UNIVERSIDAD DEL AZUAY

FACULTAD DE CIENCIA Y TECNOLOGÍA

ESCUELA DE INGENIERÍA ELECTRÓNICA

Planeación de movimientos de un brazo robótico de 5 grados de libertad para tareas de manipulación mediante visión por computador

Trabajo de graduación previo a la obtención del título de:

INGENIERO ELECTRÓNICO

Autor:

Luis Rodrigo Alvear Zeas

Director:

Mst. Andrés Cabrera

CUENCA, ECUADOR

2024

AGRADECIMIENTOS

En este momento significativo de mi vida, quiero expresar mi gratitud ante todas aquellas personas que han llegado a formar parte en mi desarrollo profesional, tanto directa como indirectamente. Su apoyo inquebrantable es algo que valoro en gran medida, así como las relaciones que se han forjado con el paso de los años. De entre todos, a quienes tengo el mayor gusto que hayan participado de manera activa en mi desarrollo son mis abuelos, quienes siempre han sido una inspiración y un pilar en mi formación tanto académica como humana. Agradezco en gran medida a mis padres y hermanos que siempre me han brindado su ayuda de forma incondicional. Y agradezco también a mi director de tesis, quien en todo momento ha ofrecido sus conocimientos para lograr el desarrollo óptimo de este trabajo. Ante todos ellos, mi meta será retribuir todo la confianza que ha sida puesta en mis manos mediante la correcta aplicación de los conocimientos adquiridos.

PLANEACIÓN DE MOVIMIENTOS DE UN BRAZO ROBÓTICO DE 5 GRADOS DE LIBERTAD PARA TAREAS DE MANIPULACIÓN MEDIANTE VISIÓN POR COMPUTADOR

El presente trabajo tiene como finalidad generar la planificación de movimientos para un brazo robótico especializado en tareas de agarre, usando la retroalimentación proporcionada por una cámara de profundidad. La programación del sistema se realizó mediante el framework para el manejo de robots (ROS) y el uso de Python para la elaboración de los scripts. La integración de estos elementos permitió la manipulación precisa de objetos de tamaño pequeño, lo cual se comprobó a través de pruebas en escenarios físicos y simulados. Además, se realizaron pruebas utilizando dos algoritmos de planeación de movimientos distintos (PRM y RRT-Connect), los cuales fueron comparados en escenarios con obstáculos en entornos simulados y físicos. El trabajo concluyó satisfactoriamente, logrando la captura del escenario físico y proporcionando esta información al sistema para planificar rutas que eviten colisiones.

Palabras clave: Planeación de movimientos, Manipulación, Cámara de profundidad, Brazo robótico, Control de robots

MOTION PLANNING FOR A 5 DEGREES OF FREEDOM ROBOTIC ARM FOR MANIPULATION TASKS USING COMPUTER VISION

The objective of this work is to develop the motion planning for a robotic arm specialized in grasping tasks, utilizing a depth camera as a feedback mechanism for the system. The system was programmed using an open-source framework designed for robot control (ROS), and the scripts were written using Python. The integration of these elements allowed the precise manipulation of small objects, which was verified through tests in physical and simulated scenarios. In addition, tests were carried out using two different motion planning algorithms (PRM and RRT-Connect), which were compared in scenarios with obstacles in simulated and physical environments. The work concluded satisfactorily, capturing the physical scenario and providing this information to the system to plan routes that avoid collisions.

Keywords: Motion planning, Manipulation, Depth camera, Robotic arm, Robot control

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Planeación de movimientos de un brazo robótico de 5 grados de libertad para tareas de manipulación mediante visión por computador

Luis Alvear
Escuela de Ingeniería Electrónica
Universidad del Azuay
Cuenca, Ecuador
laaz@es.uazuay.edu.ec

***Resumen*—El presente trabajo tiene como finalidad generar la planificación de movimientos para un brazo robótico especializado en tareas de agarre, usando la retroalimentación proporcionada por una cámara de profundidad. La programación del sistema se realizó mediante el framework para el manejo de robots (ROS) y el uso de Python para la elaboración de los scripts. La integración de estos elementos permitió la manipulación precisa de objetos de tamaño pequeño, lo cual se comprobó a través de pruebas en escenarios físicos y simulados. Además, se realizaron pruebas utilizando dos algoritmos de planeación de movimientos distintos (PRM y RRT-Connect), los cuales fueron comparados en escenarios con obstáculos en entornos simulados y físicos. El trabajo concluyó satisfactoriamente, logrando la captura del escenario físico y proporcionando esta información al sistema para planificar rutas que eviten colisiones.**

***Palabras Clave*—Planeación de movimientos, Manipulación, Cámara de profundidad, Brazo robótico**

I. Introducción

Las tareas manuales siguen siendo una parte intrínseca de muchas industrias, desde la fabricación de prendas de vestir y la construcción hasta la industria y la electrónica. Sin embargo, estas tareas no solo son exigentes en términos de esfuerzo físico, sino que también requieren un alto grado de precisión, estabilidad y resistencia a la fatiga. Dichas condiciones no son ideales para las personas, pues existen limitaciones físicas y biológicas inherentes que impiden la realización de estas actividades de forma repetitiva durante periodos prolongados antes de padecer enfermedades derivadas de las mismas.

Cuando observamos de cerca el panorama actual, nos encontramos con que las herramientas y dispositivos diseñados para ayudar a los operadores, aunque han avanzado en términos de diseño y ergonomía, todavía tienen limitaciones significativas en lo que respecta a la precisión y estabilidad necesarias para ciertas tareas. La mayoría de las herramientas manuales requieren un control minucioso, una calibración constante y una coordinación excepcional por parte del operador, sin obviar que cuentan con una curva de aprendizaje.

Debido a esto, la implementación de robots emerge como una solución invaluable para abordar tareas repetitivas, peligrosas y que requieren alta precisión. Los sistemas robóticos alcanzan su máximo potencial al integrar la visión por computadora, lo cual implica algoritmos diseñados para que el ordenador pueda extraer información valiosa de imágenes o vídeos. Existen diversas propuestas centradas en este ámbito, destacando la variedad de enfoques que se exploran para potenciar estas capacidades.

En este contexto, un ejemplo de estos sistemas sería el trabajo de Caiza [4] quien automatizó la recolección de muestras de un laboratorio mediante la implementación de visión artificial en un brazo antropomórfico con seis grados de libertad.

Dentro del área de visión por ordenador, existen diversas formas en la que ésta se aplica al control de los robots. Por ejemplo, en un trabajo realizado por Syakir [5] se presentó el control teleoperado de un brazo robótico empleando una cámara Kinect para obtener la profundidad de los objetos y así manipularlos adecuadamente.

Algo que estos trabajos tienen en común, y no ha sido mencionado directamente, es que las trayectorias que siguen los robots necesitan de algoritmos de una área conocida como planeación de movimiento, la cual se centra en encontrar los pasos requeridos para llegar de un punto a otro evadiendo obstáculos del ambiente mientras se busca satisfacer limitaciones en el torque de los motores o limitaciones angulares de las articulaciones [6]. Los algoritmos son complejos y computacionalmente intensivos, motivo por el cual escoger el adecuado es vital para el óptimo desarrollo del proyecto.

Esta situación llevó a Liu [7] a realizar un análisis de los principales algoritmos usados en este campo para mostrar las ventajas y desventajas de cada uno de estos, aplicados específicamente a brazos robóticos.

Para el manejo de robots es necesario distinguir las características propias de cada modelo, un ejemplo son los grados de libertad que son el número de articulaciones que posee el robot para su debido funcionamiento, el tipo de de articulaciones, su geometría, tipo de actuadores que usa, etc. [8].

También es importante considerar la programación de los robots, pues si no se emplea un framework común, la implementación de nuevos algoritmos se verá retrasada debido a adaptaciones continuas para hardware específicos. Como solución a esto se tiene ROS (Robot Operating System), el cual es una plataforma de software que simplifica el desarrollo y la programación de robots al proporcionar herramientas y recursos que permiten a los desarrolladores crear aplicaciones robóticas de manera más eficiente y efectiva [9].

Frente a la constante demanda de soluciones tecnológicas, el presente proyecto busca desarrollar la planeación de movimientos de un brazo robótico siguiendo el esquema delineado en la Fig. 1. En este enfoque, un ordenador dirige el brazo robótico a través del framework de ROS, aprovechando la retroalimentación proporcionada por una cámara de profundidad, y gracias a esto el sistema intenta conseguir la manipulación precisa de objetos pequeños.

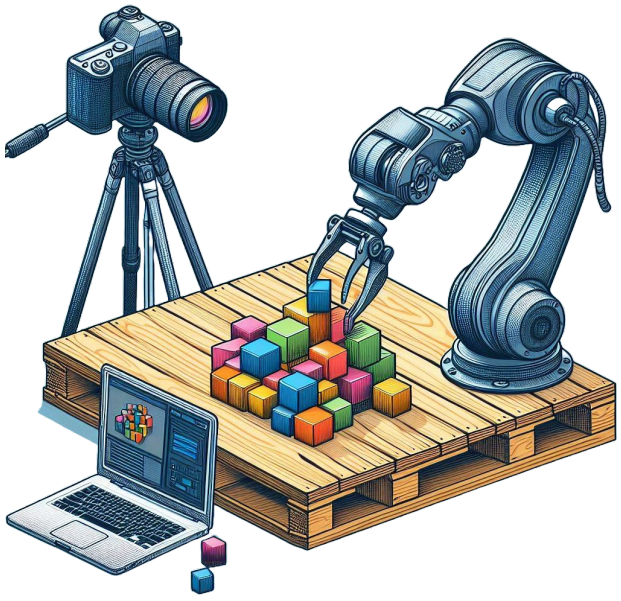

Fig. 1. Esquema general del sistema.

II. Metodología

A. Preparación del Sistema

El robot a usar es el modelo PincherX 150, diseñado por la empresa Trossen Robotics, que posee 5 grados de libertad y una carga útil de 50 gramos. Sin embargo, si se desea emplear el robot en el rango de movimiento completo, la carga se reduce a la mitad, es decir, 25 gramos. Debido a esto, los objetos a manipular cumplirán esta limitación. [10]. Los primeros pasos consistieron en la configuración de un entorno que sirva para programar el brazo robótico, el cual requiere la instalación de ROS Noetic en un computador con Linux Ubuntu 20.04 LTS. Para su uso inicial, se siguieron las indicaciones detalladas en la web del fabricante.

En el ámbito de visión artificial, se empleó la cámara de profundidad Intel D415 RealSense, la cual cuenta con un sensor RGB y uno infrarrojo para obtener la profundidad del entorno.

Para el uso adecuado del robot, éste debe estar fijado en una base firme, caso contrario, debido al peso, no se mantiene estable al estirar el brazo. Esto se logró con un pedazo de MDF como se muestra en la Fig. 2.

Fig. 2. Montaje del sistema.

Teniendo montado el sistema, se necesita calibrar la cámara con el robot para conocer la posición relativa de ambos. Esto se realiza mediante una etiqueta denominada AR tag colocada en el robot y una interfaz gráfica provista por el fabricante. Es recomendable hacer esta configuración cada vez que se vaya a usar el robot para indicar al sistema la posición relativa del brazo con la cámara. Una vez realizada la primera configuración, existe una calibración requerida para delimitar el área de interés y el procesamiento de la

nube de puntos obtenida de la cámara de profundidad.

Después de finalizar ambas calibraciones, el programa devuelve los objetos detectados dentro del área de trabajo como clústers de puntos, indicando los centros de masa de cada uno para la visualización en tiempo real, dentro de RVIZ (Herramienta de Visualización 3D para ROS) como se aprecia en la Fig. 3. Cada uno de estos centros de masa contiene información de su ubicación en el espacio, su color, el número de puntos que componen el clúster, entre otros.

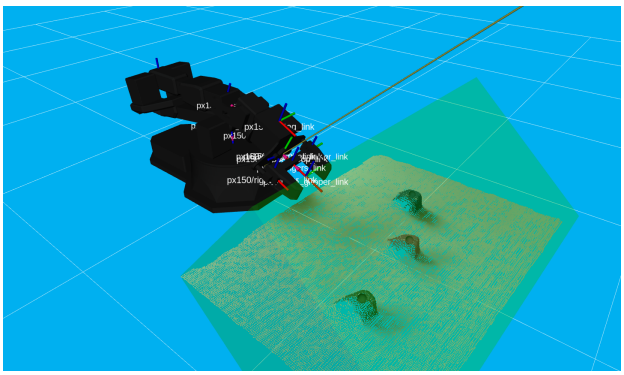

Fig. 3. Visualización de datos obtenidos de la cámara en RVIZ.

B. Planeación de Movimientos

Respecto al área de planeación de movimientos, una técnica empleada para el cálculo de las trayectorias se denomina C-space o espacio de configuración en español, la cual consiste en una transformación del espacio físico representando al robot como un punto y tomando en cuenta todas sus articulaciones y los obstáculos, formando así todas las posibles combinaciones de movimientos que el robot podría hacer [11]. En la Fig. 4 se muestra cómo se ve la transformación desde un espacio físico a C-space donde cada uno de sus ejes representa las articulaciones del robot. En caso de robots con mayor grado de libertad, no es posible graficar su representación en el C-space.

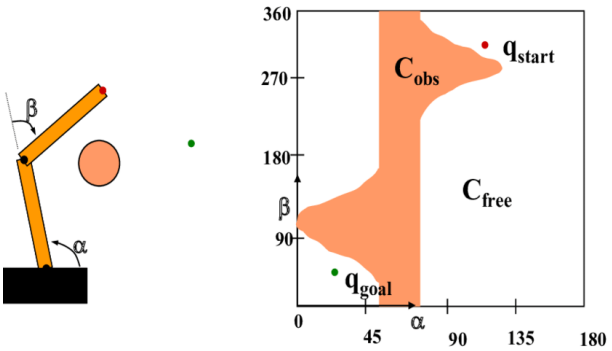

Fig. 4. Izquierda: vista física de un manipulador planar de 2 grados de libertad. Derecha: representación en C-space [1].

Debido a la construcción de los robots, sus articulaciones no pueden dar vuelta indefinidamente, tienen limitaciones en su movimiento. Esto es un detalle a considerar dentro del C-space, pues no podemos exceder estos límites sin ocasionar un daño irreversible al robot.

Tras haber definido el C-space, lo siguiente consiste en detallar el planeador de movimiento, el cual es un algoritmo que se encarga de movilizar cada una de las partes del robot para llegar a un objetivo específico de manera segura. Ejemplos de estos algoritmos son los siguientes:

1) RRT (Rapidly-Exploring Random Tree)
2) RRT-Connect
3) PRM (Probabilistic Roadmap)
4) SBL (Single-query Bi-directional Lazy collision checking planner)
5) KPIECE (Kinodynamic Planning by Interior-Exterior Cell Exploration)

En la Fig. 5 se muestra la manera en la que RRT va creando un árbol conforme va explorando el espacio disponible en búsqueda de un camino óptimo. Mientras que en la Fig. 6 se aprecia cómo PRM genera un muestreo del espacio y posteriormente busca el camino óptimo desde el punto de inicio al final.

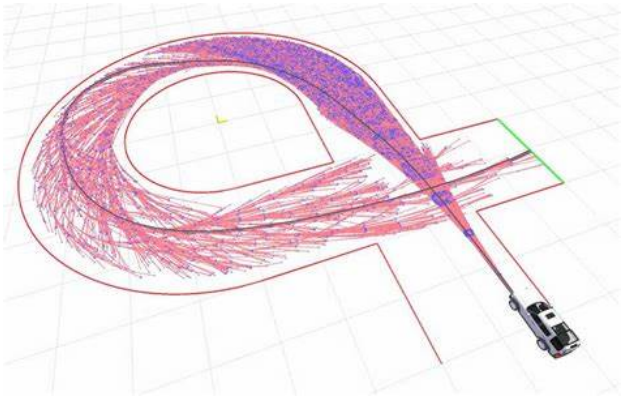

Fig. 5. Visualización de un ejemplo obtenido con RRT [2].

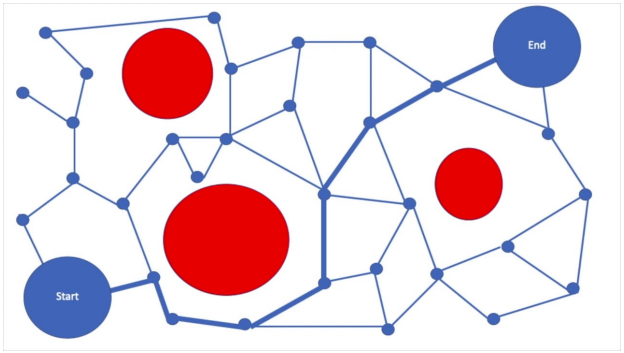

Fig. 6. Visualización de ejemplo de PRM [3].

C. Programación

La programación del robot se realiza en ROS, el cual posee una interfaz en Python que permite el desarrollo rápido de los programas gracias a su fácil sintaxis y la ayuda de librerías proporcionadas por el fabricante. Los códigos desarrollados tienen el nombre de scripts y esto es lo que ejecuta el robot.

Debido a la complejidad involucrada en la planificación de movimientos, el siguiente objetivo consistió en la selección del algoritmo adecuado; para ello, se llevó a cabo una metodología bibliográfica de los diferentes algoritmos disponibles en la literatura científica y aquellos disponibles en las librerías del fabricante. El desarrollador usa MoveIt (Plataforma de Manipulación Robótica para ROS) como backend del sistema y basa sus planeadores de movimientos en OMPL (Open Motion Planning Library) [12], el cual es un librería Open Source con diversos algoritmos avanzados integrados en ella.

Al revisar el repositorio GitHub que contiene la implementación de OMPL por parte del fabricante [13], se encontraron diversos algoritmos conocidos dentro del área de planeación de movimientos, ejemplos de los mismos son RRT, PRM, BiRRT, entre otros.

En la Fig. 7 se muestra el diagrama de funcionamiento del sistema, desde la ejecución del script de Python hasta el movimiento coordinado del brazo robótico. Teniendo este esquema como guía, se empezó con la elaboración de los scripts para coordinar los diversos componentes.

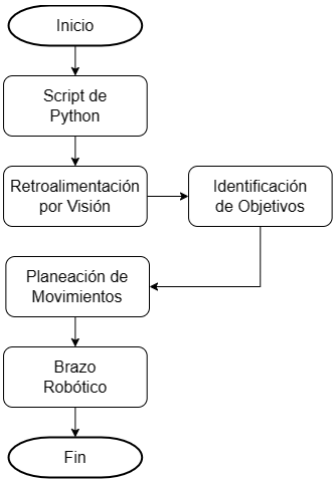

Fig. 7. Diagrama de flujo del sistema.

D. Pruebas

Para la prueba base, se buscó comprobar la calidad de la información que se recibe de la cámara a la vez que la precisión de los movimientos del robot en el entorno físico. Por ello, se realizó la tarea de apilar cubos formando una torre. En el Algoritmo 1 se muestra el pseudocódigo del programa diseñado para esta tarea.

Algoritmo 1 Pseudocódigo para apilar cubos

```
Instanciar Robot, Camera
Clusters ← Camera.get_clusters
for cluster_i ∈ Cluster do
    if cluster_1 then
        target ← cluster.location
    else
        height ← target.height
        Move arm to cluster_i.location
        Close gripper
        Move arm to (target, height)
        Open gripper
```

Para verificar el funcionamiento de los planeadores de movimientos, se realizaron pruebas que consistieron en la introducción de obstáculos de modo que el robot tome un camino que no cause daños a sí mismo ni al entorno. Estos algoritmos se evalúan con métricas como la correcta generación de un camino, la longitud del mismo, el tiempo de cálculo o el tiempo de ejecución.

En base al análisis de los planeadores de movimiento, se escogió realizar la evaluación usando RRT-Connect y PRM, pues estos son algoritmos representativos de diversos tipos de planeadores, basados en trayectoria y basados en celdas, respectivamente, siendo RRT-Connect el planeador por defecto en OMPL.

RRT basa su funcionamiento en la generación de un árbol desde el punto de partida, el cual se expande buscando el objetivo, como se demuestra en la Fig. 5. RRT-Connect difiere al emplear dos árboles que buscan encontrarse, uno desde el punto de partida y otro desde el destino. El pseudocódigo se detalla en el Algoritmo 2.

Algoritmo 2 Pseudocódigo RRT-Connect

```
Require: C-space
Require: n = iterations limit
Tree_O ← Tree.seed(Origin)
Tree_T ← Tree.seed(Target)
while iteration ≤ n do
    sample ← c_space.random()
    if not Tree_O collision towards sample then
        Tree_O grows to sample
    if not Tree_T collision towards sample then
        Tree_T grows to sample
    if Tree_O intersects Tree_T then
        Return path between Tree_O, Tree_T
```

El algoritmo de PRM comienza muestreando el C-space mediante puntos aleatorios y determina si están en una región libre, rechazando a aquellos que no. Finalizada esta etapa, se unen los puntos que no presenten colisión en su ruta.

A continuación, se determina el punto de inicio y final y mediante un algoritmo de búsqueda en nodos se busca la solución óptima. La Fig. 6 muestra una visualización de PRM.

Una implementación de PRM en pseudocódigo de la etapa de muestreo se muestra en el Algoritmo 3. Una vez acabado el muestreo, se usa un algoritmo de búsqueda en nodos para determinar el trayecto óptimo, como puede ser el algoritmo de Dijkstra [14], la búsqueda A* [15], entre otros.

Algoritmo 3 Pseudocódigo PRM

Require: C-space
Require: n = number of points to sample
  $Vertices \leftarrow \varnothing$
  $Edges \leftarrow \varnothing$
  while $Vertices.amount \leq n$ **do**
    $random \leftarrow c\_space.random()$
    $Vertices.append(random)$
    $Neighborhood \leftarrow nodes.close\_to(random)$
    $Neighborhood.sort(distance(random))$
    for $neighbor \in Neighborhood \notin Edges$ **do**
      if collision free $(random, neighbor)$ **then**
        $Edges.append(random, neighbor)$
  Return $(Vertices, Edges)$

Para la coordinación entre los sistemas de visión y de planeación de movimientos, se elaboró un script de Python que escucha el nodo que recibe la nube de puntos de la cámara y almacena estos valores como archivos .stl. A su vez, se corre el script para la detección de objetivos de modo que se tenga la posición de todos los objetos de interés. Con esto, se cambia al módulo de planeación y se importa el fichero .stl en RVIZ como una malla, y se procede a realizar la planeación de movimientos de manera normal.

III. Resultados

Al finalizar la elaboración de los scripts de las pruebas, se procedió a verificar la implementación de los mismos, de modo que se compruebe la sinergia entre cada uno de los sistemas empleados. Cada una de las pruebas fue realizada varias veces con el fin de obtener estadísticas y minimizar el error de medición.

A. Prueba Base

En la TABLA I se presentan los resultados de la prueba base, donde se aprecia que tanto el tiempo de ejecución como el tiempo de programa suben linealmente en función del número de cubos. Una foto de un resultado exitoso se aprecia en la Fig. 8.

Luego de calcular la media aritmética de los tiempos de ejecución en relación con el número de cubos, se generó el gráfico que se muestra en la Fig. 9.

TABLA I
Resultados de la prueba base

Número de cubos	Tiempo de ejecución	Tiempo de programa	Éxito
4	48,7	54,5	Sí
4	48,8	54,6	Sí
4	48,7	54,5	No
5	64,9	70,7	Sí
5	64,9	70,5	Sí
6	81.2	88.0	Sí

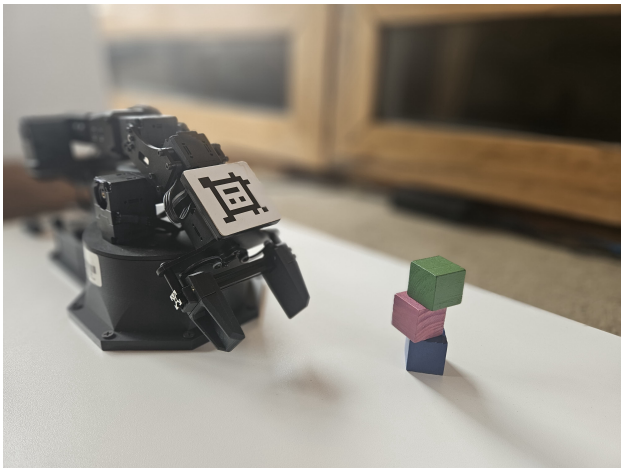

Fig. 8. Resultado exitoso de la prueba de apilar cubos.

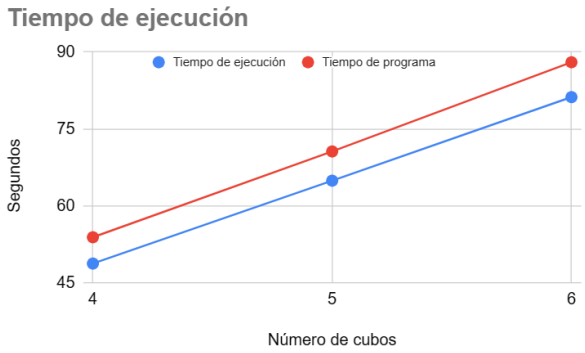

Fig. 9. Relación número de cubos y tiempos de ejecución.

B. Prueba en Simulador

Para comparar los planeadores de movimiento, se evaluaron los mismos en un ambiente simulado dentro de RVIZ, con objetos que bloquean el camino del robot. Se crearon 2 escenarios virtuales que se muestran en la Fig. 10 y Fig. 11. El robot de color negro representa la posición inicial, mientras que el naranja indica la posición objetivo.

En cada escenario se realizaron 20 repeticiones de la planeación de movimientos usando ambos algoritmos y en base a los resultados se generó un diagrama de caja y bigotes para cada escenario, como se muestra en la Fig. 12 y la Fig. 13.

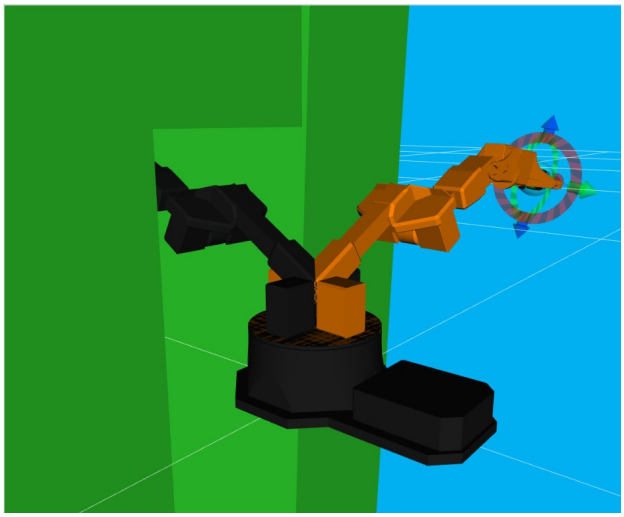

Fig. 10. Escenario 1 para pruebas.

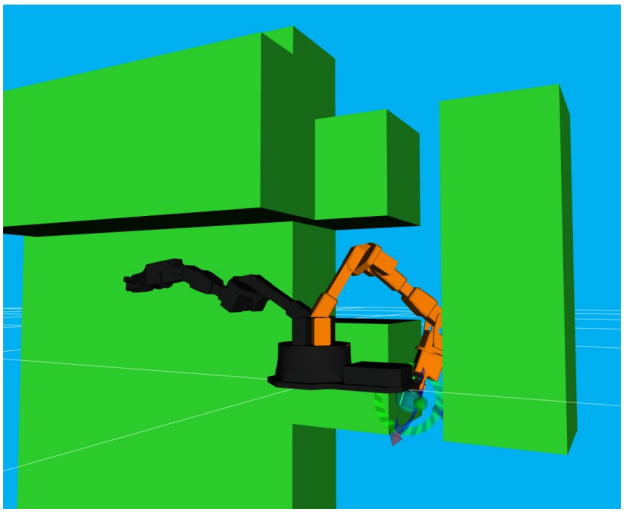

Fig. 11. Escenario 2 para pruebas.

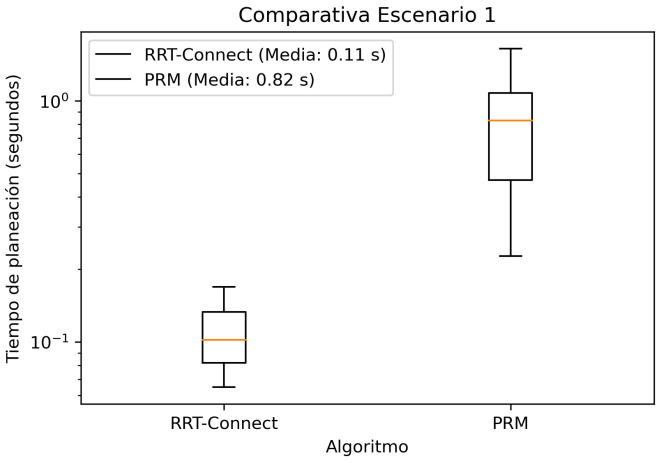

Fig. 12. Estadísticas del escenario simulado 1. Eje Y logarítmico.

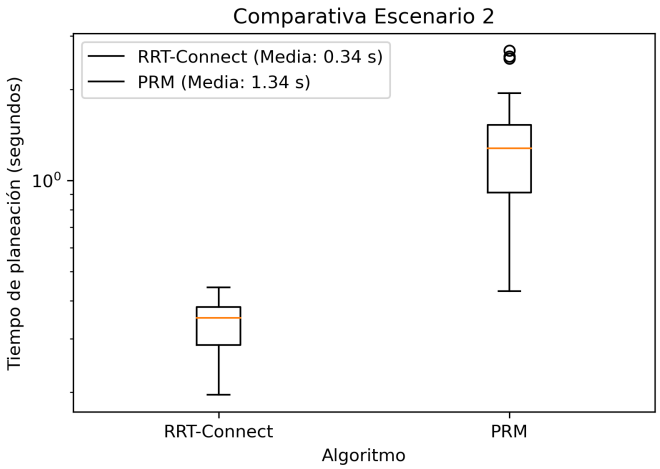

Fig. 13. Estadísticas del escenario simulado 2. Eje Y logarítmico.

C. *Prueba en entorno Físico*

Finalizada la etapa de simulación, se desarrolló un ambiente físico con obstáculos para evaluar el rendimiento de los planeadores seleccionados. El escenario físico se aprecia en la Fig. 14 y los datos obtenidos digitalmente se visualizan en RVIZ en la Fig. 15.

Fig. 14. Escenario físico con obstáculo.

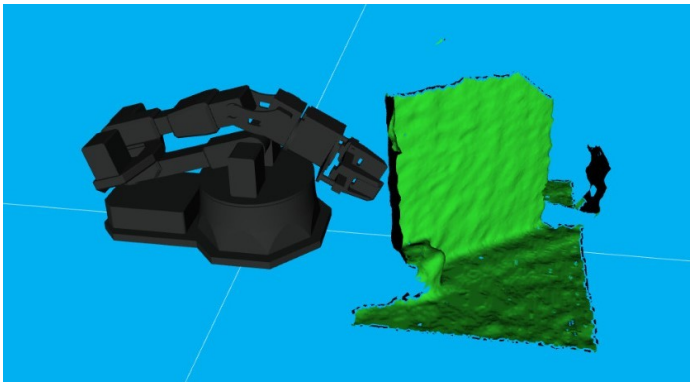

Fig. 15. Escenario con obstáculo detectado, en RVIZ.

La Fig. 16 presenta los resultados del tiempo de cálculo para la planeación de movimientos mientras que la Fig. 17 muestra los resultados de la ejecución de las trayectorias obtenidas.

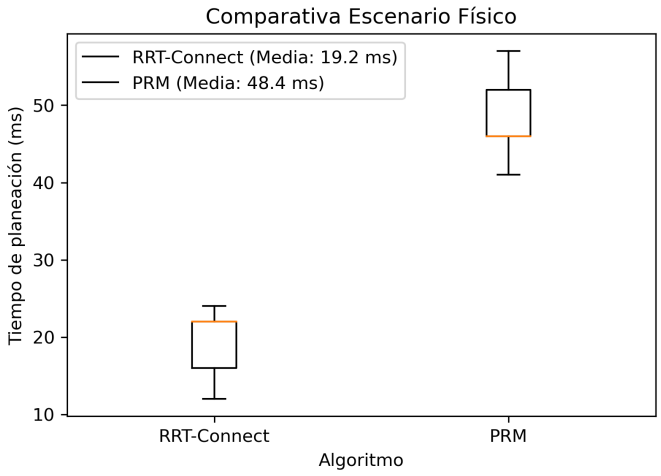

Fig. 16. Tiempos de planeación en sistema físico.

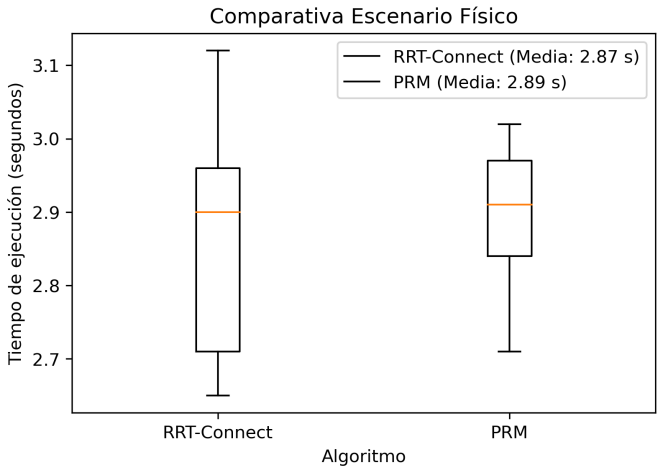

Fig. 17. Tiempos de ejecución en sistema físico.

La siguiente prueba consistió en colocar un objeto que obligue un camino específico para el brazo robótico. Su montaje se muestra en la Fig. 18 y su representación en RVIZ en la Fig. 19. Finalmente, los resultados obtenidos se indican en la Fig. 20 y la Fig. 21, los cuales muestran los tiempos obtenidos en el proceso de planeación y el de ejecución, respectivamente.

Fig. 18. Montaje para prueba 2

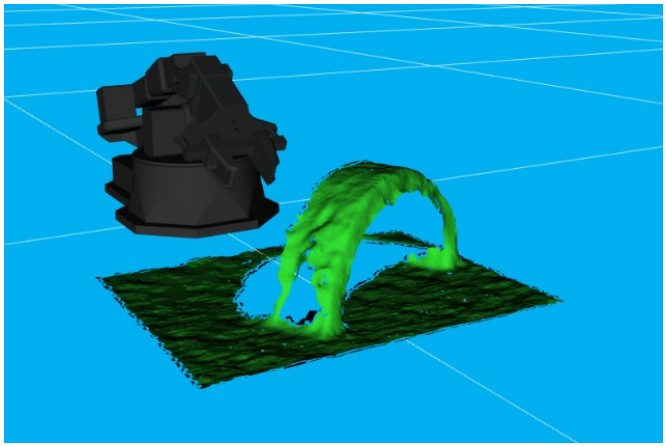

Fig. 19. Representación en RVIZ del montaje 2.

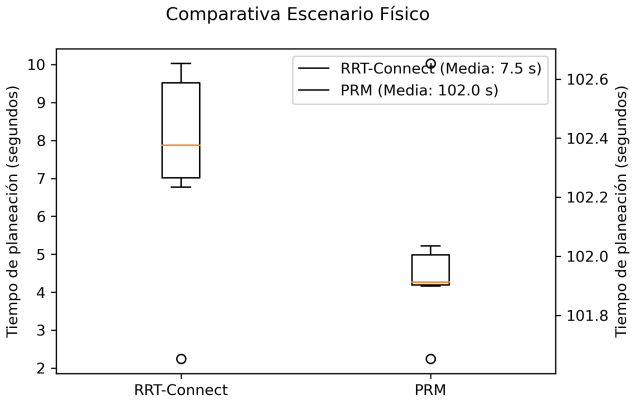

Fig. 20. Tiempos de planeación del escenario 2

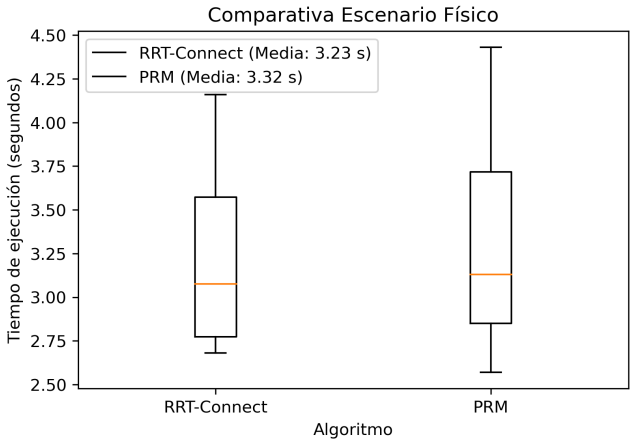

Fig. 21. Tiempos de ejecución del escenario 2

IV. Conclusiones

En el primer escenario físico, al ser considerablemente simple, los tiempos de cálculo fueron sumamente rápidos con una media de 19.2 ms para RRT-Connect y de 48.4 ms para PRM. En el caso del segundo escenario, al limitar el espacio libre de forma tan drástica, los tiempos de cálculo aumentaron considerablemente: RRT-Connect con una media de 7.5 segundos y PRM con una media de 102 segundos.

En todas las pruebas realizadas, PRM toma un tiempo superior en la parte de planeación. Esto es particularmente notorio en la última prueba realizada en el entorno físico, pues al definir como objetivo un punto rodeado por un obstáculo, la cantidad de nodos que debe generar PRM se incrementa, así como también el tiempo necesario para la búsqueda de la trayectoria óptima. En el caso de RRT-Connect, al generar árboles, el costo computacional no aumenta de forma tan marcada.

Los tiempos obtenidos en la ejecución de trayectorias son similares, tanto PRM como RRT-Connect generan tiempos comparables, con una pequeña ventaja de RRT-Connect aunque siendo la diferencia mínima: apenas un 0.7% más rápido en el escenario físico 1 y un 2.7% en el caso del escenario físico 2.

El presente trabajo ha demostrado cómo la integración de la visión por computador permite adaptar los sistemas robóticos a una mayor variedad de situaciones, volviéndolos más versátiles y ampliando sus aplicaciones en diversos campos. Los sistemas de visión se han vuelto indispensables en muchas áreas, pues los algoritmos usan la información tridimensional obtenida mediante cámaras de profundidad para generar la ruta a la posición deseada, esquivando los obstáculos presentes.

Una vez finalizado el presente trabajo, se identificaron situaciones que podrían ser mejoradas, así como observaciones para obtener mejores resultados en futuras aplicaciones del sistema.

El primer punto a detallar es la colocación de la cámara: se debe emplear una estructura más estable de modo que no exista movimientos sobre la cámara. De esta forma, el sistema no perderá la calibración realizada por movimientos inesperados.

Es importante considerar la carga máxima del robot, que al ser apenas de 50 gramos, limita su uso como manipulador. Es recomendable evitar mantenerlo en una posición estirada horizontalmente por más de unos segundos, puesto que se sobrecalientan los motores, lo cual podría generar un daño permanente en los mismos.

Respecto a la combinación de los módulos de visión y planeación, la obtención de la matriz de transformación del marco de referencia de la cámara respecto a la base no se pudo realizar de manera automática, y fue necesario obtener la matriz de la cámara respecto a la base y la matriz de la base respecto a la cámara. Luego, se extrajo la traslación de uno y las rotaciones del otro debido a una incompatibilidad interna en las librerías de Python para esta tarea.

Referencias

[1] A. Gasparetto, P. Boscariol, A. Lanzutti, and R. Vidoni, "Path planning and trajectory planning algorithms: A general overview," *Motion and Operation Planning of Robotic Systems: Background and Practical Approaches*, pp. 3–27, 2015.

[2] K. R. Group, "Rrt* algorithm on a race car (270 degree turn)," Mayo 2011. [Online]. Available: https://www.youtube.com/watch?v=p3nZHnOWhrg

[3] J. Bernatchez, "Motion planning algorithms (rrt, rrt*, prm)," Diciembre 2020. [Online]. Available: https://www.youtube.com/watch?v=gP6MRe_IHFo

[4] O. Caiza, W. G. Aguilar, P. Albán, and Y. Fernández, "Kinect and manipulator-based sample collection system for military robot," in *Developments and Advances in Defense and Security: Proceedings of MICRADS 2020*. Springer, 2020, pp. 75–87.

[5] M. Syakir, E. S. Ningrum, and I. A. Sulistijono, "Teleoperation robot arm using depth sensor," in *2019 International Electronics Symposium (IES)*. IEEE, 2019, pp. 394–399.

[6] K. M. Lynch and F. C. Park, *Introduction to Robotics: Mechanics, Planning, and Control*. Cambridge University Press, 2017.
[7] S. Liu and P. Liu, "A review of motion planning algorithms for robotic arm systems," in *RiTA 2020: Proceedings of the 8th International Conference on Robot Intelligence Technology and Applications*. Springer, 2021, pp. 56–66.
[8] K. M. Lynch and F. C. Park, "2.2. degrees of freedom of a robot – modern robotics," Junio 2023. [Online]. Available: https://modernrobotics.northwestern.edu/nu-gm-book-resource/2-2-degrees-of-freedom-of-a-robot
[9] "Ros: Why ros?" Marzo 2023. [Online]. Available: https://www.ros.org/blog/why-ros/
[10] T. Robotics, "Pincherx-150 — interbotix x-series arms documentation," Febrero 2024. [Online]. Available: https://docs.trossenrobotics.com/interbotix_xsarms_docs/specifications/px150.html
[11] R. E, "Robotics: Useful definitions," 1999. [Online]. Available: https://cs.stanford.edu/people/eroberts/courses/soco/projects/1998-99/robotics/definitions.html
[12] I. A. Sucan, M. Moll, and L. E. Kavraki, "The open motion planning library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, 2012.
[13] Interbotix, "Ompl planning · github," Julio 2021. [Online]. Available: https://github.com/Interbotix/interbotix_ros_arms/blob/master/interbotix_moveit/config/ompl_planning.yaml
[14] Wikipedia, "Dijkstra's algorithm," Mayo 2024. [Online]. Available: https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm
[15] N. J. Nilsson, *The quest for artificial intelligence*. Cambridge University Press, 2009.