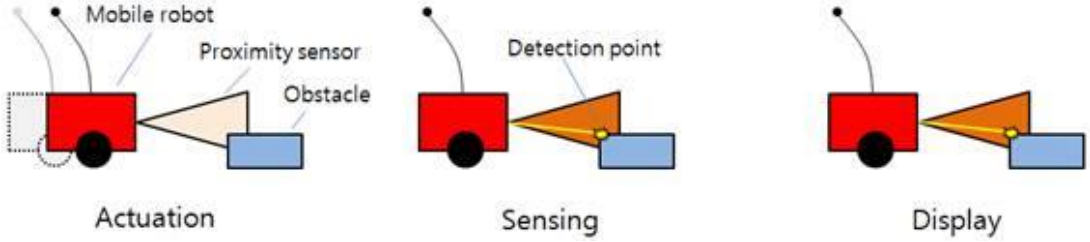


Figura 41.Ejemplo de las distintas fases del script principal y su funcionamiento de ejemplo.

Las páginas y vistas. En CoppeliaSim una página es la superficie de visualización principal de la escena, puede contener cuantas vistas sean necesarias, incluidas todas las vistas asociadas a objetos de tipo cámara y sensores de visión. Una vista es lo que se usa para mostrar el contenido visual de un objeto especifico, y tiene que ser un objeto visible. Por ejemplo, si la vista se asocia a un objeto de tipo cámara, esta mostrara lo que ve la cámara.

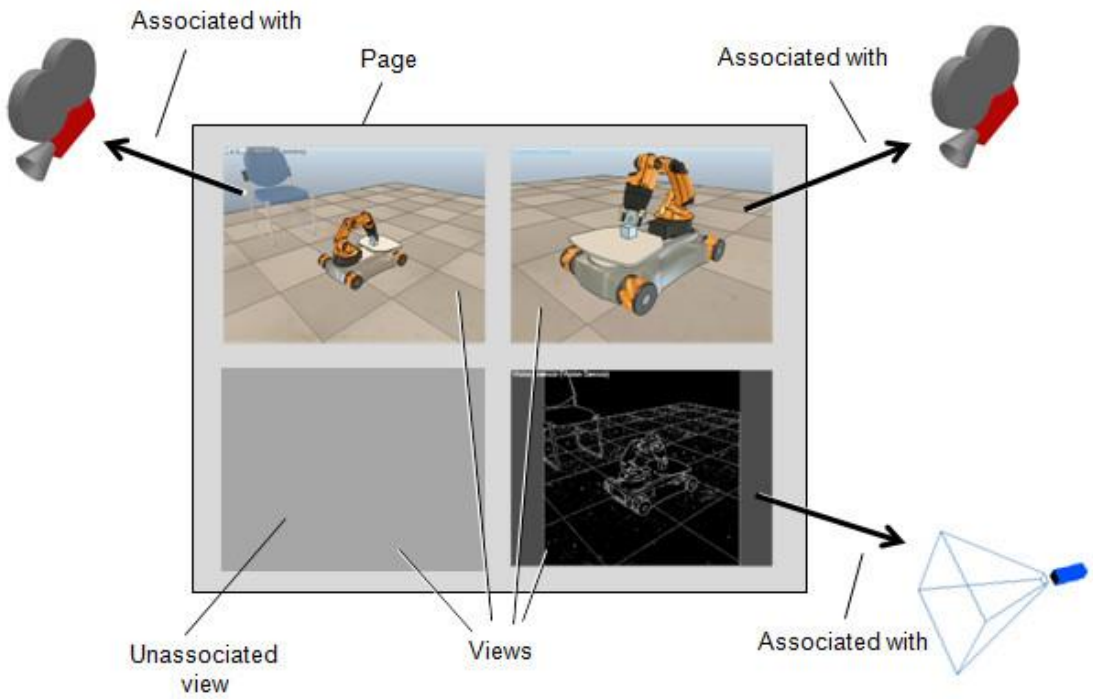


Figura 42. Las relaciones entre las páginas, las vistas y los objetos visibles.

En la Figura 42 podemos ver la relación entre página, vista y objetos visibles. Y en la

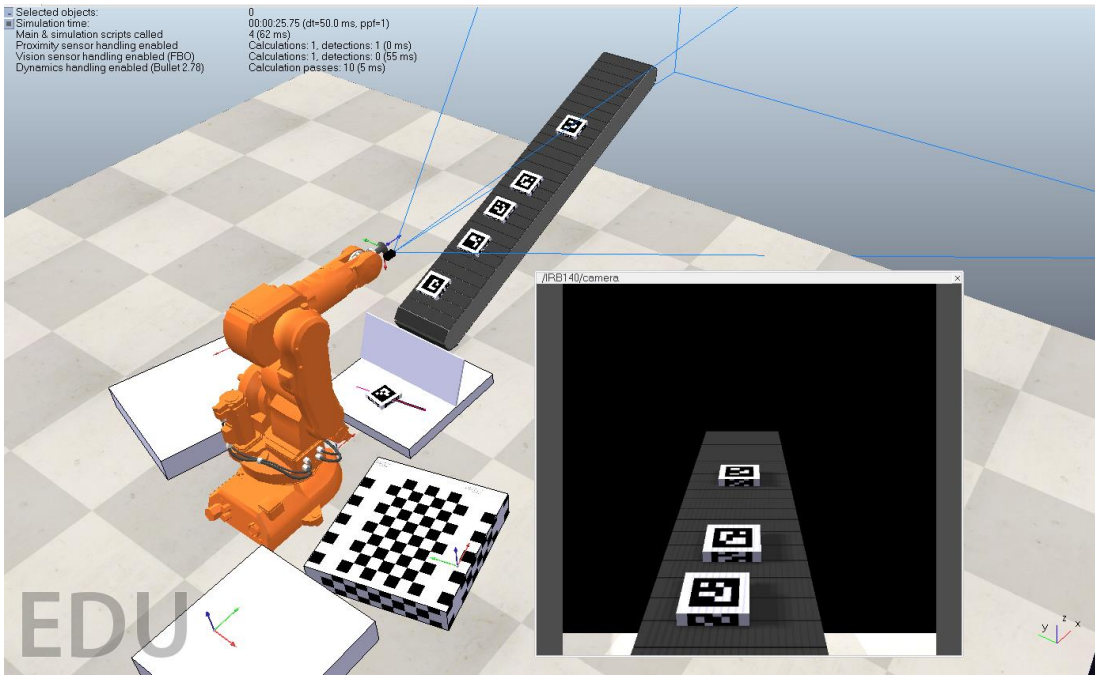


Figura 43. Vista de un sensor de visión en CoppeliaSim

Figura 43 un ejemplo de uso, la vista de un sensor de visión.

Modelos

Como hemos mencionado antes un modelo es un subelemento de la escena. El modelo no puede simularse solo, siempre tiene que estar contenido dentro de una escena para poder ser simulado. Un modelo sé puede guardar por separado si así se desea, en un archivo de tipo ".ttm" para ser importado a otras escenas (Coppelia Robotics, s.f.).

Los modelos están conformados por un conjunto de objetos de escena organizados jerárquicamente, como podemos ver en la Figura 45. Un objeto padre, puede acceder a funcionalidades que tienen sus hijos como por ejemplo una pinza que este conectada a un brazo robótico. Por otro lado, para asegurar el correcto funcionamiento de un brazo robótico es necesario crear una cadena cinemática usando la estructura de eslabón – articulación, sin esto el cálculo de la cinemática inversa no funcionaria, pues el programa no sabría cuáles son las relaciones entre los distintos eslabones del robot.

Las propiedades del modelo pueden ser ajustadas en la ventana de propiedades del modelo que podemos ver en la Figura 44. Y nos permite activar y desactivar propiedades como que el modelo sea detectable, medible, visible, etc.

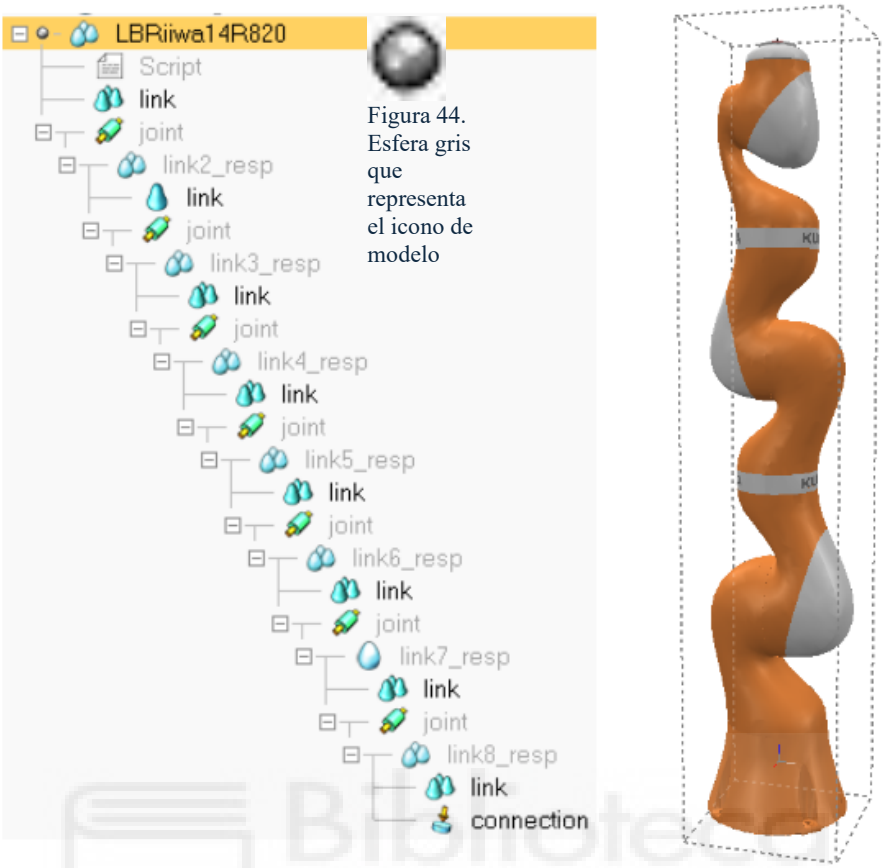


Figura 44. Esfera gris que representa el icono de modelo

Figura 45.Ejemplo de modelo en CoppeliaSim. En este caso de un robot, al que le podemos apreciar su jerarquía de modelos y su modelo 3D completo.

Objetos de escena

Los elementos principales para la construcción de una simulación en CoppeliaSim son los objetos de escena. Son objetos visibles en la jerarquía de escena y en la vista de escena. Algunos objetos tienen una representación 3D ligada a ellos como podemos ver en la Figura 46 (Coppelia Robotics, s.f.).

Figura 46. Tipos de objetos de escena en CoppeliaSim y su representación tridimensional.

Entraremos más en detalle con algunos tipos de objetos que hemos utilizado, pero la descripción general es:

Formas: Son unas mallas rígidas formadas por caras triangulares

Articulaciones: Una articulación es un objeto que permite de uno a tres grados de libertad entre objetos conectados. Puede funcionar de manera pasiva o de manera activa como un actuador.

Sensores de proximidad: Un sensor de proximidad puede detectar otros objetos de escena que estén dentro de su volumen de detección.

Sensores de visión: Son un tipo de sensor que funcionan de forma similar a una cámara. Captan colores, imágenes y reaccionan con la luz.

Sensores de Fuerza: Un sensor de fuerza es un objeto capaz de medir las fuerzas y pares que se aplican sobre este.

Árbol octal: Es una estructura de datos de partición espacial formada por vóxeles.

Script de objetos: Es un script embebido en un objeto de escena como un script de simulación o personalización.

Cámara: Una cámara es un objeto permite ver la simulación desde distintos puntos del entorno.

Luces: Las luces son objetos que permiten iluminar la escena.

Maniquí: un maniquí (no en el sentido tradicional) es un punto con una orientación que pueden tener múltiples propósitos como por ejemplo servir de punto de referencia.

Nube de puntos: Una nube de puntos es una estructura de árbol octal que contiene puntos.

Gráfica: Es usada para guardar y visualizar datos de la simulación.

Camino: Es una sucesión de puntos con orientación en el espacio.

Algunos de los objetos de escena tienen propiedades especiales configurables que les permite interactuar con otros objetos. Las principales son:

Colisionable, si se puede detectar cuando ocurre una colisión con otro objeto.
Medible, si se puede calcular la distancia ente estos y otros objetos medibles.
Detectable, si pueden ser detectados por sensores de proximidad.
Visibles, si pueden ser vistos a través de una vista.

3.1.1.4 Formas (Shapes) en CoppeliaSim

Las formas son como hemos dicho antes unos objetos compuestos por caras triangulares que forman una malla rígida. Pueden aparecer como de distintos tipos como vemos en la Figura 47 (Coppelia Robotics, s.f.).

Figura 47.Los distintos tipos de formas, de izquierda a derecha: simple, compuesta, convexa, convexa compuesta, primitiva, primitiva compuesta y campo de altura.

Las formas simples y compuestas pueden representar cualquier malla y no están optimizadas para calcular la respuesta a colisiones dinámicas. Se diferencian en que la compuesta puede tener varios colores y conjunto de atributos viduales y la simple solo puede tener un color y conjunto de atributos.

Las formas convexas y convexa compuestas representan una malla convexa y están optimizadas apara cálculo de respuesta dinámica de colisión. Se diferencian de la misma manera que las formas simples y compuestas, en la cantidad de colores y conjunto de atributos visuales.

Las formas primitivas y primitivas compuestas representan a uno a o un grupo de prismas, cilindros o esferas. Están muy optimizados para el cálculo de colisiones dado su sencillez estructural.

El campo de altura puede usarse para representar un terreno con sus diversas elevaciones y depresiones. También pueden ser considerados formas primitivas y por tanto estar optimizadas para cálculo de colisiones.

Se pueden importar modelos a CoppeliaSim, por defecto se importan como formas simples. En la Figura 48 podemos ver un modelo importado con una textura aplicada y una forma primitiva con un color simple.

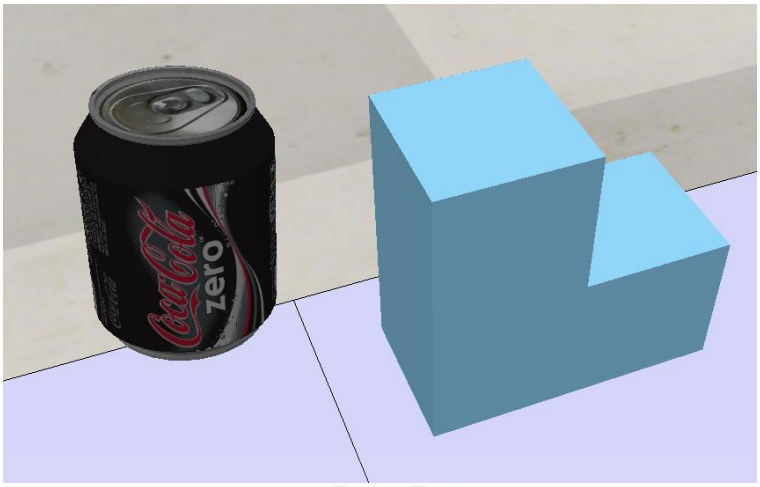


Figura 48.Modelo importado de una lata de Coca-Cola y una forma simple.

Las formas son objetos colisionables, medibles y detectables. Y por tanto pueden ser detectados por sensores de proximidad, pueden medirse su distancia con otros objetos medibles y pueden ser colisionados por otros objetos colisionables. Todas estas propiedades pueden ser alteradas si así fuera requerido.

3.1.1.5 Articulaciones en CoppeliaSim

Estos objetos permiten el movimiento relativo entre un objeto eslabón padre y sus eslabones hijos. Como ya hemos mencionado anteriormente, cuando en la jerarquía de escena tenemos unos objetos con una relación de padre e hijo, si el objeto es una articulación permite cambiar la posición angular y linear de sus hijos. Las articulaciones pueden operar en distintos modos, estos son como podemos observar en la Figura 49 (Coppelia Robotics, s.f.):

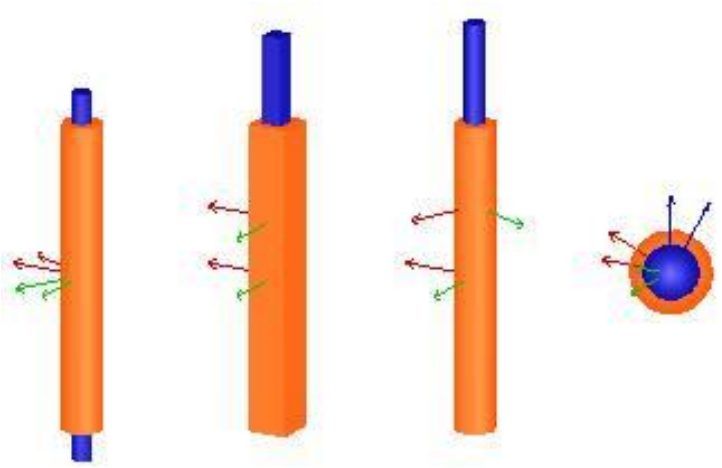


Figura 49. De izquierda a derecha: giratoria, prismática, tornillo y esférica

Las articulaciones giratorias tienen un grado de libertad y describen el movimiento rotacional entre dos eslabones. La rotación sucede desde el eje Z del eje de referencia de la articulación.

Las articulaciones prismáticas tienen un grado de libertad y describen el movimiento traslacional entre eslabones. La traslación sucede desde el eje Z del eje de referencia de la articulación.

Las articulaciones tornillo, se pueden interpretar como una combinación de una articulación giratoria y otra prismática. Tiene un grado de libertad y describe un movimiento similar al de un tornillo. La traslación/rotación sucede desde el eje Z del eje de referencia de la articulación.

Las articulaciones esféricas poseen tres grados de libertad y son usados para describir un movimiento rotacional entre eslabones. En algunas situaciones se puede aproximar una articulación esférica a tres articulaciones ortogonales y concurrentes (Figura 50). Pero si una de las articulaciones fuera o se aproximara a ser coincidentes con otra en lugar de mantener una orientación distinta entre las tres se crearía una situación de singularidad y se perdería un grado de libertad (fFigura 51). Lo anterior descrito no pasaría con una articulación esférica puesto que están pensadas para evitar estas situaciones.

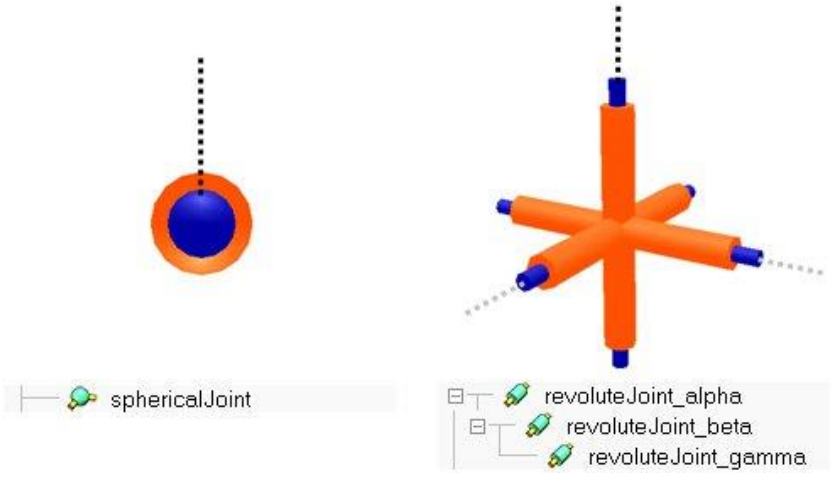


Figura 50.Tres articulaciones giratorias pueden equivaler mecánicamente a una articulación esférica

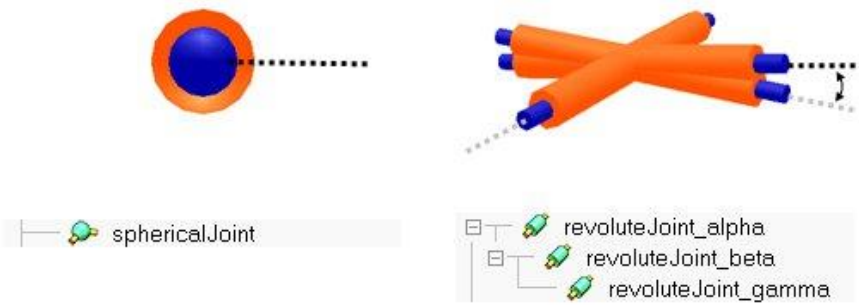


Figura 51. Situación cercana a una singularidad en la que dos de las articulaciones están a punto de solaparse

3.1.1.6 Maniquíes (Dummies) en CoppeliaSim

Estos objetos son un mero punto con una orientación y pueden ser usados como puntos de referencia. Pueden usarse para ayudar en la simulación. Por ejemplo, en mi caso s e usaron para saber cuál era el punto en donde se movía la muñeca del robot y con ello saber la posición de los objetos que están unidos a la muñeca como una cámara y una pinza.

Estos objetos son colisionables, medibles y detectables. Podemos ver su representación 3D en la Figura 52

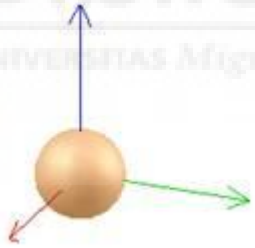


Figura 52. Dummy en CoppeliaSim

3.1.1.7 Sensores simulados en CoppeliaSim

En este apartado entraremos en detalle sobre los sensores utilizados en CoppeliaSim y especialmente en los usados en nuestra simulación.

Los sensores son componentes virtuales que permiten que un robot o cualquier objeto perciba su entorno, simulando el comportamiento de sensores reales como cámaras, láseres, sensores de proximidad, de fuerza, etc.

Son fundamentales en simulación porque permiten implementar toma de decisiones, navegación, visión artificial, manipulación y aprendizaje.

Sensores de visión

Estos sensores se usan para simular cámaras de tipo RGB, de profundidad o ambas. Si un objeto es renderizable, se mostrará en el output de la cámara cuando entre en su campo de visión, si por el contrario el objeto no es renderizable por pantalla, solo mostrara pixeles en negro, esto se puede apreciar en la Figura 54. El campo de visión de la cámara puede ser ortográfico, cuando el campo es rectangular o de perspectiva cuando es trapezoidal, podemos verlo claramente en la Figura 53

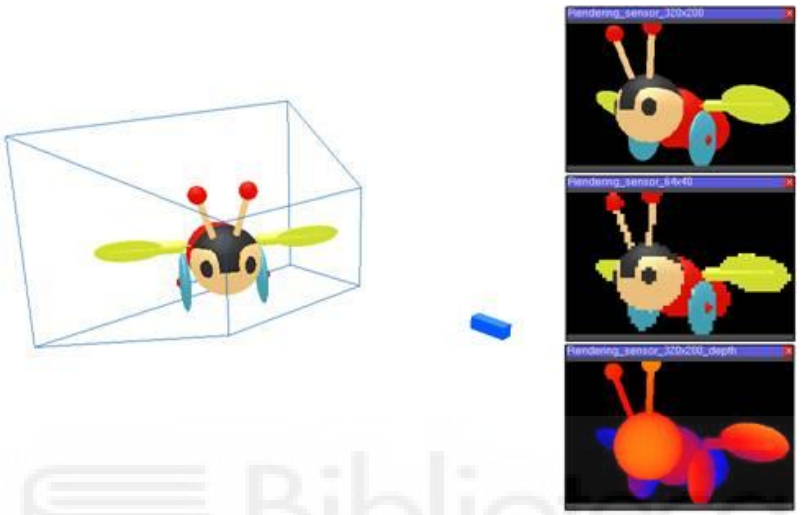


Figura 54. Demostración de uso de un sensor de visión en CoppeliaSim. El fondo es renderizado como pixeles en negro mientras que el objeto se muestra por pantalla. también se muestran opciones de posrenderizado

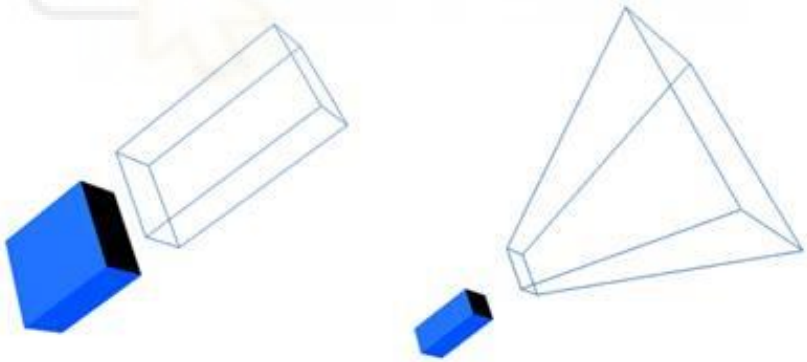


Figura 53. Distintos tipos de cámara según su tipo de campo de visión

Los sensores de visión se pueden usar de múltiples maneras. En nuestro caso emplearemos principalmente un sensor de visión de proyección de perspectiva para captar una señal de video. Esta señal será procesada con la ayuda de la librería OpenCV y un algoritmo de detección de objetos detectará los objetos de la escena.

Sensores de proximidad

Los sensores de proximidad detectan si un objeto está dentro de su campo de acción, su comportamiento es sencillo y simula distintos tipos de sensores como infrarrojos, ultrasonidos, LIDAR, etc. Pueden devolver la distancia y posición del punto del objeto detectado, siempre que este se halle dentro de su volumen de detección, en la Figura 55 se muestra un esquema. En la Figura 56 vemos los distintos tipos de sensor de proximidad en CoppeliaSim por su tipo de volumen de detección. Cada uno de ellos se puede usar para simular distintos tipos de sensores (Coppelia Robotics, s.f.).

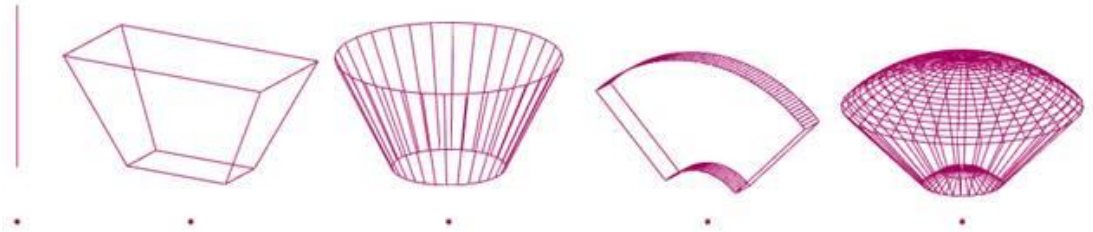


Figura 56. De izquierda a derecha: sensor de tipo haz de luz, de tipo pirámide, de tipo cilíndrico, de tipo disco y de tipo cono o de haz de rayos aleatorio

Figura 55. Esquema del funcionamiento del cálculo de la distancia de un objeto detectado.

El sensor de tipo haz de luz, es el que se ha utilizado en nuestra simulación, en nuestro caso se trataría de un sencillo sensor infrarrojo que detiene el rodillo cuando detecta que un objeto está llegando al fin de la cinta transportadora. De manera que el objeto se quedaría en una posición en donde el robot es capaz de alcanzarlo.

Sensores de fuerza

Los sensores de fuerza funcionan como una unión rígida entre dos formas que es capaz de medir las fuerzas transmitidas y los pares. Estos sensores pueden romperse sin por ejemplo una fuerza o par supera su umbral. En la Figura 57 mostramos en antes y el después (Coppelia Robotics, s.f.).

El sensor solo puede estar operacional durante la simulación si está habilitado dinámicamente.

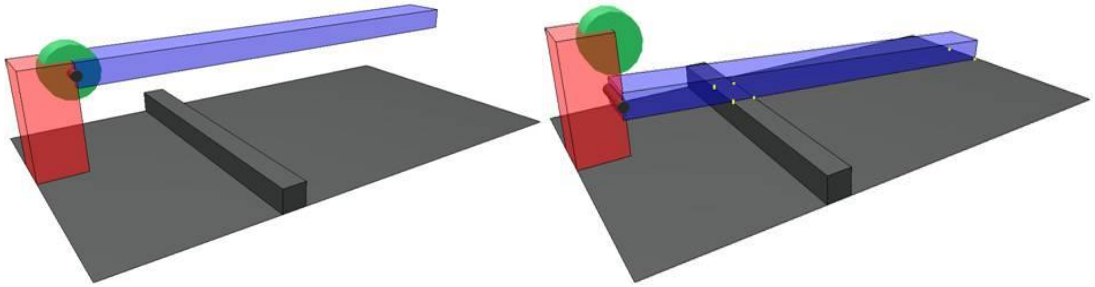


Figura 57. Ejemplo de rotura del sensor de fuerza.

Un sensor de fuerza mide una pareja de tres valores representando la fuerza en el sensor respecto de los ejes X, Y, Z y el par en el sensor respecto los mismos ejes.
Podemos ver una visualización en la Figura 58

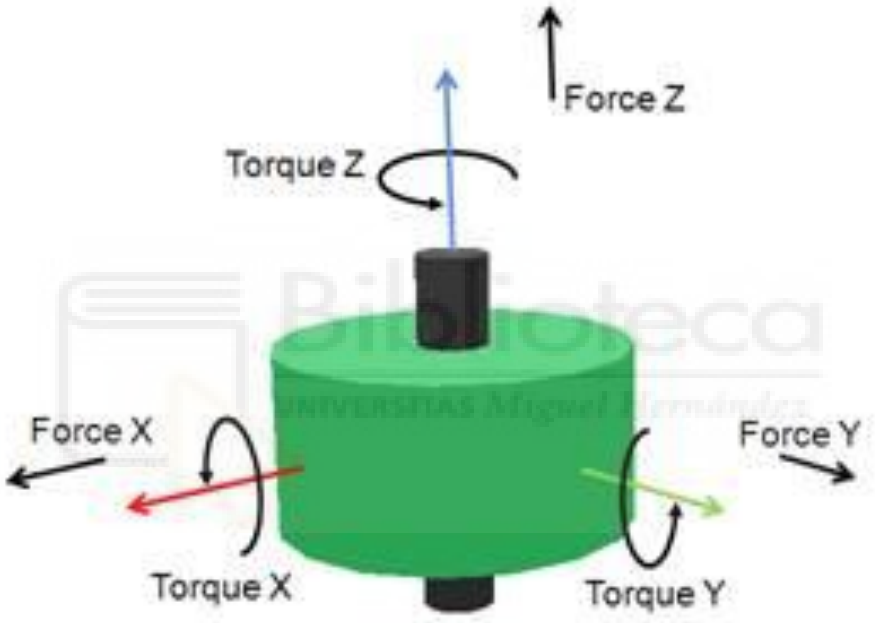


Figura 58. Las distintas fuerzas y pares sobres los tres ejes que mide un sensor de fuerza.

3.1.1.8 Scripts en CoppeliaSim

CoppeliaSim es un programa extremadamente personalizable. Casi todo en una simulación puede ser definido por el usuario. Esta flexibilidad también se extiende a los scripts que pueden usar tanto el lenguaje de programación Lua como Python, ya que posee interpretes para ambos.

Hay varios tipos de scripts soportados en CoppeliaSim como podemos ver la Figura 59: los sandbox (cajón de arena), los add-ons y los embebidos que están compuestos por el script principal, el cual ya habíamos comentado en el capítulo 3.1.1.3. Escena, y los scripts de objetos con los que vamos a ver un poco más a fondo (Coppelia Robotics, s.f.).

Los scripts son invocados mediante funciones de llamada de vuelta “callback” y siguen una orden de ejecución estricta. Pueden ser “threaded” y “non-threaded” y se

diferencian en que el primero puede correr sin interrumpir a CoppeliaSim. Por defecto se interrumpe de manera preventiva y se vuelve a activar después por Coppelia. Y el segundo su activación siempre depende de una función callback. Esto significa que siempre que se les llama deberán cumplir su función y devolver un valor, si no devuelven nada Coppelia se parara.

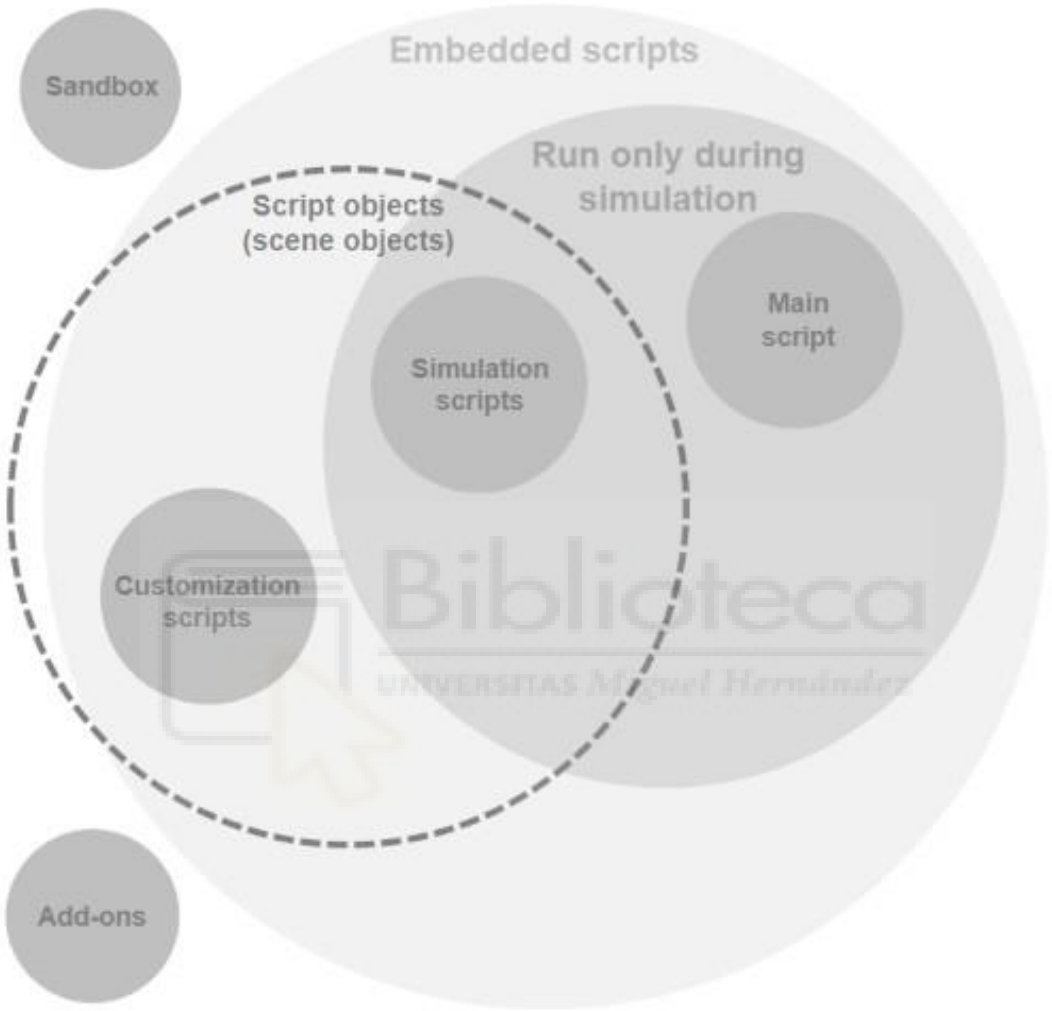


Figura 59.Distintos tipos de scripts en CoppeliaSim

Scripts de objetos

Un objeto script es un objeto de escena que encapsula un script. Este script controla partes específicas de un modelo. Hay de dos tipos y podemos ver sus iconos en la Figura 60:

Los scripts de simulación que representan un código que controla una función particular en la simulación y solo se activa cuando la simulación está en marcha.

Los scripts de personalización representan código que se ocupa de personalizar un aspecto de la escena o modelo. Puede funcionar incluso cuando la simulación está parada (Coppelia Robotics, s.f.).

Figura 60.Iconos que representan scripts de simulación y personalización respectivamente

3.1.1.9 Cintas transportadoras en CoppeliaSim

Si bien las cintas transportadoras no son un objeto único en Coppelia, se pueden colocar y personalizar de manera sencilla. Al ser usada en nuestras simulaciones es un objeto perfecto para demostrar cómo funcionan los objetos en CoppeliaSim de forma más

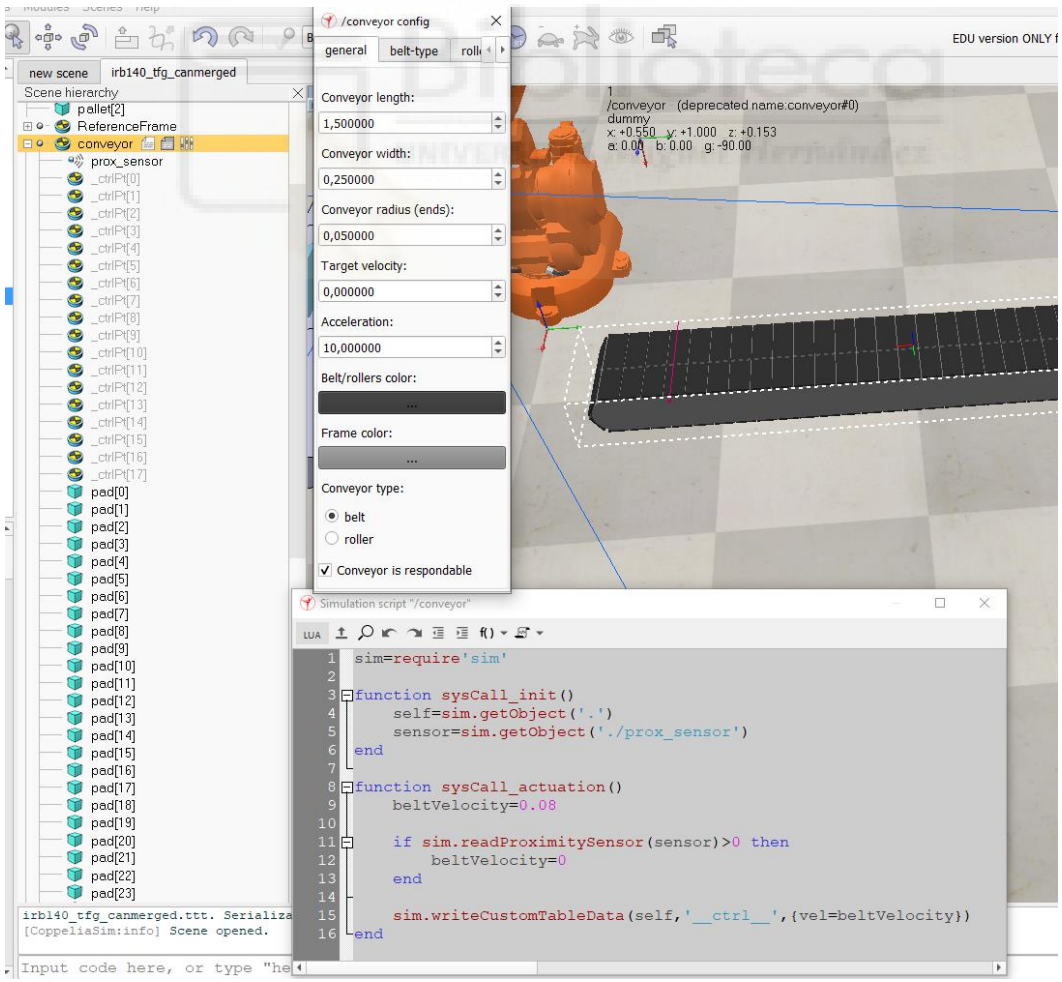


Figura 61.Cinta transportadora simulada en CoppeliaSim y mostrando los distintos componentes que la conforman

práctica. Vamos a poder ver los distintos objetos que conforman la cinta transportadora en la Figura 61.
Como podemos observar en la jerarquía de escena, tenemos los distintos modelos primitivos que conforman la cinta transportadora, por otro lado, tenemos en la parte de arriba el menú de ajustes de parámetros y en la parte de abajo el script del objeto.
Hemos añadido un sensor de proximidad de tipo haz, al final de la cinta, y el script está preparada para que cuando se detecte un objeto se pare la cinta.

3.1.2 PyCharm y librerías usadas

Figura 62. Logotipo de Pycharm

PyCharm (Figura 62) es un entorno de desarrollo integrado o IDE por sus siglas en inglés, creado por JetBrains y diseñado específicamente para el desarrollo de Python. Se caracteriza por función de Autocompletado inteligente, su depurador gráfico integrado, su sistema de gestión de paquetes con pip y su integración con Git y GitHub.

Su uso se debe principalmente a la experiencia con este IDE obtenida en asignaturas del grado.

Las librerías principales que usaremos son:

3.1.2.1 PyArte

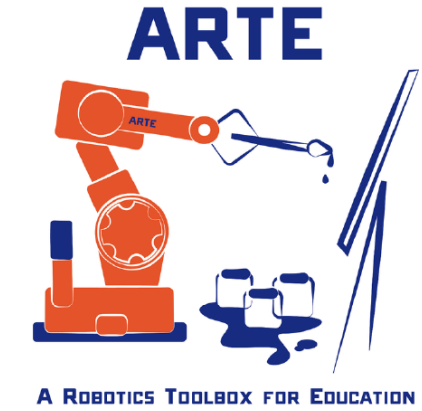


Figura 63. Logotipo de ARTE

PyArte (Figura 63) funciona como caja de herramientas ofreciéndonos múltiples funcionalidades como los cálculos de cinemática directa e inversa para el movimiento del robot, la conexión de la cámara dentro de Coppelia con scripts externos, y varias demos que sirven de ejemplos. Fue creado por nuestro tutor Arturo Gil Aparicio, para su uso junto con CoppeliaSim y Matlab.

3.1.2.2 Numpy

Figura 64.Logotipo de Numpy

Numerical Python (Figura 64) es una biblioteca esencial para el cálculo numérico en Python. Es especialmente rápida y eficiente para problemas de algebra lineal, estadística, operaciones de vectores y matrices y transformaciones de firma y tipo de datos. Su integración en múltiples bibliotecas usadas la hace indispensable para nuestro proyecto, por ejemplo, las imágenes en OpenCV son manejadas por vectores de Numpy.

3.1.2.3 Sklearn

Figura 65.Logotipo de Sklearn

Scikit-learn es una de las bibliotecas más populares para el aprendizaje automático en Python. Está construida sobre Numpy, SciPy y Matplotlib y proporciona herramientas para el desarrollo de modelos de IA como la clasificación, el cálculo regresivo, el agrupamiento o clustering, la selección de características, la validación cruzada y la evaluación de modelos. En nuestro caso la usamos para disponer del algoritmo de ordenamiento (RANSAC) Random sample consensus que usaremos para detectar si el brazo robótico está en la posición correcta para agarrar el objeto.

3.1.2.4 cv2

Figura 66.Logotipo de OpenCV

Es un módulo de OpenCV (Open Source Computer Vision) (Figura 66), y se trata de una de las librerías más populares para procesamiento de imágenes y visión por computador. Se caracteriza principalmente por capacidad para el procesamiento de imágenes y videos, su soporte para múltiples formatos, sus algoritmos implementados de reconocimiento de

objetos como cascadas de Haar y su interfaz que lo hace compatible con distintos modelos pre-entrenados.

Su uso se debe principalmente a su versatilidad, su uso en la librería PyArte y nuestra experiencia anterior gracias a la asignatura de Visión por Computador. En la Figura 67 podemos ver su funcionamiento en el programa junto a YOLO

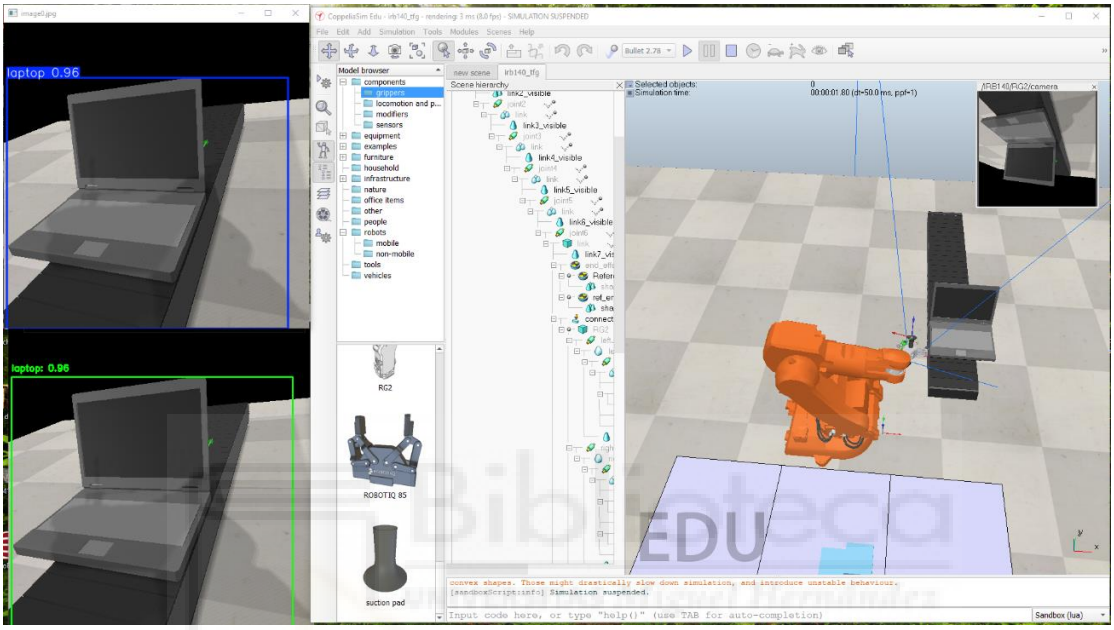


Figura 67. Ejemplo de funcionamiento, CV2 capta la imagen de la cámara de CoppeliaSim y YOLO reconoce el objeto enfocado.

3.1.2.5 Ultralytics

Figura 68.Logotipo de Ultralytics

Ultralytics (Figura 68) es la librería de Python especializada en visión artificial y aprendizaje profundo, es necesario para usar el algoritmo de reconocimiento de objetos YOLO (You only look once). Permite usar modelos pre-entrenados o empezar con uno desde cero, entrenar y ajustar modelos, realizar detección, segmentación y clasificación de imágenes y videos, exportar modelos a múltiples formatos y es de fácil implementación.

3.2 Herramientas empleadas

CoppeliaSim es un entorno muy potente de simulación robótica, pero, sigue teniendo ciertas limitaciones. Hay ciertos accesorios para robots, como cámaras especificas o ventosas que se muestran de forma genérica dentro del entorno de simulación. Cuando tratemos con uno de esos casos entraremos más en detalle sobre que accesorios se podrían usar para trasladar la aplicación de "pick and place" a la realidad

3.2.1 Robot IRB140

El robot IRB140 (Figura 69) es un brazo robot industrial con 6 ejes de libertad que podemos ver en la Figura 70 , diseñado para tareas de alta precisión en espacios reducidos. Posee un diseño compacto, un alcance de 81cm (Figura 71) y es capaz de aguantar hasta 6 Kg (ABB, 2018).

Este robot se caracteriza por su gran repetibilidad de ±0.03 mm y su flexibilidad de montaje, ya que puede instalarse en el suelo, en la pared o incluso en el techo. Además, es compatible con los controladores IRC5 y S4C+, lo que facilita su programación mediante RAPID, el lenguaje propio de ABB.

Usaremos este robot, ya que es el que mejor preparado está dentro de la librería PyArte y posee varias ventajas a la hora de realizar cálculos. Por ejemplo, su efector final se trata de una muñeca esférica, lo que nos permite una gran movilidad y adaptación.

Para más información sobre el robot podemos consultar el anexo 6.1.

Figura 69. Robot IRB140

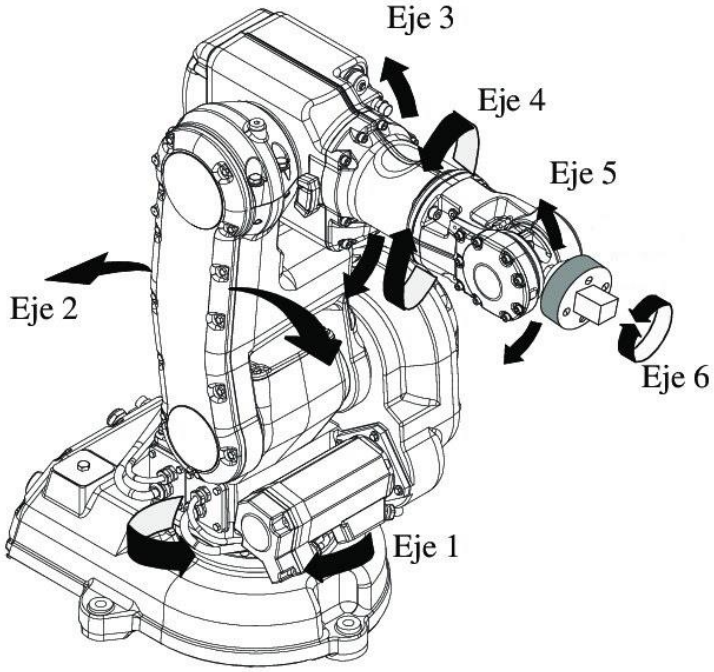


Figura 70.Ejes del robot IRB140

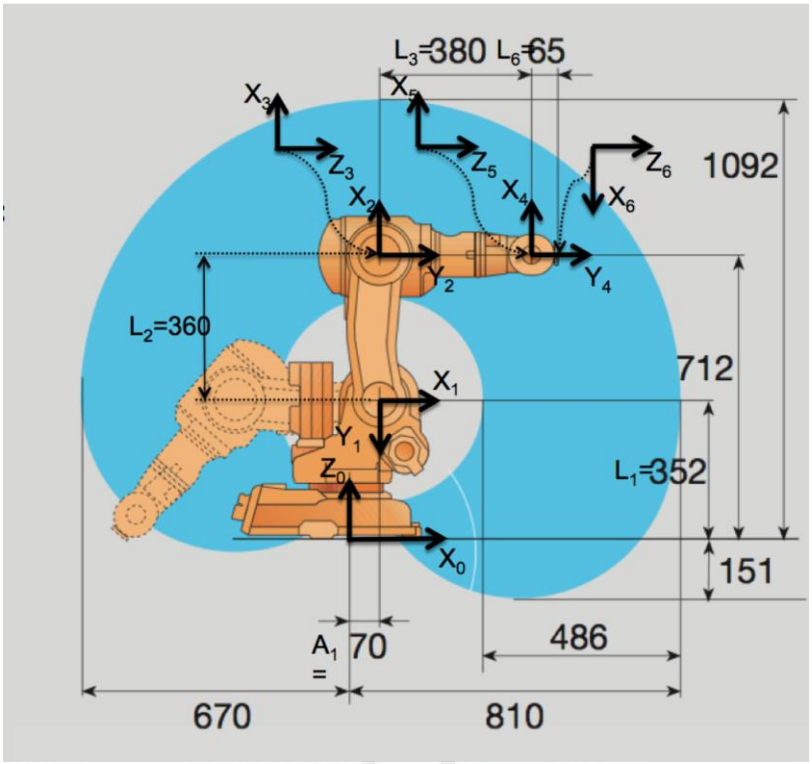


Figura 71.Rango de movimiento del robot IRB140

3.2.2 Pinza RG2 y Ventosa VGC10

Ambas fabricadas por OnRobot, (Figura 72) se trata de dos de sus herramientas más populares, ambas diseñadas para mejorar la flexibilidad y eficiencia en la automatización industrial.

Consiste en una pinza de dos dedos con una apertura de hasta 11 cm y una capacidad de carga de 2 Kg. También suelen incluir pinzas especializadas para diferentes objetos y una fuera de agarre ajustable desde 3N hasta 40N. Y su instalación suele ser muy sencilla con capacidad de “plug and play” con la mayoría de los fabricantes. Para saber más sobre sus especificaciones podemos ver el anexo 6.2.

Figura 72. *Pinza RG2 y ventosa VGC10*

Por otro lado, CoppeliaSim no tiene modelos específicos para distintas ventosas, sino que usa un modelo genérico para todas. Una posible solución en la vida real seria usar una ventosa VGC10. Entre sus características tenemos un sistema de vacío integrado, de manera que no necesita un compresor externo, una serie de ventosas ajustables y configurables dependiendo del objeto, una capacidad de carga de hasta 15Kg, y su capacidad de controlar de forma independiente los lados izquierdo y derecho de la ventosa. Podemos ver sus especificaciones en el anexo 6.3

Se han elegido estos dos modelos por compromiso para mantener la consistencia entre la simulación y una posible solución en la vida real, pero, es cierto que, aunque teóricamente se pueden montar las dos juntas, no he encontrado información definitiva al respecto. Si esta combinación no fuese posible tendría que ser sustituida, por ejemplo, por una pinza hibrida del Zimmer Group, que posee tanto pinza como ventosa integrada, que podemos ver en la Figura 73. Podemos ver las características de esta opción más en detalle en el anexo 6.4.

Figura 73. Pinza híbrida del grupo Zimmer

3.2.3 Cámara Eyes

Fabricada por OnRobot y compatible para ser montada junto a la pinza RG2, esta cámara posee un sistema de visión 2.5D, capaz de detectar posición, orientación y profundidad, no requiere de un PC externo, ya que el procesamiento se hace en la propia cámara y tiene un sistema de calibración automática. Podemos ver en la Figura 74 la cámara montada en el último efector del robot, igual que en la simulación.

Figura 74. Ejemplo de montaje de la cámara en el efector final

En CoppeliaSim los sensores de visión son objetos genéricos, los cuales no se comportan como una cámara real actual, sino como una cámara modelo Pin-Hole, sin embargo, su modelo matemático puede representar el funcionamiento de la mayoría de las cámaras actuales. En nuestro caso la cámara Eyes, consigue su visión 2.5 porque es capaz de ver tanto en el espectro visible como en el infrarrojo. La cámara proyecta un haz en infrarrojo y detecta variaciones entre ambas imágenes para calcular la distancia (Figura 75). Según datos del fabricante a una distancia de 50 cm, consigue la distancia con un error medio de 1,649 mm, lo que lo hace bastante robusto para nuestra aplicación.
Para más detalles de las características de la cámara se puede ver en el anexo 6.5

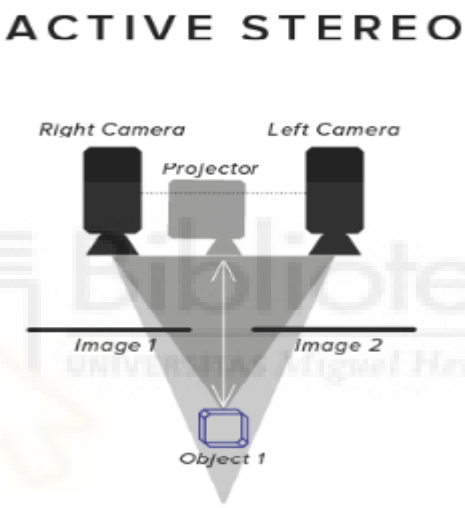


Figura 75. Representación del método active stereo para conseguir la profundidad

3.3 Descripción del código empleado

El objetivo de este proyecto es la simulación de una aplicación de "pick and place", siendo un componente muy importante la captura y el posprocesamiento de imágenes para reconocer el tipo y la posición de objetos y la situación óptima del robot para ser capaz de recogerlos y moverlos. Después de tratar el entorno de simulación en CoppeliaSim y las herramientas que podrían usarse en un caso práctico, en este punto trataremos de todo el código que se ha desarrollado usando el lenguaje de programación Python y que interactúa con el entorno de programación gracias a la API remota de CoppeliaSim. A través de esta API se puede controlar la simulación y podemos dar órdenes al robot. También hemos podido usar librerías y herramientas de procesamiento de imágenes que

funcionan en la vida real y que normalmente no son usadas dentro de la simulación. Gracias a esto podemos entrenar a un algoritmo de reconocimiento de objetos como YOLO dentro de la simulación, y en teoría podríamos usar ese algoritmo en la vida real para tener una aplicación funcional, sin tener que entrenar al algoritmo en la vida real. En otras palabras, si tuviéramos capacidad computacional de sobra, podríamos entrenar al algoritmo dentro de la simulación en todas las posibles situaciones que se pudieran dar en el sistema, y luego traspasarlo a un caso práctico con el algoritmo completamente entrenado.

En los siguientes puntos vamos a ver el flujo de funcionamiento del programa, y a entrar en detalle en las funciones que tiene.

3.3.1 Flujo de funcionamiento del programa principal

Para comprender el funcionamiento del programa y entender en que momento se usa cada función podemos ver el esquema del flujo de funcionamiento en la Figura 76. Primero se inicia el programa y se conecta a CoppeliaSim, iniciando así la simulación y los objetos de escena. A continuación, se arranca el hilo principal al cual hemos llamado "video_stream" que lo primero que hace es que el robot se mueva a su posición inicial. Cada vez que el robot se mueva pasara por la función "update_robot_state" que nos devuelve la posición global de la muñeca, la cual necesitamos para realizar ciertos cálculos. Seguidamente en "video_stream" si el sensor de la cinta trasportadora es activado, el robot mueve la muñeca a una posición donde la cámara podrá ver el objeto que viene por la cinta transportadora. Posteriormente, se llama a "process_yolo_results" que nos devolverá el tipo de objeto que esta en la cinta transportadora si YOLO lo reconoce y también nos devuelve la posición del objeto. Dependiendo del objeto se agarrará de una manera u otra. Por ejemplo, si se trata de un objeto con un agarre complicado, como un ordenador portátil, llamaremos a "generate_partial_circular_trajectory" que nos generara una trayectoria circular para el robot con el centro de la trayectoria siendo el objeto detectado, para que la cámara siempre apunte al objeto durante el recorrido llamaremos a la función "calculate_orientation". En este punto nos encontramos ante un claro problema de cinemática inversa, tenemos un punto en el espacio al que queremos llegar con la muñeca del robot y por tanto llamamos

a las funciones cinemáticas que son la función “inverse_kinematics” que a su vez llama a la función “delta_q_penrose” para solucionar el problema de cinemática inversa por cada punto de la trayectoria circular. El robot se moverá a la siguiente posición y llamará a “update_robot_state” y a “process_yolo_results” con estas dos funciones tenemos la posición de la muñeca y la distancia de la muñeca al objeto, en cada píxel de la imagen. En otras palabras, tenemos una nube de puntos en el espacio. Usaremos la función “normalize” para normalizar la información de la nube de puntos y seguidamente llamaremos a “calculate_rotation_from_depth” que usa el algoritmo RANSAC para sacar los grados de cada eje del objeto. Se repetirá todo este proceso hasta que se termine el recorrido de “generate_partial_circular_trajectory” o “calculate_rotation_from_depth” devuelva unos grados óptimos para el agarre, los cuales se han hallado de forma empírica para cada objeto. Si se encuentra el punto de agarre optimo se considera que el robot ya esta alineado con el objeto y solo tiene que avanzar en línea recta hacia el objeto y recogerlo, para seguidamente dejarlo en una posición deseada todo esto sucedería dentro del hilo principal “video_stream”. Al haberse realizado la aplicación de pick and place la simulación llega a su fin.

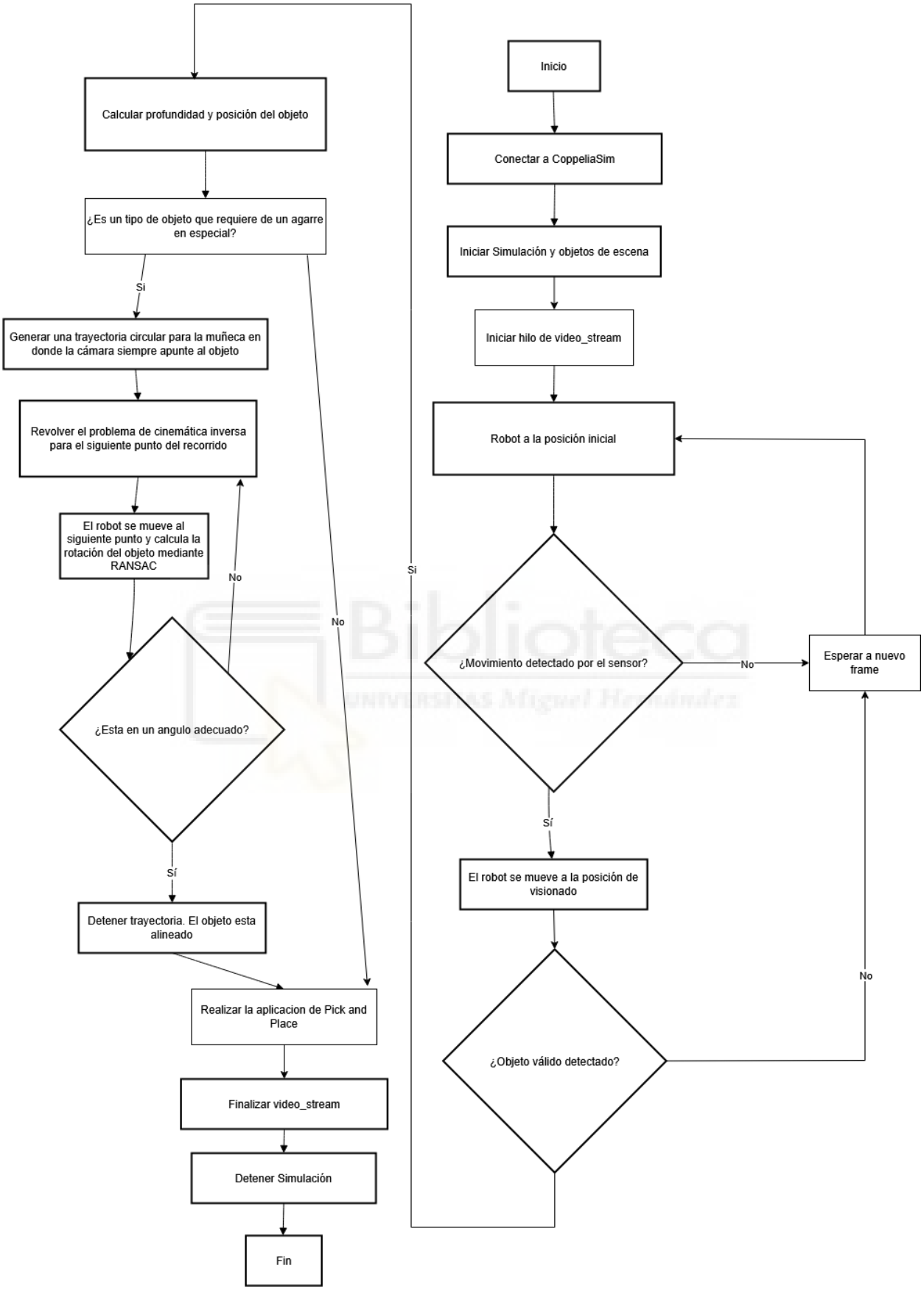


Figura 76.Diagrama de flujo del programa

3.3.2 Función principal “main”

La función principal se encarga de conectarse con la API externa de CoppeliaSim e inicializar la simulación y los distintos objetos de escenas con los que vamos a interactuar externamente como el robot, la pinza y la cámara. También se encarga de crear el hilo principal e inicializarlo. Finalmente, cuando se hayan realizado las operaciones se encargará de cerrar el hilo y terminar la simulación.

3.3.3 Función “update_robot_state”

Esta función resuelve el problema de cinemática directa del robot, habiéndonos dado el giro de las articulaciones nos devuelve la posición y orientación de la muñeca, usamos una simple matriz de transformación para a partir de esta información sacar la posición y orientación de la cámara y de la ventosa. Finalmente devolvemos estos dos valores.

3.3.4 Funciones cinematicas “delta_q_transpose” y “moore_penrose_damped”

Existen varios métodos para resolver el problema de la cinemática inversa. Una de estas funciones será ejecutada y devolverá los valores a la función “inverse_kinematics”. Estas funciones calculan un pequeño cambio en los ángulos de las articulaciones del robot, a partir del Jacobiano y el error en la posición. Estos métodos son iterativos y requieren de tiempo para ser calculados. Sus valores de entrada son el Jacobiano y el vector de error “e”. Y nos devuelven qd que representa los incrementos articulares necesarios para reducir el error cartesiano “e”

En la función delta_q_transpose se usa el método del Jacobiano traspuesto para estimar el cambio en las articulaciones. Esencialmente se está ejecutando esta fórmula:

$$\Delta q = \alpha \cdot J^T \cdot e$$

Siendo α un factor de escala adaptivo para garantizar la convergencia

Por otro lado, la función moore_penrose_damped usa la pseudo inversa del jacobiano del cual ya hemos hablado anteriormente en el punto 2.1.3. Primero se calcula la manipulabilidad del robot, ósea como de bien se puede mover. Y luego se pasa a hacer la pseudo-inversa clásica de Moore-Penrose. Cuando el Jacobiano se encuentra cerca de una singularidad, su pseudo-inversa se vuelve inestable. Para solucionarlo se añade un término de amortiguación.

3.3.5 Función "inverse_kinematics"

Esta función se encarga de resolver el problema de cinemática inversa. Se encarga de encontrar las posiciones articulares para que el robot alcance la matriz de transformación homogénea deseada. Para ello utiliza un método iterativo donde va actualizando los valores de las articulaciones en cada paso con un delta calculado en las funciones

```
Converged in  103  iterations!
Error final q:  [-1.08455954e-05  2.40881892e-04 -5.86306756e-04 -3.56677356e-04
 -1.18200294e-04  3.44944149e-04]
Error final q:  0.0008136791978473705
```

Figura 77. Ejemplo de cantidad de iteraciones necesarias para el cálculo de la cinemática inversa.

cinemáticas que hemos explicado anteriormente. Finalmente nos aseguramos de que los resultados estén dentro del límite de movimiento de las articulaciones y se devuelve el valor de q, es decir la posición de las articulaciones. Sus valores de entrada son el objeto robot que tiene asociados métodos dentro de la librería pyArte para este tipo de cálculos, la posición y orientación de destino y la configuración inicial de las articulaciones. Como podemos ver en la Figura 77, al realizarse tantas iteraciones muchas veces se ve afectada la simulación, al verse paralizada hasta que se termina el proceso.

3.3.6 Función "normalize"

Se trata de una función auxiliar que retorna un vector normalizado unitario. Se calcula la norma Euclídea del vector y se verifica que no sea nulo, es decir que todos los valores sean ceros.

3.3.7 Función “calculate_orientation”

Esta es otra función auxiliar, que nos ayuda a orientar la cámara durante el recorrido, para que apunte todo el rato al objeto y tenga una orientación estable sin giros ni rotaciones inesperadas. Ocurre que cuando se calcula la cinemática inversa a veces la solución hacia que la muñeca girara 180º grados y eso creaba problemas para nuestro algoritmo de detección de objetos. Entregándole el centro de la trayectoria circular y la posición de la cámara nos devuelve una matriz de rotación.

3.3.8 Función “generate_partial_circular_trajectory”

Con esta función se genera una trayectoria 3D en forma de arco sobre el plano XY, manteniendo una altura Z constante, de manera que tenemos una trayectoria de observación del objeto sin obstaculizaciones y desde distintos puntos, lo que permite que se inspeccione el objeto y se encuentre una posición optima de agarre. Los valores que se le entregan son el centro del círculo, el radio, el número de puntos que queremos generar para la trayectoria, la altura del arco y el ángulo inicial y final. Estos valores se pueden automatizar, para que a través de los valores conseguidos por la visión artificial se generen la ruta. Sin embargo, esto puede traer problemas porque se podría generar una ruta que esta fuera del alcance del robot. Por lo que se han conseguido unos valores de forma empírica que funcionan bien.

3.3.9 Función “calculate_rotation_from_depth

Esta función estima la rotación de un objeto a partir de una imagen de profundidad dentro de la región de interés o “ROI” por sus siglas en ingles. Utiliza los parámetros intrínsecos de la cámara y un algoritmo RANSAC para ajustar un plano 3D a la nube de puntos que obtenemos del “ROI”, consiguiendo así los ángulos de orientación del objeto. Primero sacamos los valores de profundidad del recuadro delimitante de YOLO, ya que dentro se encuentra el objeto. Luego se usa la fórmula de una cámara pin-hole para calcular las coordenadas 3D a partir de los pixeles 2D, y como dentro del “ROI” puede aparecer el fondo o algo más cercano, se ignorarán los valores más lejanos y cercanos y nos quedaremos solo con los datos razonables. Posteriormente se ajusta el plano Z a los puntos 3D usando RANSAC y se obtiene la normal del plano y se usa la función “normalize”.

Para finalizar, con la normal del plano se calculan la inclinación sobre el eje Y, y el giro sobre el eje Z. El valor de X se ignora por simplicidad. A la función le llega una imagen de profundidad, las coordenadas del ROI, las distancias focales y las coordenadas del centro óptico de la cámara. Y nos devuelve el vector normal del plano ajustado y el pitch (la rotación en torno a Y), el yaw (la rotación en torno a Z) y el roll (la rotación en torno a X) que se deja a cero por defecto.

3.3.11 Función "process_yolo_results"

Esta función procesa los resultados de la detección de objetos de YOLO. Es posible en visión artificial que se detecten varias veces el mismo objeto, podemos observarlo en la Figura 78, aunque YOLO tiene mecanismos para evitar esto, para dotarlo de más robustez, hacemos que solo trabaje con el cuadro delimitante o "bounding box" con el que más confianza tiene, ósea que el programa está más seguro de que se trata de ese objeto en particular. Por otro lado, se encarga de calcular la posición 3D del objeto usando la profundidad y los parámetros intrínsecos de la cámara, como se considera que el "bounding box" está centrado en el objeto se usa el valor del centro del recuadro para calcular la distancia. Como sabemos por otras funciones la posición exacta de la cámara y sabemos la profundidad o distancia entre la cámara y este punto central se puede conseguir así la matriz de transformación homogénea para agarrar el objeto. A esta

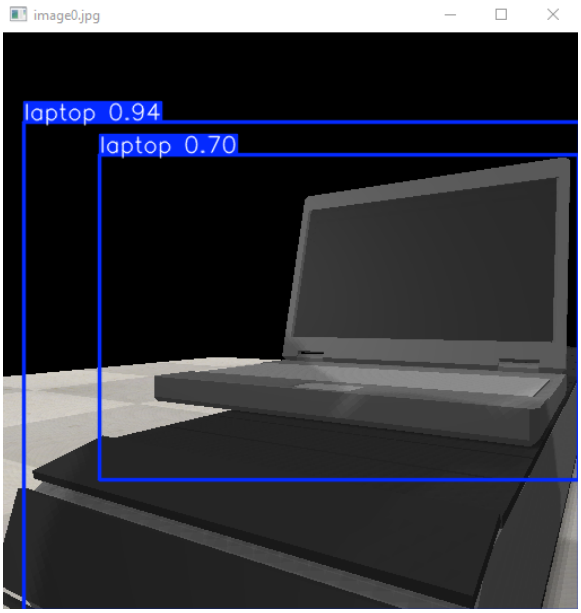


Figura 78.Detección múltiple del mismo objeto por parte de YOLO

función entran la imagen captada, los resultados predictivos de YOLO, los valores de

profundidad de la imagen (Figura 79) y los valores de los ángulos de las articulaciones. Si detecta un objeto que reconoce entonces devuelve, el nombre del objeto detectado, las posiciones del recuadro delimitador, el centro del recuadro, las distancias focales de la cámara, la matriz desde la cámara al objeto y el valor de la profundidad del centro del objeto.

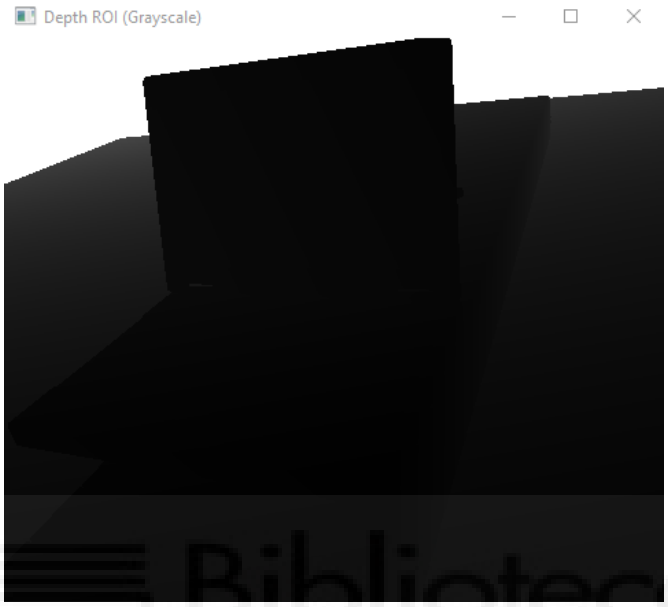


Figura 79. Imagen de profundidad, mostrado como escala de grises.

3.3.10 Función “video_stream”

Esta función es el núcleo del programa y controla el flujo continuo de la detección de objetos con YOLO, y la decisión de qué hacer con cada tipo de objeto incluyendo la ejecución de la trayectoria del robot. Cuando se inicia esta función, primero se envía al robot a una posición inicial mediante cinemática directa, y espera a que el sensor de la banda transportadora detecte un objeto. Cuando el sensor se activa, el robot se mueve nuevamente mediante cinemática directa a una posición ideal para visualizar el objeto con la cámara. Se captura la imagen, se obtiene su mapa de profundidad y se detectan los objetos con YOLO. A continuación, se llama a la función “process_yolo_results” para que nos devuelva la posición 3D del objeto detectado. Si detecta un objeto esperado se iniciarán una serie de funciones para agarrar el objeto y dejarlo en las cajas.

3.3.11 Clase robot.cameraTFG

Este código es una modificación de la clase Camera que se encuentra en la librería PyARTE. Para evitarnos problemas, se copió la clase camera y se modificó para captar video y conseguir la profundidad de los pixeles de la imagen. Así pues, en la parte de captura de imágenes en lugar de usar cv2.flip(image, 0) usamos cv2.cvtColor(image, cv2.COLOR_BGR2RGB) ya que es preferible para tomar video reordenar los colores así. También hemos añadido para obtener el mapa de profundidad la función "get_depth" que podemos ver en la Figura 80 . Que obtiene los datos de profundidad de la cámara, convierte los datos binarios a un vector de números flotantes y devuelve este vector para ser usado en las funciones.

```
10 usages (10 dynamic)
def get_depth(self):
    depth, res = self.simulation.sim.getVisionSensorDepth(self.camera,1,)
    float_array = array.array('f')
    float_array.frombytes(depth)
    depthValues = list(float_array)
    return depthValues
```

Figura 80. Función get_depth que capta los valores de profundidad de cada píxel de la imagen.

3.3.12 Clase robot.proxsensorTFG

Esta es una pequeña modificación de la clase "ProxSensor", que sirve para interactuar con sensores de proximidad.

Esencialmente hemos modificado la función "readvalues" para que nos devuelva las coordenadas relativas al sensor si el sensor detecta algún objeto. Podemos ver las modificaciones en la FIG

```
1 usage
def readvalues(self):
    result = self.simulation.sim.readProximitySensor(self.proxsensor)
    if result[0]:  # Si el sensor detecta algo (estado activado)
        point = result[2]  # Extraer las coordenadas relativas del objeto detectado
        return point
    else:
        return None  # No hay objeto detectado
```

Figura 81. Función readvalues dentro de la clase ProxSensor

4. Resultados y discusión

Tras haber mostrado las herramientas que hemos utilizado y haber presentado la estructura del código, en este apartado discutiremos sobre el desempeño del código, los resultados obtenidos y discutiremos sobre que opciones parecen preferibles para la aplicación.

4.1 Entrenamiento del algoritmo de reconocimiento de objetos y resultados.

El entrenamiento del algoritmo YOLO es esencial para el correcto funcionamiento del programa y ha pasado por varias iteraciones debido a errores consistentes en la detección de objetos. Por ejemplo, una de las primeras versiones, daba un falso positivo en donde confundía la cinta transportadora por un ordenador portátil (Figura 82). Seguramente

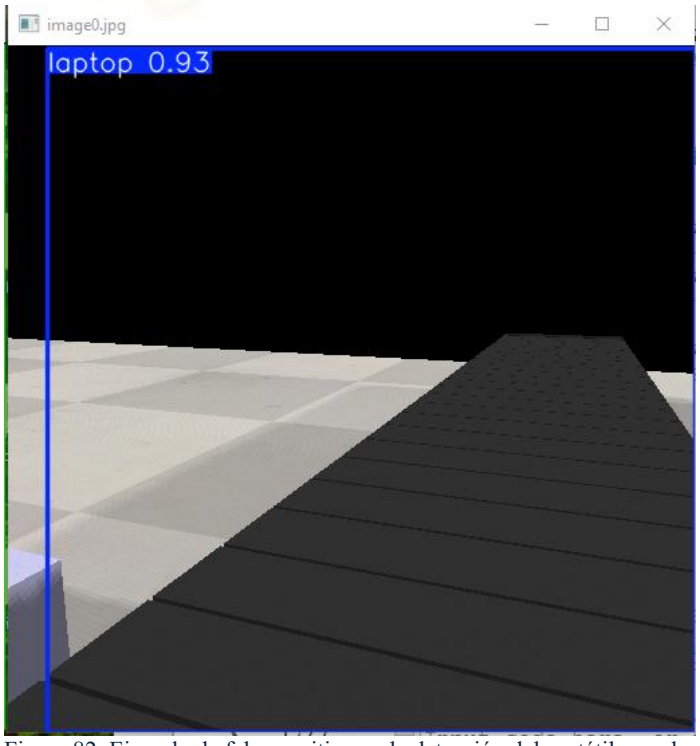


Figura 82. Ejemplo de falso positivo en la detección del portátil usando YOLO.

causado porque en la selección de imágenes de entrenamiento había muchas imágenes con el portátil sobre la cinta y no había la suficiente variedad de fondos.

En el modelo final entrenado por nosotros de YOLOv8, para tres objetos que hemos entrenado hemos usado alrededor de 300 imágenes por objeto, y luego lo hemos divido en una proporción de 90-10% para usarlas en el entrenamiento y la validación respectivamente. Las imágenes han sido tomadas desde varias perspectivas, distancias y fondos para ayudar al entrenamiento (Figura 83).

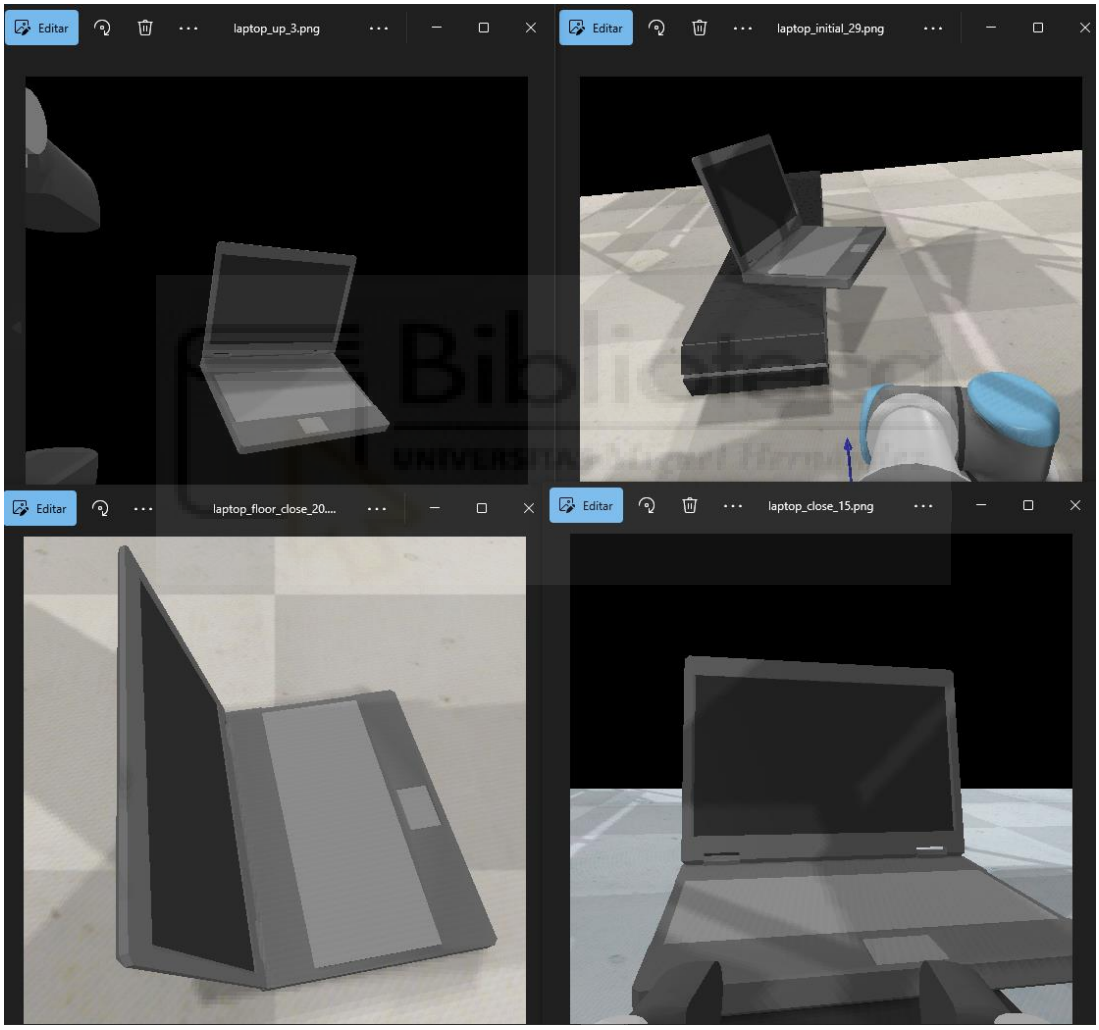


Figura 83. Ejemplo de imágenes de entrenamiento para el objeto portátil

La cantidad de imágenes y épocas realizadas para el entrenamiento resultaron en un periodo de entrenamiento largo, pero dieron lugar a un modelo muy robusto, en donde la cantidad de falsas identificaciones se ha reducido significativamente. Con alrededor de 800 imágenes para entrenar y 100 épocas, con un tiempo de entrenamiento por época medio de 160 segundos, o dos minutos y 40 segundos. El entramiento duró más de cuatro

horas y media (Figura 84), en parte por la falta de potencia computacional al no poder usar la tarjeta gráfica para acelerar el proceso ya que solo las de Nvidia son compatibles para el entrenamiento. Por lo que el procesador se ocupó de todo el proceso.

```
100 epochs completed in 4.605 hours.
Optimizer stripped from runs\detect\train3\weights\last.pt, 6.2MB
Optimizer stripped from runs\detect\train3\weights\best.pt, 6.2MB

Validating runs\detect\train3\weights\best.pt...
Ultralytics 8.3.61 🚀 Python-3.10.11 torch-2.5.1+cpu CPU (AMD Ryzen 5 5500)
Model summary (fused): 168 layers, 3,006,233 parameters, 0 gradients, 8.1 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 3/3 [00:05<00:00,  1.88s/it]
                   all         90         90      0.996          1      0.995      0.901
                   can         28         28      0.994          1      0.995      0.931
                laptop         30         30      0.995          1      0.995      0.944
               spatula         32         32      0.998          1      0.995      0.829
Speed: 2.0ms preprocess, 53.4ms inference, 0.0ms loss, 0.3ms postprocess per image
Results saved to runs\detect\train3

Process finished with exit code 0
```

Figura 84. Visión general del proceso de entrenamiento.

Tras, el entrenamiento, los resultados registrados por el programa son muy positivos. Las curvas de perdida van bajando conforme va avanzando el entrenamiento lo que demuestra que el modelo está aprendiendo de manera adecuada (Figura 85).

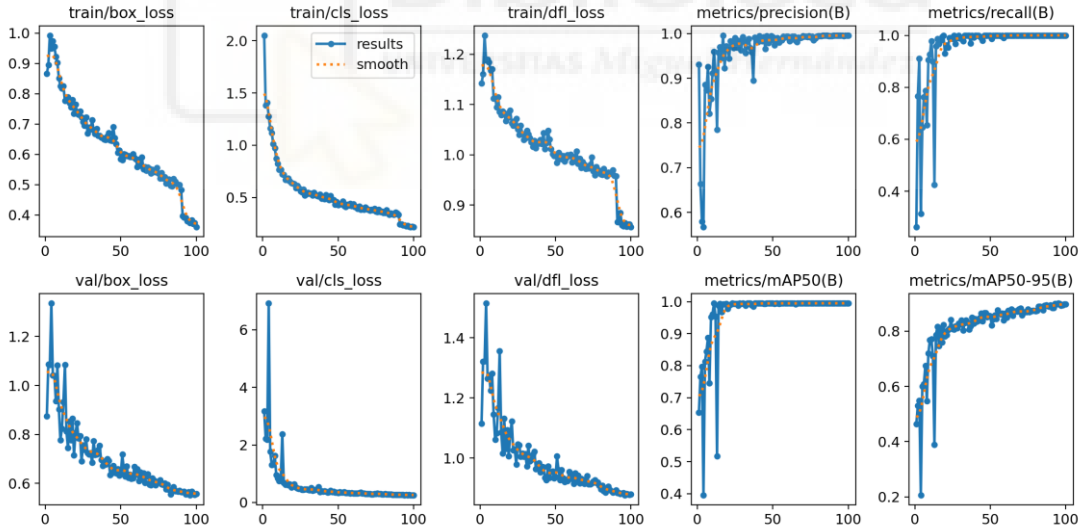


Figura 85. Imagen generada por el entrenamiento de YOLO con los resultados.

También podemos ver los resultados con las imágenes que hemos separado para la validación del modelo que muestran una alta tasa de confianza, especialmente en la lata y en el portátil (Figura 86). Seguramente la lata por su forma simple y el portátil por sus texturas planas. La espátula si bien tiene unos resultados muy buenos, su porcentaje de fiabilidad suele ser un 10% menor al de resto de objetos.

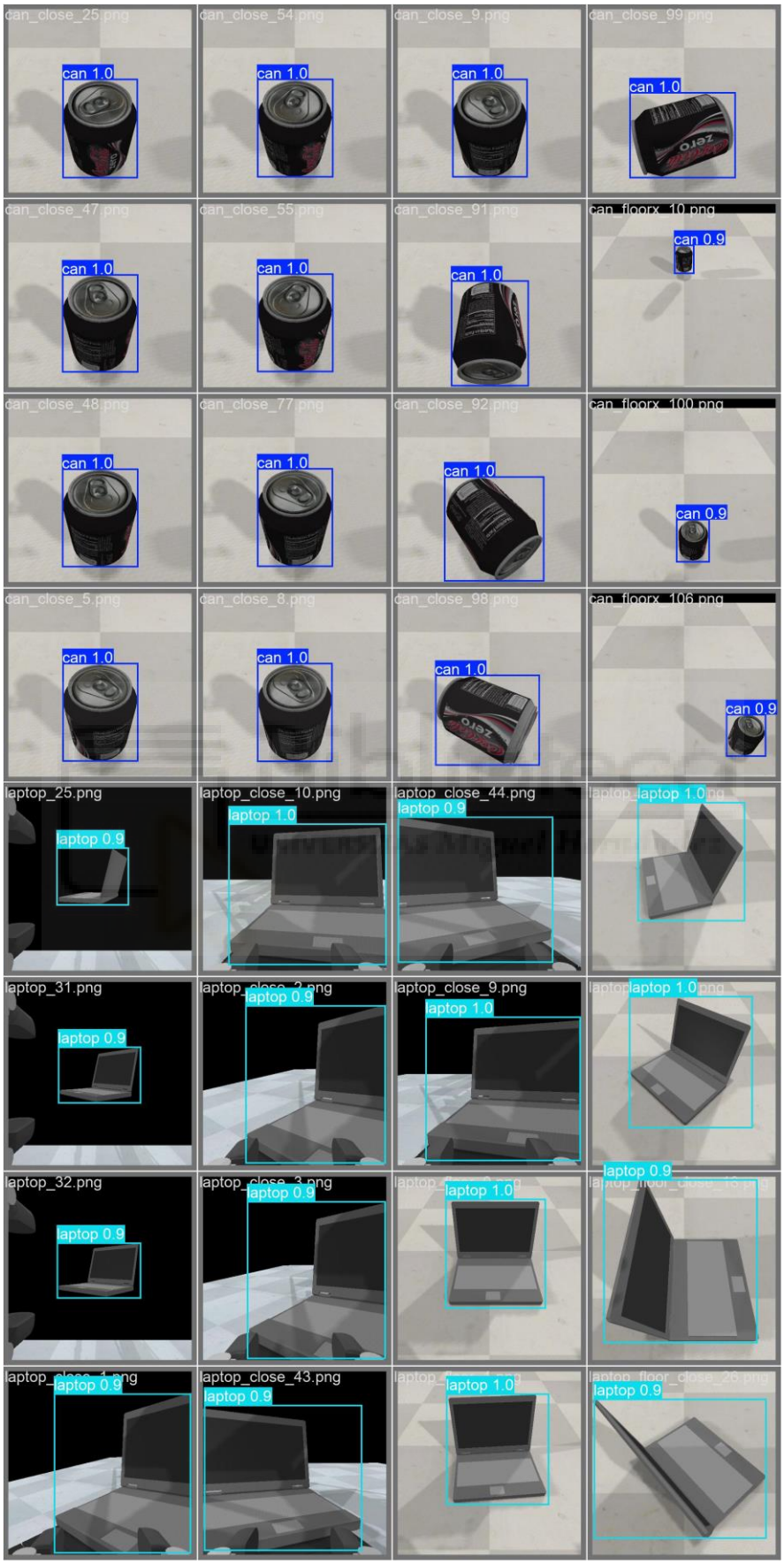


Figura 86. Resultados de la predicción de la lata y el portátil. Podemos observar altos porcentajes de fiabilidad.

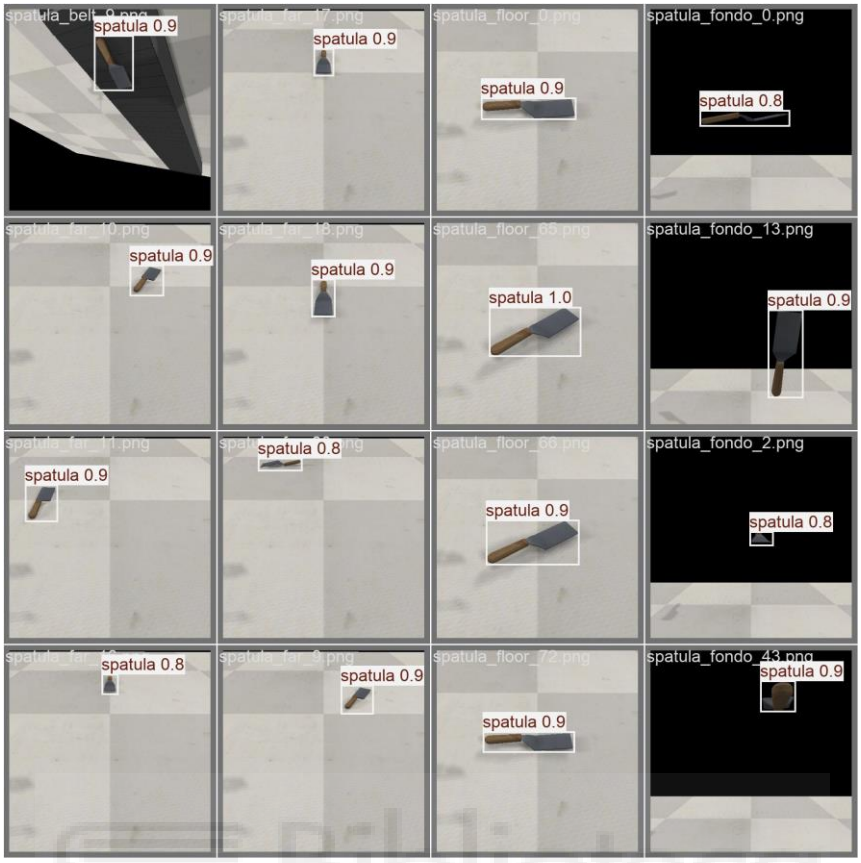


Figura 88. Resultados para la detección de la espátula usando las imágenes de validación.

Si bien los resultados son bastante buenos, y con unos valores de fiabilidad altos. Se ha detectado que a veces puede dar una falsa detección si la cámara ve un eje de referencia de CoppeliaSim (Figura 87). Esto, sin embargo, no es un problema serio, puesto que los ejes de referencia están solo como referencia visual y no son necesarios para el correcto funcionamiento del código.

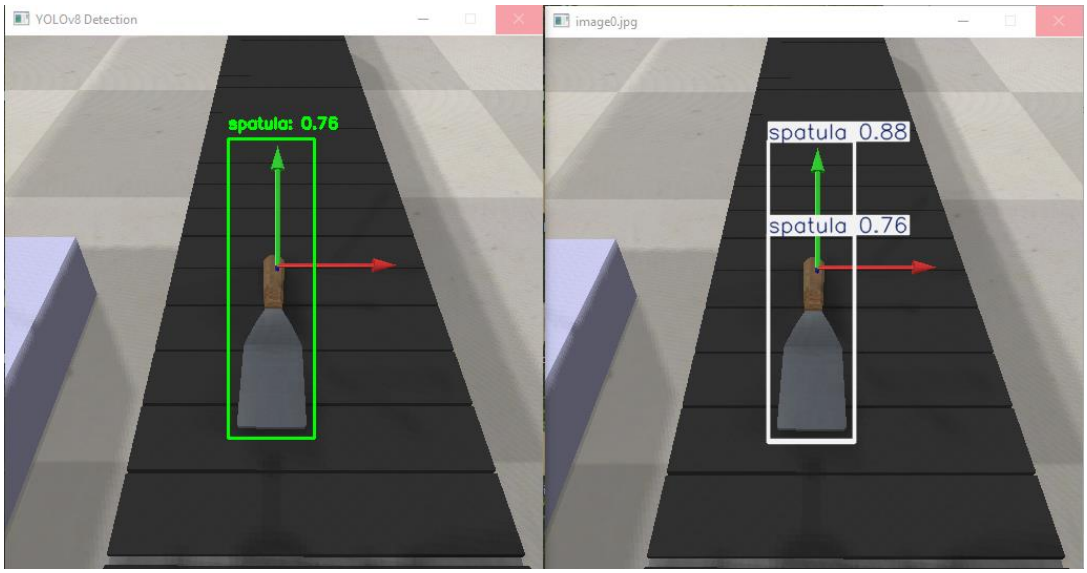


Figura 87. Ejemplo de detección errónea, YOLO detecta dos espátulas donde solo hay una.

4.2 Calculo de la posición y agarre de los objetos

Los objetos elegidos para la aplicación representan diferentes niveles de dificultad a la hora de calcular su posición y cogerlos. Hemos aplicado distintas maneras de recogida de los objetos, para representar distintas soluciones, cada una con sus ventajas y desventajas. Y vamos a asumir una serie de condiciones que se van a aplicar en cada caso.

4.2.1 Objeto: Lata

Para el objeto lata hemos importado una malla, y se le han aplicado unas texturas, dando como resultado una lata de Coca-Cola Zero (Figura 89). Las razones por las que hemos elegido este objeto son principalmente: la disponibilidad del modelo y texturas de forma gratuita y el fácil agarre que representa, debido a su simetría rotacional. En el caso de este objeto, vamos a asumir que siempre llega por la cinta transportadora y que nunca llega volcado. En aplicaciones de otros objetos vamos a ver soluciones que podrían funcionar si la lata viniese volcada. Esto se ha hecho para representar diversas soluciones con distintos grados de complejidad de implementación, y rapidez de actuación.

Figura 89. Modelo de lata de Coca-Cola que hemos usado para la simulación

En el caso de la lata, podemos ver que de manera consistente el punto que calcula, donde está el objeto, se encuentra ligeramente elevado por encima de la lata (Figura 90). Este problema no es único de este objeto, y también lo vamos a ver repetido en la espátula.

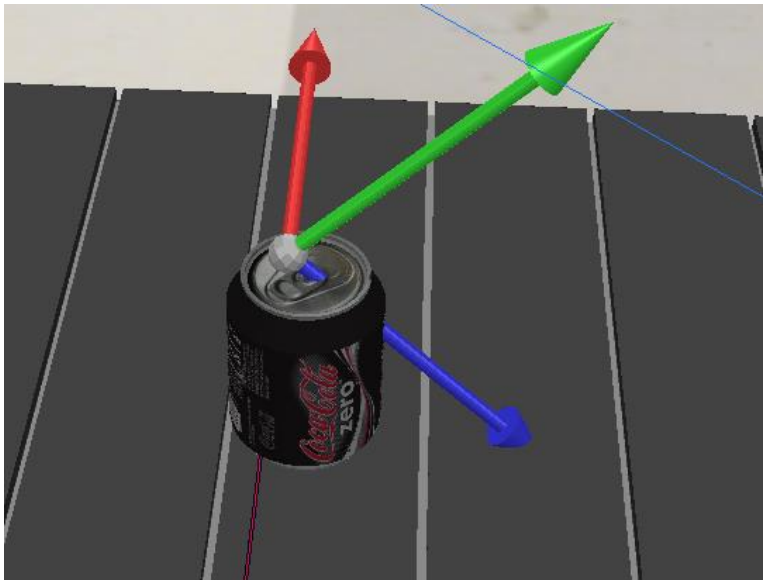


Figura 90. Error de posición en la localización del objeto lata.

Como la manera en la que consigue el punto es una media de los valores de profundidad de cada píxel que hay dentro del recuadro delimitante de YOLO, este recuadro por tanto contendrá inevitablemente algo del fondo de la imagen, y si bien ignora valores que estén muy lejos o muy cerca, al estar contando los pixeles de la cinta transportadora crea un poco de error. En la práctica, sin embargo, el error que registra es lo suficientemente consistente como para poder ser compensado sin mayor problema.

En el agarre del objeto, y por la aproximación desde arriba que debemos hacer (Figura 91), permitimos que la acción conjunta de la cinta y la pinza mueva la lata a una posición de agarre ventajosa.

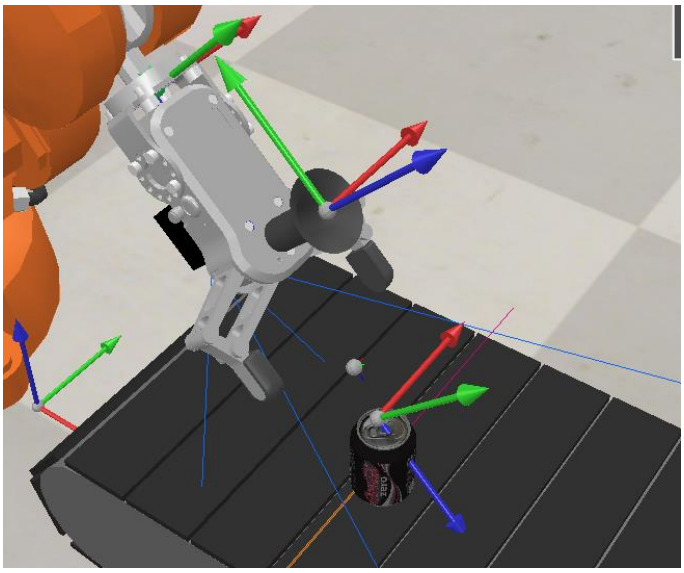


Figura 91. Aproximación de la pinza al objeto lata.

Primero, al chocar la pinza con la parte de arriba hace que la lata se tumbe ligeramente hacia un lado, por consiguiente, el sensor de la cinta ya no está activo por lo que se activa la cinta y mueve la lata hacia la dirección de la pinza consiguiendo un agarre en la parte central de la lata (Figura 92).

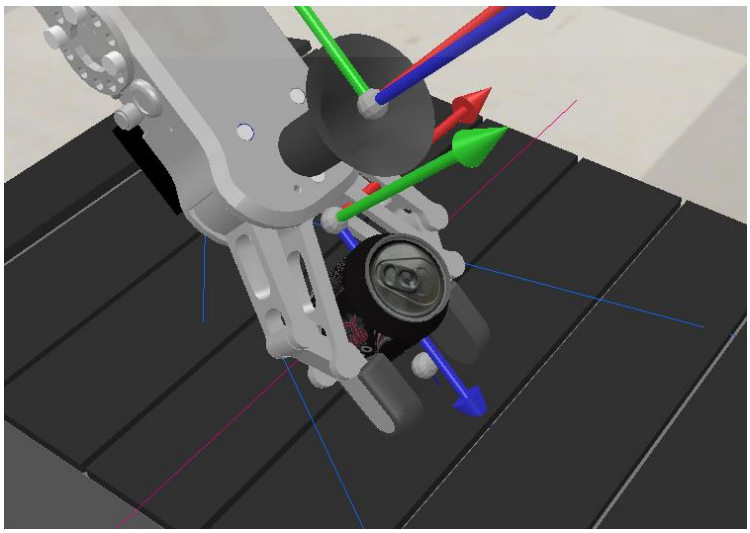


Figura 92. Agarre de la lata con la pinza

La razón por la cual tenemos que hacer esta aproximación es que, al tener tantos objetos juntos en el efector final, como la cámara y la ventosa, aparte de la pinza, reducimos mucho la movilidad que tiene al acercarse a otros objetos. Ya que no queremos dañar estas herramientas durante la aproximación. También indicar, que la elección de usar la

pinza se debe a parte de que es una solución más interesante ingenierilmente hablando, la ventosa al tener un diámetro más grande que la lata seria dudoso que pudiese crear un vacío en una situación en la vida real, y para reflejar esto, hemos decidido usar la pinza.

Finalizando, podemos asegurar que la aplicación es capaz de agarrar y dejar en un punto de manera consistente la lata (Figura 93).

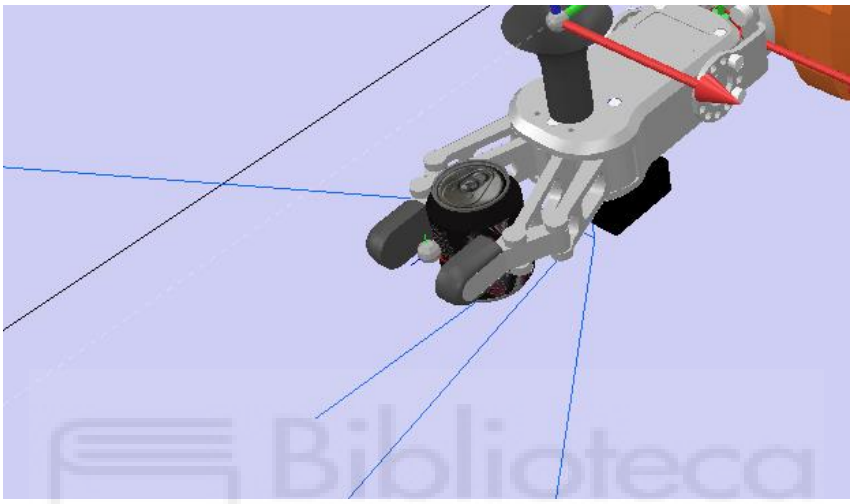


Figura 93. Una vez recogida la lata, esta es colocada en la plataforma central.

4.2.2 Objeto: Espátula

Para este objeto, hemos importado un modelo que hemos conseguido de forma gratuita y le hemos aplicado una textura que nosotros mismos hemos creado, juntando una imagen de textura metálica con otra de madera. Hemos escogido este objeto como un paso intermedio de dificultad en el agarre. El objeto muestra una simetría bilateral, con una zona de agarre clara, el mango (Figura 94). Para este objeto vamos a suponer que nos puede llegar de dos formas, en horizontal y en vertical, usamos estas suposiciones para mostrar un tipo de reconocimiento de posición, que si bien no es tan versátil como el ultimo tipo, es mucho más rápido y computacionalmente ligero. En este caso nos aprovechamos de una de las propiedades de YOLO, la caja delimitante, que encuadra al objeto que reconoce. Como sabemos las posiciones en pixeles de los vértices de este recuadro, podemos averiguar si su longitud en el eje X es mayor a su longitud en el eje Y, y viceversa. En otras palabras, sabremos si la espátula viene en posición vertical si su cuadro delimitante es más alto que ancho y vendrá en posición horizontal si el recuadro

es más ancho que alto. Hemos decidido usar la pinza la recoger este objeto, ya que sería el movimiento más análogo a lo que haría una persona, y el hecho de usar la pinza presenta ciertas complicaciones interesantes, ya que para evitar que se caiga una vez agarrada debemos programar un camino y realizarlo sin movimientos bruscos. Se podría haber utilizado la ventosa en este caso, pero la única zona de agarre en donde se podría crear un vacío es la parte metálica, que es algo pequeña, tendría que colocarse con mucha precisión, y va un poco en contra del espíritu de intentar agarrar el objeto como una persona.

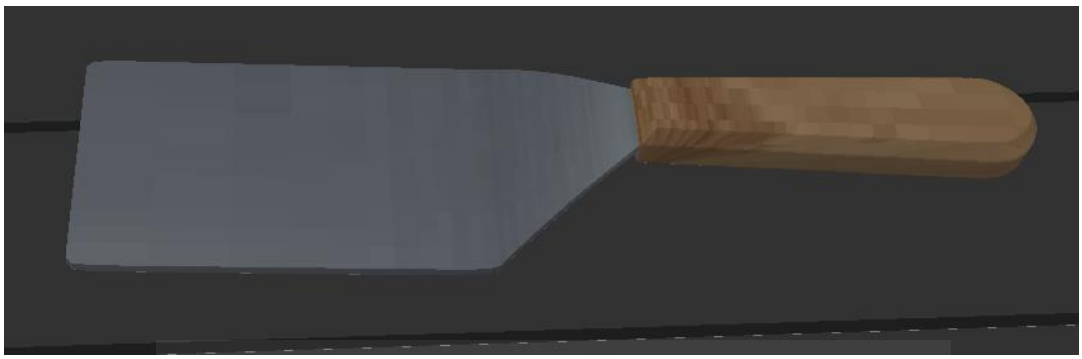


Figura 94. Modelo de la espátula utilizada.

4.2.2.1 Caso en horizontal

En este caso asumimos que la espátula viene en posición horizontal, con el mando a la izquierda desde el punto de vista del robot. Se pueden apreciar también, que el punto conseguido por el cálculo de la distancia muestra un poco de descompensación (Figura 96) sin embargo es algo bastante constante y fácilmente compensable.

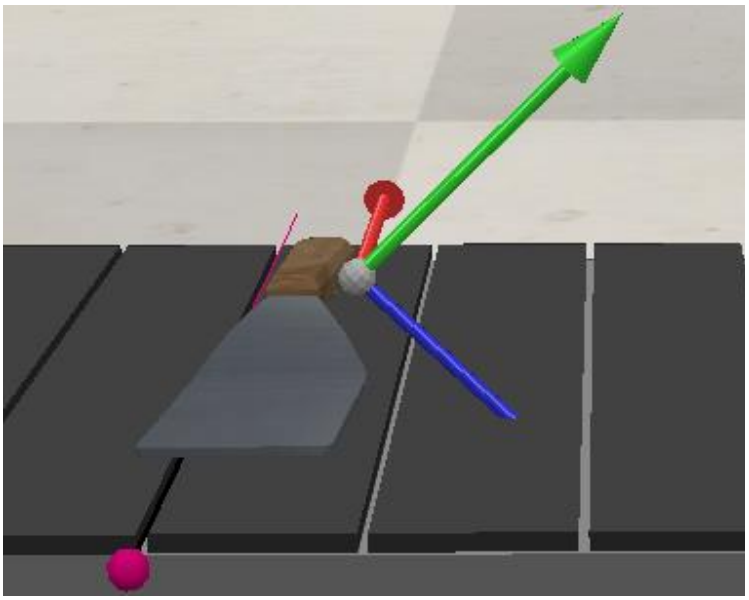


Figura 96. Offset entre el centro del objeto real y el calculo del punto para la ecuación de transformación homogénea.

A la hora de agarrar la espátula, consigue hacerlo por el mango de forma consistente (Figura 95), el mayor problema después de agarrarlo es que llegue a su punto de colocación, ya que el modelo de la espátula es algo resbaladizo y propenso a caerse durante el trayecto. Para aliviar este problema, hemos utilizado la cinemática directa, dando varios puntos al robot por donde tiene que pasar, y hemos evitado los movimientos bruscos (Figura 97).

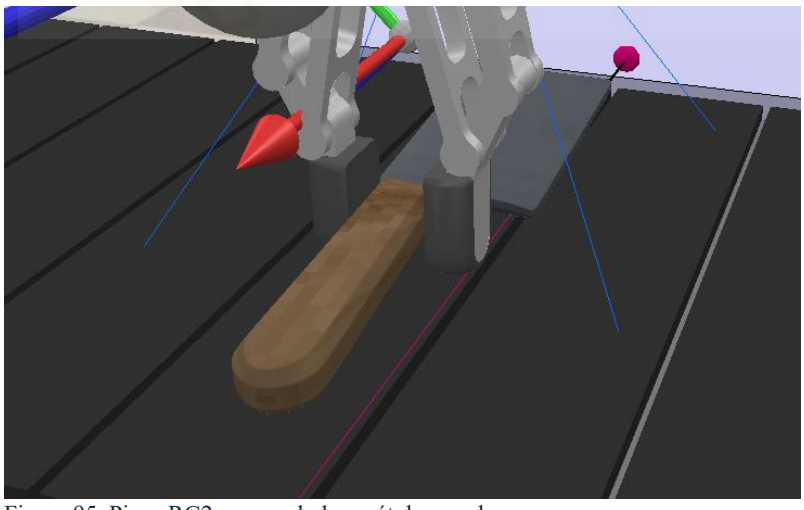


Figura 95. Pinza RG2 agarrando la espátula por el mango.

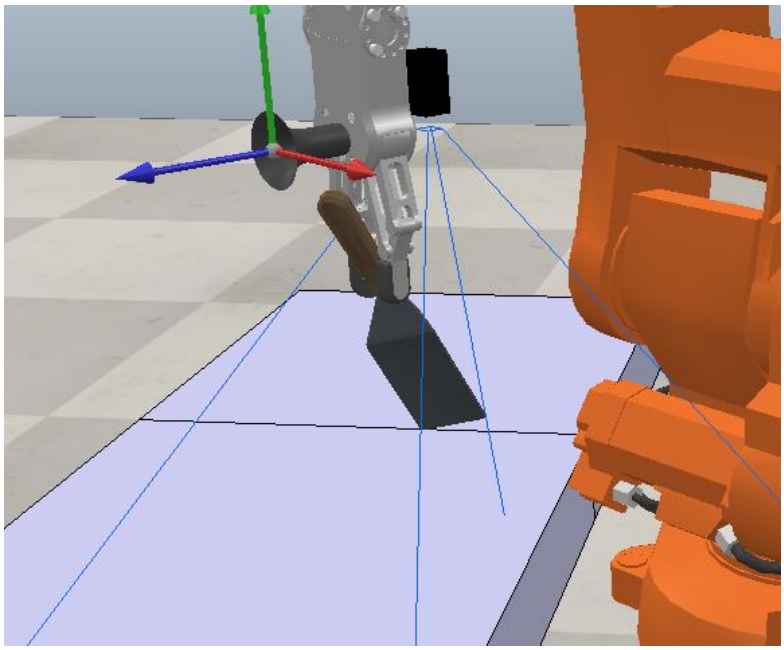


Figura 97. La espátula una vez recogida es transportada al punto de colocación, con delicadeza

4.2.2.2 Caso en vertical

En este caso, la espátula viene en posición vertical, con el mango en la parte alejada del robot (Figura 98). Esto crea un problema de descompensación interesante entre la distancia calculada y real.

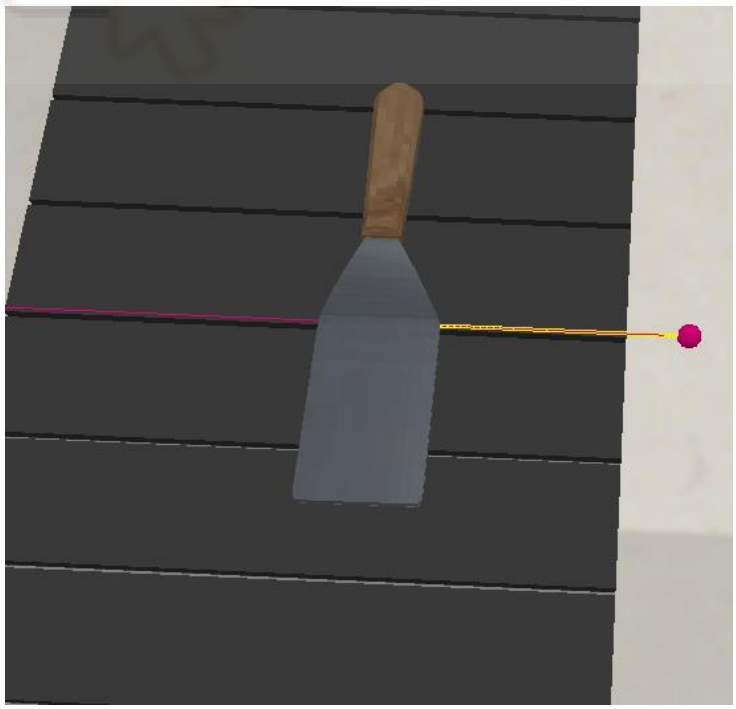


Figura 98. Espátula en posición vertical, tal y como llega por la cinta transportadora

Seguramente, porque detecta más de la cinta transportadora, y obtiene valores más alejados en comparación con el caso horizontal. Se da más a menudo el caso en que la primera aproximación para agarrar la espátula fracasa (Figura 99) y necesita de recalcular la posición y hacer otra aproximación para agarrar el objeto con éxito.

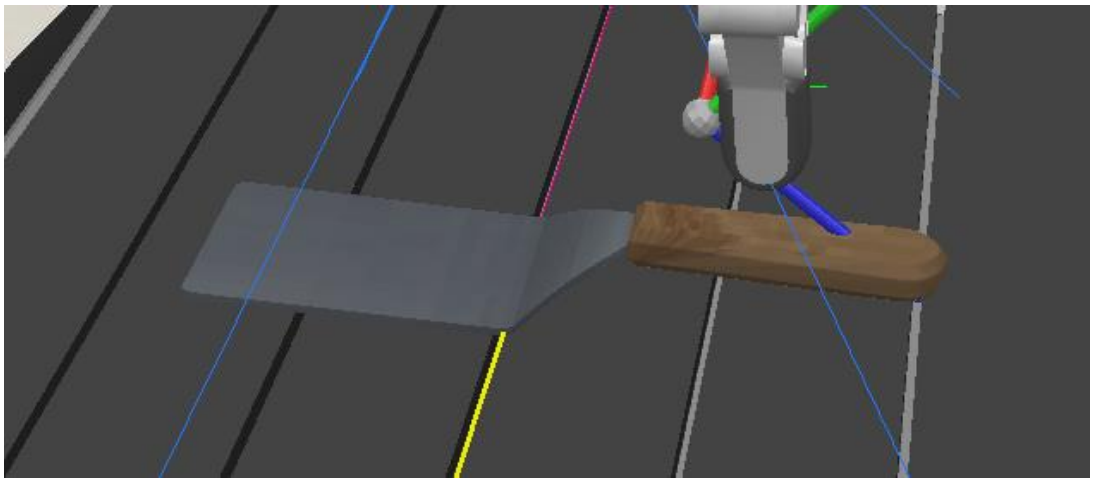


Figura 99. Intento fallido de agarre de la espátula.

Como podremos apreciar en la Figura 100, el punto ha sido recalculado después del agarre fallido y ha sido recolocado en una posición más precisa, permitiendo ahora si un agarre correcto.

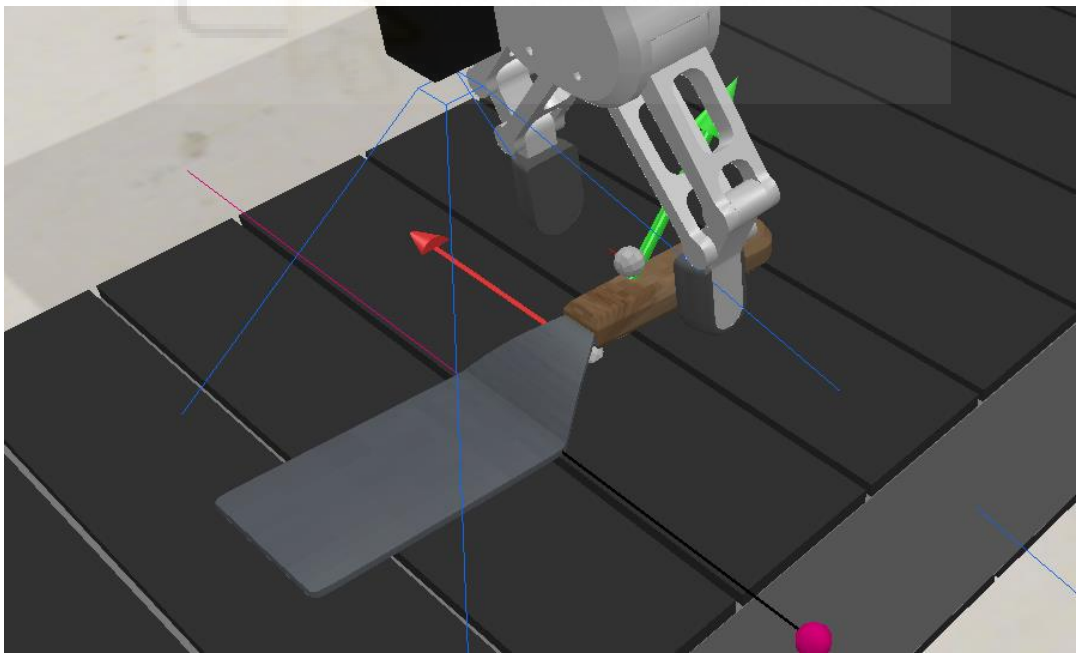


Figura 100. Agarre correcto de la espátula por parte de la pinza RG2. Nótese que el eje de referencia ha cambiado de posición, lo que significa que ha sido recalculado.

Para finalizar, movemos la espátula a su posición de colocación con delicadeza para que no caiga y la dejamos en la primera plataforma (Figura 101).

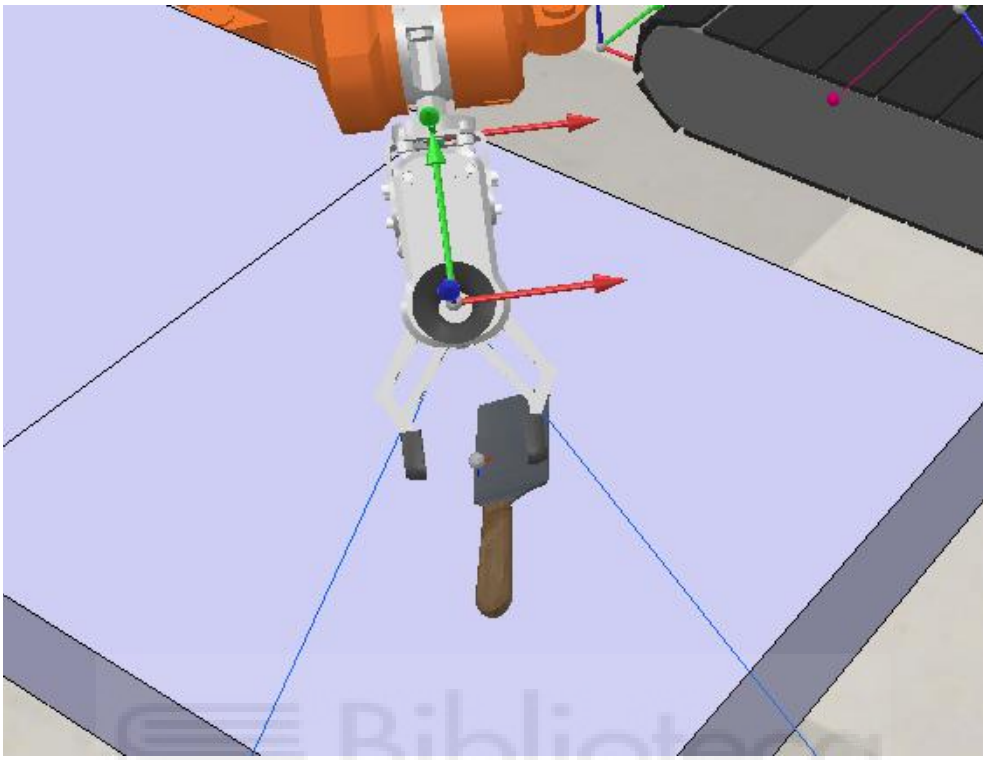


Figura 101. Espátula siendo colocada en la plataforma

4.2.3 Objeto: Ordenador portátil

El ordenador portátil es un modelo incluido por defecto en CoppeliaSim (Figura 102) y presenta una complejidad en el agarre interesante. Usar la pinza se vuelve complejo

Figura 102. Modelo de ordenador portátil de CoppeliaSim

porque por la parte de abajo no podemos agarrar el portátil ya que choca con la cinta transportadora. Y, por la parte de arriba nos arriesgaríamos a romper la pantalla si aplicamos mucha fuerza. Por lo que la mejor opción se vuelve el usar la ventosa para agarrar el portátil por la parte del teclado. Además, para esta aplicación vamos a poner a prueba el ultimo sistema de agarre.

Cuando el algoritmo de reconocimiento reconoce este tipo de objeto, creara una ruta semi circular alrededor de este para encontrar un punto óptimo de agarre (Figura 103). Esta solución para encontrar el punto de agarre optimo tiene un inconveniente y es que por las limitaciones físicas del robot no se puede realizar un círculo completo alrededor del objeto, por el mero hecho de que el robot no tiene tanto alcance y, cuando intente realizar las partes de la trayectoria más alejada, dará errores ya que son puntos inalcanzables.

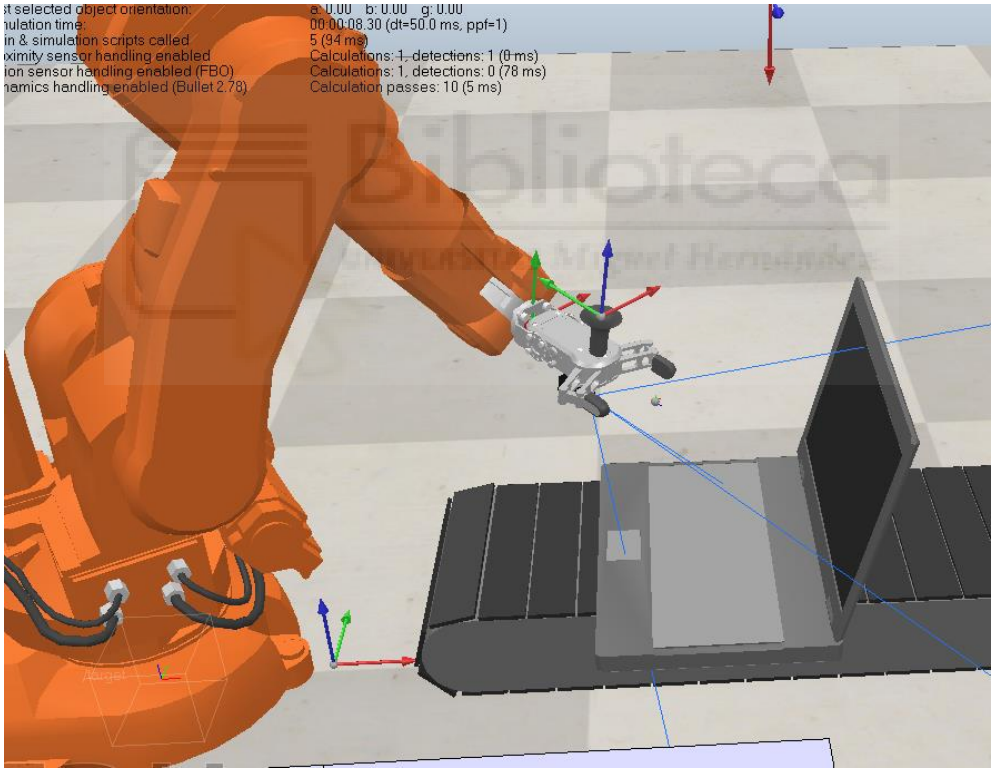


Figura 103. El robot realizando una trayectoria circular alrededor del objeto

Se podría instalar otro robot que termine de dar la vuelta, esto traería el inconveniente de sincronizar ambos brazos para que no se chocasen, pero es posible, e ingenierilmente hablando interesante. Aun así, sería un gasto económico añadido difícil de justificar en cualquier aplicación en la vida real. Por lo que vamos a asumir que siempre que llegue el

portátil nos llegara en un rango de ángulos posibles. Se podría justificar mediante un sistema de embudo, por ejemplo, como el que vemos en la Figura 105.

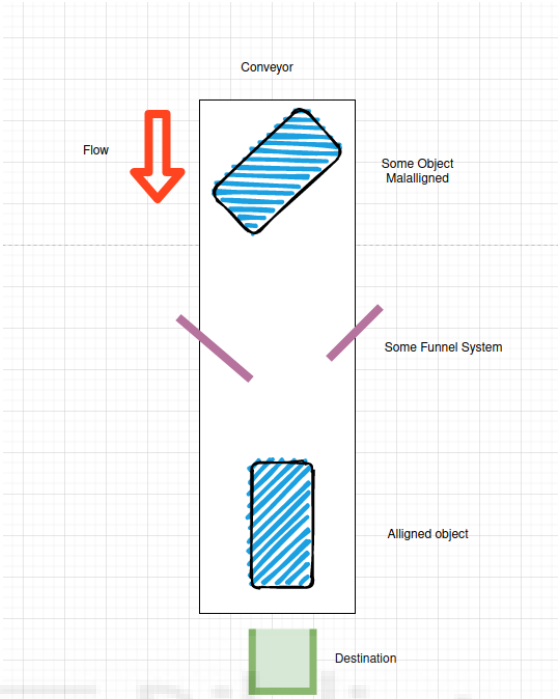


Figura 105. Ejemplo de sistema de alineamiento en una cinta transportadora

Al contrario que en otros objetos, podemos observar que aquí el cálculo de la distancia del objeto es muy preciso, esto se debe al ángulo de la cámara y altura del objeto que impide que se muestre mucho del fondo y el poco que se ve es ignorado por ser valores atípicos que se encuentran muy lejos. Por lo que la compensación que debemos hacer para agarrar el objeto una vez calculamos la distancia es mínima (Figura 104).

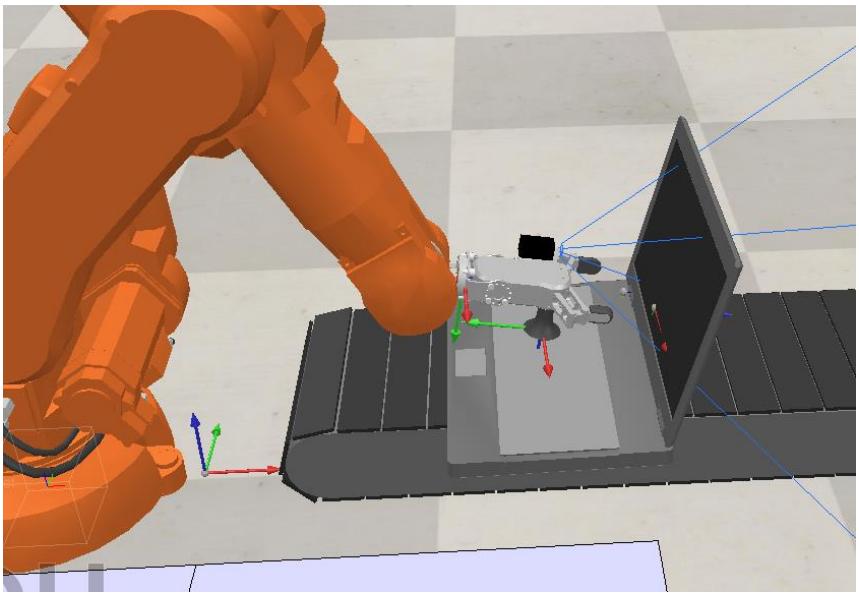


Figura 104. Agarre del portátil. Se puede observar un eje de referencia en la pantalla.

Después de agarrar el portátil con la ventosa, lo dejaremos en la ultima plataforma, como podemos ver en la Figura 106.

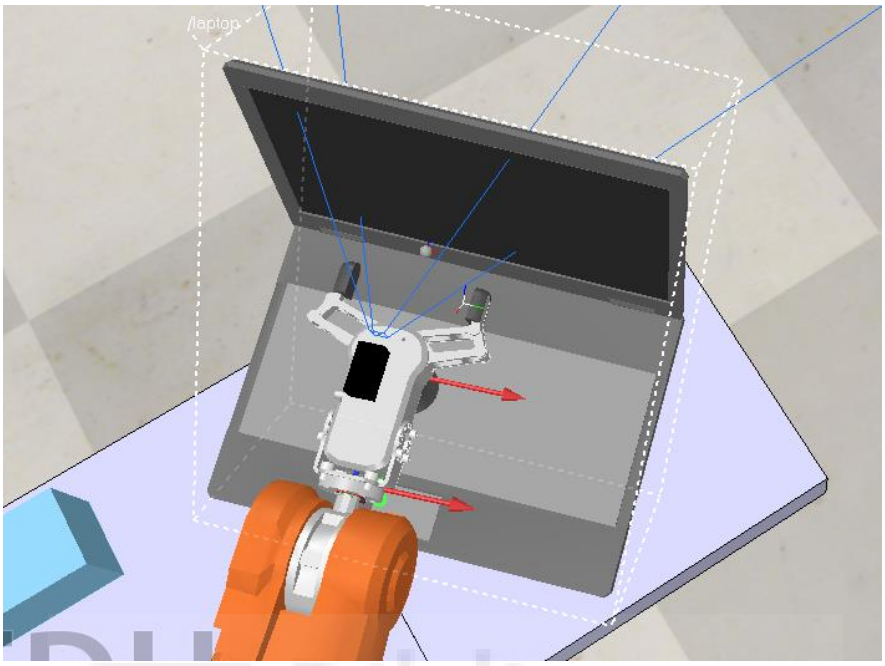


Figura 106. El portátil siendo transportado a su punto de destino

4.2.3.1 Diferencias entre los distintos métodos de cálculo de la cinemática inversa.

Se han comprobado tres métodos de cálculo del problema de la cinemática inversa que se usa para que el robot se mueva en su trayectoria circular.

El primer método es el de Moore Penrose, el cual hemos descartado por la cantidad de tiempo que tardaba en ir de un punto a otro y que muchas veces daba error.

El siguiente método es el de Moore Penrose amortiguado el cual da los resultados más rápidos, pero a la vez perdemos un poco de precisión. Lo cual hace que no consigamos alinearnos correctamente con los objetos.

El ultimo método es el de la transpuesta que consigue los segundos resultados más rápidos y una precisión aceptable para nuestra aplicación. Es por eso por lo que todas las pruebas se han hecho con este método.

Método utilizado	Tiempo tardado en calcular el punto (s)
Moore-Penrose	10
Moore-Penrose amortiguado	3.5
Método de la transpuesta	4.6

Durante el desarrollo de la aplicación se consideró en que todos los puntos de la trayectoria fueran calculados al principio y así la aplicación fuera más fluida. Se decidió finalmente abandonar esta idea después de haberla desarrollado porque, el tiempo de espera al principio era demasiado largo y podría crear confusión en el usuario, además que el tiempo de cálculo viene relacionado con la cantidad de puntos en la trayectoria que tiene que calcular. Con ese método se calculaban puntos que potencialmente no eran necesarios calcular a no ser que el punto de agarre optimo se encontrase en el último punto de la trayectoria. Por lo que realmente con el método de cálculo de punto a punto que utilizamos acabamos siendo más eficientes tanto en tiempo como en capacidad de computación.

5. Conclusiones

En este capítulo hablaremos de los resultados de la creación y simulación de la aplicación de pick and place y reconocimiento de objetos y que podría ser añadido o mejorado en el futuro.

5.1 Conclusiones generales

Durante la creación de este Trabajo de Fin de Grado han ido surgiendo complicaciones y problemáticas. Una de las preocupaciones iniciales era si era posible el usar la cámara simulada del programa y recibir una salida de datos de CoppeliaSim que pudiera ser usada con las herramientas aprendidas de la asignatura de Visión por Computador como la librería de OpenCV y el entrenamiento de algoritmos de visión usando las imágenes de CoppeliaSim. Un gran contratiempo fue el cambio del robot, que originalmente iba a ser un UR5 de Universal Robots y paso a ser un IRB140 de ABB, ya que el UR5 tenía problemas para alcanzar ciertas posiciones al no disponer de una muñeca esférica y la cinemática de la librería ARTE no funcionaba bien sin esto, para ciertos trabajos. También cabria mencionar la gran cantidad de trabajo que es preparar un conjunto de datos para entrenar a YOLO en reconocimiento de objetos. Para realizarlo se produjo un pequeño script de Python para tomar fotos de los objetos desde varios ángulos, y aunque ayudaba, no se podía automatizar el proceso en gran medida. Cabe añadir que para el entrenamiento de YOLO hay que delimitar el objeto en las imágenes y marcarlo correctamente que es también un proceso arduo. Se usó la página de CVAT o (Computer Vision Annotation Tool) para realizar el proceso. Y al principio el entrenamiento no surgió del todo correcto pues se subestimo la cantidad de imágenes necesarias para entrenar al sistema. Aun con todo podemos señalar que los objetivos que nos dispusimos a realizar han sido completados con mayor o menor éxito. Vamos a entrar en más detalle a continuación.

5.1.1 Conclusiones de los objetivos de aprendizaje

Por un lado, se ha investigado y aprendido las capacidades y características de dos robots, y una pinza. El UR5 y el IRB140, y la pinza RG2. Además, hemos estudiado y comprendido CoppeliaSim y su funcionamiento interno, y como se conecta con la API

externa para ejecutar scripts. Por otra parte, hemos investigado a fondo las partes de la librería PyArte necesarias para la aplicación. En algunos casos hemos utilizado scripts planteados como problemas de la asignatura o el máster de robótica, modificándolos para que sean compatibles o utilizándolos tal cual. Por ejemplo, "robots.camera" es necesario modificarlo para conseguir la señal de video, pero la función de cinemática inversa y sus funciones cinemáticas son las usadas en la solución de un problema de máster. Tras todas las tribulaciones con la librería, creo tener una decente comprensión de cómo funciona. Por último, aunque las bases de la visión por computador no me eran desconocidas gracias a la asignatura del mismo nombre, ha sido la primera vez que he usado YOLO y creo haber obtenido una decente comprensión tanto de su utilización como su teoría de funcionamiento.

5.1.2 Conclusiones de los objetivos de simulación

Repasando los objetivos de simulación, hemos conseguido generar una simulación del entorno de trabajo en CoppeliaSim, por ejemplo, no solo añadiendo objetos que están dentro del programa sino también importando modelos y texturas. Somos capaces de controlar el movimiento del robot y tenemos una cámara que capta las imágenes dentro de la simulación y nos manda la señal al script de Python. Originalmente como prueba de concepto se diseñó un simple programa a partir del programa de la librería PyArte "move_robot" el cual permite mover las articulaciones del robot con el teclado. Juntamos esto con un componente de visión por computador de OpenCV y comprobamos que era posible recibir imagen y moverse a la vez. Gracias a poder mover el robot y conseguir video o imágenes es como conseguimos un conjunto de imágenes para entrenar a YOLO. Si bien el modelo resultante del entrenamiento es factible, lo categorizaría de un acierto parcial. Ya sea por falta de poder computacional o un conjunto de datos no lo suficientemente grande, aunque funciona, suele dar algunos falsos positivos y le cuesta reconocer algunos objetos hasta que no están cerca, por caso, la lata de Coca-Cola que seguramente al tener una textura oscura es posible que al modelo le cueste ver cuando tiene un fondo oscuro detrás.

Por otro lado, gracias a la imagen de profundidad de CoppeliaSim y a YOLO podemos reconocer varios objetos y saber su posición, y por consiguiente podemos decidir cómo se agarran y donde se colocan después. Si se trata de una lata, conseguir su posición optima de agarre es fácil ya que es un objeto con una simetría axial y por tanto da igual la posición por la que se aproxime el robot. Pero, para otros objetos, podemos usar la

trayectoria circular y el algoritmo RANSAC para hallar la posición optima de agarre. La posición óptima para cada objeto se consigue de forma empírica y si bien no es la manera más eficiente de hacerlo se puede aprovechar el proceso para conseguir imágenes para el conjunto de datos de entrenamiento. Con todo esto dicho considero que se ha cumplido el objetivo de controlar el movimiento del robot con la información obtenida a través de la cámara y los algoritmos de reconocimiento de objetos y también de encontrar la posición óptima de agarre.

Una vez agarrado el objeto, el robot es capaz de llegar a otra posición y soltarlo dejándolo en un sitio. Si bien es cierto que es mejorable, RANSAC no va a funcionar con cualquier objeto, y una vez tuviéramos agarrado el objeto sería interesante que tuviera alguna capacidad de "path planning" para no chocar con posibles obstáculos. Entraremos en más detalle de posibles mejoras en el siguiente apartado. Pero para ir finalizando se han logrado los objetivos de todos los apartados de forma satisfactoria en su mayoría, y como mínimo se ha desarrollado un núcleo bastante robusto de aplicación de "pick and place"

5.2 Posibles futuros desarrollos

En lo que respecta a posibles ramas de trabajo que pueden derivarse del proyecto, podríamos comentar las siguientes:

Para empezar el desarrollo de un conjunto de datos de imágenes con objetos múltiples que puedan ser utilizados para entrenar a un modelo. Sería interesante encontrar una manera de automatizar el proceso, si bien la categorización de las imágenes sigue necesitando de trabajo manual.

Por las limitaciones físicas del robot, es imposible realizar una trayectoria circular completa alrededor del objeto para encontrar su punto de agarre optimo, y se podría dar el caso de que por la posición en la que viene el objeto y la limitación del arco que se realiza, para hallar con el punto óptimo de agarre, no se consiga obtener ese punto. Podrían existir múltiples soluciones para este problema, para mantener el espíritu del proyecto sin embargo se preferiría una solución que cueste lo mínimo.

El algoritmo RANSAC funciona con objetos que presenten de ciertas superficies rectas. Se podría dar el caso de que haya un tipo de objeto que no presente de superficies rectas ni de simetría radial y que encontrar el punto óptimo de agarre sea complicado. Un posible trabajo seria la implementación de otro algoritmo o quizá usar el propio algoritmo de reconocimiento de objetos para encontrar la rotación del objeto.

Como hemos mencionado antes, ya que disponemos de una cámara en la muñeca, sería interesante añadir un componente de evasión de obstáculos a la hora de que se mueva el robot, esto podría ser una manera de añadir seguridad al sistema.

Otro trabajo interesante sería el comprobar si este sistema se puede traducir bien a un caso en la vida real. En el caso del algoritmo de reconocimiento de objetos se podría comprobar si el entrenamiento con imágenes virtuales es suficiente para que el modelo funcione con un input de una cámara real. Por otra parte, se podría intentar montar y programar el sistema simulado en la vida real. Es decir, montar el robot con todos los accesorios, hacer que realice las operaciones de pick and place en la vida real y analizar cómo funciona.

6. Anexos

6.1 Enlace con el programa y escenas

https://drive.google.com/drive/folders/1oOZt5orSbXy0hU5RfvwQGqeJB_4vSVZy?usp=sharing

6.2 Hoja de características del robot IRB140

IRB 140

Industrial Robot

MAIN APPLICATIONS
Arc welding
Assembly
Cleaning/Spraying
Machine tending
Material handling
Packing
Deburring

Small, Powerful and Fast

Compact, powerful IRB 140 industrial robot. Six axis multipurpose robot comprising IRB 140 manipulator and S4Cplus industrial robot controller. Handles payload of 5kg, with long reach (810 mm) of axis 5, optional floor, wall and suspended mounting. Available as Standard, Foundry, Clean Room and Wash versions, all mechanical arms completely IP67 protected, making IRB 140 easy to integrate in and suitable for a variety of applications. Uniquely extended radius of working area due to bend-back mechanism of upper arm, axis 1 rotation of 360 degrees and flexible mounting capabilities.

The compact, robust design with integrated cabling adds to overall flexibility. The Collision Detection option with full path retraction makes robot reliable and safe.

Using IRB 140T, cycle-times are considerably reduced where axis 1 and 2 predominantly are used.

Reductions between 15-20 % are possible using pure axis 1 and 2 movements.

This faster versions is well suited for packing applications and guided operations together with PickMaster.

IRB Foundry Plus and Wash versions are suitable for operating in extreme foundry environments and other harch environments with high requirements on corrosion resistance and tightness. In addition to the IP67 protection, excellent surface treatment makes the robot high pressure steam washable. The white-finish Clean Room version meets Clean Room class 10 regulations, making it especially suited for environments with stringent cleanliness standards.

The S4Cplus controller has the electronics for controlling the robot manipulator, external axes and peripheral equipment. S4Cplus also contains system software with all basic functions for operating and programming, including two built-in Ethernet channels with 100 Mbit/s capacity. This ensures a significant increase in computing power as well as improved controller monitoring and supervision.

ABB

IRB 140

Industrial Robot

TECHNICAL DATA, IRB 140 INDUSTRIAL ROBOT

SPECIFICATION

Robot versions	Handling capacity	Reach of 5th axis	Remarks	Standard colour
IRB 140/IRB 140T	5 kg	810 mm		Orange
IRB 140F/IRB 140TF	5 kg	810 mm	Foundry Plus Protection	Orange
IRB 140CR/IRB 140TCR	5 kg	810 mm	Clean Room,	White
IRB 140W/IRB 140TW	5 kg	810 mm	Wash Protection	White

Supplementary load (on upper arm alt. wrist)	
on upper arm	1 kg
on wrist	0.5 kg
Number of axes	
Robot manipulator	6
External devices	6
Integrated signal supply	12 signals on upper arm
Integrated air supply	Max. 8 bar on upper arm

PERFORMANCE

Position repeatability	0.03 mm (average result from ISO test)
Axis movement	
Axis	Working range
1, C Rotation	360°
2, B Arm	200°
3, A Arm	280°
4, D Wrist	Unlimited (400° default)
5, E Bend	240°
6, P Turn	Unlimited (800° default)
Max. TCP velocity	2.5 m/s
Max. TCP acceleration	20 m/s²
Acceleration time 0-1 m/s	0.15 sec

VELOCITY

Axis no.	IRB 140	IRB 140T
1	200°/s	250°/s
2	200°/s	250°/s
3	260°/s	260°/s
4	360°/s	360°/s
5	360°/s	360°/s
6	450°/s	450°/s

CYCLE TIME

5 kg Picking side	IRB 140	IRB 140T
cycle 25 x 300 x 25 mm	0,85s	0,77s

ELECTRICAL CONNECTIONS

Supply voltage	200–600 V, 50/60 Hz
Rated power	
Transformer rating	4.5 kVA

PHYSICAL

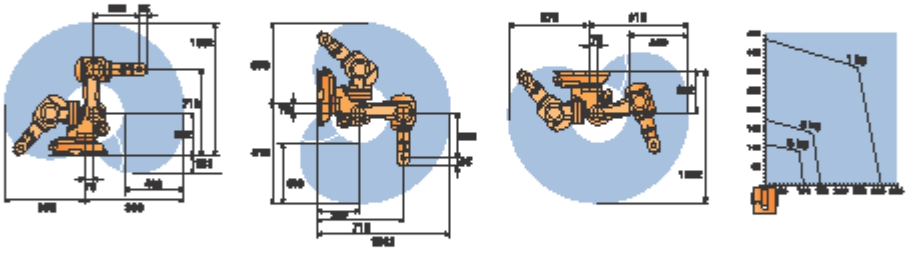
Robot mounting	Any angle
Dimensions	
Robot base	400 x 450 mm
Robot controller H x W x D	950 x 800 x 620 mm
Weight	
Robot manipulator	98 kg
Robot controller	250 kg

ENVIRONMENT

Ambient temperature	
Robot manipulator	5 – 45°C
Robot controller	5 – 52°C
Relative humidity	Max. 95%
Degree of protection, Manipulator	IP67
Foundry/Wash	High pressure steam washable
Clean Room	Class 10 (Federal Standard) class 4 (ISO)
Noise level	Max. 70 dB (A)
Safety	Double circuits with supervision, emergency stops and safety functions, 3-position enable device
Emission	EMC/EMI-shielded

Data and dimensions may be changed without notice

WORKING RANGE AND LOAD DIAGRAM



PR1003 1EN_IRB September 2003. Produced by ABB Västerås Sweden.

www.abb.com/robotics

6.3 Hoja de características de la pinza RG2

RG2 GRIPPER DATASHEET

KEY FEATURES

- ✓ No external cables
- ✓ Adjustable gripper mounting
- ✓ Gripping feedback
- ✓ Automatic payload and TPC calculation
- ✓ Depth compensation
- ✓ Customizable fingertips

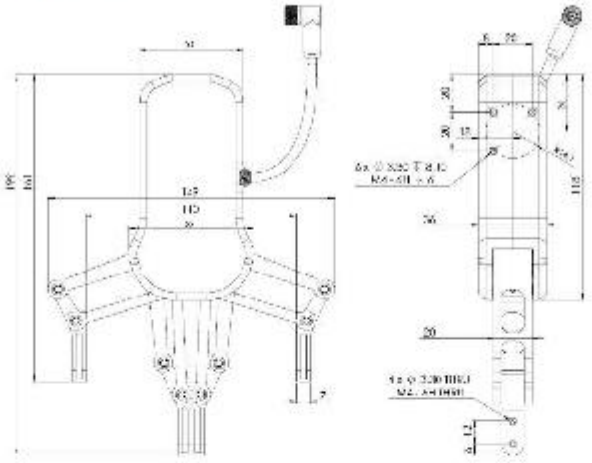
TECHNICAL SPECIFICATIONS

Technical data	Min	Typical	Max	Units
Total stroke (adjustable)	0	-	110	[mm]
Finger position resolution	-	0,1	-	[mm]
Repetition accuracy	-	0,1	0,2	[mm]
Reversing backlash	0,2	0,4	0,6	[mm]
Gripping force (adjustable)	3	-	40	[N]
Gripping force accuracy	± 0,05	± 1	± 2	[N]
Operating voltage*	10	24	26	[V DC]
Power consumption	1,9	-	14,4	[W]
Maximum Current	25	-	600	[mA]
Ambient operating temperature	5	-	50	[°C]
Storage temperature	0	-	60	[°C]
Product weight	-	0,65	-	[kg]

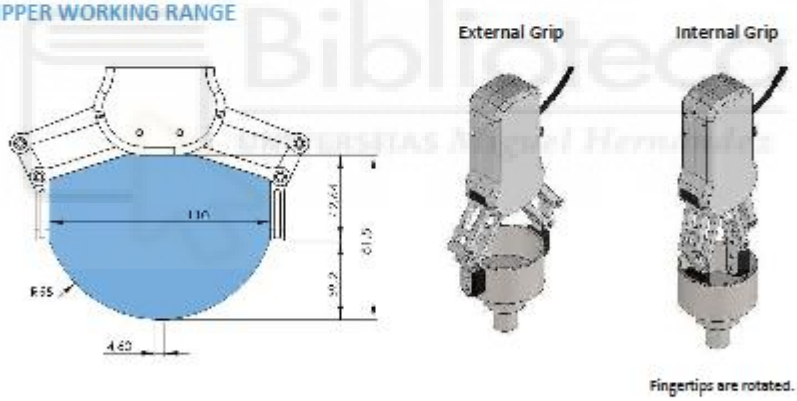
*At 12V the gripper runs at approximately half the normal speed

Version 1.0

MECHANICAL DIMENSIONS



GRIPPER WORKING RANGE



Fingertips are rotated.

TOOL CONNECTOR PINOUT

M8x1 cable, 8-pos

Pin	Wire	UR I/O	UR I/O V3
1	White	AI2	Tool analog input 2
2	Brown	AI3	Tool analog input 3
3	Green	DI9	Tool input 1
4	Yellow	DI8	Tool input 0
5	Gray	Power	24V DC
6	Pink	DO9	Tool output 1
7	Blue	DO8	Tool output 0
8	Red	GND	0V DC

6.4 Hoja de características de la ventosa VGC10

DATASHEET

1. Datasheet

1.1. VGC10

General Properties		Minimum	Typical	Maximum	Unit
Vacuum		5 % -0.05 1.5	- - -	80 % -0.810 24	[Vacuum] [Bar] [inHg]
Air flow		0	-	12	[L/min]
Payload	With default attachments	- -	- -	6 * 13.2 *	[kg] [lb]
	With customized attachments	- -	10 20	15 33.1	[kg] [lb]
Vacuum cups		1	-	7	[pcs.]
Gripping time		-	0.35	-	[s]
Releasing time		-	0.20	-	[s]
Vacuum pump		Integrated, electric BLDC			
Dust filters		Integrated 50µm, field replaceable			
IP Classification		IP54			
Dimensions		101 x 100 x 100 3.97 x 3.94 x 3.94			[mm] [inch]
Weight		0.814 1.79			[kg] [lb]

* By using three 40mm cups. More info in the table Number of Cups needed for non-porous materials depending on payload and vacuum.

Operating Conditions	Minimum	Typical	Maximum	Unit
Power supply	20.4	24	28.8	[V]
Current consumption	50	600	1500	[mA]
Operating temperature (gripper and vacuum cups)	0 32	- -	50 122	[°C] [°F]
Relative humidity (non-condensing)	0	-	95	[%]
Calculated operation life	30 000	-	-	[Hours]

2 Channels

The VGC10 has 4 holes to use fittings with vacuum cups or blinding screws as needed. It also has lines which show the holes that are communicated together. This is useful when using channels A and B independently for vacuum.

Payload

The lifting capacity of the VG grippers depends primarily on the following parameters:

- Vacuum cups
- Vacuum
- Air flow

Vacuum Cups

Choosing the right vacuum cups for your application is essential. The VG grippers come with common 15, 30 and 40 mm silicone vacuum cups (see table below) which are good for hard and flat surfaces, but not good for uneven surfaces and it might leave microscopic traces of silicone on the workpiece which can cause issues with some types of painting processes afterwards.

Image	External Diameter [mm]	Internal Diameter [mm]	Gripping Area [mm2]
	15	6	29
	30	16	200
	40	24	450

For non-porous materials, the OnRobot suction cups are highly recommended. Some of the most common non-porous materials are listed below:

- Composites
- Glass
- High density cardboard
- High density paper
- Metals
- Plastic
- Porous materials with a sealed surface
- Varnished wood

In an ideal case, working with non-porous material workpieces where there are no air flow going through the workpiece, the table below shows the number of cups and the cup size needed depending on the payload (workpiece mass) and the vacuum used.

Number of Cups needed for non-porous materials depending on payload and vacuum :

	 15 mm				 30 mm				 40 mm			
Payload (kg)	Vacuum (kPa)				Vacuum (kPa)				Vacuum (kPa)			
	20	40	60	75	20	40	60	75	20	40	60	75
0.1	3	2	1	1	1	1	1	1	1	1	1	1
0.5	13	7	5	4	2	1	1	1	1	1	1	1
1	-	13	9	7	4	2	2	1	2	1	1	1
2	-	-	-	14	8	4	3	2	4	2	2	1
3	-	-	-	-	12	6	4	3	5	3	2	2
4	-	-	-	-	15	8	5	4	7	4	3	2
5	-	-	-	-	-	10	7	5	9	5	3	3
6	-	-	-	-	-	12	8	6	10	5	4	3
7	-	-	-	-	-	13	9	7	12	6	4	4
8	-	-	-	-	-	15	10	8	14	7	5	4
9	-	-	-	-	-	-	12	9	15	8	5	4
10	-	-	-	-	-	-	13	10	-	9	6	5
11	-	-	-	-	-	-	14	11	-	9	6	5
12	-	-	-	-	-	-	15	12	-	10	7	6
13	-	-	-	-	-	-	16	13	-	11	8	6
14	-	-	-	-	-	-	-	14	-	12	8	7
15	-	-	-	-	-	-	-	15	-	13	9	7

NOTE:

To use more than 7 (15mm), 4 (30mm) or 3 (40mm) vacuum cups with the VGC10 a customized adaptor plate is needed.

The table above is created with the following formula that equalizes the lifting force with the payload considering 1.5G of acceleration.

$$Amount_{Cups} * Area_{Cup}[mm] = 14700 \frac{Payload\ [kg]}{Vacuum\ [kPa]}$$

It is often a good idea to use more vacuum cups than needed, to accommodate for vibrations, leaks and other unexpected conditions. However, the more vacuum cups, the more air leakage (air flow) is expected and the more air is moved in a grip resulting in longer gripping times.

When using porous materials, the vacuum that can be achieve by using the OnRobot suction cups will depend on the material itself and will be between the range stated in the specifications. Some of the most common non-porous materials are listed below:

- Fabrics
- Foam
- Foam with open cells
- Low density cardboard
- Low density paper
- Perforated materials
- Untreated wood

See the table below with general recommendations, in case other suction cups are needed for specific materials.

Workpiece surface	Vacuum cup shape	Vacuum cup material
Hard and flat	Normal or dual lip	Silicone or NBR
Soft plastic or plastic bag	Special plastic bag type	Special plastic bag type
Hard but curved or uneven	Thin dual lip	Silicone or soft NBR
To be painted afterwards	Any type	NBR only
Varying heights	1.5 or more bevels	Any type

NOTE:

It is recommended to consult a vacuum cup specialist to find the optimal vacuum cup where the standard types are insufficient.

Suction Cups for Foil and Bags Ø25

This suction cup improves the vacuum gripper's ability to pick and place workpieces with surface of foil, thin paper, and plastic bags during irregular and angular arm movement.

6.5 Hoja de características de la pinza hibrida del Grupo Zimmer

5000 PRODUCT RANGE
GPP/GPD5000IL SERIES

THE BEST OF BOTH WORLDS

The new 5000 premium gripper series has been expanded through the addition of a pneumatic-electric hybrid gripper. This new variant keeps the pneumatic-electric option open for the 5000 gripper series. It is the ideal solution for users who want to leap into the world of Industrie 4.0 gripping while still incorporating a highly reliable pneumatic drive unit into their processes.

The IL variant is equipped with an integrated pneumatic valve, which is controlled via IO-Link. Since there is no longer a fixed hose connection between the valve and piston that needs to be filled or emptied for each cycle, the grippers' reaction time is greatly reduced. This means that they are substantially faster than other pneumatic grippers and, as such, more than live up to their standard as the premier grippers on the market today.

Hybrid grippers help you save in three ways. Firstly, they feature a simple connection concept with an air hose and connection cable. This eliminates the need to use the valve terminal that is typically required. Secondly, they are substantially more energy efficient than all other conventional grippers thanks to an integrated valve and a state-of-the-art integrated control system. The third aspect is significant savings thanks to cost-efficient predictive maintenance, therefore leading to enhanced cost efficiency and production availability.

YOUR BENEFITS
IN DETAIL

GPP5000IL SERIES

► Technical data*

Order No.	GPP5006N-IL-10-A	GPP5006NC-IL-10-A	GPP5006NO-IL-10-A	GPP5006S-IL-10-A	GPP5006SC-IL-10-A	GPP5006SO-IL-10-A
Stroke per gripper jaw [mm]	6	6	6	3	3	3
Closing/opening grip force [N]	330/360	455/–	–/485	740/800	1020/–	–/1080
Protect. class per IEC 60529	IP64	IP64	IP64	IP64	IP64	IP64
Weight [kg]						
Order No.	**GPP5008N-IL-10-A**	**GPP5008NC-IL-10-A**	**GPP5008NO-IL-10-A**	**GPP5008S-IL-10-A**	**GPP5008SC-IL-10-A**	**GPP5008SO-IL-10-A**
Stroke per gripper jaw [mm]	8	8	8	4	4	4
Closing/opening grip force [N]	620/660	710/–	–/760	1160/1240	1680/–	–/1670
Protect. class per IEC 60529	IP64	IP64	IP64	IP64	IP64	IP64
Weight [kg]						
Order No.	**GPP5010N-IL-10-A**	**GPP5010NC-IL-10-A**	**GPP5010NO-IL-10-A**	**GPP5010S-IL-10-A**	**GPP5010SC-IL-10-A**	**GPP5010SO-IL-10-A**
Stroke per gripper jaw [mm]	10	10	10	5	5	5
Closing/opening grip force [N]	885/945	1260/–	–/1320	1940/2080	2760/–	–/2890
Protect. class per IEC 60529	IP64	IP64	IP64	IP64	IP64	IP64
Weight [kg]						

GPD5000IL SERIES

► Technical data*

Order No.	GPD5006N-IL-10-A	GPD5006NC-IL-10-A	GPD5006NO-IL-10-A	GPD5006S-IL-10-A	GPD5006SC-IL-10-A	GPD5006SO-IL-10-A
Stroke per gripper jaw [mm]	6	6	6	3	3	3
Closing/opening grip force [N]	740/800	1020/–	–/1080	1620/1760	2240/–	–/2370
Protect. class per IEC 60529	IP64	IP64	IP64	IP64	IP64	IP64
Weight [kg]	0.48	0.68	0.68	0.48	0.68	0.68
Order No.	**GPD5008N-IL-10-A**	**GPD5008NC-IL-10-A**	**GPD5008NO-IL-10-A**	**GPD5008S-IL-10-A**	**GPD5008SC-IL-10-A**	**GPD5008SO-IL-10-A**
Stroke per gripper jaw [mm]	8	8	8	4	4	4
Closing/opening grip force [N]	1260/1340	1690/–	–/1770	2780/2960	3730/–	–/3910
Protect. class per IEC 60529	IP64	IP64	IP64	IP64	IP64	IP64
Weight [kg]	0.83	1	1	0.83	1	1
Order No.	**GPD5010N-IL-10-A**	**GPD5010NC-IL-10-A**	**GPD5010NO-IL-10-A**	**GPD5010S-IL-10-A**	**GPD5010SC-IL-10-A**	**GPD5010SO-IL-10-A**
Stroke per gripper jaw [mm]	10	10	10	5	5	5
Closing/opening grip force [N]	2290/2400	3140/–	–/3260	5060/5280	6930/–	–/7160
Protect. class per IEC 60529	IP64	IP64	IP64	IP64	IP64	IP64
Weight [kg]	1.46	1.9	1.9	1.46	1.9	1.9

* All values measured at 6 bar

6.6 Hoja de características de la cámara Eyes

DATASHEET

1. Datasheet

1.1. Hardware Version

Components	Version
Eyes (camera)	v2.0
Eye Box	v1.4
Eyes Lighting Kit	v1.0

1.2. Eyes

Eyes

Camera Characteristics			Unit
Interface		USB-C 3.x	
Image Sensor Technology		Rolling Shutter. Size 1.4 x 1.4	[µm px]
RGB Camera Field of View (FOV)		69.4 x 42.5 x 77 (+/- 3)	[°]
RGB Camera Resolution	Standard	1280x720	[px]
	Close-up	1920x1080	[px]
Depth Technology		Active IR Stereo	
Depth FOV		65±2 x 40±1 x 72±2	[°]
Depth Output Resolution		1280 x 720	[px]
Working Distance		400-1000 15.75 - 39.37	[mm] [inch]
Operating Temperature		0 – 35 32 – 95	[°C] [°F]
IP Rating		IP 54	
Weight		0.260 0.57	[kg] [lb]
Calculated operation life		30 000	[h]

Eyes Features		Unit
Type of vision system	2.5 D	
Minimum workpiece size	10x10 or 15 diameter 0.39x0.39 or 0.59 diameter	[mm] [inch]

2

Eyes Features					Unit
Applications Supported	Detection, Sorting, Inspection, Landmark				
Mounting options supported	Robot and External				
Reconfigurability when Robot mounted	12 configurations (4 x 3)				
	Around robot's flange		Tilt orientations		
	0 - 90 - 180 - 270		0 - 45 - 90		[degrees]
	Processing time		Typical: 0.5 s		
Detection Repeatability	< 2 < 0.078				[mm] [inch]
Detection Accuracy (typical) measured at 500 mm	External Mount		Robot Mount		
	2 0.078		2 0.078		[mm] [inch]
Minimum Inspection Defect Size	Standard		Close-up		
	5 0.197		3 0.118		[mm] [inch]
Landmark accuracy**	Waypoint distance from Landmark	Minimum error	Typical error	Maximum error	
	200 7.874	0.2635 0.0104	0.6596 0.0260	0.9500 0.0374	[mm] [inch]
	500 19.68	0.6586 0.0259	1.6490 0.0649	2.3750 0.0935	[mm] [inch]
	1000 39.37	1.3173 0.0519	3.2981 0.1298	4.7500 0.1870	[mm] [inch]

** Depending on the distance from the waypoint (picking point) to Landmark. Obtained using dual capture approach with the camera being 300 mm (11.81 in) above the Landmark.

Application and set-up recommendations	
Lighting conditions	No drastic, instant changes
Reflections and focused light spots	Keep minimal
Characteristics of objects	Different from background
Camera with respect to workspace table	Looking straight to it

Eyes Lighting Kit

Eyes Lighting Kit Features		Unit
Input voltage	24	[V]
Maximum current	1	[A]
Connection	3-pin M8 connector	
Operating temperature	0–50 32–122	[°C] [°F]
IP rating	IP54	
Weight	0.131 0.288	[kg] [lb]
Calculated operation life	30 000	[h]

Eye Box

Eye Box	
Weight	1.01 kg 2.23 lb
Required power supply	24V (6.25A)
Calculated operation life	30 000 h

Power Supply (6.25A/150W)	Min	Typical	Max	Units
Input voltage (AC)	100	-	240	[V]
Input current	-	-	2.1	[A]
Output voltage	-	24	-	[V]
Output current	-	6.25		[A]

Power Input (24V connector)	Min	Typical	Max	Units
Supply voltage	-	24	25	[V]
Supply current	-	6.25	-	[A]

Power output (Device connector)	Min	Typical	Max	Units
Output voltage	-	24	25	[V]
Output current (EB HW v1.2)	-	4.5	4.5*	[A]

*Peak currents

Eye Box I/O Interface:

Power Reference (24V, GND)	Min	Typical	Max	Units
Reference output voltage	-	24	25	[V]

Power Reference (24V, GND)	Min	Typical	Max	Units
Reference output current	-	-	100	[mA]

Digital Output (DO1-DO8)	Min	Typical	Max	Units
Output current - altogether	-	-	100	[mA]
Output resistance (active state)	-	24	-	[Ω]

Digital Input (DI1-DI8) as PNP	Min	Typical	Max	Units
Voltage level - TRUE	18	24	30	[V]
Voltage level - FALSE	-0.5	0	2.5	[V]
Input current	-	-	6	[mA]
Input resistance	-	5	-	[kΩ]

Digital Input (DI1-DI8) as NPN	Min	Typical	Max	Units
Voltage level - TRUE	-0.5	0	5	[V]
Voltage level - FALSE	18	24	30	[V]
Input current	-	-	6	[mA]
Input resistance	-	5	-	[kΩ]

7. Bibliografía

ABB. (2018). *library.e.abb.* Obtenido de Especificaciones del Producto IRB140: https://library.e.abb.com/public/84e6cb203eef4658839e7cf66e8eaf71/3HAC041346%20PS%20IRB%20140-es.pdf?x-sign=ImRvOsT2Jz0WrduLX2Ku7h+p8psQ+kwuvQhn/UcW7RoyrJUh8FyguShi/V3lJkkI

AlexNet and ImageNet: The Birth of Deep Learning. (s.f.). Obtenido de pinecone.io: https://www.pinecone.io/learn/series/image-search/imagenet/

Aparicio, A. G. (2024). *Apuntes de la asignatura de Robotica.* Elche.

Coppelia Robotics. (s.f.). *Dynamics*. Obtenido de manual.coppeliarobotics: https://manual.coppeliarobotics.com/en/dynamicsModule.htm

Coppelia Robotics. (s.f.). *Force Sensors*. Obtenido de manual.coppeliarobotics: https://manual.coppeliarobotics.com/en/forceSensors.htm

Coppelia Robotics. (s.f.). *Joints*. Obtenido de manual.coppeliarobotics: https://manual.coppeliarobotics.com/en/joints.htm

Coppelia Robotics. (s.f.). *Models*. Obtenido de manual.coppeliarobotics: https://manual.coppeliarobotics.com/en/models.htm

Coppelia Robotics. (s.f.). *Proximity sensors*. Obtenido de manual.coppeliarobotics: https://manual.coppeliarobotics.com/en/proximitySensors.htm

Coppelia Robotics. (s.f.). *Scene Objects*. Obtenido de manual.coppeliarobotics: https://manual.coppeliarobotics.com/en/objects.htm

Coppelia Robotics. (s.f.). *Scenes*. Obtenido de manual.coppeliarobotics: https://manual.coppeliarobotics.com/en/scenes.htm

Coppelia Robotics. (s.f.). *Script objects*. Obtenido de manual.coppeliarobotics: https://manual.coppeliarobotics.com/en/scriptObjects.htm

Coppelia Robotics. (s.f.). *Scripting*. Obtenido de manual.coppeliarobotics: https://manual.coppeliarobotics.com/en/scripts.htm

Coppelia Robotics. (s.f.). *Shapes*. Obtenido de manual.coppeliarobotics: https://manual.coppeliarobotics.com/en/shapes.htm

Coppelia Robotics. (s.f.). *The main script*. Obtenido de manual.coppeliarobotics: https://manual.coppeliarobotics.com/en/mainScript.htm

Coppelia Robotics. (s.f.). *User Interface*. Obtenido de manual.coppeliarobotics: https://manual.coppeliarobotics.com/en/userInterface.htm

Derpanis, K. G. (13 de Mayo de 2010). *Overview of the RANSAC Algorithm*. Obtenido de www.cse.yorku.ca: http://www.cse.yorku.ca/~kosta/CompVis_Notes/ransac.pdf

García, L. M. (2024). *Apuntes de la asignatura de visión por computador.* Elche.

Godoy, D. (21 de Noviembre de 2018). *Understanding binary cross-entropy / log loss: a visual explanation*. Obtenido de medium.com/data-science: https://medium.com/data-science/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a

Keita, Z. (29 de Enero de 2024). *Explicacion de la deteccion de objetos YOLO*. Obtenido de datacamp: https://www.datacamp.com/es/blog/yolo-object-detection-explained

Kundu, R. (17 de Enero de 2023). *YOLO: Algorithm for Object Detection Explained [+Examples]*. Obtenido de v7labs: https://www.v7labs.com/blog/yolo-object-detection

Marathe, Y. (11 de Junio de 2020). *How Focal Loss fixes the Class Imbalance problem in Object Detection*. Obtenido de medium.com/analytics-vidhya: https://medium.com/analytics-vidhya/how-focal-loss-fixes-the-class-imbalance-problem-in-object-detection-3d2e1c4da8d7

Ortiz, P. (7 de Noviembre de 2024). *Los autómatas de Herón, las maravillas mecánicas de la antigüedad*. Obtenido de historia.nationalgeographic: https://historia.nationalgeographic.com.es/a/inventos-griegos-automatas-heron_9395

Robots Done Right LLC. (s.f.). *History of ABB Robots*. Obtenido de robotsdoneright: https://robotsdoneright.com/Articles/history-of-abb-robots.html

Scalera-, A. G. (Enero de 2019). *A Brief History of Industrial Robotics in the 20th Century.* Obtenido de researchgate.net: https://www.researchgate.net/publication/331090220_A_Brief_History_of_Industrial_Robotics_in_the_20th_Century

Ultralytics. (s.f.). *Supresión no máxima (NMS)*. Obtenido de ultralytics.es/glossary: https://www.ultralytics.com/es/glossary/non-maximum-suppression-nms

Vina, A. (7 de Junio de 2024). *¿Qué es R-CNN? Un breve resumen*. Obtenido de ultralytics.com: https://www.ultralytics.com/es/blog/what-is-r-cnn-a-quick-overview