UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dpto. de Ingeniería de Sistemas y Automática

CONTROL Y COORDINACIÓN DE UR3 PARA ROBOT DE SERVICIO EN ACTIVIDADES LOGÍSTICAS

TRABAJO FIN DE MÁSTER PRESENTADO PARA OPTAR AL TITULO DE

MÁSTER UNIVERSITARIO EN AUTOMÁTICA E INFORMÁTICA INDUSTRIAL

POR

KEVIN BAZAGA ÁLVAREZ

VALENCIA, OCTUBRE DE 2023

TRABAJO FIN DE MÁSTER

Autor:

KEVIN BAZAGA ÁLVAREZ

Director:

JUAN FRANCISO BLANES NOGUERA

ÍNDICE

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

1.1 Planteamiento

1.1.1 Situación y contexto

En los últimos años, hemos sido testigos de un crecimiento exponencial en la utilización de robots en la industria. Específicamente, en la última década, hemos presenciado un aumento notable en la adopción de cobots o robots colaborativos en diversos sectores industriales. Este fenómeno se atribuye principalmente a una serie de razones, entre las cuales destacan su versatilidad, facilidad de programación e instalación eficiente [1].

Un punto relevante a considerar es que los robots colaborativos se caracterizan por su capacidad de operar sin la necesidad de barreras de seguridad u otras estructuras auxiliares, a diferencia de los robots convencionales. Esta particularidad ha impulsado su integración en líneas de producción, donde asumen tareas que anteriormente eran desempeñadas por máquinas distintas. A medida que evoluciona el panorama industrial, se ha propuesto además la incorporación de robots colaborativos en plataformas móviles. Esta innovación permite que un único robot manipulador realice labores que antes requerían la colaboración de varios robots tradicionales [2]. Este enfoque no solo conlleva beneficios económicos al optimizar la inversión, sino que también incrementa la productividad y eficiencia de la producción, sin comprometer la disposición de la fábrica al prescindir de barreras de seguridad adicionales.

Con esta premisa como base, el presente trabajo surge con el propósito fundamental de maximizar la utilidad de dos tipos de robots: uno de naturaleza móvil y otro perteneciente a la categoría de cobots, lo que refleja un enfoque innovador y pragmático ante los desafíos actuales de la industria.

1.1.2 Resumen

El problema que se aborda en este trabajo consiste en lograr una coordinación efectiva entre varios componentes, con un enfoque particular en la colaboración entre una base móvil y un brazo robot. Este objetivo se persigue con el propósito de llevar a cabo tareas logísticas, en particular aquellas que involucran la manipulación de objetos y la interacción con seres humanos.

Para llevar a cabo este proyecto, la base móvil que se utilizará es del fabricante Robotnik [3], específicamente el modelo RB-1 BASE, mientras que el brazo robot utilizado es del fabricante Universal Robots (UR)[4], concretamente el modelo UR3e, que se encuentra montado en la parte superior de la base como se ve en la figura 1.

Figura 1. Brazo robot acoplado en robot móvil

Adicionalmente, se incorpora una cámara en el extremo del brazo del robot, la cual desempeña un papel crucial en la detección y seguimiento de objetos, y se emplea una garra como herramienta efectora. En la figura 2, se puede apreciar con mayor detalle la cámara y la garra incorporadas en el extremo del brazo robot.

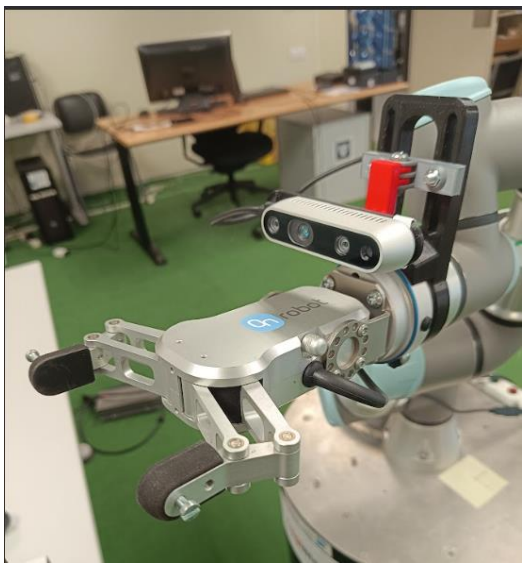

Figura 2. Herramienta efectora y cámara

La idea del trabajo consiste en desarrollar un sistema que pueda llevar a cabo operaciones logísticas de manera eficiente, haciendo uso de la visión proporcionada por la cámara, la cual estará conectada a una Raspberry Pi. Específicamente, se ha seleccionado abordar la tarea de detección, seguimiento y manipulación de objetos. El objeto utilizado para llevar a cabo las pruebas en este trabajo se muestra en la figura 3. Concretamente se trata de un cubo impreso en una impresora 3D de seis centímetros de lado con un código AruCo en una de sus caras, aunque se podría tratar de cualquier objeto identificado a través de otro método.

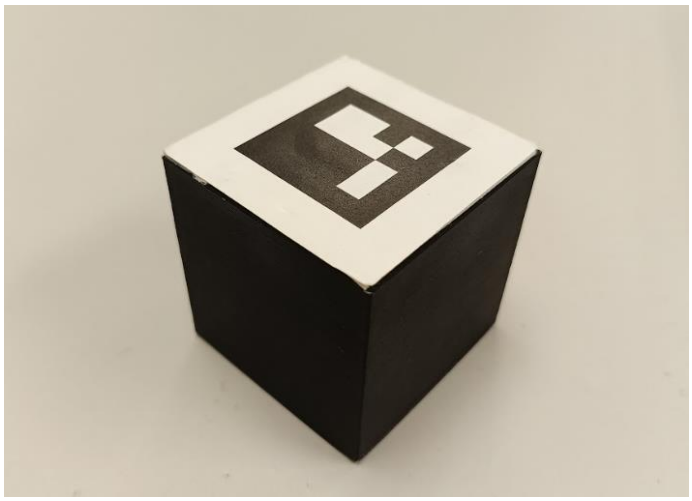

Figura 3. Objeto utilizado en el trabajo

Para lograr la tarea mencionada, se busca establecer un sistema interconectado en el cual todos los elementos se comuniquen de manera fluida y coordinada. Un aspecto fundamental de esta estrategia implica que sea la plataforma móvil la encargada de enviar comandos de movimiento al brazo del robot. Además, en situaciones donde la tarea de alcanzar el objeto pueda verse obstaculizada por singularidades del brazo o la distancia al objeto, se propone una estrategia adicional: permitir que la base móvil realice un acercamiento inicial, con el propósito de facilitar la posterior manipulación.

1.2 Objetivos

El problema expuesto en el apartado anterior, nos lleva a plantearnos una serie de objetivos que orientarán el desarrollo y la ejecución de este proyecto. A continuación, se presentan los logros concretos que se busca obtener.

1.2.1 Objetivo principal

El objetivo primordial de este trabajo es desarrollar un sistema robótico integrado y colaborativo capaz de realizar acciones de logística mediante la detección, seguimiento y manipulación eficiente de objetos, empleando una plataforma móvil y un brazo robot.

1.2.2 Objetivos específicos

Con el propósito de alcanzar el objetivo principal expuesto en la sección anterior, se desglosará en una serie de objetivos específicos interrelacionados. Estos objetivos individuales, al ser abordados en conjunto, se espera que contribuyan de manera integral a la consecución del objetivo principal.

- Establecer una comunicación fluida y de baja latencia entre el hardware involucrado, implementando *actions* y servidores de Robot Operating System (ROS) para facilitar la interconexión de los sistemas. Esto permitirá una comunicación bidireccional eficiente entre los diferentes componentes, asegurando la coordinación necesaria para las tareas logísticas y optimizando la comunicación.

- Obtener información confiable sobre objetos mediante tecnología de visión 3D, lo que es esencial para su manipulación precisa y la coordinación de movimientos entre la base móvil y el brazo robot.

- Lograr mediciones precisas de las distancias a los objetos de interés. Esto resulta fundamental para asegurar un manejo seguro y efectivo de los objetos en las operaciones logísticas.

- Obtener los vectores de distancia a los objetos en formatos compatibles para el manejo del brazo robot a partir de la información capturada mediante visión 3D. Esto es crucial para implementar una estrategia de orientación dinámica, ya que permitirá al brazo robot reorientarse en tiempo real en situaciones donde su orientación inicial no sea adecuada, mejorando así la precisión y eficiencia en la manipulación de objetos.

- Lograr un sistema de seguimiento continuo de objetos por parte del brazo robot, garantizando su capacidad de rastrear y seguir un objetivo en movimiento hasta que este se detenga. Esta capacidad de seguimiento constante es fundamental para lograr un agarre exitoso y preciso de objetos en movimiento.

- Incorporar la capacidad de evaluar si la distancia al objeto permite su agarre por parte del brazo robot sin requerir desplazamientos de la base móvil. En los casos en que la distancia sea demasiado grande, se aspira a implementar una funcionalidad que permita acercar la base móvil hasta que el brazo robot pueda acceder al objeto de manera eficiente y sin obstáculos.

1.3 Relación con la titulación

Para poder llevar a cabo este trabajo correctamente, resulta fundamental contar con los conocimientos adquiridos a lo largo de diversas asignaturas del Máster en Automática e Informática Industrial. A continuación, se enumerarán aquellas materias que guardan una mayor afinidad con el trabajo realizado, y se explicará el ámbito del trabajo con el que se relaciona. Es importante resaltar que se han aplicado conocimientos de múltiples asignaturas del Máster, pero en esta lista se incluirán las más relevantes y relacionadas.

- **Redes y sistemas distribuidos para el control:** La asignatura Redes y Sistemas Distribuidos para el Control, del segundo cuatrimestre, proporciona los conocimientos necesarios para comprender algunos de los protocolos de comunicaciones fundamentales. Entre estos protocolos se encuentra la conexión TCP/IP (Transmission Control Protocol/Internet Protocol), además de introducir el concepto de socket.

  Para llevar a cabo la comunicación en tiempo real con el brazo robot, se empleará el protocolo RTDE (Real-Time Data Exchange). Este protocolo se basa en una conexión TCP/IP para establecer la comunicación entre un robot industrial y una aplicación externa. Dicha comunicación se implementa a través de un socket TCP, lo que permite la transmisión bidireccional de datos en tiempo real entre el controlador del robot y la aplicación[5]. En este sentido, la asignatura ha sido determinante para reforzar los conocimientos sobre los protocolos de comunicación esenciales requeridos para llevar a cabo el proyecto de manera exitosa.

- **Visión por computador en la industria:** La asignatura Visión por Computador en la Industria, del segundo cuatrimestre, tiene como objetivo dar a conocer a los alumnos la problemática general de la visión por computador (VxC), como herramienta multidisciplinar para la solución de problemas de automatización de procesos de inspección en entornos industriales y de visión para robótica.

  En este proyecto, es muy necesario contar con los conocimientos adquiridos en el contexto de esta asignatura, dado que se empleará una cámara para identificar los objetivos. Los trabajos finales de esta materia desempeñan un papel crucial al consolidar y perfeccionar la comprensión de la librería OpenCV de Python. Esta herramienta resulta sumamente relevante para la realización del trabajo, ya que posibilita la detección de patrones incorporados en los objetivos, así como la estimación de las distancias a los mismos.

- **Seminarios:** En la asignatura Seminarios, del primer cuatrimestre, se tratan diversos aspectos avanzados o de máxima actualidad en el ámbito del Máster con el fin de complementar la formación del alumnado. En este proyecto, los seminarios que han tenido un papel más significativo son “Programación avanzada en Python” y “Robótica colaborativa e IA (Boston Scientific y National Univ. Ireland)”.

  - Programación avanzada en Python: Este seminario impartió conocimientos sobre el manejo del lenguaje de programación Python, abordando desde los fundamentos esenciales para aquellos que no lo habían utilizado previamente, hasta aspectos más avanzados. Asimismo, suministró documentación y ejercicios útiles para reforzar y poner en práctica las habilidades adquiridas en Python. En el proyecto, todas las implementaciones de programas se han realizado en Python, lo que resalta la gran utilidad de este seminario en el avance del estudio.

  - Robótica colaborativa e IA: Este seminario ofrecía una breve introducción sobre la naturaleza de los cobots, su evolución y las ventajas que presentan en comparación con los brazos robóticos tradicionales. También se exploraron diversas variantes de efectores finales y las múltiples aplicaciones que pueden abordar estos robots mediante la presentación de videos ilustrativos de ejemplos concretos. En el desarrollo del proyecto, se ha empleado un robot colaborativo o cobot para ejecutar las tareas de agarre de objetos, lo que demuestra la importancia de esta introducción teórica en el contexto práctico del trabajo.

- **Interfaces físicos y sistemas empotrados:** En la asignatura Interfaces Físicos y Sistemas Empotrados, del segundo cuatrimestre, se establecen los conocimientos necesarios para que el estudiante adquiera habilidades para el desarrollo de aplicaciones informáticas que contengan componentes estrechamente relacionados con el entorno físico en el que desempeñan su función. Adicionalmente, se abordan los conceptos de sistemas empotrados y se proporciona formación sobre su manipulación. Una relación particularmente relevante con el trabajo realizado es el uso del sistema operativo Linux a lo largo de la asignatura. Del mismo modo, se lleva a cabo la aplicación práctica de protocolos de seguridad SSH (Secure Shell) y SCP (Secure Copy Protocol) en diversas instancias del curso. Estos protocolos permiten la comunicación segura y el intercambio de datos entre sistemas.

En el proyecto, se emplea un sistema empotrado (Raspberry Pi) con sistema operativo Linux. Además, para lograr una comunicación segura y efectiva con este sistema empotrado, se emplean los protocolos de seguridad SSH y SCP. Por esta razón, los conocimientos adquiridos en esta asignatura adquieren un rol esencial en la ejecución exitosa del trabajo.

1.4 Estructura del proyecto

Para llevar a cabo la explicación del trabajo, se ha estructurado la memoria en seis capítulos distintos. En este primer capítulo, titulado "Introducción", se proporciona un breve resumen de la esencia del trabajo, se establece la justificación de su ejecución y se presentan de manera explícita los objetivos y subobjetivos que se pretenden alcanzar. Asimismo, se explora la conexión entre el trabajo y algunas de las asignaturas del máster cursado, con el fin de fundamentar sólidamente la realización de esta investigación.

En el segundo capítulo, se realiza una exposición detallada del estado del arte del trabajo desarrollado. En este sentido, se aborda en primer lugar la relevancia de la robótica en la actualidad, para posteriormente enfocarse en los robots colaborativos y móviles. Luego, se lleva a cabo una revisión de la literatura cercana a este trabajo, acentuando de manera particular la importancia de tres artículos fundamentales en la realización del proyecto.

Posteriormente, en el tercer capítulo, se procede a la revisión y detallada explicación de los materiales y métodos empleados en el desarrollo del proyecto. Así, se expondrán en detalle los distintos componentes tanto a nivel hardware como software que han sido necesarios para la ejecución exitosa de este trabajo.

El cuarto capítulo, se adentra en el desarrollo del proyecto, destacando las diversas etapas que se llevaron a cabo para alcanzar los subobjetivos previamente definidos en el proyecto. Se hará hincapié en el desarrollo de software, así como en la comunicación efectiva entre los componentes de hardware utilizados. Este capítulo proporciona una visión detallada de la metodología implementada para alcanzar los resultados deseados.

El quinto capítulo se centra en mostrar imágenes del brazo robot en las diferentes etapas de implementación. Para cada configuración, se brindará una explicación sobre la posición y el estado específico en el que se encuentra el brazo robot. Esto permitirá visualizar de manera concreta la evolución y del sistema a lo largo del proyecto.

En el sexto y último capítulo, se presentan las conclusiones obtenidas a partir de este trabajo, resaltando cómo se han alcanzado los objetivos establecidos. Además, se abrirá la puerta a futuras investigaciones y desarrollos que podrían surgir a partir de los resultados de este proyecto.

2 ESTADO DEL ARTE

2.1 Robótica en la actualidad

Según la norma ISO 8373:2021, la robótica es la ciencia y práctica del diseño, fabricación y aplicación de robots [6]. Es una tecnología que cobra gran importancia a día de hoy y que se espera que siga creciendo en los próximos años.

La robótica tiene el potencial necesario para mejorar nuestras vidas, cambiando aspectos del trabajo y del hogar. Esta mejora se puede ver reflejada en los niveles de eficiencia de los trabajos y, sobre todo, en la seguridad, evitando las tareas peligrosas para las personas [7]. Según el uso de los robots, existen dos principales tipos: industrial y de servicios.

Los robots industriales son los utilizados en las fábricas de automoción y manufactura en aplicaciones de automatización industrial. Además, actualmente, la robótica es un aspecto clave en la competitividad de las industrias manufactureras a gran escala (Figura 4).

Figura 4. Robots industriales

Los robots de servicios son aquellos que están diseñados para llevar a cabo un servicio específico para las personas (Figura 5). Son sistemas ideados para realizar tareas consideradas peligrosas o actividades repetitivas, como pueden ser las tareas domésticas, proporcionando eficiencia, seguridad y productividad [8]. A diferencia de los robots industriales, las tareas de los robots de servicios excluyen las aplicaciones de automatización industrial.

Figura 5. Robot de servicio

Los primeros robots que se utilizaron en la industria, eran robots manipuladores, que realizaban pocas tareas lógicas y movimientos repetitivos. A principios de los años 60, Unimation desarrolló el primer robot programable y, en los años 70, aparecieron por primera vez brazos mecánicos más avanzados. A partir de la década de los 80 se produjo una expansión de la robótica aumentando hasta en un 80% su fabricación y venta [9].

En la actualidad, la industria se ha beneficiado de esta expansión de la robótica, encargada de los trabajos más peligrosos o repetitivos, provocando una mejoría de la productividad [10]. Desde la agricultura hasta la medicina, pasando por todo tipo de fábricas, actualmente es indispensable la utilización de la robótica en gran parte de nuestros trabajos.

2.2 Robots colaborativos y robots móviles

La colaboración entre humanos y robots se ha convertido en un factor clave para la sostenibilidad de la fabricación en Europa. Los bajos costos de instalación y operación, así como un alto grado de flexibilidad, son las principales características que hacen que los cobots (Figura 6) sean atractivos para muchas empresas que buscan establecer instalaciones de producción deslocalizadas con un rápido retorno de la inversión [11].

Figura 6. Cobot

Además, el envejecimiento de la mano de obra plantea un desafío significativo para la salud y seguridad en todos los países europeos. En este contexto, los cobots pueden ser un apoyo y una extensión de las capacidades humanas, liberando a los operarios de tareas repetitivas o peligrosas y permitiéndoles asumir tareas más complejas, donde la experiencia puede tener un papel más importante y una mayor eficacia. A esto se suma que los cobots pueden operar en un entorno sin vallas de protección siguiendo uno de los cuatro modos de trabajo especificados en la ISO 10218-2:2011 [12], lo que no afecta al diseño ni a las instalaciones de la fábrica.

A pesar de que la robótica industrial ha sido ampliamente utilizada en las últimas décadas, la robótica colaborativa, pese a ser una tecnología disponible desde la década de los 2000, ha comenzado a ganar relevancia en la escena industrial recientemente. Dada su versatilidad y la característica de no requerir barreras de seguridad a su alrededor, los investigadores pronto reconocieron la viabilidad de instalarlos en robots móviles para permitir desplazarlos y cumplir diversas tareas (Figura 7). De esta manera, un solo robot móvil puede reemplazar a varios robots fijos que solo estarían en funcionamiento durante períodos cortos, lo que reduce los costos fijos de una línea de producción.

Figura 7. Cobot instalado en un robot móvil

Entre los primeros ejemplos de robots móviles y manipuladores robóticos integrados para tareas industriales, se diseñaron y desarrollaron sistemas robóticos capaces de navegar y operar en entornos específicos, como una tienda de comestibles [13] y llevar a cabo tareas complejas, como recoger cubos de basura [14]. El proyecto Robo-Partner de la UE [15] se centró en combinar las capacidades cognitivas de los humanos con la fuerza, velocidad, repetibilidad y precisión de los robots en el ensamblaje del eje trasero de un vehículo, desarrollando interfaces humano-robot intuitivas mediante sensores, servomotores visuales, reconocimiento de voz y algoritmos de control avanzados. También se ha abordado la automatización de los procedimientos de preparación de pedidos, investigando la recogida y paletización autónoma mediante una plataforma móvil específica equipada con un cobot industrial [16]. El proyecto Valeri tuvo como objetivo probar la robótica móvil en la industria aeroespacial. Se utilizó la plataforma OmniRob de Kuka y se complementó con un eje lineal vertical giratorio sobre el que se montó un manipulador ligero. Se ha empleado un sistema robótico equipado con múltiples sensores perceptivos para llevar a cabo tareas de manipulación en una planta de producción de bombas de agua [17]. En [18], los autores presentan un sistema robótico móvil diseñado para operar en los suelos móviles de las líneas de montaje final de automóviles. Finalmente, en 2019, se ofrece un repaso exhaustivo de las arquitecturas y aplicaciones de sistemas en el ámbito de los manipuladores industriales móviles colaborativos [19], mientras que en 2021 se analizaron los principales desafíos y métodos relacionados con la garantía de seguridad al tratar con cobots a través de una lista de ejemplos relevantes y casos prácticos de éxito [20].

2.3 Revisión de la Literatura

En esta sección, se presenta una revisión de la literatura relevante que respalda y contextualiza el objetivo final de este estudio. Fundamentalmente, se examinarán tres artículos. Los dos primeros ejemplifican dos potenciales propósitos adoptados por robots similares a los utilizados en este trabajo, mientras que el tercer artículo proporciona una posible guía para la metodología y los pasos requeridos para su puesta en marcha.

2.3.1 Ejemplificación del objetivo final.

Con el fin de evaluar las perspectivas futuras y la utilidad potencial que este trabajo podría alcanzar, se presentan dos artículos que ejemplifican la aplicación propuesta en este estudio, centrado en el uso de la robótica en acciones de logística.

2.3.1.1 Descarga y composición automatizada de palets

El primer artículo seleccionado [21] presenta un novedoso manipulador móvil equipado con un sistema de percepción 3D que se enfoca en la automatización de la manipulación de palets en un entorno logístico. Su objetivo principal es descargar automáticamente un palet con cajas homogéneas (palet de origen) y componer otro con cajas, posiblemente diferentes (palet mixto). Esta tarea implica la recogida de diferentes palets de origen y su posterior manipulación.

En el artículo se explica la importancia de planificar y optimizar eficazmente la secuencia de acciones en la implementación de un sistema robótico capaz de asistir a los trabajadores en sus tares. Esta planificación incluye localizar correctamente las personas y objetos e interactuar con el entorno y con otros sistemas de la red. En el artículo se acentúa el fomento del uso de vehículos guiados dentro de la industria con la llegada de la industria 4.0, así como la creciente utilización de colaboraciones entre Cobots y plataformas móviles en aplicaciones logísticas.

Para lograr este objetivo, se utiliza la colaboración entre una plataforma móvil (MiR 100 AMR) y un brazo robot “Cobot” (colaborative Robot) UR10e. Además, para detectar los bordes de los paquetes, se utiliza una cámara en el extremo del brazo del robot. La cámara utilizada es IFM Electronics O3D303.

2.3.1.2 Carga autónoma de materia prima en envasadoras automáticas

El segundo artículo analizado[22], presenta un sistema robótico capaz de llevar a cabo ciclos autónomos de carga de materia prima en una máquina automática de envasado de té, en un entorno compartido con operarios humanos. Con pesos de hasta 12 kg, la carga repetida de materia prima en las máquinas puede convertirse a largo plazo en una fuente de angustia física para un operario y, además, el operario debe interrumpir su tarea habitual para realizar esta carga, lo que puede provocar pérdidas de concentración. Por otra parte, el número de veces que debe realizarse esta operación a lo largo de cada turno no es suficiente para justificar la inversión de varios robots despaletizadores estándar, lo que lleva a la elección de esta solución.

El artículo justifica la creciente adopción de robots colaborativos, así como los beneficios de su uso en conjunto con robots móviles. Asimismo, destaca la importancia de la colaboración entre robots y operarios humanos en la industria, explicando cómo los robots colaborativos pueden ser la opción más adecuada. Además, se presentan varios ejemplos destacados en los que se han empleado tanto robots colaborativos como robots móviles.

Para lograr el objetivo mencionado, se implementó un robot móvil preintegrado compuesto por un Vehículo de Guiado Automático (AGV) y un robot colaborativo redundante en serie (cobot), equipado con diversos sensores y dispositivos adicionales adaptados al propósito del proyecto. También se emplearon estrategias de visión por computador apropiadas para generar trayectorias confiables y sólidas para la manipulación de las materias primas.

2.3.2 Guía de implementación

Una vez examinadas las posibles funcionalidades alcanzables mediante la conclusión de este proyecto, se introduce un tercer artículo que ofrece una posible guía para orientar el avance y seguir los pasos necesarios en el trabajo, conduciendo hacia la consecución del objetivo final deseado.

El articulo [23] aborda el problema del rastreo y agarre de un objetivo dinámico en un entorno real. Se presenta un sistema robotizado compuesto por un robot de cuatro ruedas con un brazo instalado que incluye una pinza, diseñado para realizar tareas de manipulación móvil. La finalidad principal del robot debe ser alinear el objetivo dinámico con la pinza integrada en el brazo robot evitando restricciones del robot manipulador, lo que implica la utilización de técnicas de control visual.

El enfoque central del artículo se basa en el concepto de control visual, también conocido como “visual servoing”, el cual aprovecha la información capturada por un sensor visual para guiar el movimiento del robot. El sistema está equipado con un sistema de percepción a bordo y algoritmos que permiten su autonomía. El control visual resulta esencial para lograr la tarea de agarrar objetos en movimiento, asegurando una alineación precisa entre la pinza y el objetivo.

Una característica distintiva del sistema es la colaboración y comunicación entre la base y el brazo robot, lo que garantiza un seguimiento efectivo del objetivo incluso cuando está fuera del rango de alcance. El artículo detalla cómo se calcula una combinación de fuerzas y momentos de atracción virtuales para lograr un agarre dinámico exitoso. La magnitud de la fuerza virtual se ajusta de acuerdo con la distancia al objetivo, permitiendo un comportamiento más agresivo a medida que el robot se acerca, mientras que el par virtual se controla de forma diferencial en función de la distancia, asegurando un agarre físico efectivo.

En términos de control, el brazo del robot se ajusta para evitar configuraciones singulares y garantizar un agarre seguro y preciso, incluyendo un factor de penalización que entra en juego cuando el robot se aleja de su posición de reposo. Además, el artículo presenta cómo se implementa el control en el brazo considerando la interacción con obstáculos y otros elementos en el entorno.

Los resultados experimentales presentados en el artículo se basan en un escenario donde se utiliza una plancha de madera con un marcador Apriltag como objetivo. Estos resultados validan la eficacia y la precisión del enfoque propuesto, demostrando su capacidad para rastrear y agarrar objetivos en movimiento de manera exitosa en situaciones del mundo real.

2.3.3 Integración y contribución

La revisión de la literatura aporta una sólida base de conocimiento para contextualizar y dar significado al objetivo final de este estudio. Los tres artículos seleccionados desempeñan un papel esencial al proporcionar una visión amplia y relevante de la aplicación de la robótica en tareas logísticas.

En todos los artículos revisados, se destaca el crecimiento en el uso de robots manipulativos y la creciente integración de cobots y robots móviles en la industria. Además, subrayan la colaboración entre humanos y robots como un factor clave para mejorar la eficiencia y la seguridad en entornos industriales.

En primer lugar, los dos primeros artículos ejemplifican de manera convincente la utilidad y la importancia de la automatización en tareas logísticas y de manipulación, lo cual es directamente relevante para el objetivo de este trabajo.

El tercer artículo, se consolida como un componente fundamental para la implementación del objetivo final de este trabajo. La guía que proporciona en el manejo del brazo robot y el control visual en este contexto resulta sumamente relevante y ofrece un punto de partida sólido.

Si bien el tercer artículo se empleará como referencia principal, es importante señalar que la implementación en este estudio requerirá adaptaciones y consideraciones específicas para abordar las limitaciones y características únicas en este trabajo. Por lo tanto, el enfoque de este estudio se basa en los principios del tercer artículo, mientras se ajusta y personaliza conforme a las condiciones y requisitos específicos del proyecto.

3 MATERIALES Y MÉTODOS

En este capítulo se va a realizar una descripción de algunos de los elementos utilizados, tanto software como hardware, para llevar a cabo el desarrollo del trabajo, así como las técnicas de las que se necesita tener conocimiento para avanzar en los objetivos del mismo. A continuación, se va a enumerar cada uno de los elementos o técnicas con una breve descripción, para posteriormente, incluir una explicación más detallada sobre cada uno de ellos:

- **ROS (*Robotic Operating System*):** es un conjunto de bibliotecas y herramientas software utilizado en robots que provee servicios estándar de un sistema operativo, como la abstracción del hardware o el control de dispositivos de bajo nivel [24]. El robot móvil utilizado en el trabajo tiene implementado este sistema.

- **Robot RB-1 BASE:** Producido por Robotnik Automation, el modelo RB-1 BASE es un robot móvil autónomo diseñado para la manipulación y transporte de cargas en entornos industriales. Equipado con ruedas omnidireccionales y sensores de navegación, este robot ofrece una gran movilidad y capacidad de adaptación [25].

- **Robot UR3e:** Fabricado por Universal Robots, el UR3e es un robot colaborativo de brazo articulado diseñado para trabajar en entornos colaborativos junto con humanos. Es versátil y preciso en una variedad de tareas. Su programación intuitiva y su capacidad de seguridad intrínseca lo convierten en una opción popular en aplicaciones industriales y de investigación [26].

- **Intel Realsense D435:** Dispositivo de visión estereoscópica que ofrece capacidades avanzadas de percepción tridimensional. Equipada con dos cámaras y un sensor de profundidad, la RealSense D435 permite la captura de imágenes y la estimación de distancias con precisión. Esta cámara es especialmente valiosa en aplicaciones de visión por computadora y robótica [27].

- **Raspberry Pi:** Es un ordenador de placa única de bajo costo y tamaño compacto, diseñado para una variedad de aplicaciones informáticas y proyectos de electrónica. Equipado con puertos GPIO y soporte para varios sistemas operativos, la Raspberry Pi se ha convertido en una plataforma popular para el desarrollo de soluciones personalizadas y la ejecución de tareas computacionales diversas [28].

- **Python:** Es un lenguaje de programación versátil y de alto nivel ampliamente utilizado en el desarrollo de software. Ofrece una sintaxis clara y legible, lo que facilita la escritura y el mantenimiento de código. Python es conocido por su flexibilidad y amplia gama de bibliotecas, lo que permite a los programadores acceder a funcionalidades diversas sin necesidad de escribir código desde cero [29]. En este proyecto, todos los programas implementados tanto en los robots como en la Raspberry han sido desarrollados con Python.

- **AruCo:** Son marcadores visuales utilizados en aplicaciones de visión por computadora y realidad aumentada. Estos códigos bidimensionales contienen información que puede ser detectada y decodificada por sistemas de visión para determinar la posición y orientación relativa de objetos en un entorno. Los códigos ArUco son útiles para el seguimiento y la

calibración de cámaras, así como para la identificación de objetos en sistemas robóticos [30].

- **OpenCV:** Librería de Python que proporciona una gama diversa de algoritmos y funciones diseñados para el procesamiento y análisis de imágenes y videos. Su versatilidad se extiende a aplicaciones de visión artificial y robótica, permitiendo la detección, seguimiento y exploración de objetos en diversos contextos [31].

3.1 ROS

ROS (Robot Operating System) es un conjunto de librerías y herramientas de código abierto diseñadas para facilitar el desarrollo de aplicaciones para robots. Este entorno proporciona una serie de servicios que son comparables a funciones de sistemas operativos tradicionales, como la abstracción de hardware, el control de dispositivos, la comunicación entre procesos y la gestión de paquetes. Aunque no es un sistema operativo completo en sí mismo, ya que ROS debe operar sobre la plataforma Ubuntu (Linux). Una de las ventajas más destacadas de ROS es su capacidad para estandarizar programas en el ámbito robótico, permitiendo la creación de aplicaciones compatibles entre distintos robots.

ROS opera a través de paquetes, que constituyen la unidad fundamental de organización del software en este entorno. Estos paquetes pueden englobar librerías, ejecutables, scripts y diversas aplicaciones. Para asegurar la compatibilidad con ROS, cada paquete debe incluir archivos que describen sus atributos y un archivo “CMakeLists.txt”, un documento de texto que especifica cómo compilar el código y gestionar su instalación.

ROS adopta una estructura basada en nodos para la ejecución de sus aplicaciones, siendo el modelo de comunicación usado el de suscripción/publicación. Los nodos representan ejecutables individuales capaces de publicar o suscribirse a temas (*topics*) para facilitar la comunicación con otros nodos. Mediante el comando *rqt_graph*, se puede visualizar un gráfico dinámico que brinda información acerca de los nodos en funcionamiento y sus interacciones. En la figura 8, se destacan varios nodos, como */base_controller*, */robot_tf_publisher*, entre otros, que establecen comunicación a través de *topics* (representados por flechas). En esta dinámica, ciertos nodos emiten mensajes en los *topics*, siendo recibidos por aquellos nodos que se han suscrito a los mismos. Un ejemplo ilustrativo es el nodo */base_controller*, que emite mensajes en un *topic* al cual el nodo */motorcmdveldx* está suscrito, posibilitando así la recepción de los mensajes transmitidos. Para habilitar la comunicación fluida entre múltiples nodos mediante un *topic*, es esencial que todos los nodos involucrados compartan el mismo tipo de mensaje.

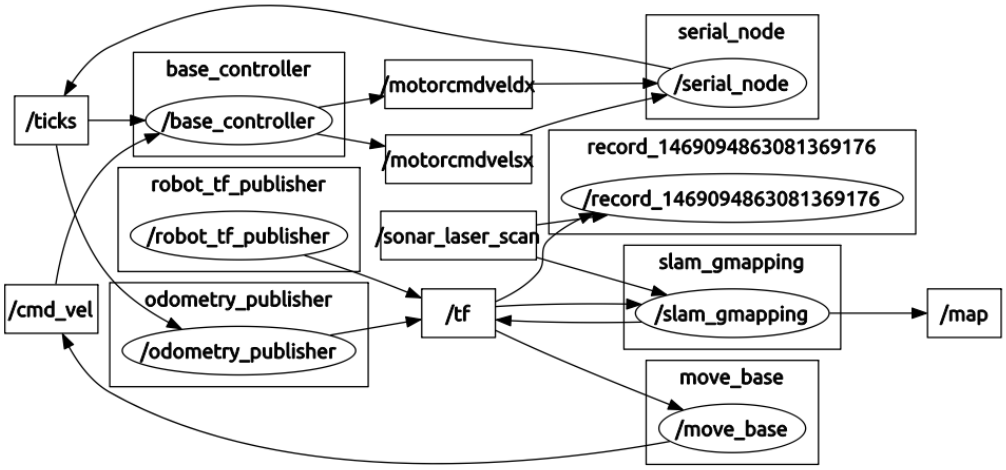

Figura 8. Ejemplo de comunicación entre nodos con rqt_graph

Otra metodología de comunicación entre nodos en ROS es a través de servicios [32]. Los servicios facilitan la interacción bidireccional entre un cliente y un servidor, a diferencia de los *topics*. Adicionalmente, esta comunicación es sincrónica, lo que significa que el cliente envía una solicitud y espera hasta recibir respuesta, por lo que únicamente se deben utilizar para para llevar a cabo procesos o cálculos rápidos. Los servicios posibilitan que los nodos envíen solicitudes y obtengan respuestas correspondientes.

Además de los servicios, existe otro mecanismo de comunicación cliente-servidor conocido como *actions* [33]. Las *actions* son similares a los servicios, pero permiten la capacidad de cancelar la solicitud durante su ejecución y recibir actualizaciones periódicas sobre el avance de dicha solicitud. A diferencia de los servicios, no bloquean el nodo mientras espera la respuesta, lo que las hace apropiadas para tareas o cálculos más complejos.

Para establecer la comunicación entre el cliente y el servidor a través de las *actions*, es necesario definir una serie de mensajes (Figura 9). Desde el cliente, se pueden emitir dos tipos de mensajes: el mensaje de objetivo (*goal*) y el de cancelación. Por otro lado, desde el servidor se pueden enviar tres tipos de mensajes: el *feedback*, el mensaje de estado y el mensaje de resultado. El *feedback* informa al cliente que el objetivo ha sido recibido. El mensaje de estado proporciona datos relevantes para el objetivo establecido. Por último, el mensaje de resultado es enviado cuando el objetivo ha sido completado.

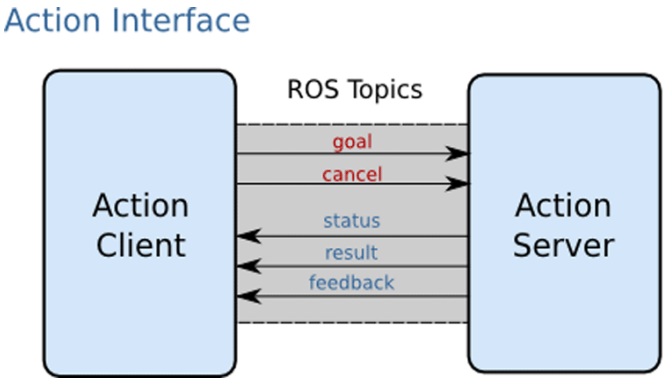

Figura 9. Comunicación a través de action [34]

3.2 Robot RB-1 BASE

El robot móvil RB-1 BASE es una plataforma de tracción diferencial diseñada por Robotnik con el propósito de desarrollar aplicaciones versátiles en interiores (Figura 10). Este vehículo ofrece una capacidad flexible para transportar diversas cargas y materiales, así como para integrar sistemas adicionales, como brazos o torsos robóticos en su estructura.

Figura 10. Robot RB-1 BASE

Uno de los aspectos destacados del RB-1 BASE es su capacidad para incorporar una gran variedad de sensores, entre ellos los sensores RGBD. Estos sensores combinan la información de una cámara de profundidad (D) con una cámara RGB convencional para proporcionar una percepción tridimensional del entorno [35]. Esta característica permite al robot capturar detalles precisos sobre la geometría y la apariencia de los objetos que lo rodean, lo que resulta crucial para la navegación y la detección de obstáculos en entornos complejos.

En la configuración estándar del RB-1 BASE, se integran dos tipos de sensores específicos: el UST-20LX y el sensor Orbbec Astra. El UST-20LX, de Hokuyo (Figura 11), es un láser de escaneo 2D que ofrece un amplio rango de detección (de 5 a 30-60 metros, según las especificaciones) y es ampliamente utilizado para la localización y la percepción del entorno [36].

Figura 11. UST-20LX

Por otro lado, el sensor Astra, de Orbbec (Figura 12) es un sensor RGBD que proporciona información detallada sobre la profundidad y el color de los objetos circundantes [37]. Esta combinación de sensores permite al robot RB-1 BASE tener una percepción sólida y confiable del entorno, lo que es esencial para la navegación autónoma y la evasión de obstáculos.

Figura 12. Orbbec Astra

La arquitectura de software del RB-1 BASE se compone de varios componentes esenciales. Esto incluye un sistema de control, un sistema de localización basado en láser, un sistema de navegación y una interfaz de usuario HMI (Human-Machine Interface) básica. Estos componentes se encuentran integrados y operan en un entorno de control modular basado en ROS (Robot Operating System), lo que proporciona una estructura robusta y altamente adaptable para la programación y operación del robot.

Este robot encuentra una amplia gama de aplicaciones, destacando su papel fundamental en el campo de la logística con su capacidad de transporte autónomo. Además, su versatilidad lo hace apto para su uso en la monitorización remota, investigación y desarrollo, así como en diversas aplicaciones de propósito general en interiores. Su diseño y funcionalidad lo convierten en una herramienta valiosa para abordar una variedad de desafíos y necesidades en el ámbito robótico y tecnológico.

3.2.1 Especificaciones técnicas

- **Dimensiones:** 515 x 303-338 mm.
- **Peso:** 30 Kg.
- **Capacidad de carga:** 50 Kg.
- **Entorno**: Interior.
- **Velocidad máxima:** 1.5 m/s.
- **Máxima pendiente:** 8 %.
- **Sistema de tracción:** Accionamiento diferencial.
- **Autonomía:** 6 horas en aplicación industrial.
- **Batería:** LiFePO4 30Ah@24V.

- **Motorización:** Servomotores 2 x 250W.
- **Rango de temperatura:** 0°C a +50°C.
- **Controlador:** Industrial PC Intel i7, arquitectura abierta basada en ROS.
- **Comunicación:** WiFi 802.11ac, 4G.
- **Conectividad:** 2 x USB, 1 x Ethernet y 1 x HDMI.

3.3 Robot UR3e

El robot colaborativo UR3e de Universal Robots (Figura 13) es una herramienta destacada en el ámbito industrial, caracterizada por su versatilidad y aplicabilidad en diversas tareas. Su presencia en entornos de trabajo ha influido en la automatización de procesos y ha impulsado la eficiencia en múltiples aplicaciones.

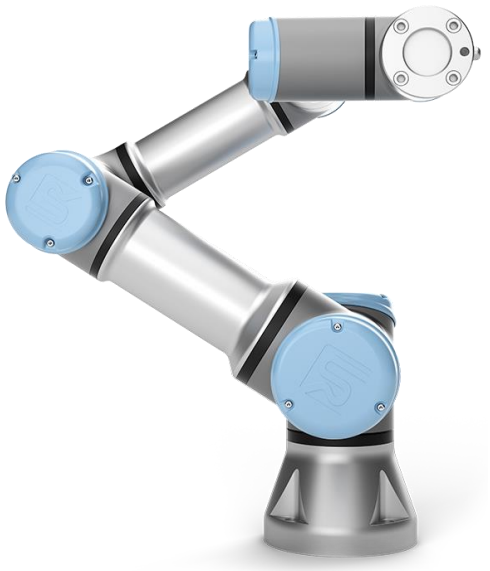

Figura 13. UR3e

El UR3e, diseñado como un sistema compacto y ligero, ha encontrado su lugar en operaciones con limitaciones de espacio. Con un peso de 11 kg y una carga útil de 3 kg, así como un radio de acción de 500 mm, se adapta a tareas de manipulación y ensamblaje en entornos reducidos. Su capacidad de rotación de ±360 grados en todas las articulaciones, junto con una rotación infinita en el extremo, le confiere precisión en actividades como el atornillado y el ensamblaje.

Este robot se distingue no solo por su solidez mecánica, sino también por su enfoque en la facilidad de uso. Su programación intuitiva y su instalación eficiente permiten la participación de operarios con diversos niveles de experiencia en la automatización de tareas. Este enfoque de accesibilidad en la tecnología robótica influye en la implementación de la automatización en las líneas de producción, aliviando a los trabajadores de tareas repetitivas y abriendo posibilidades para la creatividad y la innovación en la operación industrial.

El UR3e ha encontrado aplicaciones en diversos contextos industriales:

- Empaquetado y paletización: Asegura la eficiencia en la organización y el empaquetado de productos.
- Lijado y pulido: Realiza acabados de alta calidad en superficies con gran precisión.
- Soldadura colaborativa: Imita los movimientos humanos para realizar soldaduras.
- Atornillado preciso: Ejecuta operaciones de atornillado en componentes con exactitud.
- Montaje ágil: Contribuye a la unión precisa de piezas y componentes en ensamblajes complejos.
- Manipulación adaptable: Agarra y maneja objetos de diversas formas y tamaños.
- Alimentación de máquinas: Asegura un suministro constante de materiales a máquinas en líneas de producción.

El UR3e se presenta como un componente clave en la automatización industrial, desempeñando un papel fundamental en la potenciación de la eficiencia en diversos contextos. Su notable versatilidad y capacidad para asumir múltiples roles en entornos de producción consolidan su posición como una herramienta de gran valor en la optimización de procesos industriales. Su influencia se extiende a través de una variedad de aplicaciones, lo que lo convierte en un recurso valioso para impulsar la productividad y la calidad en distintas industrias.

3.4 Intel Realsense D435

La cámara de profundidad Intel RealSense D435 (Figura 14) se distingue por su enfoque en la calidad visual y su versatilidad funcional. Estas características la hacen adecuada para una amplia gama de aplicaciones. Su característica distintiva es su extenso campo de visión, que abarca una perspectiva más amplia de la escena y resulta muy importante en contextos donde la captura completa de detalles es esencial.

Figura 14. Cámara Intel RealSense D435

En el ámbito de la robótica y la realidad aumentada, la cámara D435 destaca por su capacidad para proporcionar una percepción tridimensional precisa y detallada, lo que la hace adecuada para la navegación de robots en entornos dinámicos y el reconocimiento de objetos en tiempo real. Esta

cámara ofrece una mejora en la percepción visual a los dispositivos inteligentes, permitiéndoles interactuar de manera más efectiva con su entorno y realizar tareas más complejas con precisión.

La D435 está equipada con un sensor de profundidad de obturador global, por lo que es capaz de capturar imágenes nítidas incluso en situaciones de movimiento rápido. Su rango de alcance de hasta 10 metros la convierte en una elección popular para aplicaciones que requieren una detección de profundidad confiable en entornos dinámicos y de gran extensión.

El obturador global es una característica técnica que permite capturar una imagen completa de la escena en un solo instante, eliminando la distorsión que podría ocurrir con un sensor de obturación rodante, en el que la captura se realiza línea por línea de manera secuencial [38]. En esencia, mientras que un sensor de obturación rodante captura la imagen en franjas a medida que el obturador se desplaza por el sensor, el obturador global captura la totalidad de la imagen simultáneamente, resultando en una representación más precisa y detallada de la escena.

Esta tecnología de obturador global en la cámara D435 tiene una influencia significativa en su capacidad para capturar imágenes de alta calidad, incluso en situaciones de movimiento rápido. Al eliminar la posibilidad de distorsión durante la captura de la imagen, el obturador global garantiza que los objetos en movimiento sean representados con precisión y claridad, lo que es especialmente crucial para aplicaciones que involucran detección y seguimiento de objetos en tiempo real, como en la robótica y la realidad aumentada.

La cámara Intel RealSense D435 cumple de manera efectiva con su función técnica en diversos campos. Su capacidad para comprender y mapear el entorno tridimensional ofrece nuevas perspectivas en el diseño de productos y soluciones. Se encuentra comúnmente en aplicaciones de automatización industrial y ha mejorado la interacción entre humanos y máquinas, lo que ha resultado en avances significativos en diversas industrias.

La capacidad de integrar la cámara D435 con el kit de desarrollo de software (SDK) Intel RealSense 2.0 posibilita a los desarrolladores y creadores una plataforma de experimentación y desarrollo robusta. Esta integración permite a los investigardores explorar y crear nuevas aplicaciones y soluciones, desde la simulación de ambientes virtuales hasta la navegación autónoma de robots.

3.5 Raspberry Pi

La Raspberry Pi (Figura 15) se define como una minicomputadora que ofrece compatibilidad con diferentes dispositivos de entrada y salida, como monitores, televisores, ratones y teclados. Esto la hace una solución de computación eficiente en términos de costos y funcionalidad.

Figura 15. Raspberry Pi

Las aplicaciones de Raspberry Pi abarcan una amplia variedad de campos, desde la enseñanza de diferentes lenguajes de programación hasta la gestión y orquestación de redes informáticas. Esta versatilidad ha contribuido a su creciente popularidad en los últimos años, superando las expectativas iniciales que se tenían para esta herramienta tecnológica.

Raspberry Pi es un dispositivo programable, equipado con las características más importantes de una placa base de computadora convencional, aunque sin incluir periféricos ni almacenamiento interno. Para poner en funcionamiento el dispositivo, se requiere la inserción de una tarjeta SD en el espacio provisto, la cual debe contener el sistema operativo necesario para el arranque. Un aspecto destacado de las Raspberry es su compatibilidad con el sistema operativo Linux, lo que contribuye a una utilización eficiente de la memoria.

Una vez configurado el sistema operativo, Raspberry Pi puede conectarse a dispositivos de salida, como monitores, mediante la interfaz multimedia de alta definición (HDMI). Adicionalmente, es esencial conectar dispositivos de entrada, como ratones o teclados, para interactuar con el dispositivo. Las posibilidades y aplicaciones exactas de la plataforma varían según las preferencias del usuario, abarcando una amplia gama de funciones y usos.

3.5.1 Características más importantes de Raspberry Pi

En la placa se incorporan varias características, cada una con sus usos específicos. En general, las diferentes características proporcionan elementos de un ordenador estándar actual: velocidad y calidad del procesador, Bluetooth, conexiones y puertos periféricos, y compatibilidad de software. Las características de las computadoras Raspberry Pi que hacen posible todo esto incluyen [39]:

- **Unidad Central de Procesamiento (CPU):** Al igual que en cualquier computadora, Raspberry Pi está equipada con una Unidad Central de Procesamiento (CPU). Es el elemento encargado de ejecutar instrucciones a través de operaciones lógicas y matemáticas. Las placas Raspberry Pi utilizan procesadores de la serie ARM11 para llevar a cabo estas tareas fundamentales.

- **Puerto HDMI**: La placa Raspberry Pi está equipada con un puerto HDMI (Interfaz Multimedia de Alta Definición), lo que posibilita opciones de visualización de salida de video. Un cable HDMI permite conectar la Raspberry Pi a televisores o monitores, ofreciendo una visualización del dispositivo. Además, se incluye un puerto RCA para alternativas de visualización.

- **Unidad de Procesamiento Gráfico (GPU):** Una parte esencial de Raspberry Pi es su Unidad de Procesamiento Gráfico (GPU), que contribuye a acelerar los cálculos relacionados con imágenes y gráficos.

- **Memoria (RAM):** La Memoria de Acceso Aleatorio (RAM) es crucial en cualquier sistema informático, ya que almacena información en tiempo real para un acceso rápido.

- **Puerto Ethernet**: La conectividad a Internet por cable se logra a través del puerto Ethernet presente en la Raspberry Pi. Este componente es esencial para actualizaciones de software, navegación web y más.

- **Ranura para tarjeta SD**: Raspberry Pi emplea una ranura para tarjetas Secure Digital (SD) como su medio de almacenamiento principal. Al insertar una tarjeta SD con el sistema operativo adecuado, se puede iniciar y almacenar datos esenciales.

- **Pines de Entrada y Salida de Propósito General (GPIO)**: Los pines GPIO, dispuestos en un lado de la placa, son elementos cruciales para expandir las capacidades de Raspberry Pi. Estos pines permiten la interacción con otros dispositivos y circuitos electrónicos, lo que amplía la gama de aplicaciones del dispositivo

- **LEDs**: Cinco diodos emisores de luz (LEDs) forman un grupo que ofrece información visual sobre el estado actual de Raspberry Pi, proporcionando una referencia rápida al usuario.

- **Puertos USB**: La capacidad de conectar dispositivos periféricos es fundamental, y los puertos USB en Raspberry Pi facilitan la conexión de teclados, ratones, discos duros y otros componentes, como cámaras.

- **Fuente de alimentación**: La Raspberry Pi se alimenta a través de un conector micro USB de 5V, con la cantidad de energía consumida variando según el uso y los dispositivos periféricos conectados. Este componente es esencial para mantener en funcionamiento la computadora y asegurar un rendimiento óptimo.

3.6 Python

Python es un lenguaje de programación ampliamente reconocido por su versatilidad y utilidad en diversas aplicaciones tecnológicas. Introducido a finales de la década de 1980, Python ha ganado una amplia aceptación en numerosos campos, desde el desarrollo web hasta el análisis de datos y el aprendizaje automático.

La legibilidad es una de las características sobresalientes de Python. Su sintaxis clara y estructurada permite a los programadores escribir y entender el código de manera eficiente. Este enfoque favorece la colaboración en equipos de desarrollo y facilita la creación de proyectos bien documentados y más manejables.

La simplicidad es otro principio fundamental en el diseño de Python, enfatizando la claridad y la simplicidad en lugar de la complejidad. Esta filosofía de diseño se traduce en un lenguaje fácil de aprender y usar, lo que lo convierte en una opción atractiva para programadores principiantes y experimentados por igual.

La flexibilidad es otra característica distintiva de Python. Admite múltiples estilos de programación y ofrece una amplia biblioteca estándar que abarca desde manipulación de cadenas hasta operaciones de red y matemáticas. Esta riqueza de recursos facilita el desarrollo rápido de aplicaciones y sistemas complejos.

En términos de aplicaciones, Python ha encontrado un hogar en una variedad de campos. En el desarrollo web, frameworks como Django [40] y Flask [41] permiten la creación eficiente de aplicaciones dinámicas y sitios interactivos. En análisis de datos, bibliotecas como NumPy [42] y pandas [43] proporcionan herramientas poderosas para manipular y procesar información. Además, Python se ha convertido en un pilar en el aprendizaje automático y la inteligencia artificial, gracias a bibliotecas como TensorFlow [44], que permite la construcción de modelos avanzados.

En conclusión, Python ha experimentado un crecimiento constante desde su concepción, convirtiéndose en un lenguaje de programación altamente influyente en diversos campos tecnológicos [45]. Su legibilidad, simplicidad y flexibilidad lo han convertido en una herramienta esencial para desarrolladores de todo el mundo, y su presencia en la industria es una evidencia de su impacto duradero en la programación moderna. En la figura 16, se presenta un gráfico que ilustra la evolución del uso de los lenguajes de programación hasta el año 2022, destacando el crecimiento de Python, que se convirtió en el lenguaje más utilizado en ese período.

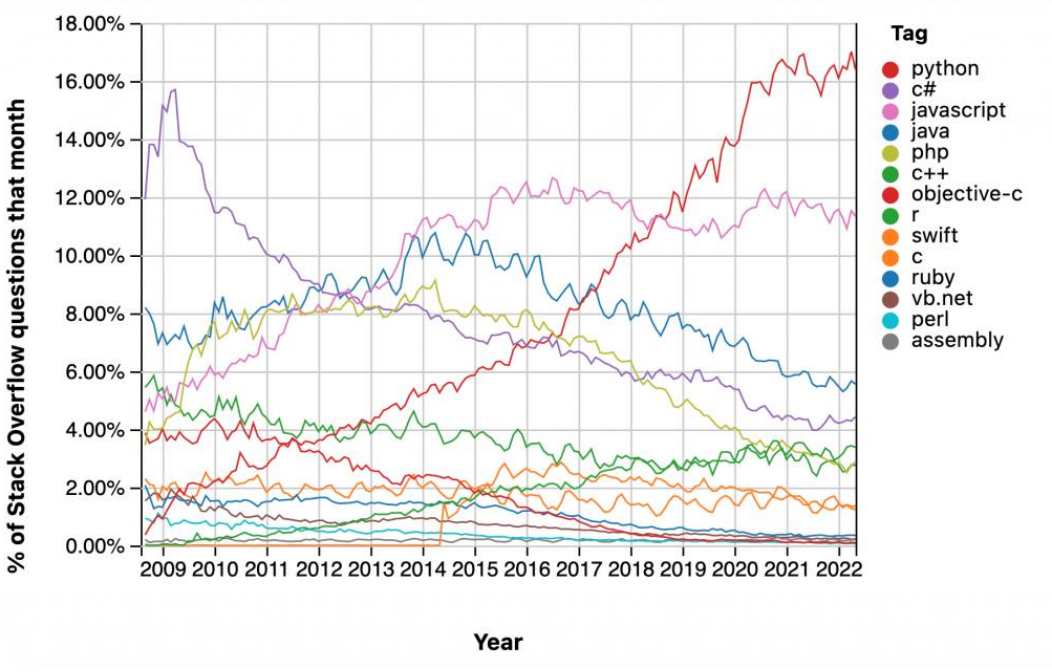

Figura 16. Uso de lenguajes de programación por año [45]

3.7 AruCo

Los códigos ArUco (Figura 17) representan una innovadora solución en el ámbito de la visión por computadora y la realidad aumentada, desempeñando un papel esencial en la identificación y seguimiento de objetos en tiempo real. Estos códigos, cumplen una función crucial al proporcionar puntos de referencia visuales que permiten a los sistemas de visión por computadora ubicar y rastrear objetos con precisión [47].

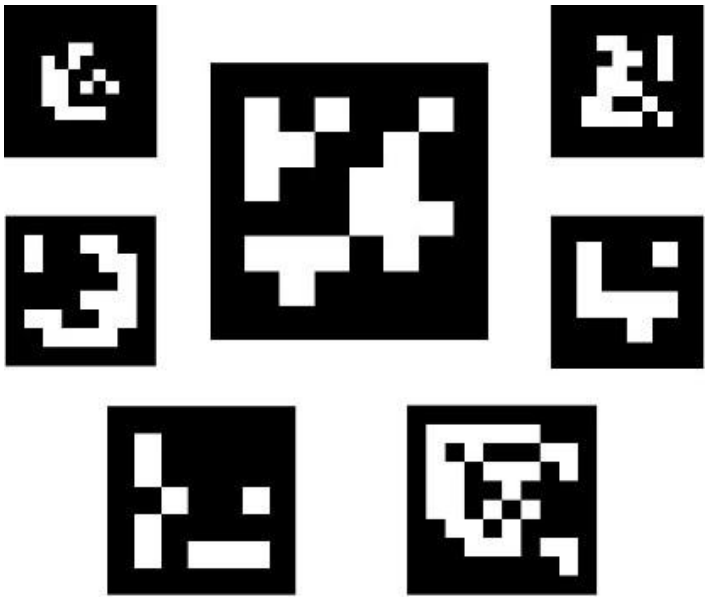

Figura 17. Códigos AruCo

Estos patrones bidimensionales, compuestos por cuadrados negros y blancos dispuestos en configuraciones específicas, se han convertido en una herramienta esencial en una variedad de aplicaciones. Desde la detección y rastreo de objetos en imágenes o secuencias de video hasta la superposición de objetos virtuales en entornos de realidad aumentada, los códigos ArUco ofrecen una solución robusta y eficiente en términos de procesamiento.

Los códigos ArUco destacan por una serie de atributos esenciales que los convierten en herramientas valiosas en su campo de aplicación. Entre las características más significativas, cabe resaltar las siguientes:

- **Robustez:** Los códigos ArUco están diseñados para resistir la distorsión, el ruido y las posibles variaciones en la iluminación. Esta cualidad les permite ser detectados y seguidos con precisión en una gran variedad de entornos y situaciones.

- **Facilidad de Detección**: Los códigos ArUco son rápidos y eficientes en su detección y decodificación, lo que los hace adecuados para aplicaciones en tiempo real. Los algoritmos de detección no requieren un alto poder de procesamiento, facilitando su implementación en diversos sistemas.

- **Versatilidad**: Los códigos ArUco están disponibles en diferentes tamaños y niveles de complejidad, lo que los hace adaptables a diversas aplicaciones, desde la calibración de cámaras hasta la navegación de robots y la creación de experiencias de realidad aumentada.

En lo que respecta a sus utilidades, algunas de sus aplicaciones más relevantes son las que se detallan a continuación:

- **Detección y seguimiento:** Los códigos ArUco se emplean para detectar y rastrear objetos en imágenes o secuencias de video capturadas por cámaras. Mediante el análisis de la disposición de los cuadrados, un algoritmo puede identificar de manera única cada código y determinar su posición y orientación.

- **Calibración de Cámaras**: La inserción de códigos ArUco en entornos conocidos permite calibrar cámaras en sistemas de visión por computador. Analizando su apariencia en las imágenes capturadas, se pueden estimar los parámetros intrínsecos y extrínsecos de la cámara, mejorando así la precisión del proceso de visión.

- **Realidad Aumentada y realidad virtual**: En el campo de la realidad aumentada y la realidad virtual, los códigos ArUco actúan como marcadores visuales que posibilitan la superposición de objetos virtuales en el mundo real. La detección de estos códigos permite a los algoritmos determinar la posición y orientación precisa del objeto virtual en relación con los códigos.

- **Navegación de Robots**: En la navegación de robots, los códigos ArUco desempeñan un papel crucial al ayudar a determinar la posición y orientación del robot en su entorno. Esto es especialmente relevante en situaciones donde la navegación precisa es importante, como en espacios estrechos o en la interacción segura con seres humanos.

3.8 OpenCV

OpenCV (Open Source Computer Vision Library) es una potente biblioteca de software de código abierto que ha cambiado la manera en la que se interactúa con imágenes y videos a través de la visión por computador. Diseñada para realizar una gran variedad de tareas relacionadas con el procesamiento y análisis de imágenes, OpenCV se ha convertido en una herramienta fundamental para científicos, ingenieros e investigadores [48].

Su base está en C++, pero su flexibilidad se extiende a un extenso abanico de lenguajes de programación, incluido el Python, lo que facilita su adopción y uso por parte de una comunidad diversa y global. OpenCV ofrece una colección completa de funciones y algoritmos optimizados que abarcan desde operaciones básicas de manipulación de imágenes hasta tareas más avanzadas, como el reconocimiento de objetos y la detección de rostros. Una de las características más notables de OpenCV es su capacidad para trabajar con imágenes y videos en tiempo real, lo que permite aplicaciones interactivas y de respuesta rápida.

OpenCV desempeña un papel importante en diversos campos y aplicaciones, entre los cuales se destacan:

- **Visión por computador**: OpenCV proporciona herramientas para el procesamiento y análisis de imágenes y videos, siendo esencial en aplicaciones de visión por computador como la detección de objetos, el seguimiento de movimiento, la segmentación de imágenes y la clasificación de objetos.

- **Robótica**: En el ámbito de la robótica, OpenCV permite que los robots perciban y comprendan su entorno a través de cámaras y sensores. Esto facilita la navegación de los robots, la identificación de objetos y la toma de decisiones basadas en información visual.

- **Automatización industrial**: En la industria se emplea para la inspección de productos, el control de calidad, el seguimiento de objetos en líneas de producción y la automatización de procesos.

- **Realidad aumentada y virtual**: OpenCV es esencial en el desarrollo de aplicaciones de realidad aumentada y virtual, permitiendo rastrear objetos y superponer contenido digital en el mundo real.

- **Reconocimiento facial**: OpenCV ofrece técnicas de detección y reconocimiento facial, utilizadas en seguridad, autenticación y análisis de emociones.

- **Vehículos autónomos**: OpenCV se emplea en la percepción visual de vehículos autónomos para detectar peatones, señales de tráfico, vehículos y otros objetos en la carretera.

OpenCV ofrece una amplia variedad de funciones y módulos para el procesamiento y análisis de imágenes y videos. Algunas de las funciones más destacadas son:

- **Carga y visualización de imágenes y videos**: Permite cargar imágenes y videos desde diversos formatos y fuentes, proporcionando herramientas para mostrarlos en ventanas y dispositivos de visualización.

- **Procesamiento de imágenes**: Ofrece un conjunto completo de operaciones de procesamiento de imágenes, como filtrado, convolución, suavizado, mejora de contraste y detección de bordes.

- **Detección y seguimiento de objetos**: Incluye algoritmos para detectar y seguir objetos en imágenes y videos, como detección de rostros, características (como esquinas), seguimiento óptico de flujo y detección de patrones.

- **Reconocimiento de patrones y aprendizaje automático**: Soporta la implementación de algoritmos de aprendizaje automático y reconocimiento de patrones, como SVM [49] (Support Vector Machines), redes neuronales y algoritmos de agrupación.

- **Calibración de cámaras y geometría estéreo**: Ofrece herramientas para calibrar cámaras, estimar la geometría estéreo y realizar reconstrucción 3D a partir de imágenes.

- **Detección de bordes y contornos**: Incluye algoritmos para detectar bordes y contornos en imágenes, como Canny [50], detección de líneas de Hough [51] y contornos activos.

4 DESARROLLO DEL TRABAJO

En este capítulo se explicarán las diferentes fases que se han seguido en el trabajo para lograr los subobjetivos marcados previamente en el proyecto, de forma que se consiga finalmente cumplir con el objetivo principal especificado en la primera sección de esta memoria. Para lograrlo de manera efectiva, las tareas se han agrupado en tres áreas fundamentales: calibración de la cámara y transformaciones de los sistemas de coordenadas, la implementación del control de agarre de objetos y componentes hardware y comunicación entre ellos. Cada una de estas áreas representa una etapa esencial en el camino hacia la consecución del objetivo principal. A continuación, se procederá a realizar una explicación detallada acerca de cómo se ha llevado a cabo la implementación de estas tareas, detallando los pasos que se han seguido en cada una de ellas.

4.1 Calibración de la cámara y sistemas de coordenadas

Para lograr una manipulación precisa de los objetos, es esencial determinar las coordenadas del objetivo dentro de un sistema de coordenadas específico. En el contexto de este proyecto, se emplean comandos *moveJ_IK* para controlar el brazo robótico UR3. Estos comandos inducen movimientos lineales en el espacio de las articulaciones, lo que significa que controlan el desplazamiento de las articulaciones de manera que estas se muevan a lo largo de una trayectoria lineal, modificando sus posiciones de forma uniforme y continua. La utilización de este tipo de movimientos permite un seguimiento fluido y suave del objeto, especialmente cuando este se encuentra en movimiento. Sin embargo, para ejecutar este tipo de movimiento, es necesario especificar la posición final de la herramienta en relación con la base del brazo robótico. Por lo tanto, es esencial determinar las coordenadas del objeto en referencia a este sistema de coordenadas.

Para alcanzar este objetivo, se deben seguir una serie de pasos específicos. En primer lugar, es fundamental realizar una calibración precisa de la cámara que se utiliza para detectar los objetos o, en este caso, el marcador AruCo presente en el objeto. Una vez que la cámara está calibrada correctamente, podemos obtener la posición del objeto en relación con el sistema de coordenadas de la cámara.

Luego, es necesario llevar a cabo las transformaciones de base necesarias para obtener la posición del objeto en relación con la base del brazo robot. Esto implica realizar los ajustes adecuados para asegurarse de que las coordenadas del objeto sean coherentes y utilicen el mismo marco de referencia que el brazo robot. Esta alineación precisa es esencial para poder enviar las coordenadas del objeto al brazo robot de forma adecuada.

4.1.1 Calibración de la cámara

La calibración de una cámara en robótica es un proceso de gran importancia debido a su impacto en la precisión y confiabilidad de las medidas realizadas a partir de las imágenes capturadas. Las cámaras a menudo introducen distorsiones en las imágenes, especialmente en los bordes y esquinas, debido a imperfecciones en las lentes o a la geometría misma de la cámara. La calibración permite modelar y corregir estas distorsiones, lo que es crítico para aplicaciones que requieren medidas precisas

La calibración de una cámara se traduce en la obtención de sus parámetros intrínsecos y extrínsecos que son esenciales para el procesamiento y análisis de imágenes. Estos parámetros permiten

corregir las imperfecciones y distorsiones inherentes de la cámara, lo que, a su vez, facilita la conversión precisa entre las coordenadas de píxeles de la imagen y las coordenadas del mundo real.

Los parámetros intrínsecos son propiedades internas de la cámara que describen cómo la cámara captura la luz y la convierte en una imagen. Estos parámetros son específicos de cada cámara y son independientes de su posición o movimiento en el espacio. Se incluyen la distancia focal de la cámara, el punto principal (el centro de la imagen) y los coeficientes de distorsión radial y tangencial. Estos parámetros son cruciales para corregir las distorsiones y mapear con precisión los píxeles de la imagen a las coordenadas del mundo real.

Los parámetros extrínsecos describen la posición y orientación de la cámara en el espacio 3D en relación con un sistema de coordenadas del mundo real. Estos parámetros son fundamentales para la reconstrucción tridimensional y la navegación, ya que permiten determinar la ubicación precisa de la cámara en un entorno tridimensional.

A partir de los parámetros intrínsecos y extrínsecos, se pueden calcular matrices de transformación que permiten la conversión entre coordenadas de píxeles y coordenadas del mundo real.

Para realizar una calibración precisa de la cámara utilizada en este proyecto, se empleó un patrón como el que se ve en la figura 18. Se utilizó un patrón de tipo tablero de ajedrez ya que contiene líneas rectas y esquinas bien definidas que son fácilmente detectables en una imagen. Al capturar fotografías de este patrón desde varios ángulos y posiciones, es posible utilizar las esquinas de los cuadrados para estimar los parámetros de la cámara.

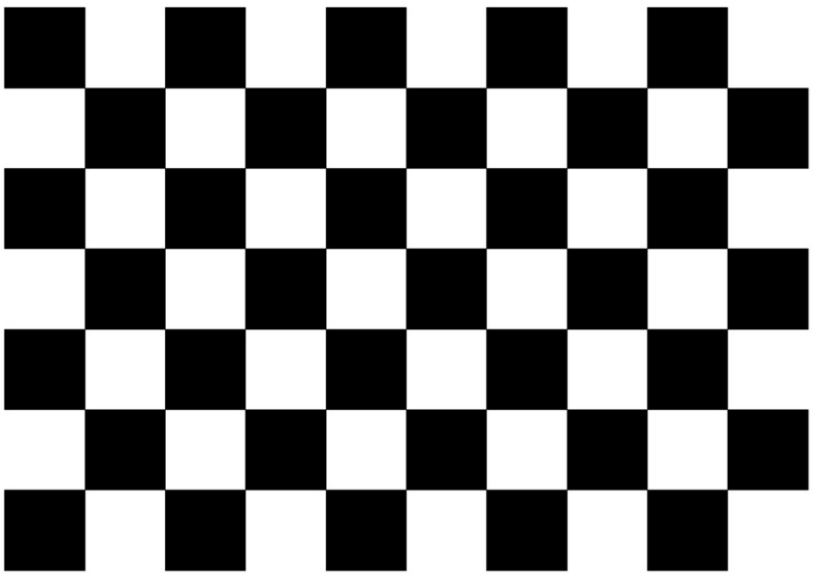

Figura 18. Patrón utilizado en la calibración

Para adquirir imágenes desde diversos ángulos y posiciones del patrón, se desarrolló un sencillo programa en Python que permitía guardar las fotografías en una carpeta designada. El programa mostraba la imagen capturada por la cámara y se esperaba a que el usuario presionara la tecla "s" del teclado. Al hacerlo, la imagen se almacenaba en una carpeta llamada "fotos" con un nombre que incluía un número que iba incrementando con el objetivo de identificar y no sobreescribir las

imágenes. Las figuras 19 y 20 muestran dos de las imágenes utilizadas en el proceso de calibración. En estas imágenes, se han señalado las esquinas detectadas por la cámara utilizando la función *drawChessboardCorners* de OpenCV.

Figura 19. Imagen utilizada en la calibración de la cámara

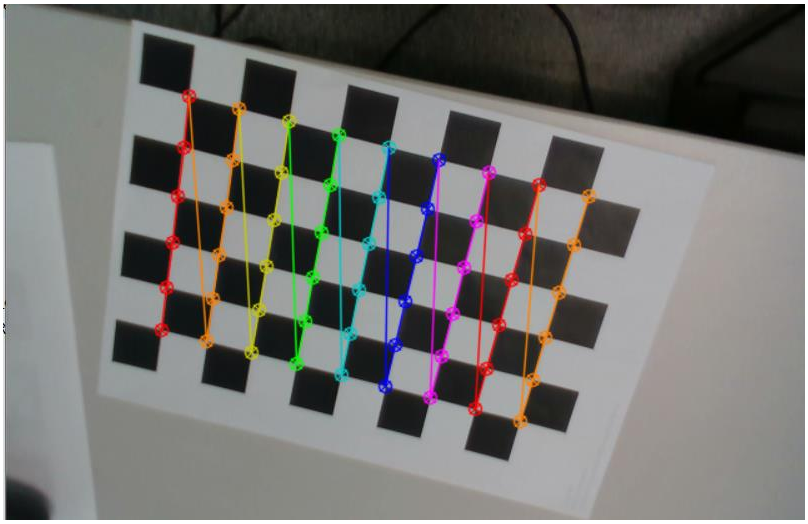

Figura 20. Imagen utilizada en la calibración de la cámara

Una vez que se capturaron suficientes imágenes, se procedió a llevar a cabo la calibración propiamente dicha. Para ello, se disponía de las coordenadas 3D de las esquinas de los cuadrados en el mundo real, conociendo el tamaño de los cuadrados en el patrón. Luego, se obtuvieron las coordenadas en píxeles de cada esquina de los cuadrados en cada una de las fotografías mediante el uso del comando *findChessboardCorners* de OpenCV. Finalmente, se utilizó tanto el conjunto de puntos en 3D como el conjunto de puntos en 2D para realizar la calibración de la cámara, usando la función *calibrateCamera* de OpenCV. Por último, los datos obtenidos de la calibración se guardaron en un archivo con formato *.yaml* para su uso en las aplicaciones posteriores.

La información resultante de la calibración de la cámara proporciona la matriz de la cámara y los coeficientes de distorsión necesarios para corregir cualquier distorsión de la lente de la cámara. A continuación, se detallan estos parámetros en la matriz 1

$$\begin{bmatrix} 574{,}20 & 0 & 338.83 \\ 0 & 574.63 & 241.58 \\ 0 & 0 & 1 \end{bmatrix}$$

Matriz 1. Matriz de la cámara

Estos valores representan la matriz de la cámara calibrada. La distancia focal en el eje x es de aproximadamente 574.20 píxeles, mientras que en el eje y es de 574.63 píxeles. El punto principal, también conocido como el centro de la imagen, se encuentra en el píxel (338.83, 241.58).

Los coeficientes de distorsión que se obtuvieron con este código son los que se pueden ver en la matriz 2:

$$[0.084 \quad 0.059 \quad 0.00077 \quad -0.0014 \quad -0.68]$$

Matriz 2. Coeficientes de distorsión

Estos coeficientes de distorsión se dividen en dos grupos: los coeficientes radiales y los coeficientes tangenciales. La distorsión radial se refiere a la deformación de los objetos en las esquinas de la imagen en comparación con el centro. En este caso, los coeficientes de distorsión radial son 0.084, 0.059 y -0.68. Por otro lado, la distorsión tangencial se refiere al desplazamiento lateral de los objetos en la imagen en relación con el centro. Los coeficientes de distorsión tangencial son 0.00077 y -0.0014.

4.1.2 Transformaciones de sistemas de coordenadas

Como se mencionó previamente, para enviar comandos de movimiento al brazo robot utilizando el comando *MoveJ_IK*, es fundamental que las coordenadas se expresen en función del sistema de coordenadas del brazo robot. El robot UR3 emplea un sistema de coordenadas tridimensional fijo que se encuentra anclado en su base (Figura 21). En este sistema, el eje Z se alinea verticalmente hacia arriba desde el centro de la base del robot, el eje Y apunta hacia la parte trasera de la base móvil y el eje X se sitúa en el plano horizontal, siendo perpendicular a los ejes Y y Z, siguiendo la convención estándar de la mano derecha.

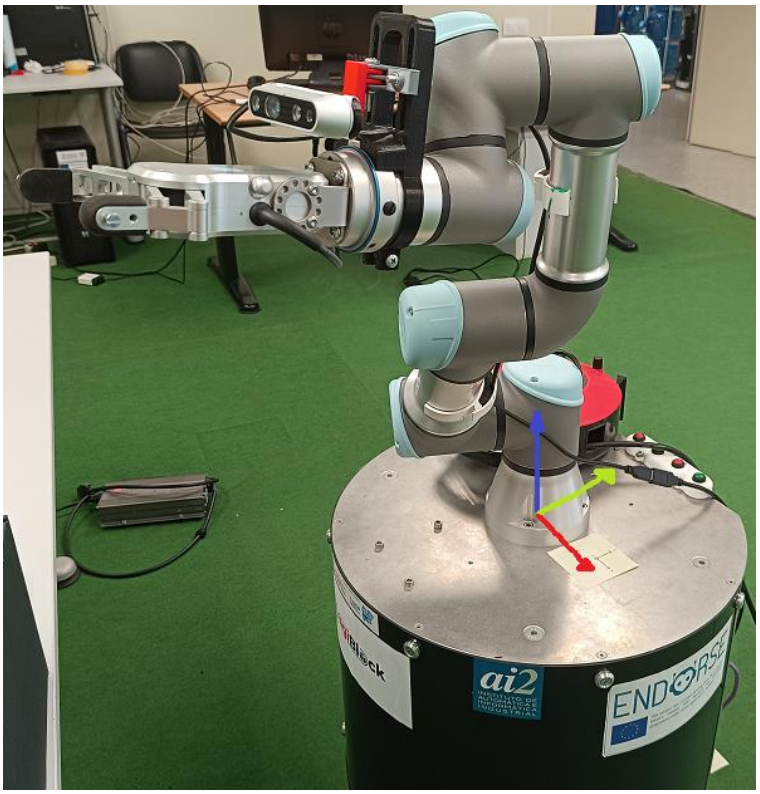

Figura 21. Sistema de coordenadas del robot

Cuando se utiliza el comando MoveJ_IK, se debe proporcionar un vector de 6 elementos que describe tanto la posición como la orientación del brazo robot en el espacio tridimensional. Estos 6 elementos se refieren a las coordenadas en los ejes X, Y y Z, así como a los ángulos de Roll (rotación alrededor del eje Z), Pitch (rotación alrededor del eje Y) y Yaw (rotación alrededor del eje X) en relación con el sistema de coordenadas de referencia del brazo robot.

Al detectar el código AruCo incorporado en el objeto que se debe manipular, utilizando la función *estimatePoseSingleMarkers* de OpenCV, se obtienen dos vectores separados: uno para la traslación (las coordenadas X, Y y Z) y otro para la rotación (los ángulos de Roll, Pitch y Yaw). En esta función, es necesario emplear como argumento las matrices 1 y 2 obtenidas durante el proceso de calibración. Una calibración precisa garantiza una mayor exactitud en los vectores resultantes de esta función, por lo que llevar a cabo una calibración adecuada de la cámara se vuelve esencial para obtener resultados correctos y coherentes en esta etapa del proceso. Sin embargo, estos vectores indican la posición del marcador Aruco en el sistema de coordenadas de la cámara. Por lo tanto, surge la necesidad de transformar estos dos vectores en un solo vector de 6 elementos que pueda representar la posición y la orientación del objeto en el sistema de coordenadas del brazo robot. En la figura 22, se puede observar una imagen obtenida por la cámara, donde se puede ver también el sistema de coordenadas del marcador.

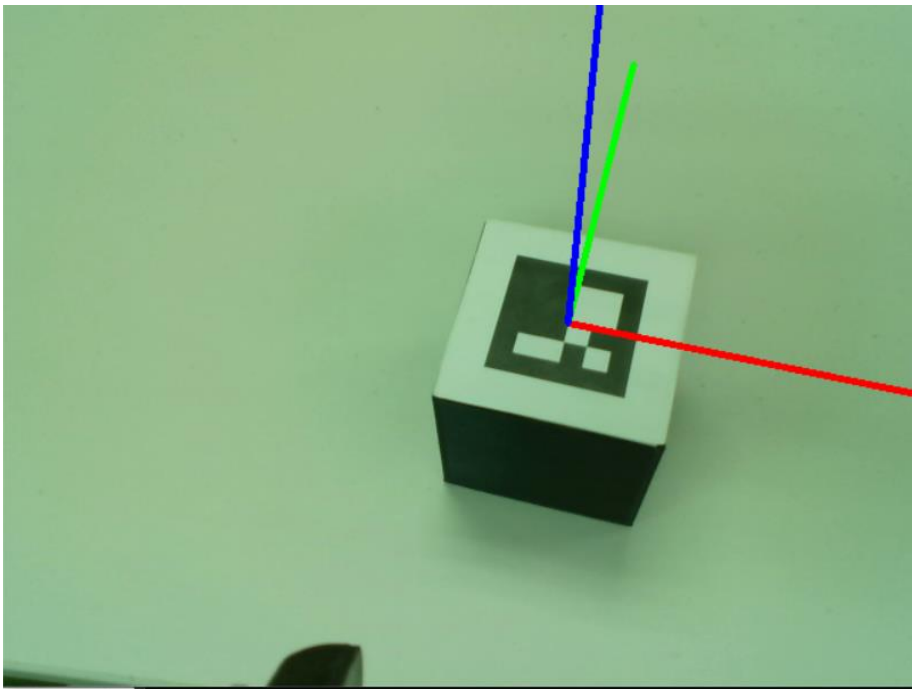

Figura 22. Imagen obtenida de la cámara y sistema de coordenadas del marcador

A continuación, se detallan las transformaciones que se deben llevar a cabo para alcanzar el vector de 6 elementos necesario para controlar de manera precisa el movimiento del brazo robot. Además, se explicará cómo se han implementado estas transformaciones en los programas Python utilizados.

- **Translación en los tres ejes.** Ya se ha mencionado que con la función *estimatePoseSingleMarkers*, podremos obtener los vectores de traslación y rotación del marcador en el sistema de coordenadas de la cámara, pero, una vez conocida la posición del objeto con respecto a la cámara, es necesario conocer su localización con respecto a la herramienta del brazo robot. Para obtener esta posición, es necesario considerar la diferencia en la traslación entre la cámara y la pinza en los ejes X, Y y Z. Dado que la cámara está instalada en la última articulación del brazo robot, ambos sistemas de coordenadas tienen la misma orientación, por lo que no se requiere realizar rotaciones adicionales en los sistemas de coordenadas. Sin embargo, existe una diferencia fija en las posiciones de la cámara y la pinza a lo largo de estos ejes.

  Tras realizar mediciones, se puede concluir que en el eje X existe una diferencia de -3.25 centímetros (negativo en función del sistema de coordenadas de la cámara) entre la cámara y la pinza. En el eje Y, la diferencia es de -5.5 centímetros, y en el eje Z, es de -11.5 centímetros. Por lo tanto, la matriz de transformación utilizada para esta traslación se muestra en la matriz 4 (en metros).

$$\begin{bmatrix} 1 & 0 & 0 & -0.0325 \\ 0 & 1 & 0 & -0.0550 \\ 0 & 0 & 1 & -0.1150 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Matriz 3. Matriz de translación en los tres ejes

Para llevar a cabo esta transformación, es esencial construir una matriz 4x4. Esta matriz se compone de la matriz 3x3 de rotación, obtenida con la función *Rodrigues* de OpenCV, que describe la orientación del marcador en relación con la cámara, y el vector 3x1 de traslación, que representa la posición del marcador con respecto a la cámara. Para completar esta matriz, se agrega una nueva fila de tres ceros y un 1 al final que garantiza que la matriz sea homogénea y esté lista para realizar la transformación necesaria. Posteriormente, se multiplica esta matriz 4x4 por la matriz 3, lo que da como resultado otra matriz 4x4 de la que se pueden extraer los vectores de rotación y traslación que proporcionan la posición del objeto en relación con el sistema de coordenadas de la pinza.

- **Transformación de la pinza a la base mediante cinemática directa**. Por último, es necesario realizar una última transformación que pase del sistema de coordenadas de la pinza al sistema de coordenadas del robot, que es el que se utilizará a la hora de enviar el movimiento a través del comando *MoveJ_IK*.

  Para realizar esta transformación, se emplearán dos funciones facilitadas por la librería *UR_RTDE* (Universal Robots Real-Time Data Exchange) [52] en Python, diseñada para la comunicación en tiempo real con robots Universal Robots (UR).

  La primera etapa de esta transformación implica obtener la posición y orientación actual de la herramienta del robot en el espacio de trabajo. Para ello, empleamos la función *getForwardKinematics*. Esta función nos brinda la ubicación precisa de la herramienta en el sistema de coordenadas del robot. Esta ubicación incluye tanto la posición como la orientación relativa de la herramienta.

  Posteriormente, para lograr la transición del sistema de coordenadas de la herramienta al sistema de coordenadas del robot, utilizamos la función *poseTrans*. Esta función nos permite llevar a cabo una transformación específica en una posición dada. En este caso, utilizamos los vectores de traslación y rotación que se obtuvieron en la transformación previa (vectores que indican la posición del objeto en el sistema de coordenadas de la pinza). Aplicamos estos vectores como la transformación para la posición actual de la herramienta, que previamente obtuvimos mediante la función *getForwardKinematics*. Al aplicar la función *poseTrans* con estos parámetros, obtenemos la ubicación precisa que la herramienta debe alcanzar en el sistema de coordenadas del robot para poder manipular el objeto con precisión.

Una vez completadas las transformaciones mencionadas, obtendremos un vector de 6 elementos que proporciona información tanto sobre la posición como sobre la orientación del brazo robot en el espacio tridimensional. Este vector tiene las características necesarias para poder controlar el brazo robot utilizando el comando moveJ_IK. Si la calibración se ha realizado con precisión y las transformaciones se han llevado a cabo de manera correcta, la pinza del robot finalizará en la posición exacta necesaria para realizar operaciones de agarre y manipulación del objeto con total eficacia.

4.2 Implementación del control de agarre

Una vez superado el paso anterior, donde se ha conseguido detectar correctamente la posición del objetivo y se han realizado las transformaciones necesarias para enviar correctamente el

movimiento al brazo robot, se propone realizar la implementación del control del agarre y manipulación de los objetos. Para avanzar en ello, se adoptó un enfoque progresivo y estructurado. Se abordaron problemas de menor complejidad inicialmente, y a medida que se resolvían, se incorporaban nuevas funcionalidades al programa existente. El proceso se dividió en varias etapas, cada una enfocada en resolver desafíos específicos.

En primer lugar, en la sección 4.2.1 se explica cómo abordar el agarre de objetos estáticos con la pinza en una orientación fija que coincide con la del objeto. Luego, en la sección 4.2.2 se detalla cómo resolver el problema cuando la orientación de la pinza o del objeto no coincide utilizando dos enfoques diferentes: mediante varios movimientos (sección 4.2.2.1) y mediante la incorporación de transformaciones adicionales (sección 4.2.2.2). A continuación, se aborda el escenario en el que el objeto está en movimiento, como se describe en la sección 4.2.3, y finalmente, en la sección 4.2.4, se explica cómo resolver el problema cuando el brazo robot no puede alcanzar el objeto, lo que requiere ajustar la posición de la base móvil. En la figura 23 se presenta un esquema que ilustra las diferentes etapas del proceso, destacando cómo la realización exitosa de cada una de estas etapas es fundamental para alcanzar el objetivo final.

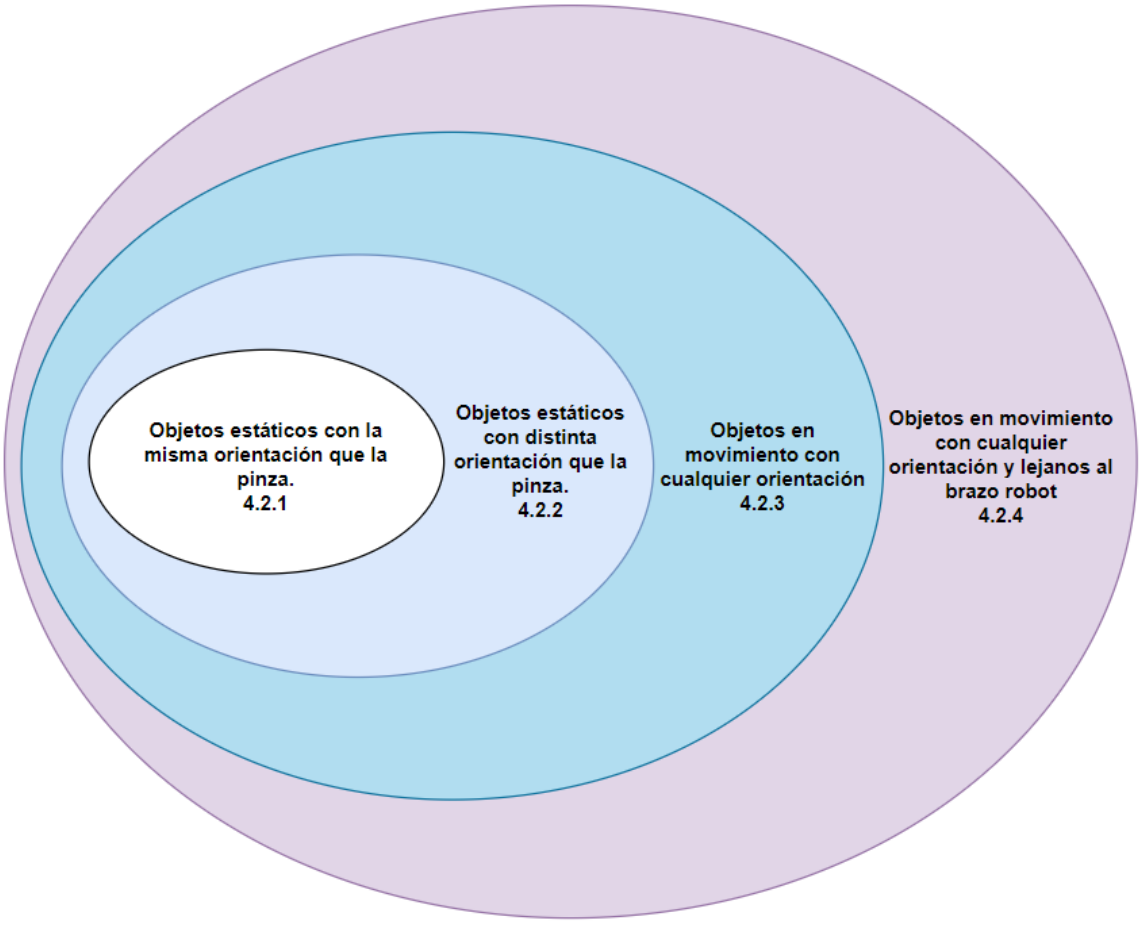

Figura 23. Etapas del desarrollo del trabajo

A lo largo de estas implementaciones, se fueron agregando capacidades al programa existente. A continuación, se detallará cómo se llevaron a cabo estas implementaciones, destacando las adiciones específicas a medida que se enfrentaban a desafíos más complejos.

4.2.1 Objetos estáticos con la misma orientación que la pinza.

Esta sección sienta las bases para todas las implementaciones posteriores, ya que representa el caso más simple de abordar. Aquí, se coloca la pinza del brazo robot de tal manera que no es necesario cambiar su orientación para lograr un agarre efectivo del objeto, es decir, la acción principal requerida en este caso es una traslación. Se puede ver la configuración mencionada en la figura 24.

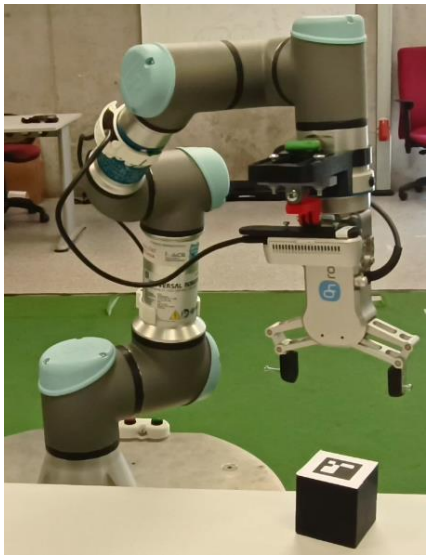

Figura 24. Objeto estático y misma orientación que la pinza

Para comprender la implementación completa del control de agarre, primero se describirán dos funciones fundamentales que se llevaron a cabo para facilitar y acelerar las posteriores implementaciones: *find_Aruco()* y *tag2grap()*.

- ***find_Aruco***. Esta función es una parte crucial de la implementación del control de agarre y manipulación de objetos. Su objetivo principal es detectar el código AruCo y devolver los vectores de traslación y rotación asociados. Sin embargo, si no se encuentra el código AruCo, comienza una búsqueda activa moviendo el brazo robot en busca del código.

  El código mencionado se ejecuta de la siguiente manera: primero, espera a que lleguen imagenes de la cámara utilizando la función *wait_for_frames* de la librería *pyrealsense2.* Luego, detecta los códigos AruCo presentes en la imagen mediante la funcion *detectMarkers* y se obtienen los vectores de traslación y rotación con la función *estimatePoseSingleMarkers*. Estos vectores son los que devuelve la función como salida.

  Si en algún momento el programa arroja un error debido a la falta del código AruCo, se inicia el proceso de búsqueda. La búsqueda implica mover el brazo robot en incrementos de un centímetro en los tres ejes espaciales, siguiendo la forma de un cubo cada vez de mayor tamaño. El programa sigue realizando estos movimientos hasta que encuentra el código AruCo, momento en el cual retoma la detección del objeto y obtención de los vectores. Este mecanismo de búsqueda garantiza que, en caso de no encontrar el código AruCo de inmediato, el brazo robot realizará movimientos sistemáticos para localizarlo.

  Para llevar a cabo este proceso de búsqueda, se han empleado dos variables auxiliares: "i" y "j". La variable "i" se utiliza para indicar la magnitud del desplazamiento en centímetros,

mientras que la variable "j" determina el eje en el que se realizará el movimiento. La variable "j" puede tomar los valores 0, 1 y 2, que representan los tres ejes del espacio tridimensional. A medida que avanza el proceso de búsqueda, la variable "j" aumenta su valor en 1, progresando de 0 a 1, luego a 2, y finalmente volviendo a 0. Cuando esto ocurre, el signo de la variable "i" se invierte y aumenta su valor absoluto en 1 centímetro. De esta manera, se logra un desplazamiento uniforme en los tres ejes, seguido de un desplazamiento adicional de 1 centímetro en sentido opuesto en cada uno de los ejes hasta que se encuentre el código AruCo.

Esta función es fundamental para la implementación global del control de agarre, ya que es la base para la detección precisa del objeto que se va a manipular. En la figura 25 se presenta un diagrama de bloques que ilustra de manera sintética y esclarecedora el funcionamiento de la función previamente descrita.

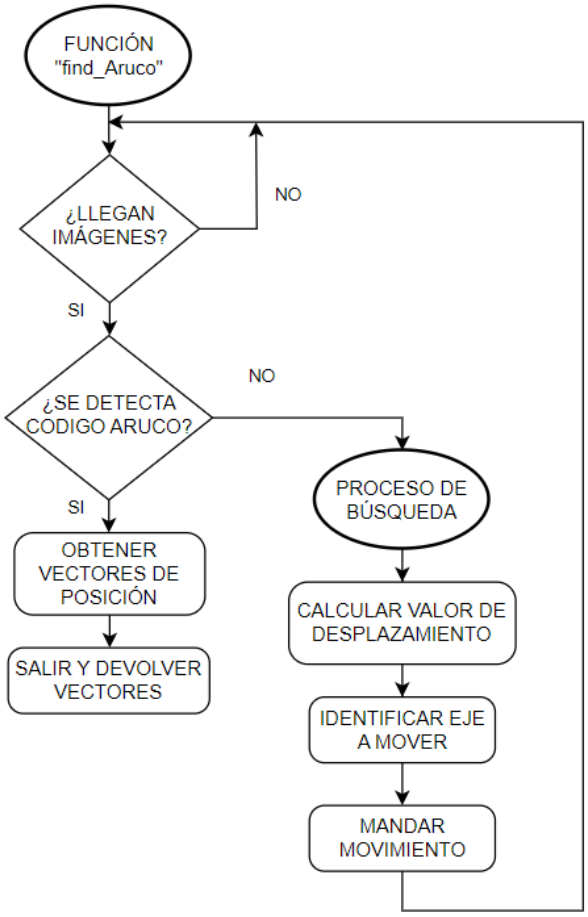

Figura 25. Función "find_Aruco"

- ***tag2grap***. La función *tag2grap* también desempeña un papel crucial en la implementación del control de agarre y manipulación de objetos. Su objetivo principal es realizar las transformaciones necesarias ya mencionadas para cambiar del sistema de coordenadas de la cámara al sistema de coordenadas de la pinza del brazo robot.

  Esta función toma como entrada los vectores de rotación y traslación que se obtienen de la función *find_Aruco*, los cuales determinan la posición del código AruCo en el sistema de

coordenadas de la cámara. Luego, realiza traslaciones en los tres ejes, como se explicó en el apartado anterior.

Una vez completada esta transformación, se obtienen los vectores de rotación y traslación del marcador AruCo con respecto al sistema de coordenadas de la pinza del brazo robot. Esta función desempeña un papel fundamental en el proceso, ya que garantiza que la información de posición y orientación capturada por la cámara se pueda traducir posteriormente con precisión en comandos de movimiento para el brazo robot.

Cabe destacar que en esta función no se ha incluido la última transformación para pasar al sistema de coordenadas del robot por motivos que se explicarán en la siguiente sección.

En la figura 26, se muestra un diagrama de bloques que proporciona una representación visual del proceso descrito, lo que simplifica la comprensión de la función *tag2grap*. Además, se ha indicado a la derecha con una flecha la matriz o el vector que resulta de la operación correspondiente, con el fin de aclarar la secuencia seguida por la función.

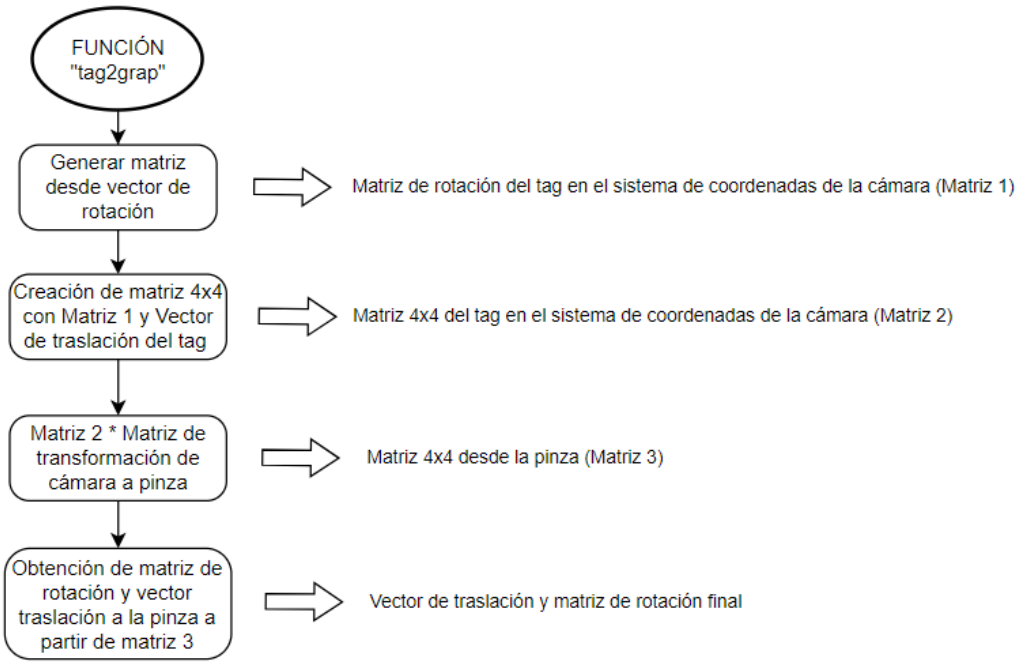

Figura 26. Función *tag2grap*

Una vez detallada estas dos funciones que constituyen la base fundamental de la implementación del control de agarre en todas sus variantes, ahora se puede avanzar para explicar el siguiente paso en la implementación del primer caso de agarre, que se refiere a la situación en la que tanto la pinza como el objeto tienen la misma orientación.

En este caso, que es el más sencillo de todos, no se requieren muchos ajustes adicionales más allá de los que ya se han explicado en las dos funciones fundamentales. Una vez que se han aplicado estas funciones, se obtienen los vectores de rotación y traslación del objeto en relación con el sistema de coordenadas de la pinza. Por lo tanto, los siguientes pasos son realizar el movimiento de traslación para llegar hasta el objeto y cerrar la pinza para agarrarlo.

Para lograr esto, se lleva a cabo la última transformación siguiendo los pasos explicados anteriormente, que proporciona la posición del objeto en el sistema de coordenadas del robot, y se ejecuta el comando de movimiento utilizando moveJ_IK. Sin embargo, este movimiento se divide en dos pasos. En primer lugar, se realiza un movimiento en los ejes X e Y de la pinza, y una vez completado, se realiza el movimiento en el eje Z. Esta división se implementa para evitar que la pinza intente atravesar el objeto en su camino hacia la posición deseada, ya que primero se colocará en la posición vertical del objeto y posteriormente descenderá para agarrar el objeto, evitando cualquier colisión con el mismo.

Una vez que la pinza del brazo robot se ha posicionado en la ubicación adecuada para el agarre, solo queda enviar un comando al brazo para que cierre la pinza que está incorporada en su extremo. Esto se logra utilizando la función *setToolDigitalOut* de la librería UR_RTDE, estableciendo el argumento en 1 para indicar que la pinza debe cerrarse, mientras que configurándolo en 0 se indicaría que la pinza debe abrirse.

La figura 27 presenta un diagrama de bloques que detalla la implementación del control de agarre para objetos estáticos cuando la orientación de la pinza coincide con la del objetivo. Esto proporciona una visión general de cómo se estructura la implementación en términos de procesos y funciones clave.

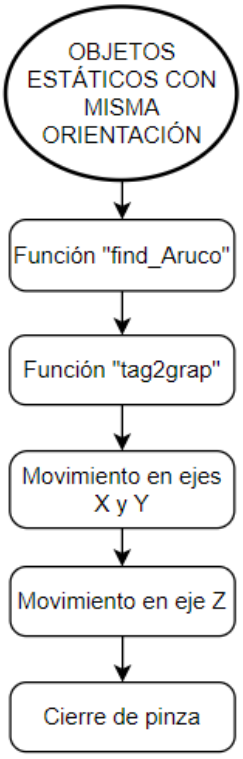

Figura 27. Diagrama de bloques objetos estáticos con misma orientación

4.2.2 Objetos estáticos con distinta orientación que la pinza

En esta nueva implementación, el objeto vuelve a permanecer estático, pero en este caso, la pinza se coloca de tal manera que no tiene la misma orientación que el objeto. Este cambio de orientación puede ocurrir en cualquiera de los tres ejes. Por lo tanto, se requiere realizar una traslación y una

rotación de la pinza para lograr un agarre efectivo del objeto. Un ejemplo de esta configuración se puede observar en la figura 28.

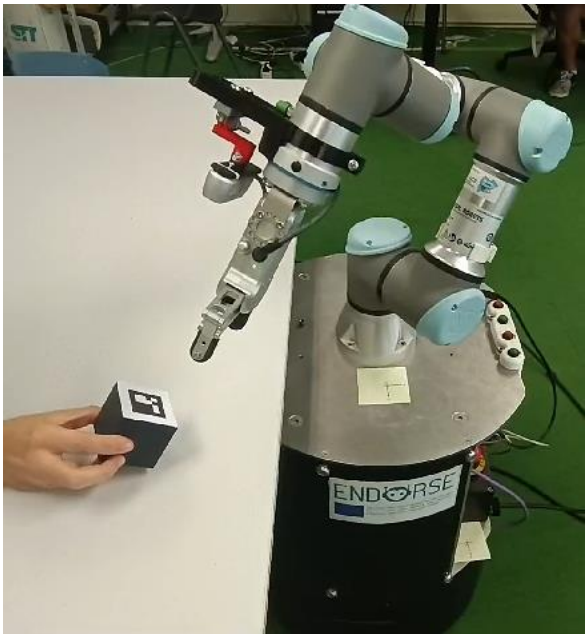

Figura 28. Objeto estático y distinta orientación que la pinza

El cambio de orientación en la pinza ocasionaba un problema significativo en los vectores de traslación obtenidos a través de la biblioteca OpenCV. Para ilustrar mejor este problema, se utilizará la figura 29 como ejemplo del error que se producía.

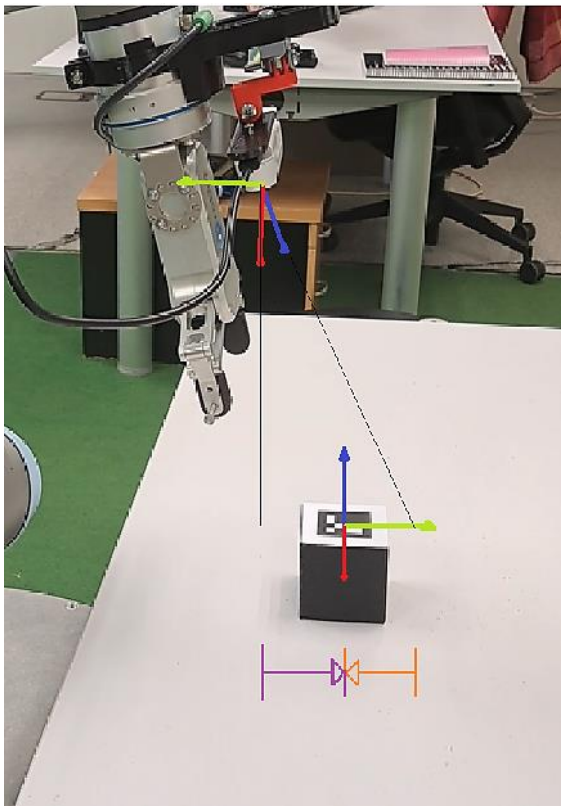

Figura 29. Error en los vectores de traslación

Una vez obtenidos los vectores de traslación y rotación del objeto en función del sistema de coordenadas de la cámara, el vector de rotación era correcto, pero se observó un problema en el vector de traslación. A medida que la diferencia de orientación entre el objeto y la pinza aumentaba, este vector arrojaba errores cada vez mayores.

Para ilustrar este problema, consideremos el ejemplo de la figura anterior. Cuando la cámara esté orientada verticalmente (como debe acabar el movimiento para realizar un agarre correcto del objeto), la distancia que se debe mover la cámara para posicionarse justo encima del código AruCo en el eje Y (verde) es la que se puede ver en color morado y en sentido negativo del eje. Sin embargo, debido a la rotación entre el sistema de coordenadas de la cámara y el del objeto, la distancia que devolvía en ese mismo eje la librería OpenCV es la que se ve en color naranja. Como se puede observar, la distancia calculada por la librería no solo tiene una magnitud diferente, si no que también es del sentido opuesto al que se debería mover.

Este error se agrava a medida que la orientación de la pinza se aleja más de la del objeto, lo que impide la utilización de la implementación previamente explicada para casos en los que la orientación difiere entre el objeto y la pinza. En la mayoría de estos casos, el error resultante provoca una colocación incorrecta o incluso movimientos en la dirección opuesta a la necesaria.

Dado este problema, se requiere una adaptación en el código inicial para abordar este desafío y garantizar que el sistema funcione correctamente en situaciones donde la orientación difiera entre el objeto y la pinza. A continuación, se explicarán dos soluciones distintas que abordan este problema de maneras diferentes.

4.2.2.1 Implementación con dos capturas de imagen

La primera solución para abordar este problema se basa en la observación de que el error en el vector de traslación solo ocurre cuando la pinza y el objeto no tienen la misma orientación. Por lo tanto, una solución posible sería realizar el movimiento de agarre en dos etapas, es decir, capturar dos imágenes en lugar de una, como se hacía anteriormente.

En primer lugar, se captura la primera imagen para obtener los vectores de traslación y rotación. Sin embargo, en esta etapa, solo se realiza el movimiento de rotación para alinear la orientación de la pinza con la del objeto. Luego, se captura una segunda imagen una vez que la pinza está orientada correctamente, y se realiza el movimiento de traslación, siguiendo el mismo proceso que en el caso anterior: primero en los ejes X e Y, y luego en el eje Z.

Sin embargo, esta solución plantea un posible problema: si la pinza tiene una inclinación significativa, al orientarse correctamente, el objeto podría quedar fuera del campo de visión de la cámara. Para abordar este desafío, se podría considerar realizar un movimiento en traslación de una parte del vector de traslación obtenido por la librería OpenCV, es decir, por ejemplo, la mitad del desplazamiento en cada eje. No obstante, esta solución puede no ser adecuada en casos como el ejemplo de la figura 29, donde la traslación adicional alejaría aún más el brazo robot del objeto y podría provocar la pérdida del mismo en la imagen.

Por lo tanto, la solución adoptada es la siguiente: una vez que la pinza está orientada de manera adecuada, sin haber hecho ningún movimiento de traslación, se captura nuevamente la imagen para obtener los vectores de rotación y traslación. En el caso en el que, al realizar el giro para alinear la pinza con el objeto, la cámara deje de captar el objeto, se continúa con el proceso de búsqueda que

se ha explicado en el apartado anterior. Una vez se localice nuevamente el objeto, la orientación de la pinza será correcta, lo que permite aplicar el movimiento según los vectores de traslación para colocarse en la posición ideal para el agarre, seguido por el cierre de la pinza.

En cuanto al código, debido al uso de las funciones *find_Aruco* y *tag2grap*, es relativamente sencillo implementar esta solución, ya que incorporan la búsqueda del objeto en la captura de la imagen. Únicamente es necesario agregar una captura de imagen adicional al inicio y un movimiento siguiendo únicamente el vector de rotación. En la figura 30, se presenta un diagrama de bloques que muestra cómo se estructura esta implementación. Los elementos en color rojo indican lo que se ha agregado en esta versión, mientras que los elementos en color negro ya estaban implementados en el caso anterior donde la pinza tenía la misma orientación que el objeto.

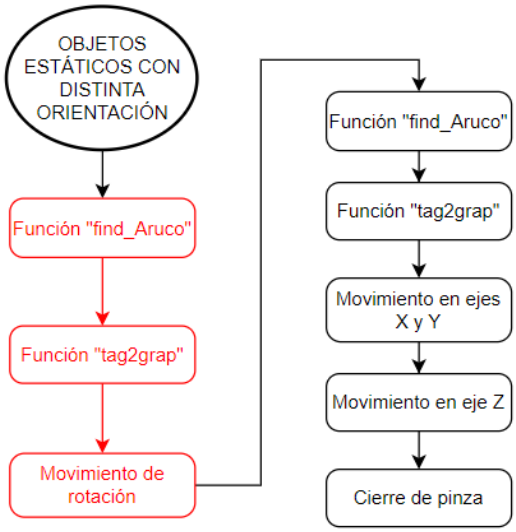

Figura 30. Diagrama de bloques objetos estáticos con distinta orientación

4.2.2.2 Corrección de vector de traslación

La segunda solución diseñada para abordar este problema se centra en corregir el vector de traslación generado por la librería OpenCV, que es el que presenta errores en estos casos. Además, esta solución implica realizar el movimiento necesario capturando solo una imagen, a diferencia del enfoque anterior donde se requerían dos imágenes.

La forma de corregir este vector de traslación es añadir dos nuevas transformaciones anteriores a las explicadas en el apartado 4.1.2, aplicadas en la función *tag2grap*. El objetivo de estas nuevas transformaciones es conseguir un vector que nos de la posición de la cámara con respecto al tag, es decir, en el sistema de referencia que se observa en la figura 22 y, posteriormente, realizar un giro para ajustar los sentidos de los vectores. De esta forma, el vector de traslación que se obtenga, tendrá la orientación adecuada, es decir, la misma que la del objeto.

Las trasformaciones que se llevan a cabo en este caso son las que se explican a continuación:

- **Inversa de la matriz de rotación.** Para obtener los vectores de traslación en función del sistema de referencia del código AruCo, es necesario llevar a cabo la siguiente operación:

$$t\_vec_{tag\_cam} = -R^{-1}_{cam\_tag} * t\_vec_{cam\_tag}$$

Siendo $t\_vec_{tag\_cam}$ el vector de traslación del tag a la cámara, R_{cam_tag} la matriz de rotación de la cámara al marcador, y $t\_vec_{cam\_tag}$ el vector de traslación de la cámara al tag (el obtenido directamente por la cámara).

Por lo tanto, para llevar a cabo esta transformación, se realiza la inversa de la matriz de rotación del AruCo en función del sistema de coordenadas de la cámara y, posteriormente, se multiplica por el vector de traslación en sentido negativo.

Una vez realizada esta transformación, obtendremos un vector de traslación que nos indicará la posición de la cámara en el sistema de coordenadas del marcador AruCo.

- **Giro de 180 grados en el eje x**. En términos prácticos, esta transformación significa invertir los signos de los elementos de traslación en Y y Z. En un ejemplo visual (como se muestra en la figura 22), el sentido positivo de la coordenada Y (representada por la línea verde) cambiaría de derecha a izquierda, mientras que el sentido positivo de coordenada Z (representada por la línea azul) cambiaría de arriba a abajo. Importante destacar que el eje X permanece sin cambios en este proceso de transformación.

  Para realizar esta transformación se utilizó una matriz de rotación siguiendo las especificaciones indicadas en la figura 31, que muestra las matrices de rotación principales alrededor de los ejes cartesianos X, Y y Z.

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Figura 31. Matrices de rotación principales

Por ello, la matriz que se utiliza para realizar esta transformación se corresponde con la matriz 4, que se define como:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

Matriz 4. Matriz de rotación de 180 grados en x

Esta matriz es la encargada de realizar una rotación de 180 grados en el eje X. Para aplicar la rotación, se multiplican tanto la matriz de rotación como el vector de traslación que indican la posición de la cámara en el sistema de referencia del tag por la matriz 4.

Finalmente, una vez incorporadas estas dos nuevas transformaciones, el vector de traslación obtenido tras aplicar todas las transformaciones, indicaba el desplazamiento real que debía hacer el robot para colocarse en la posición de agarre, eliminando los errores debido a la diferencia de orientación entre la pinza y el objeto.

En la figura 32, se puede observar un diagrama de la función *tag2grap* tras incorporar estas nuevas transformaciones, marcando en rojo las novedades en esta implementación.

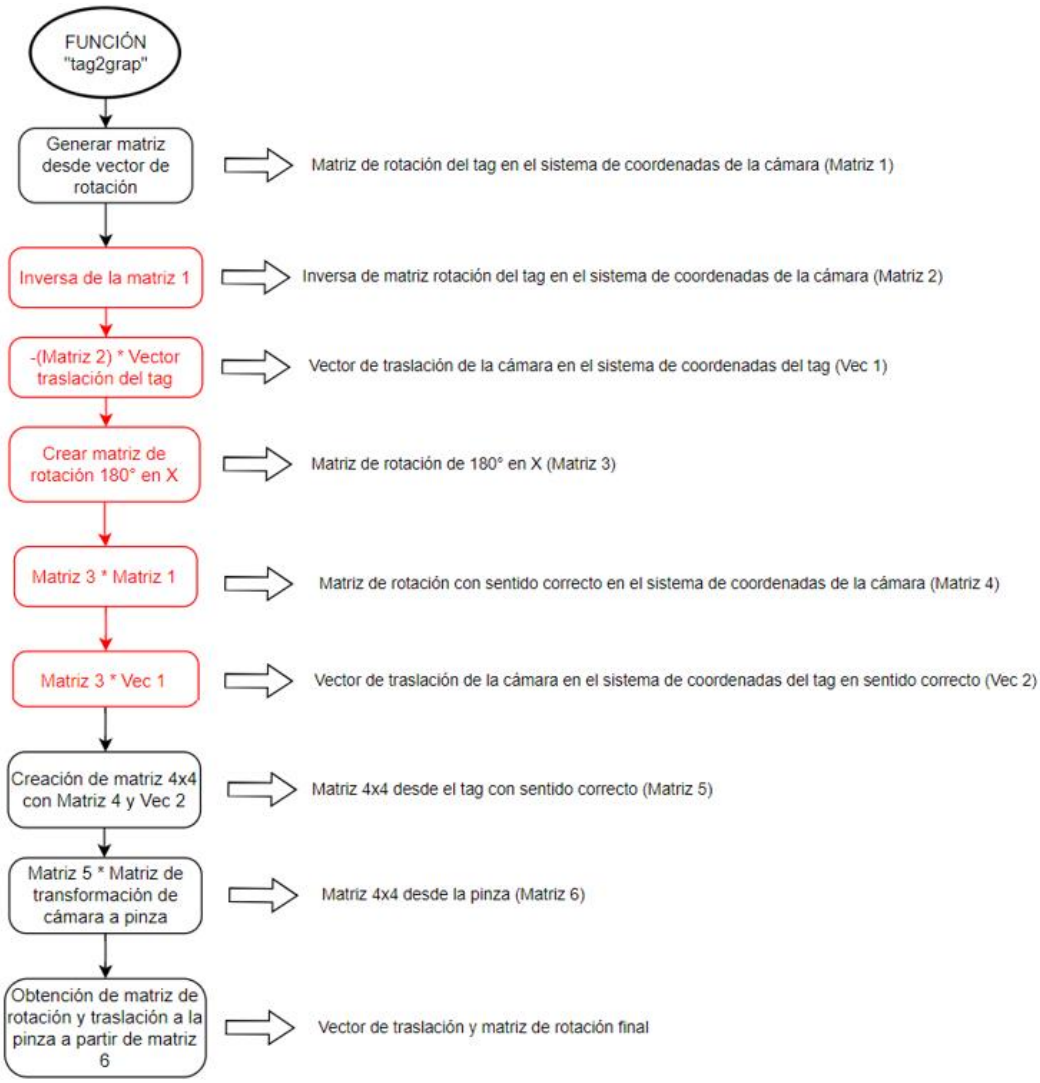

Figura 32. Diagrama de bloques para función *tag2grap*

4.2.2.3 Implementación final para objetos estáticos

Una vez explicadas las dos posibles soluciones para abordar el problema mencionado anteriormente, fue necesario tomar una decisión sobre cuál de ellas podría optimizar al máximo la tarea de manipulación de objetos.

Cada una de las dos soluciones tiene ventajas y desventajas. La primera solución, que involucraba dos capturas de imagen, podría ralentizar el proceso por varias razones. Primero, se requería obtener los vectores en dos ocasiones, lo que significaba aplicar las transformaciones dos veces y enviar más comandos de movimiento al brazo robot. En segundo lugar, una vez que la pinza estaba

orientada correctamente, era probable que el objeto saliera del rango de visión de la cámara, lo que requería iniciar la tarea de búsqueda nuevamente, lo que llevaría tiempo.

Sin embargo, la primera implementación tiene una ventaja crucial: al tomar la segunda imagen con la misma orientación que el objeto, se minimizan los errores de distorsión causados por posibles errores de calibración de la cámara. Por ello, el robot siempre terminaría en la posición precisa para agarrar correctamente el objeto.

La segunda solución, que implicaba la corrección del vector de traslación después de una sola captura de imagen, era más rápida, ya que solo se necesitaba una imagen para obtener todos los datos necesarios para los movimientos. Sin embargo, tenía la desventaja de ser sensible a inclinaciones extremas de la cámara con respecto al objeto, lo que podría generar errores en las medidas y, por lo tanto, posicionar incorrectamente el brazo robot.

Dada esta evaluación, se optó por una tercera implementación que combinaba elementos de ambas soluciones. En esta implementación final, se capturaban dos imágenes, al igual que en la primera solución. Sin embargo, a medida que el robot se orientaba después de la primera imagen, también realizaba un movimiento en la dirección del vector de traslación corregido, como se hacía en la segunda solución. La magnitud de este movimiento se estableció en un quinto de la distancia calculada en el vector de traslación corregido, de esta forma, el brazo se acercaría siempre al objeto, evitando errores de mediciones.

Esta solución híbrida permitió aprovechar al máximo las ventajas de ambas opciones y minimizar sus desventajas, mejorando así la eficiencia y la precisión del sistema. Esta implementación lograba la precisión de la primera solución al tomar una segunda imagen con la pinza ya orientada y, además, evitaba en muchos casos la necesidad de realizar una segunda búsqueda del objeto, ya que el brazo se acercaba a él durante la orientación.

El diagrama de bloques de esta implementación final para el control de agarre de un objeto estático cuando no tiene la misma orientación que la pinza se muestra en la figura 33.

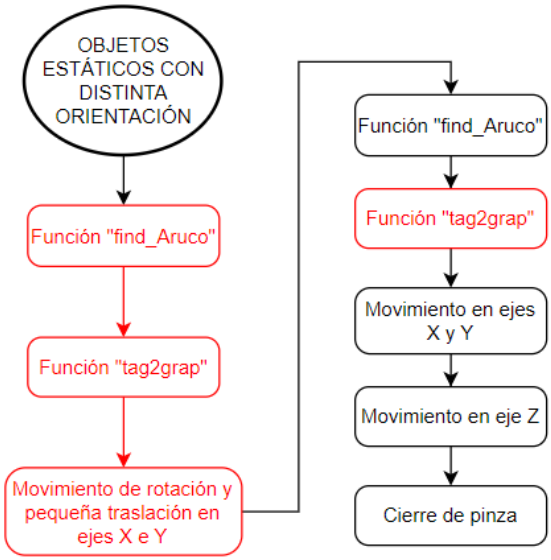

Figura 33. Diagrama de bloques implementación final objetos estáticos

En el diagrama de bloques anterior, se ha mantenido en rojo las diferencias con la implementación cuando el objeto tiene la misma orientación que la pinza (Figura 27). Se ha indicado en rojo la función *tag2grap* ya que se utiliza la que corrige el vector de traslación (Figura 32).

4.2.3 Control de agarre para objetos en movimiento

En esta nueva implementación, el objeto no permanecerá estático, si no que estará en movimiento, cambiando tanto de posición como de orientación. Además, estará lo suficientemente cerca del robot como para que no sea necesario mover el robot móvil. Por lo tanto, es crucial que las implementaciones explicadas anteriormente funcionen correctamente. En la figura 34, se muestra un ejemplo de una configuración de este tipo..

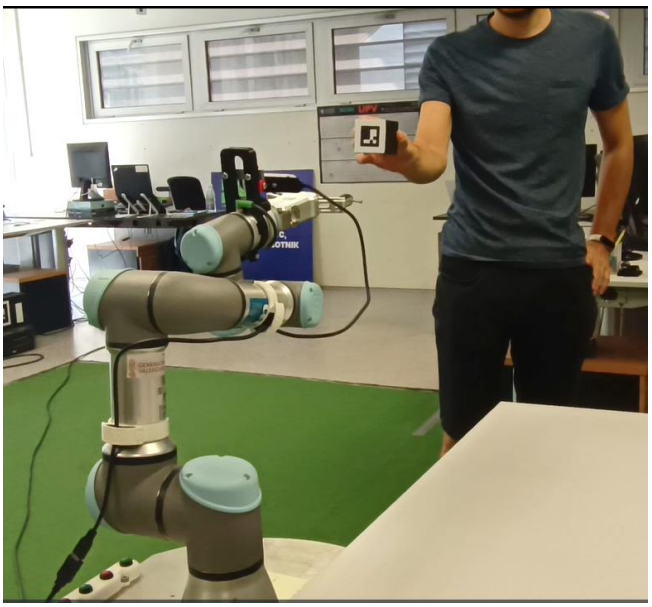

Figura 34. Situación inicial para agarre de objetos en movimiento

La estrategia adoptada para abordar esta implementación es la que se detalla a continuación. La cámara del brazo robot seguirá al objeto hasta que este se detenga. Una vez que el objeto esté inmóvil, se procederá a agarrarlo siguiendo las implementaciones previamente explicadas. En la figura anterior, se puede ver que se está sujetando el objeto con la intención de moverlo durante un tiempo y, después de un período, se dejará inmóvil para que el robot lo alcance.

La implementación de esta estrategia requiere modificaciones en el código de control del agarre. El nuevo código debe abordar dos aspectos adicionales.

Primero, debe ser capaz de seguir al cubo en movimiento, alineando la orientación y la posición de la cámara con la del código AruCo. La razón para utilizar la cámara para el seguimiento del objetivo es porque, si el seguimiento se realizase con la garra, existiría la posibilidad de que en algún momento el cubo saliese de la visión de la cámara. En cambio, al seguirlo con la cámara, se garantiza que siempre estará dentro de su campo de visión.

En segundo lugar, el código debe detectar cuándo el cubo permanece inmóvil, ya que será en este momento cuando realizará las tareas de agarre, siguiendo los mismos procedimientos que se han explicado anteriormente para el cubo en estado estático.

Para abordar el primer aspecto, que implica el seguimiento del cubo por parte de la cámara, se implementará un bucle en el que se enviarán continuamente comandos de movimiento al brazo robot. Estos movimientos consistirán en traslaciones en los ejes X e Y, así como rotaciones en los tres ejes. La razón detrás de no incluir traslaciones en el eje Z es que el seguimiento de la pieza es óptimo si se hace en un mismo plano, tratando de no alejarse ni acercarse al objeto.

Además, los movimientos de traslación en los ejes X e Y serán de una quinta parte de la magnitud del vector obtenido desde la cámara. Esto se debe a que es necesario esperar a que el movimiento actual se complete antes de enviar uno nuevo, por lo que realizar movimientos más cortos en dirección al objeto permite realizar un seguimiento más rápido y fluido del mismo.

Este bucle de seguimiento de la pieza se ejecutará hasta que se detecte que la pieza se ha quedado inmóvil. Para lograrlo, se almacena el vector de traslación obtenido de la cámara en cada instante y se compara con el vector anterior. Cuando dos vectores de traslación sean iguales (permitiendo un pequeño margen) en dos instantes consecutivos, se considerará que el objeto está estático. En ese momento, se saldrá del bucle y se procederá con el agarre.

En la figura 35 se presenta un diagrama de bloques que ilustra esta nueva implementación. Los elementos nuevos en comparación con la implementación del control de agarre para un objeto estático con distinta orientación a la pinza se han resaltado en rojo.

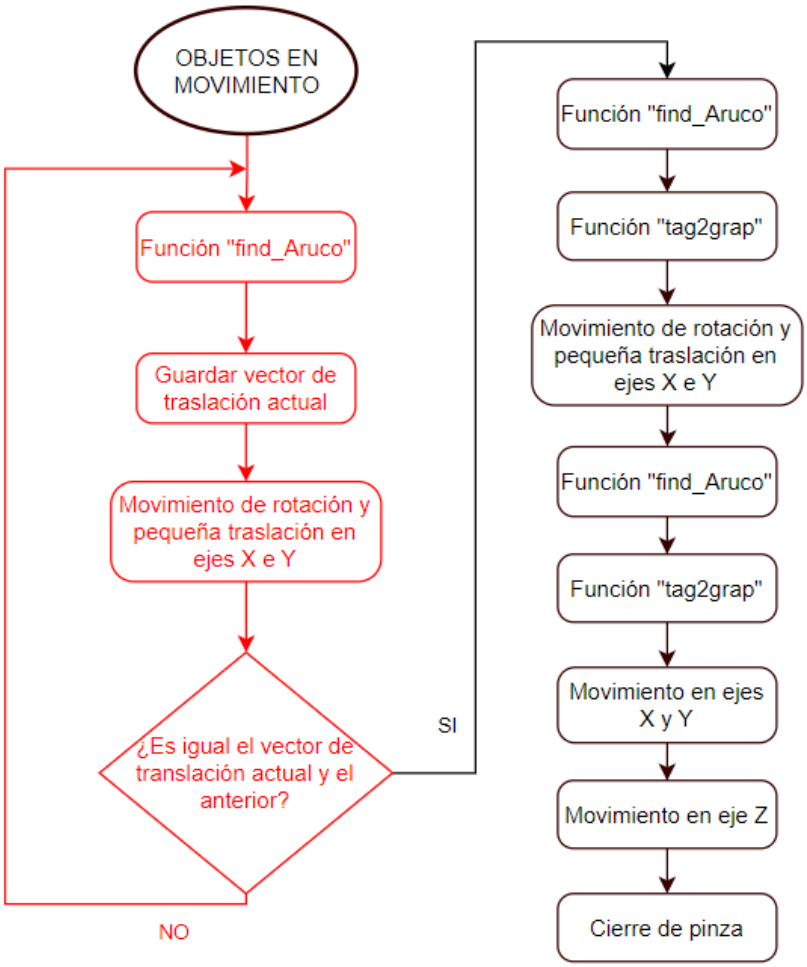

Figura 35. Diagrama de bloques objetos en movimiento

La sección resaltada en rojo, en su conjunto, es la que permite al control realizar el seguimiento de los objetos mientras estos no se encuentran en una posición estática. Es importante mencionar que en esta parte del proceso no se utiliza la función tag2grap, ya que los vectores utilizados para mover el brazo robot se basan en las coordenadas de la cámara como se ha mencionado con anterioridad.

4.2.4 Control de agarre con movimiento de la base

Esta es la última implementación y la que completa el sistema integral para realizar las tareas de logística mencionadas anteriormente. En esta implementación, el cubo objetivo puede estar en estado estático o en movimiento, pero, además, puede encontrarse a una distancia tal que el brazo robot no pueda alcanzarlo debido a limitaciones físicas, como singularidades o una longitud insuficiente del brazo. La figura 36 muestra el inicio del movimiento para una implementación de este tipo.

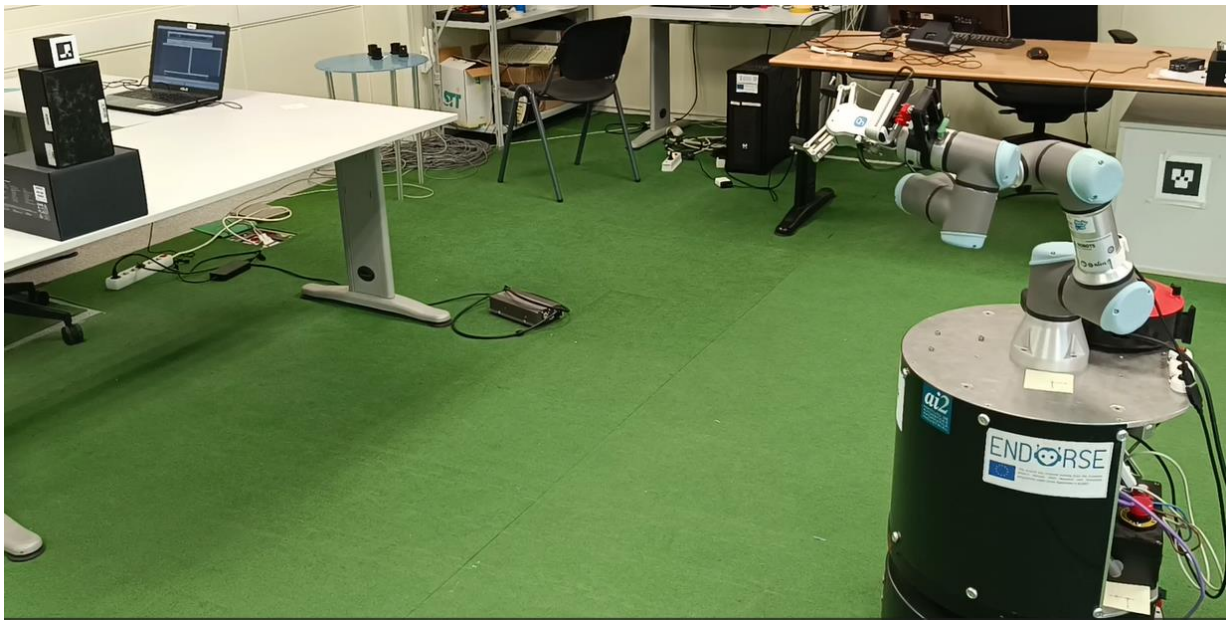

Figura 36. Inicio de movimiento para implementación con movimiento de la base

La estrategia empleada en esta implementación se detalla a continuación. Si el cubo no se encuentra en el rango de visión de la cámara, el brazo robot deberá llevar a cabo la tarea de búsqueda como se realiza normalmente. Sin embargo, una vez que esté dentro del rango de visión, se evaluará si la distancia entre el objeto y el brazo robot permite un agarre efectivo sin restricciones físicas, como singularidades o una longitud insuficiente del brazo.

Para esta nueva implementación, se modificará el bucle de seguimiento de la pieza descrito en el apartado anterior. Ahora, se esperará a que el brazo robot esté lo suficientemente cerca del objeto para evitar limitaciones físicas antes de realizar el agarre. Es decir, después de usar la función *find_Aruco* y obtener los vectores de traslación y rotación del objeto con respecto a la cámara, se evaluará la distancia en el eje Z hasta el objeto.

Si la distancia en el eje Z es menor de 35 centímetros desde la cámara, se considerará que el brazo robot está lo suficientemente cerca para realizar la tarea de agarre sin problemas. Este valor de la distancia se eligió de forma empírica, ya que, para valores mayores de esta distancia, el robot alcanzaba una singularidad en la mayoría de las ocasiones. En los casos donde la distancia fuese

menor al valor especificado, el funcionamiento del sistema es el mismo que se ha descrito hasta esta sección. Sin embargo, si la distancia en este eje supera los 35 centímetros, se determinará que el objeto se encuentra a una distancia que requerirá que el brazo robot se acerque antes de realizar el agarre.

Cuando se considere que el objeto está demasiado lejos, se enviará un comando de movimiento a la base móvil para que avance hacia el objeto hasta situarse a una distancia aproximada de 20 centímetros de este. Esta distancia permitirá que el brazo robot pueda agarrar el objeto de manera efectiva.

El comando de movimiento de la base móvil se envía cada vez que se detecta la posición del objeto. Esto permite corregir posibles errores en la medición debido a la distancia entre la cámara y el código Aruco. A medida que el brazo robot se acerca, es probable que este error disminuya, y al enviar comandos de movimiento de forma continua, se estaría corrigiendo constantemente. Otro beneficio de esta estrategia es que, si en cualquier momento el objeto se aleja, el brazo robot podría volver a acercarse a él debido a que el bucle sigue comprobando la distancia entre ellos.

Además, gracias al uso de la función *find_Aruco* para la obtención de los vectores de traslación, cuando el objeto salga del campo de visión de la cámara debido al movimiento de la base, esta función iniciará nuevamente el proceso de búsqueda del objeto. Mientras tanto, la base móvil podría continuar su movimiento hacia el objeto utilizando la última distancia estimada. Esta funcionalidad permitiría corregir constantemente la posición del brazo robot a medida que se acerca al objeto, garantizando que este siempre se encuentre dentro del campo de visión de la cámara.

En la figura 37, se presenta un diagrama de bloques que detalla esta nueva implementación. De nuevo, las adiciones y modificaciones introducidas en esta versión se destacan en rojo, mientras que se conservan en negro los elementos que ya formaban parte de las implementaciones anteriores.

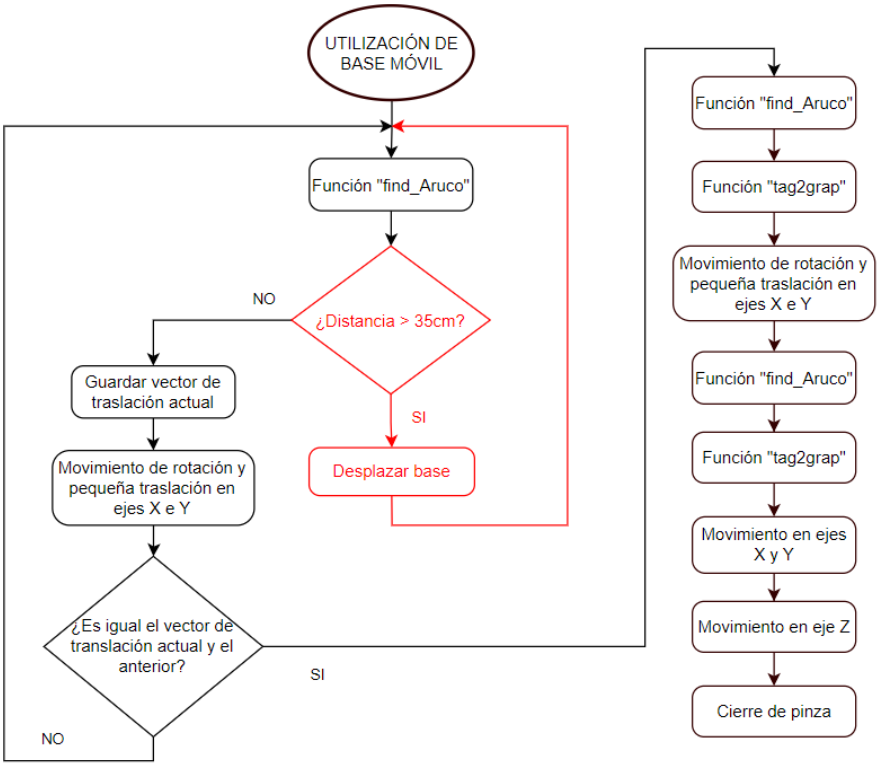

Figura 37. Implementación final del sistema

Como se puede apreciar en el diagrama, la única modificación introducida en esta implementación es la verificación de la distancia al objeto y, si este se encuentra lejos del brazo robot, se envía el comando de movimiento a la base móvil. Este es el esquema de funcionamiento del código final integrado en el sistema. Con esta implementación, se logra un agarre eficiente del objeto en una variedad de situaciones, ya sea cuando el objeto está estático con la misma u otra orientación con respecto a la cámara, cuando el objeto está en movimiento, o cuando se encuentra lejos de la base móvil.

4.3 Componentes hardware y comunicación

Una vez se ha logrado una precisa detección de la posición del objetivo mediante la adecuada calibración de la cámara y las transformaciones de sistemas de coordenadas previamente explicadas, y se ha implementado la versión final que será integrada en el sistema, es esencial comprender la manera en que los distintos componentes del hardware se comunicarán y funcionarán de manera coordinada. La eficacia y fluidez de esta comunicación resulta importante para el correcto funcionamiento de la implementación final descrita. Para lograr esta comunicación de manera óptima, es esencial llevar a cabo una organización previa de los datos que se utilizarán y determinar en qué parte del hardware serán gestionados. En esta sección, se describirán detalladamente las funciones de cada sistema que conforma el sistema global, los datos que manipularán y cómo se llevará a cabo la comunicación entre estos sistemas. Esta planificación es fundamental para garantizar la sincronización de todas las partes del proyecto y lograr una comunicación sin problemas entre los distintos sistemas. A continuación, se detallará la función de cada sistema y su rol en la comunicación general del proyecto.

4.3.1 Componentes hardware

Como se mencionó anteriormente, el hardware utilizado en este proyecto consta de varios componentes clave, cada uno desempeñando un papel específico en la consecución de los objetivos del proyecto. Estos componentes incluyen una cámara Intel Realsense D435, un brazo robot UR3, una base móvil RB1-BASE y una Raspberry Pi.

La cámara Intel Realsense D435 es esencial para la percepción visual y la detección de objetos mediante la captura de imágenes y la identificación de marcadores AruCo. A través de la cámara, y utilizando la librería OpenCV, podremos obtener los vectores que nos describen la posición del objeto.

El brazo robot UR3 es la herramienta principal para la manipulación física de objetos. Su tarea principal consiste en ejecutar movimientos precisos y controlados para posicionarse adecuadamente para realizar el agarre de objetos, así como abrir o cerrar la pinza en el momento apropiado. El brazo se mantiene en un estado de espera constante hasta que recibe comandos específicos para llevar a cabo estos movimientos o acciones de pinza. De esta manera, se adapta de manera eficiente a las necesidades del proyecto.

La base móvil RB1-BASE desempeña un papel fundamental al proporcionar la movilidad necesaria para el sistema. Su función principal radica en permitir que el robot se desplace a ubicaciones específicas, lo que resulta fundamental para llevar a cabo tareas logísticas y de manipulación de objetos de manera efectiva. En su operación, la base móvil permanece en espera hasta recibir un mensaje enviado por la Raspberry Pi. Dependiendo del tipo de mensaje, la base móvil ejecuta acciones como enviar comandos de movimiento al brazo robot o iniciar su propio desplazamiento a una ubicación específica. De esta forma, la base móvil se integra de manera flexible en el sistema y responde a las necesidades de movimiento del proyecto.

Aunque los sistemas previamente mencionados podrían ser suficientes para cumplir con los objetivos del proyecto, se ha incorporado un componente adicional: la Raspberry Pi. Su función principal consiste en aliviar la carga de trabajo de la base móvil, lo que permite una distribución de responsabilidades más eficiente en el sistema global. Esto, a su vez, mejora la coordinación y la ejecución de tareas, contribuyendo significativamente a la eficiencia general del proyecto. La tarea principal de la Raspberry Pi es monitorear el estado del sistema utilizando la información proporcionada por la cámara, y tomar las acciones necesarias en función de esta información. Luego, la Raspberry Pi envía mensajes específicos a la base móvil según la situación detectada. Esta comunicación inteligente entre la Raspberry Pi y la base móvil permite una respuesta ágil y adecuada a diversas condiciones y requisitos del proyecto.

4.3.1.1 Comunicación entre componentes

En este apartado, se explicará en qué sistemas hardware se llevan a cabo las diversas acciones que componen la implementación utilizada en el sistema global (el que se muestra en el diagrama de la figura 37). Además, se definirán las comunicaciones que se establecen entre los diferentes componentes y cómo se ejecutan.

Primero, se establece una conexión inicial entre la cámara y la Raspberry Pi mediante un cable USB 3.0. Esta conexión permite que la Raspberry Pi obtenga los vectores de traslación y rotación, utilizando los datos capturados por la cámara.

El segundo tipo de conexión se realiza entre la Raspberry Pi y el robot móvil RB1-BASE. Estos dos componentes se comunican a través de una red compartida y utilizan el entorno de desarrollo ROS como plataforma de comunicación. Para que se puedan intercambiar información y comandos, ambos dispositivos deben estar conectados a la misma red. Esto se logra mediante una red local específica, establecida a través de un router incorporado en la base RB1-BASE. La existencia de esta red local permite que los dispositivos se comuniquen eficientemente sin depender de la red Wi-Fi principal. En el entorno de desarrollo ROS, la comunicación entre los nodos del sistema se gestiona mediante un núcleo central conocido como el "core de ROS" o *Roscore*. Tanto la Raspberry Pi como la base RB1-BASE deben conectarse al mismo Roscore para garantizar una comunicación efectiva.

Finalmente, la última comunicación se establece entre la base RB1-BASE y el brazo robot UR3 mediante RTDE. RTDE es una interfaz que permite el intercambio de datos en tiempo real entre dos dispositivos o sistemas, en este caso, la base y el brazo robot, en una configuración de automatización industrial. Para lograrlo, la librería UR_RTDE de Python incorpora comandos para controlar el movimiento del robot, obtener datos del mismo y gestionar sus entradas y salidas.

Esta triple comunicación (Raspberry Pi – RB1-BASE – UR3) permite controlar de manera adecuada los movimientos del brazo robot para lograr una manipulación precisa de los objetos. Para gestionar esta comunicación, la RB1-BASE incluye un servidor de acciones de ROS. Un servidor de acciones es un programa que proporciona un mecanismo para realizar tareas que requieren una secuencia predefinida y a menudo secuencial. En ROS, los servidores de acciones se basan en el concepto de *actions* como una forma de gestionar tareas complejas y secuenciales en sistemas robóticos.

A través de este servidor de acciones, se logra la comunicación con la Raspberry Pi mediante ROS y con el UR3 mediante RTDE. Para el funcionamiento adecuado del sistema global, se requieren tres acciones diferentes que deben comunicarse. La primera de ellas es el cierre o apertura de la pinza. La segunda es el movimiento del brazo robot a una coordenada específica. Y la última es el movimiento de la base a una posición específica. Para gestionar estas tres acciones, se crearon tres *actions* que debe manejar el servidor. Estas permiten la publicación de mensajes en una serie de *topics* específicos para cada una de ellas. Por lo tanto, la Raspberry Pi debe publicar en los *topics* /goal correspondientes a cada *action* para comunicarse con la base. Además, para conocer el estado de cada tarea, se puede utilizar el *topic /feedback* correspondiente a cada una de ellas para verificar si se ha completado la acción.

A continuación, se detallan los diferentes *actions* incluidos en el servidor de acciones:

- ***Action picking_as:*** Este *action* se emplea para llevar a cabo las acciones de apertura y cierre de la pinza incorporada en el brazo robot. El tipo de mensaje utilizado para este *action* es un entero que puede tomar valores 0 o 1. Cuando se completa el movimiento y el brazo se encuentra en la posición de agarre del objeto, la Raspberry Pi publica un valor de 0 en el *topic picking_as/goal.* En ese momento, el servidor de acciones de la base móvil recibe este mensaje y ejecuta la tarea asociada a él, que en este caso implica enviar a través de RTDE al brazo robot la orden de cerrar la pinza.

- ***Action move_base_as:*** El segundo *action* implementado se utiliza para llevar a cabo los movimientos de la base del robot hacia una coordenada específica. El tipo de mensaje utilizado para esta acción es un vector de 6 elementos que contiene números reales. Cuando la Raspberry Pi detecta que el objeto se encuentra a una distancia que impide que el brazo robot lo alcance, envía a través del *topic move_base_as/goal* el vector de traslación y rotación obtenido. Cuando el servidor de acciones de la base móvil detecta un mensaje en este *topic*, ejecuta la acción correspondiente. En este caso, la acción implica realizar un movimiento en la base del robot, publicando en el *topic /robot/move/goal* el tercer elemento del vector recibido, que representa la distancia en el eje Z desde la cámara hasta el objeto, restando 20 centímetros para asegurar que el robot se coloque a una distancia adecuada para realizar una manipulación precisa del objeto.

- ***Action move_as:*** El tercer *action* implementado tiene la finalidad de ejecutar movimientos específicos en el brazo robot para posicionarlo en una coordenada particular. El tipo de mensaje utilizado en este *action* consiste en un vector de 6 elementos que contiene números reales, que representan la ubicación del objetivo con respecto a la pinza del robot. Cuando el programa en la Raspberry Pi requiere que el brazo robot realice un movimiento, ya sea para buscar, seguir o posicionarse para agarrar un objeto, se publica un mensaje en el *topic move_as/goal*. Es importante destacar que se espera a que cada movimiento se complete antes de continuar con el programa, para evitar interferir con las mediciones en curso. Para lograr esto, la Raspberry Pi se suscribe al *topic move_as/feedback*, donde el servidor envía un mensaje de *True* cuando el movimiento ha finalizado.

  Cuando se recibe un mensaje desde la Raspberry Pi en el *topic move_as/goal*, se lleva a cabo la tarea correspondiente a este *action*. En este caso, se realiza la última transformación necesaria, que convierte las coordenadas del sistema de referencia de la pinza en las coordenadas del sistema de referencia del brazo robot. Esta transformación se realiza en la base, ya que es necesario estar conectado al UR para obtener los datos relativos a la cinemática directa del robot. Finalmente, se verifica si el movimiento específico es posible sin llegar a una singularidad del robot y, en caso afirmativo, se envía al brazo robot UR3 para su ejecución

En la figura 38, se presenta un diagrama que proporciona una representación visual de las funciones realizadas por cada componente y las comunicaciones que ocurren entre ellos. Cada columna representa las acciones llevadas a cabo por un componente particular y se ha resaltado en color rojo el método de comunicación entre los sistemas individuales.

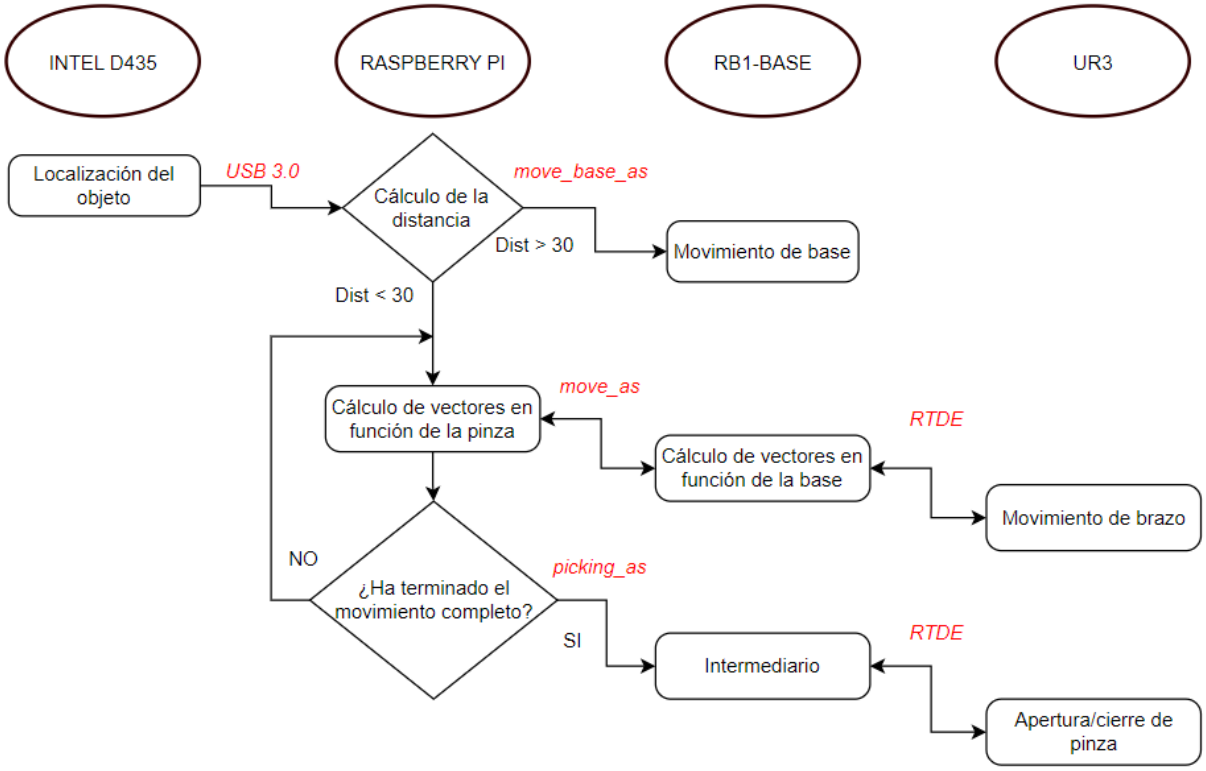

Figura 38. Funciones y comunicación entre componentes

5 RESULTADOS

En este capítulo se van a mostrar imágenes del brazo robot para cada una de las implementaciones que se han explicado en la sección anterior, explicando para cada una de ellas el momento y posición en el que se encuentra cada vez el brazo robot. De esta forma, se va a poder comprobar el correcto funcionamiento de cada una de estas implementaciones y, por tanto, del sistema global.

5.1.1 Objetos estáticos con la misma orientación que la pinza.

En este primer caso, la pinza se encuentra inicialmente con la misma orientación que el objeto, lo que significa que no se necesita realizar ningún movimiento de rotación para agarrarlo. En la figura 39 se puede ver la posición inicial en un ejemplo para este caso.

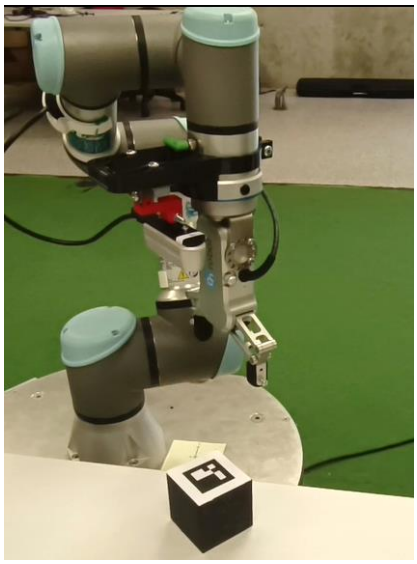

Figura 39. Posición inicial objeto estático con misma orientación

El primer paso en esta implementación es realizar un movimiento en los ejes X e Y para colocarse directamente encima del cubo, como se muestra en la figura 40.

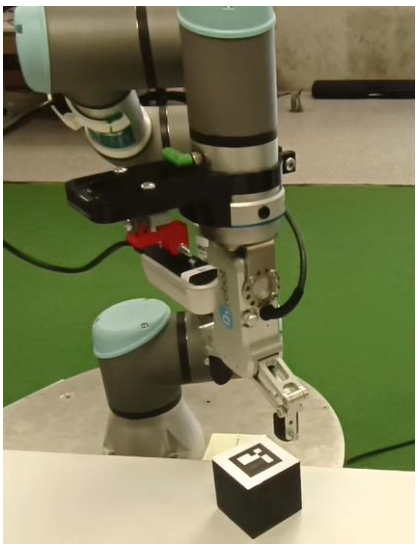

Figura 40. Brazo robot encima del cubo

A continuación, el brazo robot realiza un desplazamiento en el eje Z, es decir, desciende para posicionarse en la ubicación ideal para el agarre del objeto, como se observa en la figura 41.

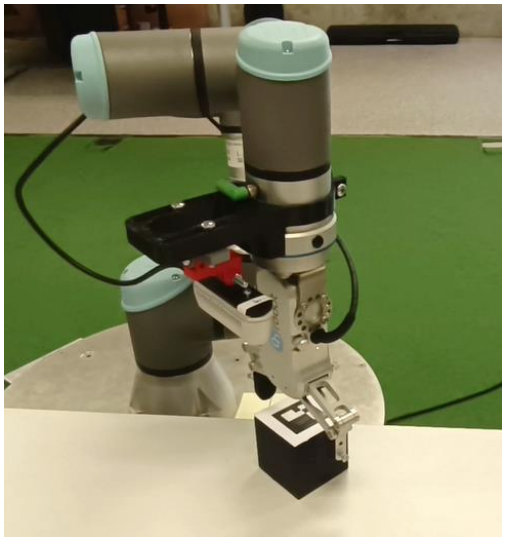

Figura 41. Brazo robot en posición de agarre

Por último, el brazo cierra la pinza para agarrar finalmente el cubo, como se muestra en la figura 42.

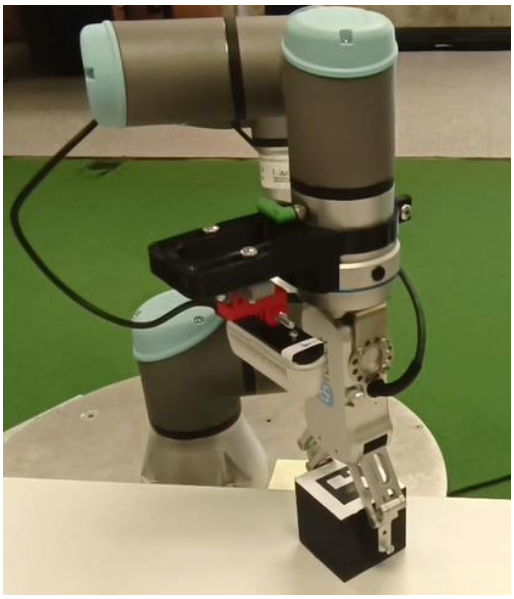

Figura 42. Objeto agarrado

Esta secuencia de movimientos demuestra la capacidad del brazo robot para agarrar objetos estáticos que tienen la misma orientación que la pinza.

5.1.2 Objetos estáticos con distinta orientación que la pinza

En este apartado se van a ver ejemplos de los dos casos explicados para la manipulación de un objeto estático cuando no tiene la misma orientación que la pinza.

5.1.2.1 Implementación con dos capturas de imagen

Inicialmente, el robot se encuentra en la posición que se muestra en la figura 43, es decir, no es correcta su posición ni su orientación para un agarre correcto del objeto, por lo que se necesitará llevar a cabo una traslación y una rotación de la pinza.

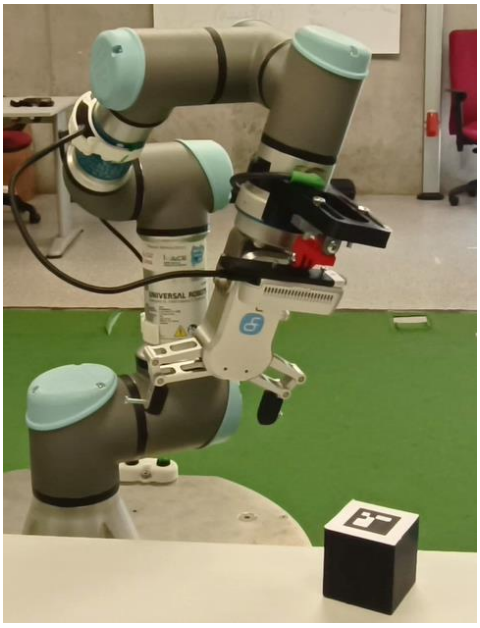

Figura 43. Posición inicial para implementación con dos capturas de imagen

El primer paso es detectar el código AruCo, ya que, como se muestra en la imagen anterior, el objeto no se encuentra en el campo de visión de la cámara. Una vez que se encuentra el código, el brazo robot se ha desplaza a la posición que se observa en la figura 44.

Figura 44. Posición del robot al encontrar objeto

Como se puede observar, la búsqueda del objeto simplemente implica un movimiento de traslación de la pinza; es decir, no hay ningún cambio en la orientación de la pinza del robot. El siguiente paso, una vez encontrado el objeto, es el proceso de orientación. En la siguiente figura se muestra cómo queda el brazo robot tras rotar para colocarse con la misma orientación que el cubo.

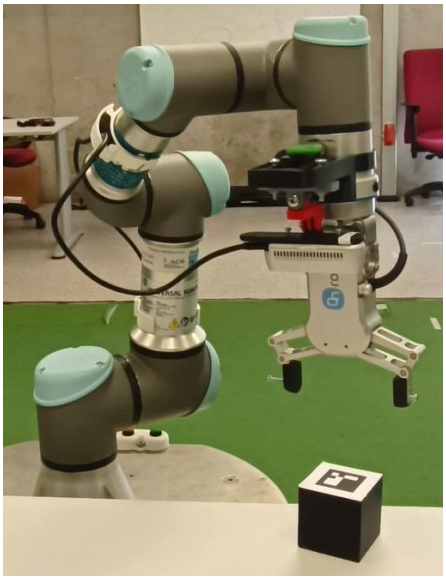

Figura 45. Brazo robot orientado con la misma orientación que el objeto

Una vez encontrado el objeto, el brazo robot se coloca encima del cubo después de desplazarse en los ejes X e Y, como se muestra en la figura 46.

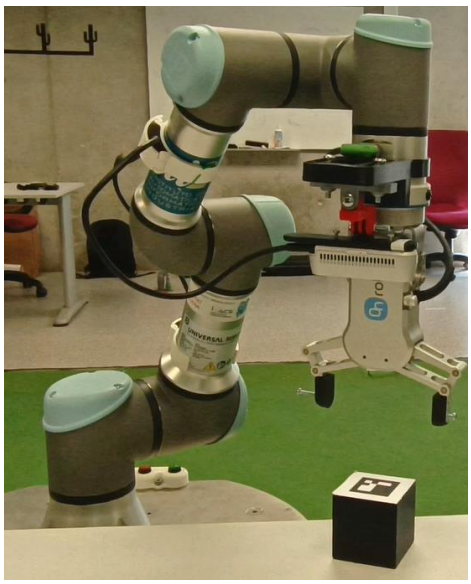

Figura 46. Brazo robot encima del cubo

Finalmente, el brazo robot debe desplazarse en el eje Z y cerrar la pinza para agarrar el objeto, como se ilustra en la figura 47.

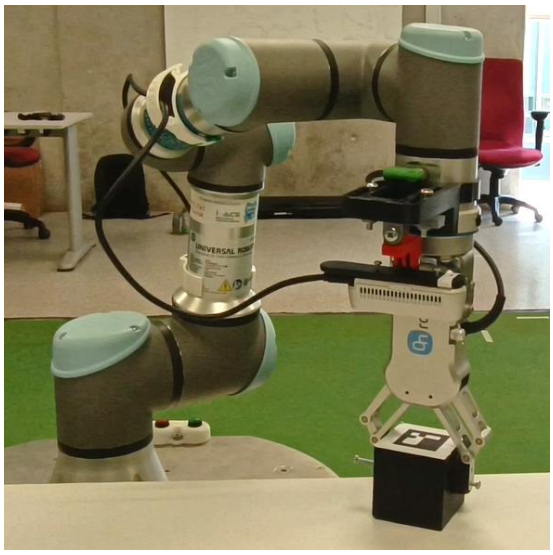

Figura 47. Objeto agarrado

5.1.2.2 Corrección de vector de traslación

En esta segunda implementación, como se ha mencionado previamente, se capturan los vectores hacia el objeto utilizando solo una imagen y se envían los movimientos de rotación junto con los de traslación en los ejes X e Y. La situación inicial para un movimiento de este tipo se muestra en la figura 48.

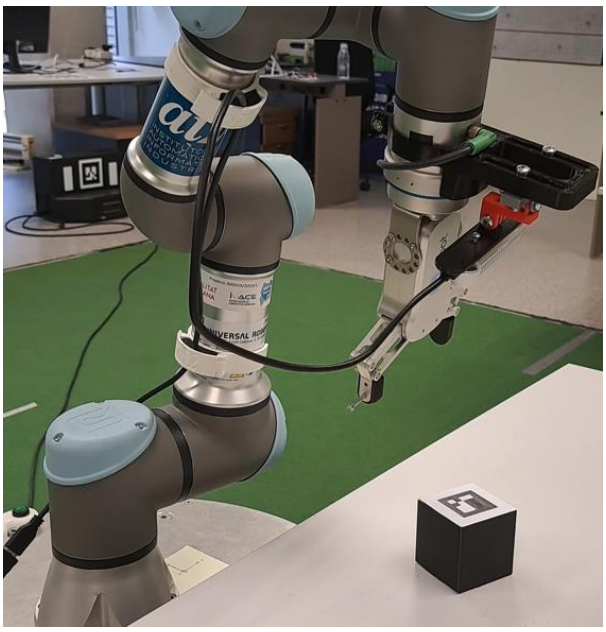

Figura 48. Situación inicial para corrección de vector de traslación

El primer paso que realiza esta implementación consiste en colocarse encima del cubo con la orientación correcta en un solo movimiento, como se observa en la figura 49.

Figura 49. Garra colocada encima del cubo

Finalmente, el último paso en esta implementación, al igual que en la anterior, es descender su posición en el eje Z para colocarse en la zona de agarre y cerrar la pinza, como se muestra en la figura 50.

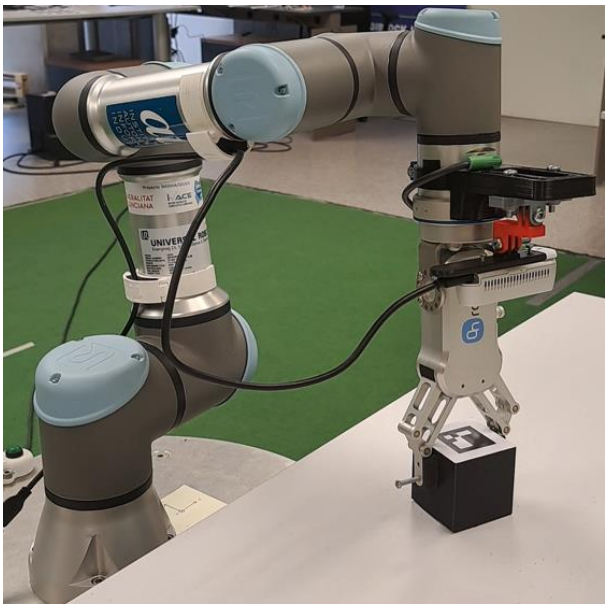

Figura 50. Brazo robot con objeto agarrado

5.1.3 Control de agarre para objetos en movimiento

En esta nueva implementación, como se ha mencionado previamente, el brazo robot sigue al cubo utilizando los vectores que se obtienen en función de la cámara hasta que el objeto se detiene. En ese momento, se procede con el agarre de la misma manera que se ha explicado anteriormente. La figura 51 muestra el inicio de una configuración de este tipo.

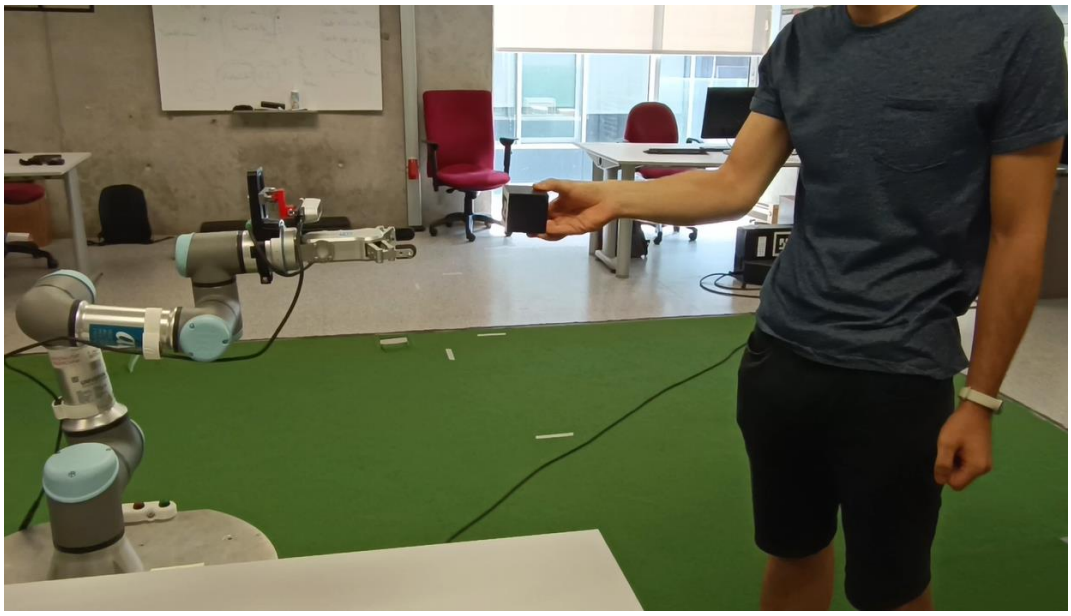

Figura 51. Situación inicial para objeto en movimiento

En ese momento, el robot debe ejecutar una serie de movimientos para garantizar un seguimiento preciso del cubo. En la figura 52, se ilustra cómo el brazo robot ajusta la posición de la cámara para alinearla con el cubo.

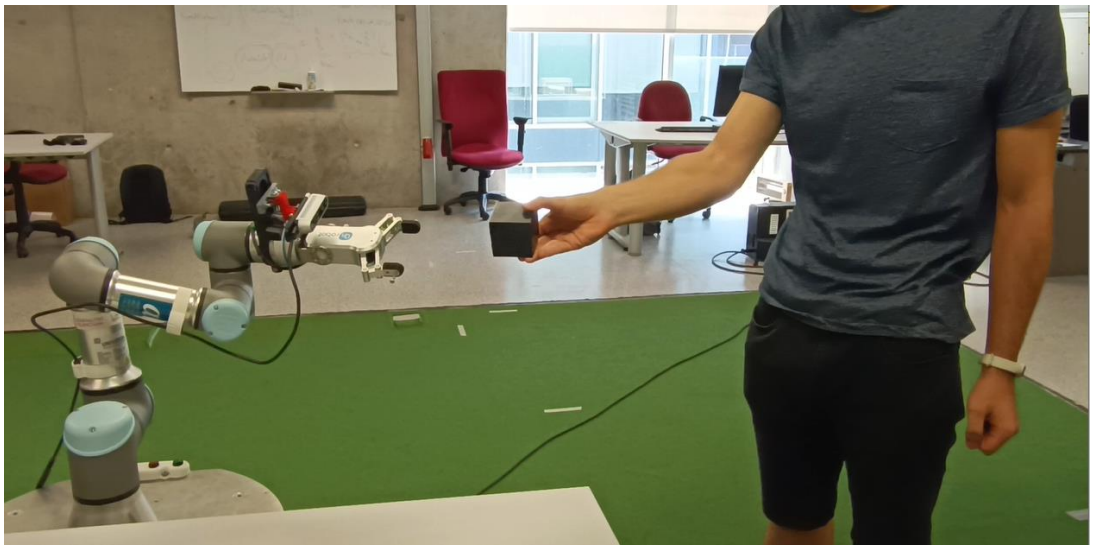

Figura 52. Brazo robot realizando seguimiento del objeto

Una vez que el objeto se detiene, se realizan los movimientos de la misma manera que en las implementaciones anteriores. Primero, se ajusta la orientación, luego se efectúa el movimiento en los ejes X e Y, después en el eje Z, y finalmente, se cierra la garra para agarrar el objeto. En la figura 53, se muestra el final del movimiento una vez que el brazo robot ha logrado agarrar el objeto.

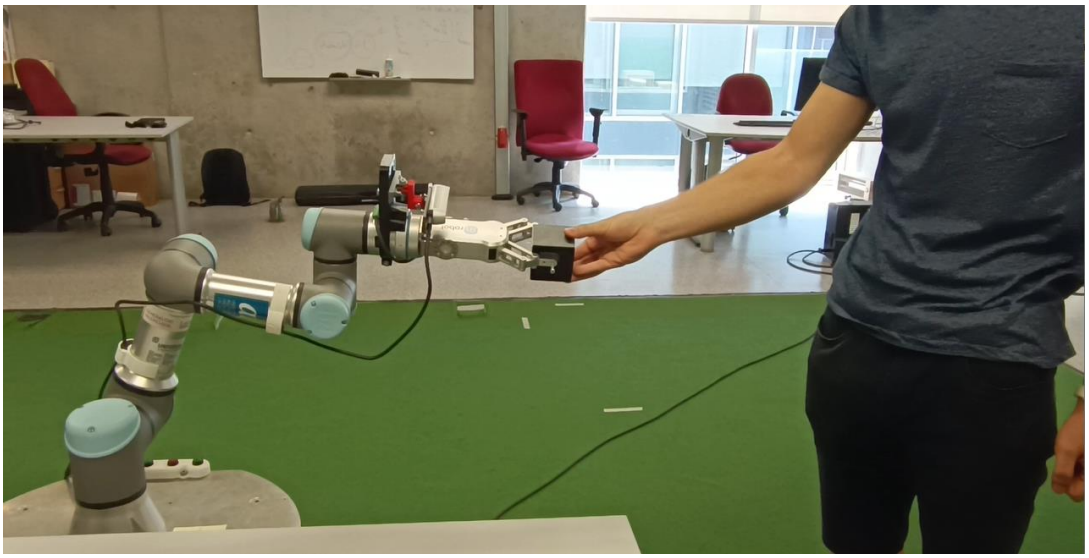

Figura 53. Brazo robot con el objeto agarrado

5.1.4 Control de agarre con movimiento de la base

Este último caso es la implementación final del sistema, es decir, permite el agarre y manipulación del objeto en cualquiera de las situaciones anteriores y, además, permite al robot móvil acercarse cuando el brazo robot no alcanza el objetivo. En la figura 54 se ve el inicio de este movimiento. Inicialmente, se ha tapado el cubo para observar que el brazo robot ejecuta la tarea de búsqueda del objeto.

Figura 54. Brazo robot buscando objeto

Una vez que se destapa el objeto, la cámara lo detecta y evalúa que la distancia para el brazo robot es demasiado alta, por lo que comienza a moverse la base móvil hasta alcanzar una posición asequible para el brazo robot, como se ve en la figura 55.

Figura 55. Robot en posición asequible para agarrar el objeto

A partir de aquí, el brazo robot trabaja como en los casos anteriores, es decir, primero realiza un seguimiento de la pieza y posteriormente, cuando el objeto se para, se realiza el proceso de agarre. En la figura 56 se observa como el robot sigue la pieza a medida que el objeto se mueve.

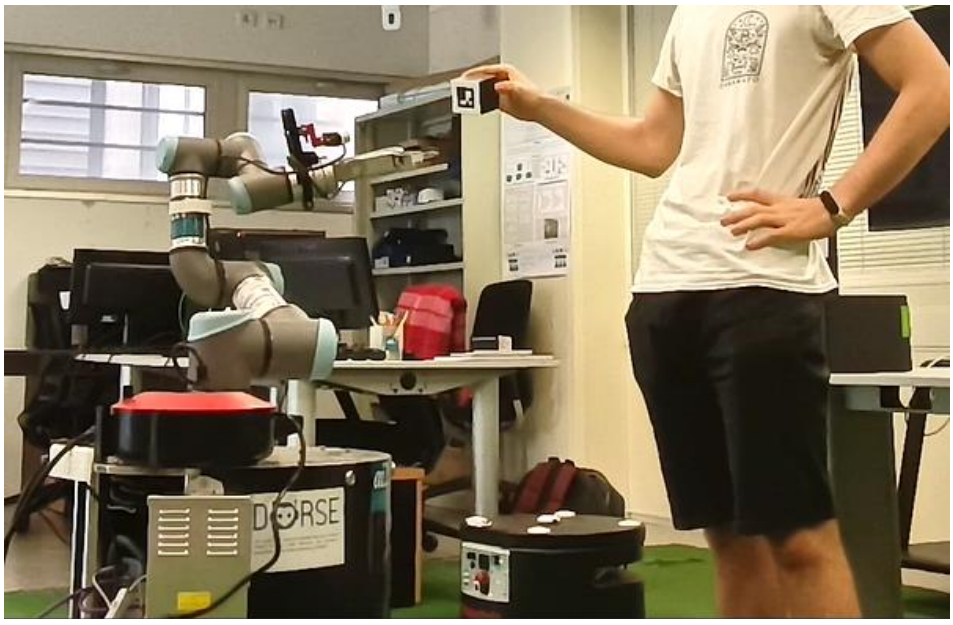

Figura 56. Robot realizando seguimiento de la pieza

Por último, una vez que el objeto queda inmóvil, el brazo realiza el agarre, como se ve en la figura 57.

Figura 57. Brazo robot con el objeto agarrado

6 CONCLUSIONES

La realización de este trabajo ha resultado en el desarrollo de un sistema completo que logra la coordinación entre varios componentes hardware, en particular, entre una base móvil y un brazo robot, para llevar a cabo tareas logísticas y de manipulación de objetos identificados a través de una cámara incorporada en el extremo del brazo robot.

Para alcanzar el correcto funcionamiento de este sistema, se adoptaron dos enfoques principales. En primer lugar, se centró en el aspecto software del sistema, trabajando en una implementación final que permitiera su funcionamiento independientemente de los componentes integrados. Se siguió un enfoque progresivo, resolviendo problemas más sencillos inicialmente y avanzando hacia los más complejos, lo que llevó a una implementación final que cumplió con los requisitos iniciales y funcionó correctamente para todos los casos.

El segundo enfoque se centró en la elección y comunicación entre los componentes software. Esta etapa permitió dividir las aplicaciones software entre varios componentes y, por tanto, aligerar la carga de trabajo de los mismos. Para lograrlo, se utilizó ROS, incluyendo un servidor de acciones en la base móvil como elemento central de la comunicación.

Respecto a los objetivos mencionados en el apartado 1.2 de esta memoria, se ha demostrado su cumplimiento a lo largo del trabajo como se detallará a continuación:

- **Comunicación fluida y de baja latencia entre el hardware.** La sección 4.3.1.1 de esta memoria explica cómo se logró una comunicación efectiva entre los diferentes componentes hardware, incluyendo la creación de *actions* y un servidor de acciones de ROS incluido en la base móvil.

- **Obtención de información confiable sobre los objetos mediante la visión 3D.** En el apartado 4.1.1, se describe cómo se llevó a cabo la calibración de la cámara para obtener información fiable sobre los objetos.

- **Medición precisa de distancias a los objetos de interés.** Para lograr este objetivo, la sección 4.2.1 explica la función *find_Aruco* utilizada en la implementación final, que utiliza la librería OpenCV para obtener vectores de traslación y rotación desde la cámara al objeto de interés.

- **Obtención de vectores de traslación y rotación para su uso en el brazo robot.** Los apartados 4.1.2 y 4.2.2.2 explican las transformaciones realizadas para convertir los vectores obtenidos desde la cámara en vectores compatibles con el sistema de coordenadas del brazo robot.

- **Seguimiento continuo de un objeto en movimiento por parte del brazo robot.** En el apartado 4.2.3, se detalla la implementación para el seguimiento de objetos en movimiento y su posterior agarre cuando se detienen.

- **Evaluación de la distancia al objeto y movimientos de la base móvil si es necesario.** El apartado 4.2.4 describe cómo se determina si la distancia al objeto permite el agarre sin mover la base móvil y cómo se realiza el movimiento de la base cuando es necesario, seguido por el posterior agarre del objeto.

El logro de todos estos objetivos en su conjunto ha contribuido al cumplimiento del objetivo principal del trabajo, que es desarrollar un sistema robótico integrado y colaborativo capaz de realizar acciones de logística mediante la detección, seguimiento y manipulación eficiente de objetos, empleando una plataforma móvil y un brazo robot.

En cuanto a las posibles mejoras y trabajos futuros derivados de esta investigación, se identifican principalmente en la última implementación, la cual involucra el movimiento de la base móvil para alcanzar el objeto. La implementación actual se enfoca únicamente en analizar la distancia en el eje Z con respecto al objeto y moverse en una dirección específica, es decir, hacia adelante de la base móvil. Sin embargo, existe margen para considerar la posición lateral del objeto con respecto a la base, lo que podría implicar realizar giros en la base móvil. Esta adición permitiría una mayor libertad en los movimientos del robot, lo que resultaría en un seguimiento más preciso del objeto, especialmente si este se desplaza lateralmente.

Además, este trabajo sienta las bases para la aplicación del sistema desarrollado en diversas áreas, como las mencionadas en el apartado 2.3.1. Entre estas aplicaciones, se destaca la coordinación entre robots y humanos en la industria. La capacidad de este sistema para detectar, seguir y manipular objetos de manera eficiente podría utilizarse en entornos de colaboración entre humanos y robots, lo que podría mejorar la eficiencia y seguridad en una variedad de tareas industriales.

7 BIBLIOGRAFÍA

[1] «Valuer. Cobots in manufacturing: the future of industry,» [En línea]. Available: https://www.valuer.ai/blog/cobots-in-manufacturing-the-future-of-industry.

[2] «Fanuc. Industrial Robots for AGV Robots and Autonomous Mobile Robots (AMRs),» [En línea]. Available: https://www.fanucamerica.com/products/robots/robot-options/mobile-robots.

[3] «Robotnik. Página principal de Robotnik,» [En línea]. Available: https://robotnik.eu .

[4] «Universal Robots. Págnia principal de Universal Robots,» [En línea]. Available: https://www.universal-robots.com/.

[5] «Universal Robots. Real-Time Data Exchange (RTDE) Guide,» [En línea]. Available: https://www.universal-robots.com/articles/ur/interface-communication/real-time-data-exchange-rtde-guide/.

[6] «ISO 8373:2021. Robotics — Vocabulary,» Organización Internacional de Normalización, 2021.

[7] «Only natural energy. Why is robotics important?,» [En línea]. Available: https://www.onlynaturalenergy.com/why-is-robotics-important/.

[8] «Integritas. Robots de servicio, la atención del futuro para las industrias,» [En línea]. Available: https://www.integritas.mx/blog/transformacion-digital-4/robots-la-atencion-del-futuro-31.

[9] A. Gasparetto y L. Scalera, «A Brief History of Industrial Robotics in the 20th Century,» *CINECA IRIS,* 2019.

[10] F. Chiacchio, G. Petropoulos y D. Pichler, «The impact of industrial robots on EU employment and wages: A local labour market approach,» *Bruegel Working Paper,* 2018.

[11] «Robotnik. What is a cobot and its benefits?,» [En línea]. Available: https://robotnik.eu/cobots-and-its-benefits-for-industry/#:~:text=Cobots%20or%20collaborative%20robots%20are,of%20the%20goals%20of%20cobots..

[12] «ISO 10218-2:2011,» Organización Internacional de Normalización, 2011.

[13] K. Berntorp, K. Arzen y A. Robertsson, «Mobile manipulation with a kinematically redundant manipulator for a pick-and-place scenario,» *IEEE,* 2012.

[14] M. Nieuwenhuisen, D. Droeschel y D. Holz, «Mobile bin picking with an anthropomorphic service robot,» *IEEE,* 2013.

[15] G. Michalos, S. Makris y J. Spiliotopoulos, «ROBO-PARTNER: seamless human-robot cooperation for intelligent, flexible and safe operations in the assembly factories of the future,» 2014.

[16] R. Krug, T. Stoyanov, V. Tincani y H. Andreasson, «The next step in robot commissioning: autonomous picking and palletizing,» *IEEE,* 2016.

[17] A. Dömel, S. Kriegel y M. Kassecker, «Toward fully autonomous mobile manipulation for industrial environments,» 2017.

[18] V. Unhelkar, S. Dörr, A. Bubeck y P. Lasota, «Mobile robots for moving-floor assembly lines: design, evaluation, and deployment,» *IEEE,* 2018.

[19] M. Yang, E. Yang, R. Zante y M. Post, «Collaborative mobile industrial manipulator: A review of system architecture and applications,» *IEEE,* 2019.

[20] Z. Bi, C. Luo, Z. Miao y B. Zhang, «Safety assurance mechanisms of collaborative robotic systems in manufacturing,» *Robot Comput-Integr Manuf,* 2021.

[21] J. Aleotti, A. Baldasarri y M. Bonfè, «Toward Future AutomaticWarehouses: An Autonomous Depalletizing System Based on Mobile Manipulation and 3D Perception,» *MDPI,* 2021.

[22] S. Comari, R. Di Leva y M. Carriato, «Mobile cobots for autonomous raw-material feeding of automatic packaging machines,» *Elsevier,* 2022.

[23] A. Prateek y C. Papachristos, «Mobile Manipulator Robot Visual Servoing and Guidance for Dynamic Target Grasping,» *Springer,* 2020.

[24] «ROS. Página principal de ROS,» [En línea]. Available: https://www.ros.org/ .

[25] «Robotnik. RB-1 BASE,» [En línea]. Available: https://robotnik.eu/es/productos/robots-moviles/rb-1-base/.

[26] «Universal Robots. Ur3-robot,» [En línea]. Available: https://www.universal-robots.com/products/ur3-robot/.

[27] «Intel. Depth Camera D435,» [En línea]. Available: https://www.intelrealsense.com/depth-camera-d435/.

[28] «Raspberry Pi. Página principal de Raspberry Pi,» [En línea]. Available: https://www.raspberrypi.com/.

[29] «Python. Página principal de Python,» [En línea]. Available: https://www.python.org/.

[30] «Uco. Aruco,» [En línea]. Available: https://www.uco.es/investiga/grupos/ava/portfolio/aruco/.

[31] «OpenCv. Página principal OpenCV,» [En línea]. Available: https://opencv.org/.

[32] «Robotics backend. What is a ROS service,» [En línea]. Available: https://roboticsbackend.com/what-is-a-ros-service/.

[33] «MathWorks. ROS Actions Overview,» [En línea]. Available: https://es.mathworks.com/help/ros/ug/ros-actions.html.

[34] «The construct. What is ROS Action?,» [En línea]. Available: https://www.theconstructsim.com/ros-5-mins-034-ros-action/.

[35] «e-con Systems. What are RGBD cameras?,» [En línea]. Available: https://www.e-consystems.com/blog/camera/technology/what-are-rgbd-cameras-why-rgbd-cameras-are-preferred-in-some-embedded-vision-applications/.

[36] «ROS Components. Hokuyo UST-20LX,» [En línea]. Available: https://www.roscomponents.com/es/lidar-escaner-laser/86-ust-20lx.html.

[37] «ROS Components. Orbbec Astra,» [En línea]. Available: https://www.roscomponents.com/es/camaras/76-orbbec.html.

[38] «Teledyne Photometrics. Rolling vs global shutter,» [En línea]. Available: https://www.photometrics.com/learn/advanced-imaging/rolling-vs-global-shutter.

[39] X. Zhong y Y. Liang, «Raspberry Pi: An Effective Vehicle in Teaching the Internet of Things in Computer Science and Engineering,» *IEEE,* 2016.

[40] «Django. Página principal de Django,» [En línea]. Available: https://www.djangoproject.com/.

[41] «Flask. Página principal de Flask,» [En línea]. Available: https://flask.palletsprojects.com/en/2.3.x/.

[42] «Numpy. Página principal de Numpy,» [En línea]. Available: https://numpy.org/.

[43] «pandas. Página principal de pandas,» [En línea]. Available: https://pandas.pydata.org/.

[44] «TensorFlow. Página principal de TensorFlow,» [En línea]. Available: https://www.tensorflow.org/.

[45] R. Vallat, «Pingouin: statistics in Python,» *JOSS,* 2018.

[46] «Datademia. ¿Qué lenguajes de programación deberías conocer en 2022?,» [En línea]. Available: https://datademia.es/blog/que-lenguajes-de-programacion-deberias-conocer-en-2022.

[47] Z. Siki y T. Bence, «Automatic Recognition of ArUco Codes in Land Surveying Tasks,» 2021.

[48] «tutorialspoint. OpenCV - Overview,» [En línea]. Available: https://www.tutorialspoint.com/opencv/opencv_overview.

[49] «MathWorks. Support Vector Machine (SVM),» [En línea]. Available: https://es.mathworks.com/discovery/support-vector-machine.html.

[50] «OpenCV. Canny Edge Detection,» [En línea]. Available: https://docs.opencv.org/3.4/da/d22/tutorial_py_canny.html.

[51] «OpenCV. Hough Transform,» [En línea]. Available: https://docs.opencv.org/3.4/d2/d15/group__cudaimgproc__hough.html.

[52] «UR_RTDE. Página principal de UR_RTDE,» [En línea]. Available: https://sdurobotics.gitlab.io/ur_rtde/index.html.