UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

GRADO EN INGENIERÍA ELECTRÓNICA Y AUTOMÁTICA INDUSTRIAL

"Aplicación de *pick and place* para la clasificación de paquetes con el robot *Youbot*."

TRABAJO FIN DE GRADO

Febrero-2024

AUTOR: Laura García Ferri

DIRECTOR/ES: Mónica Ballesta Galdeano

ÍNDICE DE CONTENIDO.

ÍNDICE DE FIGURAS.

ÍNDICE DE TABLAS.

1. INTRODUCCIÓN.

El desarrollo de este Trabajo de Fin de Grado, "Aplicación de *pick and place* para la clasificación de paquetes con el robot *Youbot*", ha surgido por el creciente uso de la robótica en todos los ámbitos de la vida. En especial se ha centrado en la robótica móvil y la robótica manipuladora para representar una de las aplicaciones más frecuentes en la industria, el transporte de objetos, más conocida como *pick and place*, con una característica adicional, la clasificación de los objetos por tamaño.

Una de las áreas más marcadas por este rápido avance de la robótica en los últimos años ha sido la industria. Actualmente podemos encontrar robots realizando todo tipo de funciones: soldadura, pintura, recogida y colocación, embalado o paletizado, entre otras. La automatización se ha convertido en una de las formas más útiles de mejorar la eficiencia y precisión de los procesos.

La aplicación de *pick and place* consiste principalmente en seleccionar y colocar objetos de forma automatizada, una tarea que para un trabajador es muy repetitiva y monótona, llegando a ser una carga mental, puede pasar a ser realizada independientemente por un robot de la misma forma. Para esta aplicación el uso de un robot móvil es lo más útil si los espacios en los que tomamos y en los que dejamos las piezas están a una gran distancia. Asimismo, hay infinidad de variantes si combinamos esta función con la variedad de sensores robóticos que encontramos en el mercado, pudiendo eliminar del proceso productivo elementos defectuosos o realizar clasificaciones de distintos tipos de piezas, ya sea por tamaño, color y/o peso.

El proyecto incluye el desarrollo de la programación de una simulación de un robot real como es el *Youbot*, creado por la empresa *KUKA*, para explorar y comprobar su funcionalidad en tareas de recoger y clasificar objetos en colaboración con sensores de visión. El código principal se ha escrito en el lenguaje *Python* y se enviarán las instrucciones mediante una *API* remota al simulador de *CoppeliaSim*, donde se ha diseñado una escena desde cero para que se adapte visualmente a lo que se quiere representar.

En este trabajo se encuentra la información recogida en varios capítulos. Se ha empleado un primer capítulo para poner en contexto y poder comprender la situación robótica en la actualidad. Se hace un recorrido en cómo ha evolucionado a lo largo de la historia esta ciencia, tipos de robots, mercados y sus aplicaciones, especialmente en la industria. En segundo lugar, era necesario tener base en los fundamentos teóricos de la robótica en áreas como la cinemática y control de robots móviles. Posteriormente se han tratado conocimientos del robot con el que se ha trabajado, sus características y cálculos cinemáticos tanto para la base como para el brazo. En el quinto capítulo se realiza una introducción al software utilizado, *CoppeliaSim*, un software gratuito conocido en el mundo de la educación y la investigación. Por último, se dividen en dos capítulos el trabajo elaborado, realizando una discusión de las pruebas desarrolladas y las conclusiones obtenidas después de haberlo completado, con posibles direcciones en las que se podría continuar esta investigación.

2. ESTADO DEL ARTE. ROBOTS MANIPULADORES.

2.1. EVOLUCIÓN DE LA ROBÓTICA.

El termino robótica se puede definir como la ciencia que recoge varias disciplinas con el objetivo de diseñar máquinas programadas para realizar tareas o trabajos de forma automática, generalmente en sustitución de la mano de obra humana. Por otra parte, el término robot deriva de la palabra checa "*robota",* que se puede traducir como "trabajo obligatorio", usada por primera vez en la obra dramática R.U.R. escrita en 1920 por el escritor Karel Capek, donde se representaban humanoides artificiales para realizar trabajos repetitivos.

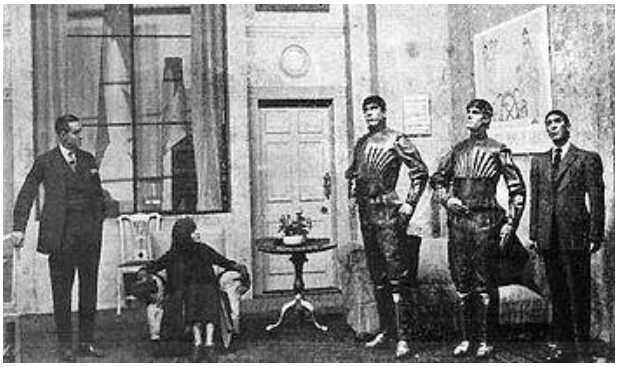


Figura 1. Imagen de la representación de la obra R.U.R de Karel Capek. https://es.wikipedia.org/wiki/R.U.R._%28Robots_Universales_Rossum%29

La robótica, a lo largo de la historia de la humanidad ha sufrido una evolución constante, estando presente en la vida de los seres humanos en distintas áreas, siempre centrada en mejorar la eficiencia de las operaciones.

Para conocer el origen de la robótica tenemos que remontarnos a la antigüedad, dónde se creaban los primeros mecanismos automatizados. Se creaban máquinas que eran capaces de realizar tareas repetitivas que realizaba el hombre creando un complejo sistema, sin embargo, algunos artefactos no tenían más utilidad que entretener.

Los primeros ejemplos de autómatas se registran en Etiopía, en el año 1500 a.C., con una estatua que emitía sonidos al recibir el sol del amanecer. Otro ejemplo, esta vez en la Antigua Grecia, Platón adaptó un reloj de agua para que realizara la función de despertador y sus alumnos no llegaran tarde. Posteriormente, en el siglo XVIII tenemos los primeros registros de autómatas humanoides.

Figura 2. Representación de funcionamiento de reloj de agua. http://quellegamos.com/el-despertador-de-platon

Sin embargo, no es hasta la Revolución Industrial cuando esta área comienza a tomar una forma más acercada a lo que conocemos hoy en día, ya que, históricamente, es el hito más importante en el desarrollo de la tecnología. A finales del siglo XIX, se comenzaron a sentar las bases de la automatización tal y como la conocemos, gracias a ingenieros o inventores como Nikola Tesla y Leonardo Torres Quevedo que sentaron las bases para la automatización moderna.

En el siglo XX comenzaron a aparecer los robots en la industria, se automatizaron funciones de recogida y colocación, soldadura, o ensamblado con brazos robóticos, en industrias como la automotriz.

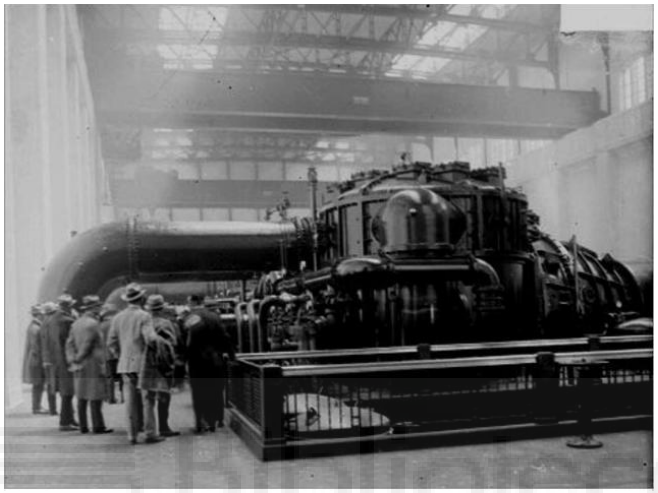


Figura 3. Maquinaria automatizada del siglo XX. https://www.timetoast.com/timelines/robotica-1b2b7b78-a8f8-488f-afef-5145cf2ab6f0

La robótica industrial continúa evolucionando durante las últimas décadas gracias a la inteligencia artificial, la informática y la electrónica. Los robots han ido ganando independencia en sus tareas y en cooperación con el ser humano, tomando decisiones en tiempo real.

El impacto de la robótica está cada vez más presente en todos los sectores profesionales. Los robots permiten aumentar la productividad, eficiencia y rentabilidad de las empresas, por lo que cada vez más compañías se decantan por incorporar estos avances en sus procesos productivos. Además, se han conseguido nuevas funciones como el reconocimiento de voz o la toma de decisiones autónomas.

Si miramos al futuro, la robótica puede ser capaz de transformar las vidas y el trabajo, su impacto continuará siendo cada vez mayor buscando siempre un mayor nivel de autonomía y libertad. Se espera que se disminuya el tamaño de los componentes, se creen sensores más sofisticados y se optimice la inteligencia artificial.

Podemos llegar a la conclusión de que la robótica ha sufrido una gran evolución, pasando de pequeños autómatas a equipos altamente independientes y colaborativos con el ser humano, teniendo claro que todavía queda mucho por avanzar.

2.2. TIPOS DE ROBOTS EN LA ACTUALIDAD.

Con los múltiples avances que ha tenido esta tecnología en los últimos años, resulta necesario hacer hincapié en visualizar con claridad qué tipos de robots existen y sus distintas clasificaciones.

En primer lugar, vamos a tratar la clasificación por generaciones de la robótica [1], separadas por etapas que han tenido cambios significativos en las capacidades de la robótica, siguiendo un orden cronológico en su desarrollo:

1. PRIMERA GENERACIÓN (1940-1970). Consisten en sistemas multifuncionales que pueden ser programados con las instrucciones a llevar a cabo, se encargan de mover objetos, lo que conocemos como robots manipuladores.

2. SEGUNDA GENERACIÓN (1970-1990). En esta generación tenemos a robots que son capaces de recibir cierto *feedback* de su entorno y son capaces de realizar movimientos más complejos. Disponen de sistemas de control de lazo cerrado y sensores.

3. TERCERA GENERACIÓN (1990-2010). Tienen sensores para su control y la percepción de su entorno y se realizan las órdenes mediante un programa en una computadora. Con estos robots se inicia la era de los robots inteligentes.

4. CUARTA GENERACIÓN (2010-actualidad). Los robots de esta generación son capaces de percibir su entorno y controlar el proceso. La diferencia con los de la tercera generación la encontramos en que, además de recibir órdenes de la computadora, también son capaces de enviar de vuelta la información recogida por los sensores.

5. QUINTA GENERACIÓN (A futuro). La generación actual, todavía en desarrollo, la podemos relacionar con el desarrollo de la inteligencia artificial y su aplicación a los nuevos controladores y actuadores, lo que les brindará mayor autonomía y capacidad de reacción.

Otra clasificación que se suele realizar es a través de la estructura del robot, un aspecto que influye considerablemente en la función que tendrá, ya que influirá en características como eficiencia, adaptabilidad o peso.

1. ROBOTS POLIARTICULADOS. En este grupo podemos encontrar robots muy variados, pero lo que encontraremos en común es que son robots sedentarios, principalmente para el desplazamiento de objetos de un lugar a otro. Podemos encontrar los robots cartesianos, manipuladores e industriales.

2. ROBOTS ZOOMÓRFICOS. Podemos definirlos como los robots que imitan a algún ser vivo en su sistema de movimiento. A su vez se pueden clasificar en caminadores y no caminadores.

3. ROBOTS MÓVILES. Robots dotados de un gran nivel de inteligencia y de información relativa a su entorno para que sean capaces de hacer desplazamientos en plataformas o carros.

4. ROBOTS ANDROIDES. Robots que tratan de reproducir el comportamiento y/o movimiento humano. Actualmente están dedicados al estudio y experimentación.

5. ROBOTS HÍBRIDOS. Robots los cuales pueden pertenecer a varias de las clasificaciones anteriores a la vez. Por ejemplo, un dispositivo segmentado articulado y con ruedas es, al mismo tiempo, uno de los atributos de los robots móviles y de los robots zoomórficos.

Estos son solo algunos ejemplos de la rica diversidad de robots que comparten nuestro mundo. A medida que la tecnología avanza, es emocionante imaginar cómo evolucionarán y se diversificarán aún más estos compañeros mecánicos en el futuro.

2.3. MERCADO DE LA ROBÓTICA EN LA ACTUALIDAD.

El mercado de la robótica experimenta un auge sin precedentes, impulsado por avances tecnológicos, la creciente demanda de automatización y la expansión de aplicaciones en diversos sectores. Este aumento de ventas de robots se debe a diversos factores, cada vez hay más personas en la industria con conocimiento sobre tecnología y su potencial para colaborar, a la misma vez que estos robots van mejorando sus prestaciones para ser más amistosos con el usuario y con otros robots.

La robótica es una tecnología con futuro, además de tener un espectro muy amplio. La integración de la inteligencia artificial está permitiendo que se realicen funciones más avanzadas que requieren aprendizaje automático y toma de decisiones. Con esto se consiguen robots capaces de adaptarse para actuar en distintos entornos.

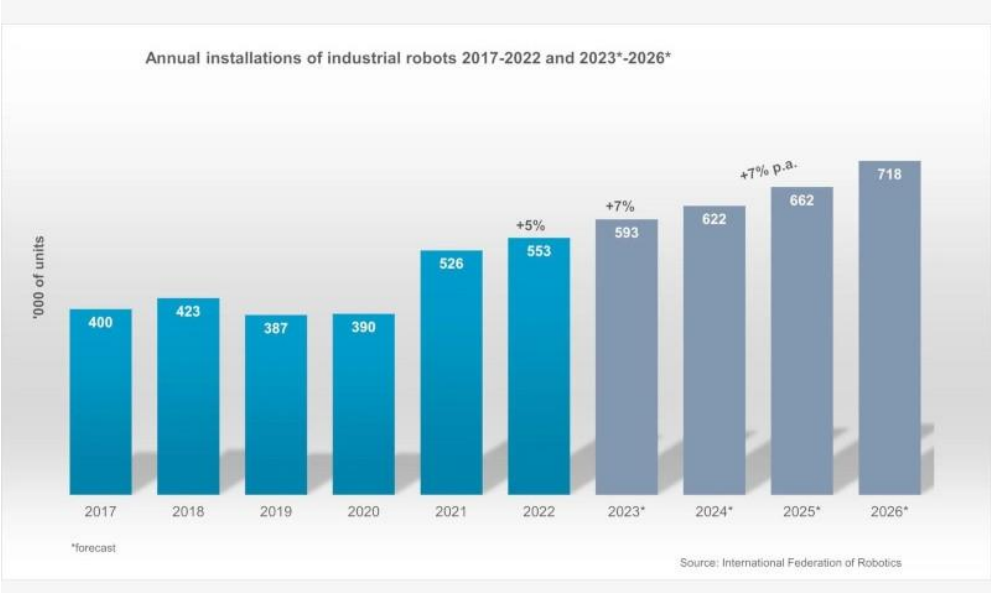


Figura 4. Maquinaria automatizada del siglo XX. https://www.timetoast.com/timelines/robotica-1b2b7b78-a8f8-488f-afef-5145cf2ab6f0

Aplicación de pick and place para la clasificación de paquetes con el robot Youbot.

La robótica médica es una de las áreas que han experimentado un auge. Desde robots quirúrgicos hasta asistentes para el cuidado de los pacientes [2], obteniendo más precisión en las intervenciones médicas y aliviando el trabajo del personal sanitario.

También podemos destacar el crecimiento de la presencia de esta tecnología en la agricultura. Se está beneficiando de robots autónomos y drones que optimizan la siembra, el riego y la cosecha. Aumenta la eficiencia en la producción de alimentos y abordan los desafíos del cambio climático y la escasez de mano de obra.

Podemos observar que se van implantando nuevas formas de trabajo en los que no estábamos tan acostumbrados a relacionar la robótica, como la hostelería o atención al cliente, gracias a los robots de servicio. Son los responsables de ofrecer una experiencia mejorada al cliente y optimizar las relaciones comerciales.

El mercado de la robótica está siendo testigo de grandes colaboraciones y estrategias que impulsan su crecimiento. A menudo empresas grandes establecidas en el sector invierten en startups con nuevas ideas de desarrollo, proporcionándoles el capital necesario para que se conviertan en una realidad. También es posible que directamente estas empresas adquieran a los startups para incorporar en su cartera las nuevas tecnologías.

En cambio, en otras ocasiones tenemos colaboraciones entre empresas del sector, como podría ser entre una empresa automotriz y una de robótica, encargada de proveer de los robots necesarios para la fabricación de vehículos. Estas colaboraciones pueden ser nacionales o internacionales, dado que hablamos de un campo global.

A medida que se consiguen avances también surgen problemas y preocupaciones éticas, por lo que es necesario estar en constante desarrollo de nuevas normas y regulaciones. Nos surgen dudas respecto al empleo de las personas, nuestra privacidad, seguridad o el impacto ambiental y hay que plantear hasta qué punto queremos y/o debemos dotar a estas máquinas de autonomías y responsabilidades.

2.4. ROBOTS MANIPULADORES.

La robótica industrial, junto a la automatización, ha producido un aumento de la productividad, un mayor aprovechamiento de recursos y disminución de errores y costes en la cadena de producción. La ISO (*International Organization for Standarization)* define al robot industrial como "un manipulador multipropósitos, reprogramable y controlado automáticamente en tres o más ejes". Estos robots son capaces de realizar tareas repetitivas para las que antes era necesario disponer de personas, evitándose peligros y, además, consiguiendo mayor velocidad y precisión. Otro punto a favor sería su fácil manejabilidad por los humanos mediante un dispositivo externo.

Los robots industriales más comunes son los conocidos como robots manipuladores o brazos robóticos. Su nombre se debe a la similitud que encontramos con el brazo humano, ya que normalmente están formados por siete segmentos metálicos unidos por seis articulaciones, obteniendo el equivalente a un hombro, un codo y una muñeca. Son robots con seis grados de libertad, pudiendo pivotar de seis formas distintas, partiendo de una base fija.

Figura 5. Brazo robótico IRB1100 de la compañía ABB. Fuente: https://webshop.robotics.abb.com/es/catalog/product/view/id/153/s/articulated-robot-irb-1100-loading-and-unloading-r-0-475/category/3/

Un robot manipulador suele construirse utilizando distintos enlaces rígidos que están interconectados por distintas articulaciones, siendo los componentes móviles que permiten al robot el movimiento entre los distintos componentes conectados entre sí. Cabe recalcar otros componentes que hacen posible el funcionamiento de este tipo de máquinas como los actuadores, sensores, sistema de control para gobernar los actuadores y sistema de decisión, encargado de elaborar el movimiento a seguir a partir de las instrucciones transmitidas por el operador.

Estos tipos de robots suelen dividirse en tres partes:

- Base: Punto de fijación y referencia de la estructura.

- Brazo y cuerpo: Su principal función es colocar el extremo en un punto concreto dentro del espacio de trabajo gracias a las articulaciones conectadas entre sí por los eslabones. Suele estar compuesto por tres de las articulaciones.

- Muñeca: Su función es organizar los objetos en los espacios de trabajo. Le corresponden los movimientos de giro, elevación y desviación, aunque existen muñecas que no realicen las 3.

Para lograr un control de posición y orientación del extremo correcto necesitaremos 6GDL, los tres primeros los encontraremos en el brazo, encargándose de la posición, y, los tres restantes, dedicados a la orientación en la muñeca.

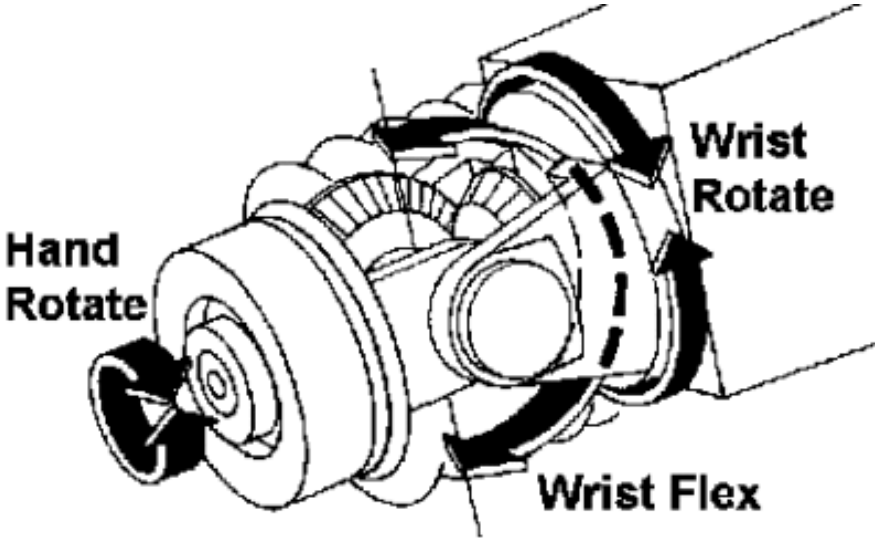


Figura 6. Representación del movimiento de una muñeca esférica.
https://isc8vo.es.tl/configuraciones-de-la-mu%F1eca.htm

Existen distintos tipos de estructuras cinemáticas, en primer lugar, podemos distinguir entre robots en paralelo y robots en serie. Los robots en paralelo son de gran utilidad en aplicaciones en las que se necesita buena distribución de cargas o altas velocidades, los podemos diferenciar al observar que están compuestos por uno o más eslabones con grado de conectividad 3 o mayor. Por otra parte, los robots en serie son los más comunes, compuestos por una base y un efector final.

A su vez, los robots en serie se pueden dividir en cadena abierta o cerrada, de cadena abierta serán los robots los cuales todos sus eslabones tengan grado de conectividad 2 excepto la base y efector final, por otra parte, la cadena cinemática cerrada se distingue por todos sus eslabones conectados entre sí, teniendo cada uno de ellos conectividad 2.

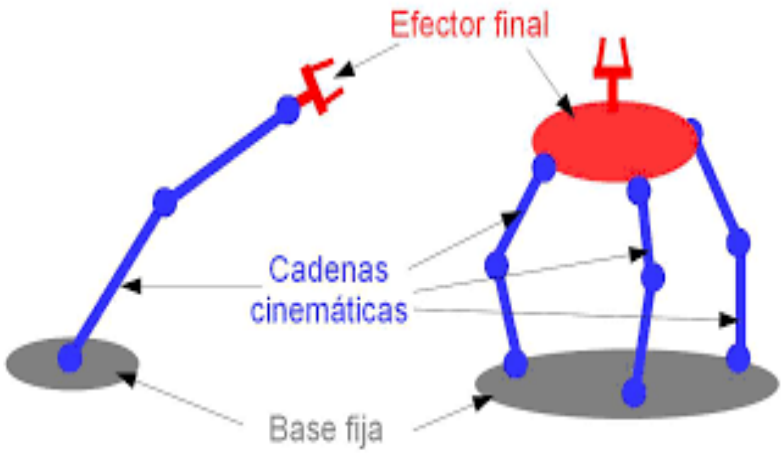


Figura 7. Diferencias entre robot en serie (de cadena cinemática abierta) y robot en paralelo.
https://arvc.umh.es/label/prac1_es.pdf

Podemos distinguir varios tipos de robots manipuladores dependiendo de sus características y capacidades y su elección se realizará dependiendo de la aplicación a la que se vaya a destinar y los requisitos de precisión, velocidad y carga útil. A continuación, se realiza una clasificación de algunos de los tipos más comunes:

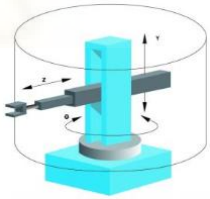
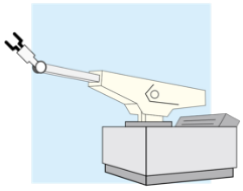
Robots cartesianos		Su movimiento se realiza a través de tres ejes de movimiento lineal (x,y,z). Idóneos para aplicaciones de movimientos repetitivos y precisos.
Robots SCARA		Dos ejes de movimiento. Comunes en aplicaciones de montaje. Realizan movimientos en el plano horizontal rápidos y precisos.
Robots articulados		Al menos 4 o 6 ejes de movimiento. Gran flexibilidad. Soldadura, pintura, incluso cirugía.
Robots Delta		Forma de pirámide invertida. Se utilizan en la industria alimentaria y fabricación de piezas electrónicas. Muy precisos y rápidos.
Robots cilíndricos		Se utilizan en aplicaciones en las que no alcanzan el resto de los robots. Por ejemplo, el ensamblado de motores.
Robots polares		Su espacio de trabajo es esférico, se usa en aplicaciones en las que requieren gran variedad de movimientos.
Robots colaborativos		Trabajan junto al ser humano, por ejemplo, en montaje o manipulación de objetos. Contienen más sistemas de seguridad.

Tabla 1. Clasificación de robots manipuladores.

Al encontrar una gama tan amplia de robots industriales podemos guiarnos por una serie de características que se diferencian entre sí a la hora de escoger el adecuado. Algunas de las características para tener en cuenta pueden ser las siguientes:

Capacidad de carga	Carga máxima en kg que es capaz de mover un robot sin afectar a sus prestaciones.
Espacio de trabajo	Espacio físico que delimita el extremo del robot cuando este recorre los rangos límite del movimiento de las articulaciones.
Grados de libertad	Cada uno de los movimientos independientes que una articulación puede realizar respecto al eslabón anterior, suma un grado de libertad al robot.
Precisión	Capacidad del sistema para situar el extremo de la muñeca en un punto determinado en el espacio de trabajo.
Repetibilidad	Capacidad del robot para situarse sobre un mismo punto las veces que sean necesarias.

Tabla 2. Características a tener en cuenta en robótica.

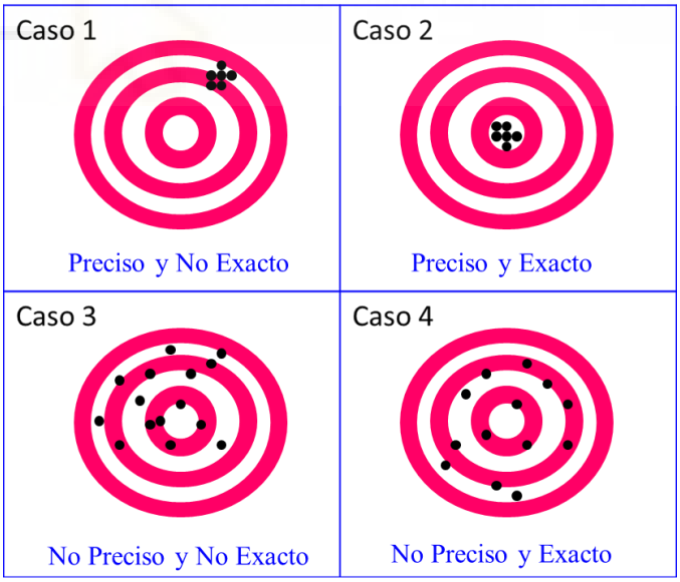


Figura 8. Representación gráfica de las propiedades de precisión y repetibilidad. https://www.fuyumotion.com/es/news/what-are-accuracy-and-repeatability-in-industrial-robots-2/

Una parte indispensable en este tipo de robots es el elemento terminal, está encargado de interaccionar directamente con el entorno del robot. Ofrecen muchas ventajas ya que suelen ser independientes al robot, pudiendo dar diferentes usos y aumentar su versatilidad. Podemos realizar una clasificación general según el tipo de elemento de

sujeción: pinzas, ventosas o electroimanes, o en cambio, por su función: soldadura, corte, pintura, etc.

Figura 9. Distintos tipos de grippers utilizados en manipuladores. https://revistaderobots.com/sistemas-de-agarre/tipos-de-gripper-y-pinzas-roboticas/

A modo de conclusión, los robots manipuladores son una gran representación de la potencia de la automatización y la robótica para mejorar la productividad y la calidad de vida. En los últimos años se están creando nuevas posibilidades en el lugar de trabajo a modo de colaboración y flexibilidad.

2.5. APLICACIONES EN LA INDUSTRIA.

Los robots manipuladores han conseguido un lugar entre los elementos imprescindibles de una industria, ya que desempeñan una gran cantidad de actividades, aumentando la eficiencia y calidad de producción. La implantación de un robot industrial en un determinado proceso exige un estudio previo y las consecuencias de implementar este robot, tanto positivas como negativas

Las aplicaciones de los brazos robóticos en la industria abarcan toda la cadena de producción, encontrándolos en trabajos de soldadura, corte, pintura o montaje entre otros. Los robots son capaces de realizar tareas repetitivas y precisas con una gran velocidad y consistencia.

Analizando las principales aplicaciones industriales de los robots manipuladores encontramos:

- Soldadura.

Centrándonos en el sector automovilístico, uno de los principales impulsores de la robótica en la industria, destacan las aplicaciones de soldadura de carrocerías. Dos piezas metálicas se unen en un punto para la fusión conjunta de ambas partes.

Existen dos formas de realizar este trabajo, un primer método consiste en que el robot transporte la pieza a los electrodos o, en cambio, que el robot transporte la pinza de la soldadura posicionando los electrodos en el punto exacto.

Es una de las funciones para las que existe mayor demanda de robots, esto ha provocado que se comiencen a diseñar robots especializados en soldadura, tienen

entre 5 y 6 grados de libertad para posicionar y orientar la pieza en lugares de difícil acceso y tienen una capacidad de carga de 50 a 100 kg.

Será necesario, en primer lugar, programar manualmente o a través de sistemas de programación asistida los parámetros del robot y las trayectorias a seguir. Antes de la soldadura, las piezas se preparan, limpiándose y posicionándose correctamente. El robot realiza el proceso siguiendo los parámetros introducidos, mientras tanto, la unidad de control supervisa activamente el proceso siendo capaz de realizar pequeños ajustes automáticos si fuera necesario, para mantener el nivel de calidad de la soldadura.

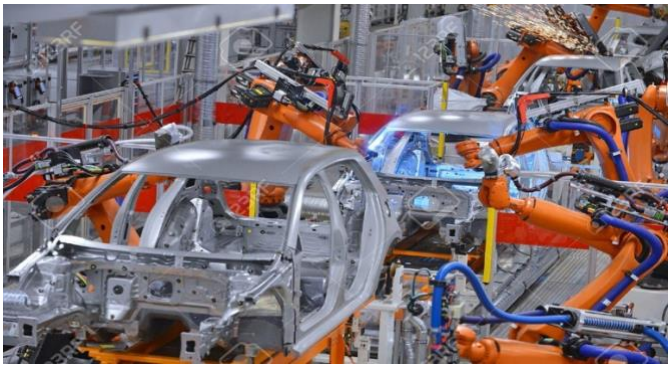


Figura 10. Ejemplo de disposición de robots de soldadura en la industria del automóvil. https://es.123rf.com/photo_16221553_robots-de-soldadura-en-una-f%C3%A1brica-de-autom%C3%B3viles.html

La elección del tipo de robot depende de diversos factores, como el área de trabajo, el tipo de soldadura o la velocidad de producción. Los robots más comunes son los robots manipuladores, tienen varios grados de libertad y un brazo que les permite abarcar una gran área de trabajo, son ideales para líneas de montaje.

Cuando necesitemos alta velocidad y precisión la mejor opción serán los robots *SCARA*, gracias a sus brazos articulados en paralelo, también será buena opción los robots Delta, con su estructura paralela con distintos brazos conectados a una estructura central.

Existen robots especializados para tipos de soldadura específicos, por ejemplo, para las soldaduras MIG (*Metal Inert Gas*) o TIG (*Tungsten Inert Gas*), se crean con herramientas concretas para realizar estos procesos.

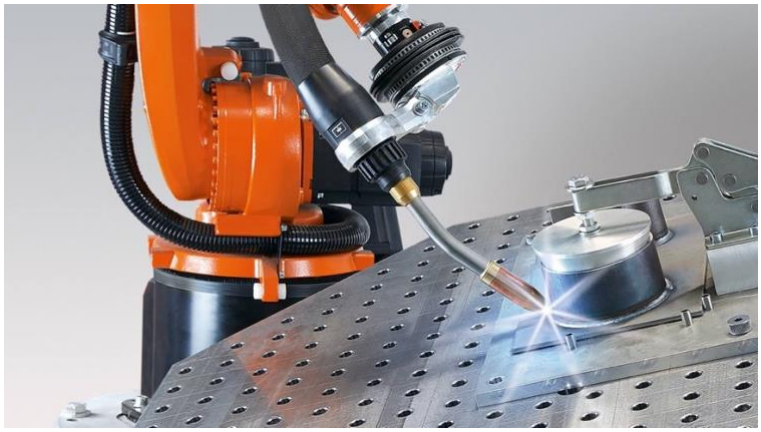


Figura 11. Robot de soldadura con gas de protección. https://www.kuka.com/es-mx/sectores/industria-del-metal/soldadura-con-gas-de-protecci%C3%B3n/robots-de-soldadura-con-gas-de-protecci%C3%B3n-de-kuka

- Paletización.

La paletización se realiza al final de la producción y es una de las tareas que más ha sufrido la automatización. El paletizado consiste en disponer la mercancía sobre un pallet para almacenarla y transportarla. Las piezas en un pallet ocupan normalmente posiciones predeterminadas, procurando asegurar la estabilidad, facilitar su manipulación y optimizar su extensión. Los pallets son transportados por diferentes sistemas (cintas transportadoras, carretillas, etc.) llevando su carga de piezas, bien a lo largo del proceso de fabricación, bien hasta el almacén o punto de expedición.

En general, estas tareas implican manejar grandes cargas, en cuanto a peso y dimensiones, por lo que a la hora de seleccionar un robot capaz de realizar esta tarea deberemos tener en cuenta su carga útil, alcance o precisión. Además, podemos valorar para cada situación sus sistemas de visión, si necesitamos clasificar los objetos, por ejemplo, por color, también podremos valorar sus herramientas de agarre, según el tipo de producto necesitaremos ventosas, pinzas o paletizadores.

Para realizar la paletización es necesario que se recoja información en primer lugar, gracias a los sistemas de visión y distintos sensores, los cuales estudiarán el entorno para crear la trayectoria más fácil de seguir, la información recogida también es útil para que se planifique la disposición que se va a seguir al dejar los productos en el pallet. Posteriormente el robot comenzará su tarea, recogerá piezas y las apilará en sus respectivas posiciones, además es necesario que estén conectados con otros sistemas del centro de logística para enviar y recibir información.

El robot paletizador cartesiano es uno de los robots más eficientes en este tipo de aplicaciones. Realiza movimiento a través de los tres ejes, es un robot que es capaz de mantener una gran estabilidad con las cargas en diferentes direcciones. También es muy común encontrar brazos robóticos diseñados para soportar grandes cargas y capaces de realizar movimientos a una alta velocidad.

Figura 12. Robot Despaletizador y paletizador. https://www.edsrobotics.com/blog/robot-despaletizador-cajas/

- Montaje.

Es común que los robots también realicen tareas de montaje o desmontaje de un producto, como pueden ser componentes electrónicos, de automoción o muebles. Estas acciones requieren robots de gran precisión y habilidad siendo complicado conseguir cierta flexibilidad en las operaciones que permitan introducir frecuentes modificaciones en los productos con unos costes mínimos. En cambio, no es necesario robots que sean capaces de manejar grandes cargas.

El montaje y el desmontaje son una secuencia coordinada de muchas operaciones individuales, identificación, separación, manipulación, inserción o fijación/unión de las piezas y también podemos incluir acciones posteriores como la limpieza, el sellado o el etiquetado. Estos robots contienen un controlador recibiendo datos, de la pieza o del trabajo o enviando datos a la herramienta de control. Para obtener un proceso lo más eficiente posible es deseable proporcionar las piezas de forma ordenada y en la posición correcta, debiendo estar el robot coordinado con otros componentes de la línea de montaje como sistemas de alimentación de piezas o transportadoras.

Para este proceso son útiles los robots *SCARA* y los cartesianos, conocidos por su precisión y movimientos rectilíneos a través de un plano. Los robots móviles también pueden llegar a ser muy útiles en esta aplicación ya que son capaces de moverse a lo largo de la cadena de montaje, siendo más flexibles y adaptables.

Cuando se necesita que se realicen varias operaciones al mismo tiempo pueden ser necesarios robots de doble brazo, los cuales pueden realizar múltiples movimientos simultáneos coordinados entre sí. Por último, siempre podemos encontrar robots de brazos articulado en las cadenas de montaje, ya que tienen flexibilidad y capacidad para acceder a múltiples posiciones, o robots delta, utilizados en cadenas de ensamblaje que requieren movimientos ágiles y precisos.

Este proceso dinámico no solo transforma la manera en la que se ensamblan los productos, impulsa la innovación en la fabricación, ofreciendo soluciones más eficientes y de mayor calidad.

Figura 13. Robot SCARA en cadena de montaje. https://new.abb.com/news/es/detail/82902/abb-amplia-la-gama-de-robots-scara-para-un-montaje-mas-rapido-y-de-alta-precision

- Corte.

El corte de materiales es una de las aplicaciones más recientes. El robot se encarga de transportar la herramienta de corte sobre la zona a cortar realizando el movimiento con

gran precisión gracias a la definición previa por un programa de diseño asistido por computador. Los métodos de corte no mecánico más empleados son oxicorte, plasma, láser y chorro de agua, dependiendo de la naturaleza del material a cortar.

Los robots empleados requieren control de trayectoria continua y elevada precisión. Su campo de acción varía con el tamaño de las piezas a cortar siendo, en general, de envergadura media (de 1 a 3 metros de radio). En este sentido, como se ha comentado, con mucha frecuencia se dispone al robot suspendido boca abajo sobre la pieza.

Figura 14. Corte por chorro de agua. https://www.chemours.com/es/brands-and-products/chemours-minerals/applications/waterjet-cutting

Para que sea posible automatizar el proceso de corte es necesario realizar un minucioso diseño y planificación, es necesario tener en cuenta factores como el material a cortar o la forma necesaria para su posterior programación. Los robots utilizarán para el corte herramientas especializadas, como láseres, chorro de agua o herramientas de fresado, con las que realizarán el corte en el material deseado. Al igual que para el resto de las aplicaciones, es necesario que se controle continuamente el proceso verificando la calidad del corte, en el caso de que se pierda el nivel de precisión se harán saltar los ajustes automáticos.

En este proceso podemos encontrar distintos tipos de robots, similares a las aplicaciones anteriores: robots cartesianos, *SCARA* o robots con brazos articulados, su elección dependerá del caso concreto. Podemos encontrar estos robots integrados con Sistemas de Corte CNC (Control Numérico Computarizado), son herramientas avanzadas que permiten una programación detallada y la capacidad para realizar trayectorias de cortes complejos, esenciales en industrias como la manufactura, fabricación de metales o carpintería.

2.6. EL SOFTWARE DEDICADO A LA ROBÓTICA.

El software enfocado en la robótica es imprescindible para poder programar, controlar y operar con robots en diversas aplicaciones. Este tipo de software tiene múltiples funciones, como programar los movimientos a realizar con gran precisión, la simulación en entornos virtuales o integrar sistemas de visión. Principalmente existen tres áreas de

desarrollo de software enfocadas a la robótica: entornos de desarrollo robótico, programación de robots y simulación y diseño.

Un entorno de desarrollo es un espacio de trabajo formado por un conjunto de procedimientos y diversas herramientas que permiten a los desarrolladores crear un programa o código fuente y probar si funciona. Podemos destacar ROS (*Robot Operating System*), uno de los sistemas operativos más utilizados para robots, ofrece una plataforma de software libre de código abierto para el desarrollo de aplicaciones robóticas, permitiendo controlar robots, simular entornos y facilitar la comunicación entre componentes robóticos. Se utiliza en gran variedad de aplicaciones robóticas, tanto en la. Industria como en investigación académica.

Figura 15. Ejemplo entorno ROS. https://www.pilz.com/es-ES/products/robotics/ros-modules

Antes de la aparición de los entornos de desarrollo cuando el usuario cambiaba de robot o adquiría uno distinto era necesario aprender a utilizar un nuevo software, pero ahora es posible compartir códigos, funciones y programas de uso común entre distintos robots

Para programar un robot se sigue el mismo proceso que seguiríamos a la hora de elaborar cualquier programa informático para cualquier otra aplicación. Deberemos tener claro el algoritmo idóneo que permita al robot llevar a cabo las tareas que se traducirá al lenguaje de programación del sistema de control del robot. No existe un lenguaje de programación generalizado, existen multitud de lenguajes, ya que en muchos casos los fabricantes desarrollan su propio lenguaje de programación como pueden ser: *VAL* o *RCL*. Para algunos robots destinados a fines de investigación o educativos podemos utilizar lenguajes como *C*, *Visual Basic* o *Logo*.

Para programar un robot puede realizarse directamente, en contacto directo con el robot, o a modo "*Offline*", mediante un software de programación se puede programar el robot sin necesidad de estar físicamente conectado al robot, por lo que no es necesario sacar al robot de su proceso de producción.

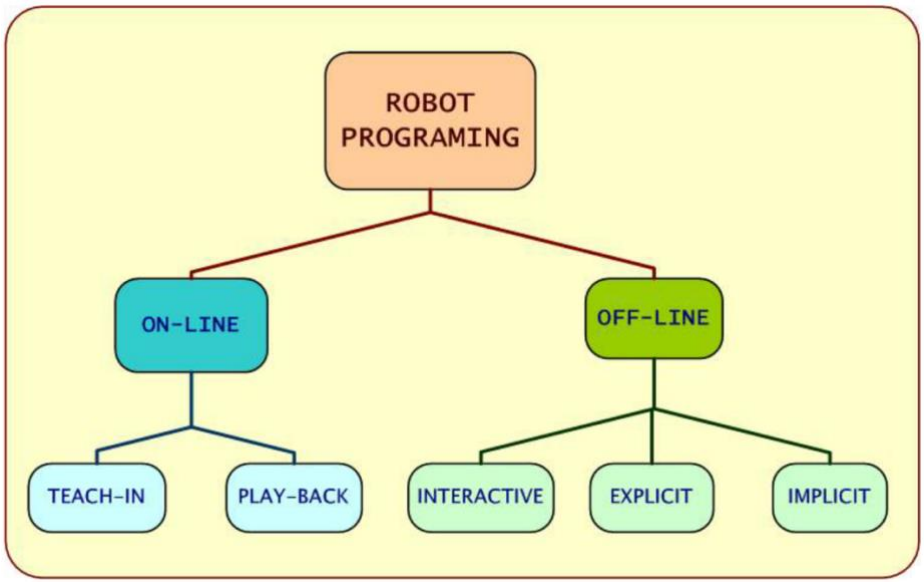


Figura 16. Diagrama de la clasificación de los tipos de programación de robots. https://biblus.us.es/bibing/proyectos/abreproy/3721/fichero/PFC.pdf

Por otra parte, mediante los simuladores es posible modelar sistemas robóticos de forma similar a como lo hacen en la realidad, podemos destacar la plataforma V-REP (*Virtual Robot Experimentation Platform*), utilizada en investigación, desarrollo y pruebas de algoritmos robóticos. Este software es muy útil para comprobar el comportamiento de aplicaciones multi robot, sistemas robóticos completos o subsistemas, como pueden ser sensores o mecanismos.

Una de las soluciones para que las máquinas tengan la capacidad de interpretar la información visual proporcionada por el mundo real es la visión por computador, un campo que pertenece a la inteligencia artificial. El sistema analiza mediante algoritmos vídeos o imágenes y mediante un software especializado identifica y clasifica objetos, así como puede ser capaz de detectar caras y emociones.

Dado que la visión artificial se utiliza para resolver distintos problemas, existen sistemas especiales dependiendo de qué tarea se deba resolver. Los sistemas típicos de visión se componen de cámaras, un software, procesadores, fuentes de luz, aplicaciones de software y varios sensores.

En conclusión, el software permite crear soluciones robóticas para la industria, la investigación y la educación, mejora la eficiencia y ayuda a la automatización de tareas. Podemos ver que existe una gran variedad de herramientas disponibles, en continuo desarrollo.

3. FUNDAMENTOS TEÓRICOS.

Para poder comprender el movimiento de los mecanismos en la automatización industrial es necesario realizar un estudio previo para analizar las posibilidades de movimiento del sistema dentro del espacio, así como las limitaciones mecánicas desde un punto de vista físico.

3.1. CINEMÁTICA DIRECTA.

La base del estudio de la cinemática directa se encuentra en definir cuál será la posición y orientación del sistema en su elemento final dado unos valores determinados a sus articulaciones. Este conocimiento es imprescindible para planificar y controlar los movimientos de un robot, ya que nos proporciona la capacidad de poder calcular la posición del efector final del robot en función de las coordenadas articulares.

Para describir la localización de un objeto en el espacio tridimensional con respecto a un sistema de referencia fijo se utiliza fundamentalmente el álgebra vectorial y matricial. A un robot lo podemos considerar como una cadena de eslabones unidos entre sí mediante articulaciones, se puede establecer un sistema de referencia fijo situado en la base del robot y describir las localizaciones del resto de eslabones respecto al sistema de referencia.

Por lo tanto, la cinemática directa se puede reducir a encontrar la matriz homogénea de transformación que relacione la posición y orientación del extremo del robot respecto al sistema de referencia fijo situado en la base de este. En algunos casos con robots sencillos las relaciones pueden conseguirse con consideraciones geométricas, pero para robots de más grados de libertad será necesario el uso de estas matrices de transformación homogénea.

A cada eslabón se le puede asignar un sistema de referencia solidario a él y, utilizando las transformaciones homogéneas, es posible representar las rotaciones y traslaciones relativas entre los distintos eslabones.

La matriz de transformación homogénea es una matriz con dimensión 4x4, se puede considerar compuesta por 4 submatrices: matriz de rotación, vector de traslación, transformación de perspectiva y escalado global.

$$T = \begin{bmatrix} R_{3x3} & p_{3x1} \\ f_{1x3} & w_{1x1} \end{bmatrix} = \begin{bmatrix} Rotación & Traslación \\ Perspectiva & Escalado \end{bmatrix} \quad (1)$$

Por lo que, cuando tengamos una rotación y/o traslación de un vector r con respecto a un sistema de referencia fijo para transformarlo en r' respecto a un segundo sistema de referencia tendremos que realizar la siguiente operación:

$$\begin{bmatrix} r'_x \\ r'_y \\ r'_z \\ 1 \end{bmatrix} = T \begin{bmatrix} r_x \\ r_y \\ r_z \\ 1 \end{bmatrix} \quad (2)$$

Podemos encontrar casos en los que solo se realice translación o rotación en alguno de los ejes, quedando las siguientes matrices homogéneas (tomando la perspectiva y escalado como nulos):

- Traslación:

$$T = \begin{bmatrix} 1 & 0 & 0 & p_x \\ 0 & 1 & 0 & p_y \\ 0 & 0 & 1 & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3)$$

- Rotación sobre eje x:

$$Rx(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -sen(\alpha) & 0 \\ 0 & sen(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4)$$

- Rotación sobre eje y:

$$Ry(\phi) = \begin{bmatrix} \cos(\phi) & 0 & sen(\phi) & 0 \\ 0 & 1 & 0 & 0 \\ -sen(\phi) & 0 & \cos(\phi) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5)$$

- Rotación sobre eje z:

$$Rz(\gamma) = \begin{bmatrix} \cos(\gamma) & -sen(\gamma) & 0 & 0 \\ sen(\gamma) & \cos(\gamma) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (6)$$

Por otra parte, en los casos en los que tengamos rotación y traslación, dos rotaciones y una traslación, u otra combinación, se podrá conseguir la matriz homogénea compuesta realizando la multiplicación de las distintas matrices homogéneas simples, recordando que la multiplicación de matrices no es conmutativa, por lo que deberá seguir un orden lógico. Esto se aplicará al encontrar la matriz de transformación homogénea que relaciona el sistema de referencia fijo con el efector final, a partir de los distintos subsistemas de las articulaciones.

Para obtener las relaciones entre eslabones de un robot se utiliza el algoritmo Denavit-Hartenberg [3], se trata de un procedimiento sistemático para describir la estructura cinemática de una cadena articulada constituida por articulaciones con un solo grado de libertad con la ayuda de sistemas de referencia en cada uno de los eslabones del robot siguiendo una serie de normas.

Las transformaciones en cuestión que encontraremos son las siguientes:

- Rotación alrededor del eje Z_{i-1} un ángulo θ_i.
- Traslación a lo largo de Z_{i-1} una distancia d_i.
- Traslación a lo largo de X_i una distancia a_i.
- Rotación alrededor del eje X_i un ángulo α_i.

Por lo tanto, la matriz de transformación homogénea que obtendremos para cada eslabón será la multiplicación de estas 4 transformaciones. Los cuatro parámetros D-H son la vuelta al enlace (α_i.), la longitud del enlace (a_i), desplazamiento del enlace (d_i) y ángulo de las articulaciones (θ_i), se puede recurrir a la Figura 17 para una representación gráfica.

En la matriz únicamente podemos tener un valor variable, por lo tanto, tres de estos parámetros serán fijos y uno variable, siendo la variable el valor θ_i cuando la articulación sea de revolución y el valor d_i cuando sea prismática. Teniendo en cuenta las 4 transformaciones posibles quedará la siguiente matriz de transformación homogénea para cada articulación:

$$T_{z,d}T_{z,\theta}T_{x,a}T_{x,\alpha} = \begin{bmatrix} \cos(\theta_i) & -\cos(\alpha_i)sen(\theta_i) & sen(\alpha_i)sen(\theta_i) & a_i cos(\theta_i) \\ sen(\theta_i) & cos(\alpha_i)cos(\theta_i) & -sen(\alpha_i)cos(\theta_i) & a_i sen(\theta_i) \\ 0 & sen(\alpha_i) & cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (7)$$

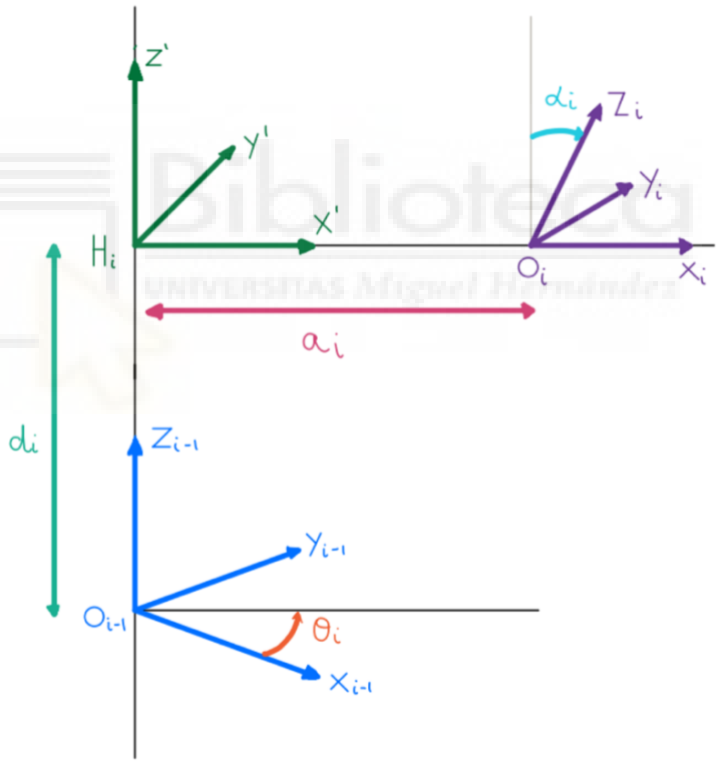


Figura 17. Representación de las 4 transformaciones para pasar de un sistema de referencia i-1 a i.

3.1.1. EJEMPLO APLICACIÓN CINEMÁTICA DIRECTA 2GDL.

Para comprender mejor el proceso de aplicación de los parámetros D-H para conseguir la cinemática directa se ha realizado un ejemplo práctico de un mecanismo de dos articulaciones rotacionales siguiendo los pasos a realizar.

Para asignar correctamente los sistemas de referencia se ha numerado, en primer lugar, las articulaciones desde 1 hasta n y los ejes Z desde Z_0 hasta Z_{n-1}, estos ejes se colocarán para que correspondan como los ejes de rotación de la articulación, o en el caso de que sea prismática, serán el eje a lo largo del cuál se mueve el eslabón.

El sistema de coordenadas origen puede colocarse a lo largo del eje Z_0, con X_0 e Y_0 pudiéndose colocar aleatoriamente siempre que sea un sistema dextrógiro. Para el resto de los sistemas, el eje Z_i se colocará en la normal común a Z_i y Z_{i+1}, si se cortan ambos ejes, se colocará en el punto de corte, si son paralelos, en algún punto de la articulación i+1. El eje X va en la dirección normal común de Z_{i-1} a Z_i. Una vez situados los ejes Z y X, los Y tienen su dirección determinada por la restricción de que sea un sistema dextrógiro. Por último, para el último sistema, el del extremo del robot, se colocará con su eje Z paralelo a Z_{i-1} y X e Y en cualquier dirección válida.

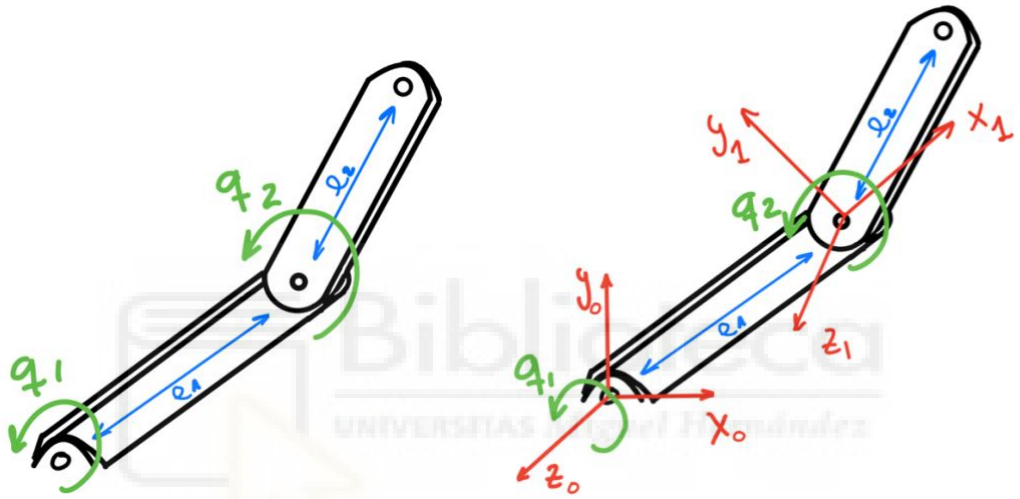


Figura 18. Colocación de sistemas de coordenadas en robot siguiendo normas D-H.

Una vez tengamos los sistemas colocados correctamente, podremos ver las transformaciones entre ellos obteniendo los 4 parámetros D-H con los que formaremos las matrices de transformación homogénea para cada una de las transformaciones entre articulaciones, obteniendo con su multiplicación el resultado de la matriz de transformación homogénea final, la cual permite resolver el problema de cinemática directa en robots manipuladores, ya que dando valores concretos a cada uno de los grados de libertad del robot, obtenemos la posición y orientación 3D de la herramienta en el extremo del brazo.

Articulación	θ	d	a	α
1	q_1	0	L1	0
2	q_2	0	L2	0

Tabla 3. Tabla de parámetros D-H partiendo de representación en la Figura 18.

Las ecuaciones 8 y 9 corresponden a las matrices de transformación de cada articulación por separado y la ecuación 10 a la matriz de transformación homogénea final para este mecanismo.

$$ {}^{0}A_1 = \begin{pmatrix} c(q1) & -s(q1) & 0 & L1c(q1) \\ s(q1) & c(q1) & 0 & L1s(q1) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (8) $$

$$ {}^{1}A_2 = \begin{pmatrix} c(q2) & -s(q2) & 0 & L2c(q2) \\ s(q2) & c(q2) & 0 & L2s(q2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (9) $$

$$ T = {}^{0}A_2 = \begin{pmatrix} c(q1+q2) & -s(q1+q2) & 0 & L1c(q1)+L2c(q1+q2) \\ s(q1+q2) & c(q1+q2) & 0 & L1s(q1)+L2s(q1+q2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (10) $$

3.2. CINEMÁTICA INVERSA.

La cinemática inversa resuelve la configuración que deben adoptar las coordenadas articulares del robot para una posición y orientación del extremo conocidas, por lo que nos encontramos el problema opuesto a la cinemática directa.

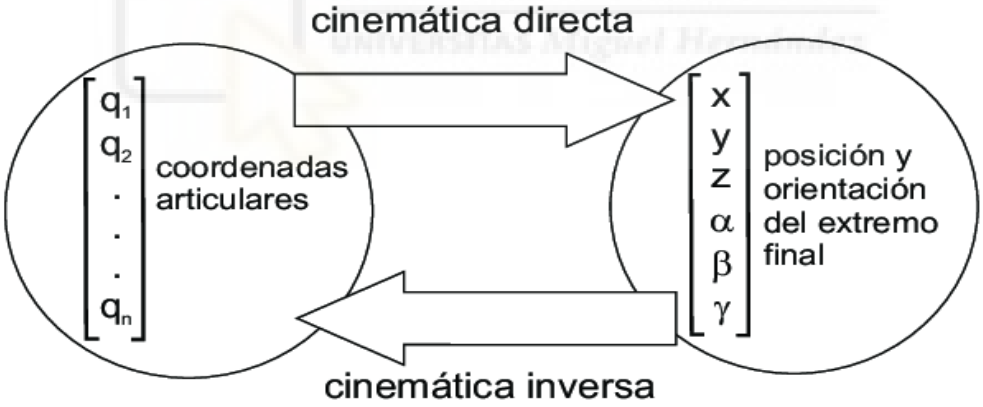


Figura 19. Relación entre cinemática directa e inversa. https://www.researchgate.net/figure/Figura-216-Relacion-entre-cinematica-directa-e-inversa_fig15_287995531

El procedimiento de obtención de las ecuaciones es dependiente de la configuración del robot, encontraremos problemas debido a las singularidades, el espacio de trabajo alcanzable por el robot, podremos encontrar más de una, una o ninguna solución y también pueden presentarse problemas debido a las ecuaciones no lineales, ya que tendremos senos y cosenos en las matrices de rotación.

Para encontrar la solución a estos problemas se prefieren expresiones analíticas con soluciones cerradas, como métodos geométricos y métodos algebraicos. La cinemática inversa ha de resolverse en muchas aplicaciones en tiempo real y es importante la velocidad de la solución, ya que si se necesita calcular todas las soluciones posibles

llevará un tiempo mucho mayor. Por otra parte, si resolvemos la cinemática inversa con soluciones cerradas podremos añadir requisitos adicionales como límites en los recorridos o evitar algún obstáculo.

Para robots con pocos grados de libertad es útil resolver la cinemática inversa mediante métodos geométricos, encontrando relaciones geométricas entre las distintas partes del robot, mediante identidades trigonométricas, como puede ser el teorema del coseno. Además, la orientación del extremo siempre será la suma de los valores de las variables articulares. Los datos de partida que nos encontramos serán la posición y orientación deseada en el efector final referidas al sistema de referencia.

3.2.1. EJEMPLO APLICACIÓN CINEMÁTICA INVERSA 2GDL.

Para el método geométrico siempre podemos tomar de ejemplo un robot estándar con 3 grados de libertad:

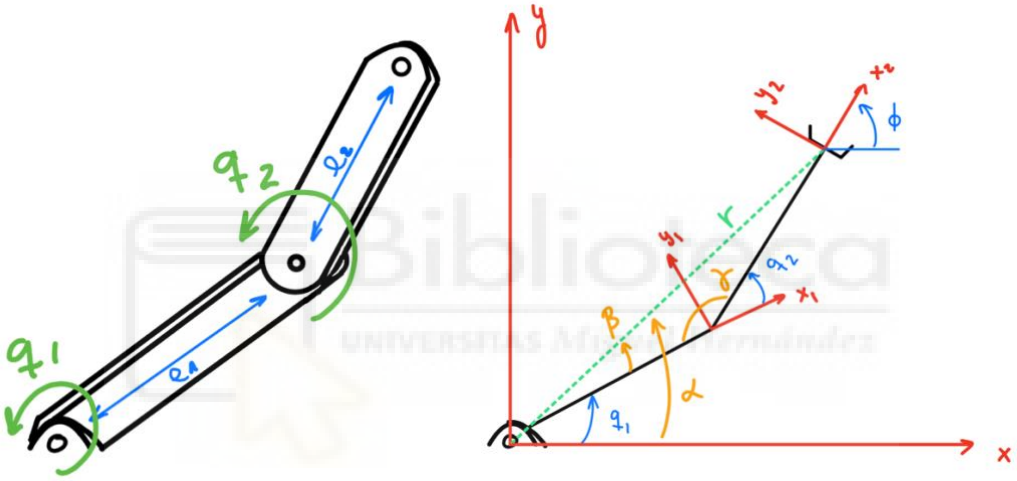


Figura 20. Representación de robot de 2GDL tomado para cálculos de cinemática inversa.

Fijándonos en los valores de la Figura 20, el valor de "r" corresponde a una relación pitagórica entre p_x y p_y, siendo la raíz cuadrada de la suma de estos valores al cuadrado.

$$r = \sqrt{P_y{}^2 + P_x{}^2} \quad (11)$$

Para el primer valor de ángulo se ha aplicado la tangente de α:

$$\alpha = arctg(\frac{p_y}{p_x}) \quad (12)$$

Posteriormente, para el resto de los ángulos (β y γ) Se ha aplicado el teorema del coseno:

$$a^2 = b^2 + c^2 - 2bcos(\alpha) \quad (13)$$

Para β se obtiene la siguiente expresión:

$$L2^2 = L1^2 + r^2 - 2rL1cos(\beta) \quad (14)$$

$$\beta = \text{acos}\left(\frac{L1^2+r^2-L2^2}{2rL1}\right) \quad (15)$$

Al igual que para γ:

$$r^2 = L1^2 + L2^2 - 2L2L1cos(\gamma) \quad (16)$$

$$\gamma = \text{acos}\left(\frac{L1^2+L2^2-r^2}{2L2L1}\right) \quad (17)$$

Una vez que se han obtenido los valores de α, β y γ, podemos obtener los valores de las coordenadas articulares q_1 y q_2. Existirán dos soluciones para cada posición combinando los valores de q_1 y q_2, codo arriba (Ecuaciones 18 y 19) y codo abajo (Ecuaciones 20 y 21) como se muestra en la Figura 21.

$$q1 = \alpha + \beta \quad (18)$$

$$q2 = \gamma - \pi \quad (19)$$

$$q1 = \alpha - \beta \quad (20)$$

$$q2 = \pi - \gamma \quad (21)$$

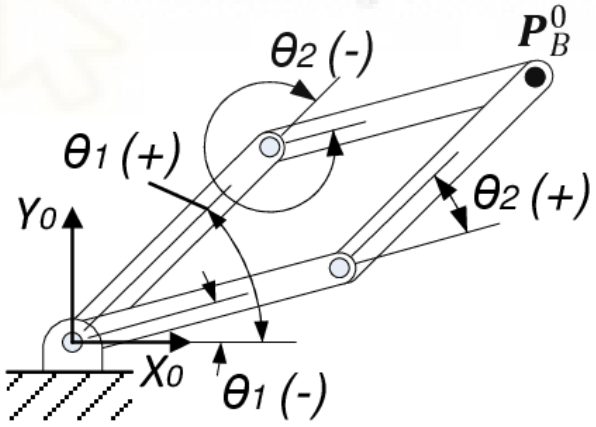


Figura 21. Representación soluciones codo arriba y codo abajo.

Si nos resulta confuso encontrar relaciones geométricas podemos llegar a la misma solución con el método de la matriz de transformación homogénea [4]. Este método parte de suponer conocidas las relaciones en el modelo directo, es decir, partimos de las relaciones que tenemos en la matriz de cinemática directa. En la práctica puede llegar a ser complejo. En primer lugar, aplicaremos los parámetros D-H, obtendremos las matrices A, y, posteriormente, la matriz T. Obtenida la matriz T en función de las coordenadas articulares, y conociendo los valores de posición y orientación deseados, podemos trabajar con las ecuaciones obtenidas en la matriz para despejar las incógnitas, las coordenadas articulares del robot, aunque conseguir esto directamente normalmente es más complicado.

3.2.2. DESACOPLO CINEMÁTICO.

Los métodos de resolución en la cinemática inversa pueden ser aplicados para robots más complejos a cambio de también mayor complejidad de cálculos, pero, en general, es preferible descomponer el robot y reducir los cálculos.

Esto suele suceder en robots de 6 grados de libertad en los que tenemos una muñeca que dota al extremo del robot de una orientación determinada. El método de desacoplo cinemático [5] separa el problema de la orientación, que se consigue mediante las tres primeras coordenadas articulares, y el de la orientación, que se consigue mediante las coordenadas del punto de corte de los últimos 3 ejes.

En el desacoplo cinemático se obtienen en primer lugar el valor de las primeras coordenadas articulares como hemos visto en el apartado anterior, y, posteriormente, con el dato de la orientación deseada, se obtiene el valor del resto de las variables articulares. Si hemos realizado la siguiente descomposición de la matriz de rotación, despejamos los valores que nos quedan por conocer:

$$ {}^{3}R_{6} = R_{ij} = ({}^{0}R_{3})^{-1}\,{}^{0}R_{6} \quad (22) $$

Las matrices de rotación están compuestas de columnas ortonormales, por lo que su inversa es igual a su transpuesta. Sabiendo esto, podremos obtener los valores de las coordenadas articulares q_4, q_5 y q_6.

3.3. CONTROL DE POSICIÓN Y ORIENTACIÓN DE LA BASE.

Para el control de la base de un robot móvil tanto de posición como de orientación respecto a un sistema de referencia inercial se utiliza la cinemática diferencial, la principal diferencia con la cinemática anterior es que se trabaja a nivel de velocidad. La ventaja de la cinemática diferencial es la linealidad en las operaciones.

La forma de representar la posición y la orientación en la base dependerá del tipo de robot móvil que tengamos. Por ejemplo, a la hora de representar un robot con patas deberemos describir la posición y orientación de la base flotante y las coordenadas articulares de las distintas patas, cada pata se modela como una cadena cinemática en serie respecto a la base del robot, después se expresa respecto al sistema inercial.

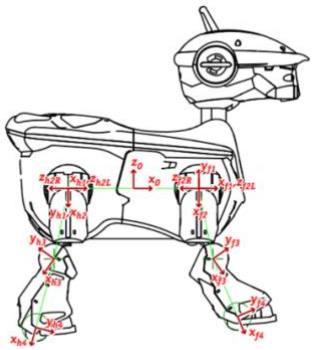


Figura 22. Representación cadena cinemática en robot con patas. http://oramosp.epizy.com/teaching/201/fund-robotica/clases/8_Cinematica_Robots_Moviles.pdf?i=1

La representación más común de robot móvil es la de un robot con ruedas, el cual se representa en un plano de dos dimensiones, con componentes x e y, siempre circulará por el suelo, y, por lo tanto, la componente Z siempre será cero. Por otra parte, sí que podremos tener orientación, realizando giros alrededor del eje Z. En robótica móvil se suele describir la posición y la orientación del robot en 2D con un solo vector:

$$\xi = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \quad (23)$$

Derivado de este vector, podemos obtener las coordenadas de la velocidad:

$$\dot{\xi} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} \quad (24)$$

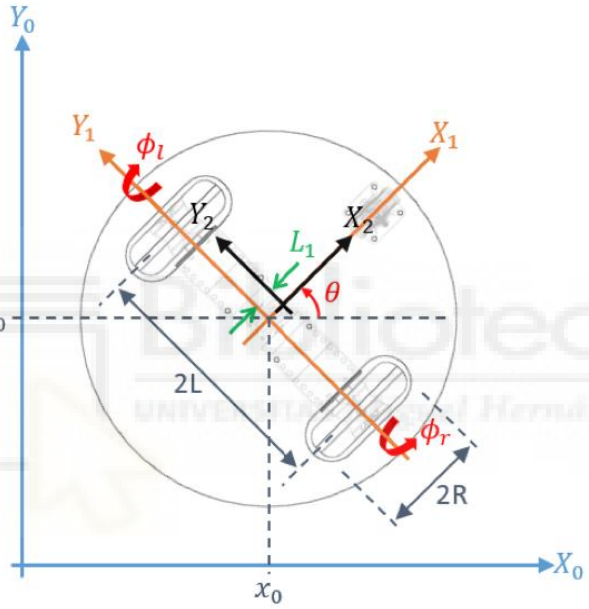


Figura 23. Representación en el espacio de robot con ruedas respecto a sistema de referencia inercial. http://somim.org.mx/memorias/memorias2017/articulos/A3_189.pdf

Para controlar la posición y orientación se utiliza la cinemática, en este caso, relaciona el movimiento de las ruedas con el movimiento del robot en el espacio cartesiano. Se relaciona a nivel de velocidad, mediante cinemática diferencial, ya que las ruedas imponen restricciones a la velocidad del robot, muchas de estas restricciones son no integrables y no se permite obtener las relaciones con la posición.

Como sucedía en la cinemática a nivel de posición, también tendremos cinemática diferencial directa y cinemática diferencial inversa. Si conocemos las características de la rueda, su rapidez de giro y la variación de la orientación, podremos obtener la velocidad del robot directamente, en cambio, si deseamos que el robot alcance una velocidad determinada tendremos que realizar la inversa para pasar al robot la información sobre la velocidad de giro y variación de orientación que tienen que tomar sus ruedas.

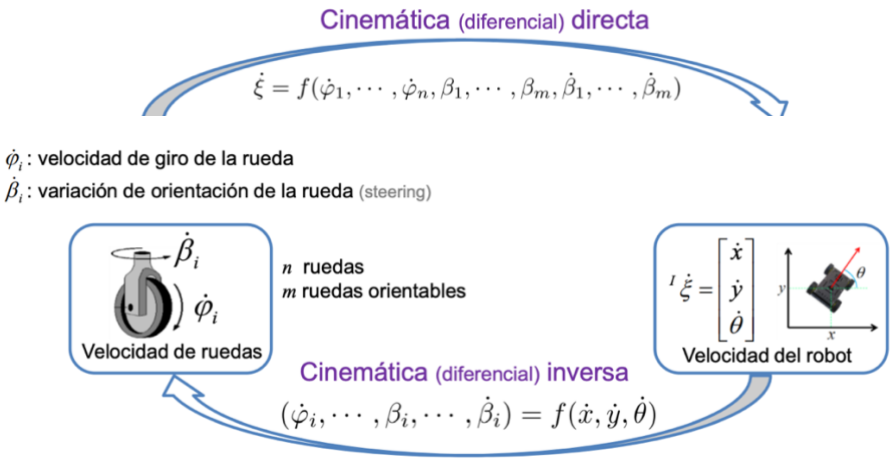


Figura 24. Representación de la relación entre la cinemática diferencial directa e inversa. http://oramosp.epizy.com/teaching/201/fund-robotica/clases/8_Cinematica_Robots_Moviles.pdf?i=1

Esta relación está sujeta a limitaciones según las ruedas que tenga instaladas el robot, cada rueda impondrá restricciones a su movimiento, dependiendo de cómo está colocada y de qué tipo es.

Para los cálculos con las ruedas supondremos una rueda ideal, con estructura rígida, plano vertical y con un solo punto de contacto entre el suelo y la rueda. Además, no habrá deslizamiento, la rueda únicamente puede rodar, y no resbala lateralmente. Con estas condiciones, si colocamos los sistemas de referencia como en la siguiente imagen, la velocidad lineal de la rueda solo tendrá componente x, en el sentido del desplazamiento, y la componente Y será 0, correspondiendo al deslizamiento.

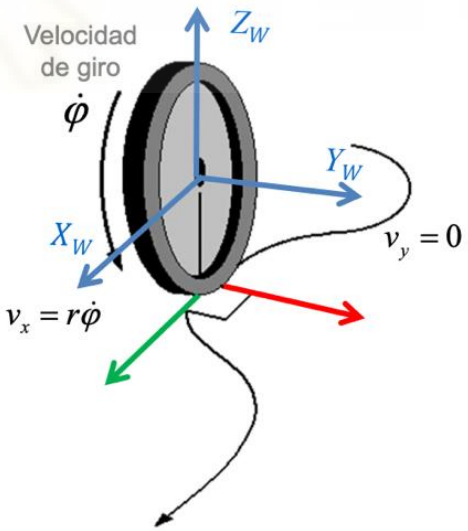


Figura 25. Componentes de la velocidad en rueda ideal. http://oramosp.epizy.com/teaching/201/fund-robotica/clases/8_Cinematica_Robots_Moviles.pdf?i=1

Por lo que tendremos el siguiente vector de velocidad respecto al sistema de referencia de la rueda:

$$\xi = \begin{bmatrix} r\dot{\varphi} \\ 0 \\ 0 \end{bmatrix} \quad (25)$$

Para calcular las restricciones que ofrecen las ruedas tendremos que determinar las componentes de velocidad en las direcciones de la rueda, para rodadura (componente x) y deslizamiento (componente y). Partiendo de una rueda convencional ideal, a la que se le aplica la velocidad lineal del robot, en la dirección del movimiento de la rueda y su perpendicular obtendremos 4 componentes. Para la velocidad angular del robot, obtendremos dos componentes en la dirección del movimiento de la rueda y su perpendicular. Igualando esto a nuestras limitaciones para las componentes anteriores, obtenemos las siguientes restricciones por tipo de rueda:

- Ruedas convencionales.

    - Rueda fija: Sólo gira alrededor del eje de la rueda y su orientación se mantiene constante, ya que no cambia respecto del chasis.

    - Rueda orientable centrada: Gira alrededor del eje propio de la rueda, su orientación no es constante, puede cambiar en contacto con el suelo.

Para el primer caso de rueda convencional, la rueda fija, β será un valor fijo, en cambio, para la rueda orientable centrada, este valor será variable.

-Restricción de rodadura:

$$[sen(\alpha+\beta) \quad -\cos(\alpha+\beta) \quad -l\cos(\beta)]\xi = r\dot{\varphi} \quad (26)$$

-Restricción perpendicular al movimiento:

$$[\cos(\alpha+\beta) \quad \text{sen}(\alpha+\beta) \quad -l\cos(\beta)]\xi = 0 \quad (27)$$

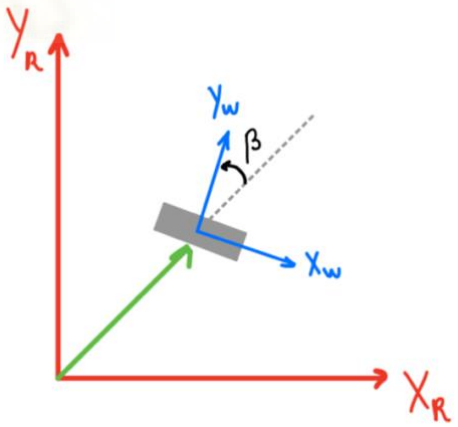


Figura 26. Representación componentes en la rueda convencional.

- Rueda orientable descentrada.

    Como en la rueda orientable centrada, su giro es alrededor del eje de la rueda y cambia su orientación con respecto al chasis, girando sobre el eje descentrado.

-Restricción de rodadura:

$$[sen(\alpha+\beta) \quad -\cos(\alpha+\beta) \quad -lcos(\beta)]\xi = r\dot{\varphi} \quad (28)$$

-Restricción perpendicular al movimiento:

$$[cos(\alpha+\beta) \quad \text{sen}(\alpha+\beta) \quad -lsen(\beta+d)]\xi = -d\dot{\beta} \quad (29)$$

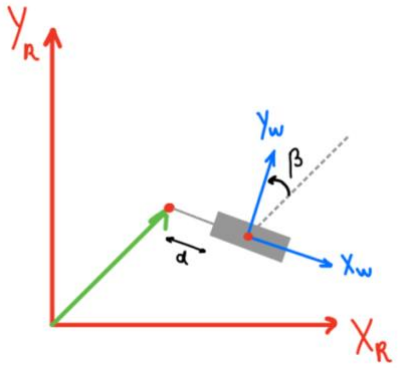


Figura 27. Representación componentes en la rueda orientable descentrada.

- Rueda omnidireccional.

  Está compuesta por pequeñas ruedas, las cuales pueden tener distintas orientaciones, siendo los más comunes de 90º y 45º [6]. Realizan el giro alrededor del eje de la rueda y otro giro secundario alrededor de las pequeñas ruedas.

  Para el caso de γ, cuando las pequeñas ruedas estén orientadas 90º será 0, cuando estén 45º, también valdrá 45º, en sentido positivo o negativo.

  -Restricción de rodadura:

$$[sen(\alpha+\beta+\gamma) \quad -\cos(\alpha+\beta+\gamma) \quad -lcos(\beta+\gamma)]\xi = r\varphi\dot{\cos}(\gamma) \quad (30)$$

  -Restricción perpendicular al movimiento:
  No existe ya que las pequeñas ruedas pueden girar libremente.

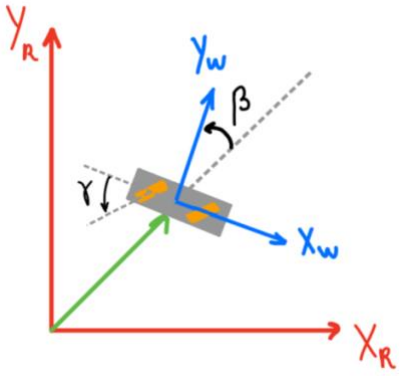


Figura 28. Representación componentes en la rueda omnidireccional.

- Rueda esférica.

  Tienen rotación instantánea alrededor de los 3 ejes, lo común es que sean ruedas de apoyo y no sean actuadoras. Son omnidireccionales.

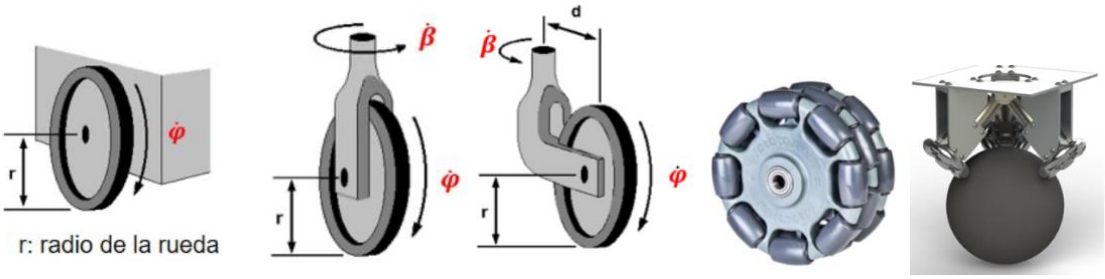


Figura 29. Distintos tipos de ruedas utilizadas en robots, de izquierda a derecha: rueda fija, rueda orientable centrada, rueda orientable descentrada, rueda omnidireccional y rueda esférica. https://ciladi.org/wp-content/uploads/Libro-Robots-VF3.pdf

Estas restricciones las aplicaremos en el modelo cinemático, se pueden agrupar de forma matricial y operar con ellas, ya que tendremos un sistema lineal, para despejar cada una de las partes que queramos obtener para la cinemática directa y la cinemática inversa.

$$A^R\dot{\xi} = B\dot{\varphi} \quad (31)$$

- A: Matriz en la que cada fila contendrá las restricciones por rueda.
- B: Contiene el radio de cada rueda, o cero, siguiendo el lado derecho de la ecuación de las restricciones.
- ${}^R\dot{\xi}$: Velocidad del robot en su sistema de referencia.

Las ecuaciones para la solución de los dos problemas que podemos plantear serán las siguientes:

- Cinemática directa: Determinar la velocidad del robot en función de la velocidad de las ruedas.

$${}^R\dot{\xi} = \dot{A^{\#}B\dot{\varphi}} \quad (32)$$

  Siendo $A^{\#} = (A^T A)^{-1}A^T$ (Pseudo-inversa de Moore-Penrose)

- Cinemática inversa: Determinar las velocidades de las ruedas en función de la velocidad del robot.

$$\dot{\varphi} = \dot{B^{\#}A\dot{\varphi}} \quad (33)$$

4. ROBOT YOUBOT.

4.1. CARACTERÍSTICAS.

El robot *YouBot* forma parte de las creaciones de la empresa alemana de robótica *Kuka Roboter GmbH*, una de las principales fabricantes mundiales de robots industriales y soluciones automatizadas, en colaboración con la Universidad Técnica de Múnich. Es un pequeño robot móvil que ha sido desarrollado como una plataforma de código abierto para la investigación y la enseñanza científica, por lo que los desarrolladores pueden personalizar y ampliar sus capacidades según las necesidades de cada aplicación. Es muy útil en aplicaciones de gran precisión al ser un robot con movimientos pequeños, en cooperación con otros robots, con humanos, o trabajando en enjambre en logística.

El robot es controlado mediante el sistema operativo *ROS*, lo que facilita su integración con otros robots y sistemas compatibles, ya que la comunidad de usuarios de este sistema operativo es extensa, teniendo acceso a gran variedad de recursos, admite múltiples lenguajes de programación como *C++* o *Python* y podremos probar las distintas partes del robot de manera independiente.

Podemos distinguir dos partes principales: el brazo robótico y la plataforma móvil, las cuales pueden usarse por separado, combinadas, o con otros elementos adicionales.

Figura 30. Brazo y plataforma Youbot. https://www.kuka.com/es-es

Figura 31. Posibles combinaciones brazo-plataforma Youbot. https://www.kuka.com/es-es

El brazo está formado por 5 articulaciones de rotación con 5 grados de libertad, siendo capaz de trabajar en un gran espacio de trabajo. En las tablas 1 y 2 podemos observar sus características en profundidad.

El brazo está diseñado de forma modular, lo que quiere decir que cada eslabón del brazo es una unidad independiente, permitiendo mayor personalización de la configuración. Además, cada uno de estos eslabones está formado por sensores y actuadores para el control preciso de los movimientos, creando una retroalimentación en tiempo real de la posición de cada eslabón, además se realizará el movimiento mediante sistemas de motores sin escobillas.

Características brazo *Youbot*	
Cinemática en serie	5 ejes
Altura	655 mm
Superficie de trabajo	0,513m3
Peso	5,8kg
Carga útil	0,5kg
Estructura	Fundición de magnesio
Repetibilidad de posición	0,1mm
Comunicación	EtherCAT: ciclo 1ms
Voltaje de conexión	24V DC
Potencia de tren de transmisión limitada a	80W

Tabla 4. Características brazo Youbot. Fuente: https://www.kuka.com/es-es

Nº eje	**Rango trabajo**	**Velocidad movimiento**
Eje 1	+/- 169º	90º/s
Eje2	+90º/-65º	90º/s
Eje3	+146º/-151º	90º/s
Eje4	+/-102º	90º/s
Eje5	+/-167º	90º/s

Tabla 5. Rango de movimiento por eje del robot Youbot. Fuente: https://www.kuka.com/es-es

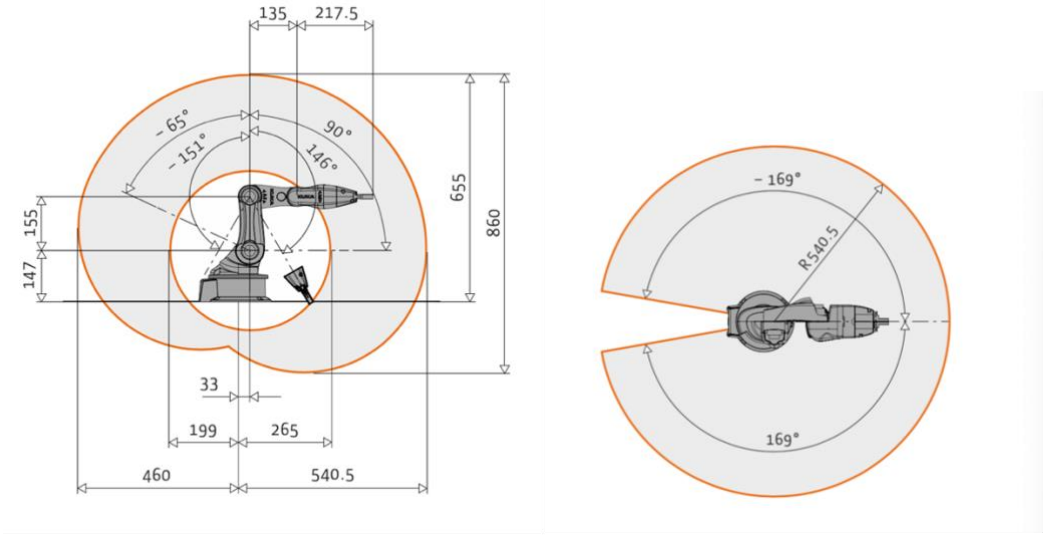


Figura 32. Representación medidas y rango de trabajo brazo Youbot. https://www.kuka.com/es-es

El extremo del brazo es compatible con multitud de herramientas útiles en distintas funciones como ensamblaje, soldadura o movimiento de objetos. La forma más común de encontrarlo es con una pinza formada por dos dedos que pueden moverse independientemente gracias a dos motores, pudiendo coger objetos de hasta 70 mm y un peso de hasta 500 gramos.

Por otra parte, tenemos la plataforma, es muy útil para estudios de desarrollo en el área de logística y navegación, ya sea para planificación de trayectorias o navegación autónoma. Tiene integrado un controlador *PC* y es posible ampliar su interfaz con sensores o actuadores. Para su movimiento autónomo usa una batería recargable de plomo que no necesita mantenimiento, esta batería de 24V y 5 Ah, con una potencia de carga de 200W, puede durar aproximadamente 90 minutos completamente cargado.

Además, tiene gran libertad de movimiento, ya que el sistema de las ruedas es omnidireccional. Se tratan de unas ruedas especiales cuya superficie de desplazamiento está compuesta por rodillos en ángulo de 45º y son capaces de moverse independientemente, lo que permite realizar desplazamientos laterales y diagonales.

Figura 33. Ruedas omnidireccionales Youbot. https://www.luisllamas.es/robot-con-mecanum-wheel-controlado-por-arduino/

Características plataforma *Youbot*	
Cinemática omnidireccional	4 *KUKA omniWheels*
Longitud	580mm
Anchura	376mm
Altura	140mm
Holgura	30mm
Peso	20kg
Carga útil	20kg
Estructura	Acero
Velocidad	0.8m/s
Comunicación	*EtherCAT:* ciclo 1ms
Voltaje de conexión	24V DC

Tabla 6. Características plataforma Youbot. Fuente: https://www.kuka.com/es-es

4.2. CINEMÁTICA DIRECTA.

A continuación, vamos a analizar la cinemática directa del robot *YouBot* basándonos en lo explicado en el Capítulo 3. De las 5 articulaciones que tiene el *YouBot* únicamente tendremos en cuenta para los cálculos 3 de ellas, eliminando la primera y la última articulación, ya que son suficientes para obtener las posiciones que necesitaremos. No tendremos en cuenta el giro en la base ya que controlaremos esta orientación con el control de la base, dejando el robot orientado de frente, ni necesitaremos la articulación que realiza el giro de la pinza para coger piezas en horizontal, en otras aplicaciones que requieren movimientos más complejos sí que será necesario tener en cuenta las 5 articulaciones [7].

Figura 34. Representación de las 5 articulaciones del robot YouBot.

Tomaremos como base y punto de referencia la articulación q_2, y obtendremos los parámetros D-H siguiendo las reglas. Los datos conocidos que tenemos son las coordenadas articulares que deseamos y las longitudes entre articulaciones, las tomaremos de la siguiente forma:

- L1: Longitud entre q_2 y q_3.
- L2: Longitud entre q_3 y q_4.
- L3: Longitud entre q_4 y el extremo del robot (hasta comienzo de las pinzas).

Por lo tanto, nos quedará un robot de 3 articulaciones de rotación que podemos reducir a 3 segmentos con 3 grados de libertad como en la siguiente imagen:

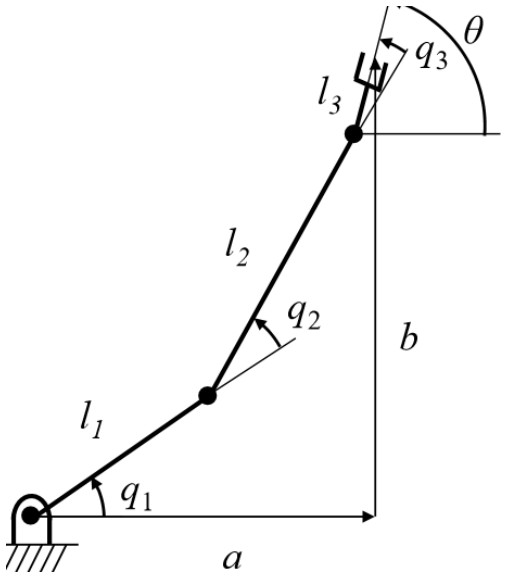


Figura 35. Esquemático de robot de 3 articulaciones de rotación.
https://robotics.stackexchange.com/questions/11901/how-to-get-max-torque-on-robot-arm-s-joints-rrr

Obtendremos los siguientes parámetros D-H:

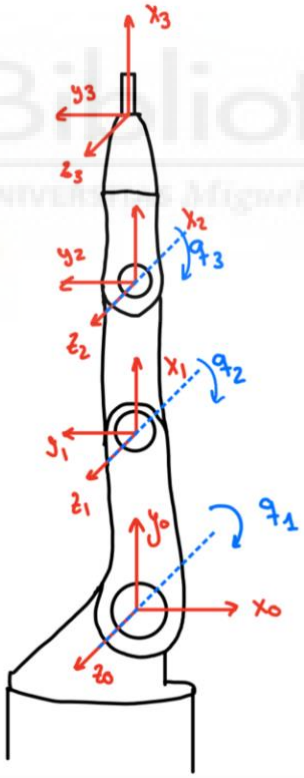


Figura 36. Colocación de ejes en robot YouBot.

Articulación	θ	d	a	α
1	q_1	L1	0	0
2	q_2	L2	0	0
3	q_3	L3	0	0

Tabla 7. Parámetros D-H del robot YouBot.

Partiendo de la Tabla 7 obtendremos las matrices de transformación entre las articulaciones, posteriormente las multiplicaremos en orden para obtener la matriz de transformación homogénea que relaciona la primera articulación con el extremo:

- Primera matriz de transformación:

$$^{0}A_{1}=\begin{pmatrix}\cos(q_1) & -\text{sen}(q_1) & 0 & L1\cos(q_1)\\ \text{sen}(q_1) & \cos(q_1) & 0 & L1\text{sen}(q_1)\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{pmatrix} \quad (34)$$

- Segunda matriz de transformación:

$$^{1}A_{2}=\begin{pmatrix}\cos(q_2) & -\text{sen}(q_2) & 0 & L2\cos(q_1+q_2)\\ \text{sen}(q_2) & \cos(q_2) & 0 & L2\text{sen}(q_1+q_2)\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{pmatrix} \quad (35)$$

- Tercera matriz de transformación:

$$^{2}A_{3}=\begin{pmatrix}\cos(q_3) & -\text{sen}(q_3) & 0 & L3\cos(q_1+q_2+q_3)\\ \text{sen}(q_3) & \cos(q_3) & 0 & L3\text{sen}(q_1+q_2+q_3)\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{pmatrix} \quad (36)$$

- Resultado matriz de transformación homogénea total:

$$^{0}A_{3}=\begin{pmatrix}\cos(q_1+q_2+q_3) & -\text{sen}(q_1+q_2+q_3) & 0 & L3*\text{a}\\ \text{sen}(q_1+q_2+q_3) & \cos(q_1+q_2+q_3) & 0 & L3*\text{a}\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{pmatrix} \quad (37)$$

Siendo a=$L1\cos(q_1)+L2\cos(q_1+q_2)+\text{L3}\cos(q_1+q_2+q_3)$

4.3. CINEMÁTICA INVERSA.

Para el cálculo de la cinemática inversa del robot *YouBot*, en nuestro caso particular, realizaremos las mismas simplificaciones que en la cinemática directa, por lo tanto, continuaremos teniendo un robot de 3 grados de libertad con 3 articulaciones de rotación. Realizaremos los cálculos mediante el método geométrico, por lo que buscaremos relaciones entre los ángulos y longitudes que encontramos en el brazo, obteniendo las siguientes ecuaciones para las coordenadas articulares:

- Articulación q_1:

$$q_1=\alpha\pm\beta \quad (38)$$

- Articulación q_2:

$$\pm q_2 = \pi - \gamma \text{ (39)}$$

El signo dependerá de si queremos tener una solución codo arriba o codo abajo:

- Solución codo arriba:

$$q_1 = \alpha + \beta \text{ y } q_2 = \gamma - \pi \text{ (40)}$$

- Solución codo abajo:

$$q_1 = \alpha - \beta \text{ y } q_2 = \pi - \gamma \text{ (41)}$$

- Articulación q_3:

$$q_3 = \theta - q_1 - q_2 \text{ (42)}$$

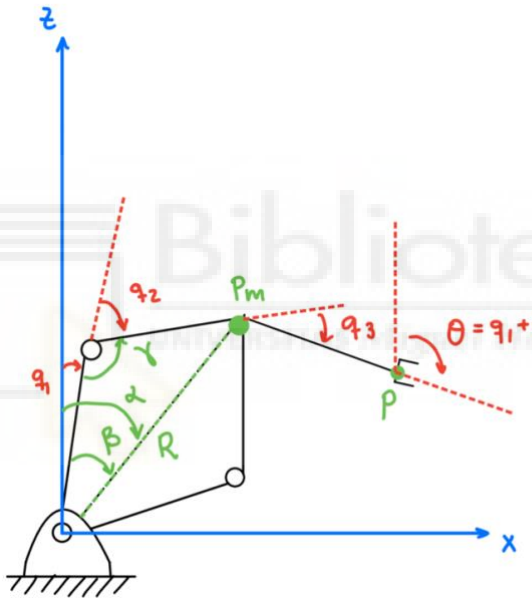


Figura 37. Relaciones geométricas en esquemático del robot YouBot.

Teniendo las siguientes relaciones geométricas (recurrir a la Figura 37 para mayor aclaración):

$$R = \sqrt{Pm_x{}^2 + Pm_z{}^2} \text{ (43)}$$

$$Pm = P - L3 * X_3 \text{ (44)}$$

$$\alpha = \tan^{-1}\left(\frac{Pm_x}{Pm_z}\right) \text{ (45)}$$

Para obtener β partimos del teorema del coseno:

$$L2^2 = L3^2 + R^2 - 2L3Rcos(\beta) \text{ (46)}$$

$$\beta = \cos^{-1}\left(\frac{L3^2+R^2-L4^2}{2L3R}\right) \text{ (47)}$$

Al igual que para β, obtenemos γ mediante el teorema del coseno:

$$R^2 = L3^2 + L4^2 - 2L3L4\cos(\gamma) \text{ (48)}$$

$$\gamma = \cos^{-1}\left(\frac{L4^2+L3^2-R^2}{2L3L4}\right) \text{ (49)}$$

4.4. CONTROL DE POSICIÓN Y ORIENTACIÓN DE LA BASE.

El robot *YouBot* está equipado con 4 ruedas omnidireccionales con un ángulo de inclinación de las pequeñas ruedas que las componen de 45º, por lo que deberemos tener en cuenta las restricciones que aplican a estos tipos de ruedas.

Para controlar la base lo más sencillo será aplicar la cinemática inversa diferencial, ya que podremos marcar la velocidad a la que queremos que se mueva el robot, y, a partir de este dato, obtendremos la velocidad de giro de la rueda y la variación de orientación de la rueda [8].

Según la configuración del robot, los rodillos de la rueda de la esquina superior derecha tienen la misma orientación que los de la rueda inferior izquierda, para las otras dos pasa lo mismo, pero en sentido contrario, esto explicará las distintas combinaciones de velocidades que obtendremos al necesitar distintos movimientos, por ejemplo, para un movimiento de frente las velocidades laterales se contrarrestan evitando el movimiento lateral. Además, Se calculará el movimiento respecto al punto central del robot.

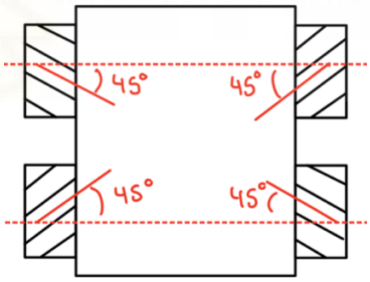


Figura 38. Representación de colocación de las ruedas ominidireccionales del Youbot.

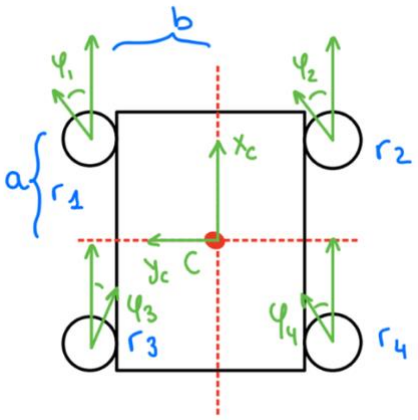


Figura 39. Representación de componentes de velocidad en cada una de las ruedas.

Si damos por conocido la velocidad en sus componentes x, y, y orientación en la que se produce podremos obtener la velocidad de giro necesaria en cada una de las ruedas:

$$v_1 = v_y - v_x + a\omega + b\omega \quad (50)$$
$$v_2 = v_y + v_x - a\omega - b\omega \quad (51)$$
$$v_3 = v_y - v_x - a\omega - b\omega \quad (52)$$
$$v_4 = v_y + v_x + a\omega + b\omega \quad (53)$$

Si aplicamos que la velocidad en las ruedas puede ser:

$$v_i = \omega_i R \quad (54)$$

Nos quedará la siguiente matriz relacionando las velocidades de las 4 ruedas:

$$\begin{bmatrix} \dot{\varphi}_{r1} \\ \dot{\varphi}_{r2} \\ \dot{\varphi}_{r3} \\ \dot{\varphi}_{r4} \end{bmatrix} = \frac{1}{R} \begin{bmatrix} 1 & -1 & -(a+b) \\ 1 & 1 & (a+b) \\ 1 & 1 & -(a+b) \\ 1 & -1 & (a+b) \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} \quad (55)$$

Donde a y b serán las distancias desde las ruedas al centro del robot y R el radio de las ruedas.

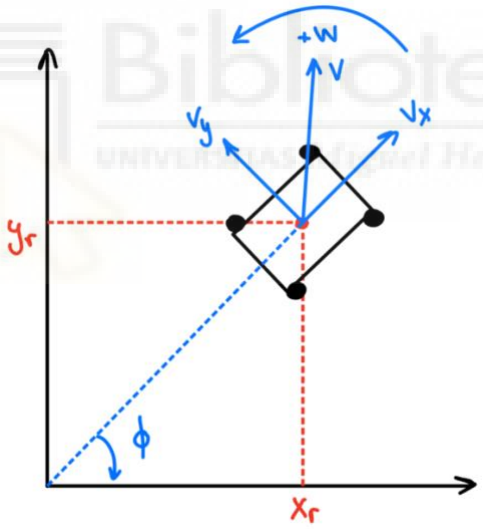


Figura 40. Coordenadas de la velocidad del robot en plano 2D.

Para un movimiento de rotación las ruedas en el mismo lado tienen la misma velocidad, pero sentidos opuestos a los del otro lado. Para un movimiento lateral las ruedas en esquinas opuestas tienen la misma velocidad, pero sentidos opuestos de giro con las otras [9].

5. ENTORNO DE SIMULACIÓN Y SOFTWARE.

5.1. INTRODUCCIÓN A COPPELIASIM.

CoppeliaSim es un simulador de entornos virtuales con robots usado en educación, en la industria y en investigación, conocido anteriormente como *Virtual Robot Experimentation Platform* (V-REP). Pertenece a la empresa *Coppelia Robotics, Ltd*. De Zúrich, Suiza. Es posible su instalación en los tres mayores sistemas operativos: Windows, IOS y Linux y existen tres versiones distintas, para nuestras simulaciones utilizaremos la versión Edu:

- **Player:** Con todas las posibilidades de simulación y uso comercial pero limitado en la capacidad de edición. Gratuito para todo el mundo.

- **Edu:** Capacidad completa de simular y editar, pero no se puede utilizar para uso comercial, ya que está dedicado a universidades, profesores, estudiantes o centros educativos.

- **Pro:** Con esta versión tenemos acceso a todas las funcionalidades de simulación y edición y además se puede dar uso comercial, es la única versión no gratuita, dedicada a profesionales.

Gracias a *CoppeliaSim* es posible simular en 3D sistemas de automatización de fábricas, presentar un producto o diseñar un prototipo y verificar su funcionamiento, entre otras funciones. Puede utilizarse como única aplicación en los proyectos o integrarse con aplicaciones de un nivel más alto. Para explorar todas las posibilidades se puede acceder al manual de *Coppelia Robotics* en el que se puede encontrar información sobre cualquier aspecto de la aplicación [10].

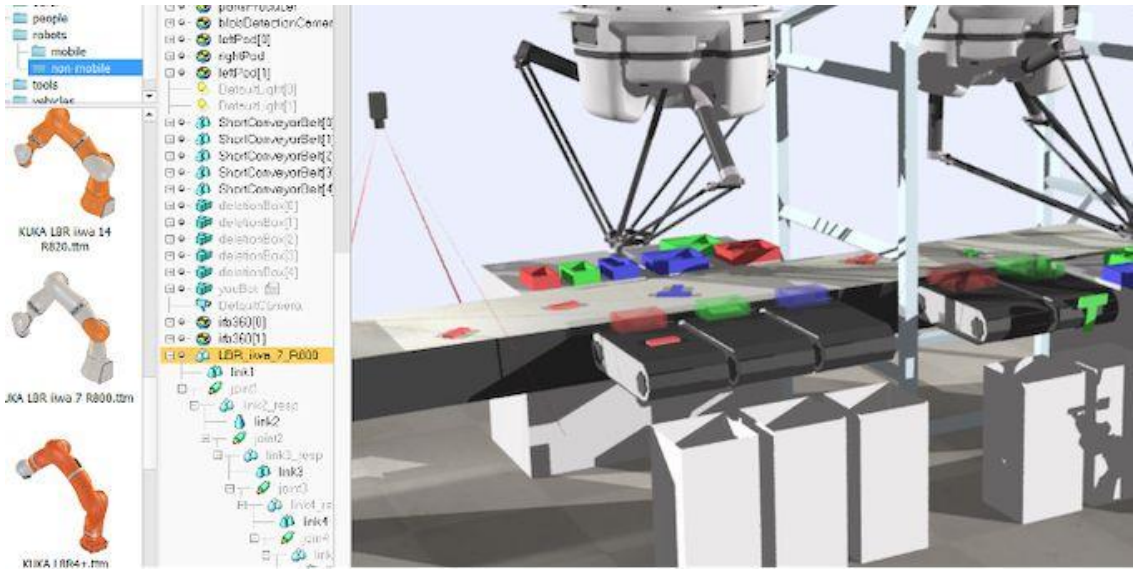


Figura 41. Ejemplo de escena en el software CoppeliaSim.
https://www.mathworks.com/products/connections/product_detail/coppeliasim.html

Este programa se rige por un algoritmo de control en el que cada objeto puede ser controlado independientemente (incluso con la simulación en funcionamiento), ya sea

mediante scripts embebidos, *plugins* o *remote API* entre otros. Entre los objetos del simulador tendremos sensores de visión y de proximidad, robots, figuras inertes y partículas dinámicas que simulan el agua o aire. Para añadir los objetos a la simulación será suficiente con seleccionar en el navegador el objeto deseado y arrastrar a la simulación.

Este simulador permite obtener resultados similares a la realidad ya que trabaja con cinco motores físicos distintos, en cada uno de los modelos existen diferencias y limitaciones y sólo uno de ellos puede estar activo en cada instante, aunque en el momento que sea necesario se pasará de uno a otro, lo que hace que gane mayor precisión, y, por lo tanto, mayor parecido con la realidad. Por lo tanto, podremos conseguir objetos que colisionan entre ellos, caen, o, incluso, permitirá el funcionamiento de una cinta para que transporte las piezas.

A la hora de trabajar con robots en las simulaciones, si queremos que tengan mayor interacción con el entorno cabe la posibilidad de añadir sensores de proximidad, para el cálculo de distancias, por ejemplo, o, de visión con su posterior capacidad de procesamiento de imágenes. En cambio, si lo que necesitamos es recoger información relevante de la simulación podemos emplear los objetos de tipo *Graph,* los cuales permiten almacenar datos en listas secuenciales que se podrán mostrar en gráficos.

Las instrucciones pueden ser enviadas en distintos lenguajes de programación siendo los más comunes *C/C++*, *Python* o *Lua* y además hay registradas más de 400 funciones *API* que permitirán obtener y modificar información de la simulación.

En este proyecto para enviar las órdenes emplearemos los clientes *remote API*, los cuales consisten en mandar las instrucciones a *CoppeliaSim* mediante una aplicación de código externa al programa, esto se realizará en nuestro caso con *Pycharm*, donde tendremos todas las instrucciones y cálculos y utilizaremos *CoppeliaSim* para visualizar los resultados que obtendríamos al realizar distintas acciones con el robot.

5.2. INTERFAZ DEL USUARIO.

Al abrir *CoppeliaSim* encontraremos varias ventanas, de las cuales cada una tiene su función. En primer lugar, aparecerá la ventana de consola, en Windows esta ventana se crea y se minimiza directamente, en cambio, en Linux, *CoppeliaSim* se inicia directamente desde la consola y queda visible a lo largo de la sesión, esta ventana es meramente informativa sobre los *plugins* que se han cargado y si se ha inicializado la sesión correctamente. Posteriormente, se abre la ventana principal, la de la aplicación, con la que podemos editar las escenas y realizar simulaciones. Por último, a lo largo de la interacción con la aplicación nos encontraremos diversas ventanas de diálogo utilizadas principalmente para el ajuste de parámetros.

Una vez abierta la ventana de la aplicación, siempre nos encontraremos con una escena en blanco que se crea por defecto, podremos trabajar directamente sobre esta o abrir otra en la que estemos trabajando. En las siguientes figuras podemos observar los distintos elementos que encontraremos en la ventana por defecto:

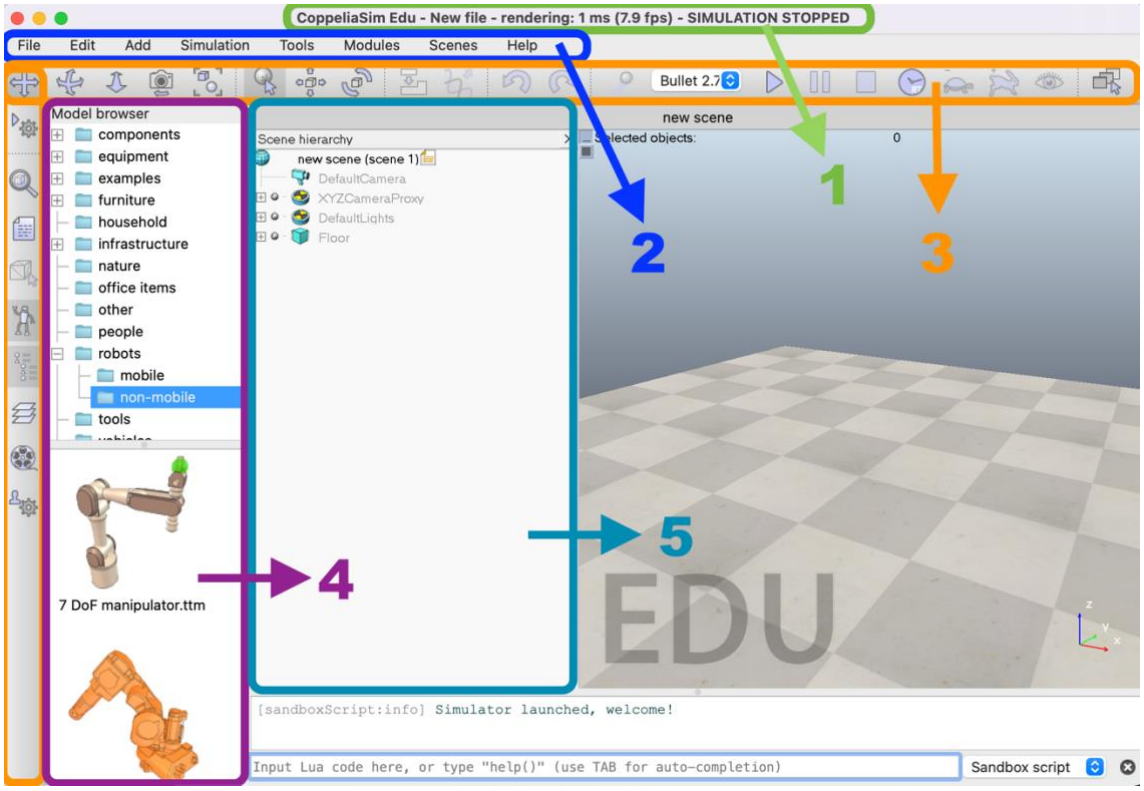


Figura 42. Interfaz de CoppeliaSim.

1. **Barra de la aplicación**: Indica el tipo de licencia que se está utilizando, el nombre de la escena que está activa y el estado actual de la simulación.

2. **Barra de menú:** Permite acceder a la mayoría de las funcionalidades de la aplicación. También es posible acceder a muchas de estas funciones haciendo doble clic sobre elementos en la jerarquía de la escena.

3. **Barras de herramientas:** La mayoría de las funciones presentadas en las barras de herramientas también están disponibles en el menú, pero al ser las más frecuentes las encontramos más accesibles. Puede elegirse que se muestren o se oculten según las necesidades.

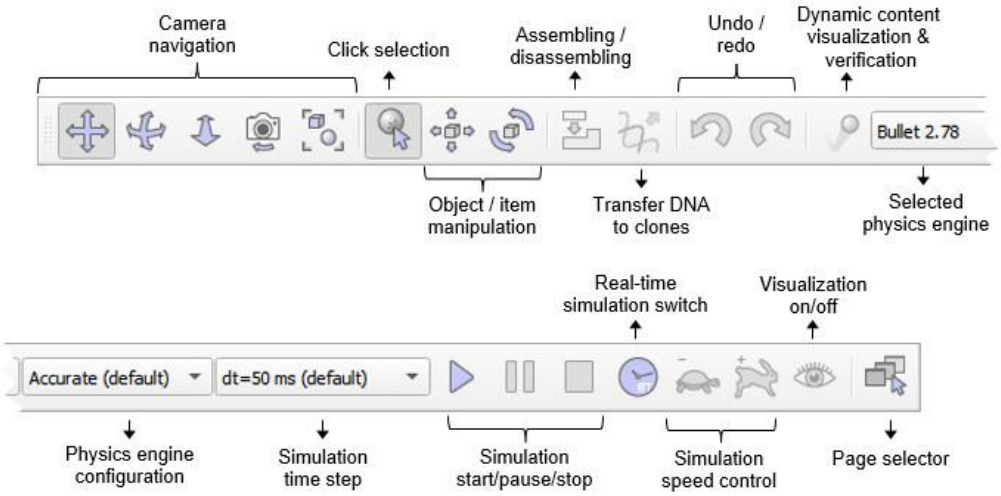


Figura 43. Ejemplo de escena en el software CoppeliaSim.

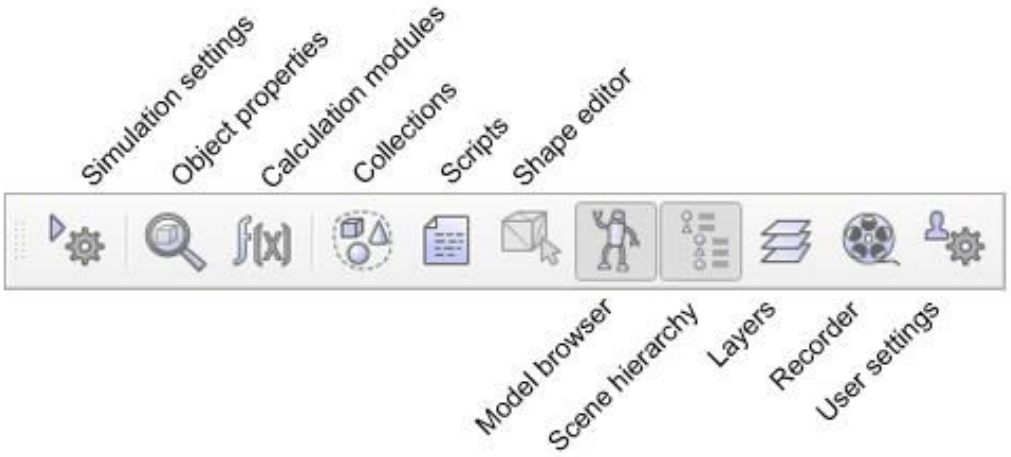


Figura 44. Ejemplo de escena en el software CoppeliaSim.

4. **Navegador de modelos:** En esta ventana encontramos dos secciones, en la superior se encuentra la estructura de carpetas donde se encuentran los modelos, y, en la inferior, las miniaturas de los modelos que contiene la carpeta seleccionada. Los modelos pueden arrastrarse directamente a la escena.

5. **Jerarquía de la escena:** Muestra el contenido de una escena en forma de árbol, en la que cada elemento puede expandirse o contraerse. Se puede hacer doble clic sobre cada uno de los elementos para modificar sus propiedades, los objetos pueden ser arrastrados a través del árbol para crear relaciones padre-hijo entre ellos.

5.3. ESCENAS DE SIMULACIÓN.

Una escena en una simulación es el elemento principal que engloba al resto de objetos y parámetros, al exportar una escena completa seremos capaces de recrear la misma simulación en cualquier sistema.

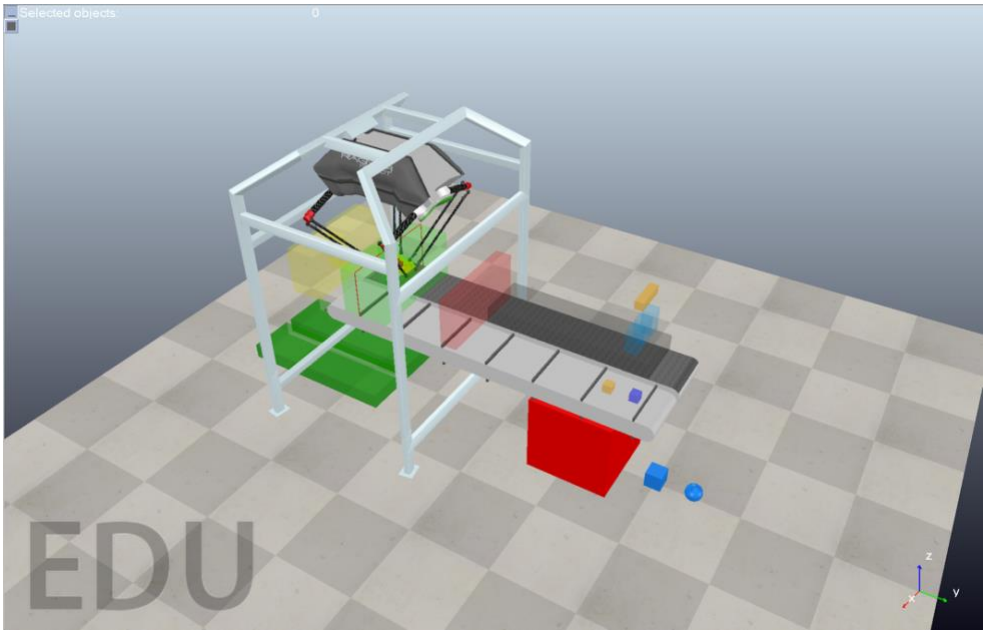


Figura 45. Ejemplo de escena en el software CoppeliaSim.

Al crear una escena, por defecto están incorporados una serie de objetos que hacen posible su visualización, como pueden ser los objetos cámara que asociados a una vista permiten observar la simulación, además de objetos de luz que iluminan la escena. Las escenas, una vez configuradas pueden guardarse en un archivo "*.*ttt*" para abrirse tanto desde el explorador de archivos con doble clic como desde *CoppeliaSim* en la opción "Abrir escena".

5.3.1. MODELOS.

Una escena estará compuesta a su vez por distintos elementos, mayormente por modelos, los modelos son representaciones simplificadas de un sistema real, como puede ser un robot, configurado con un determinado comportamiento y compuesto a su vez por objetos simples de la escena y por *Child scripts*, elementos que puede compartir con la escena.

Los modelos no pueden existir por sí mismos, dependen de una escena para ser ejecutados, sin embargo, sí pueden guardarse en archivos independientes del tipo "*.*ttm*" para ser reutilizados en distintas escenas.

Para añadir un modelo a la escena será suficiente con arrastrar desde el navegador de modelos hasta el propio escenario de la escena el que deseamos añadir, una vez añadido, lo podremos observar en la jerarquía de la escena, junto a cada uno de los objetos individuales que lo componen. Al hacer doble clic sobre el objeto en la jerarquía de la escena se abrirá una ventana en la que observaremos el *ChildScript* en el que se recoge la información del modelo. En su defecto, también será posible crear nuestros propios modelos [11].

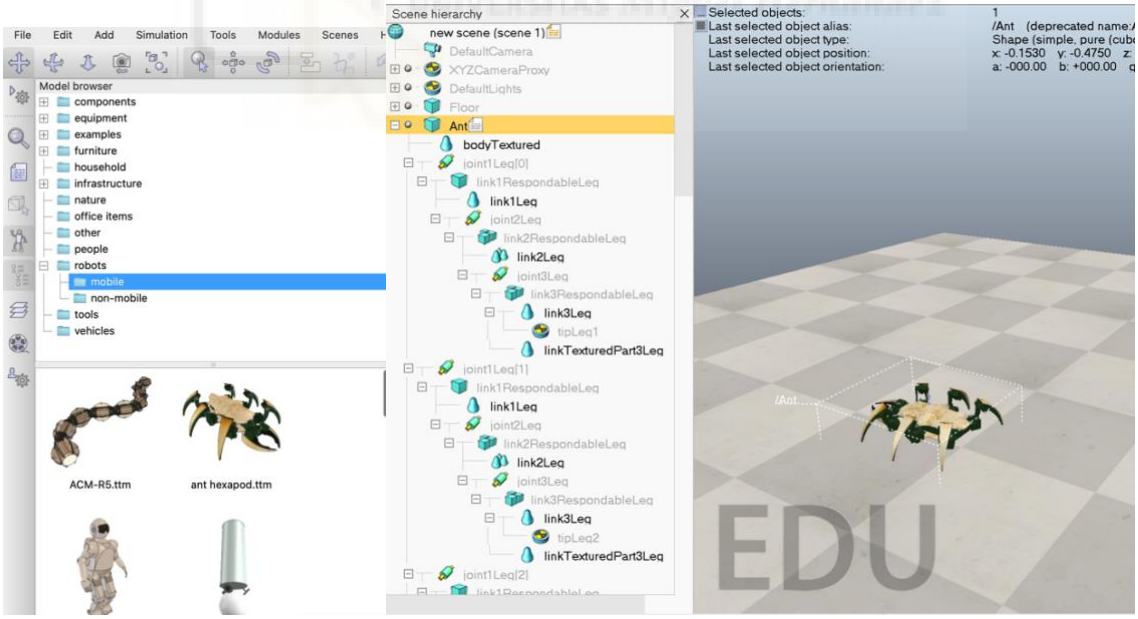


Figura 46. Navegador de modelos y Jerarquía de la escena en CoppeliaSim.

Las propiedades de un modelo se podrán ajustar en la ventana de propiedades del modelo, al pulsar sobre el icono de modelo (la esfera gris junto al nombre del modelo):

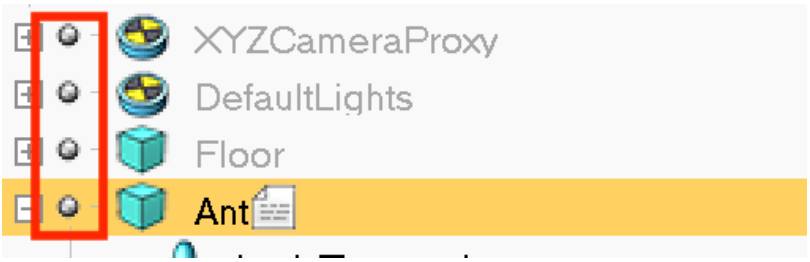


Figura 47. Icono que indica que un objeto de la escena es un modelo.

En este menú podremos modificar la miniatura del modelo, se puede añadir información relacionada, como el autor del modelo si va a ser compartido a otros usuarios o activar y desactivar propiedades como si el modelo necesitamos que sea visible, medible o detectable.

5.3.2. ENTORNO.

El entorno define una serie de parámetros y propiedades que son parte de la escena pero que no dependen de ningún modelo, como los colores del fondo, y que son modificables. Cuando modifiquemos el entorno únicamente se podrá guardar al guardar la escena.

Al igual que ocurre con los modelos, existe una ventana de propiedades del entorno, que puede abrirse al hacer clic sobre el icono en la jerarquía de la escena, donde se pueden modificar estas configuraciones. Será posible realizar ajustes sobre los colores del fondo, luces del ambiente o parámetros de difuminado, estos últimos únicamente afectan en la visualización de la escena, sin interactuar con los modelos.

Una propiedad por destacar es la ventana de texturas, que permite modificar las texturas relacionados con una forma. Una textura es una imagen que puede ser aplicada a las superficies para dar un aspecto más real, pueden importarse desde un archivo (JPEG, PNG, GIF, etc.) o generarse gracias a los sensores de visión.

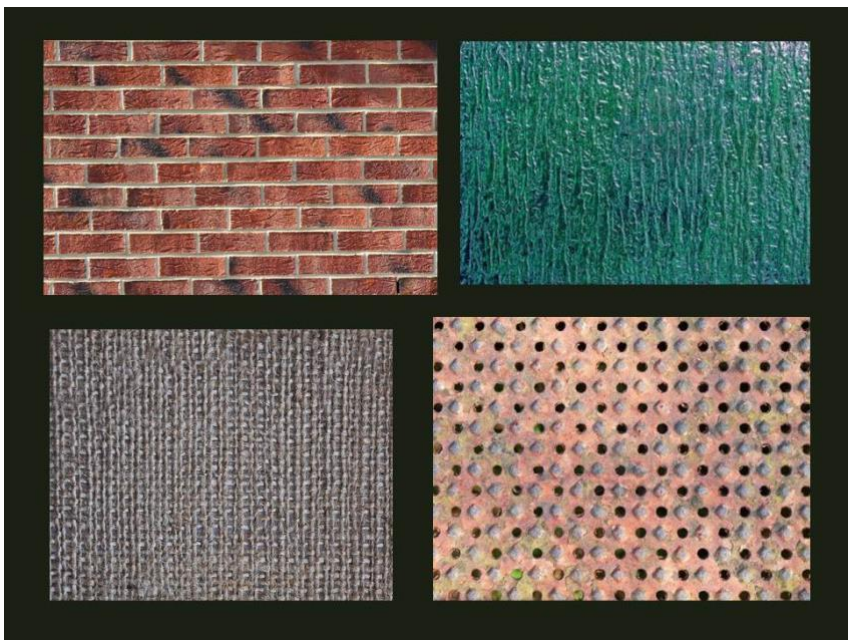


Figura 48. Ejemplos de texturas que podemos aplicar a objetos en CoppeliaSim. https://navegandoenclase.blogspot.com/2012/06/textura.html

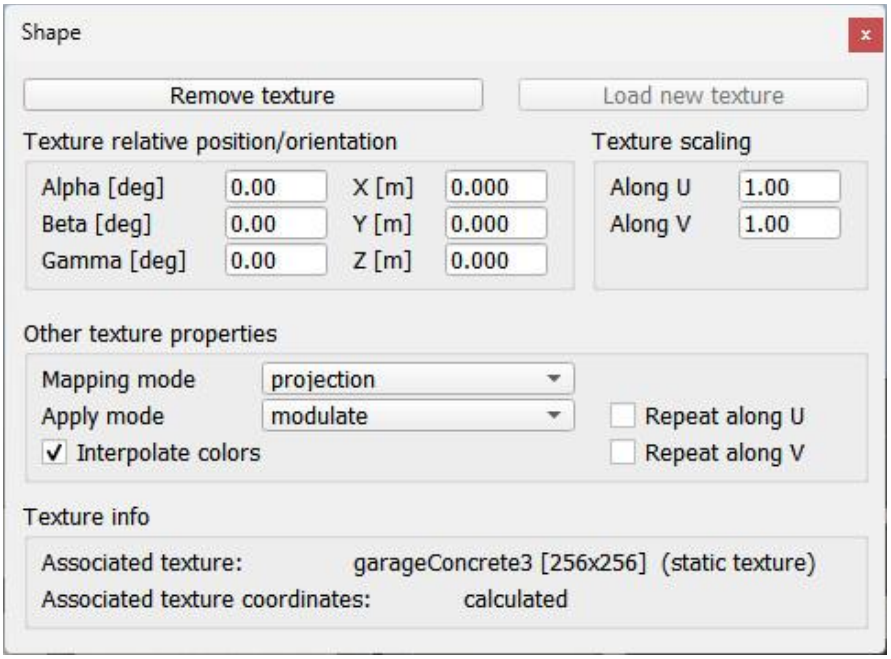


Figura 49. Ventana de configuración de textura.

5.3.3. PÁGINAS.

En ocasiones, para una misma escena necesitaremos observar distintos puntos de vista para un mismo instante, para lo que podemos utilizar las páginas. En las páginas podremos agrupar distintas vistas, enfocadas a un punto fijo, enfocadas sobre un objeto en concreto o asociarse a un objeto de visión. Existe la opción de crear para una misma escena hasta 8 páginas de libre configuración.

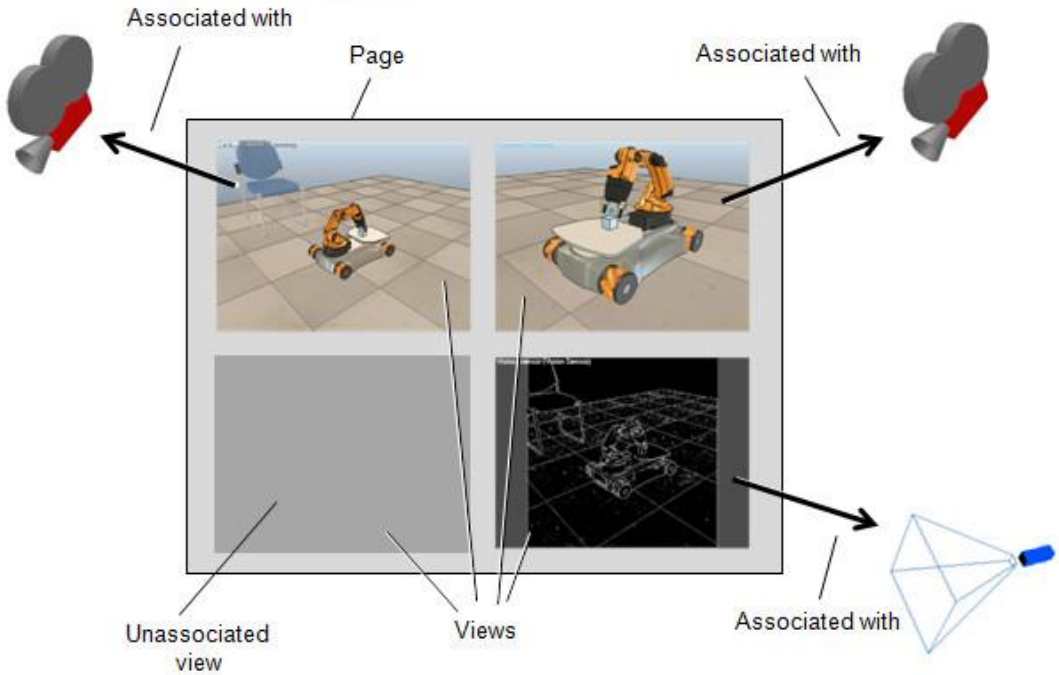


Figura 50. Ejemplo de configuración de página con relación entre vistas y objetos de visión. https://manual.coppeliarobotics.com/index.html

5.4. SENSORES EN LA SIMULACIÓN.

Los sensores se caracterizan por dar la capacidad de sentir e interactuar al robot, ya que le permiten detectar objetos, sonidos, colores o incluso la temperatura. Así mismo, los sensores permiten que el robot pueda recibir información de su entorno y que le ayude a la toma de decisiones autónomas.

Para añadir un sensor en una escena deberemos pulsar sobre "Add" y colocar el cursor sobre el tipo de sensor que deseamos, para que aparezca un nuevo desplegable con las opciones disponibles, por último podremos modificar sus propiedades como en el resto de objetos.

En las simulaciones de CoppeliaSIm encontraremos tres tipos de sensores: sensores de proximidad, de visión y de fuerza, en los que profundizaremos en los siguientes apartados.

5.5. SENSORES DE PROXIMIDAD.

Tanto en la vida real como en la simulación un sensor de proximidad detecta objetos o señales que se encuentran dentro del rango del elemento sensor. En el simulador podremos encontrar 6 tipos diferentes, clasificados dependiendo de la forma del haz de detección del sensor empleado, como se puede observar en la Figura 12.

Figura 51. Tipos de sensores de proximidad, de izquierda a derecha: rayo, pirámide, cilindro, disco y cono/rayos aleatorios. https://manual.coppeliarobotics.com/en/proximitySensorDescription.htm

Los sensores de proximidad realizan una medida desde su punto de detección hasta cualquier entidad detectable dentro de su volumen de detección. En otros simuladores los sensores solo tienen en cuenta las situaciones en las que el objeto interfiere con los bordes de detección del sensor, pero en este caso se tiene en cuenta todo objeto que entre dentro de este volumen.

En *CoppeliaSim* los sensores sólo podrán detectar objetos de tipo *dummy*, formas, árboles OC, nubes de puntos y colecciones de objetos. Sin embargo, esto no quiere decir que cualquiera de estos objetos pueda ser detectado por cualquiera de los sensores de proximidad, existen limitaciones. Por ejemplo, las nubes de puntos y los dummies, al estar formados por puntos, no podrán ser detectados por sensores del tipo rayo o rayo aleatorio.

Además, también podremos seleccionar voluntariamente qué objetos queremos que sean detectables y qué objetos no, esta opción la podremos marcar y desmarcar en las propiedades del objeto.

Para trabajar con este sensor una de las funciones más utilizadas es:

Res, dist, point, obj, n = Sim.CheckProximitySensor(sensorHandle, entityHandle)

Ya que introduciendo parámetros como el sensor a tomar y el objeto a detectar devolverá los siguientes valores:

- *Res*: 0 cuando no haya detección de ningún objeto y 1 cuando sí se detecte.
- *Dist*: Distancia desde el sensor a al objeto. Sólo tendrá valor cuando sea Res=1.
- *Point*: Posición respecto al sensor del punto del objeto detectado. Sólo tendrá valor cuando sea Res=1
- *Obj*: Objeto detectado. Sólo tendrá valor cuando sea Res=1
- *N*: vector normal de la superficie del punto detectado. Sólo tendrá valor cuando sea Res=1

En nuestro proyecto se utilizará esta función para detectar la posición de las piezas en el extremo de la cinta transportadora, para provocar que al detectar una pieza la cinta pare hasta que ésta sea recogida por el robot, posteriormente, se reactivará de nuevo la cinta hasta que llegue una nueva pieza.

5.5.1. SENSORES DE FUERZA.

Se colocan como enlaces rígidos entre dos formas entre las que se tienen que transmitir fuerzas y torques, su rigidez es modificable y pueden crearse condiciones que rompan el enlace cuando se supere una cierta fuerza.

Un sensor de fuerza realiza la medición del par de 3 valores, representando las fuerzas realizadas en cada uno de los ejes. Las fuerzas también pueden ser tomadas directamente de las articulaciones, pero en este caso solamente obtendremos las realizadas a través del eje Z, por lo que si necesitamos información del resto de ejes son muy útiles.

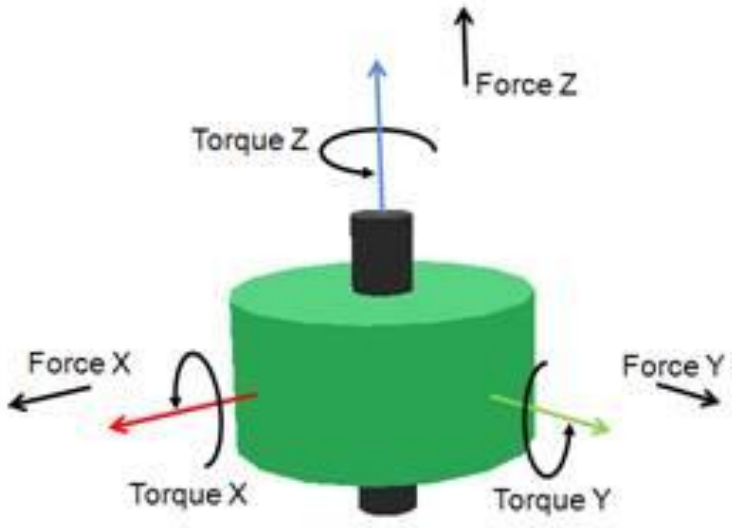


Figura 52. Fuerzas y torques capaces de medir un sensor de fuerza.
https://manual.coppeliarobotics.com/en/forceSensors.htm

5.5.2. SENSORES DE VISIÓN.

Los sensores de visión entran dentro del grupo de objetos visibles, objetos que permiten ver a través de ellos o que pueden mostrar alguna imagen, en este grupo se encuentran las cámaras y los sensores de visión. La característica principal de estos objetos es que pueden asociarse a una vista en la simulación y ver lo que están proyectando, esto lo podemos conseguir pulsando botón derecho sobre el sensor en la simulación, posteriomente poniendo el cursor sobre "*View*" y hacer clic sobre la opción "*Associate view with selected vision sensor*".

Pueden capturar objetos en su campo de visión y filtrarlos con un determinado límite y se recomienda utilizarlos en lugar de los sensores de proximidad cuando en la intervención de objetos intervienen colores, luces o estructuras [12].

Los sensores de visión tienen una resolución fija que no es posible modificar y muestra principalmente formas, a diferencia de las cámaras con las que podemos observar todo tipo de objetos. Podremos utilizar las imágenes tomadas por el sensor con funciones a través de la *API* y procesarlas.

Como sucede con los sensores de proximidad, en los sensores de visión también tendremos varias opciones y su elección dependerá de la aplicación que queramos dar:

- De proyección ortográfica: El campo de visión de este tipo de sensores es rectangular, son apropiados para aplicaciones de corto alcance.

- De proyección en perspectiva: El campo de visión en perspectiva es trapezoidal y son adecuados cuando se van a utilizar como cámara.

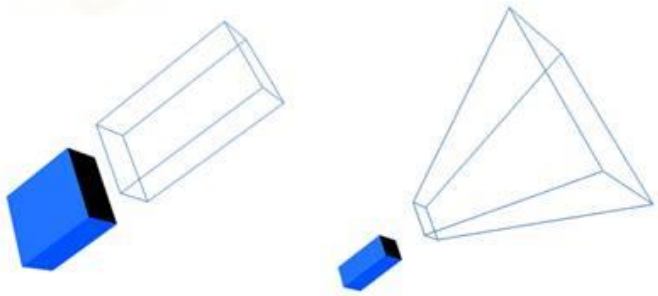


Figura 53. Tipos de sensor de visión CoppeliaSim. https://manual.coppeliarobotics.com/index.html

Los sensores de visión tienen gran importancia en las simulaciones por sus numerosas aplicaciones, en este proyecto se emplea un sensor de visión de proyección en perspectiva para la toma de imágenes. Se ha elegido procesar estas imágenes mediante la librería *openCV* desde el código remoto para realizar un filtrado hasta obtener la únicamente la forma de la pieza, medir su tamaño y realizar una clasificación en relación si el tamaño obtenido es grande o pequeño, el uso de las funciones de la librería se detallará en el siguiente capítulo.

6. SIMULACIONES REALIZADAS Y RESULTADOS.

6.1. ESCENARIO DE SIMULACIÓN.

El objetivo principal de la simulación de este proyecto consiste en que el robot *Youbot*, un robot manipulador con base móvil, sea capaz de tomar piezas y, mediante visión por computador, las clasifique en base a su tamaño y llegue a depositarlas en cada una de las mesas.

Se ha creado una escena desde cero en *CoppeliaSim*, que podemos observar en la Figura 54, en la que se ha añadido la cinta transportadora, el modelo del robot *Youbot* desde la biblioteca de modelos, al igual que las mesas, la mesa marcada con el número 3 en la imagen será la mesa en la que se depositarán las piezas grandes y en la número 4 se depositarán las pequeñas. Se ha modificado el suelo y se han aplicado texturas en las mesas para dar mayor sensación de realismo. Para los cálculos se ha tenido en cuenta que todos los parámetros recibidos de *CoppeliaSim* siempre tendrán las unidades de metros, kilogramos, segundos y radianes, en cada caso correspondiente.

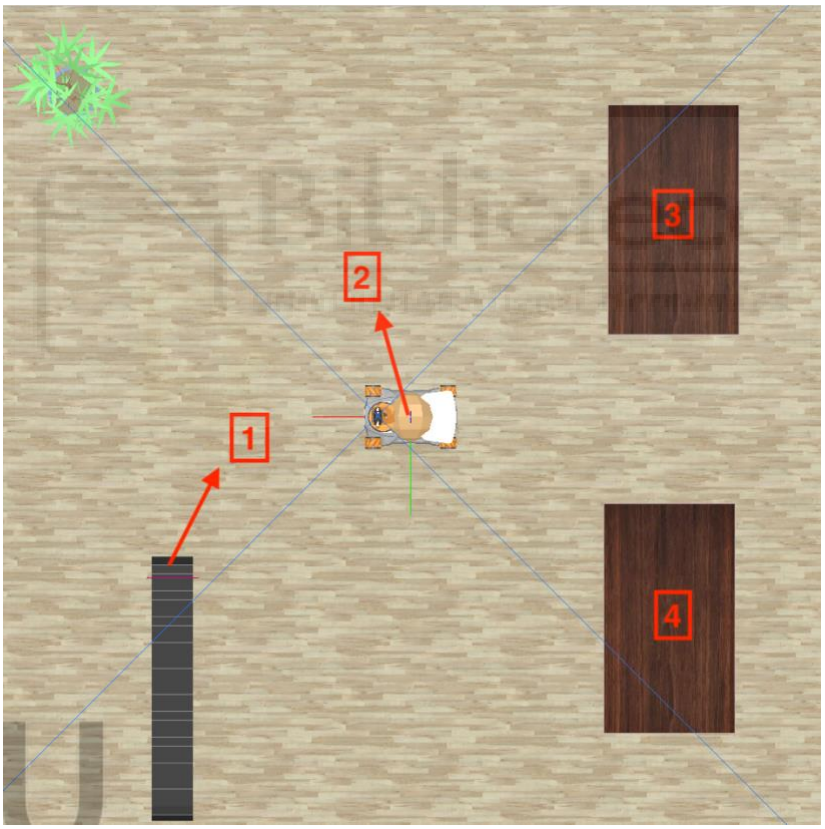


Figura 54. Objetos de la escena numerados: 1. Cinta transportadora, 2. Robot Youbot, 3. Mesa para piezas grandes 4. Mesa para piezas pequeñas.

Entre las librerías añadidas en el código encontramos *numpy*, para trabajar con vectores y matrices, algo muy útil al trabajar con coordenadas en 3D, y la librería *openCV*, para el procesamiento de imágenes. El código del script principal, ***youbot_move.py***, también depende de otros scripts contenidos en la librería *pyARTE* que contienen las funciones de inicialización y control general de los objetos que se emplean, como el robot, la cámara o las pinzas.

Para cada uno de los objetos sobre los que necesitemos tener control en la simulación es necesario conectar nuestro código con su servidor en *CoppeliaSim*, para lo que se utilizaron las funciones de inicialización anteriormente mencionadas. Para esto, en primer lugar, se deberá conseguir el control de la simulación. Como se comentaba en capítulos anteriores existen varias *API* remotas que permiten realizar esto, en nuestro caso se ha utilizado la *API* remota *ZeroMQ*, ofrece todas las funciones que tendríamos disponibles en *CoppeliaSim* (todas las que comiencen por *sim.**) tanto para *Python* como para otros lenguajes de programación como *java* o *C++*. Este proceso de conexión se realiza de forma oculta para el usuario y previamente se necesita instalar el paquete del cliente, esto puede realizarse con el siguiente comando: "**$ python3 -m pip install coppeliasim-zmqremoteapi-client**", una vez instalado, se puede importar para su uso en el código de la siguiente forma:

```
from coppeliasim_zmqremoteapi_client import *
```

Figura 55. Importación de ZeroMQ.

Después, desde el código principal se ha llamado a la clase ***simulation*** junto a la definición ***start()***, en la Figura 57 se puede ver como se ha llamado a esta definición en el código principal, y, en la Figura 56, el contenido de esta. En esta definición aparece ***RemoteAPIClient()***, en esta parte del código es dónde se realiza la conexión, se toma la simulación, su estado y el identificador de cliente.

```
class Simulation():
    def __init__(self):
        self.sim = None
        self.client = None
        self.simulation_speed = 3

    def start(self):
        """
        Connect python to the server running on Coppelia.
        """
        # Python connect to the V-REP client and start simulation
        self.client = RemoteAPIClient()
        self.sim = self.client.getObject('sim')
        # self.client.setStepping(True)
        state = self.sim.getSimulationState()

        # simulation is stopped
        if state == 0:
            self.sim.startSimulation()
        else:
            # try to stop the simulation if is in a zombie state
            self.sim.stopSimulation()
            time.sleep(3)
            self.sim.startSimulation()
        # apply stepping True after the simulation is actually created
        self.client.setStepping(True)
        # Modify simulation speed to get a nicer result
        self.sim.setInt32Param(self.sim.intparam_speedmodifier, self.simulation_speed)
        print('CONNECTED TO COPPELIA!')
```

Figura 56. Clase SImulation() y definición start().

Al realizar la conexión con el servidor de *CoppeliaSim* desde *Python* y comenzar la simulación se obtiene el valor de la variable ***ClientID***, este valor es muy importante ya que es el identificador de la conexión con el servidor para la simulación y se utiliza como parámetro para realizar la conexión del resto de objetos en la simulación correctamente, en la Figura 57 podemos observar cómo se aplica esto a los objetos:

```
simulation = Simulation()
clientID = simulation.start()
robot = YouBotRobot(clientID=clientID)
robot.start()
obj = Dummy(clientID=clientID)
obj.start()
```

Figura 57. Llamada a simulation.start() y uso de la variable ClientID.

Para el resto de los objetos, su inicialización sigue una estructura muy similar, ya que todos utilizan la misma función dentro de la definición ***start (self, name):***

"Objeto_a_conectar" = self.simulation.sim.GetObject("Nombre_Objeto_a_conectar")

```
def start(self, name='/conveyor/prox_sensor'):
    prox_sensor = self.simulation.sim.getObject(name)
    self.proxsensor = prox_sensor
```

Figura 58. Código para iniciar el control del sensor de la cinta transportadora.

En cuanto a la nomenclatura cuando necesitemos llamar a los objetos de la escena en el código es importante destacar que deberá ser de la forma "/Nombre_Objeto" y cuando se trate de un objeto que se encuentra anidado dentro de otro deberá ser "/Nombre_Objeto_1/Nombre_Objeto_2" siendo los nombres con los que aparecen en la jerarquía de la escena.

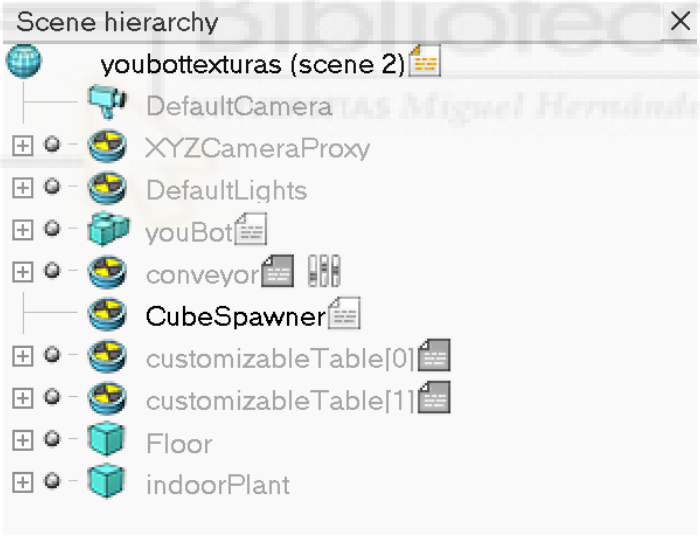


Figura 59. Jerarquía de la escena creada en CoppeliaSim.

La jerarquía de la escena quedó compuesta por modelos en su mayoría, que a su vez contienen objetos de diferentes tipos. Los modelos del robot *YouBot* y la cinta transportadora contienen detalles más complejos por lo que se verán en detalle en los puntos 6.1.1. y 6.1.2. de este mismo capítulo. El suelo y la planta son modelos predeterminados de la biblioteca de modelos de *CoppeliaSim* y se añadieron a la simulación realizando un arrastre. Por último, las mesas, también son modelos añadidos desde la biblioteca, aunque este tipo de mesa permite mayor personalización que otros objetos, se encuentran formadas por objetos del tipo juntas y formas, habiendo añadido la textura en la parte superior, llamada "*top*", aunque también podría personalizarse el resto de la mesa si se desea.

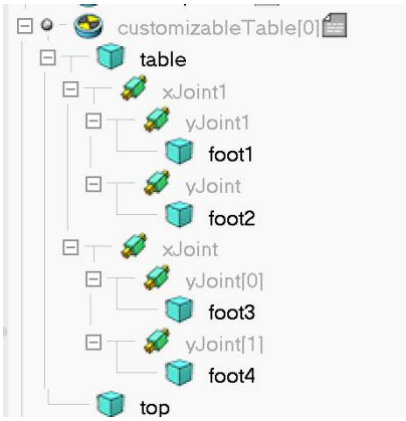


Figura 60. Jerarquía en la escena del modelo customizableTable.

6.1.1. ROBOT YOUBOT.

El modelo del robot *YouBot* que se ha añadido a la escena está compuesto por objetos individuales tanto para el brazo como para la base. En la base se toman 4 articulaciones actuadoras que corresponden a las cuatro ruedas de la base, que se diferencian por nomenclatura: **fl, rl, rr** y **fr,** la primera letra hace referencia a si la rueda es delantera o trasera, *rear* o *front* en inglés, y la segunda letra a si es izquierda o derecha, *left* o *right*. En la jerarquía del modelo se agrupan de forma que dos de las ruedas van unidas a las otras dos mediante una articulación adicional, como se puede observar en la Figura 61, llamada "*Joint*", además de observar los distintos objetos que pertenecen a cada una de las ruedas y un último objeto fuera de esta jerarquía para la plataforma, siendo una forma compuesta ya que esta parte no realizará ninguna acción dinámica:

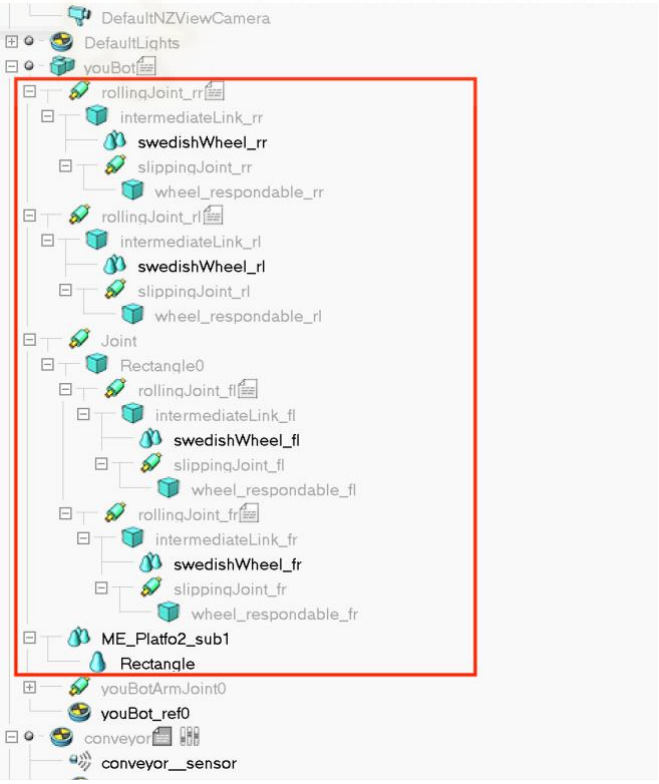


Figura 61. Jerarquía del modelo del robot Youbot - Base.

Por otra parte, para el brazo se tienen en cuenta las 5 articulaciones, anidadas desde la base al extremo y numeradas de 0 a 4, además del *gripper* y de la cámara colocada en el extremo del robot en nuestro caso.

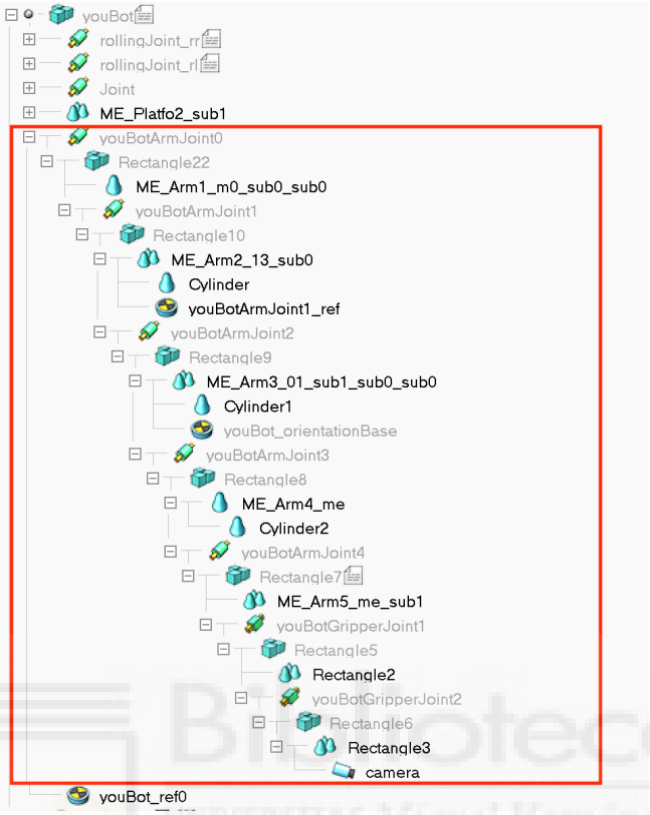


Figura 62. Jerarquía del modelo del robot Youbot - Brazo.

Otro objeto que destacar en el robot es el objeto del tipo *Dummy* llamado "youBot_ref0" que se ha colocado en el centro del modelo del robot que servirá para tomar los valores de posición y orientación en el espacio del robot de forma precisa.

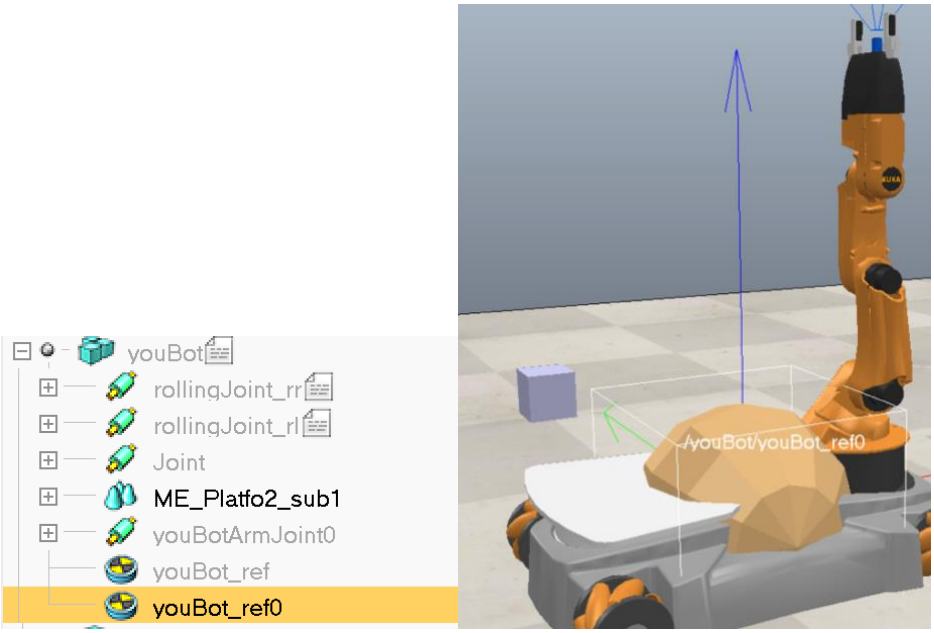


Figura 63. Jerarquía del modelo del robot Youbot - Objeto de referencia.

Para tomar el control del robot se ha tomado como base un script de la librería *pyARTE* [13] llamado "*Youbot.py*" en la que se han creado dos clases, una para la base y otra para el brazo, cada una de ellas contiene la inicialización con los valores de velocidades y rangos de movimiento de las articulaciones entre otros parámetros y la toma del control de los objetos, de las 5 articulaciones y las 4 ruedas, además, dentro de la clase de la base tendremos la definición ***set_base_speed*** que permite mover la base y se ha utilizado en el código principal:

```
def set_base_speed(self, forwardspeed, leftrigthspeed, rotspeed):
    """
    Given a speed in forward/backward direction
    - left/right directino
    and - rotation speed
    Computes the speeds of each wheel so a to apply the commanded speed to the robot base.
    """
    # B = np.array([b+d/r])
    # V = np.array([forwardspeed, leftrigthspeed, rotspeed])
    # w = np.dot(B, V)
    self.simulation.sim.setJointTargetVelocity(self.wheeljoints[0], forwardspeed - leftrigthspeed - rotspeed)
    self.simulation.sim.setJointTargetVelocity(self.wheeljoints[1], -forwardspeed + leftrigthspeed - rotspeed)
    self.simulation.sim.setJointTargetVelocity(self.wheeljoints[2], -forwardspeed - leftrigthspeed + rotspeed)
    self.simulation.sim.setJointTargetVelocity(self.wheeljoints[3], -forwardspeed + leftrigthspeed + rotspeed)
```

Figura 64. Código encargado de dar velocidad al robot.

Podremos controlar la velocidad de movimiento de la base hacia delante/atrás, izquierda/derecha y velocidad de rotación en sentido horario o antihorario, se selecciona el sentido en cada una de las opciones colocando velocidades positivas o negativas.

6.1.2. CINTA TRANSPORTADORA.

Uno de los objetos destacables de la simulación además del robot es la cinta transportadora que proporcionará las piezas. Está formada por el modelo predeterminado de la cinta, un sensor de proximidad y un objeto *Dummy* llamado *CubeSpawner* en el que se ha añadido en su *ChildScript* la configuración que crea las piezas cuando es necesario.

El sensor elegido es un sensor de tipo rayo colocado a través del extremo de la cinta y sobre una altura superior a la cinta que permite detectar a las piezas. Condicionará a los otros dos objetos, ya que en el *ChildScript* de la cinta se ha añadido la condición de que únicamente avance cuando no esté activo, al igual que en el generador de piezas ya que sólo creará piezas cuando tampoco esté activo. Por lo tanto, mientras que no tengamos pieza alguna en la posición en la que el robot debe tomar los objetos se crearán piezas e irán avanzando. En el instante en el que la primera de las piezas llegue a esta posición y el sensor lo detecte, se paralizará tanto la cinta como la creación de piezas nuevas, hasta que el robot la tome y vuelva a quedar libre el sensor. Los *ChildScripts* completos para estos objetos se pueden ver en el Anexo para códigos, en el Capítulo 8.

```
function sysCall_actuation()
    beltVelocity=0.5

    if sim.readProximitySensor(sensor)>0 then
        beltVelocity=0
    end

    sim.writeCustomDataBlock(conveyor.model,'CONVMOV',sim.packTable({vel=beltVelocity}))

    _S.conveyor.actuation()
end
```

Figura 65. ChildScript de la cinta transportadora.

```
while sim.readProximitySensor(sensor) < 1 do

  if (i%2==0) then
    local spawn = sim.createPureShape(PIECE_TYPE, 10, {0.042,0.042,0.042}, 0.1, NULL)
    sim.setObjectPosition(spawn, -1, START_POS)
    sim.setObjectSpecialProperty(spawn, sim.objectspecialproperty_detectable_all)
   else
    local spawn = sim.createPureShape(PIECE_TYPE, 10, {0.032,0.032,0.032}, 0.1, NULL)
    sim.setObjectPosition(spawn, -1, START_POS)
    sim.setObjectSpecialProperty(spawn, sim.objectspecialproperty_detectable_all)
   end
  sim.wait(WAIT_TIME)
  i = i + 1
end
```

Figura 66. ChildScript del objeto generador de piezas, CubeSpawner.

6.2. FLUJO DE FUNCIONAMIENTO.

El objetivo de este apartado es comprender mejor la aplicación que se buscaba conseguir con los objetos anteriores y entender en qué momento es útil cada uno de ellos. A modo esquemático, en la Figura 67 se muestra el flujo de acciones principales que se llevan a cabo en la simulación. El robot comienza en la posición inicial de la simulación y se dirigirá hacia la cinta transportadora, con el brazo orientado en el mismo sentido que el movimiento, una vez llegue a una distancia horizontal a la altura de la cinta, el robot parará para orientarse con el brazo de cara a la cinta y se aproximará a ella.

La aproximación del robot a la cinta se ha dividido en dos partes ya que el punto intermedio de esta aproximación era la distancia más adecuada para tomar la imagen desde la cámara en el extremo del robot para realizar su medición posteriormente. Una vez tomada la foto, se finaliza la aproximación a la posición de pick. Colocará el brazo en su posición final mediante la cinemática inversa y cerrará pinza, para levantar el brazo ya con la pieza tomada y se alejará de la cinta hacia atrás.

Una vez el robot ha tomado la pieza debe alejarse primeramente hacia atrás ya que se coloca en una posición muy próxima para poder tomar la pieza en la que no existe espacio para realizar ningún otro movimiento sin chocar con la cinta. Por lo tanto, una vez se ha alejado, puede cambiarse de nuevo la orientación para orientarse en dirección a las mesas y comenzar su movimiento hacia ellas.

El movimiento hacia las mesas para dejar la pieza en un primer momento se ha hecho horizontal, ya que para ambas mesas había una distancia horizontal común que se podía resumir. Una vez llegado al punto de divergencia hacia una u otra, el robot debe decidir hacia cuál debe ir, esto se realiza mediante el procesamiento de la imagen tomada anteriormente.

Una vez el robot se ha dirigido a una de las mesas, es necesario llevar un recuento de cuántas piezas se ha depositado en cada una de ellas para respetar las distancias entre piezas, esto se realiza mediante un contador para cada mesa en el código y el robot recorrerá a lo largo de la mesa una distancia extra por cada una de las piezas.

El proceso de dejar la pieza será similar al de tomarla, se aproximará el brazo y se abrirá la pinza. Después, el robot volverá a alejar el brazo y posteriormente la base, para volver a la posición inicial y orientarse de nuevo en dirección al movimiento hacia la cinta para que continúe la simulación con una nueva pieza.

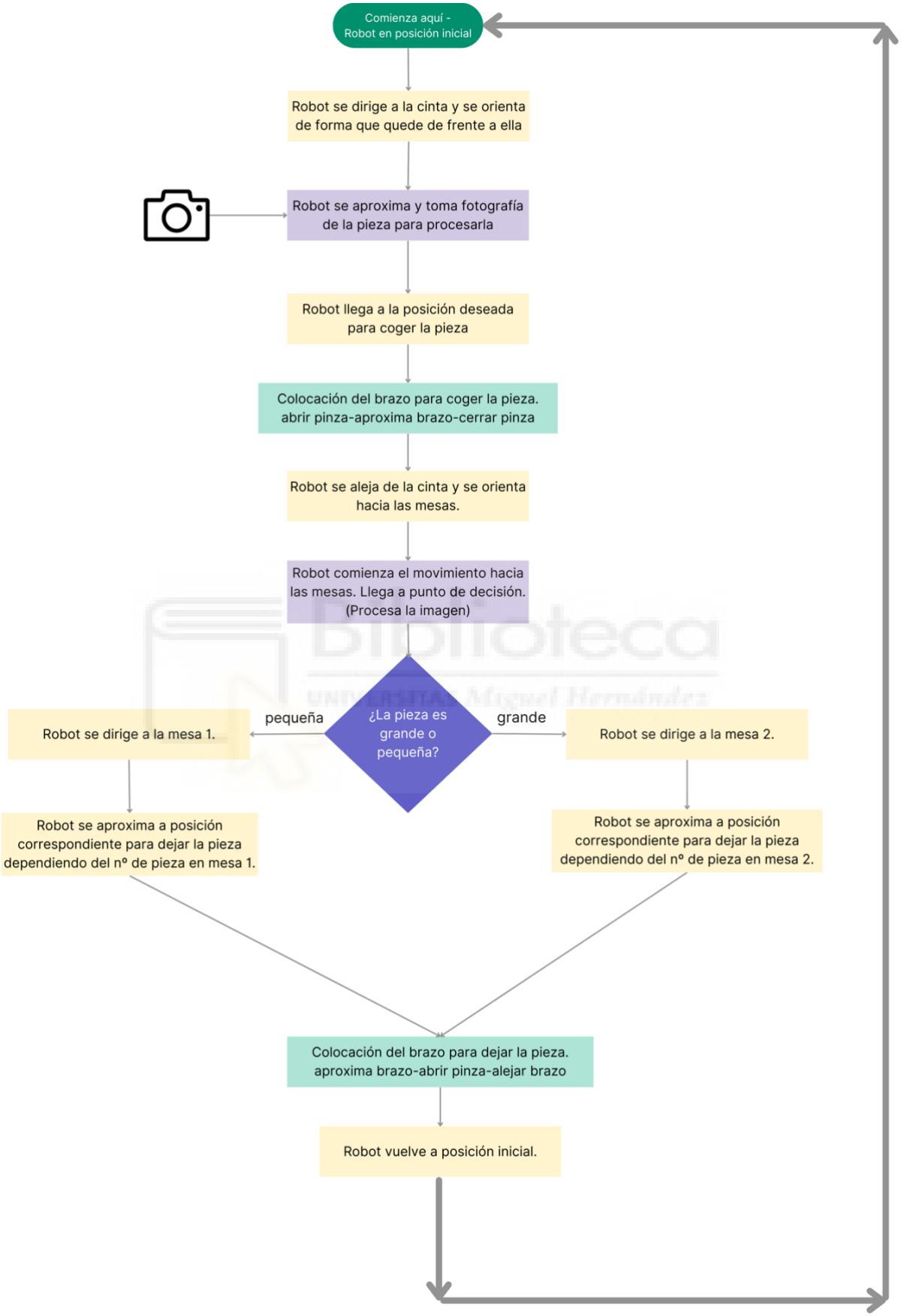


Figura 67. Diagrama del flujo de funcionamiento.

6.3. CONTROL DE POSICIÓN Y ORIENTACIÓN DE LA BASE MÓVIL.

Para el desplazamiento del robot a lo largo del plano disponíamos, como hemos visto en el apartado anterior, de la función de ***set_base_speed***, la cual nos permitía dar velocidad al robot, sin embargo, se necesitaba un control preciso de la distancia que se necesitaba recorrer con esa velocidad marcada. Para esto, en primer lugar, se plantearon los desplazamientos que debía hacer el robot para llevar a cabo su función. Principalmente son tres como muestra la Figura 67, un desplazamiento hacia la cinta donde tomará la pieza y otros dos hacia cada una de las mesas.

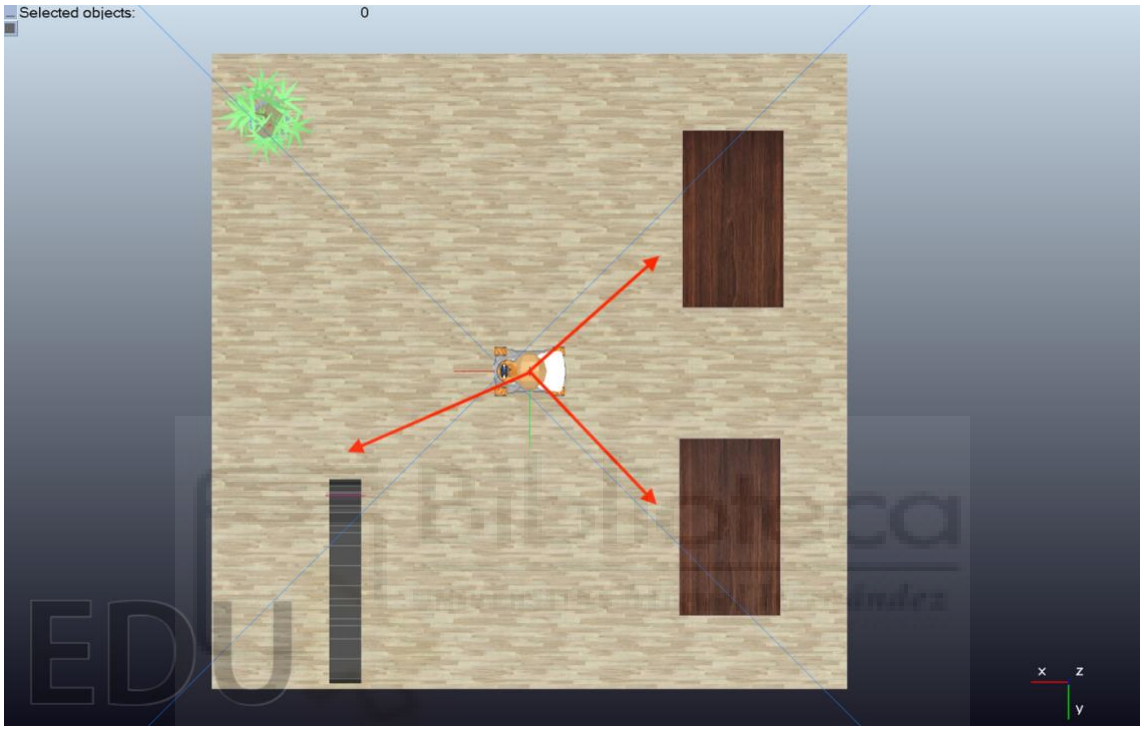


Figura 68. Descripción general de los desplazamientos del robot a lo largo de la simulación.

Se comenzó por crear dos definiciones, llamadas ***movimiento_x*** y ***movimiento_y***, en las que tomaba como parámetro el objeto que se toma como referencia y la posición objetivo a conseguir para este objeto, dependiendo de cuál llamáramos se tenía en cuenta la componente x o y (valor en el array de posición de 0 y 1, respectivamente), para hacer movimientos a lo largo de cada uno de los ejes.

```
def movimiento_x(obj, posicion_objetivo_x):

    posicion_robot = obj.get_position()
    posicion_robotx = posicion_robot[0]
    print(posicion_objetivo_x - posicion_robotx)
    while abs(posicion_objetivo_x - posicion_robotx) > 0.075 :
        posicion_robot = obj.get_position()
        posicion_robotx = posicion_robot[0]
        robot.wait(1)
```

Figura 69. Código inicial para movimiento de la base a lo largo del eje X.

Dentro de estas definiciones se creó un bucle en el que durante cada pulso de la simulación se tomara la posición del objeto de referencia del robot y se comparara la posición actual con la deseada, en el caso en el que haya una diferencia mayor del margen de error que

se ha marcado, es decir, que sea una posición que no sea lo suficientemente cercana a la deseada, sigamos en el bucle un pulso más. En el momento en el que esta condición no se cumpla porque el robot ha llegado a su lugar, se finalizará el bucle y pasará a la siguiente acción.

Se comenzaron a realizar pruebas con estas dos definiciones y a realizar los primeros desplazamientos, los cuales sólo podían realizarse a lo largo del eje x o y en cada caso. Posteriormente, esto se unificó para poder realizar movimientos con velocidades tanto en X como en Y y así poder realizar desplazamientos diagonales, quedando la definición siguiente:

```
def movimiento(robot, obj, posicion_objetivo_x, posicion_objetivo_y):

    posicion_robot = obj.get_position()
    posicion_robotx = posicion_robot[0]
    posicion_roboty = posicion_robot[1]

    while abs(posicion_objetivo_x - posicion_robotx) > 0.075 or abs(posicion_objetivo_y - posicion_roboty) > 0.075:
        posicion_robot = obj.get_position()
        posicion_robotx = posicion_robot[0]
        posicion_roboty = posicion_robot[1]
        robot.wait(1)
```

Figura 70. Código encargado de controlar el desplazamiento de la base del robot.

En todas las funciones en las que se miden posiciones del robot respecto a posiciones deseadas se ha colocado un pequeño margen de error que no afecte al funcionamiento general, ya que hacer coincidir las posiciones que toma a cada pulso con la posición ideal cuando el robot está en continuo movimiento sería imposible.

Una vez ya se tenía control de los desplazamientos, la siguiente problemática era orientar el robot frente a los objetos con los que tenía que interactuar, ya que en los desplazamientos siempre continuaba con la orientación inicial, es decir, con el brazo orientado hacia el eje X, como previamente se había explicado en la Figura 67 sobre el flujo de la simulación. Para resolver esto, se creó una nueva definición, de estructura similar a la de movimiento, llamada orientación, en la que en lugar de tomar las posiciones tomara las orientaciones del robot, y, tomando en concreto la del eje Z, la comparara con la deseada.

```
def orientacion1(robot, obj, orientacion_objetivo_z):
    orientacion_robot = obj.get_orientation()
    orientacion_robot_z = orientacion_robot[2]
    while (abs(orientacion_objetivo_z - abs(orientacion_robot_z)) > 0.12):
        orientacion_robot = obj.get_orientation()
        orientacion_robot_z = orientacion_robot[2]
        robot.wait(1)
```

Figura 71. Código encargado de controlar la orientación de la base del robot.

Por último, estos movimientos con velocidades altas al realizar desplazamientos o giros de gran tamaño tomaban poca precisión, ya que desde un pulso de la simulación al siguiente el robot había recorrido demasiada distancia como para ser lo suficientemente exacto. Disminuir la velocidad no era una opción ya que se alargaba mucho el tiempo de simulación, por lo que se optó por crear dos nuevas definiciones, las aproximaciones para desplazamientos y para giros. Estas definiciones se muestran en las Figuras 72 y 73, respectivamente.

```
def aprox(robot, obj, posicion_aproximacion, valor):
    posicion_robot = obj.get_position()
    posicion_roboty = posicion_robot[valor]
    while (abs(posicion_aproximacion - posicion_roboty) > 0.03):
        posicion_robot = obj.get_position()
        posicion_roboty = posicion_robot[valor]
        robot.wait(1)
```

Figura 72. Código para aproximación en el desplazamiento de la base del robot.

```
def orientacion2(robot, obj, orientacion_objetivo_z):
    orientacion_robot = obj.get_orientation()
    orientacion_robot_z = orientacion_robot[2]
    while (abs(orientacion_objetivo_z - abs(orientacion_robot_z)) > 0.006):
        orientacion_robot = obj.get_orientation()
        orientacion_robot_z = orientacion_robot[2]
        robot.wait(1)
```

Figura 73. Código para aproximación en la orientación de la base del robot.

El funcionamiento de las dos nuevas definiciones es igual a las anteriores con la única diferencia de que tienen un margen de error menor, que combinado con una menor velocidad para una mayor exactitud de los valores que se toman se consiguieran posiciones más cercanas a las deseadas y que no dieran problemas al funcionamiento que le queríamos dar. Inicialmente, en el desplazamiento, se dio un margen de error de 7,5 cm, que posteriormente, con la función de aproximación, se ha llegado a conseguir reducir a 3 cm. Para la orientación se consiguió reducir de 6,87 grados a 0,34 grados, por lo que la exactitud mejoró significativamente con este método.

En la siguiente imagen se muestra la combinación de estas definiciones con las velocidades, utilizando las velocidades altas para grandes movimientos y posteriormente velocidades pequeñas para las aproximaciones:

```
robot.set_base_speed(7, 0, 0)
movimiento(robot, obj, 1.3, 0)

robot.set_base_speed(2, 0, 0)
aprox(robot, obj, 1.47, 0)

robot.set_base_speed(0, 0, 0)
robot.wait(10)

robot.set_base_speed(0, 0, -12)
orientacion1(robot, obj, 1.22)

robot.set_base_speed(0, 0, -1)
orientacion2(robot, obj, 1.5708)
```

Figura 74. Ejemplo de control de desplazamientos y rotaciones del robot.

6.3. *PICK AND PLACE.*

El código principal se basa en la definición de *pick and place*, ya que la aplicación principal del objeto es tomar y dejar piezas, utilizando herramientas de visión por computador. Por lo que el proceso que seguirá el robot será de realizar un pick, tomar una pieza, y, dependiendo del tamaño, que se obtendrá en forma de 0 o 1 (0 para pequeño y 1 para grande) crearemos una condición que llame a cada una de las funciones place para dirigirse a cada una de las mesas. Además, se ha incluido aquí el recuento dentro de cada una de las condiciones las distancias a las que se debe ir dejando la pieza en cada una de las mesas, dependiendo de cuantas ya hay colocadas, para dejar una distancia adecuada, se profundizará en esto en el siguiente apartado de *place*.

Tanto para tomar como para dejar la pieza en un lugar se debe mover el brazo y, por lo tanto, se debe aplicar la cinemática inversa ya que conoceremos la posición en la que queremos colocar el extremo del brazo y se calcularán las coordenadas articulares necesarias. Para esto se ha añadido una definición con los cálculos explicados en el Capítulo 4 del robot *Youbot*, donde quedaban explicadas las ecuaciones que se han añadido al código, expuesta en la Figura 75. La definición de cálculo de la cinemática inversa devolverá los valores de las variables articulares q_1, q_2 y q_3. Para aplicar estos valores se creó otra definición adicional que llama a las articulaciones afectadas a tomar las posiciones obtenidas como resultado, actuando como la cinemática directa del robot.

```
def cin_inversa(x, y, orientacion):
    L1 = 0.155
    L2 = 0.135
    L3 = 0.181
    posicion = np.array([x, y, 0])
    vector = np.array([np.cos(orientacion), np.sin(orientacion), 0])
    pm = posicion - (L3 * vector)
    R = pow((pow(pm[0], 2) + pow(pm[2], 2)), 0.5)
    alfa = np.arctan2(pm[1], pm[0])
    beta = np.arccos((pow(L2, 2) - pow(L1, 2) - pow(R, 2)) / (-2 * L1 * R))
    gamma = np.arccos((pow(L2, 2) + pow(L1, 2) - pow(R, 2)) / (2 * L1 * L2))
    q2 = gamma - np.pi
    q1 = alfa + beta
    q3 = orientacion - q1 - q2
    return q1, q2, q3
```

Figura 75. Código para el cálculo de la cinemática inversa.

En algunas posiciones la fuerza del movimiento del brazo era demasiado fuerte y provocaba que el robot se inestabilizara durante unos segundos, lo que era un problema, por lo que en algunos casos se puede observar, como en la Figura 76, que se introdujo una posición intermedia para que el desplazamiento fuera menos brusco. En este caso en concreto se dividió en dos alturas, para que la inercia del movimiento hacia delante fuera menor.

```
robot.set_base_speed(0, 0, 0)
q1, q2, q3 = cin_inversa(0.28, 0.09, 0)
robot.set_arm_position(q1, q2, q3)
robot.wait(2)
q1, q2, q3 = cin_inversa(0.28, -0.065, 0)
robot.set_arm_position(q1, q2, q3)
gripper.open()
robot.wait(20)
```

Figura 76. Aplicación de posición intermedia para mayor control del brazo.

6.3.1. PICK

La tarea de pick tiene como objetivo desplazarse y tomar la pieza de la cinta transportadora, siguiendo la imagen de la Figura 77 que muestra la serie de desplazamientos que realiza el robot, procedemos a realizar en este apartado un análisis del proceso.

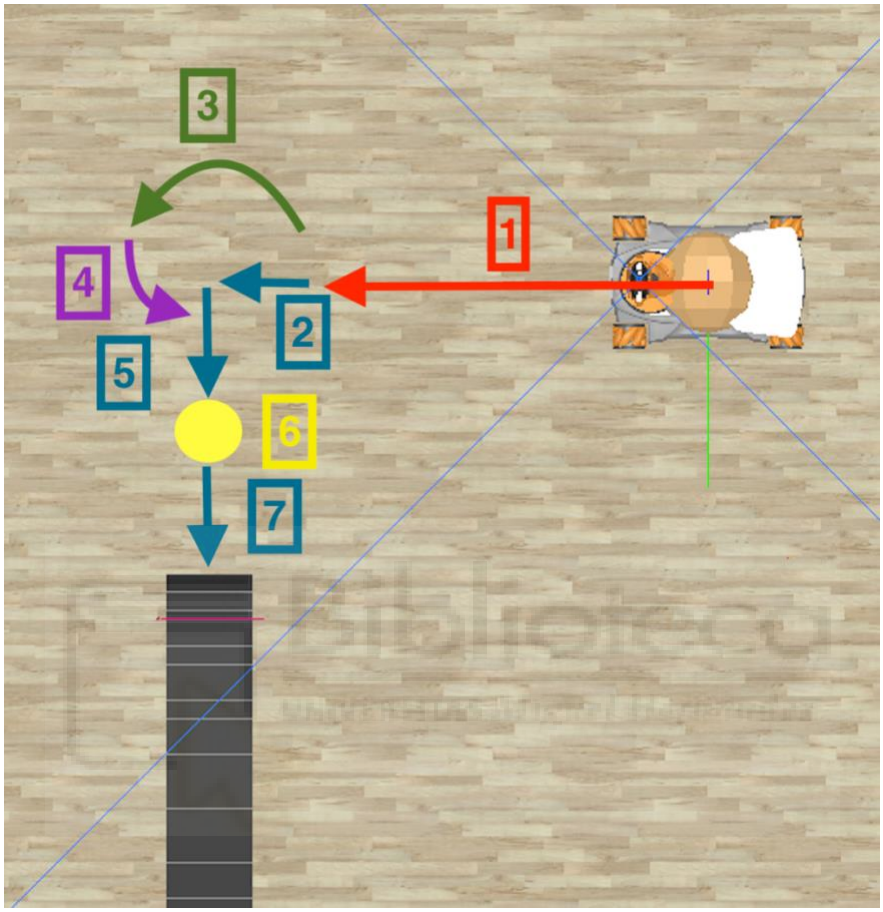


Figura 77. Representación de los movimientos a realizar el robot para tomar la pieza.

Siguiendo la numeración de la Figura 77, se realizan las siguientes acciones:

1. Desplazamiento a lo largo del eje X a alta velocidad hasta aproximarse a la coordenada X de la cinta.
2. Se continúa desplazando con la misma orientación, pero a menor velocidad hasta alcanzar la posición en X deseada con mayor precisión.
3. Se comienza a girar a alta velocidad, hasta un ángulo aproximado al deseado.
4. Al igual que ocurre con el desplazamiento, se reduce la velocidad y nos aproximamos con más precisión al ángulo deseado.
5. Una vez orientado hacia la cinta, el robot comienza a aproximarse a baja velocidad. En este paso el brazo del robot se coloca con las pinzas abiertas enfocando a la cinta.
6. Posición en la que el brazo del robot toma la imagen.
7. Llega a la posición deseada frente a la cinta aproximándose lentamente. Una vez en esta posición, el robot acerca el brazo hasta la pieza situándose entre las dos pinzas y las cerrará, quedando la pieza agarrada.

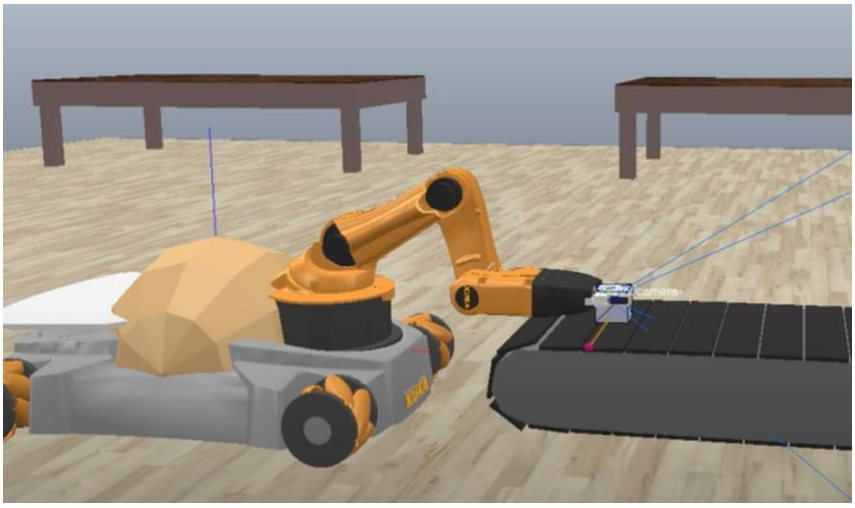


Figura 78. Posición del brazo tomando la pieza.

Una vez que el robot ha cogido la pieza se realizan los mismos pasos en el orden inverso, es decir, se levanta el brazo, se aleja hacia atrás y comienza a realizar el giro y desplazamiento enfocado a dirigirse a una de las mesas. Finalizando esta tarea se realizará el análisis de la imagen para obtener su tamaño y obtendremos un el valor "0" cuando la pieza sea pequeña y "1" cuando la pieza sea grande.

6.3.2. PLACE

Una vez el robot ha realizado la tarea de *Pick* se encontrará realizando el desplazamiento hacia el lado derecho de la escena comienza la siguiente parte de la simulación, la cual consiste en dejar la pieza en el lugar que corresponde. En la primera parte de la simulación se ha realizado la clasificación de la pieza por tamaño, si se ha obtenido un 0 en el apartado anterior, es decir, que la pieza es clasificada como pequeña se dirigirá a la mesa inferior, si por el contrario el resultado es de 1, es una pieza grande y se dirigirá a la mesa superior. Por esta razón se han creado dos definiciones para realizar el proceso de dejar la pieza, ya que los desplazamientos se realizarán a lugares diferentes, aunque ambas posibilidades comenzarán y terminarán en el mismo lugar para retomar de nuevo el proceso de tomar una nueva pieza.

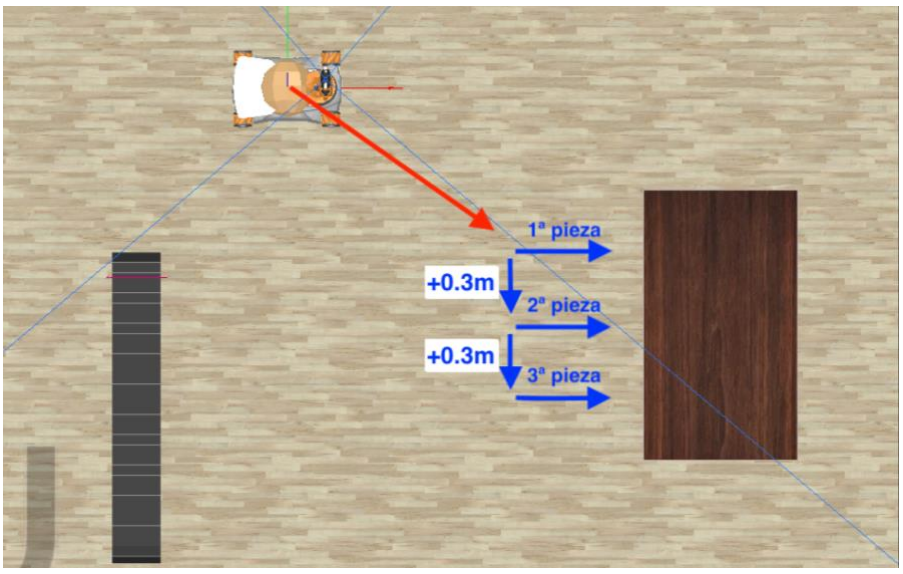


Figura 79. Desplazamientos para dejar las piezas.

En la Figura 79 se muestra el orden en el que se dejarán las piezas en cada mesa, se comenzará por el desplazamiento en diagonal que se dirigirá hacia una de las mesas hasta un punto base cercano, desde ahí, dependiendo del número de pieza que vaya a dejar en esa mesa deberá desplazarse (o no, en el caso de la primera pieza que es directo) 0,8m avanzando en la mesa y dejando la misma distancia entre todas las piezas, por lo que cada vez que se dirija a esa mesa se sumarán de nuevo los 0,8 m. Una vez frente a la mesa se acercará el brazo y soltará pieza abriendo las pinzas.

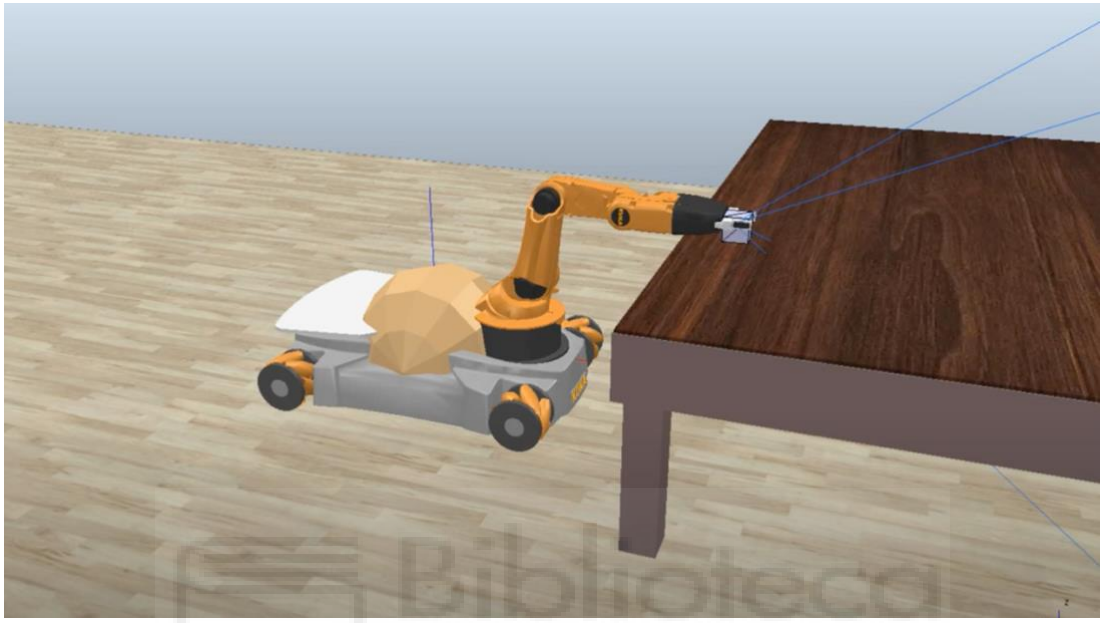


Figura 80. Posición del brazo dejando la pieza en una de las mesas.

Por último, como ocurría en el caso frente a la cinta, se realizarán los pasos de nuevo en el sentido inverso, es decir, se alejará de la mesa, volverá hacia el punto base y realizará un nuevo cambio de orientación para quedar de la forma necesaria y subirá hasta la posición inicial para dirigirse de nuevo a realizar un nuevo pick de otra pieza.

6.4. CLASIFICACIÓN POR TAMAÑO DE PIEZAS CON VISIÓN POR COMPUTADOR.

La última parte en la que se ha dividido este proyecto ha sido la clasificación de piezas por tamaño. El objeto generador de piezas se ha configurado para que genere piezas en orden alternativo de grandes y pequeñas, para las grandes se ha generado un cubo de 4,2cm de lado y para el pequeño de 3,2cm. Como se ha mencionado anteriormente, en el proceso de acercarse a la cinta para tomar la pieza realiza una pequeña pausa en un punto en el que se puede ver por completo la pieza sin que intervenga ningún otro objeto. En este instante se toma una foto que se guarda bajo el nombre de "**imagen.png**".

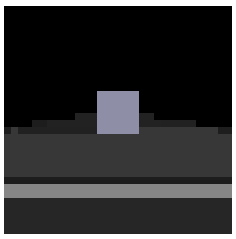


Figura 81. Imagen tomada por la cámara de la pieza en la cinta.

La imagen se toma gracias a la librería *pillow* instalada previamente, ya que su uso es bastante sencillo para tomar y guardar una imagen, pero para el resto de procesamiento se utilizará la librería *openCV*.

Se ha creado una definición llamada ***tamaño_pieza*** (Figura 83) que se encarga de abrir la imagen anteriormente guardada, convertirla a blanco y negro y aplicar un *threshold*, lo que filtra la imagen dejando únicamente el contorno de la pieza en blanco, al ser más claro previamente y haberse filtrado el resto de la imagen pasando a negro, aquí usamos la función que busca todos los contornos de la imagen.

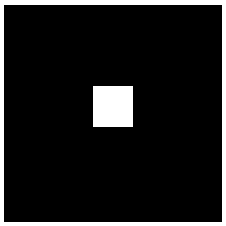


Figura 82. Imagen después de realizar el filtrado para medir el contorno de la pieza.

Aunque en las pruebas realizadas se ha conseguido que únicamente aparezca el contorno deseado, se ordenan los contornos de mayor a menor y nos quedamos con el más grande, por si sucede el caso de que aparezca alguna interferencia no deseada en la imagen.

Para decidir finalmente si la pieza es grande o pequeña, se tendrán en cuenta dos de los cuatros valores que se obtienen al medir un contorno, *w (width)* y *h (height*). En las pruebas realizadas previamente se ha observado que las piezas pequeñas devolvían un valor de 5 píxeles y las grandes de 6, por lo que se filtran por si es mayor o menor a estos valores.

```
def tamaño_pieza():
    path = r'C:\Users\USUARIO\Desktop\pyarte_master\demos\imagen.png'
    path2 = r"C:\Users\USUARIO\Desktop\pyarte_master\demos\gris.png"
    imagen2 = cv2.imread(path)
    imagengris = cv2.cvtColor(imagen2, cv2.COLOR_BGR2GRAY)
    ret, thresh = cv2.threshold(imagengris, 140, 255, cv2.THRESH_BINARY)
    contours, ret1 = cv2.findContours(thresh, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)
    cv2.imwrite(path2, thresh)

    lista_contornos = []
    for c in contours:
        area = cv2.contourArea(c)
        lista_contornos.append(area)

    mas_grande = contours[lista_contornos.index(max(lista_contornos))]
    x, y, w, h = cv2.boundingRect(mas_grande)

    if w <= 5 and h <= 5:
        return 0
    elif w >= 6 and h >= 6:
        return 1
```

Figura 83. Definición tamaño_ pieza().

6.5. RESULTADOS.

Finalmente, al realizar todas las acciones mencionadas en los apartados anteriores en conjunto se obtiene como resultado una simulación en la que podremos controlar que el robot sea capaz de desplazarse a lo largo del plano, realizando una secuencia de tomar y dejar objetos siguiendo una clasificación por tamaño.

Se ha conseguido un entorno con orden entre todos los objetos para optimizar al máximo el flujo de la simulación. Por ejemplo, controlando la generación de piezas con el sensor de proximidad se ha evitado que se crearan piezas hasta que no fuera necesario de nuevo, así se evita que suceda la situación de encontrar varias a la vez en la cinta, ya que el robot está preparado para tomar únicamente una pieza colocada en el centro. Asimismo, se ha tratado de encontrar los desplazamientos más eficientes, para reducir su número al mínimo.

La integración de sistemas de visión por ordenador juega un papel clave en el éxito de la simulación. La capacidad del robot *YouBot* para interpretar y procesar información visual facilita una interacción eficiente con los paquetes y una clasificación precisa. La precisión del robot *YouBot* a la hora de identificar y clasificar paquetes fue satisfactoria. El algoritmo implementado demuestra una capacidad confiable identificar y distinguir diferentes tipos de paquetes con márgenes de error mínimos. Las decisiones tomadas en cuanto a márgenes de error se tomaron tras realizar numerosas pruebas y ajustes.

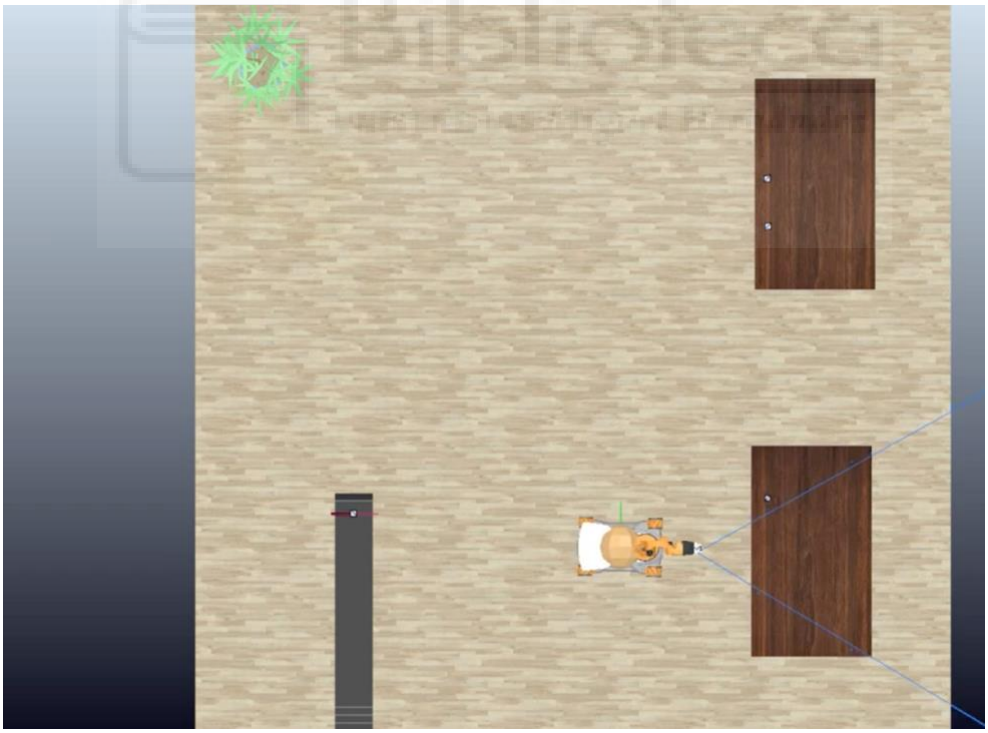


Figura 84. Imagen tomada de la simulación final.

Los robots *YouBot* han demostrado ser una opción rentable y eficaz para aplicaciones de recogida y colocación en entornos de clasificación de paquetes. Además de tener una gran adaptabilidad a diferentes entornos que se puedan crear para la recogida de paquetes, pudiendo combinar sensores para realizar distintas clasificaciones, ya sea por forma, tamaño, color, etc.

En conclusión, los resultados obtenidos respaldan la viabilidad y eficacia de las aplicaciones de recogida y colocación que utilizan robots *YouBot* para la clasificación de paquetes. Estos hallazgos proporcionan una base sólida para futuras investigaciones y desarrollo en los campos de la automatización industrial y la robótica.

7. CONCLUSIONES Y TRABAJOS FUTUROS.

Durante la creación de este Trabajo de Fin de Grado se han encontrado diversas problemáticas y también se han conseguido logros significativos en la robótica aplicada. Previamente se había utilizado el *software* de una forma más general, sin embargo, al comenzar un nuevo proyecto siempre van a surgir dudas y se van a buscar formas alternativas de conseguir los objetivos del trabajo.

En cuanto a crear una simulación han surgido dificultades a la hora de trabajar con *ChildScripts* ya que se emplea el lenguaje *LUA*, que, aunque es un lenguaje sencillo de comprender y se pueden conseguir funcionamientos con menos líneas de código que en otros lenguajes de programación, es necesario entender la forma en la que se realizan las llamadas a otros objetos de la simulación y a las funciones para el control de los mismos, esto ha sucedido a la hora de conectar el sensor de proximidad con el movimiento de la cinta transportadora y el generador de piezas, ya que en un inicio no se podía controlar la creación de las piezas y llegaban a amontonarse dificultando que el robot pudiera coger las piezas adecuadamente.

Otro de los retos de este trabajo ha sido estudiar en profundidad la estructura y las características del robot *YouBot*, conocer su cinemática directa e inversa del brazo robótico y el funcionamiento de la cinemática diferencial para la base, ya que, aunque cada vez es un robot más utilizado en la investigación y la educación, no se encuentra tanta información práctica como se puede encontrar para otros robots más reconocidos y con más trayectoria en el desarrollo de aplicaciones industriales, por esto el manual técnico de *KUKA*, sus creadores, ha sido la principal ayuda en esta parte del trabajo, además de simplificar la estructura del robot para quedarnos únicamente con las articulaciones a las que se les iba a dar uso, reduciendo el problema de 5 articulaciones a 3, del tipo rotacional.

A una aplicación tan típica como es coger piezas de un lugar para transportarlas y depositarlas en otro distinto, se le ha añadido la característica de la visión por computador, demostrando una forma efectiva de poder trabajar con objetos con alguna diferencia y que el proceso de separarlos sea automatizado y no se necesiten separar previa o posteriormente mediante el ser humano u otro robot, agilizando el proceso. He conocido la gran variedad de opciones que se tienen para trabajar con las imágenes gracias a librerías como *Pillow* u *OpenCV*, que siendo gratuitas de código abierto se puede realizar todo tipo de reconocimiento de objetos y persona, medidas tanto en 2D como 3D o contribuir en la independencia de la robótica móvil.

Creo que existen posibles trabajos que se podrían realizar a partir de este escrito, se podrían llegar a conseguir movimientos más precisos reformulando la programación del movimiento, ya que el movimiento actual se realiza mediante aproximaciones. Para esta aplicación se ha buscado que el nivel de exactitud que se ha conseguido sea suficiente para realizar la simulación sin inconvenientes, pero creo que podría llegar a ser mayor si existiera la necesidad de moverse a puntos exactos del espacio por alguna razón.

Uno de los puntos que más modificaciones puede admitir es el depositado de las piezas, en este caso se ha hecho una demostración de 3 piezas por mesa y en un orden lineal, pero podrían colocarse de forma que se apilen unas sobre otras para empaquetados o añadir filas y columnas, pudiéndose modificar también el tipo de objeto que dejamos.

Acerca del uso de sensores creo que es otro aspecto en el que se podría continuar innovando, ya que las posibilidades de realizar combinaciones son infinitas, o, incluso, mantener únicamente el sensor de la cámara, pero se puede buscar conseguir clasificaciones más complejas, por ejemplo, aplicando la detección de distintos objetos que se podrían encontrar en la cadena de producción con la visión por computadora, siendo capaz de reconocerlos como un tipo de objeto directamente, sin centrarse en aspectos más básicos como el color o tamaño. Ya que en una instalación real de una cadena de producción raras veces se van a encontrar cubos sencillos para medir o piezas de únicamente un color, por lo que sería útil tener un registro de las posibilidades de objetos que pueden aparecer y realizar su reconocimiento.

A raíz de la posibilidad de tener mayor variedad de piezas también se podría ampliar el estudio de efectores finales de distintos tipos para seleccionar el más adecuado dependiendo de los objetos con los que se trabaje. Por ejemplo, si tenemos objetos de mayor tamaño necesitaremos buscar una pinza de mayor abertura, o incluso una ventosa que tome la pieza por succión, esto es muy útil si tenemos piezas frágiles.

Por último, otro de los aspectos a mejorar gracias a los sensores podría ser conseguir que el robot sea capaz de realizar una detección de obstáculos. Esto se podría conseguir incluyendo objetos detectables en la simulación a lo largo del plano. Se podría instalar en el robot un sensor láser que provoque que cada vez que se active al detectar un objeto se modifique la trayectoria del robot de forma que evite el obstáculo y se dirija a alcanzar la posición objetivo.

8. ANEXO

8.1. YOUBOT_MOVE.PY

```python
#!/usr/bin/env python
# encoding: utf-8

import numpy as np
import cv2

import sim

from robots import camera
from robots.objects import Dummy
from robots.simulation import Simulation
from robots.youbot import YouBotRobot
from robots.grippers import YouBotGripper
from robots.camera import Camera
from robots.proxsensor import ProxSensor


simulation = Simulation()
clientID = simulation.start()
robot = YouBotRobot(clientID=clientID)
robot.start()
obj = Dummy(clientID=clientID)
obj.start()
gripper = YouBotGripper(clientID=clientID)
gripper.start()
camera = Camera(clientID=clientID)
camera.start()
conveyor_sensor = ProxSensor(clientID=clientID)
conveyor_sensor.start()

def movimiento(obj, posicion_objetivo_x, posicion_objetivo_y):
    posicion_robot = obj.get_position()
    posicion_robotx = posicion_robot[0]
    posicion_roboty = posicion_robot[1]
    while abs(posicion_objetivo_x - posicion_robotx) > 0.075 or abs(posicion_objetivo_y - posicion_roboty) > 0.075:
        posicion_robot = obj.get_position()
        posicion_robotx = posicion_robot[0]
        posicion_roboty = posicion_robot[1]
        robot.wait(1)


def orientacion1(obj, orientacion_objetivo_z):
    orientacion_robot = obj.get_orientation()
    orientacion_robot_z = orientacion_robot[2]
    while (abs(orientacion_objetivo_z - abs(orientacion_robot_z)) > 0.12):
        orientacion_robot = obj.get_orientation()
        orientacion_robot_z = orientacion_robot[2]
        robot.wait(1)


def orientacion2(obj, orientacion_objetivo_z):
    orientacion_robot = obj.get_orientation()
    orientacion_robot_z = orientacion_robot[2]
    while (abs(orientacion_objetivo_z - abs(orientacion_robot_z)) > 0.006):
        orientacion_robot = obj.get_orientation()
        orientacion_robot_z = orientacion_robot[2]
        robot.wait(1)


def aprox(obj, posicion_aproximacion, valor):
    posicion_robot = obj.get_position()
    posicion_roboty = posicion_robot[valor]
    while (abs(posicion_aproximacion - posicion_roboty) > 0.03):
        posicion_robot = obj.get_position()
        posicion_roboty = posicion_robot[valor]
        robot.wait(1)


def tamaño_pieza():
    path = r'C:\Users\USUARIO\Desktop\pyarte_master\demos\imagen.png'
    path2 = r"C:\Users\USUARIO\Desktop\pyarte_master\demos\gris.png"
    imagen2 = cv2.imread(path)
    imagengris = cv2.cvtColor(imagen2, cv2.COLOR_BGR2GRAY)
    ret, thresh = cv2.threshold(imagengris, 140, 255, cv2.THRESH_BINARY)
    contours, ret1 = cv2.findContours(thresh, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)
    cv2.imwrite(path2, thresh)

    lista_contornos = []
    for c in contours:
        area = cv2.contourArea(c)
        lista_contornos.append(area)

    mas_grande = contours[lista_contornos.index(max(lista_contornos))]
    x, y, w, h = cv2.boundingRect(mas_grande)

    if w <= 5 and h <= 5:
        return 0
    elif w >= 6 and h >= 6:
        return 1
```

```python
def cin_inversa(x, y, orientacion):
    L1 = 0.155
    L2 = 0.135
    L3 = 0.181
    posicion = np.array([x, y, 0])
    vector = np.array([np.cos(orientacion), np.sin(orientacion), 0])
    pm = posicion - (L3 * vector)
    R = pow((pow(pm[0], 2) + pow(pm[2], 2)), 0.5)
    alfa = np.arctan2(pm[1], pm[0])
    beta = np.arccos((pow(L2, 2) - pow(L1, 2) - pow(R, 2)) / (-2 * L1 * R))
    gamma = np.arccos((pow(L2, 2) + pow(L1, 2) - pow(R, 2)) / (2 * L1 * L2))
    q2 = gamma - np.pi
    q1 = alfa + beta
    q3 = orientacion - q1 - q2
    return q1, q2, q3

def pick():

    robot.set_base_speed(0, 0, 0)

    robot.set_base_speed(7, 0, 0)
    movimiento(obj, 1.3, 0)

    robot.set_base_speed(2, 0, 0)
    aprox(obj, 1.47, 0)

    robot.set_base_speed(0, 0, 0)
    robot.wait(10)

    robot.set_base_speed(0, 0, -12)
    orientacion1(obj, 1.22)

    robot.set_base_speed(0, 0, -1)
    orientacion2(obj, 1.5708)

    robot.set_base_speed(0, 0, 0)
    q1, q2, q3 = cin_inversa(0.28, 0.09, 0)
    robot.set_arm_position(q1, q2, q3)
    robot.wait(2)
    q1, q2, q3 = cin_inversa(0.28, -0.065, 0)
    robot.set_arm_position(q1, q2, q3)
    gripper.open()
    robot.wait(20)


    robot.set_base_speed(4, 0, 0)
    aprox(obj, 0.3, 1)
    robot.set_base_speed(0, 0, 0)

    camera.save_image('imagen.png')
    size = tamaño_pieza()

    robot.set_base_speed(4, 0, 0)
    aprox(obj, 0.45, 1)

    robot.set_base_speed(0, 0, 0)
    q1, q2, q3 = cin_inversa(0.364, -0.065, 0)
    robot.set_arm_position(q1, q2, q3)
    robot.wait(15)
    gripper.close()

    robot.wait(20)
    q1, q2, q3 = cin_inversa(0.4, 0.3, 0)
    robot.set_arm_position(q1, q2, q3)
    robot.wait(20)

    robot.set_base_speed(-4, 0, 0)
    aprox(obj, 0, 1)

    robot.set_base_speed(0, 0, -12)
    orientacion1(obj, 2.9)

    robot.set_base_speed(0, 0, -1)
    orientacion2(obj, np.pi)

    robot.set_base_speed(6, 0, 0)
    aprox(obj, 0.8, 0)
    robot.set_base_speed(0,0,0)
    robot.wait(10)

    return size

def placesmall(l):

    robot.set_base_speed(6.5, 6, 0)
    movimiento(obj, -0.25, 0.9)

    robot.set_base_speed(0, 0, 0)
    robot.wait(10)
```


Aplicación de pick and place para la clasificación de paquetes con el robot Youbot.

```
    q1, q2, q3 = cin_inversa(0.4, 0.3, 0)
    robot.set_arm_position(q1, q2, q3)

    if l>0.8:
        robot.set_base_speed(0,4,0)
        aprox(obj,l,1)

    robot.set_base_speed(4, 0, 0)
    aprox(obj, -0.75, 0)

    robot.set_base_speed(0, 0, 0)
    robot.wait(20)

    q1, q2, q3 = cin_inversa(0.4, 0.19, 0)
    robot.set_arm_position(q1, q2, q3)
    robot.wait(20)
    gripper.open()
    robot.wait(20)

    q1, q2, q3 = cin_inversa(0.4, 0.3, 0)
    robot.set_arm_position(q1, q2, q3)

    robot.set_base_speed(-4, 0, 0)
    aprox(obj, -0.25, 0)

    if l >0.8:
        robot.set_base_speed(0, -4, 0)
        aprox(obj, 0.8, 1)

    robot.set_base_speed(0, 0, 12)
    orientacion1(obj, 0.45)

    robot.set_base_speed(0, 0, 1)
    orientacion2(obj, 0)

    robot.set_base_speed(3.2, 7, 0)
    movimiento(obj, 0, 0)

    robot.set_base_speed(0, 0, 0)
    robot.wait(10)

def placelarge(l):

    robot.set_base_speed(6.5, -5.6, 0)
    movimiento(obj, -0.25, -0.9)

    robot.set_base_speed(0, 0, 0)
    robot.wait(10)

    q1, q2, q3 = cin_inversa(0.4, 0.3, 0)
    robot.set_arm_position(q1, q2, q3)

    if l < - 0.8:
        robot.set_base_speed(0,-4,0)
        aprox(obj,l,1)

    robot.set_base_speed(4, 0, 0)
    aprox(obj, -0.75, 0)

    robot.set_base_speed(0, 0, 0)
    robot.wait(20)

    q1, q2, q3 = cin_inversa(0.4, 0.19, 0)
    print(q1, q2, q3)
    robot.set_arm_position(q1, q2, q3)
    robot.wait(20)
    gripper.open()
    robot.wait(20)

    q1, q2, q3 = cin_inversa(0.4, 0.3, 0)
    robot.set_arm_position(q1, q2, q3)

    robot.set_base_speed(-4, 0, 0)
    aprox(obj, -0.25, 0)

    if l < - 0.8:
        robot.set_base_speed(0,4,0)
        aprox(obj,-0.8,1)

    robot.set_base_speed(0, 0, 12)
    orientacion1(obj, 0.45)

    robot.set_base_speed(0, 0, 1)
    orientacion2(obj, 0)

    robot.set_base_speed(2.8, -7, 0)
    movimiento(obj, 0, 0)

    robot.set_base_speed(0, 0, 0)
    robot.wait(10)
```


Aplicación de pick and place para la clasificación de paquetes con el robot Youbot.

```python
def pick_and_place():
    i=0
    ls=0.9
    ll=-0.9

    while i < 6:

        piece_size = pick()

        if piece_size == 0:
            placesmall(ls)
            ls = ls + 0.3

        elif piece_size == 1:
            placelarge(ll)
            ll = ll - 0.3

        i = i + 1

if __name__ == "__main__":
   pick_and_place()
   simulation.stop()
```

8.2. *CHILDSCRIPT* CINTA.

Customization script "/conveyor"

```
conveyor=require('conveyor_customization')

function sysCall_init()
    config={}

    -- Modify following to customize your conveyor:
    ------------------------------------------------
    config.padSize={0.05,0.25,0.005}
    config.interPadSpace=0.002
    config.radius=0.05
    config.length=1.5
    config.padCol={0.2,0.2,0.2}
    config.col={0.5,0.5,0.5}
    config.respondablePads=true
    config.initPos=0
    config.initVel=0.1
    ------------------------------------------------

    sensor=sim.getObjectHandle('conveyor_sensor')

    conveyor.init(config)
end

function sysCall_actuation()
    beltVelocity=0.5

    if sim.readProximitySensor(sensor)>0 then
        beltVelocity=0
    end

    sim.writeCustomDataBlock(conveyor.model,'CONVMOV',sim.packTable({vel=beltVelocity}))

    _S.conveyor.actuation()
end
```

8.3. *CHILDSCRIPT* CUBESPAWNER.

Child script "/CubeSpawner"

```
function sysCall_init()
    corout=coroutine.create(coroutineMain)
    sensor=sim.getObjectHandle('/conveyor_sensor')
end

function sysCall_actuation()
    if coroutine.status(corout)~='dead' then
        local ok,errorMsg=coroutine.resume(corout)
        if errorMsg then
            error(debug.traceback(corout,errorMsg),2)
        end
    end
end

function coroutineMain()

local CUBE_COUNT = 100
local START_POS = {1.45, 2.25, 0.2}
local WAIT_TIME = 6
local PIECE_TYPE = 0 -- 0: cuboid, 1: sphere, 2: cylinder, 3: cone
local i = 0

while i < CUBE_COUNT do
    while sim.readProximitySensor(sensor) < 1 do

      if (i%2==0) then
        local spawn = sim.createPureShape(PIECE_TYPE, 10, {0.042,0.042,0.042}, 0.1, NULL)
        sim.setObjectPosition(spawn, -1, START_POS)
        sim.setObjectSpecialProperty(spawn, sim.objectspecialproperty_detectable_all)
       else
        local spawn = sim.createPureShape(PIECE_TYPE, 10, {0.032,0.032,0.032}, 0.1, NULL)
        sim.setObjectPosition(spawn, -1, START_POS)
        sim.setObjectSpecialProperty(spawn, sim.objectspecialproperty_detectable_all)
       end
      sim.wait(WAIT_TIME)
      i = i + 1
    end

end
```

8.4. *CHILDSCRIPT* YOUBOT.

Child script "/youBot"

```
function sysCall_init()
    corout=coroutine.create(coroutineMain)
end

function sysCall_actuation()
    if coroutine.status(corout)~='dead' then
        local ok,errorMsg=coroutine.resume(corout)
        if errorMsg then
            error(debug.traceback(corout,errorMsg),2)
        end
    end
end

function setMovement(forwBackVel,leftRightVel,rotVel)
    -- Apply the desired wheel velocities:
    sim.setJointTargetVelocity(wheelJoints[1],-forwBackVel-leftRightVel-rotVel)
    sim.setJointTargetVelocity(wheelJoints[2],-forwBackVel+leftRightVel-rotVel)
    sim.setJointTargetVelocity(wheelJoints[3],-forwBackVel-leftRightVel+rotVel)
    sim.setJointTargetVelocity(wheelJoints[4],-forwBackVel+leftRightVel+rotVel)
end

function coroutineMain()
    --Prepare initial values and retrieve handles:
    wheelJoints={-1,-1,-1,-1} -- front left, rear left, rear right, front right
    wheelJoints[1]=sim.getObjectHandle('rollingJoint_fl')
    wheelJoints[2]=sim.getObjectHandle('rollingJoint_rl')
    wheelJoints[3]=sim.getObjectHandle('rollingJoint_rr')
    wheelJoints[4]=sim.getObjectHandle('rollingJoint_fr')
    armJoints={}
    for i=0,4,1 do
        armJoints[i+1]=sim.getObjectHandle('youBotArmJoint'..i)
    end


end

function sysCall_cleanup()
end
```

Aplicación de pick and place para la clasificación de paquetes con el robot Youbot.

9. REFERENCIAS.

[1] Ignacio G.R. Gavilán. (2019, October 14). Las cinco generaciones de robots según Michael Knasel | Ignacio G.R. Gavilán. Retrieved February 1, 2024, from Ignacio G.R. Gavilán Página web: https://ignaciogavilan.com/las-cinco-generaciones-de-robots-segun-michael-knasel/

[2] Industria robótica - Tamaño del mercado y crecimiento. (2023). Retrieved February 1, 2024, from Mordorintelligence.com Página web: https://www.mordorintelligence.com/es/industry-reports/robotics-market

[3] Kryscia, R., & Benavides. (n.d.). CI-2657 Robótica. Obtenido de https://www.kramirez.net/Robotica/Material/Presentaciones/CinematicaDirectaRobot.pdf

[4] Universidad Continental - Modalidad a Distancia. (2021). Resolución del problema cinemático inverso a partir de matrices de transformación homogénea [YouTube Video]. Obtenido de https://www.youtube.com/watch?v=T7963P4dWtw

[5] Escuela politécnica nacional facultad de ingeniería mecánica modelación y análisis de la cinemática directa e inversa del manipulador Stanford de seis grados de libertad proyecto previo a la obtención del título de ingeniero mecánico María Victoria Granja Oramas. (n.d.). Obtenido de https://bibdigital.epn.edu.ec/bitstream/15000/8693/1/cd-5831.pdf

[6] Bugarin, E., Aguilar-Bustos, A., & Borrego-Ramirez, O. A. (2013). Validación Experimental del Modelo Cinemático de un Robot Móvil Omnidireccional de 4 Ruedas. In Congreso Internacional de Robótica y Computación (pp. 24-29). Obtenido de https://www.researchgate.net/profile/Luis-Sanchez-Medel/publication/305617563_Integracion_de_reconocimiento_de_voz_e_imagen_en_robot_humanoide/

[7] Barahona Guamani, E. S. (2019). *Navegación autónoma basada en maniobras bajo estimación de posturas humanas para un robot omnidireccional kuka youbot* (Bachelor's thesis, Universidad Técnica de Ambato. Facultad de Ingeniería en Sistemas, Electrónica e Industrial. Carrera de Ingeniería Electrónica y Comunicaciones). Obtenido de https://repositorio.uta.edu.ec/bitstream/123456789/30119/1/Tesis_t1642ec.pdf

[8] Álvarez Tobar, S. J. (2019). *Navegación autónoma de un robot móvil omnidireccional en trayectorias predeterminadas con evitación de obstáculos* (Master's thesis, Universidad Técnica de Ambato. Facultad de Ingeniería en Sistemas, Electrónica e Industrial. Maestría en Automatización y Sistemas de Control). Obtenido de https://repositorio.uta.edu.ec/bitstream/123456789/29173/1/Tesis_%20t1526masc.pdf

[9] Rivero González, F. J. (2021). Simulación del robot KUKA YouBot en el entorno de CoppeliaSim. Obtenido de: https://hdl.handle.net/11441/127069

[10] CoppeliaSim User Manual. (2024). Retrieved February 1, 2024, from Coppeliarobotics.com Página web: https://manual.coppeliarobotics.com

[11] Cárdenas Millán, I. (2021). Programación de robot móvil con prevención de colisiones CoppeliaSim. Obtenido de: https://repositorio.uniandes.edu.co/flip/?pdf=https://repositorio.uniandes.edu.co/server/api/core/bitstreams/1cfea2e5-2449-42e3-8cbb-bc2a1f84b588/content

[12] Oswal, S., & Saravanakumar, D. (2021). Line following robots on factory floors: Significance and Simulation study using CoppeliaSim. In *IOP Conference Series: Materials Science and Engineering* (Vol. 1012, No. 1, p. 012008). IOP Publishing. Obtenido de https://iopscience.iop.org/article/10.1088/1757-899X/1012/1/012008/pdf

[13] 4rtur1t0. (2021). GitHub - 4rtur1t0/pyARTE: Robot simulations using Coppelia. Retrieved February 1, 2024, from GitHub website: https://github.com/4rtur1t0/pyARTE