UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Desarrollo de un sistema de visión para un robot industrial KUKA KR 5-2 (Reconocimiento y Simulación)

TESIS

Que para obtener el título de

Ingeniero Mecatrónico

P R E S E N T A

Diego Alberto Nava Arsola

DIRECTOR DE TESIS

Ing. Román Victoriano Osorio Comparán

Ciudad Universitaria, Cd. Mx., 2022

Agradezco a…

A mis padres Felipe y María Concepción por haberme dado la vida, así como su amor, comprensión, sacrificio y apoyo en cada momento de mi vida. No existe mayor orgullo que ser su hijo.

A mis hermanas Nancy Gabriela y Nadia Edith por ser mis mejores amigas y por siempre estar ahí en las buenas y en las malas. El amor que siento por ustedes es inefable.

A la Universidad Nacional Autónoma de México (UNAM) y a la Facultad de Ingeniería (FI) por mi formación académica y humanística.

Al Ing. Román Osorio por permitirme ser parte de este proyecto y por ayudarme en todo lo que he necesitado.

A Kevin por ser mi compañero en la realización de este proyecto y por ser un gran amigo. Esto no podría haber sido posible sin tu ayuda.

A Alex, Efraín, Fernando, Magaly y Mario por su amistad y por todos los momentos que hemos vividos juntos.

Al Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas (IIMAS) por prestarnos sus instalaciones para la realización de este proyecto.

旅に病んで
夢は枯野を
かけ廻る
〃松尾芭蕉

Resumen

Un sistema de visión es un conjunto de elementos que tienen la capacidad de realizar la detección o reconocimiento de objetos. Estos sistemas son importantes en el ámbito de la robótica ya que les confieren a robots la capacidad de interactuar con su entorno de manera más orgánica y flexible. Sin embargo, no todos los robots cuentan con un sistema de visión, por esta razón el objetivo de la tesis fue desarrollar un sistema de visión por computadora que permitiese a un robot industrial KUKA KR5-2 arc hw de seis grados de libertad realizar tareas de identificación y manipulación de objetos.

Para realizar esto se utilizó un Kinect 2.0 y ROS para obtener la imagen y la profundidad de los objetos. Posteriormente, esta información fue procesada por los algoritmos de reconocimiento de objetos: FAST-ORB y BRISK, así como un detector de objetos basado en los algoritmos RANSAC y PCA. Las trayectorias fueron calculadas con el uso de la librería *Moveit* para finalmente realizar una simulación del robot en *Gazebo*.

Aunado a lo anterior se realizaron diversas pruebas (con objetos de frente y varios ángulos, con objetos sobrepuestos y con objetos horizontales) para determinar la exactitud y precisión de dichos algoritmos.

A partir de dichas pruebas se concluye que en general los algoritmos son precisos y exactos para calcular la posición, sin embargo, son inestables al calcular la orientación de los objetos. El algoritmo FAST-ORB es el mejor al haber varios objetos en escena y al reconocer objetos sobrepuestos; BRISK presentó un peor desempeño, pero fue capaz de reconocer objetos horizontales; RANSAC-PCA fue el más estable y el que menos error presentó, pero solo es capaz de detectar un objeto a la vez.

Palabras clave: Visión por computadora; Reconocimiento de objetos; Procesamiento de imágenes; Robot Operating System; Kinect; Robot industrial; Planificación de movimiento; Simulación.

Abstract

A computer vision system is a set of elements that are capable of detecting or recognizing objects. These systems are important in the robotics field because they allow robots the ability to interact with their environment in a way more flexible and organic. However, not all robots have one of these systems, for this reason the main objective of this thesis was to develop a computer vision system so that a 6DOF industrial robot KUKA KR5-2 arc hw could perform object identification and object manipulation tasks.

To do that a Kinect 2.0 as well as the framework *ROS* were used to obtain images and depth values from certain objects. Subsequently, this information was processed by the object recognition algorithms FAST-ORB and BRISK, as well as an object detector based in the RANSCAC and PCA algorithms. The trajectories of the robot were computed with the *ROS* library *Moveit* and finally a simulation of the robot was performed in *Gazebo*.

Coupled with the above, several tests were performed (headed on and turned objects, overlapped and lied down objects) in order to determine precision and accuracy of said algorithms.

Based on the tests we conclude that in general the algorithms are precise and accurate to estimate the object's position, nevertheless they are unstable when estimating the orientation of the objects. FAST-ORB has the best performance when several objects are in scene and when objects are overlapping; BRISK had the worst performance but it was capable of recognizing objects that were laid down; RANSAC-PCA was the most stable and the one with less error, but it is only able to detect one object at a time.

Keywords: Computer vision; Object recognition; Image processing; Robot Operating System framework; Kineck; Industrial robot; Motion planning; Simulation.

Índice general

Capítulo 1 Introducción 1

1.1. Planteamiento del problema 1

1.2. Estado del arte 2

1.3. Objetivos 3

1.4. Hipótesis 4

Capítulo 2 Marco teórico 5

2.1. Descripción de objetos en el espacio 5

2.2. Robótica 6

2.2.1. Clasificación de robots 8

2.3. Robots industriales 9

2.3.1. Cinemática de robots industriales 12

2.4. Visión por computadora 15

2.4.1. Reconocimiento de objetos 16

2.4.2. Imágenes digitales 16

2.4.3. Detección y descripción de características 17

2.4.4. Espacio - escala 18

2.4.5. Algoritmo RANSAC 20

2.4.6. Análisis de Componentes Principales (PCA) 22

2.5. Modelos y simulaciones 23

Capítulo 3 Robot manipulador KUKA KR5-2 arc hw 25

3.1. Características del robot 25

3.1.1. Efector final 28

3.2. Cinemática directa 29

3.2.1. Parámetros de Dénavit-Hartenberg 29

3.2.2. Descripción con el formato URDF (ROS) 30

3.3. Cinemática inversa 32

Capítulo 4 Reconocimiento de objetos 33

4.1. Kinect 2 33

4.2. Robot Operating System (ROS) 35

4.2.1. Paquetes ... 35
4.2.2. Nodos ... 35
4.2.3. Tópicos ... 35
4.2.4. Servicios ... 36
4.2.5. Mensajes ... 36
4.3. Reconocimiento de objetos con ROS ... 37
4.3.1. Librería *iai_kinect2* ... 37
4.3.2. Librería *find_object_2d* ... 37
4.3.3. Algoritmos utilizados ... 37
4.3.3.1. FAST (Feature from Accelerated Segment Test) ... 38
4.3.3.2. BRIEF (Binary Robust Independent Elementary Features) ... 38
4.3.3.3. ORB (Oriented FAST and Rotative BRIEF) ... 39
4.3.3.4. BRISK (Binary Robust Invariant Scalable Keypoints) ... 40
4.3.3.5. Detector RANSAC-PCA ... 43
Capítulo 5 Simulación ... 46
5.1. Rviz ... 46
5.2. Gazebo ... 46
5.3. Simulación de la cinemática inversa ... 47
5.3.1. Moveit ... 47
Capítulo 6 Metodología ... 49
6.1. Calibración del Kinect 2 ... 49
6.2. Modelo URDF y SRDF ... 50
6.3. Pruebas iniciales con programas ... 51
6.3.1. YOLO (You Only Look Once) ... 51
6.3.2. ORK (Object Recognition Kitchen) ... 52
6.3.3. find_object_2d ... 52
6.4. Diagrama de los programas ... 52
6.6. Metodología de las pruebas ... 54
6.6.1. Pruebas de los algoritmos de reconocimiento de objetos ... 55
6.6.1.1. Determinación del espacio de trabajo. ... 55
6.6.1.2. Pruebas objetos de frente y con rotaciones negativas y positivas. ... 56
6.6.1.3. Pruebas con objetos sobrepuestos. ... 63

6.6.1.4. Pruebas con objetos horizontales 65
6.2 Simulación 66
Capítulo 7 Resultados y discusión 67
7.1. Calibración del Kinect 2.0 67
7.2. Espacio de trabajo 69
7.3. Pruebas con objetos de frente a 0° y con rotaciones negativas y positivas 69
7.3.1. Objetos a 0° 69
7.3.2. Objetos con giro negativo 74
7.3.3. Objetos con un giro positivo 78
7.3.4. Resultados por objeto 82
7.4. Pruebas con objetos sobrepuestos 83
7.5. Pruebas con objetos horizontales 89
Capítulo 8 Conclusiones y trabajo a futuro 93
Referencias 95

Capítulo 1 Introducción

Actualmente, la demanda en la producción industrial ha aumentado debido a factores como la globalización y la urbanización, lo cual ha conllevado al aumento en las últimas décadas en la búsqueda y desarrollo de nuevas formas de innovación en las líneas de producción, proceso conocido como la cuarta revolución industrial o industria 4.0. Esta última es una iniciativa hecha por el gobierno alemán en 2011 en la feria de Hannover [1] para implementar nuevas tecnologías en la actividad industrial, tales como el internet de las cosas (IoT), inteligencia artificial, robótica, *big data,* aprendizaje automático, computación en la nube, procesamiento de datos en tiempo real, interconectividad, manufactura aditiva, entre otras.

Una consecuencia de la industria 4.0 en el ámbito de la robótica es la búsqueda de la flexibilidad en los procesos que involucran robots. Por ejemplo, se han implementado nuevas tecnologías como el uso de la visión por computadora o el desarrollo de robots colaborativos. La visión por computadora y el reconocimiento de objetos resultan muy importantes ya que proveen a los sistemas industriales un elemento de percepción semejante al humano, lo que les confiere una gran capacidad adaptativa, así como una precisión y exactitud capaces de superar las humanas.

Sin embargo, debido a la extensión en el uso de robots en la industria es necesario mejorar los estándares de seguridad, ya que su uso puede poner en riesgo la integridad de las personas que los operan. En este sentido, la visión artificial y las simulaciones puede ayudar a mejorar la seguridad de estos sistemas ya que permiten predecir el comportamiento del robot antes de ejecutar cualquier operación y así prevenir posibles daños humanos y económicos.

1.1. Planteamiento del problema

En las últimas décadas México se ha establecido como una potencia industrial, en la que destaca la industria manufacturera. Esto debido a la apertura del mercado y la inversión de empresas extranjeras en nuestro país. Sin embargo, si bien esto ha impulsado la economía nacional, al mismo tiempo es un factor que retrasa el desarrollo de tecnología. En los últimos 20 años el porcentaje promedio de patentes nacionales ha sido del 3% respecto al total de patentes desarrolladas en el país [2]. Por ello es necesario desarrollar tecnología nacional para que el mercado mexicano pueda competir con el extranjero y así impulsar las economías locales, sin embargo, el costo y la dificultad de implementación de

las nuevas tecnologías sitúa en desventaja a las pequeñas y medianas empresas. Para resolver este problema se propone el uso de tecnologías libres de código abierto.

En la actualidad, en la industria está normalizado utilizar sistemas robóticos con movimientos manuales o programados. Estos sistemas son suficientes en industrias de ensamble o de fabricación de productos en masa, pero fallan al ofrecer flexibilidad en los procesos. Esto último es importante, ya que el avance de la industria a nivel mundial requiere un grado de flexibilidad mayor que los sistemas tradicionales no pueden ofrecer. El uso de la visión artificial se presenta como un factor clave a la hora de flexibilizar procesos, ya que emula el sistema humano de percepción y manipulación de objetos.

1.2. Estado del arte

En 2019, Dos Reis Douglas et al. [3] propusieron un sistema de visión artificial basado en el algoritmo YOLO [4] para identificar objetos que pudieran ser considerados estáticos por un robot móvil. Para ello utilizaron un Kinect 2 y una tarjeta gráfica Nvidia Jetson TX2 para mejorar el rendimiento del algoritmo. Sus resultados indicaron que los cálculos de la profundidad obtenidos por el Kinect 2 presentaban un error por debajo del 3.64%, una vez que se había entrenado al algoritmo con una base de datos con los obstáculos predefinidos.

Por su parte, Ji Yang Lee y Cheol-soo Lee en 2018 [5] planearon los movimientos de un robot SCARA utilizando marcadores y el etiquetado de estos. El método utilizado para la planeación de trayectorias determina la posición y orientación de los marcadores, y los reconoce para alcanzar los puntos objetivo. La detección de los marcadores fue hecha con el algoritmo SURF (*Speeded-Up Robust Features*) [6] y para obtener las posiciones en tres dimensiones de los marcadores utilizaron un Kinect.

Andhare y Rawat, en 2016 [7], fueron capaces de controlar un robot industrial de 6 grados de libertad, para hacer operaciones de *pick and place* mediante visión por computadora. Ellos utilizaron una cámara para obtener las imágenes de los objetos y *OpenCV* para hacer el reconocimiento de estos. Dado que no utilizaron un sensor 3D, trabajaron solo sobre un plano, y las coordenadas y posiciones de los objetos se tuvieron que estimar mediante transformaciones lineales.

En 2020, Esa Apriaskar et al. [8] diseñaron un sistema de visión artificial basado en color para que un robot de cuatro grados de libertad hiciera tareas de ordenamiento de objetos. Para ello utilizaron un Kinect como sensor de color y profundidad, un Arduino MEGA 2560 como controlador y el software OpenCV como entorno de desarrollo para el algoritmo de visión. Sus resultados experimentales muestran un error promedio de las posiciones de efector final en el eje x, y y z de 5.83%, 5.89% y 8.59% respectivamente.

Por otra parte, en 2019, O. Glaufe et al. [9] desarrollaron un controlador PID para un brazo robótico OWI-535 de cuatro grados de libertad integrado a un sistema de visión por computadora usando un Kinect. El sistema captura objetos entre 6 posibles posiciones según su color. Utilizaron OpenNI y OpenCV para el procesamiento de las imágenes. Sus resultados fueron satisfactorios, ya que el porcentaje mínimo de capturas exitosas entre todas las pruebas fue del 83.33%

Finalmente, Vázquez en 2018 [10] desarrolló un sistema de detección de objetos para el robot de servicio *Justina* [11]. Para esto, utilizó un Kinect 2 conectado a una computadora con ROS. La detección de objetos la realizó en diferentes partes, primero con la segmentación de planos haciendo uso del algoritmo RANSAC, después utilizó un algoritmo para extraer los objetos, y finalmente la posición y orientación de los objetos se calculó con un análisis de componentes principales (PCA). Con la información de los objetos detectados, se calculó la cinemática inversa de los manipuladores de 7 grados de libertad que actúan como las manos del robot. Él hizo dos cálculos de la cinemática: uno con el paquete de ROS llamado Moveit y otro con un algoritmo geométrico. Sus resultados fueron éxitos, ya que el error en la segmentación de planos fue del 8.83% y en el caso del cálculo de la orientación de los objetos con el algoritmo PCA, el rango de error no pasó de los 5°. Está información permitió mejorar la manipulación *Justina* de un 21,4% a 72.8%. El único inconveniente de este trabajo es que no existe ninguna forma de reconocer los objetos, por lo que falla al haber más de un objeto en escena.

1.3. Objetivos

Objetivo general

Desarrollar un sistema de visión por computadora capaz de reconocer objetos con el uso de un Kinect 2.0 para que un robot industrial KUKA KR5-2 arc hw realice tareas de manipulación con dichos objetos.

Objetivos específicos

- Implementar un sistema de visión por computadora a través del uso de un Kinect 2 y el framework para robótica llamado ROS para que calcule la posición y orientación de determinados objetos.
- Observar el desempeño de los algoritmos de visión ante cambios de orientación sobre el eje de los objetos, y con objetos sobrepuestos y horizontales.
- Simular con el programa Gazebo el comportamiento del robot.

1.4. Hipótesis

- Un sistema de visión artificial hará posible conocer la posición y orientación de objetos.
- Objetos ubicados atrás de otros y objetos horizontales serán más difíciles de reconocer.
- Es posible simular el comportamiento del robot con el uso de software.

Capítulo 2 Marco teórico

2.1. Descripción de objetos en el espacio

La descripción de un objeto en el espacio se refiere a los parámetros que definen la posición y la orientación de dicho objeto referido a un cierto sistema de referencia. La posición de un objeto en un espacio tridimensional se representa mediante un vector de 3x1 llamado vector de posición [12]. Si bien este vector se puede definir en cualquier tipo de sistema de coordenadas como cilíndricas o esféricas, usualmente en robótica se utilizan sistemas de coordenadas cartesianas. De esta manera, un vector de posición P referido al sistema de referencia A queda definido como:

$$^{A}P = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}$$

Por otro lado, para definir la orientación de un objeto es necesario colocar un sistema de referencia relativo en la posición del objeto, referido a un sistema de referencia global o absoluto y luego obtener las relaciones geométricas entre ambos sistemas. Estas relaciones se representan mediante una matriz de 3x3 llamada matriz de rotación. Las columnas de esta matriz representan los vectores unitarios que definen las direcciones de los ejes del sistema de referencia relativo. De esta manera, una matriz de rotación R representa una rotación entre el sistema de referencia absoluto A y el sistema relativo B:

$$^{A}_{B}R = [^{A}_{B}\hat{X} \quad ^{A}_{B}\hat{Y} \quad ^{A}_{B}\hat{Z}] = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

Si bien la descripción de un objeto sirve para ubicar un objeto en el espacio, no ayuda a representar las transformaciones de posición u orientación que dicho objeto puede tener. Para ello se utilizan las matrices de traslación y de rotación homogéneas.

Una traslación es el movimiento de un punto en una cierta distancia siguiendo la dirección de un vector director. De esta manera, si se desea trasladar un punto $P = [x_0\ y_0\ z_0\ 1]^T$ a una nueva posición $Q = [x_1\ y_1\ z_1\ 1]^T$, siguiendo al vector director $u = [u_x\ u_y\ u_z]^T$, se debe multiplicar P con una matriz de traslación homogénea $T(u)$ que tiene la siguiente forma:

$$T(u) = \begin{bmatrix} 1 & 0 & 0 & u_x \\ 0 & 1 & 0 & u_y \\ 0 & 0 & 1 & u_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Por la tanto, la ecuación matricial que representa la traslación del punto P siguiendo el vector u es:

$$Q = T(u) * P$$

Por su parte, una rotación es el cambio en la orientación de un vector alrededor de un eje en un determinado ángulo. En el espacio cartesiano se tienen tres ejes principales (x, y, z), por lo tanto, existen tres matrices homogéneas de rotación [13], estas son:

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

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

Entonces, de forma análoga a la traslación, para rotar un vector en un cierto ángulo se debe multiplicar dicho vector con una matriz de rotación. Por ejemplo, para rotar un vector $v = [v_x \; v_y \; v_z \; 1]^T$ sobre el eje X de un sistema de referencia en un ángulo θ, se efectúa la siguiente operación (donde $v' = [v'_x \; v'_y \; v'_z \; 1]^T$ es el vector rotado):

$$v' = R_x(\theta) * v$$

Para representar varios cambios de posición y orientación de un objeto, se multiplican las matrices de rotación y traslación tomando en cuenta, que, el orden en el que se llevan a cabo las transformaciones afecta la posición y orientación de dichos objetos. Por ejemplo, no es lo mismo trasladar un objeto en una cierta dirección y luego rotarlo un cierto ángulo, a primero rotar el objeto y luego trasladarlo. Esto se ve reflejado en el orden en el que se efectúan las multiplicaciones entre matrices ya que la multiplicación entre estas no es conmutativa.

2.2. Robótica

El término “robot” apareció por primera vez en el libro de 1920 R.U.R (Robot Universales Rossum) del autor checo Karel Čapek y proviene de la palabra checa *robota*, que significa trabajar o trabajo forzado. La Real Academia de la lengua española (RAE) define “robot”

como una “máquina o ingenio electrónico programable que es capaz de manipular objetos y realizar diversas operaciones”. Por su parte, el Instituto Americano de Robótica lo definió en 1979 como “un manipulador reprogramable y multifuncional diseñado para mover materiales, partes, herramientas o equipo especializado mediante varios movimientos programados para la ejecución de diversas tareas”.

Algunos conceptos importantes en el estudio de los robots son los siguientes:

- Grados de libertad (GDL o DOF): Es el número de coordenadas independientes necesarias para expresar la posición y orientación de un mecanismo o robot.
- Eslabón: Cuerpo rígido que posee al menos dos nodos (puntos de unión). Los eslabones se unen entre sí para formar cadenas cinemáticas (Figura 2.1)

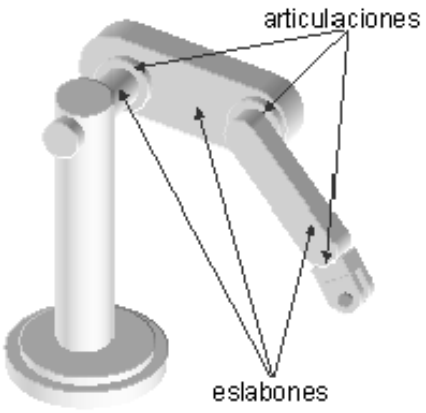

Figura 2.1. Diferencia entre eslabón y articulación

- Junta o articulación: Punto donde se conectan los eslabones. Las juntas tienen asociadas un número de grados de libertad y existen diferentes tipos (fija, rotacional, prismática, cilíndrica, esférica, planar, entre otras) (Figura 2.2).

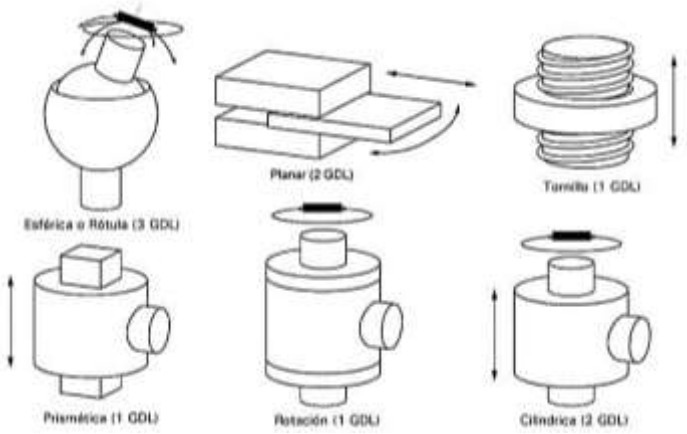

Figura 2.2. Algunos tipos de articulaciones

- Efector final: Es la herramienta especial que permite al robot realizar una tarea o aplicación en especial. Algunos ejemplos son pinzas, ventosas, herramientas de soldadura, etc.
- Espacio de trabajo: Es el espacio tridimensional definido por los puntos que el robot puede alcanzar (Figura 2.3). Es una representación de las limitaciones geométricas de un robot y por ende cualquier análisis hecho afuera de este espacio no será válido. Los parámetros más importantes de este concepto son el volumen (dado por el tamaño y la longitud de los eslabones) y la forma (dado por el tipo de robot).

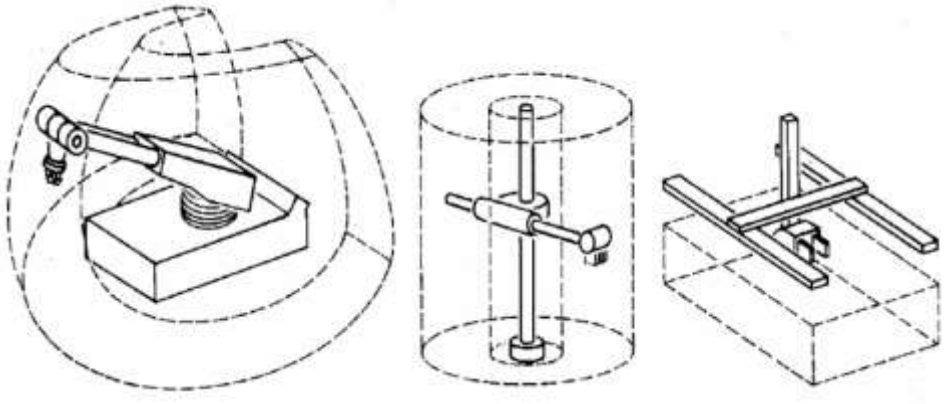

Figura 2.3. El espacio de trabajo de algunos robots

2.2.1. Clasificación de robots

Los robots se pueden clasificar en diversas categorías, pero generalmente se clasifican como:

a) Robots humanoides o androides

Son aquellos cuya forma recuerda a la forma humana, por ejemplo, brazos, cabeza y torso. Algunos modelos más avanzados incluso poseen piernas y son capaces de caminar.

Figura 2.4. Robot humanoide NAO

b) Robots móviles

Tienen la capacidad de desplazarse en su entorno y no están fijos a una cierta posición. Generalmente se desplazan mediante ruedas, pero también pueden utilizar otros medios como rieles o hélices. Estos robots pueden navegar de forma autónoma o en trayectorias predefinidas. Sus aplicaciones son muy variadas, yendo desde robots de servicio, drones, robots de rescate, robots militares y robots de exploración.

Figura 2.5. Robot móvil Turtlebot.

c) Robots suaves (*soft robots*)

Se diferencian de los demás tipos de robots debido a que usan materiales flexibles y porque los movimientos que hacen estos robots se asemejan a algunos encontrados en la naturaleza como orugas, tentáculos o ventosas. Suelen estar hechos de materiales que se deforman con cambios de corriente eléctrica, cambios de temperatura o cambios de presión. Estos robots permiten la realización de tareas más flexibles que otros robots no pueden hacer y también son más seguros para los humanos.

2.3. Robots industriales

Los robots industriales también conocidos como robots manipuladores, son aquellos utilizados en la industria para labores de automatización de procesos o para realizar tareas peligrosas para los seres humanos. La norma ISO8373 [14] define el término "robot industrial" como "un manipulador reprogramable, multipropósito, controlado automáticamente, con tres o más ejes, que puede ser puesto en un lugar fijo o móvil para uso en aplicaciones de automatización industrial".

Existen varias configuraciones de robots industriales y su uso varía dependiendo de su aplicación. Estas son cartesiano, cilíndrico, polar, articulado, SCARA y delta [15], [16]. Las primeras cinco configuraciones se pertenecen al grupo de robots seriales, debido a que sus eslabones se encuentran conectados uno tras otro. Los robots delta también son

conocidos como robots paralelos, ya que sus eslabones están interconectados en una sola base común. Las características específicas de cada configuración son dadas a continuación.

a) Cartesiano

También llamados robots lineales, son aquellos que se mueven de linealmente en tres diferentes ejes (*x*, *y* y *z*), por lo tanto, sus espacios de trabajo tienen la forma de un paralelepípedo. Se suelen utilizar en máquinas de control numérico (CNC), grabado laser y en impresoras 3D.

Figura 2.6. Robot tridimensional cartesiano YXCR de Festo:

b) SCARA (*Selective Compliance Assembly Robot Arm*)

Consisten en dos juntas giratorias paralelas que se mueven en un mismo plano, y una junta prismática al final. Este tipo de robot se usa en aplicaciones de ensamble rápido, carga de material, empacado y en la industria farmacéutica.

Figura 2.7. Robot SCARA IRB 910SC de ABB

c) Articulado

La forma de estos robots asemeja a un brazo humano. Estos robots se montan en una base mediante una junta rotacional. A su vez el brazo está unido por juntas rotacionales

cuyo número puede variar entre dos y diez, pero lo más común es que sean cuatro, cinco o seis. Su configuración les permite ubicar su efector final en diversas orientaciones, lo cual los hace muy flexibles y les confiere una mayor utilización en la industria. Sus aplicaciones varían desde ensamble, soldadura, manejo de materiales, empaque, etc.

Figura 2.8. Robot articulado KR3 AGILE de KUKA

d) Cilíndrico

Son aquellos que poseen una junta rotatoria en la base y al menos dos juntas prismáticas. Su espacio de trabajo tiene la forma de un cilindro, de ahí su nombre. Estos robots ofrecen movimiento lineal horizontalmente y verticalmente, y permite que el efector final llegue a lugares con poco espacio sin perder velocidad o precisión. Sus aplicaciones suelen ser ensamble, pintado de superficies, transporte de paneles, y en general en aplicaciones con espacio limitado.

Figura 2.9. Robot cilíndrico PlanetCrane SciClops de Hudson Robotics

e) Delta o paralelo

Son aquellos cuyos eslabones están conectados en paralelo a un base común. Estos robots son bastante rápidos y precisos debido a que la posición y orientación del efector final dependen directamente de cada eslabón. Se usan en industrias que requieren aplicaciones de “pick and place” a gran velocidad como la industria alimenticia, farmacéutica y electrónica.

Figura 2.10. Robot paralelo Veloce de Penta Robotics

2.3.1. Cinemática de robots industriales

La cinemática es la rama de la mecánica que estudia el movimiento sin considerar las fuerzas que lo provocan. En robótica, la cinemática estudia las posiciones, orientaciones, velocidades y aceleraciones de las articulaciones de los robots.

Los valores de las juntas forman el llamado espacio articulado o espacio de juntas, cuyo número de elementos es igual a los grados de libertad del robot. Los valores de posición y orientación del efector final conforman el espacio de trabajo y siempre tiene seis elementos (los valores de posición *x*, *y* y z, y la orientación θ, φ y ψ). De esta manera, si *q* es el espacio de juntas, *n* son los grados de libertad y *p* es el espacio de trabajo, se tiene lo siguiente:

$$q = \begin{bmatrix} q_1 \\ q_2 \\ \vdots \\ q_n \end{bmatrix} \qquad p = \begin{bmatrix} x \\ y \\ z \\ \theta \\ \varphi \\ \psi \end{bmatrix}$$

Existen dos tipos de cinemática, la directa y la inversa. La cinemática directa busca determinar la posición y orientación del efector final en función de las posiciones de las juntas del robot, mientras que con la cinemática inversa se determinan los valores de las juntas en función de la posición del efector final. Para controlar robots manipuladores se

utiliza la cinemática inversa, sin embargo, para obtenerla es necesario determinar primero la cinemática directa.

Para obtener la cinemática directa primero se define un sistema de referencia en cada junta del robot y después se describen las transformaciones geométricas entre cada una de estas mediante matrices de traslación y rotación homogéneas. Al final se obtiene una matriz que relaciona geométricamente la base del robot con el efector final. Las relaciones que contienen la matriz están en función de los valores de las juntas. Entonces, si $M_{CD} = f(q)$ es la matriz de la cinemática directa, *p* es el espacio de trabajo y $q = [q_1\ q_2 \dots q_n]$ es el espacio de juntas se tiene que:

$$p = M_{CD}(q)$$

El método de Dénavit-Harteneberg es comúnmente utilizado para resolver de forma rápida la cinemática directa. Este se verá a detalle en la sección 2.4.1.1.

En la cinemática inversa es necesario obtener los valores de las juntas en función del espacio de trabajo. Dependiendo del número de grados de libertad del robot, es posible encontrar múltiples soluciones o ninguna para un conjunto de valores del espacio del trabajo. La figura 2.11 es un ejemplo de múltiples soluciones para la misma posición y orientación de efector final.

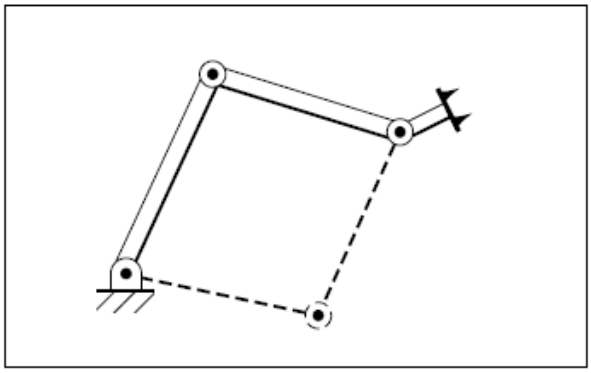

Figura 2.11. Múltiples soluciones para un robot articulado de 3GDL

Si bien es posible obtener las ecuaciones analíticas de la cinemática inversa a partir de las expresiones de la cinemática directa, lo más común es resolver las ecuaciones de la cinemática inversa mediante métodos numéricos a través de algoritmos y softwares especializados.

2.3.1.1. Parámetros de *Dénavit-Hartenberg*

El objetivo de este método es la obtención de cuatro parámetros que definen las relaciones de un sistema de coordenadas i con el sistema anterior $i-1$. Estos parámetros son [17]:

- a_i: En el caso de articulaciones giratorias se refiere a la distancia a lo largo del eje x_i, que va desde la intersección del eje z_{i-1} con el origen del sistema i-ésimo,. En el caso de articulaciones prismáticas, se calcula como la distancia más corta entre z_i y z_{i-1}.
- $\propto_i$: Es ángulo de separación entre z_i y el eje z_{i-1}, medido en un plano perpendicular al eje x_i, utilizando la regla de la mano derecha.
- d_i: Es la distancia a lo largo del eje z_{i-1} desde el origen del sistema de coordenadas (i-1)-ésimo hasta la intersección del eje z_{i-1} con el eje x_i. Se trata de un parámetro variable en las articulaciones prismáticas.
- θ_i: Es el ángulo que forman los ejes x_i y x_{i-1} medido en un plano perpendicular al eje z_{i-1}utilizando la regla de la mano derecha. Se trata de un parámetro variable en las articulaciones giratorias.

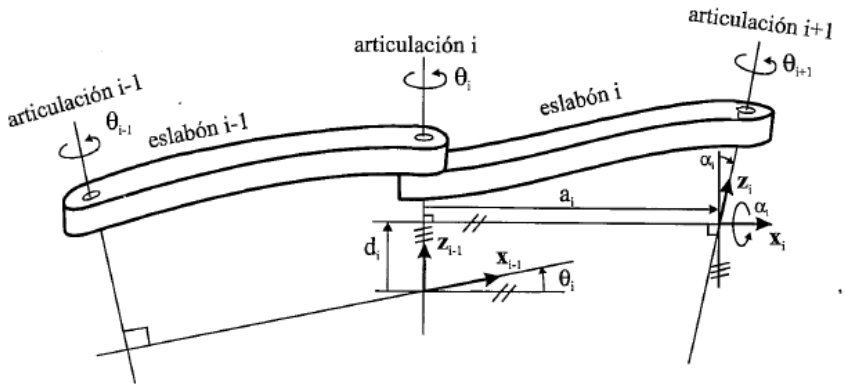

Figura 2.12. Parámetros de Dénavit-Hartenberg para un eslabón rotatorio [17]

Para la obtención de los parámetros mencionados anteriormente, es necesario seguir la siguiente serie de pasos para la asignación de los sistemas de referencia en cada articulación:

1. Numerar los eslabones e iniciar con 1 (este será el primer eslabón de la cadena) y acabando en n (último eslabón móvil). La base fija del robot será numerada como 0.
2. Numerar cada articulación de igual forma que los eslabones, iniciar con 1 y terminar en n.
3. Localizar el eje de cada articulación. Si esta es giratoria, el eje será su eje de giro. Si es prismática, será el eje a lo largo del cual se produce desplazamiento.

4. Para i de 0 a n-1, situar el eje z_1 sobre el eje de la articulación $i-1$.
5. Ubicar el origen del sistema de la base S_0 en cualquier parte del eje z_0 y asignar los siguientes ejes de tal manera que estos cumplan con la regla de la mano derecha.
6. Situar el sistema S_i en la intersección del eje z_i con la línea normal común de z_{i-1} y z_i. Si ambos ejes se cortasen se situaría S_i en el punto de corte. Si fuesen paralelos se situaría S_i en la articulación $i+1$.
7. Ubicar x_i en la línea normal común a z_{i-1} y z_i.
8. Asignar y_i de modo que cumpla con la regla de la mano derecha con x_i y z_i.
9. Colocar le sistema S_n en el extremo del robot de modo que z_n coincida con la dirección de z_{n-1} y x_n sea normal a z_{n-1} y z_n.
10. Obtener θ_i como el ángulo que tenemos que girar z_{i-1} para que queden paralelos x_i y x_{i-1}.
11. Obtener d_i como la distancia que necesitamos desplazar sobre z_{i-1} de manera que queden alineados x_i y x_{i-1}.
12. Obtener a_i como la distancia que tenemos que desplazar sobre x_i de modo que coincidan x_i y x_{i-1}.
13. Obtener $\propto_i$ como el ángulo que tenemos que girar x_{i-1} para que coincida el sistema S_i con S_{i-1}.

Posteriormente, la cinemática directa se obtiene mediante la siguiente expresión:

$$M = {}^0A_1 * {}^1A_2 * \dots * {}^{i-1}A_i$$

Donde M expresa la posición y rotación del extremo del robot en función de sus coordenadas articulares y ${}^{i-1}A_i$ es la matriz de transformación homogénea que representa el cambio de posición y traslación (primero en el eje x_i y luego en el eje z_i) entre el sistema actual de coordenadas y el anterior.

$${}^{i-1}A_i = R_z(\theta_i) * T(0,0,d_i) * T(a_i,0,0) * R_x(\alpha_i)$$

$${}^{i-1}A_i = \begin{bmatrix} cos\theta_i & -cos\alpha_i * sen\theta_i & sen\alpha_i * sen\theta_i & a_i * cos\theta_i \\ sen\theta_i & cos\alpha_i * cos\theta_i & -sen\alpha_i * cos\theta_i & a_i * sen\theta_i \\ 0 & sen\alpha_i & cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2.4. Visión por computadora

La visión por computadora o visión artificial es una disciplina científica que utiliza técnicas, métodos y algoritmos que tienen como objetivo reconstruir y reinterpretar escenas del mundo real basándose en el contenido de imágenes capturadas mediante cámaras

digitales [18]. Estas imágenes consisten básicamente en pixeles, bordes, ángulos, color, geometría de la imagen, forma y textura [19].

En la actualidad, la visión artificial es muy utilizada en diversas áreas, por ejemplo: robótica, navegación automática, procesamiento de señales, neurobiología, automóviles autónomos, búsqueda de imágenes, etc. Algunas áreas de estudio de la visión por computadora son: detección de rostros, reconocimiento de objetos, reconocimiento de formas, detección de objetos, reconocimiento de patrones, realidad aumentada, realidad virtual, segmentación de imágenes y reconstrucción de escenas.

2.4.1. Reconocimiento de objetos

El reconocimiento de objetos es una aplicación del procesamiento de imágenes y la visión por computadora. Este término es utilizado en una gran variedad de aplicaciones y algoritmos. La idea básica del reconocimiento de objetos es que, dado un cierto conocimiento sobre la apariencia de ciertos objetos, una o varias imágenes son examinadas para evaluar qué objetos están presentes y en qué lugar, sin embargo, cada aplicación tiene ciertas particularidades y restricciones, es por ello que existe una gran cantidad de algoritmos [20]. Algunas aplicaciones del reconocimiento de objetos son la medición de posición, inspección, clasificación, contado y detección de objetos, así como categorización de escenas.

2.4.2. Imágenes digitales

Una imagen digital es una representación discreta de elementos visuales que tienen información espacial e intensidad [19]. Esta imagen puede ser representada en tres formas: en escala de grises, en blanco y negro (también denominada binaria) y a color. Una imagen binaria, consiste enteramente en pixeles blancos (Intensidad = 1) y negros (Intensidad = 0). Una imagen en escala de grises se conforma con pixeles variantes en intensidad de negro y blanco, es representada por una función de intensidad $I(x,y)$, donde x, y son las coordenadas espaciales e $I(x,y)$ es proporcional al valor de intensidad de luz que impacta sobre un sensor óptico y grabado en el pixel o punto correspondiente.

Figura 2.13. Imagen en diferentes representaciones. Izquierda: Binaria, Centro: Escala de grises, Derecha: Color (RGB). [Autor: GlobalIP | Crédito: Getty Images/iStockphoto]

Las imágenes binarias y en escala de grises se representan como arreglos bidimensionales (dos valores de intensidad), en contraste, una imagen a color RGB (RED, GREEN, BLUE) es representada como un vector tridimensional, dado que los pixeles se componen por tres canales de color. Las imágenes RGB viven en el espacio de color RGB, sin embargo, existen diversos tipos de espacios de color como el HSV (*Hue, Saturation, Value*) u otros con más dimensiones como el RGBY o el RGBA.

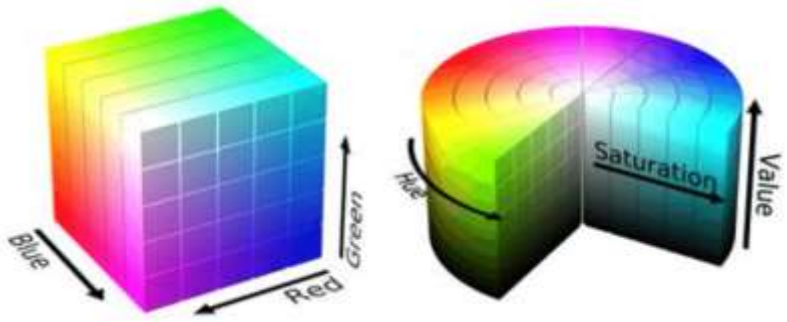

Figura 2.14. Derecha: Espacio de color RGB, Izquierda: Espacio de color HSV

Un pixel o punto de la imagen es un punto físico que contiene la información que representa la respuesta de un sensor óptico a una partícula de luz (fotón) reflejada como parte de un objeto en un campo de visión. Los pixeles generalmente se modelan como un pequeño cuadrado.

2.4.3. Detección y descripción de características

En visión por computadora, una característica o *feature* es una región de la imagen que posee un patrón de pixeles diferente al de su vecindad inmediata. Usualmente está asociado a cambios en las propiedades de la imagen. Estas propiedades pueden ser intensidad de luz, color y textura. Las características pueden ser puntos, bordes o pequeñas regiones de la imagen. [21]

Un algoritmo de detección de características es aquel que toma una imagen para obtener las regiones de interés de dicha imagen. Algunos ejemplos de esto son los detectores de esquinas o de bordes. El problema de estos algoritmos es que no obtienen mayor información de las características detectadas.

Un descriptor es un algoritmo que obtiene vectores de características (también llamados descriptores de características) de una imagen. Estos vectores contienen información sobre las características de ciertas regiones de la imagen, que permiten diferenciar estas regiones de otras. Esta información, idealmente, se mantiene invariante incluso si la imagen es trasformada de alguna forma. Algunos ejemplos de esta clase de algoritmos son SURF, SIFT u ORB.

Idealmente, un algoritmo de detección de características debería tener las siguientes propiedades [21], [22]:

- Cantidad: Debe existir una cantidad suficiente de características para que incluso objetos pequeños puedan ser detectados.
- Repetibilidad: Si se toman dos imágenes de una escena en diferentes condiciones, una gran cantidad de las características encontradas en la región compartida por ambas imágenes deben ser encontradas.
- Localidad: Las características deben ser locales para reducir la probabilidad de oclusión y para simplificar el modelo aproximado de las deformaciones geométricas y fotométricas entre dos imágenes tomadas bajo diferentes condiciones.
- Distinción: Se refiere a la variación en la intensidad de los patrones de las características detectadas.
- Eficiencia: La detección de características de una imagen debe ser lo suficientemente rápida para las aplicaciones en tiempo real.
- Precisión: Las características deben ser respetadas, es decir, se deben localizar de forma precisa incluso cuando hay cambios en la escala y en la forma de la imagen.

2.4.4. Espacio - escala

El espacio – escala, también conocido como espacio de escala o *space-scale* es una propiedad inherente a los objetos en el mundo real es que solo existen algunas entidades significativas sobre un determinado rango de escala. Un ejemplo es la rama de un árbol, este concepto solo tiene sentido en un rango de escala de unos cuantos centímetros a unos pocos metros. No es significativo discutir el concepto de rama de árbol a escala molecular o a varios kilómetros de distancia.

Al observar objetos con los ojos o con una cámara existe un problema adicional de escala debido a la perspectiva. Un objeto cercano parecerá más grande en comparación con un objeto lejano, a pesar de que ambos tengan el mismo tamaño en el mundo real. Estos hechos, indican que es necesario considerar el concepto de escala a la hora de estudiar percepción natural y artificial.

Por esta razón, un sistema de visión debe ser capaz de manejar estructuras de imágenes en todas las escalas posibles. La idea principal de crear una representación multiescala de una señal, es generar una familia de señales derivadas con un parámetro en común. Las señales derivadas deben suprimir cualquier información detallada de forma sucesiva, para remover detalles innecesarios y para suprimir el ruido. Esta familia de señales es la que compone el espacio-escala [23].

El espacio-escala de una imagen bidimensional se representa generalmente con la convolución gaussiana. Para una imagen bidimensional $f(x,y)$, su representación en espacio-escala es una familia de señales derivadas $L(x,y,t)$ definida por la convolución entre $f(x,y)$ y la función gaussiana $g(x,y,t)$, donde:

$$g(x,y,t) = \frac{1}{2\pi t} e^{-\frac{x^2+y^2}{2t}}$$

De tal forma que:

$$L(x,y,t) = g(x,y,t) * f(x,y) = \int_{(\xi,\eta)\in\mathbb{R}^2} f(x-\xi, y-\eta) g(\xi,\eta,t) d\xi d\eta$$

Y el parámetro de escala $t = \sigma^2$ es la varianza del filtro gaussiano. Cuando $t = 0$ se trata de la imagen original y conforme el valor de t aumenta, se pronuncia el suavizado de la imagen (figura 2.15).

A su vez, el espacio-escala también puede definirse como la solución de la ecuación de difusión con la condición inicial $L(x,y,t) = f(x,y)$:

$$\frac{\partial L}{\partial t} = \frac{1}{2} \nabla^2 L$$

Si bien esta forma de representar el espacio-escala es común, la forma de definir y representarlo varía dependiendo de los algoritmos y sus objetivos (velocidad de procesamiento, precisión, etc).

Figura 2.15. Representación del espacio-escala de una imagen con diversos valores de t. De izquierda a derecha y de arriba a abajo, t = 0,1,4,16,64,256

2.4.5. Algoritmo RANSAC

El algoritmo RANSAC (*RANdom SAmple Consensus*) fue desarrollado por Fischer y Bolles en 1981 [24] y se trata de un método iterativo para estimar los parámetros del modelo matemático de un conjunto de datos con varios valores atípicos (*outliers*). Este algoritmo se compone básicamente de dos grandes pasos que se repiten hasta que se halla un modelo satisfactorio: 1) Se selecciona aleatoriamente una pequeña muestra del conjunto original de datos a partir de los cuales y con un modelo de ajuste se calculan sus correspondientes parámetros; 2) Después, el algoritmo determina qué elementos del conjunto original de datos son consistentes con el modelo instanciado por los parámetros estimados en el primer paso (consenso). Un elemento será considerado un valor atípico si no se ajusta al modelo obtenido, dentro de un umbral de error.

La principal ventaja de este algoritmo es su habilidad para hacer estimaciones robustas de los parámetros del modelo, es decir, que puede estimar dichos parámetros con un gran grado de precisión incluso si existe una gran cantidad de valores atípicos en el conjunto de datos, sin embargo, esto conlleva una gran carga computacional. El pseudocódigo de este algoritmo es el siguiente:

```
Parámetros de entrada:
    datos – El conjunto de datos
    modelo – El modelo para explicar los datos observados
    n – Número mínimo de puntos requeridos para estimar los parámetros del modelo
    k – Número máximo de iteraciones
    t – Umbral para determinar los puntos que se ajustan al video
Salida:
    mejorModelo – El modelo determinado del conjunto de datos

Inicialización:
    iteraciones = 0
    modeloAjustado = NULL
    errorModelo = 10000      # Algún número grande

while iteraciones < k do:
    # Selección aleatoria de datos
    modeloPropuesto = random(datos)
    inliers = 0
    n = 800    # Ajustable
    for puntos in datos and not in inliers do:
        # Función para calcular el error. Esta puede ser la distancia al plano, en caso de
        #detección de planos
        calcularError();
        if punto ajusta a modeloPropuesto with error < t:
                inliers = punto
                errorModelo += error
    #  Si se tiene un buen modelo
    if numeroElementos in inliers > n:
        # Se verifica si el modelo calculado es mejor que el anterior calculado
        if errorModelo < menorError:
                mejorAjuste = mojorModelo
                menorError = errorModelo
    iteraciones++
return mejorAjuste
```

2.4.6. Análisis de Componentes Principales (PCA)

El análisis de componentes principales (PCA, por sus siglas en inglés) es un método estadístico usado para reducir las dimensiones grandes conjuntos de datos de tal manera que se conserve la mayor cantidad de información posible [25]. Existen dos métodos para calcular los componentes principales: uno basado en el cálculo de la matriz de covarianza, mientras que el segundo se basa en la descomposición en valores singulares (SVD, por sus siglas en inglés).

En este caso solo se cubrirá el primer método, el cual se divide en dos grandes pasos, el primero consiste en el cálculo de la matriz de covarianza y el segundo en el cálculo de los valores y vectores propios de dicha matriz. Cabe resaltar que antes de hacer los cálculos se normalizan o centran los datos, esto es restar el valor de la media de cada variable a los datos. Posteriormente se hace el cálculo de la matriz de covarianza a partir de los datos centrados.

2.4.6.1. Varianza y matriz de covarianza

La varianza (σ^2) de cualquier variable mide la desviación de dicha variable respecto al valor de su media (μ) y se define como (donde *E(x)* es el valor esperado):

$$\sigma^2(x) = var(x) = E[(x-\mu)^2] = E[x^2] - (E[x])^2$$

La covarianza se utiliza cuando hay más de una variable y se define de la siguiente forma:

$$cov(x,y) = E\big[(x-\mu_x)\big(y-\mu_y\big)\big] = E[xy] - E[x]E[y]$$

La matriz de covarianzas (C) no es más que el cálculo de la covarianza entre todas las variables del conjunto de datos y su posterior ordenamiento en forma de matriz. Esto es:

$$C = \begin{bmatrix} var(x_1) & cov(x_1,x_2) & \dots & cov(x_1,x_n) \\ cov(x_2,x_1) & var(x_2) & \dots & cov(x_2,x_n) \\ \vdots & \vdots & \ddots & \vdots \\ cov(x_n,x_1) & cov(x_n,x_2) & \dots & var(x_n) \end{bmatrix}$$

Algunas propiedades de esta matriz es que se trata de una matriz simétrica por lo que cumple con: $C = C^T$. A su vez también es una matriz positiva definida, lo que tiene consecuencia que todos sus valores propios son mayores a cero.

2.4.6.2. Valores y vectores propios

Se conocen como valores propios de una matriz *A* son aquellos que cumplen con la siguiente ecuación (donde I es la matriz identidad):

$$\det(A - \lambda I) = 0$$

Esto se traduce a la resolución de un polinomio de grado n, donde n es la dimensión de A:

$$\det(A - \lambda I) = (\lambda_1 - \lambda)(\lambda_2 - \lambda) \dots (\lambda_n - \lambda) = 0$$

Donde $\lambda_i \mid i = \{1,2, \dots, n\}$ son los valores propios de dicha matriz. Puede observarse que λ es un vector que contiene dichos valores.

De esta manera, existe un vector propio asociado ($\bar{v}_i$) a cada uno de los valores propios de la matriz. Para hallar cada vector propio, se resuelven las siguientes ecuaciones:

$$\begin{gathered}(A - \lambda_1 I)\bar{v}_1 = 0 \\ (A - \lambda_2 I)\bar{v}_2 = 0 \\ \vdots \\ (A - \lambda_i I)\bar{v}_i = 0\end{gathered}$$

Los vectores propios de esta matriz forman un espacio vectorial, y son linealmente independientes.

Para el PCA, estos vectores propios representan los componentes principales del conjunto de datos. A su vez, dichos vectores indican las direcciones del espacio de los componentes principales. Por su parte, los valores propios representan la escala y magnitud de los vectores propios. El vector propio con el valor propio más grande representa el primer componente principal y así sucesivamente. Esto se puede utilizar como forma de reducir las dimensiones del conjunto de datos al solo seleccionar los primeros componentes.

En el caso de la matriz de covarianza, dado que esta es simétrica y real (no contiene valores complejos), la base formada por sus vectores propios es ortogonal. Esto es especialmente importante para encontrar la dirección de un objeto en el espacio.

2.5. Modelos y simulaciones

Un modelo es una entidad utilizada para representar otra entidad para un motivo en específico. En general, son abstracciones simplificadas con nivel de detalle suficiente para que satisfagan los objetivos de estudio y se emplean cuando la investigación del sistema real es impráctica, peligrosa o imposible.

Por otro lado, una simulación es en principio una prueba de campo, pero con la diferencia de que el sistema es reemplazado con un modelo físico o computacional. Una simulación involucra crear un modelo que imite los comportamientos del sistema; experimentar con dicho modelo para generar observaciones; e intentar entender, resumir y/o generalizar estos comportamientos [26].

Para realizar una simulación, se requiere de un simulador, el cual es un modelo físico, o computacional que permite realizar la simulación de un sistema. Por ejemplo, existen algoritmos que simulan el comportamiento del clima, el comportamiento de mercados financieros, la dinámica de sistemas físicos, etc. En la actualidad se utilizan las simulaciones con propósitos predictivos, para evaluar u optimizar el desempeño de algún sistema, hacer pruebas, evaluar la seguridad de algún proceso o sistema, en educación y en videojuegos.

Capítulo 3 Robot manipulador KUKA KR5-2 arc hw

Figura 3.1. KUKA KR5-2 arc hw

3.1. Características del robot

El KUKA KR 5-2 arc hw es un robot industrial articulado de seis grados de libertad diseñado principalmente para tareas de soldadura y manipulación de objetos. Presenta una muñeca hueca (*hollow wrist*) que permite hacer el cambio de herramientas más rápido. La figura 3.2 muestra los componentes principales del robot.

Las características específicas se muestran en la tabla 3.1, mientras que en la figura 3.3 se pueden observar los ejes de rotación del robot [27]. Más características se pueden ser encontradas en el manual de este robot [28].

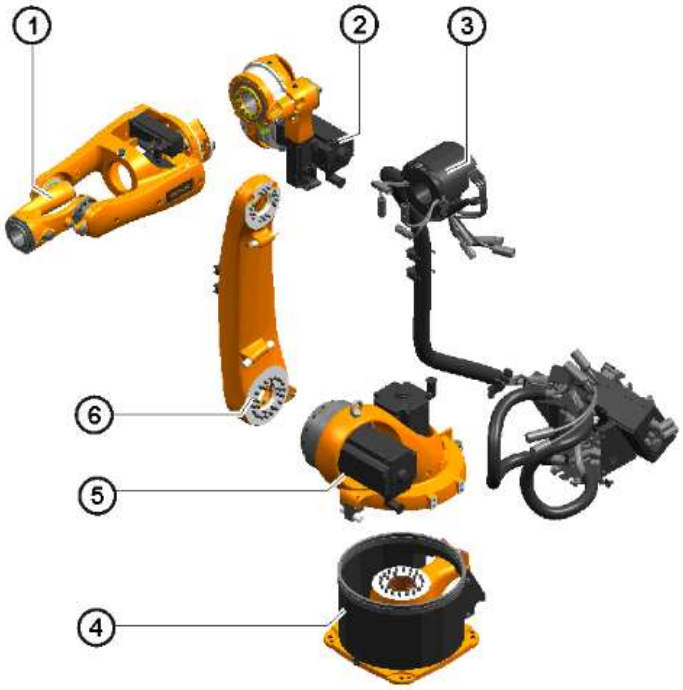

Figura 3.2. Elementos principales del robot. 1. Muñeca de eje hueco (hollow-shaft wrist), 2. Brazo, 3. Instalaciones eléctricas, 4. Base, 5. Columna rotatoria y 6. Brazo enlace.

Tabla 3.1: DATOS DEL ROBOT "KUKA KR5-2 arc hw"

Característica	**Valor**
Alcance máximo	1423 mm
Carga nominal	5 kg
Carga total máxima	37 kg
Número de ejes	6
Posición de montaje	Piso, techo
Repetitividad de la posición	±0.04 mm
Tipo de controlador	KR C4
Peso aproximado (sin contar el controlador)	126 kg
Temperatura durante la operación	+10 °C a +55°C
Clasificación de protección	IP54
Área de la base	324mm x 324 mm
Conexión	7.3 kVA
Nivel de ruido	< 75 dB

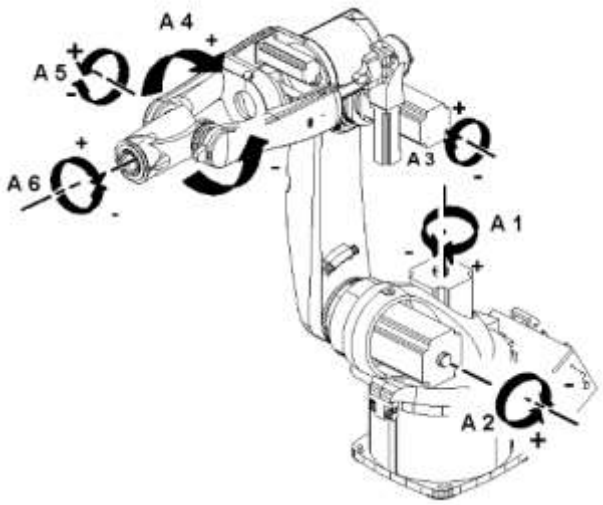

Figura 3.3. Ejes de rotación del robot

Los datos de las posiciones y velocidades máximas de los ejes se encuentran en la tabla 3.2.

TABLA 3.2. CARACTERÍSTICAS DE LOS EJES DEL KUKA KR5-2 ARC HW [27]

Eje	Rango de posición	Velocidad (Con carga de 5kg)
A1	±155°	156°/s
A2	-180° a +65°	156°/s
A3	-110° a +170°	227°/s
A4	±165°	390°/s
A5	±140°	390°/s
A6	Sin límite	858°/s

El alcance y el espacio de trabajo de este robot se pueden observar en la figura 3.4

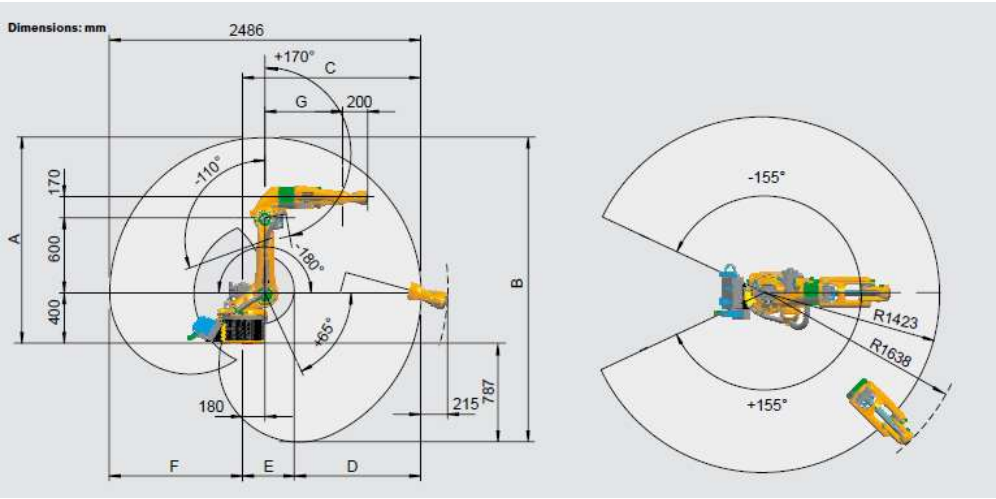

Figura 3.4. Alcance y espacio de trabajo del KUKA KR5-2 arc hw

3.1.1. Efector final

El efector final es una pinza neumática PZN-plus 100-2 de SCHUNK (figura 3.5), cuyas características se encuentran en la tabla 3.3 [29].

Figura 3.5. Pinza (gripper) PZN-plus 100-2 de SHUNK

TABLA 3.3: ALGUNAS CARACTERÍSTICAS DE LA PINZA PZN-PLUS 100-2 [29]

Característica	**Valor**
Carrera por mordaza [mm]	5
Fuerza de cierre [N]	4000
Peso [kg]	1.41
Peso recomendado de la pieza [kg]	20
Presión de trabajo mínima [bar]	2
Presión de trabajo máxima [bar]	8
Tiempo de cierre [s]	1
Longitud máx. admisible de los dedos [mm]	135
Protección IP	40
Rango de temperatura [°C]	5-90
Repetibilidad [mm]	0.01
Diámetro D [mm]	120
Altura Z [mm]	59.3
Momento Mx máx. [Nm]	80
Momento My máx. [Nm]	115
Momento Mz máx. [Nm]	70
Fuerza axial Fz máx. [N]	2000

Dado que la pinza no incluye los dedos, se utilizaron unos hechos a base de aluminio. Las medidas de los dedos pueden ser observadas en la figura 3.6.

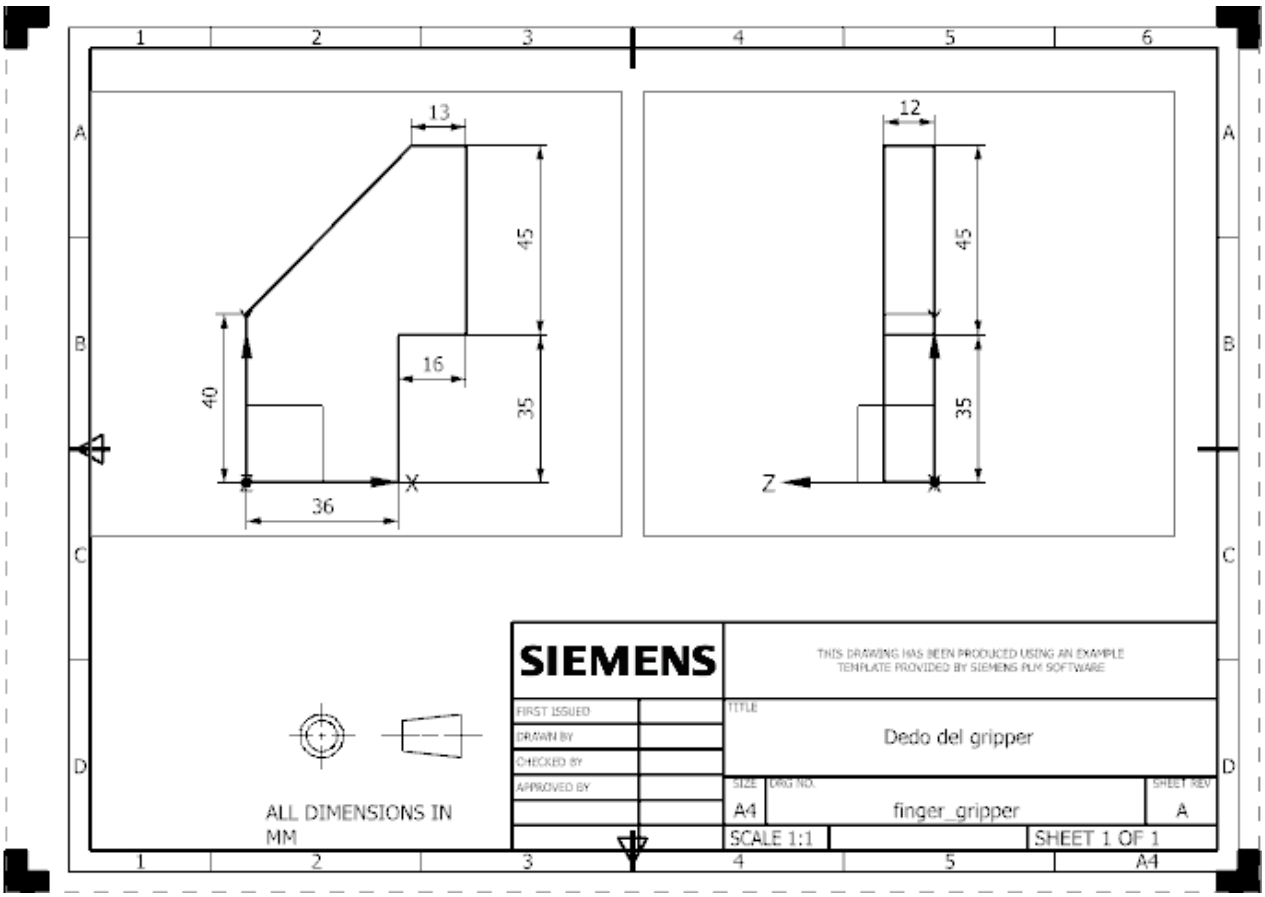

Figura 3.6. Plano de los dedos del gripper (Específico para este caso).

3.2. Cinemática directa

3.2.1. Parámetros de Dénavit-Hartenberg

Las medidas del robot se obtuvieron del archivo CAD que la empresa KUKA Robotics pone a disposición general en su página de internet. Se consideró la base del robot como el sistema global de referencia con eje X apuntando hacia enfrente del robot y el Z hacia arriba. Después, se asignaron sistemas de referencia en cada articulación, tal como lo indica el método de Dénavit-Hartenberg (figura 3.7).

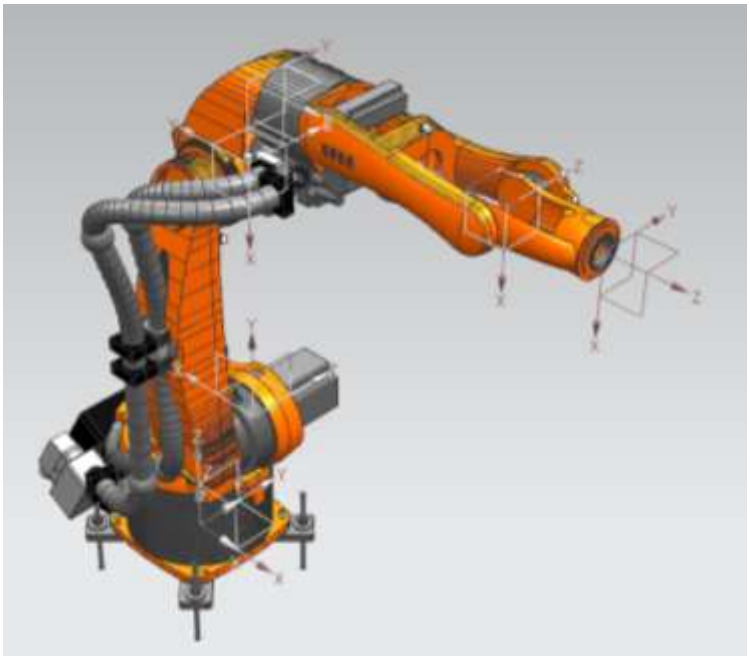

Figura 3.7. Sistemas de referencia asignados a los ejes del robot, siguiendo el método de Dénavit-Hartenberg

Posteriormente, se obtuvo la siguiente tabla con los parámetros de Dénavit-Hartenberg:

TABLA 3.4. PARÁMETROS DE DÉNAVIT-HARTENBERG DEL ROBOT

i	θ_i [rad]	d_i [m]	$\propto_i$ [rad]	a_i [m]
1	$\theta_1 + \pi$	0.4	π	-0.18
2	$\theta_2 - \frac{\pi}{2}$	0	0	-0.6
3	θ_3	0	π	-0.17
4	θ_4	0.62	π	0
5	θ_5	0	$-\pi$	0
6	θ_6	0.2	0	0

3.2.2. Descripción con el formato URDF (ROS)

El formato URDF (*Universal Robot Description Format*) es utilizado por el *framework* para robots ROS (explicación más detallada en la sección 4.2) y consiste en describir las transformaciones del robot en un archivo con formato XML [30]. En este archivo se deben definir las características de cada eslabón (nombre, posición y orientación del origen, elemento visual, colisión, masa e inercias), así como las características de las articulaciones (nombre, tipo de articulación, posición y orientación del origen, eslabón padre y eslabón hijo).

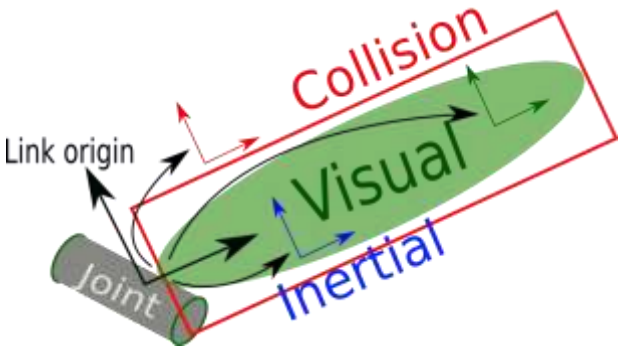

Figura 3.8. La constitución de un eslabón (link) en el formato URDF

El archivo resultante es leído por una librería de ROS llamada *tf*, la cual genera de forma automática las matrices de transformación entre los sistemas de referencia, y después publica estas matrices en forma de tópicos para que otras aplicaciones de ROS tengan acceso a esta esta información, ya sea para propósitos de visualización del robot o para resolver la cinemática inversa.

Posteriormente, otra librería de ROS llamada *joint_state_publisher* recolecta los datos provistos por *tf* y el parámetro *robot_description* presente en el servidor de parámetros, para modificar los valores de las juntas y publicarlos en el tópico */joint_states*. Este tópico es utilizado, por ejemplo, por Gazebo para hacer la simulación del robot.

La figura 3.9 es un extracto del formato URDF del robot, donde se puede apreciar la sintaxis para definir un eslabón (*link*) y una articulación (*joint*).

```
<?xml version="1.0" encoding="utf-8" ?>
<robot name="assembly_kuka_simp">

    <link name="base_link">
        <inertial>
            <origin xyz="-0.001558 0.0 0.1124" rpy="0 0 0" />
            <mass value="48.9475" />
            <inertia ixx="0.53585" ixy="0.0" ixz="-0.00395" iyy="0.54899" iyz="-0.00001" izz="0.66970" />
        </inertial>
        <visual>
            <origin xyz="0 0 0" rpy="0 0 0" />
            <geometry>
                <mesh filename="package://assembly_kuka_simp/meshes/base_link.dae" />
            </geometry>
            <material name="black" />
        </visual>
        <collision>
            <origin xyz="0 0 0" rpy="0 0 0" />
            <geometry>
                <mesh filename="package://assembly_kuka_simp/meshes/base_link.dae" />
            </geometry>
        </collision>
    </link>

    <link name="link_1">

    <joint name="base_to_link1" type="continuous">
        <origin xyz="0 0 0.2255" rpy="0 0 1.5686" />
        <parent link="base_link" />
        <child link="link_1" />
        <axis xyz="0 0 1" />
        <dynamics damping="0.7"/>
        <limit lower="-2.7053" upper="2.7053" effort="10.0" velocity="2.6878" />
    </joint>

    ...

</robot>
```

Figura 3.9. Ejemplo del formato URDF

3.3. Cinemática inversa

La cinemática inversa se calculó mediante un paquete de ROS llamado *Moveit* [31], el cual permite planear los movimientos del robot, resolver la cinemática inversa, detección y evasión de obstáculos y manejar escenas. Se dará una mayor explicación del funcionamiento de esta librería en la sección 5.3.1.

Moveit posee diversos plugin con diferentes algoritmos para la resolución de la cinemática inversa. La librería default es KDL (*Kinematics and Dynamics Library*) de Orocos [32] con el resolvedor numérico jacobiano [33]. Esta librería permite transformar vectores y sistemas de referencia en 3D, y resolver la cinemática y dinámica de cadenas cinemáticas.

Capítulo 4 Reconocimiento de objetos

Para el reconocimiento de objetos se utilizó un Kinect 2 como sensor de profundidad y cámara. Para leer los datos de este dispositivo se hizo uso de algunas librerías de ROS: iai_kinect2 para leer y convertir los datos provenientes del Kinect y find_object_2d para reconocer objetos con varios algoritmos como FAST, ORB, BRISK, etc.

4.1. Kinect 2

El Kinect 2 es un dispositivo desarrollado por Microsoft cuyo objetivo es sustituir a los controles convencionales de la XBOX. Este aparato incorpora un sensor de profundidad, sensores infrarrojos, una cámara RGB y un arreglo de micrófonos (figura 4.1) con el fin de hacer captura de movimiento de cuerpo completo en tiempo real, así como reconocimiento de voz. Las características (tabla 4.1.) de este dispositivo le hacen bastante útil en aplicaciones relacionadas a la robótica, medicina, entornos virtuales y realidad aumentada.

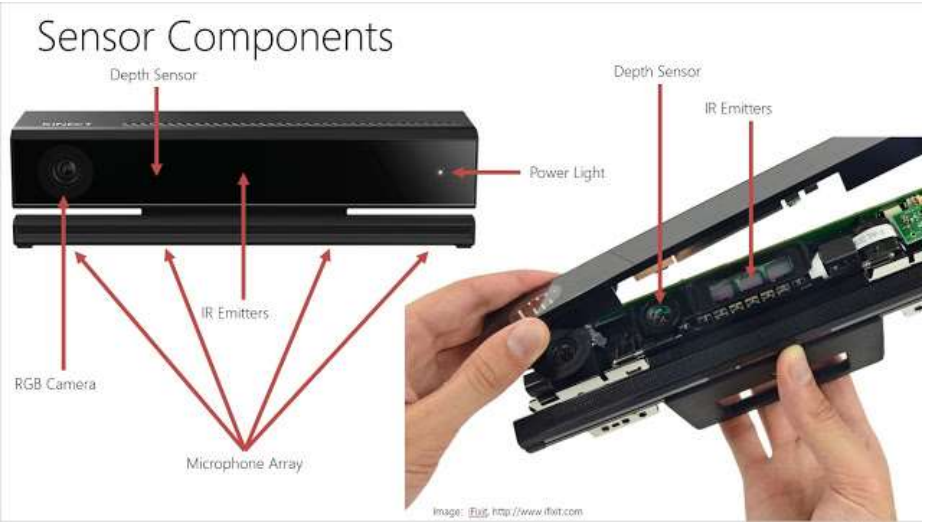

Figura 4.1. Detalles de los componentes del Kinect 2 de Microsoft.

La cámara RGB se encarga de obtener imágenes a color. Por su parte, el sensor de profundidad se encarga de medir la distancia entre el Kinect y los objetos en su rango. La forma en la que el Kinect obtiene esta distancia se basa en el principio ToF (*Time of Flight* o tiempo de vuelo), el cual es el siguiente: conociendo la velocidad de la luz, la distancia a medir es proporcional al tiempo necesario por la fuente de iluminación activa para viajar del emisor hacia el objetivo. [34].

Las cámaras que funcionan con este principio, emiten señales infrarrojas a una cierta frecuencia hacia el objetivo. Estas se reflejan en los objetos y son detectadas de nuevo. El rebote produce un desfase en la señal emitida, el cual se traduce como una diferencia de tiempo. Con esta información, la distancia es calculada de la siguiente forma [35]:

$$d_{obj} = \frac{c\Delta\varphi}{4\pi f_{mod}}$$

Donde d_{obj} es la distancia de la cámara al objetivo, c es el valor de la velocidad de la luz cuyo valor es 299,792,458 $\frac{m}{s}$; $\Delta\varphi$ es el desfase entre la señal emitida y la señal reflejada, y f_{mod} es la frecuencia de la señal emitida por la cámara. La figura 4.2 muestra el principio de funcionamiento de este tipo de cámaras.

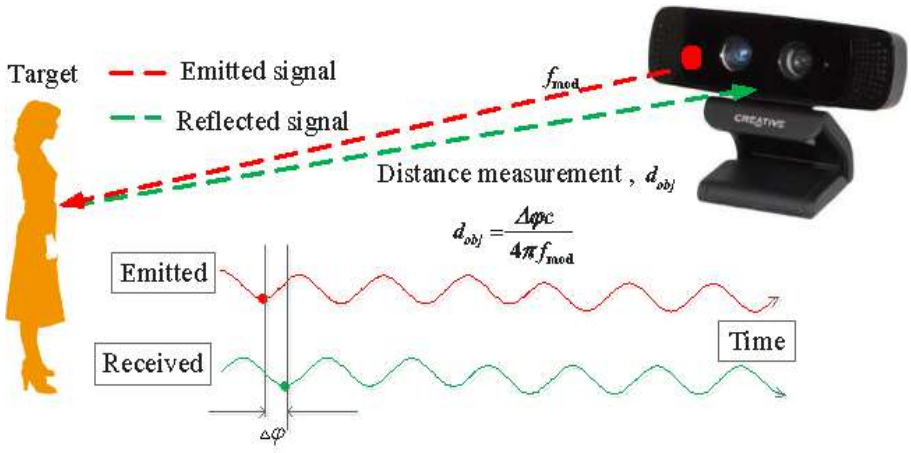

Figura 4.2. Principio de funcionamiento de una cámara ToF [35]

TABLA 4.1. ESPECIFICACIONES DEL KINECT 2 [34]

Características	**Valor**
Resolución de las cámaras infrarrojas	512 x 424, 11 bit rango dinámico
Resolución de sensor de profundidad	512 x 424 x 16 bpp, 13 bit
Resolución de la cámara a color	1920 x 1080 x 16 bpp 16:9 YUY2
Campo de visión	70° (HOR) x 60° (VER)
Cuadros por segundo (Framerate)	30 fps
Rango operativo de medición	De 0.5 m a 4.5 m
Tamaño del objeto por pixel (GSD)	Entre 1.4 mm (@ rango 0.5 m) y 12 mm (@ rango 4.5 m)
Conexión	USB 3.0
Audio	48 kHz

4.2. Robot Operating System (ROS)

ROS es un *framework* para robots que está bajo la licencia *open source* BSD [36] y que provee librerías y herramientas para la creación de aplicaciones para robots, así como abstracción de software, controladores de dispositivos, herramientas de visualización, administración de paquetes, entre otros. La principal ventaja de ROS es que trabaja de manera modular, donde cada módulo es un programa que realiza una función en específico dependientemente o independientemente de otros módulos. ROS permite comunicar los distintos procesos de manera sencilla.

4.2.1. Paquetes

Los paquetes son la forma en la que se organiza ROS y consisten en carpetas que pueden contener los códigos de los nodos, conjuntos de datos, archivos de configuración, algún tipo de software de terceros, etc. El objetivo de estos paquetes es proveer funcionalidad de manera sencilla para que puedan ser reutilizados y modificados fácilmente por los miembros de la comunidad.

4.2.2. Nodos

Un nodo es un proceso que realiza cálculos. Estos se combinan entre sí formando un grafo, a su vez estos se comunican entre sí haciendo uso de tópicos, servicios y del servidor de parámetros. Un sistema de control de un robot usualmente tiene varios nodos, por ejemplo, un nodo se encarga del movimiento de las ruedas del robot, otro se encarga de la planeación del movimiento, otro de la localización, etc.

El uso de nodos tiene varias ventajas ya que los sistemas de ROS son más tolerantes a los fallos ya que los errores se encuentran asilados en sus respectivos nodos, la complejidad de código es menor en comparación con sistemas monolíticos y finalmente, los nodos pueden ser escritos en Python o C++, lo cual hace más flexible la implementación de estos.

4.2.3. Tópicos

Un tópico es un bus o medio de comunicación por el cual los nodos intercambian mensajes. En general, los nodos no tienen forma de saber con qué otros programas se comunican, en cambio, cuando los nodos requieren algún dato en específico se deben suscribir al tópico con esa información; los nodos que generan datos publican dicha información a los tópicos. Pueden existir varios publicadores y suscriptores para un solo tópico.

Por otro lado, los tópicos están diseñados para comunicación continua y unidireccional, es decir, si se requiere la información de forma no continua, es necesario utilizar algún otro medio de comunicación como los servicios.

4.2.4. Servicios

Los servicios, a diferencia de los tópicos, proveen intercambio de información por medio de un sistema de petición-repuesta entre nodos que se comportan como servidores y clientes. La petición y la repuesta son mensajes definidos en archivos con extensión *.srv. Un nodo de ROS que actúa como servidor ofrece un servicio bajo un cierto nombre, y el nodo cliente llama al servicio mediante un mensaje de petición y después espera la respuesta del servidor. Los servicios son convenientes cuando solo se desea realizar cálculos en determinados momentos o en situaciones específicas. La figura 4.3 muestra la diferencia entre el uso de tópicos y servicios.

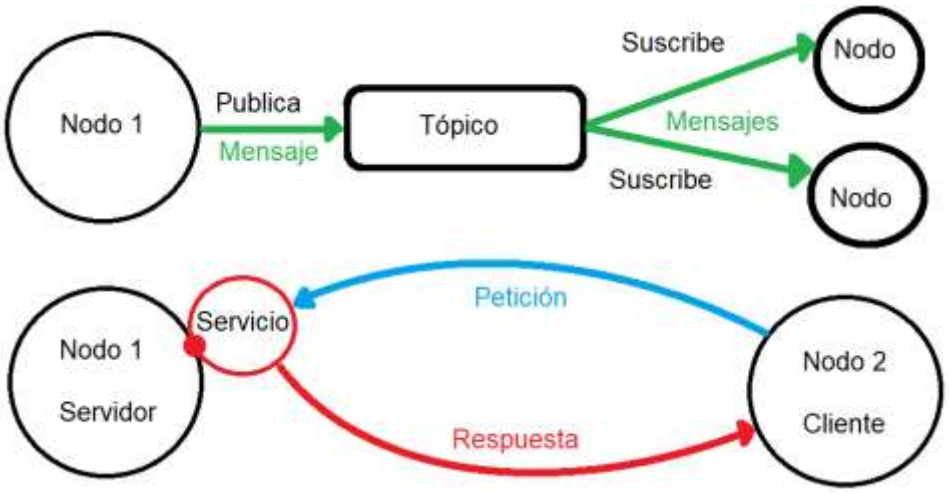

Figura 4.3. Diferencia entre tópico y servicio en ROS

4.2.5. Mensajes

Los nodos de ROS se comunican entre sí publicando mensajes a los tópicos. Estos mensajes son estructuras de datos simples y pueden ser datos enteros, flotantes, booleanos, entre otros, y los cuales se definen en archivos de texto con la extensión *.msg y que especifican los nombres de los mensajes, así como el tipo de datos de estos.

4.3. Reconocimiento de objetos con ROS

Para la recolección de la información proveniente del Kinect se utilizaron los drivers no oficiales para el Kinect 2, los cuales se encuentran en la librería *libfreenect2* [37]. Se utilizaron estos drivers, ya que no existe soporte oficial de Microsoft para el Kinect en Linux. Por su parte, para el reconocimiento de objetos se utilizaron dos paquetes de ROS: *iai_kinect2* y *find_object_2d.*

4.3.1. Librería *iai_kinect2*

La librería *iai_kinect2* [38] contiene una serie de herramientas que publican los datos provenientes del Kinect como las imágenes captadas por la cámara RGB, la cámara infrarroja, la información de profundidad y las nubes de puntos. Estas herramientas son:

- Una herramienta para calibrar el sensor infrarrojo del Kinect 2 con la cámara RGB y las medidas de profundidad.
- Una librería para registrar la profundidad con soporte de OpenCL.
- Un puente entre la librería libfreenect2 y ROS.
- Una herramienta para visualizar las imágenes y las nubes de puntos.

La función más importante de este paquete es trasformar la información obtenida por la librería *libfreenect2* en tópicos de ROS para que, de esta manera, cualquier aplicación desarrollada con ROS pueda acceder a las imágenes captadas por el Kinect.

4.3.2. Librería *find_object_2d*

Esta librería [39] provee una interfaz de usuario en la que se puede utilizar varios algoritmos de detección de características y descriptores, tales como SIFT, SURF, FAST, BRIEF, entre otros. Con una cámara web se pueden reconocer objetos, los cuales, posteriormente son publicados en un tópico de ROS junto con un número identificador y su posición en la imagen (en pixeles).

Si se hace uso de alguna cámara 3D como el Kinect, es posible obtener la posición de los objetos en el espacio. Esta posición se publica en un nodo que puede ser leído por otros paquetes de *ROS*, tales como *Rviz* para visualización, *tf* para transformación de coordenadas y *Moveit* para cálculo de la cinemática inversa.

4.3.3. Algoritmos utilizados

Se utilizaron tres algoritmos, dos de reconocimiento de objetos y uno de detección de objetos. Los algoritmos de reconocimiento fueron una combinación entre el algoritmo FAST y el ORB, y el algoritmo BRISK presentes en el paquete find_object_2d de ROS. El

algoritmo de detección utilizado fue desarrollado por Vázquez en 2018 [10] para el robot de servicio *Justina* [11] y se basa en los algoritmos RANSAC y PCA.

4.3.3.1. FAST (Feature from Accelerated Segment Test)

El algoritmo FAST es un método de detección de esquinas en imágenes desarrollado por Edward Rosten y Tom Drummond en 2006 [40]. Este algoritmo genera un círculo de Bresenham con un radio de tres pixeles alrededor de cada pixel de la imagen. A cada pixel (denotado como *x*) en el perímetro de este círculo se le asigna un número del 1 al 16 ($x \in \{x_1, x_2, \dots, x_{16}\}$) y también se le asigna un estado S_x que depende del valor de intensidad I_x comparado con el del pixel núcleo I_P.

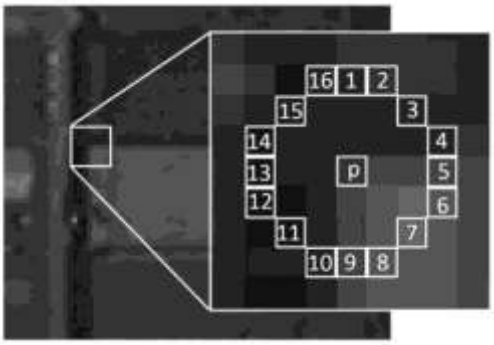

Figura 4.4. Pixeles utilizados en el algoritmo FAST

El algoritmo procede a clasificar como esquina al pixel núcleo *p* si existe un segmento del perímetro (denotado como *L*) de al menos 12 pixeles que sean más brillantes o más oscuros que él, considerando un cierto umbral *T*. De lo contrario, el pixel *p* es calificado como no esquina [41]. Esto es:

$$S_x = \begin{cases} d\ (oscuro), & I_x \leq I_P - T \\ b\ (brillante), & I_x \geq I_P + T \\ s(similar), & I_P - T < I_x < I_P + T \end{cases}$$

El problema con este algoritmo es que se trata solo de un detector, es decir, que no obtiene ningún otro tipo de descripción de los pixeles detectados como esquinas, por lo que cambios en la rotación y escala de una imagen afectan su rendimiento al no haber invariancia de rotación y escala.

4.3.3.2. BRIEF (Binary Robust Independent Elementary Features)

Como su nombre lo indica, este algoritmo es un descriptor de características de uso general. Debido a su naturaleza, es necesario utilizar otros algoritmos para hacer la detección de características, un ejemplo es el algoritmo FAST [42].

La principal ventaja de este algoritmo es que reduce el uso de memoria con respecto a otros algoritmos como SIFT y SURF, ya que estos utilizan vectores de 128 y 64 elementos respectivamente.

Lo que hace BRIEF es que una vez hecha la detección de características con algún otro algoritmo, se crean los descriptores directamente como cadenas de números binarios, las cuales se construyen al hacer comparaciones de intensidad entre los puntos de interés detectados, en áreas (*patches*) de un cierto número de pixeles en la imagen. Esto se refiere a que para cada par de puntos seleccionado se mide la intensidad, si un punto tiene mayor intensidad que el otro, entonces se le asigna un 1, de lo contrario se le asigna un 0.

El número de parejas a comparar puede ser 128, 256 o 512 dependiendo de la precisión y tiempo de procesamiento deseados, y una vez seleccionados, se hace el emparejamiento de puntos entre imágenes utilizando la distancia de Hamming.

En general este algoritmo ofrece buen rendimiento y reconocimiento, sin embargo, es susceptible a cambios no constantes de iluminación y las rotaciones.

4.3.3.3. ORB (Oriented FAST and Rotative BRIEF)

El algoritmo ORB es la combinación entre el algoritmo FAST y BRIEF con algunas modificaciones para compensar las debilidades de ambos y mejorar su rendimiento. Se creó como una alternativa de licencia libre a los algoritmos SIFT y SURF, ya que estos tienen licencias patentadas y solo se pueden utilizar en aplicaciones académicas [43], [44], [45].

Para la detección de características, como se mencionó anteriormente, se utiliza FAST, sin embargo, estas no son invariantes respecto a escala y rotación. Por esta razón y para resolver este problema se crea una pirámide de escala utilizando la imagen a diferentes resoluciones y así buscar las características con FAST en cada nivel de la pirámide.

Posteriormente se calcula la orientación de cada punto de interés. Para ello se calcula el centroide de intensidad en un área (*patch*) alrededor de cada punto y se asume este se encuentra desfasado respecto al punto de interés. El centroide se calcula con los momentos del área de la siguiente forma:

$$m_{pq} = \sum_{x,y} x^p y^q I(x,y)$$

Posteriormente, con esos momentos se calcula la posición del centroide, así como su orientación:

$$C = \left(\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}}\right)$$

$$\theta = \arctan\left(\frac{m_{01}}{m_{10}}\right)$$

Los descriptores se generan con algoritmo BRIEF, pero haciendo algunas modificaciones. La primera es que se hace un suavizado con la integral de la imagen antes de hacer las pruebas.

Una vez obtenidos los pares de puntos y las cadenas binarias, se genera una matriz S de $2 \times n$, donde n es el número de pares y (x,y) son los coordenadas de los puntos.

$$S = \begin{pmatrix} x_1 & x_2 & \cdots & x_n \\ y_1 & y_2 & \cdots & y_n \end{pmatrix}$$

Esta matriz S se multiplica con una matriz de rotación para obtener los valores de los descriptores cada 12°. Con estos valores se crea una tabla de valores pre calculados y de esta manera, mientras existan cambios de orientación consistentes en todos los puntos (que los giros se hagan respecto a un solo plano), los descriptores serán invariantes respecto a la rotación.

Finalmente, se seleccionan los pares de puntos con alta varianza, media cercana a 0.5 y poca correlación. Esto se hace para el algoritmo sea más discriminatorio y menos sensible a valores atípicos.

4.3.3.4. BRISK (Binary Robust Invariant Scalable Keypoints)

Desarrollado por Stefan Leutenegger, Margarita Chli y Roland Siegwart Roland en 2011 [46], este algoritmo es una propuesta que busca obtener precisión y exactitud similares a las de otros algoritmos de visión como SURF, así como una invariancia de rotación y escala, pero a un costo computacional menor. El método se compone en tres partes: 1. la detección de los puntos clave en el espacio-escala, 2. la descripción de los puntos clave y 3. la construcción de los descriptores.

1. *Detección de puntos clave en el espacio-escala:*

La metodología de detección está basada en algoritmo AGAST de Mair et al. [47], el cual es una extensión de desempeño acelerado del algoritmo FAST. Con el fin de obtener invariancia de escala, se realiza una búsqueda de máximos en el plano de la imagen y en el espacio-escala con el parámetro de score *s* del algoritmo FAST como medida de prominencia, además, se estima el valor la escala de cada punto de interés en el espacio-escala continuo.

Las capas de la pirámide del espacio-escala se construyen con *n* octavas c_i y *n* intraocatavas d_i con $i = \{1,2,\ldots,n-1\}$ y generalmente $n = 4$. Estas octavas se construyen al escalar progresivamente la imagen original (c_0) a la mitad. La primer intraocatva d_0 se obtiene al subescalar la imagen original por un factor de 1.5 y las subsiguientes se obtiene al escalar a la mitad sucesivamente d_0. De esta manera, si *t* denota el valor de la escala, entonces $t(c_i) = 2^i$ y $t(d_i) = (1.5)(2^i)$

Por otro lado, en BRISK se utiliza una máscara de 9 - 16, lo que significa que se requieren al menos 9 pixeles consecutivos en un círculo de 16 pixeles sean lo suficientemente brillantes u oscuros que el pixel central. Inicialmente, el detector FAST con mascara 9-16 se aplica a cada octava e intraoctava con el mismo valor del umbral *T* para identificar posibles regiones de interés. Después, a los puntos en dichas regiones se les aplica una "supresión de no máximos" en el espacio-escala que consiste en que el punto a evaluar debe tener el valor máximo *s* (máximo umbral considerando dicho punto como una esquina) con respecto a sus 8 vecinos en la misma capa del espacio-escala. Después se verifica si esta condición se cumple con respecto a la capa superior e inferior.

Posteriormente, cada máximo detectado es ubicado en parches de score de 3x3, y se aplica un refinamiento de escala continua al ajustar una función cuadrática en cada uno de los tres parches de score (uno en la capa del punto de interés, uno en la capa de arriba y otro en la de abajo), los cuales son refinados y se usan para ajustar una parábola y así estimar el valor máximo del score y la escala. Finalmente, con esta información se interpolan las coordenadas de la imagen con los valores máximos (Figura 4.5).

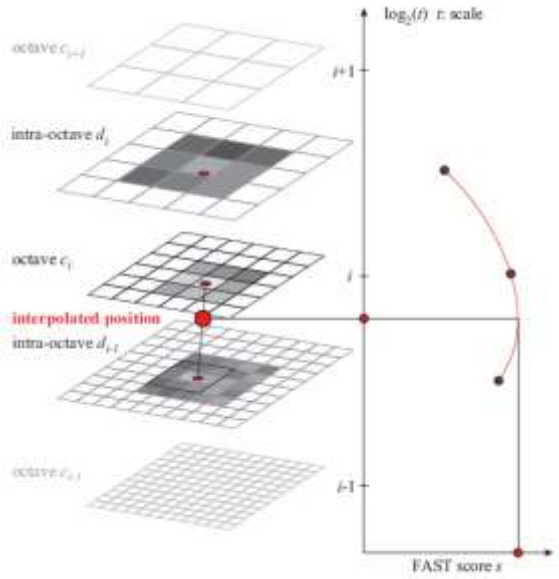

Figura 4.5. Espacio-escala de BRISK y a forma en la que se obtiene el valor de la escala para cada punto de interés [46]

2. *Descripción de los puntos clave:*

El descriptor BRISK se compone como una cadena binaria que concatena los resultados de test de brillo, lo que reduce el tiempo de computación. Para realizar dichos test se hace uso de un patrón para muestrear la vecindad del punto clave (figura 4.6) y después, a cada punto p_i se aplica un suavizado gaussiano con una desviación estándar σ_i proporcional a la distancia entre los puntos del círculo. Para un punto en particular *k* se toma uno de los $N(N-1)/2$ pares (p_i, p_j). Los valores de intensidad suavizados son $I(p_i, \sigma_i)$ y $I(p_j, \sigma_j)$ respectivamente y son utilizados para estimar el gradiente local $g(p_i, p_j)$ con:

$$g(p_i, p_j) = (p_j - p_i)\frac{I(p_j, \sigma_j) - I(p_i, \sigma_i)}{|p_j - p_i|^2}$$

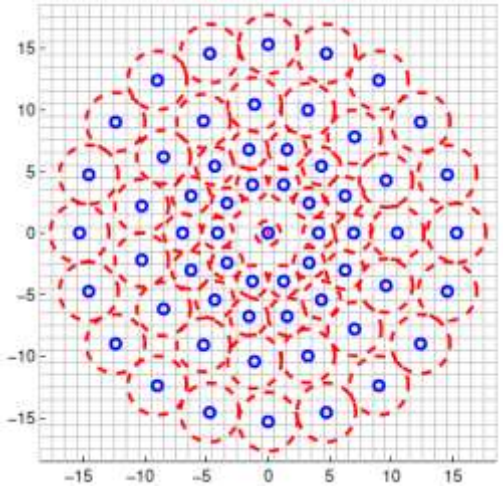

Figura 4.6. Patrón de muestreo utilizado en BRISK. Los puntos azules denotan las localizaciones del muestreo y las líneas rojas denotan la desviación estándar del kernel gaussiano

Posteriormente, se definen dos conjuntos de parejas de puntos con una distancia obtenida a partir de dos umbrales δ_{max} y δ_{min}, un conjunto con poca distancia entre sí (S) y otro con una mayor distancia (L). Dichos valores umbrales están definidos como $\delta_{max} = 9.75t$ y $\delta_{min} = 13.67t$, donde t es la escala de k iterando sobre el conjunto L: Finalmente se estima la dirección g del punto clave k:

$$g = \begin{pmatrix} g_x \\ g_y \end{pmatrix} = \frac{1}{L} \sum_{(p_i, p_j) \in L} g(p_i, p_j)$$

Se itera sobre el conjunto L, ya que los gradientes del conjunto A se eliminan entre sí y por ende no son necesarios para calcular el gradiente global.

3. *Construcción de los descriptores*:

Para la formación del descriptor se aplica el patrón de muestra rotado en un ángulo con valor de $\alpha = \arctan2\left(\frac{g_y}{g_x}\right)$ alrededor del punto k. El descriptor binario d_k se ensambla al realizar comparaciones de intensidad entre los pares de puntos rotados y con poca distancia entre sí (subconjunto S). Estos pares son denotados como: $(p_i^\alpha, p_j^\alpha) \in S$. Las comparaciones se hacen de tal manera que el bit *b* corresponda a:

$$b = \begin{cases} 1, & I(p_j^\alpha, \sigma_j) > I(p_i^\alpha, \sigma_i) \\ 0, & en\ caso\ contrario \end{cases} ;\ \forall (p_i^\alpha, p_j^\alpha) \in S$$

Finalmente, para saber si dos descriptores pertenecen al mismo punto clave en diferentes imágenes (*matching*), se determina la distancia de Hamming, es decir, se aplica una operación XOR a ambos vectores para saber qué tan similares son y se cuentan los bits que son distintos; si son pocos, entonces se trata del mismo descriptor.

4.3.3.5. Detector RANSAC-PCA

Desarrollado por Vázquez S. en 2018 [10] con el objetivo de dotar al robot de servicio *Justina* [11] con un sistema de detección de objetos, el cual se basa en el algoritmo RANSAC y el PCA. El primero es utilizado para segmentar los planos de la imagen captada por el Kinect 2 y para extraer los objetos; el segundo se usa para determinar la orientación de los objetos extraídos. En este caso, los datos con los que se trabaja son con la nube de puntos que proporciona el Kinect a través de su sensor infrarrojo. Si bien el Kinect 2.0 puede otorgar imágenes a una resolución de 1920x1080, en este caso se utilizó una resolución de 960x540 para reducir el tiempo de procesamiento. El método se compone de tres partes: 1. Detección de planos, 2. Extracción de objetos y 3. Determinación de la orientación con PCA.

1. Detección de planos:

Para obtener el modelo del plano con el algoritmo RANSAC, se seleccionan tres puntos de la matriz de forma aleatoria con los cuales se construyen vectores y se obtiene el producto cruz entre estos.

$$p_1, p_2, p_3 = random(nube\_de\_puntos)$$

$$\bar{v}_1 = p_1 - p_2\ ;\ \bar{v}_2 = p_1 - p_3$$

$$\bar{N} = \bar{v}_1 \times \bar{v}_2$$

A partir de la ecuación general del plano: $Ax + By + Cx + D = 0$, se tiene lo siguiente:

$$D = -(Ap_{1x} + Bp_{1y} + Cp_{1z})$$

Donde A, B y C son las componentes del vector normal al plano $\bar{N}$ y p_x, p_y, p_z son las componentes de algún punto perteneciente al plano (en este caso se toma p_1 pero puede ser cualquiera de los tres puntos).

A partir de este primer modelo, se empieza determina qué tan bueno es considerando como error a la como la distancia euclidiana de cada punto al plano, si dicho error está dentro de un cierto umbral, se considera como parte del modelo y se incrementan los puntos considerados como valores típicos (*inliers*) hasta hallar el modelo del plano.

El plano calculado es el más prominente en la imagen, de momento este algoritmo no distingue entre planos horizontales y verticales, por ello es necesario tener cierto control en el ambiente donde se realizan la detección y procurar que solo un plano quede en escena.

Figura 4.7. Plano detectado con RANSAC

2. Extracción de objetos:

La extracción de objetos se hace al desechar los puntos pertenecientes al plano y aquellos que no son parte del plano ni del objeto. Para determinar qué puntos no son un componente del objeto o del plano, se considera la siguiente condición:

$$p_{extra} = \{(x,y,z) | (x < 0.5\ [m]) \wedge (z < p.z), \forall p \in p_{plano}\}$$

Esta condición elimina los puntos más allá de 50 [cm] respecto al punto más cercano en *x* y a los puntos que se encuentran por debajo del plano. Los puntos pertenecientes al objeto son calculados y extraídos (Figura 4.8) con la siguiente manera:

$$p_{objeto} = p_{totales} - p_{plano} - p_{extra}$$

Figura 4.8. Izquierda: Objeto extraído, Derecha: Puntos pertenecientes al objeto

3. Determinación de la orientación con PCA:

Se aplica el PCA sobre los puntos del objeto para obtener su centroide y su orientación. La base ortogonal formada por los vectores propios de la matriz de covarianzas representa la distribución de los puntos del objeto y por ende su orientación.

Para calcular el centroide, se obtiene la media de las coordenadas de los objetos del objeto. Cabe destacar que esta forma de cálculo puede no representar con exactitud su valor real, ya que la información de los puntos depende de la perspectiva a la que se encuentra la cámara con respecto al objeto. Una vez calculado se encuentra la matriz de covarianza de los puntos pertenecientes al objeto. Después se sigue la metodología del método PCA y se calculan los valores y vectores propios de esta matriz. Estos vectores representan la dirección del objeto y a partir de estos se calculan l los giros roll, pitch y yaw (RPY).

Capítulo 5 Simulación

Una vez detectados los objetos con la librería *find_object_2d*, su posición y orientación son publicadas en un tópico. Esta información es utilizada por otras librerías de ROS como Rviz para visualización, Gazebo para simulación y Moveit para la resolución de la cinemática inversa.

5.1. Rviz

Es una interfaz gráfica que permite la visualización de la información contenida en los tópicos de ROS [48]. Entre las tareas que se pueden realizar en Rviz están la visualización modelos de robots, árboles de transformaciones, nubes de puntos, videos, mapeos de escenas y localización, navegación, etc. También es posible agregar plugins personalizados para realizar otro tipo de tareas.

Si bien Rviz es una herramienta de visualización muy útil, no es capaz de hacer simulaciones ya que carece de un sistema de físicas. El uso que se le dio a esta librería fue para fines de visualización del modelo del robot, y de la nube de puntos. También se utilizó el plugin de Moveit para Rviz para interactuar con el modelo del robot.

5.2. Gazebo

Gazebo es un software gráfico de código abierto creado por el Dr. Andrew Howard y Nate Koenig [49], cuyo principal objetivo es la simulación de robots en ambientes dinámicos y con múltiples sistemas de físicas (ODE, Bullet, Simbody y DART). Algunas de las características de este simulador se encuentran su motor de renderizado avanzado, su repositorio con una gran variedad de modelos de robots comerciales, su capacidad de simular cámaras y sensores para aplicaciones de reconocimiento de imágenes y su soporte para plugins.

Todo lo anterior hace a Gazebo una herramienta útil para tareas que varían desde la aplicación de algoritmos de control en robots sin la necesidad de poseer físicamente al robot, creación de modelos de robots, hacer mapeos de escenarios reales en tiempo real o incluso entrenar sistemas de inteligencia artificial. Aunado a eso, Gazebo es que se puede integrar con ROS y hacer simulaciones con los robots creados con en este *framework*, lo cual se vuelve útil ya que Rviz solamente es una herramienta de visualización.

Este programa se utilizó en conjunto con las librerías de ROS: *tf, Moveit* y *joint_state_publisher* para simular el comportamiento del robot en un ambiente con propiedades físicas como gravedad, fricción y amortiguamiento.

5.3. Simulación de la cinemática inversa

Si bien la cinemática inversa se puede resolver utilizando varios métodos como el geométrico o desacople cinemático, se optó por utilizar el paquete de ROS llamado Moveit para resolver la cinemática inversa de forma bastante sencilla.

5.3.1. Moveit

Moveit [31] es un *framework* para robots de código abierto, desarrollado para ROS, que ofrece las siguientes características: planeación de movimiento, manipulación, resolución de la cinemática inversa, control, percepción 3D y chequeo de colisiones. Este paquete también ofrece un plugin para Rviz que permite controlar de forma interactiva la posición del robot, ya sea en una simulación o con el robot físico y también es posible hacer simulaciones en Gazebo con los controladores de bajo nivel de ROS Control.

Otra ventaja es que existe un repositorio con una gran variedad de robots comerciales listos para ser utilizados, pero en caso de se necesite desarrollar el modelo de un robot que no esté en este catálogo, Moveit posee un asistente para configurar cualquier robot descrito en formato URDF para que se pueda utilizar en este paquete.

La interfaz de Moveit integra, además, algunos tipos de planeadores de movimiento. Estos son:

- Open Motion Planning Library (OMPL)
- Search-Based Planning Library (SBPL)
- Covariant Hamiltonian Optimization for Motion Planning (CHOMP)
- Stochastic Trajectory Optimization for Motion Planning (STOMP)

La arquitectura de Moveit se basa en el nodo *move_group* (Figura 5.1), el cual actúa como un integrador de los diversos elementos que componen esta librería para proveer acciones y servicios de ROS para uso del usuario.

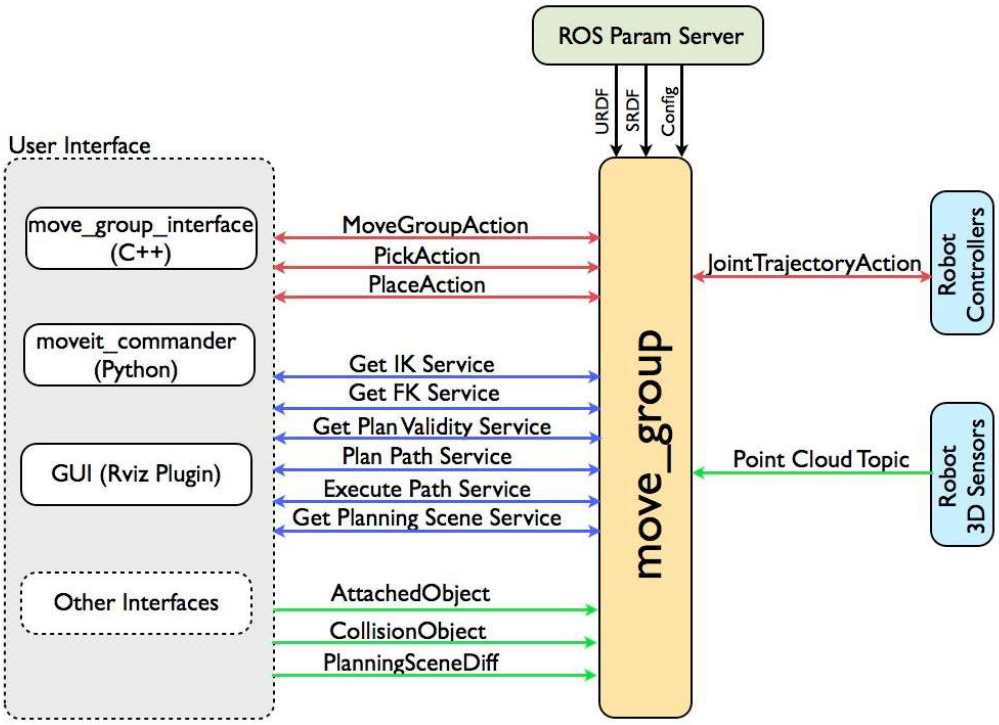

Figura 5.1. Arquitectura de Moveit

Capítulo 6 Metodología

En este apartado se describen las diversas metodologías para el desarrollo del proyecto, que van desde la calibración del Kinect a las pruebas realizadas para obtener la precisión y orientación de los algoritmos de reconocimiento de objetos. Las pruebas fueron realizadas en una computadora con sistema operativo Ubuntu 18.04 LTS y los siguientes programas instalados: *ROS Melodic* (Con las librerías *iai_kinect2, Moveit y find_object_2d*), *Gazebo* y *libfreenect2*.

6.1. Calibración del Kinect 2

Primero el Kinect se colocó en un trípode para poder controlar su posición y altura. La calibración del Kinect se realizó con la librería de ROS *iai_kinect2* para que las imágenes registradas por la cámara RGB y la nube de puntos generada por el sensor infrarrojo se sincronizaran. Se utilizó un patrón de calibración tipo ajedrez 7x8 con cuadros de 2.5 cm pegado a una superficie plana.

El procedimiento para la calibración fue el siguiente:

1. Correr el programa kinect2_bridge a un número bajo de cuadros por segundo (por ejemplo 2 FPS).
2. Grabar y tomar fotos del patrón de calibración con la cámara RGB a diferentes distancias y ángulos.
3. Calibrar los parámetros internos (*intrinsics*) de la cámara RGB.
4. Repetir el paso 2 pero con la cámara infrarroja.
5. Calibrar los parámetros internos de la cámara infrarroja.
6. Repetir el paso 2 pero con ambas cámaras a la vez. En este caso, las fotos deben ser tomadas cuando el patrón aparezca en ambas cámaras.
7. Calibrar los parámetros externos (*extrinsics*).
8. Calibrar las medidas de profundidad.
9. Copiar los archivos con las matrices de distorsión calculadas en el directorio *data* de kinect2_bridge.

En promedio se tomaron 120 fotos por paso, por lo que al final de cada iteración se obtuvieron alrededor de 350 fotos (Figura 6.1). Por otro lado, al final del proceso, los programas devolvieron los porcentajes de error en los cálculos, los cuales fueron útiles para determinar y seleccionar la calibración que presentaba menos porcentajes de error. Los datos obtenidos en la calibración serán guardaron para su posterior análisis.

Figura 6.1. Ejemplo de las fotos tomadas al patrón para la calibración. Derecha: Imagen RBG, Izquierda: Imagen Infrarroja

6.2. Modelo URDF y SRDF

A continuación, se explica cómo se obtuvo el modelo del robot en formato URDF con el plugin de Solidworks: *sw_urdf_exporter* [50]. Primero se obtuvo el modelo CAD del robot en la página oficial de KUKA, el cual está constituido de varias partes unidas como un solo cuerpo, sin embargo, el plugin necesita un archivo tipo ensamble para poder exportar el modelo. Para ello, se separaron las partes y se guardaron en diferentes archivos para posteriormente realizar el ensamble con los mismos y definir los sistemas de referencia para cada articulación.

Una vez creado el ensamble se definieron de manera semiautomática las propiedades (masa e inercias) y los parámetros (posiciones, tipo de articulación, límites de seguridad, etc.) de las partes del robot con el asistente para exportación del plugin. Al final del procedimiento se obtuvo una carpeta con los archivos necesarios para conformar el paquete ROS. Sin embargo, y debido a que varios parámetros se calcularon de forma automática, se presentaron errores en el archivo URDF generado, por lo que se realizaron los ajustes necesarios para que el archivo trabajara correctamente.

Una vez generado el archivo URDF, se procedió a crear el archivo SRDF (*Semantic Robot Description Format*). Este formato complementa al URDF y especifica grupos de articulaciones, configuraciones default del robot, información adicional sobre las colisiones y transformaciones adicionales que se puedan necesitar para determinar completamente la pose del robot. Para obtener el archivo SRDF se utilizó el asistente de Moveit, cuya función, aparte de generar dicho archivo, es generar otros archivos necesarios en un nuevo paquete de ROS para poder utilizar el robot con esta librería. En este asistente se calcularon y especificaron los siguientes elementos:

- Matriz de autocolisión: Matriz generada al buscar pares de eslabones con los que se puede deshabilitar de forma segura el chequeo de colisión para reducir el tiempo de procesamiento.
- Articulaciones virtuales: Su única función es unir al robot con el mundo para que no se desplace en las simulaciones.
- Grupos de planeación: Grupos utilizados para definir partes diferentes del robot, como los eslabones y articulaciones que componen al brazo y el efector final.
- Poses del robot: Se definen algunas poses predefinidas que se deseen poseer, tales como una posición principal (*home*), una contraída, etc.
- Efector final: Se designa el grupo especial que va a actuar como efector final.
- Articulaciones pasivas: Articulaciones del robot que no son actuadas nunca. En nuestro caso no se definió ninguna articulación pasiva.
- Percepción 3D: Se definen los tópicos y parámetros necesarios para que Moveit pueda detectar la información de profundidad del Kinect para que pueda hacer planeación de trayectorias de los elementos presentes en el ambiente.
- Simulación en Gazebo: Genera las líneas faltantes en el archivo URDF para que este pueda ser utilizado por Gazebo para hacer simulaciones.
- ROS Control: Se seleccionó un controlador de posición de bajo nivel para poder actuar el robot en la simulación.

Con los parámetros anteriores bien definidos se generó un nuevo paquete de ROS para utilizar el robot en Moveit y poder observar su comportamiento en Gazebo.

6.3. Pruebas iniciales con programas

Con el Kinect 2 calibrado y funcional, se realizaron pruebas con algoritmos y programas de detección y reconocimiento de objetos tales como YOLO, ORK y find_object_2d, todo para comprobar compatibilidades con el software y para identificar errores que pudieran impedir el uso viable de los mismos. A continuación, se describen dichos programas.

6.3.1. YOLO (You Only Look Once)

YOLOv3 [4] es un estado de arte para reconocimiento de objetos en tiempo real que se distingue de otros en que utiliza una red neuronal que divide la imagen en regiones y predice los cuadros delimitadores (*bounding boxes*) y probabilidades de cada región, lo cual le confiere una gran velocidad pero disminuye la precisión.

Para las pruebas se utilizó el modelo que viene por default con este programa y se observó que tiene un buen reconocimiento, pero su procesamiento es muy lento (~0.2 FPS). Esto

puede ser modificado al aumentar la velocidad de reconocimiento con la GPU pero únicamente es compatible con GPUs de NVIDIA, lo cual impidió su uso.

6.3.2. ORK (Object Recognition Kitchen)

Object Recognition Kitchen (ORK) [51] es un proyecto iniciado en Willow Garage que compila diferentes algoritmos y técnicas de reconocimiento de objetos para su uso en situaciones diversas, por ejemplo, cuando hay objetos con o sin texturas, transparentes, articulados, etc. Para las pruebas se buscó detectar una caja de cartón pequeña y una lata de refresco, mientras que, para entrenar la base de datos, se crearon modelos en formato *.stl de los objetos en cuestión.

Cuando se realizaron las pruebas, se probaron diferentes distancias y orientaciones del Kinect a los objetos, sin embargo, el programa no detectó ningún objeto, lo cual se asoció a la posible incompatibilidad debido a la desactualización de la librería con la versión de ROS utilizada (ROS Kinectic). Esto descartó la opción para realizar el reconocimiento de objetos.

6.3.3. find_object_2d

Como se explicó en la sección 4.3.2, *find_object_2d* es una librería de ROS para el reconocimiento de objetos [39]. La ventaja en su uso radica en que este paquete tiene a su disposición varios algoritmos de reconocimiento de objetos mediante características como SURF, SIFT, FAST, BRIEF, ORB, entre otros. Razón por la cual fue la librería seleccionada para realizar las pruebas, además de que es sencilla de usar y permite obtener la posición en el espacio de los objetos reconocidos. Por otro lado, se ha observado que si bien, puede presentar problemas al reconocer objetos con formas complejas funciona adecuadamente con objetos estáticos con formas simples.

6.4. Diagrama de los programas

Debido a la naturaleza modular de ROS, el sistema se dividió en varios módulos con funciones diferentes e interconectados entre sí. La arquitectura del sistema (figura 6.2) es la siguiente:

- Módulo de visión: Incluye *libfreenect2, iai_kinect2* y *find_object_2d*. Los dos primeros son necesarios para obtener datos del Kinect 2 y el último para hacer el reconocimiento de objetos.
- Módulo de cálculo de trayectorias: Compuesto por *Rviz* y *Moveit* para el cálculo de trayectorias y la resolución de la cinemática inversa.

- Módulo de control: *ros_control* permite la implementación de controladores de bajo nivel y *ros_industrial* contiene librerías necesarias para controlar el robot.
- Módulo de simulación: Conformado por *Rviz* para propósitos de visualización y Gazebo para la simulación en un ambiente virtual con propiedades físicas.
- Módulo de acción: Compuesto por la librería de *ROS kuka_experimental* que permite hacer la conexión entre *ROS* y el controlador CR4 del robot.

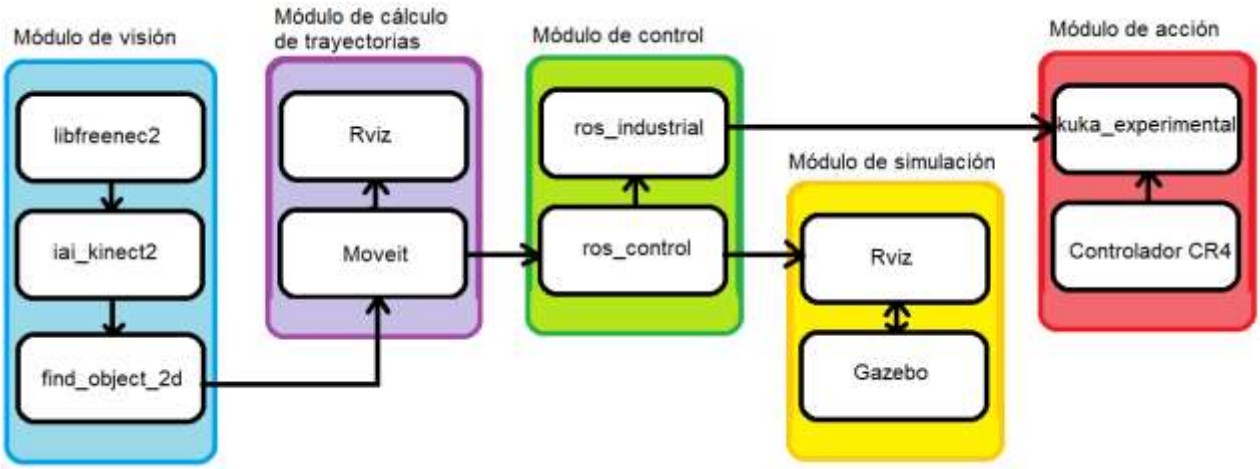

Figura 6.2. Diagrama de los programas utilizados y sus relaciones entre sí

6.5. Conexión entre la computadora y el robot

El procedimiento que se pretendía realizar era que la computadora se conectase al controlador KUKA CR4, haciendo uso de la librería de ROS llamada *kuka_experimental* y de la interfaz RSI (Robot Sensor Interface) del controlador, asegurándose primero que tanto la computadora como el robot se encuentren en la misma red local.

Después, configurar y pegar algunos archivos KRL en la interfaz RSI del controlador. En estos archivos se especifican parámetros como la dirección IP de la computadora, el número del puerto de conexión, los límites mínimos y máximos de las articulaciones, la posición inicial y el tiempo de procesamiento. De la misma manera, se modificaron algunos archivos de la librería *kuka_experimental,* asegurándose que los parámetros introducidos en los archivos KRL fuesen los mismos.

NOTA: Lo anterior quedará pendiente ya que no fue posible utilizar el manipulador en el laboratorio por cuestiones de la pandemia.

6.6. Metodología de las pruebas

Primero se seleccionaron cuatro objetos de control con dimensiones y centros de masa conocidos (esfera, una lata de refresco, un cubo rubik y una caja de leche) (figura 6.3).

Figura 6.3. Objetos de control. De izquierda a derecha: Caja, esfera, lata y cubo rubik

Dichos objetos se colocaron sobre una mesa ubicada frente al robot, mientras que el Kinect se colocó frente a estos (figura 6.4). Los objetos fueron posicionados en coordenadas conocidas y relativas a la base del robot en el suelo, la cual actuó como el origen del sistema de coordenadas global. Las medidas y las posiciones de los centroides de los objetos se encuentran en la tabla 6.1.

TABLA 6.1. MEDIDAS Y POSICIÓN DE LOS CENTROIDES DE LOS OBJETOS

	Ancho [cm]	**Largo [cm]**	**Altura [cm]**	**Diámetro [cm]**	**Posición X [cm]**	**Posición Y [cm]**	**Posición Z [cm]**
Caja	6.2	4.2	10.6	N/A	9.54	-2.13	7.87
Esfera	N/A	N/A	7.1	6	9.63	-0.71	7.70
Lata	N/A	N/A	12.4	6.5	9.66	0.70	7.96
Cubo	5.7	5.7	5.7	N/A	9.62	2.12	7.63

Figura 6.4. Kinect colocado frente a los objetos

6.6.1. Pruebas de los algoritmos de reconocimiento de objetos

Para las pruebas se utiliza la combinación FAST-ORB, BRISK y el algoritmo de detección RANSAC-PCA. Para cada prueba se busca determinar la posición y la orientación de los objetos para después compararlas con los valores reales y de esta forma obtener los valores de los errores absolutos.

Las pruebas de reconocimiento de objeto son las siguientes: 1) Determinación del espacio de trabajo, 2) Pruebas con los objetos de frente, rotados en un ángulo negativo y rotados en un ángulo positivo, 3) Pruebas con objetos sobrepuestos y 4) Pruebas con objetos horizontales.

6.6.1.1. Determinación del espacio de trabajo.

Para la determinación del espacio de trabajo del Kinect se selecciona un objeto de prueba (caja) y se coloca en posiciones límite hasta que dejase de ser detectado y se toma nota de los valores, los cuales son considerados como los límites del espacio de trabajo. Lo anterior se realiza para cada algoritmo de reconocimiento (FAST-ORB y BRISK), ya que ambos utilizan diferentes métodos para la detección de objetos y por ende el rango del espacio de trabajo es distinto (Figuras 6.5 y 6.6).

Figura 6.5. Posiciones límite del objeto para el algoritmo FAST-ORB

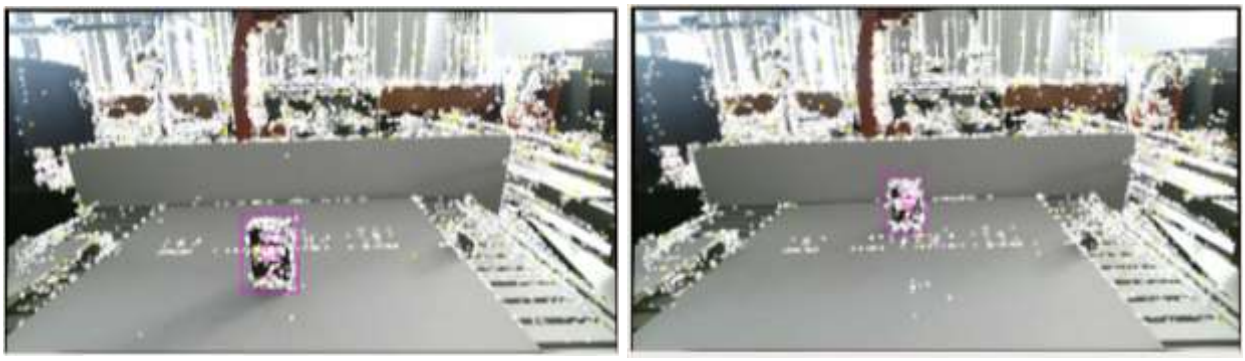

Figura 6.6. Posiciones límite del objeto para el algoritmo BRISK

6.6.1.2. Pruebas objetos de frente y con rotaciones negativas y positivas.

Para estas pruebas se colocan los objetos sobre la mesa en sus respectivas posiciones con un giro de 0° respecto a su eje (Figuras 6.7 y 6.8). Una vez ubicados los objetos, se corren las librerías mediante un archivo con formato *.launch, el cual corre los paquetes de *ROS*: *iai_kinect2, find_object_2d* (para FAST-ORB y BRISK), los paquetes del algoritmo RANSCA-PCA, rviz, moveit y Gazebo.

Como primer análisis se selecciona el algoritmo FAST-ORB para obtener la posición y orientación de cada objeto. Los datos se obtienen al acceder al paquete *tf* de ROS y son guardados para su posterior análisis.

Figura 6.7. Reconocimiento de objetos a 0° (FAST-ORB)

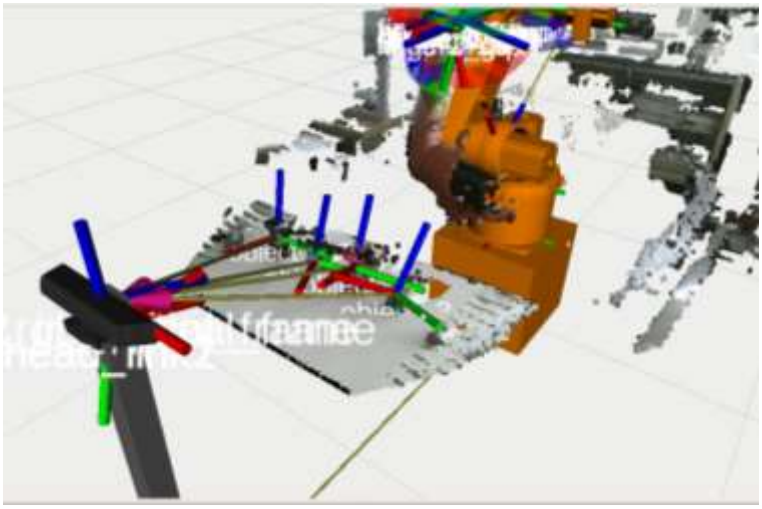

Figura 6.8. Posición y orientación de los objetos a 0°, visto en Rviz (FAST-ORB)

Después, los objetos se giran sobre su eje y se anotan sus respectivas posiciones y orientaciones. Esto se hace primero en un ángulo negativo de -30° para la esfera, lata y cubo, y de -20° para la caja (Figura 6.9); y luego en un ángulo positivo de +10° para la esfera, +20° para el cubo y +30° para la lata y la caja

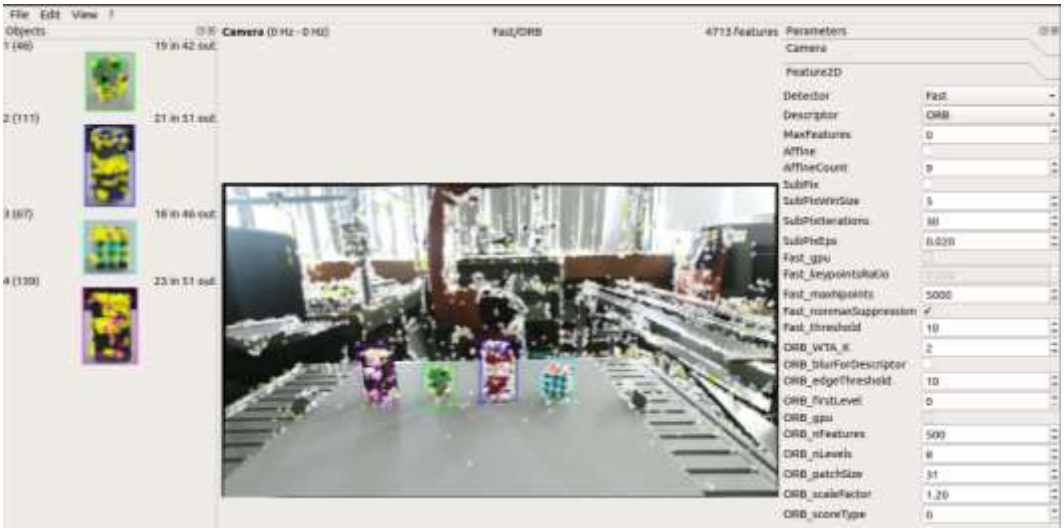

Figura 6.9. Objetos girados un ángulo negativo (FAST-ORB)

En la segunda prueba se utiliza el algoritmo BRISK de forma análoga al anterior, es decir, primero se colocan los objetos en las mismas posiciones con un giro de 0° (Figuras 6.10 y 6.11) y luego se giran en un ángulo negativo y positivo, pero con la diferencia de que estos últimos son menos pronunciados debido a que el algoritmo dejaba de detectar los objetos. Los ángulos negativos son -15° para la lata y el cubo y -10° para la caja, mientras que los positivos son +15° para todos. Por otro lado, en esta prueba la esfera deja de ser detectada por lo que no fue considerada en los análisis.

Figura 6.10. Reconocimiento de objetos a 0° (BRISK)

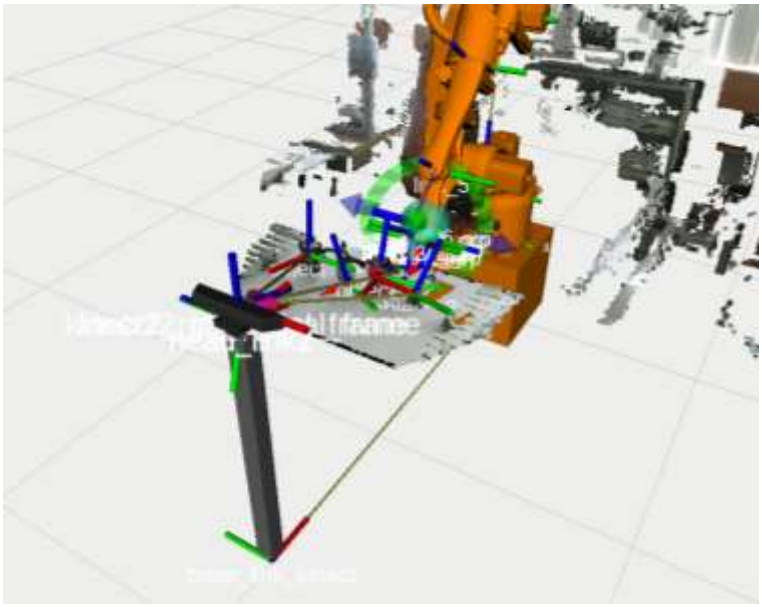

Figura 6.11. Posición y orientación de los objetos a 0°, visto desde Rviz (BRISK)

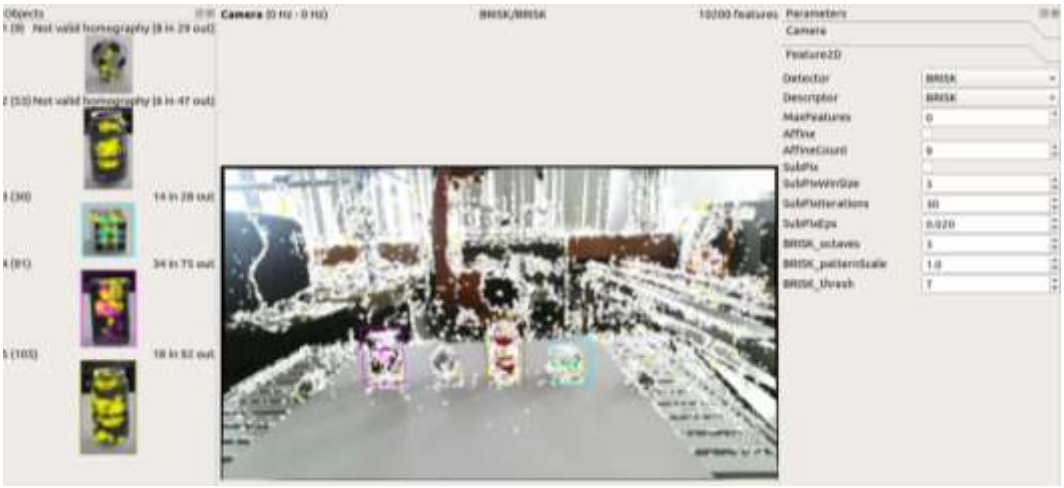

Figura 6.12. Objetos girados en un ángulo postivo (BRISK)

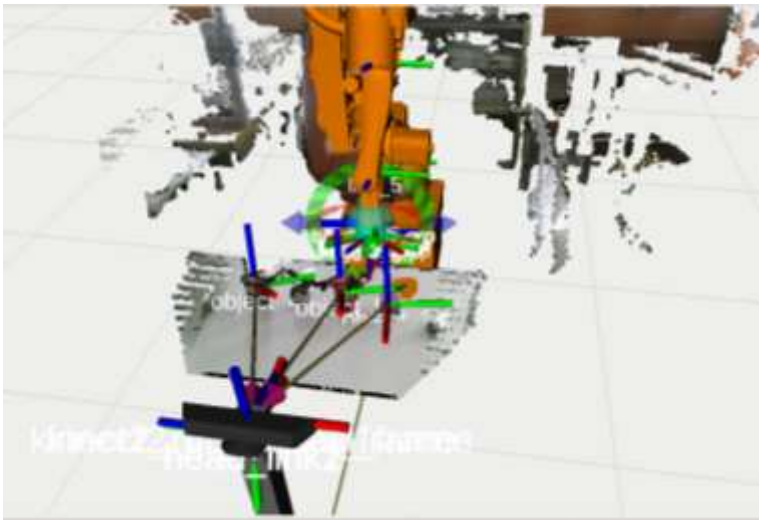

Figura 6.13. Posición y orientación de objetos girados un ángulo positivo, visto desde Rviz (BRISK). Se aprecia que la esfera dejó de ser detectada.

Finalmente, se realizaron pruebas con el algoritmo RANSAC-PCA, el cual solamente es capaz de detectar en solitario cada objeto por lo que las pruebas se hicieron con un objeto a la vez. Primero se obtiene la posición y la orientación de cada objeto con un ángulo nulo (figuras 6.14 y 6.15) y luego se hace lo mismo para la caja y el cubo, pero girados en ángulos de -30° (figuras 6.16 y 6.17) y +30° (figuras 6.18 y 6.19). Se omitieron las pruebas con la esfera y la lata ya que, debido a la naturaleza simétrica de sus formas redondas, al girar dichos objetos sus formas en el plano permanecían constantes, lo que ocasiona que los giros calculados por el algoritmo sean imprecisos e inexactos.

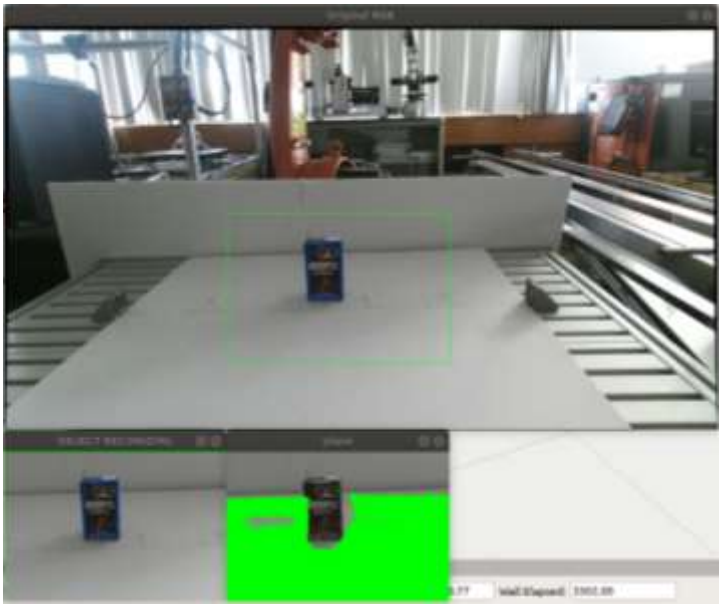

Figura 6.14. Detección de la caja a 0° (RANSAC-PCA)

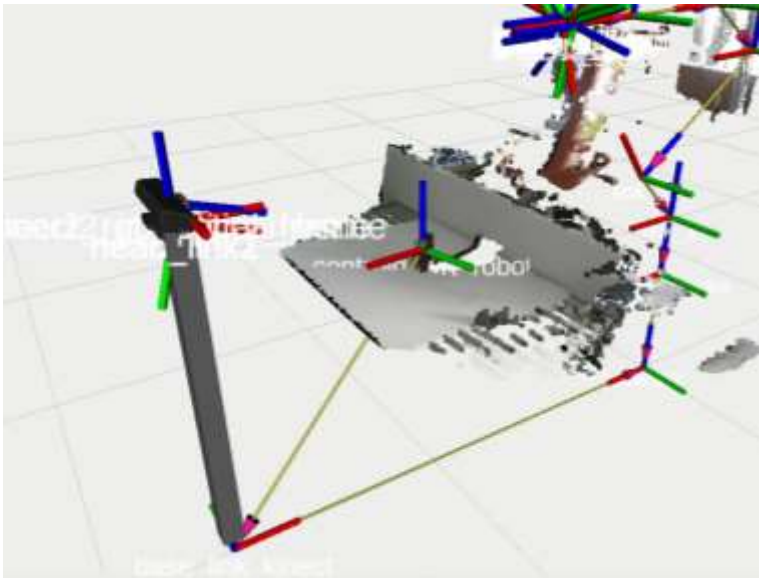

Figura 6.15. Posición y orientación de la caja a 0°, visto desde Rviz (RANSAC-PCA)

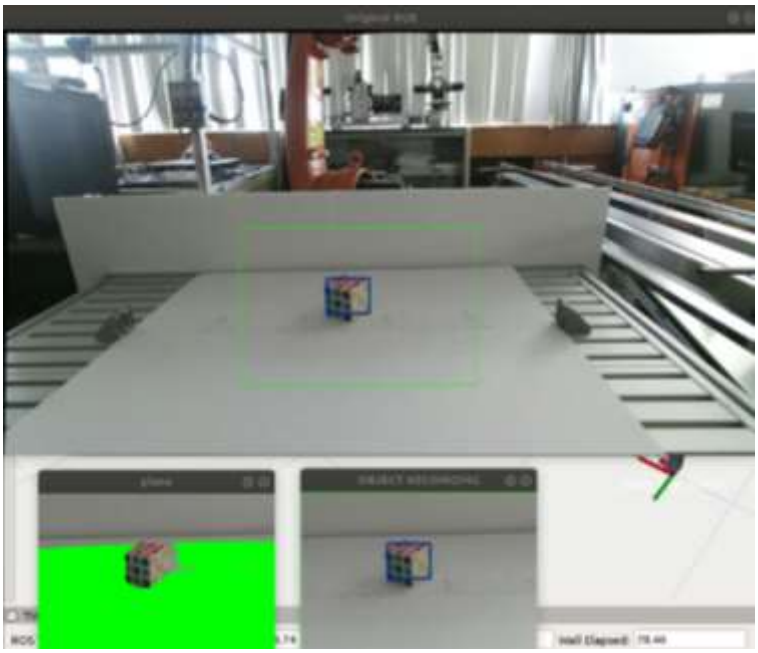

Figura 6.16. Detección del cubo rubik a -30° (RANSAC-PCA)

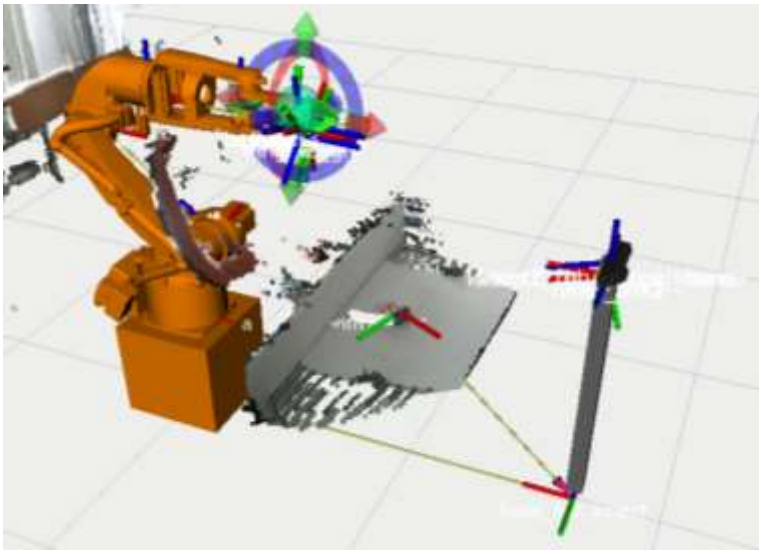

Figura 6.17. Posición y orientación del cubo en un ángulo negativo, visto desde Rviz (RANSAC-PCA)

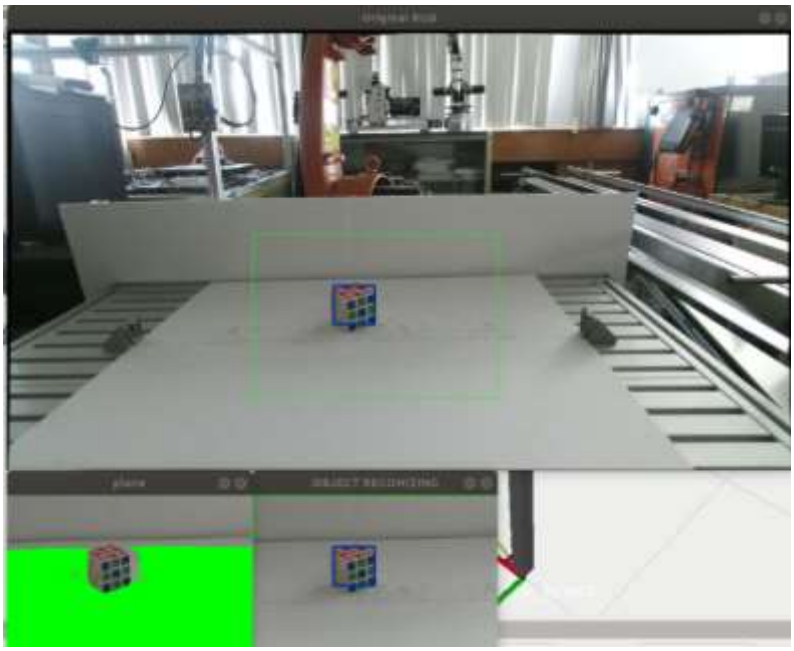

Figura 6.18. Detección del cubo en un ángulo positivo (RANSAC-PCA)

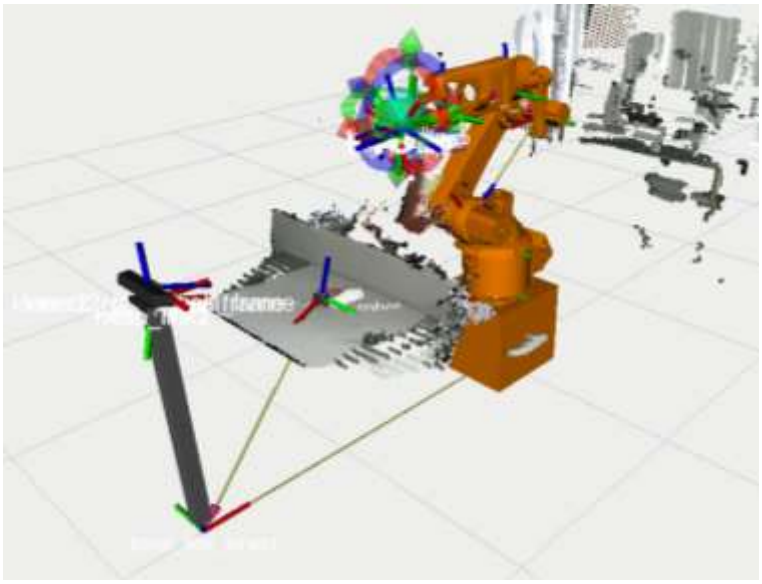

Figura 6.19. Posición y orientación del cubo en un ángulo positivo, visto desde Rviz (RANSAC-PCA).

6.6.1.3. Pruebas con objetos sobrepuestos.

Para esta prueba se busca determinar qué tan precisos y exactos son los algoritmos cuando los objetos (a excepción de la esfera) se encuentran parcial o totalmente detrás de otros. Para esto se coloca un objeto al centro y otro atrás de él, este último ubicado en cinco diferentes posiciones relativas al objeto frontal (completamente a la izquierda, parcialmente a la izquierda, al centro, parcialmente a la derecha y completamente a la derecha) y con sus permutaciones. En cada posición se observa si los algoritmos (FAST-ORB y BRISK) realizan o no la detección del objeto trasero. En total se hicieron dos tablas donde se anotan los resultados de las pruebas, es decir, si se trata de una detección fallida (×), una detección inestable (~) o una detección estable (✓).

Figura 6.20. Pruebas con objetos sobrepuestos. En este caso cubo enfrente y caja atrás (FAST-ORB). De izquierda a derecha las cinco posiciones consideradas: Izquierda completo, izquierda parcial, centro, derecha parcial y derecha completo.

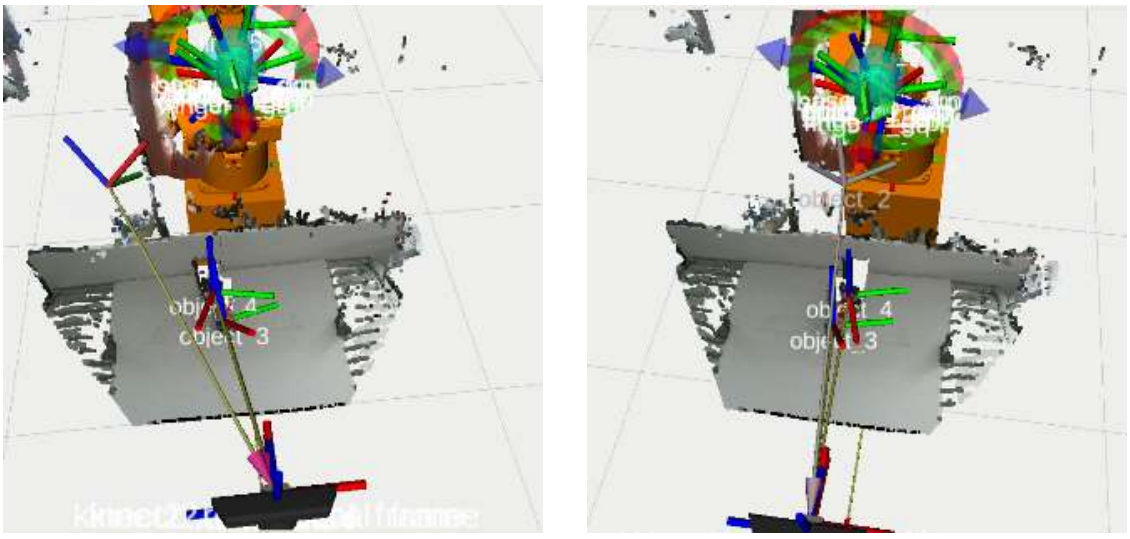

Figura 6.21: Posición y orientación del cubo (enfrente) y la caja (atrás). Izquierda: Parcialmente a la izquierda, Derecha: Parcialmente a la derecha. Visto desde Rviz

6.6.1.4. Pruebas con objetos horizontales

El objetivo de esta prueba es determinar qué tan eficientes son los algoritmos al reconocer objetos horizontales. Para esto se toma la caja como objeto control, ya que su forma cambia con respecto al plano del Kinect cuando es puesta de forma horizontal, a diferencia del cubo y la esfera. Primero se puso la caja de frente y luego horizontalmente en dos posiciones distintas: a +90° y a -90°, siguiendo la regla de la mano derecha (Figura 6.22), en todos los casos se toma su posición y orientación. Este procedimiento se hace para el algoritmo BRISK. El algoritmo FAST-ORB no es capaz de hacer la detección del objeto al ser puesto horizontalmente, por lo que fue descartado para la prueba.

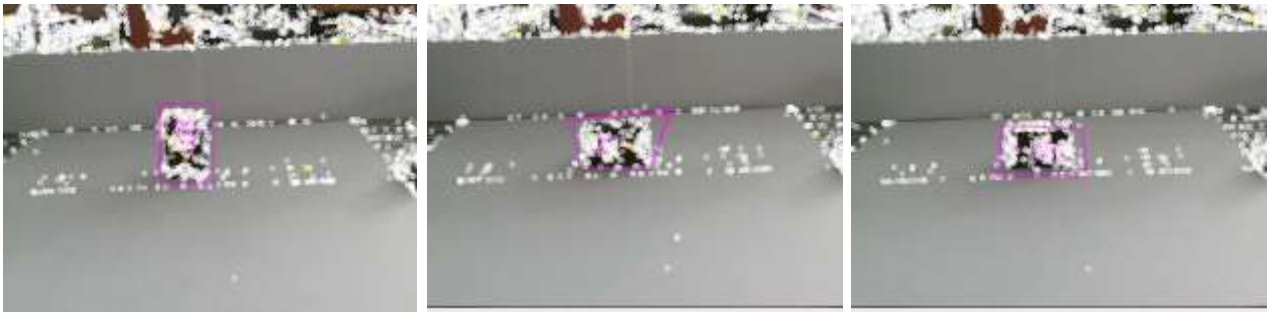

Figura 6.22. Reconocimiento del objeto (caja) horizontal en tres distintas posiciones. De izquierda a derecha: Vertical, horizontal a +90° y horizontal a -90°

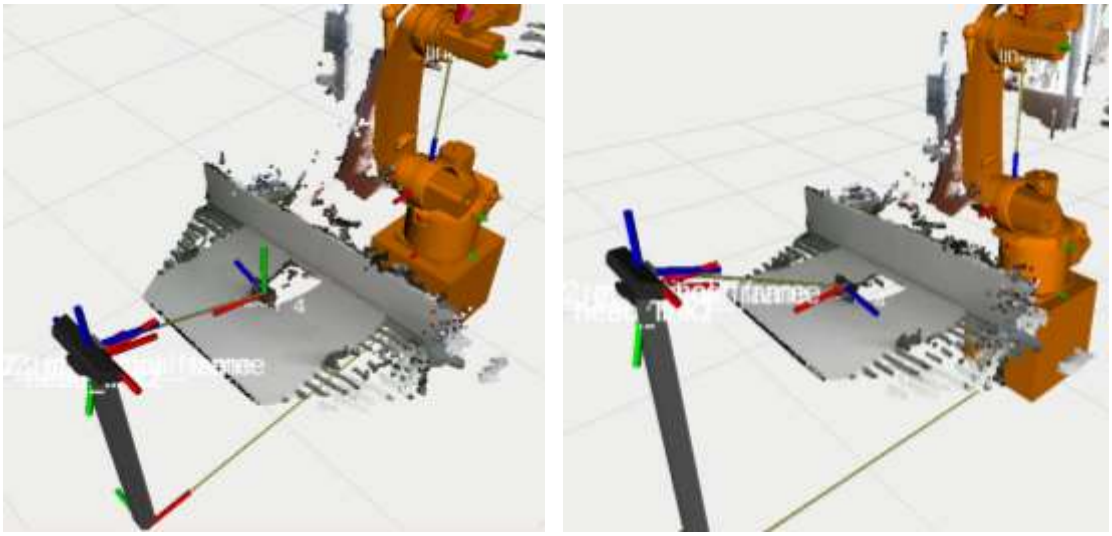

Figura 6.23. Posición y orientación del objeto (caja) horizontal en dos posiciones (BRISK), visto desde Rviz. Izquierda: +90°, Derecha: -90°

6.2 Simulación

La simulación del manipulador se realiza en *Gazebo*, utilizando la clase *MoveGroupInterface* dentro de la interfaz de C++ de *Moveit.* El código desarrollado se encarga de realizar la simulación de las trayectorias paso a paso, primero se observan los movimientos que realiza el robot en *Rviz* y después estos son ejecutados paralelamente en Gazebo (figura 6.24). Los puntos que conforman las trayectorias, los valores de los ángulos y los valores de esfuerzo de las juntas del robot son guardados en diferentes tópicos de *ROS* para ser usados posteriormente en análisis de posición, velocidad, aceleración y esfuerzo (figura 6.25), y/o para ser enviados al controlador CR4 del manipulador y que este realice dichos movimientos.

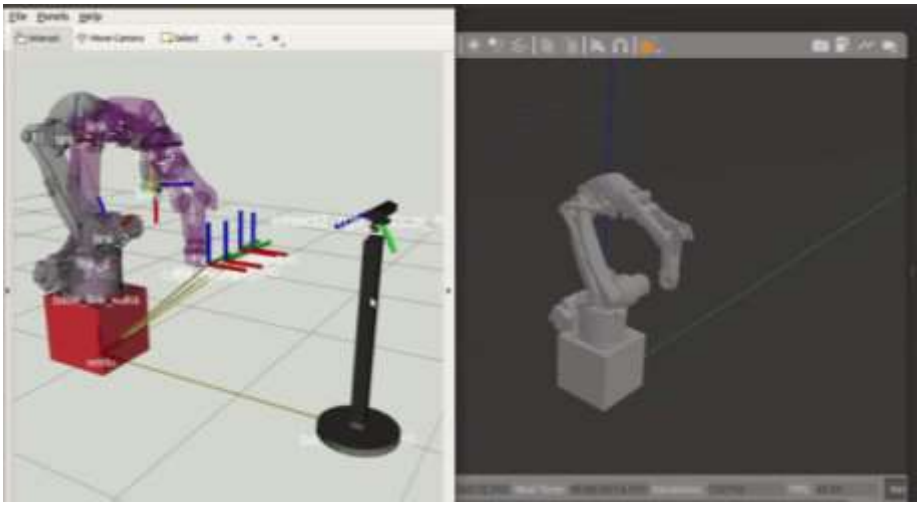

Figura 6.24. Captura de una simulación. A la izquierda se muestran los movimientos en Rviz y a la derecha se ejecutan en Gazebo

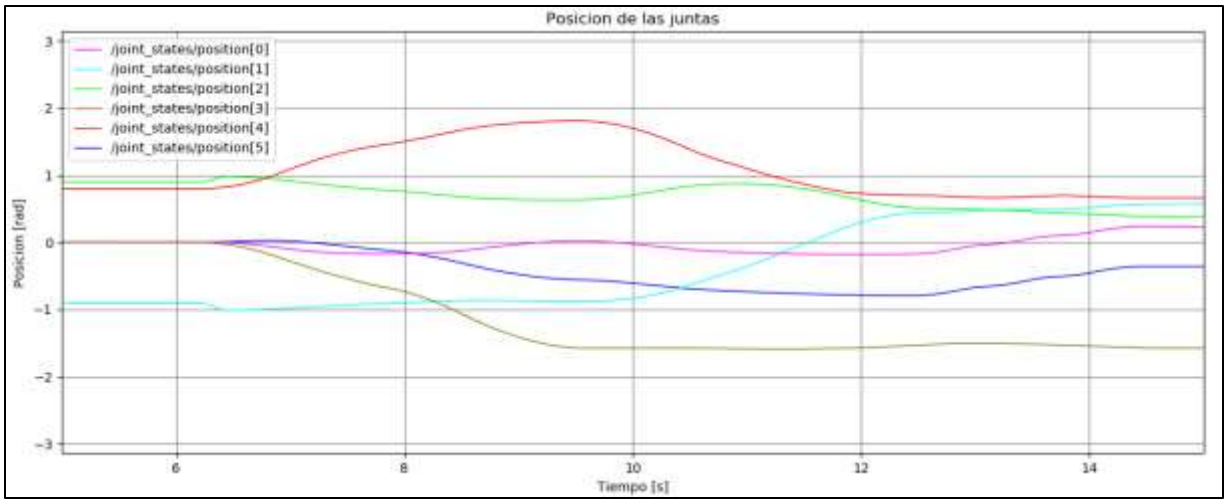

Figura 6.25. Ejemplo de grafica de posición de las juntas del robot simulado

Capítulo 7 Resultados y discusión

En esta sección se presentan y discuten los datos obtenidos en las diferentes pruebas realizadas.

7.1. Calibración del Kinect 2.0

Al finalizar la calibración, el paquete iai_kinect2 devolvió una base de datos con 702,985 observaciones, de la cual, se tomó una muestra representativa (9475 observaciones) seleccionada de forma aleatoria, con un nivel de confianza del 95% y un margen de error del 1%. Por otro lado, el error absoluto promedio entre la distancia medida por el Kinect y la distancia calculada por la librería después de la calibración fue de 1.067 [cm], el error relativo promedio fue de 1.46% y la desviación estándar de 1.129 [cm].

En la figura 7.1 se puede observar una gráfica con el comportamiento de las distancias, en la cual se aprecia que a partir de 0.8 [m] la distancia calculada es estable y existen algunas zonas (0.4 – 0.5 [m] y en 0.68 – 0.78 [m]) donde hay un mayor error. La primera zona de error se debe a que se encuentra en el inicio del rango operativo del Kinect 2.0 (0.5 [m] a 4.5 [m]) y los sensores de profundidad tienden a fallar con objetos que se hallan muy cerca de estos, mientras que la segunda zona es menos pronunciada y se asocia a un error en el proceso de calibración.

En la figura 7.2 se observan los puntos en X (horizontales) y en Y (verticales) respecto al Kinect, en este caso los puntos con menor valor en X representan los que se encuentran a la derecha, mientras que los que tienen menor valor en Y se encuentran arriba. Se puede observar que la diferencia entre la distancia calculada y la distancia medida es mayor cuando se toman medidas en la zona inferior del Kinect, y que esta diferencia puede llegar a ser de 7 [cm] en algunos casos. Se determinó que el error promedio en X y Y es de -1.074 [cm] y la desviación estándar es de 1.331 [cm].

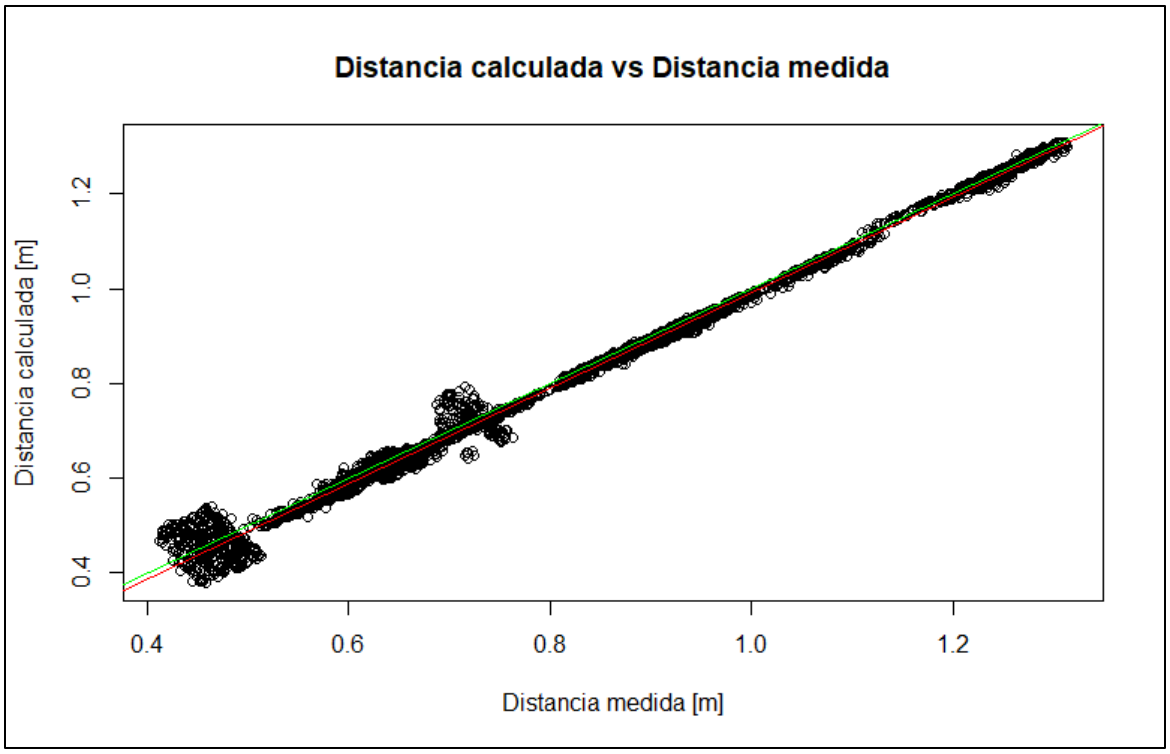

Figura 7.1. Distancia medida por el Kinect 2.0 vs Distancia calculada por la librería después de la calibración. En negro: Los datos de la muestra; En rojo: Regresión lineal de los datos; En verde: Comportamiento ideal

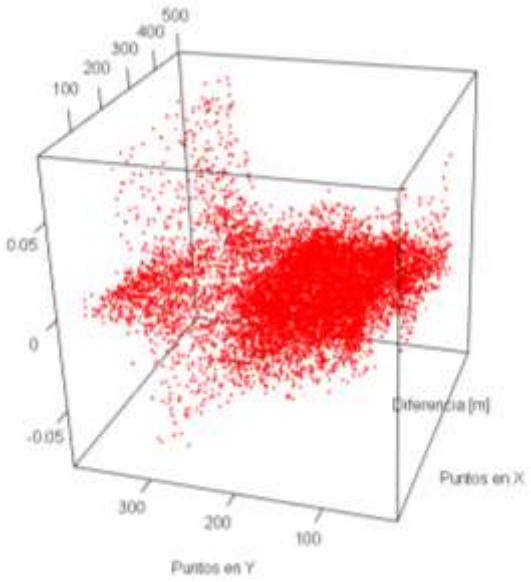

Figura 7.2. Diferencia entre la distancia medida y la calculada en los puntos en X y Y. Se observa un aumento de la diferencia en la zona inferior del Kinect (Y ~ 300).

7.2. Espacio de trabajo

Se observó que lo algoritmos son capaces de detectar objetos en todo el espacio horizontal, razón por la cual, el espacio de trabajo queda en términos de la profundidad. Estos resultados denotan que la profundidad afecta el reconocimiento de los objetos, lo que era de esperarse debido a que los puntos clave de los algoritmos varían con la escala de los objetos, que es dependiente de la profundidad a la que estos se encuentran. La tabla 7.1 muestra los rangos del espacio de trabajo del Kinect.

TABLA 7.1. RANGO DE DETECCIÓN PARA LOS ALGORITMOS FAST-ORB Y BRISK

Algoritmo	**Espacio de Trabajo [cm] respecto al kinect**
FAST-ORB	55.2 – 95.2
BRISK	55.2 – 77.2

7.3. Pruebas con objetos de frente a 0° y con rotaciones negativas y positivas

Los resultados obtenidos de posición y orientación fueron restados de los valores reales para obtener los errores absolutos. La media y la desviación estándar de estos datos fueron calculadas para determinar qué tan precisos y exactos fueron los algoritmos.

7.3.1. Objetos a 0°

En la tabla 7.2 se presentan los errores promedio en las coordenadas y se puede apreciar que la coordenada con mayor error es la Y (laterales del robot), con un error que varía entre 6 [cm] y 8 [cm], mientras que los ejes X y Z tienen un error que varía entre 3 [cm] y 5 [cm]. El error más grande en los ángulos se presentó en la esfera y el cubo, en los que fue difícil determinar la dirección específica de los ejes.

Se determinó que los tres algoritmos son precisos para calcular las coordenadas de los objetos ya que su desviación estándar está en el orden de los milímetros. Así mismo, los algoritmos también fueron capaces de calcular adecuadamente la orientación de los objetos, sin embargo, en algunos casos se observaron imprecisiones (caja con FAST-ORB y esfera con BRISK). Cabe resaltar que el algoritmo RANSAC-PCA fue el que presentó un menor en los cálculos de las coordenadas de los objetos debido a que el cálculo de la posición y orientación de los objetos fue la más precisa.

Como ejemplo se muestran las gráficas de la caja para observar el comportamiento de los datos (Figuras 7.3 - 7.8). Se aprecia que los algoritmos son precisos al calcular las coordenadas de los objetos, pero pueden ser inexactos (observar coordenada Y o laterales del robot). Por otro lado, es observable la imprecisión en el cálculo de la orientación que tiene el algoritmo FAST-ORB.

TABLA 7.2. ERRORES ABSOLUTOS PROMEDIO DE LOS OBJETOS A 0°.

Algoritmo	Objeto	X [cm]	Y [cm]	Z [cm]	Roll [°]	Pitch [°]	Yaw [°]
FAST-ORB	Esfera	-4.161	7.191	5.135	10.75	6.57	-28.74
	Lata	-3.422	7.334	4.281	3.61	15.57	-9.46
	Cubo	-4.201	7.560	3.272	-50.31	-51.81	24.93
	Caja	-4.381	7.830	4.246	7.32	19.89	-26.76
BRISK	Esfera	-5.071	6.012	5.222	3.77	25.19	-3.36
	Lata	-3.590	6.641	5.074	0.13	16.47	-2.51
	Cubo	-4.059	7.556	3.445	-7.14	27.96	18.79
	Caja	-4.647	7.596	4.264	-2.17	8.92	3.24
RANSAC-PCA	Esfera	-3.584	7.280	4.753	-141.57	73.16	146.87
	Lata	-3.118	6.803	6.060	-1.30	16.11	-2.01
	Cubo	-3.118	6.665	4.904	-60.49	70.60	-3.89
	Caja	-4.434	6.961	5.007	-2.60	3.05	-7.02

TABLA 7.3. DESVIACIONES ESTÁNDAR DE LOS OBJETOS A 0°

Algoritmo	Objeto	X [cm]	Y [cm]	Z [cm]	Roll [°]	Pitch [°]	Yaw [°]
FAST-ORB	Esfera	0.157	0.081	0.140	3.21	3.25	3.58
	Lata	0.106	0.056	0.085	1.18	6.19	3.04
	Cubo	0.166	0.055	0.071	2.93	2.94	8.96
	Caja	0.120	0.103	0.131	9.86	9.68	26.23
BRISK	Esfera	0.711	2.548	2.327	14.93	24.47	29.62
	Lata	0.186	0.910	0.445	6.26	18.12	16.39
	Cubo	0.122	0.218	0.204	7.32	9.83	17.61
	Caja	0.106	0.154	0.105	2.99	3.37	8.39
RANSAC-PCA	Esfera	0.075	0.021	0.032	9.69	1.56	9.97
	Lata	8.245	0.577	2.086	0.45	2.3	0.77
	Cubo	0.061	0.033	0.052	5.55	0.47	0.41
	Caja	0.048	0.035	0.043	0.43	0.28	1.12

Figura 7.3. Comparación del error absoluto en X para la caja a 0°.

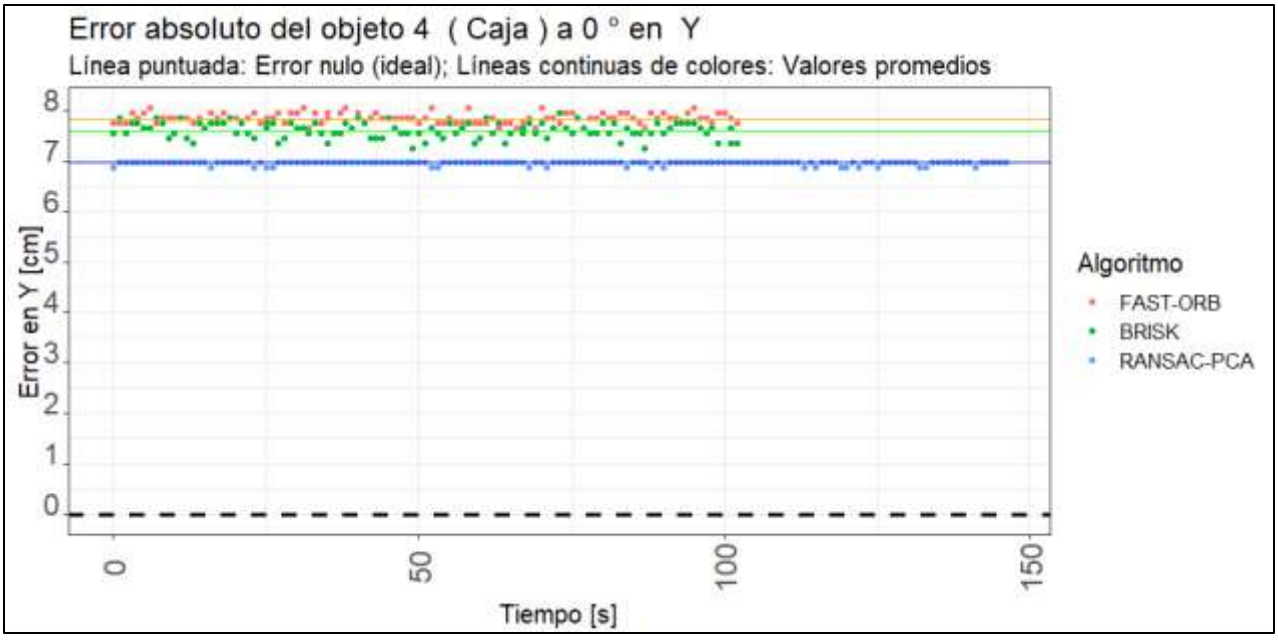

Figura 7.4. Comparación del error absoluto en Y para la caja a 0°.

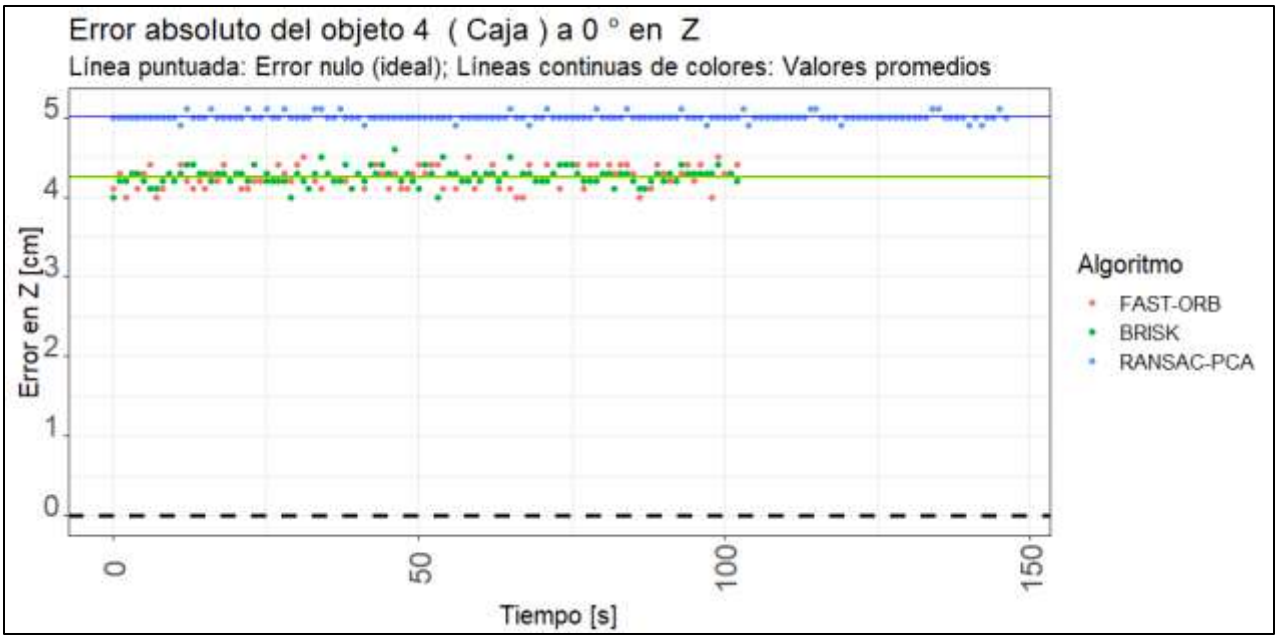

Figura 7.5. Comparación del error absoluto en Z para la caja a 0°.

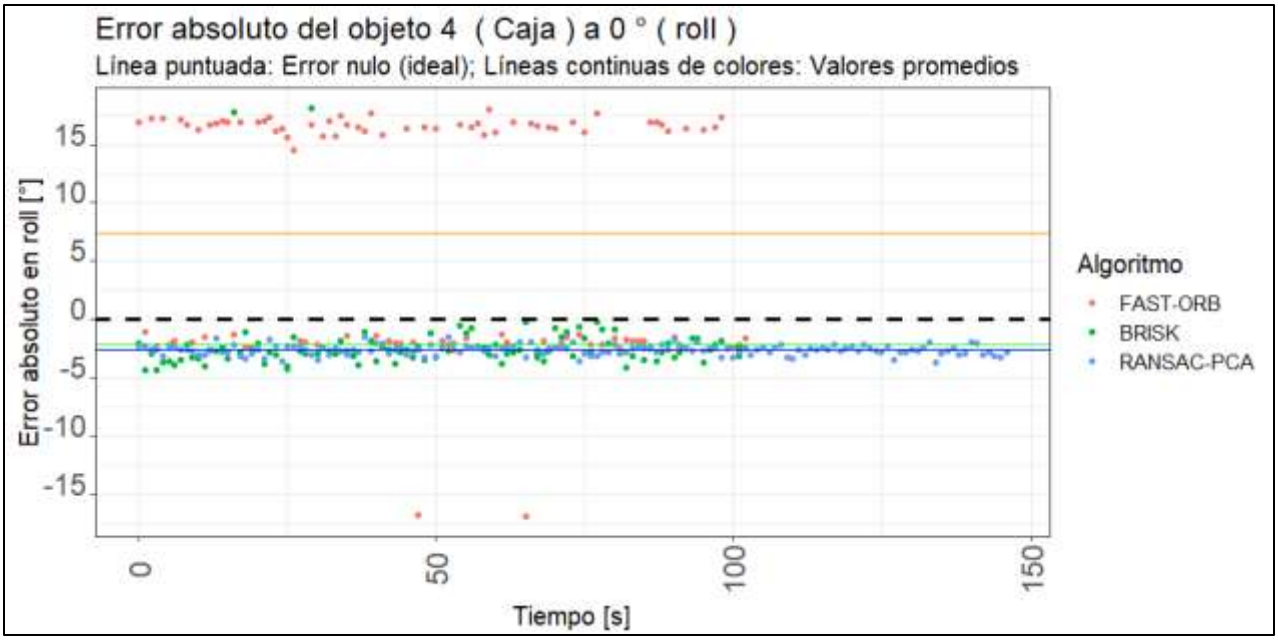

Figura 7.6. Comparación del error absoluto en roll (alabeo) para la caja a 0°.

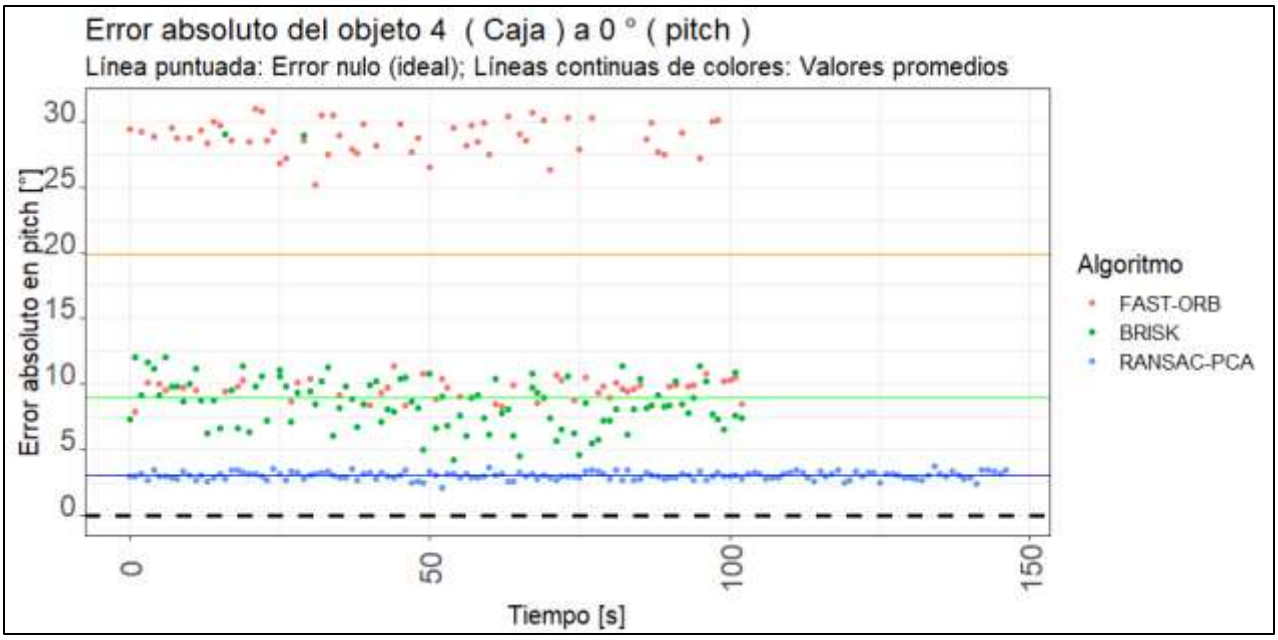

Figura 7.7. Comparación del error absoluto en pitch (cabeceo) para la caja a 0°.

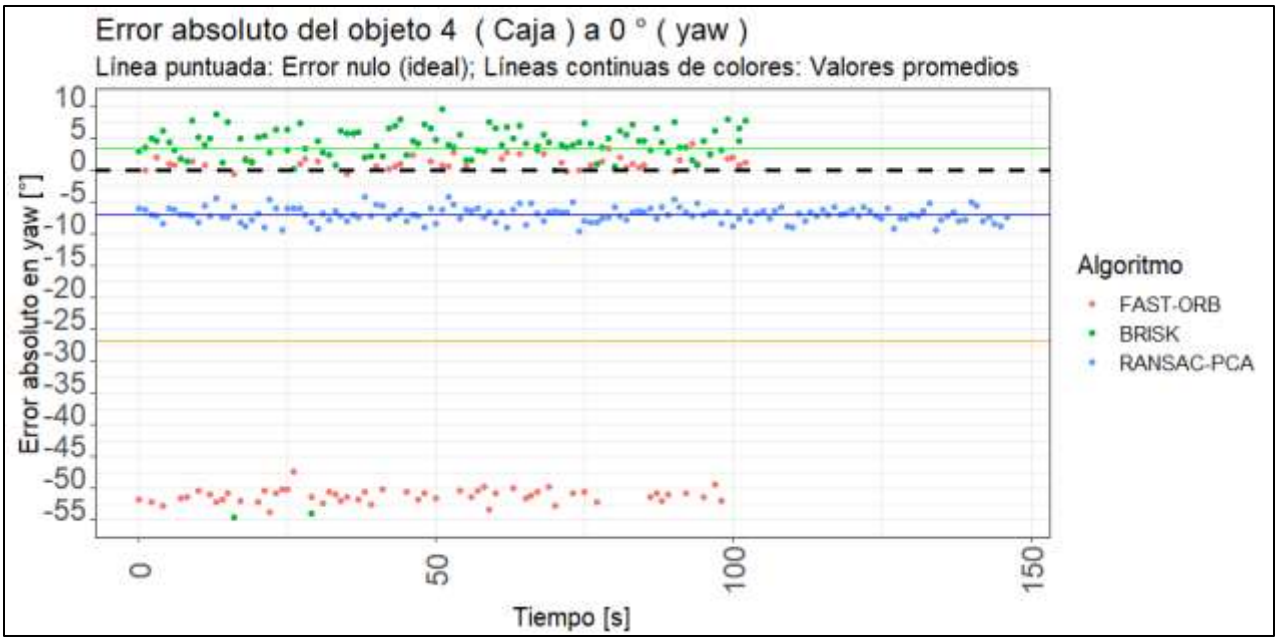

Figura 7.8. Comparación del error absoluto en yaw (giñada) para la caja a 0°.

7.3.2. Objetos con giro negativo

De manera general se observa un comportamiento parecido al de la prueba a 0°, aunque se nota una ligera reducción en el error en el eje X ya que disminuyen a entre 2 [cm] y 4.5 [cm], pero se observa un ligero aumento en los errores de los ejes Y y Z (Tabla 7.4).

En cuanto a la orientación se puede notar que se reduce considerablemente el error en roll, mientras que en pitch se mantiene casi igual y en yaw aumenta considerablemente, lo que indica que estos algoritmos pueden ser sensibles a este tipo de giro.

La desviación estándar (Tabla 7.5) de la posición se mantiene en el mismo orden, lo que indica que el giro de los objetos no afecta la precisión en el cálculo de las coordenadas. Sin embargo, en el caso de la orientación la desviación aumentó ligeramente y la precisión se redujo. Se destaca un aumento significativo de la imprecisión en yaw asociado al aumento significativo en el error. Por otro lado, e igual que en el caso anterior el algoritmo RANSAC-PCA destaca al ser el más preciso, aunque con el giro se disminuyó la exactitud especialmente en el cubo.

Como ejemplo en las figuras 7.9 - 7.14 se muestra el comportamiento de las pruebas con la caja girada en un ángulo negativo sobre el eje Z (yaw) y se aprecia un mayor error en el eje y, así como la poca precisión del algoritmo BRISK en el cálculo de los giros.

TABLA 7.4. ERRORES ABSOLUTOS PROMEDIO DE LOS OBJETOS GIRADOS EN UN ÁNGULO NEGATIVO SOBRE Z (YAW).

Algoritmo	Objeto	Giro [°]	X [cm]	Y [cm]	Z [cm]	Roll [°]	Pitch [°]	Yaw [°]
FAST-ORB	Esfera	-30	-4.53	7.40	4.87	12.64	24.79	14.16
	Lata	-30	-3.21	7.73	4.31	0.777	14.47	31.81
	Cubo	-30	-3.65	8.99	3.21	-4.818	7.95	66.48
	Caja	-20	-4.50	8.42	4.29	-2.11	6.61	37.22
BRISK	Lata	-15	-3.49	7.78	4.32	-2.92	17.62	32.10
	Cubo	-15	-4.20	8.46	3.50	1.87	27.74	22.69
	Caja	-10	-4.23	8.21	4.16	6.41	11.52	-5.03
RANSAC-PCA	Cubo	-30	-2.11	7.02	5.29	-4.98	-19.37	23.61
	Caja	-30	-2.19	6.78	5.87	1.78	42.03	0.92

TABLA 7.5. DESVIACIONES ESTÁNDAR DE LOS OBJETOS GIRADOS EN UN ÁNGULO NEGATIVO SOBRE Z (YAW).

Algoritmo	Objeto	Giro [°]	X [cm]	Y [cm]	Z [cm]	Roll [°]	Pitch [°]	Yaw [°]
FAST-ORB	Esfera	-30	0.18	0.20	0.31	10.94	15.67	28.55
	Lata	-30	0.12	0.06	0.09	3.31	6.54	1.076
	Cubo	-30	0.14	0.11	0.23	4.39	5.68	10.38
	Caja	-20	0.14	0.15	0.16	8.00	4.46	23.28
BRISK	Lata	-15	0.22	0.57	0.67	8.62	16.60	18.33
	Cubo	-15	0.20	0.47	0.43	8.04	14.67	17.15
	Caja	-10	0.46	0.97	1.40	14.84	10.88	44.52
RANSAC-PCA	Cubo	-30	0.07	0.05	0.05	1.09	0.47	0.79
	Caja	-30	0.05	0.02	0.07	2.84	2.77	1.51

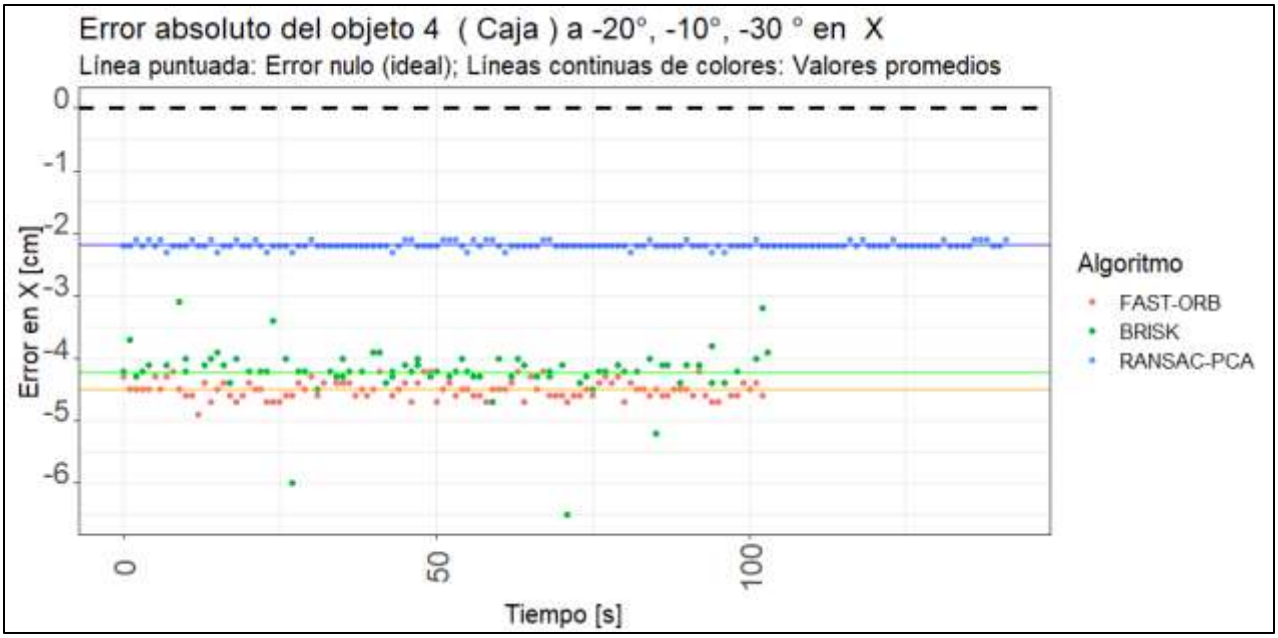

Figura 7.9. Comparación del error absoluto en X para la caja con giro negativo.

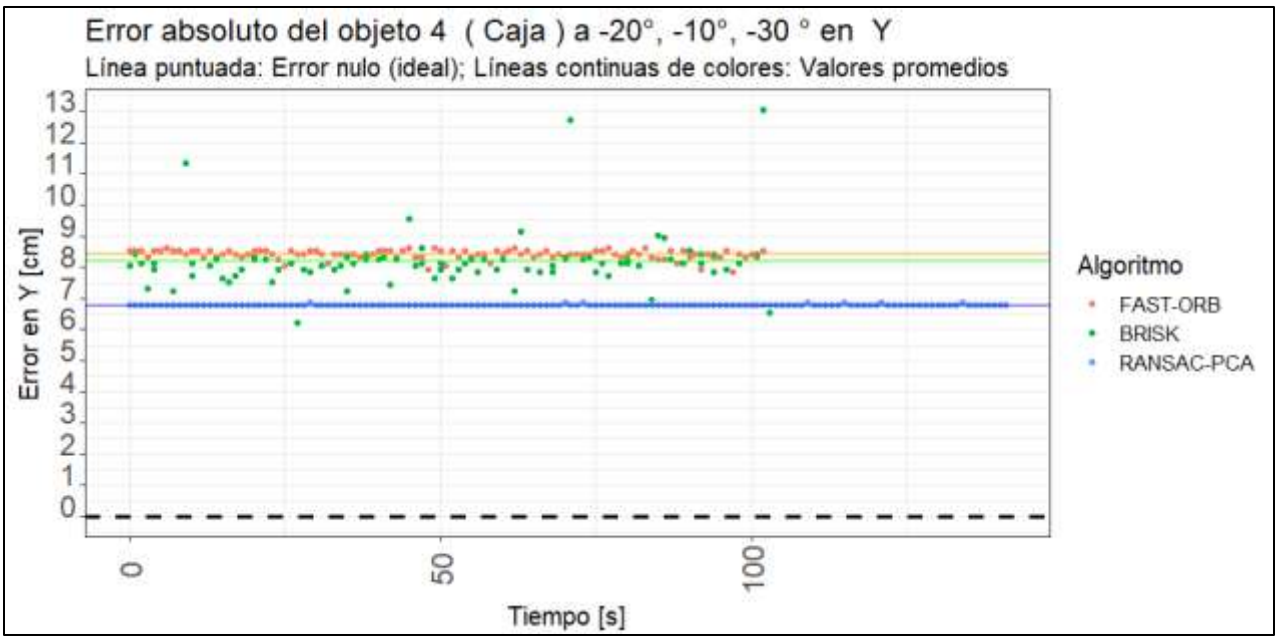

Figura 7.10. Comparación del error absoluto en Y para la caja con giro negativo.

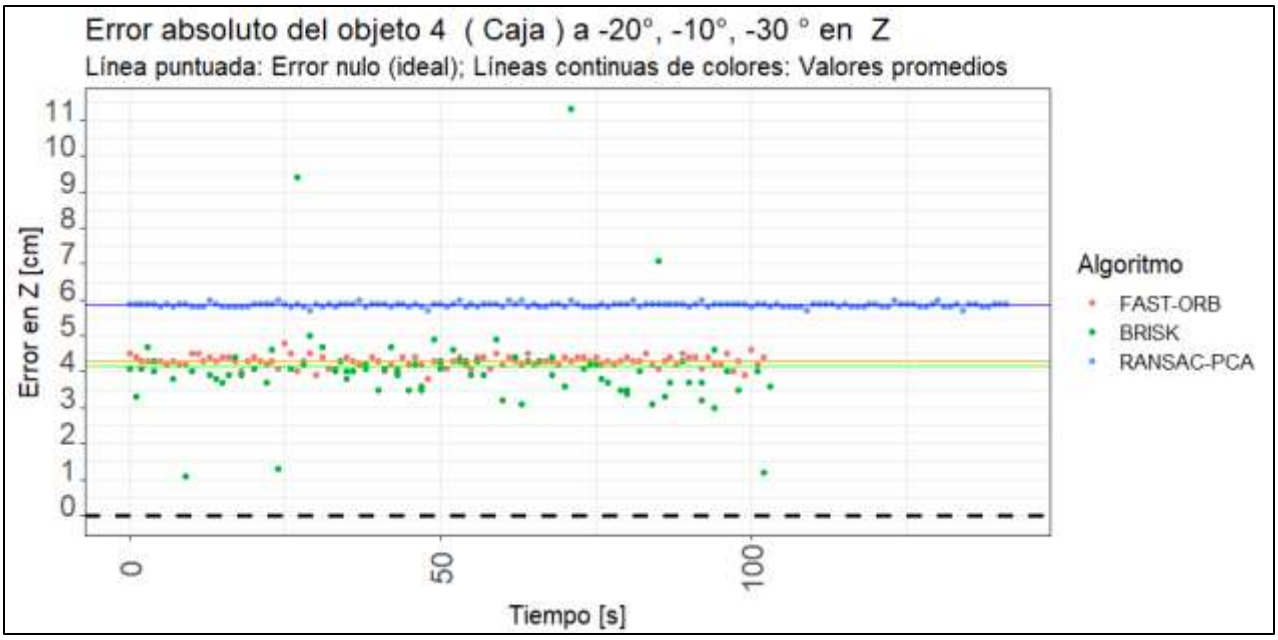

Figura 7.11. Comparación del error absoluto en Z para la caja con giro negativo.

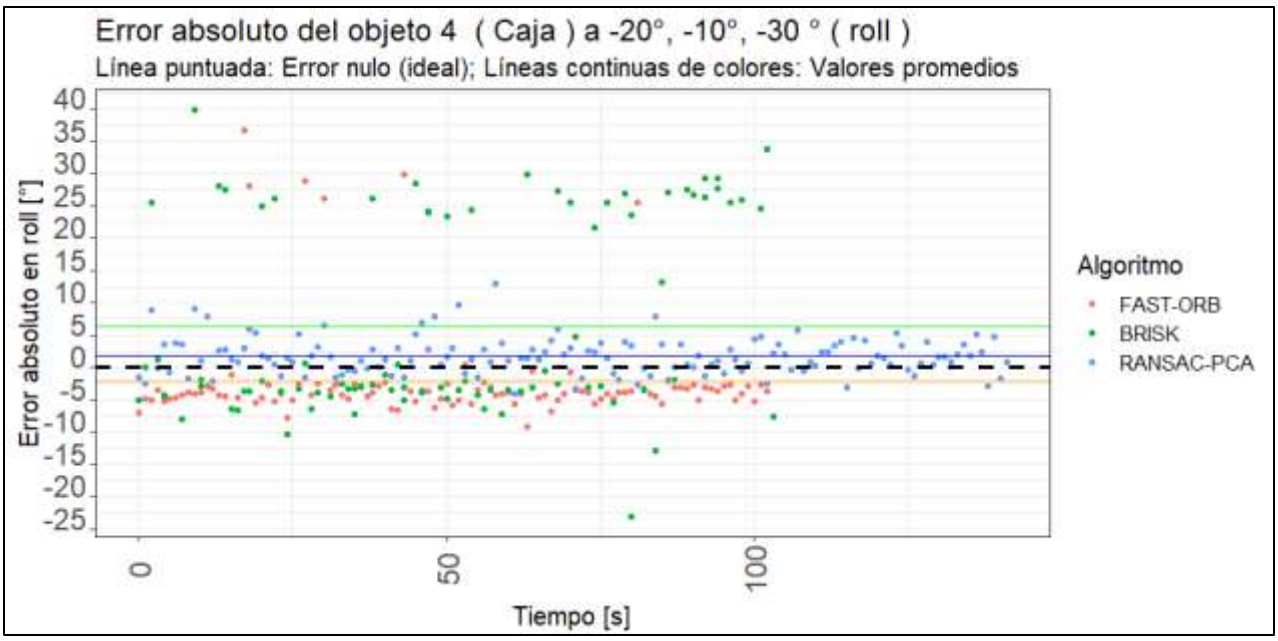

Figura 7.12. Comparación del error absoluto en roll (alabeo) para la caja con giro negativo.

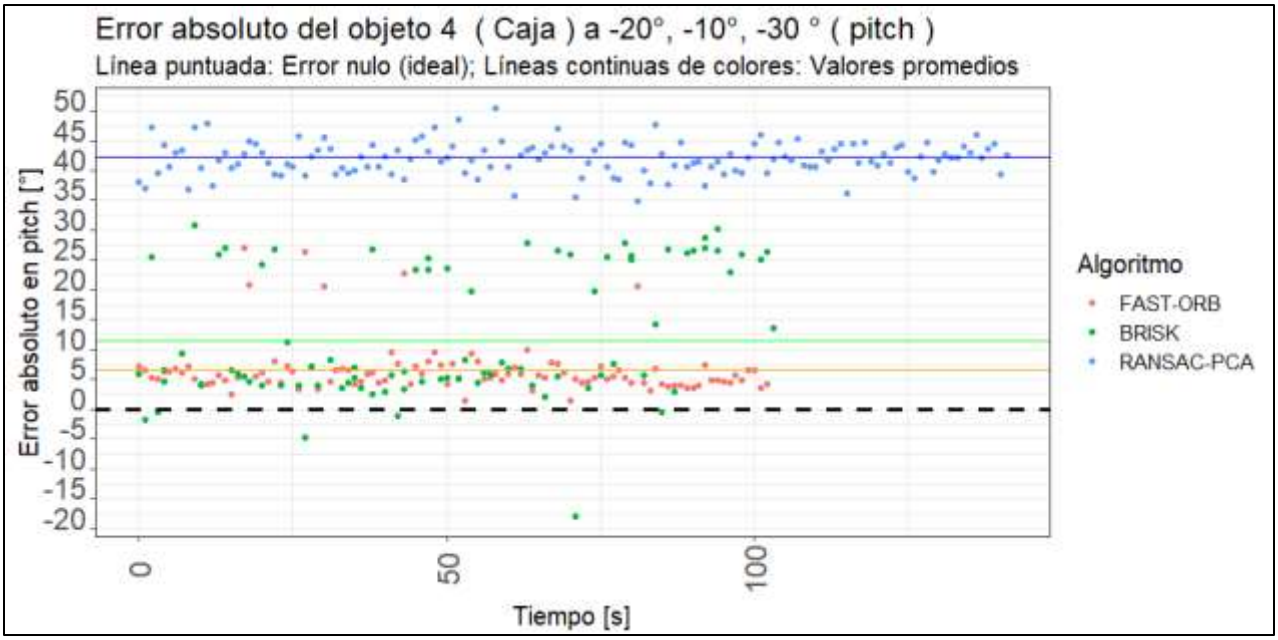

Figura 7.13. Comparación del error absoluto en pitch (cabeceo) para la caja con giro negativo.

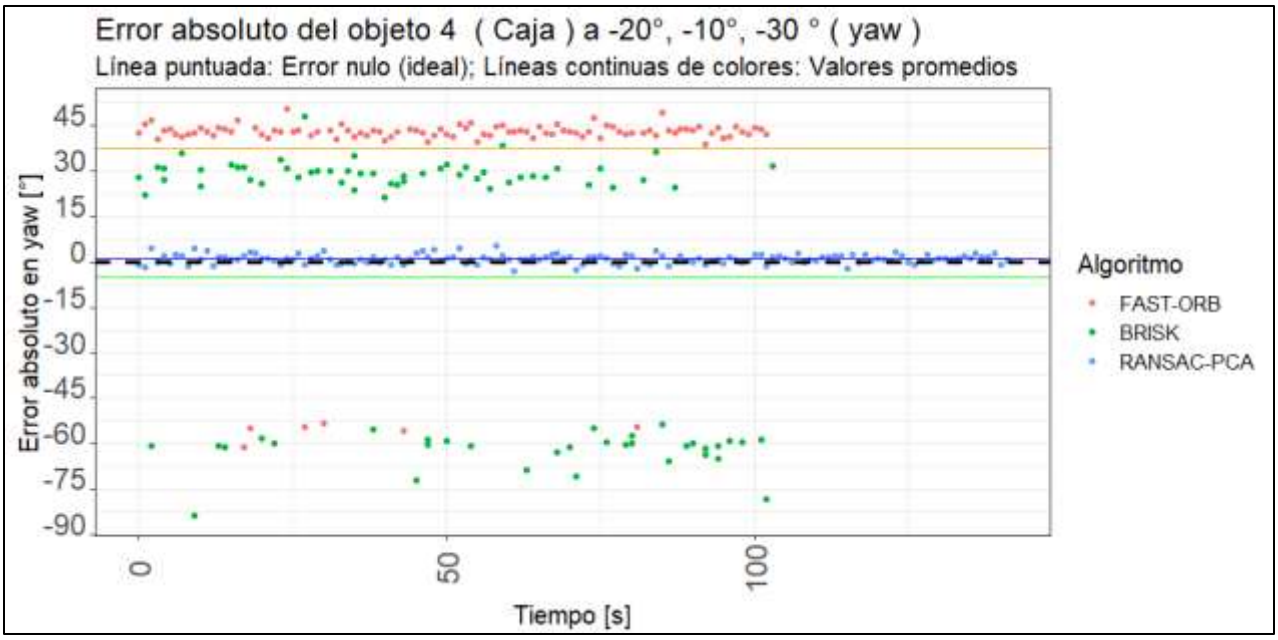

Figura 7.14. Comparación del error absoluto en yaw (guiñada) para la caja con giro negativo.

7.3.3. Objetos con un giro positivo

Se observa que el error y la desviación estándar tienen un comportamiento similar al de las pruebas anteriores, aunque hay una reducción en el error en Y (Tabla 7.6 y 7.7)

Se destaca la exactitud y precisión de los algoritmos al determinar el giro sobre el eje X (roll) pero al mismo tiempo existe un gran error en el cálculo del giro sobre Z (yaw) y fue más notorio con el algoritmo FAST-ORB y en menor medida con RANSAC-PCA.

TABLA 7.6. ERRORES ABSOLUTOS PROMEDIO DE LOS OBJETOS GIRADOS EN UN ÁNGULO POSITIVO SOBRE Z (YAW).

Algoritmo	Objeto	Giro [°]	X [cm]	Y [cm]	Z [cm]	Roll [°]	Pitch [°]	Yaw [°]
FAST-ORB	Esfera	+10	-4.71	6.93	4.78	-5.92	29.59	36.41
	Lata	+30	-3.46	6.96	4.21	4.37	18.14	20.26
	Cubo	+20	-3.99	6.53	3.26	-3.08	10.40	20.56
	Caja	+30	-4.09	7.36	4.33	3.12	9.46	-13.21
BRISK	Lata	+15	-3.56	6.39	3.89	0.867	21.00	-23.79
	Cubo	+15	-4.61	5.48	2.61	-15.37	25.90	11.45
	Caja	+15	-4.20	6.06	4.11	1.10	19.28	-30.23
RANSAC-PCA	Cubo	+30	-0.92	8.08	5.34	18.38	-23.76	-18.92
	Caja	+30	-2.32	6.99	5.72	-2.26	21.18	-2.82

TABLA 7.7. DESVIACIONES ESTÁNDAR DE LOS OBJETOS GIRADOS EN UN ÁNGULO POSITIVO SOBRE Z (YAW).

Algoritmo	Objeto	Giro [°]	X [cm]	Y [cm]	Z [cm]	Roll [°]	Pitch [°]	Yaw [°]
FAST-ORB	Esfera	+10	0.29	0.26	0.21	8.20	12.97	20.71
	Lata	+30	0.11	0.12	0.13	4.89	7.87	14.68
	Cubo	+20	0.12	0.13	0.22	12.16	9.87	30.52
	Caja	+30	0.14	0.30	0.26	2.88	8.54	10.03
BRISK	Lata	+15	0.17	0.26	0.34	6.42	3.56	17.59
	Cubo	+15	0.16	0.26	0.34	9.79	9.92	23.45
	Caja	+15	0.11	0.07	0.08	6.57	5.44	1.76
RANSAC-PCA	Cubo	+30	0.08	0.02	0.05	3.91	0.58	1.91
	Caja	+30	0.09	0.02	0.06	1.01	1.00	1.20

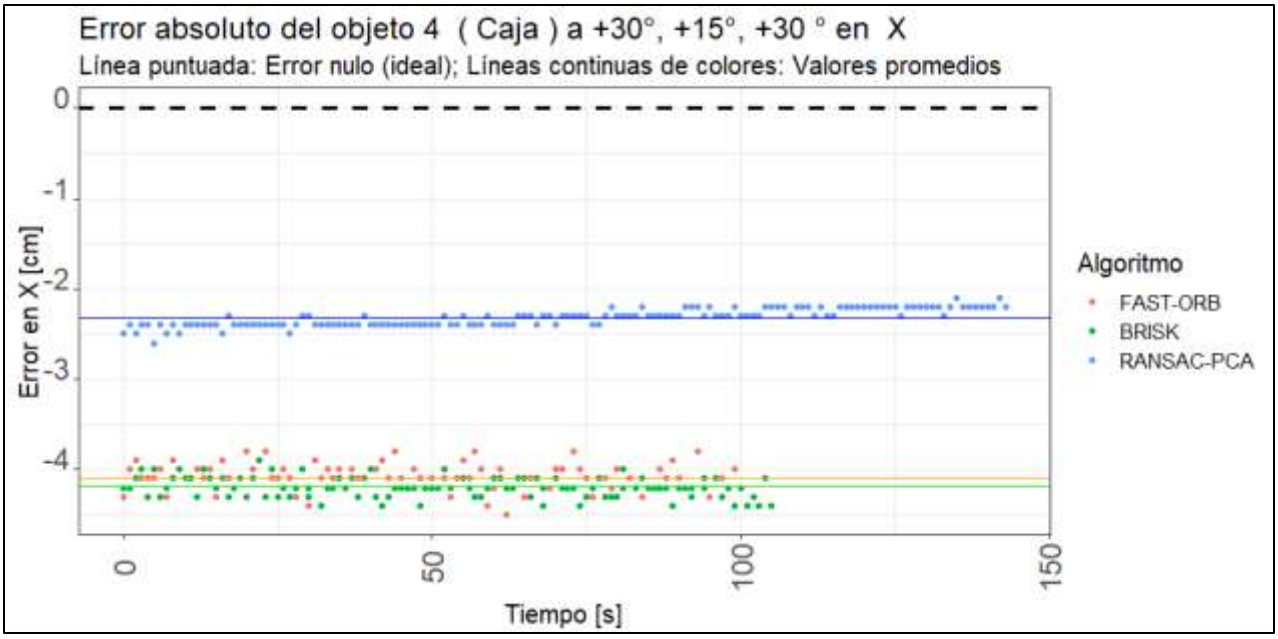

Figura 7.15. Comparación del error absoluto en X para la caja con giro positivo.

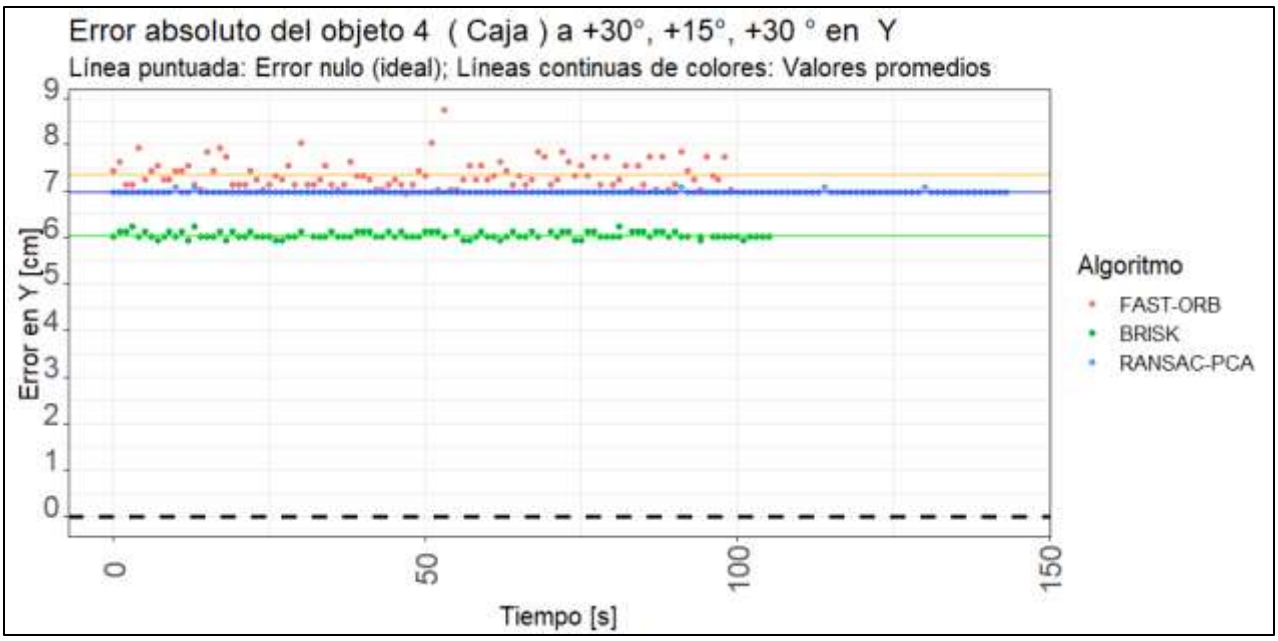

Figura 7.16. Comparación del error absoluto en Y para la caja con giro positivo

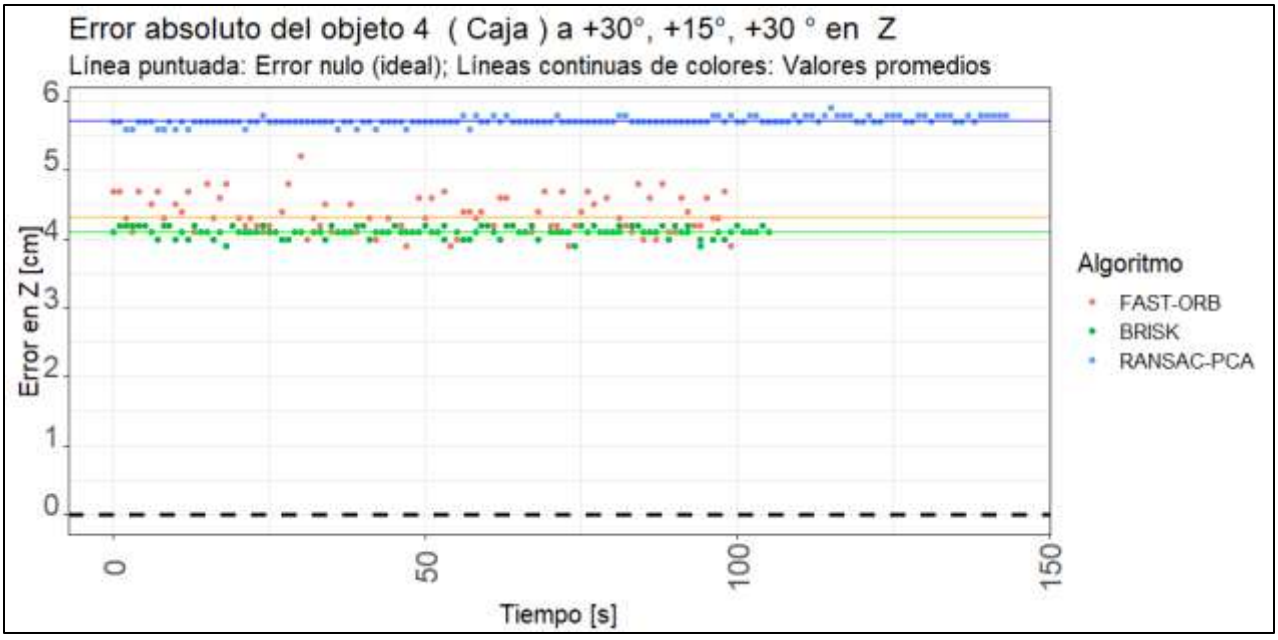

Figura 7.17. Comparación del error absoluto en Z para la caja con giro positivo

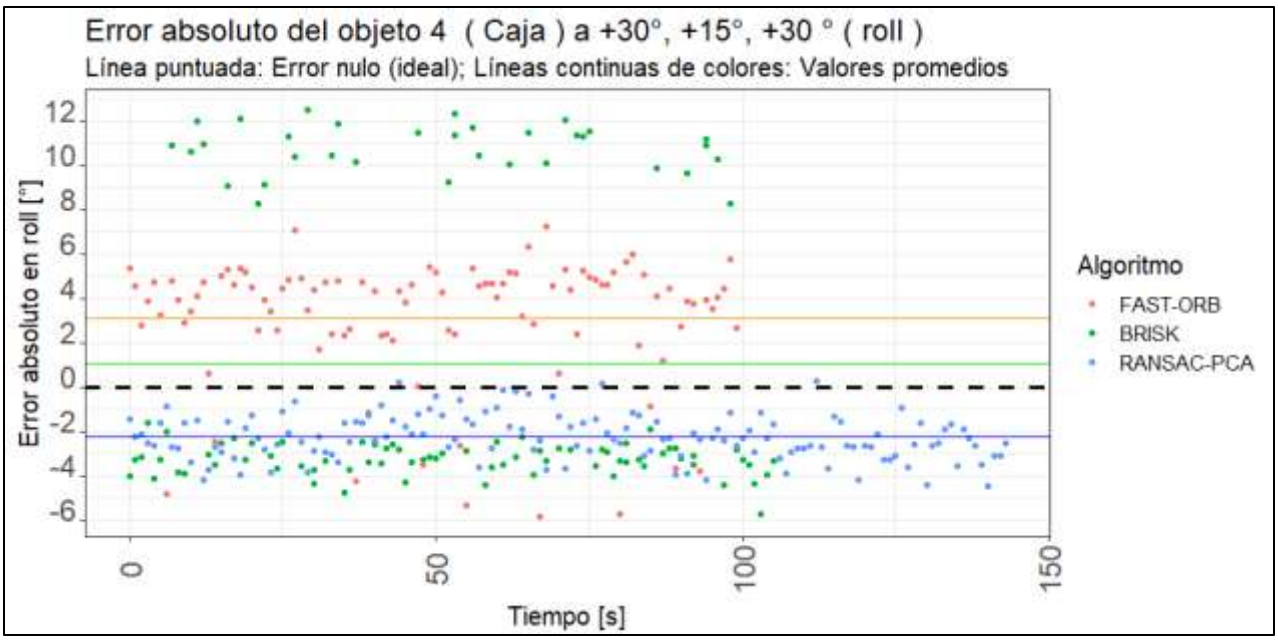

Figura 7.18. Comparación del error absoluto en roll (alabeo) para la caja con giro positivo

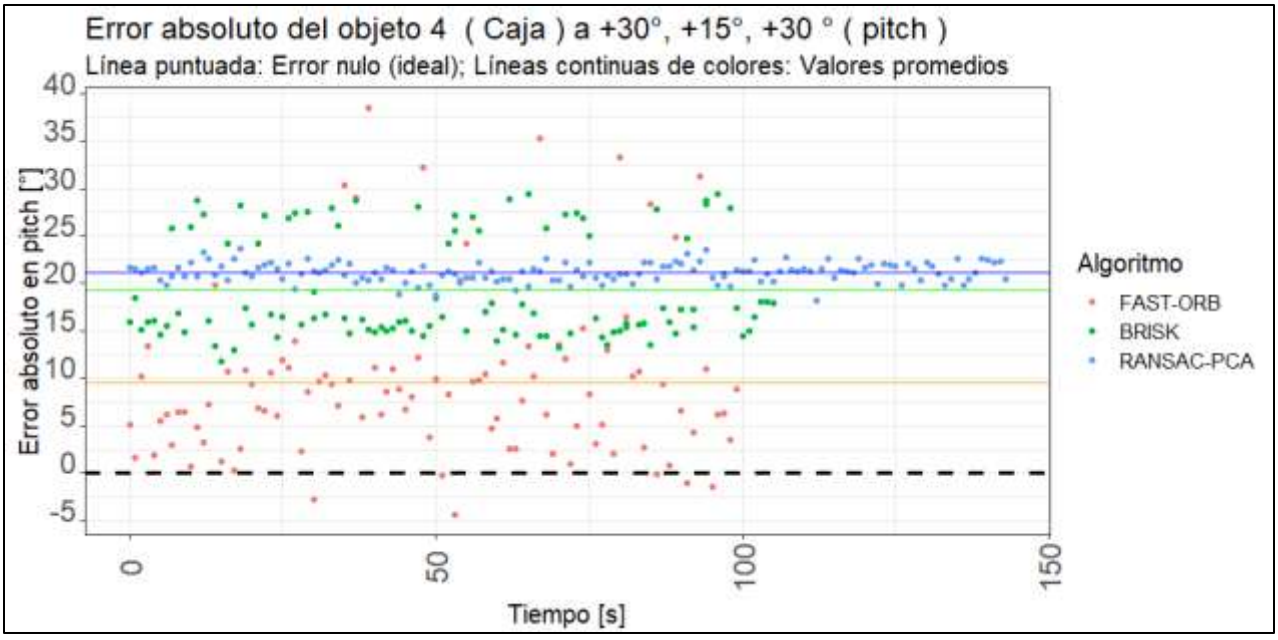

Figura 7.19. Comparación del error absoluto en pitch (cabeceo) para la caja con giro positivo

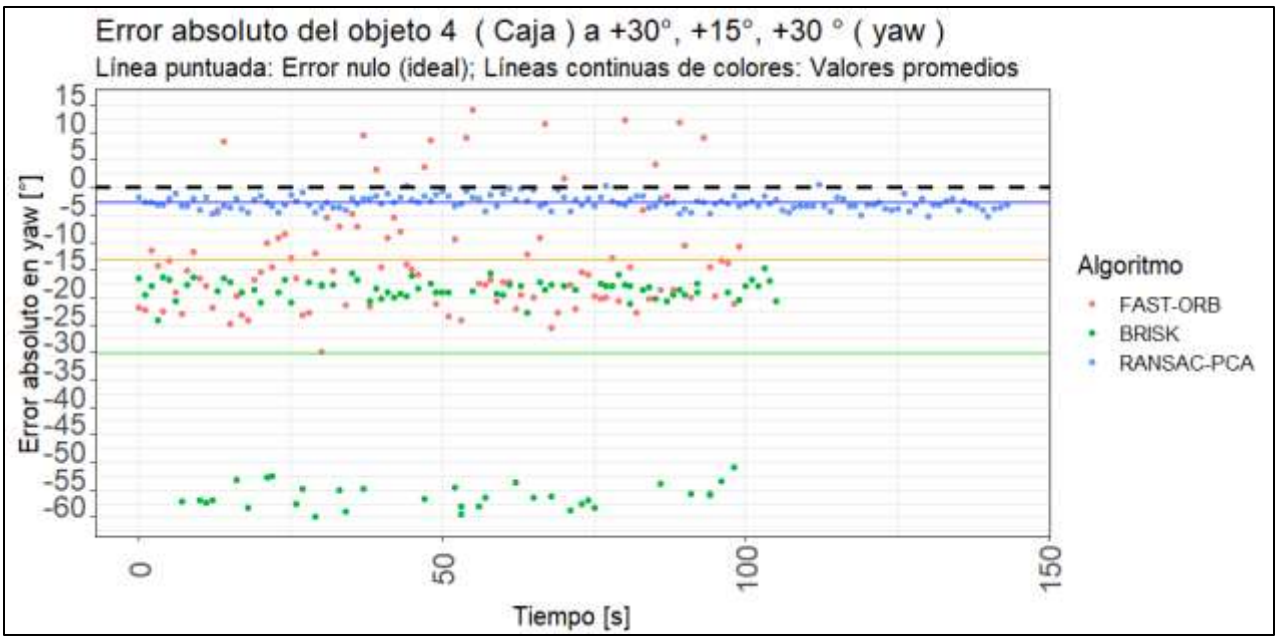

Figura 7.20. Comparación del error absoluto en yaw (guiñada) para la caja con giro positivo

7.3.4. Resultados por objeto

En la tabla 7.8 se encuentran los errores promedio en el reconocimiento de cada objeto en cada prueba, considerando los tres algoritmos. Se aprecia que en general, tanto en el eje X y el eje Z el error es bajo (-3.81 y 4.48 [cm]), mientras que el eje Y es el que más error presenta de forma general con un error promedio total de 7.22 [cm]. Estos errores se deben principalmente a la geometría de los objetos, a la calibración del Kinect y al posicionamiento del este en el trípode, el cual pudo haberse movido ligeramente.

A pesar de esto, el porcentaje de error general se mantuvo bajo con un mínimo de 4.1% y un máximo de 5.55%, lo cual demuestra que la detección y reconocimiento de los objetos es confiable y que es posible obtener mejores resultados si se posiciona, fija y calibra mejor el Kinect.

TABLA 7.8. RESULTADOS DE LAS PRUEBAS POR OBJETO.

Ángulo	Objeto	Eje X [cm]	Eje Y [cm]	Eje Z [cm]	Error [cm]	Error [%]
Frente	Esfera	-4.27	6.83	5.04	2.53	5.49
	Lata	-3.38	6.93	5.14	2.89	4.91
	Cubo	-3.79	7.26	3.87	2.44	4.51
	Caja	-4.49	7.46	4.51	2.49	5.21
Negativo	Esfera	-4.53	7.40	4.87	2.58	5.51
	Lata	-3.35	7.75	4.31	2.91	4.44
	Cubo	-3.32	8.15	4.00	2.94	4.35
	Caja	-3.64	7.80	4.77	2.98	4.87
Positivo	Esfera	-4.71	6.93	4.78	2.33	5.55
	Lata	-3.51	6.67	4.05	2.40	4.36
	Cubo	-3.17	6.69	4.05	2.42	4.10
	Caja	-3.53	6.80	4.72	2.66	4.85
Error total promedio [cm]		-3.81	7.22	4.48		

7.4. Pruebas con objetos sobrepuestos

Los resultados con las pruebas con objetos sobrepuestos se encuentran en las tablas 7.8 y 7.9 Se puede apreciar que el algoritmo FAST-ORB fue el mejor para detectar objetos ya que tiene una mayor cantidad de reconocimientos estables e inestables. También se observan variaciones entre los objetos ubicados en las posiciones “izquierda completo” y “derecha completo”, ya que estos últimos entre más cercanos a la derecha se encuentren tienden a ser mejor reconocidos que cuando se encuentran más cercanos a la izquierda. Esto es debido a que la cámara RGB del Kinect se encuentra ubicada a la derecha de su centro.

El reconocimiento de los objetos disminuyó considerablemente cuando fueron sobrepuestos. Específicamente, en las posiciones parciales y central hubo menos reconocimientos estables e inestables en ambos algoritmos (FAST-ORB y BRISK), mientras que en BRISK se observa un reconocimiento casi nulo en estas posiciones.

Como era de esperarse, cuando un objeto de menor tamaño se colocó detrás de uno más grande, no hubo reconocimiento. Esto debido a que los objetos más grandes ocultan las características de los objetos más pequeños, como en el caso de colocar el cubo detrás de otros objetos.

TABLA 7.9. RESULTADOS DE LAS PRUEBAS CON OBJETOS SOBREPUESTOS, USANDO EL ALGORITMO FAST-ORB. LOS RESULTADOS POSIBLES SON: SIN RECONOCIMIENTO (×), RECONOCIMIENTO INESTABLE (~) Y RECONOCIMIENTO ESTABLE (✓).

Objeto al frente	Objeto trasero	Izquierda completo	Izquierda parcial	Centro	Derecha parcial	Derecha completo
Lata	Cubo	×	×	×	×	✓
	Caja	✓	×	×	~	✓
Cubo	Lata	✓	✓	~	✓	✓
	Caja	✓	✓	✓	✓	✓
Caja	Lata	~	×	×	~	✓
	Cubo	×	×	×	×	✓

TABLA 7.10. RESULTADOS DE LAS PRUEBAS CON OBJETOS SOBREPUESTOS, USANDO EL ALGORITMO BRISK. LOS RESULTADOS POSIBLES SON: SIN RECONOCIMIENTO (×), RECONOCIMIENTO INESTABLE (~) Y RECONOCIMIENTO ESTABLE (✓).

Objeto al frente	Objeto trasero	Izquierda completo	Izquierda parcial	Centro	Derecha parcial	Derecha completo
Lata	Cubo	×	×	×	×	×
	Caja	×	×	×	×	✓
Cubo	Lata	~	×	×	×	~
	Caja	✓	~	~	~	✓
Caja	Lata	×	×	×	×	×
	Cubo	×	×	×	×	×

Las pruebas se realizaron con la caja detrás del cubo debido a que fue la combinación más consistente con ambos algoritmos (FAST-ORB y BRISK). Los errores de posición y orientación de la caja y sus desviaciones estándar se muestran en las tablas 7.10 y 7.11 y representados en las figuras 7.21 -7.26, en las que se puede observar que los errores de posición (X, Y, Z) se encuentran en un rango parecido a los de las pruebas de frente y con giros, aunque se nota un ligero aumento sobre el eje Y hasta llegar a ser de 9 [cm]. Por su parte, los errores en la orientación mejoran ya que se observa que en su mayoría están en un rango de [-1°,3°] para roll, [9°,17°] para pitch y [-14°,3.5°] en yaw.

En cuanto a la precisión, BRISK y FAST-ORB tienen valores bajos de desviación estándar en el cálculo de la posición, por lo que se les puede considerar precisos, mientras que en la orientación el algoritmo FAST-ORB fue superior sobre BRISK.

TABLA 7.11. ERRORES ABSOLUTOS PROMEDIO DE LA CAJA ATRÁS DEL CUBO, EN LAS POSICIONES PARCIAL IZQUIERDA, CENTRO Y PARCIAL DERECHA.

Algoritmo	Posición	X [cm]	Y [cm]	Z [cm]	Roll [°]	Pitch [°]	Yaw [°]
FAST-ORB	Parcial izquierda	-3.87	8.99	3.46	3.07	9.07	-14.79
	Centro	-4.41	8.90	3.46	1.24	10.17	-3.38
	Parcial derecha	-4.25	8.33	3.56	-0.12	17.39	-3.48
BRISK	Parcial izquierda	-3.70	8.03	3.21	-1.35	15.15	-1.11
	Centro	-3.90	8.36	3.73	1.02	13.34	-11.02
	Parcial derecha	-04.02	8.13	3.58	0.43	16.02	-10.77

TABLA 7.12. DESVIACIONES ESTÁNDAR DE LA CAJA ATRÁS DEL CUBO, EN LAS POSICIONES PARCIAL IZQUIERDA, CENTRO Y PARCIAL DERECHA.

Algoritmo	Posición	X [cm]	Y [cm]	Z [cm]	Roll [°]	Pitch [°]	Yaw [°]
FAST-ORB	Parcial izquierda	0.15	0.17	0.26	3.31	7.51	10.91
	Centro	0.19	0.34	0.54	4.51	11.82	14.87
	Parcial derecha	0.12	0.09	0.15	4.76	6.69	1.49
BRISK	Parcial izquierda	0.74	1.47	2.29	10.94	15.69	25.37
	Centro	0.57	0.58	1.83	12.84	12.13	26.80
	Parcial derecha	0.25	0.48	0.67	7.85	7.86	26.29

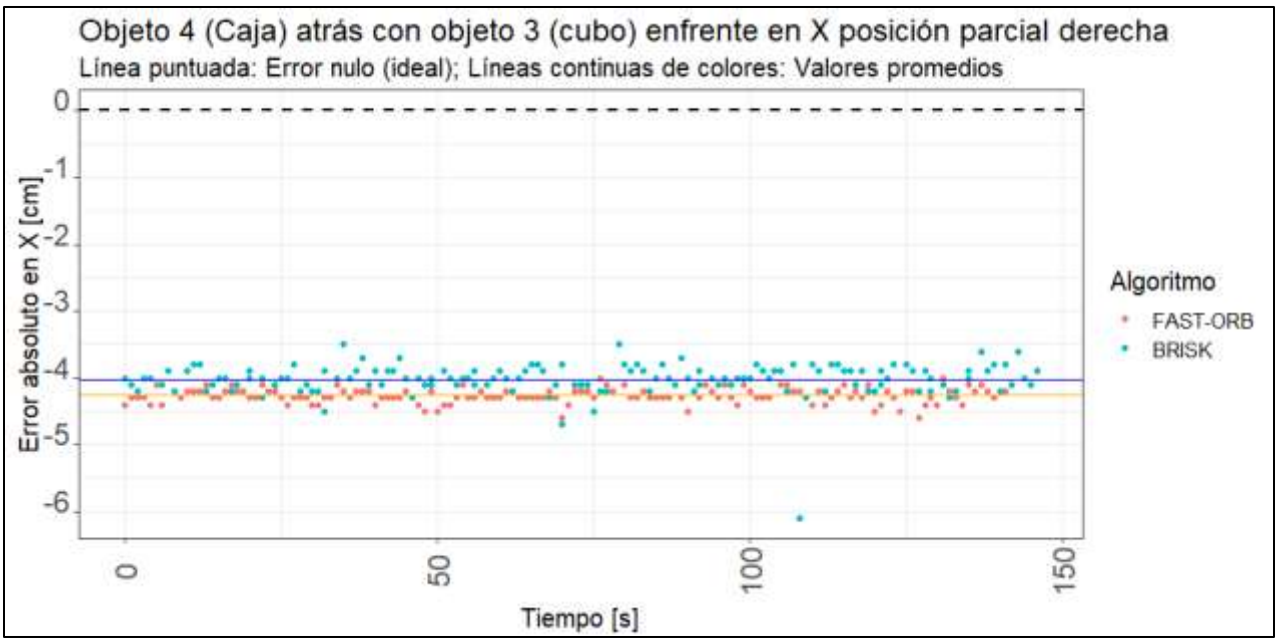

Figura 7.21. Comparación del error absoluto en X. Caja atrás de cubo. Posición parcial derecha.

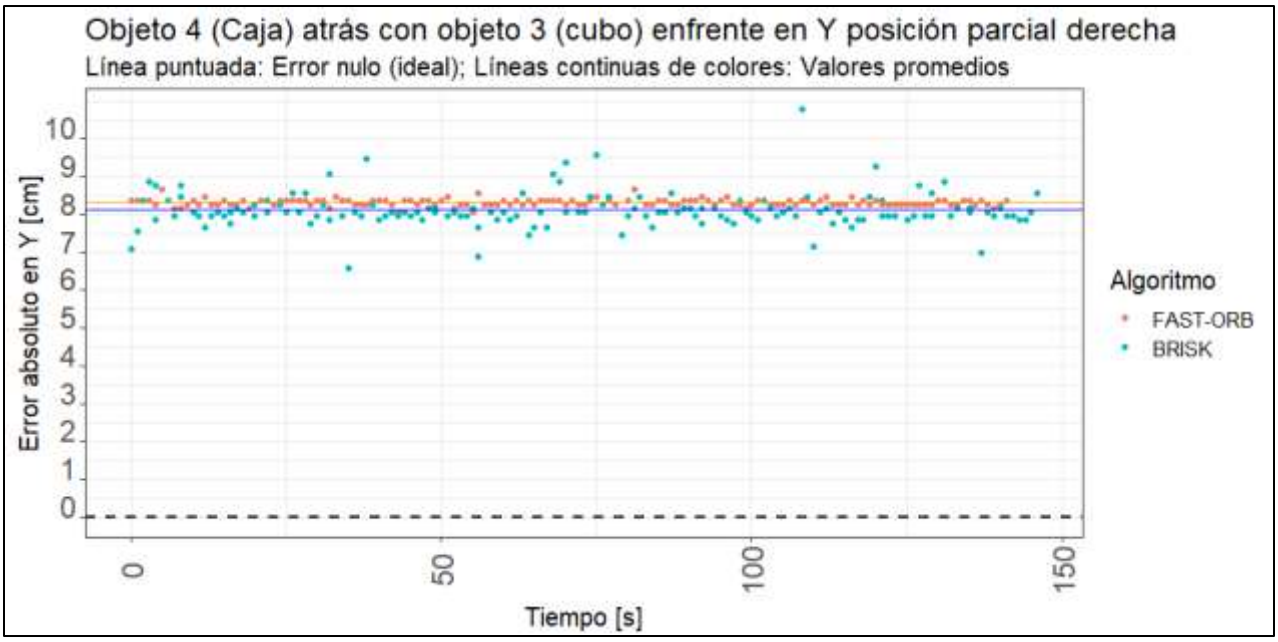

Figura 7.22. Comparación del error absoluto en Y. Caja atrás de cubo. Posición parcial derecha.

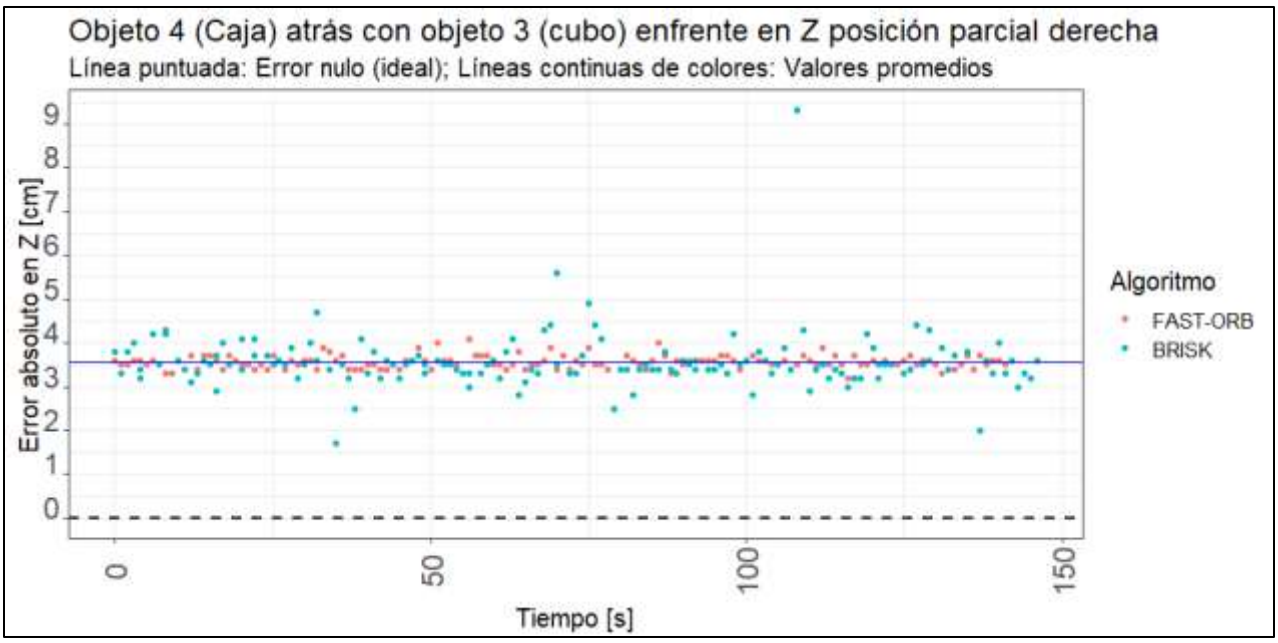

Figura 7.23. Comparación del error absoluto en Z. Caja atrás de cubo. Posición parcial derecha

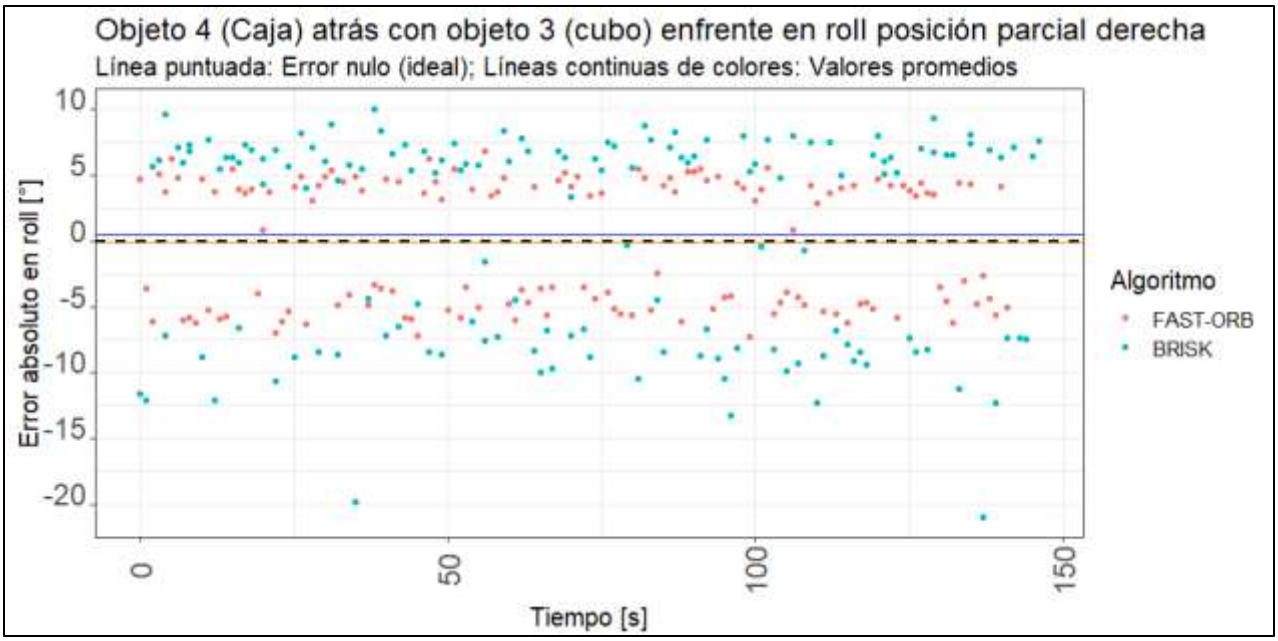

Figura 7.24. Comparación del error absoluto en roll. Caja atrás de cubo. Posición parcial derecha.

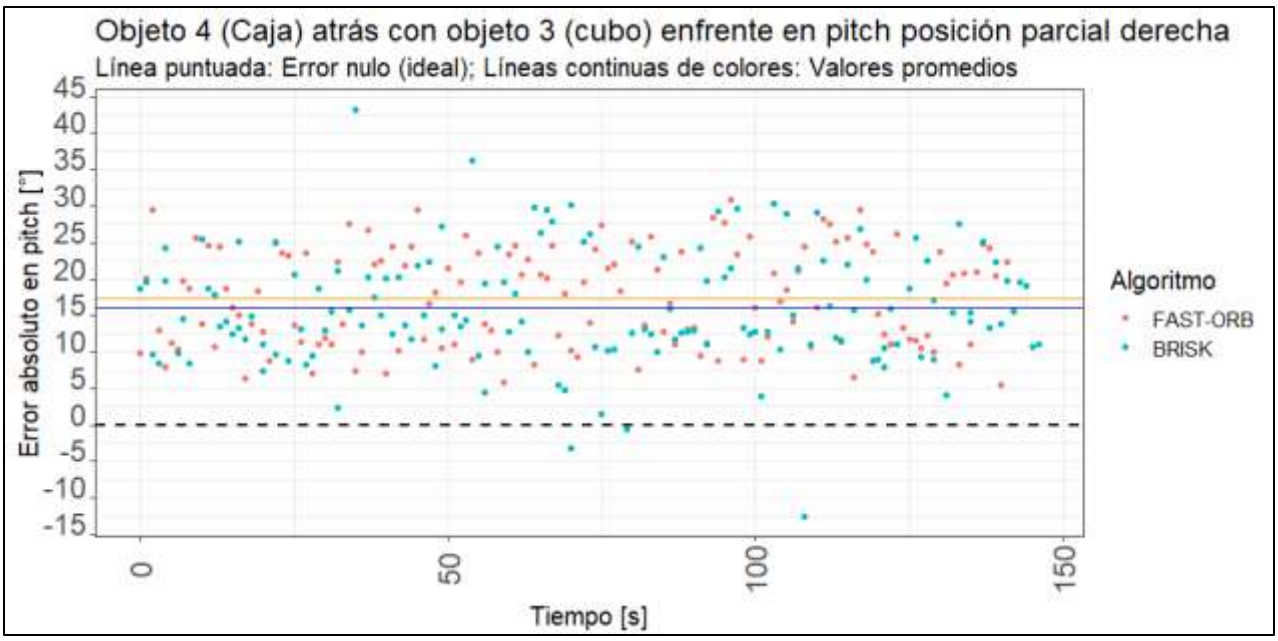

Figura 7.25. Comparación del error absoluto en pitch. Caja atrás de cubo. Posición parcial derecha.

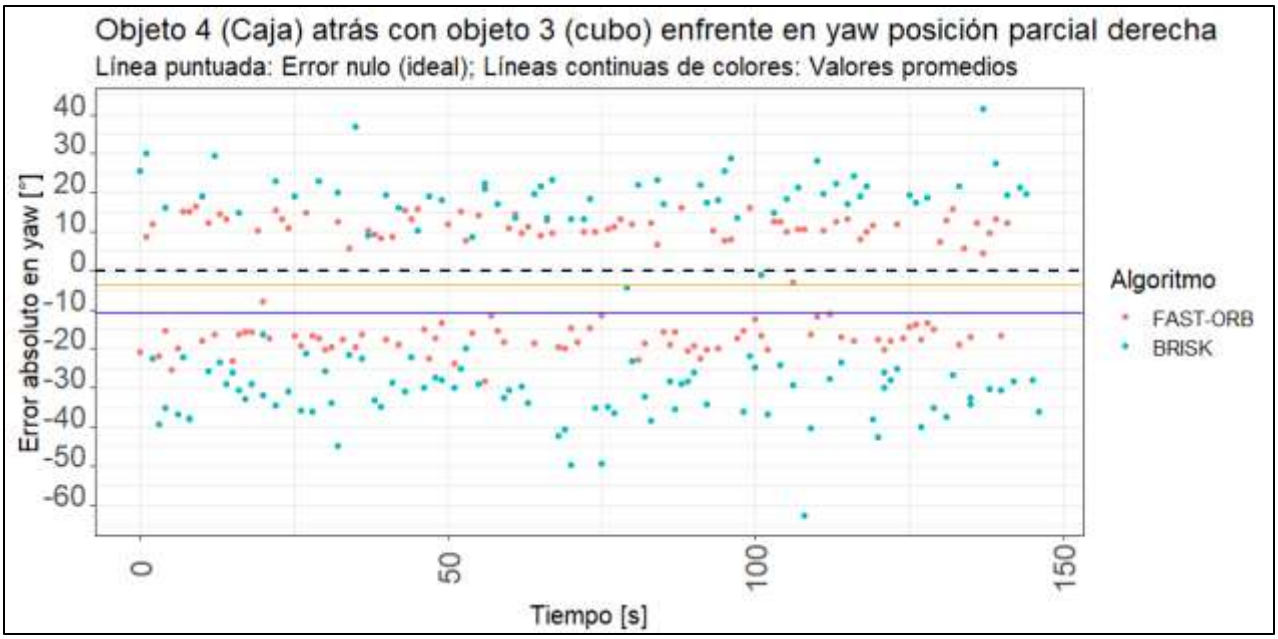

Figura 7.26. Comparación del error absoluto en yaw. Caja atrás de cubo. Posición parcial derecha.

7.5. Pruebas con objetos horizontales

Los resultados de las pruebas con la caja horizontal se encuentran en las tablas 7.12 y 7.13. El comportamiento del error es parecido a los casos anteriores, aunque el error del giro sobre el eje Y (*pitch*) es elevado. De manera general, la precisión del cálculo de las coordenadas se puede considerar buena, a excepción del eje Y con la caja horizontal.

Por su parte, la precisión del cálculo de la orientación fue buena cuando la caja estaba vertical, pero esta disminuyó cuando se colocó en posición horizontal y fue menor cuando el objeto se acostó +90°. Esto es a que las características principales de la caja quedaron de lado izquierdo al realizarse el giro, y existe la tendencia de que los algoritmos presentan más fallas entre más cercano a la izquierda se encuentre el objeto con respecto al Kinect.

Las gráficas 7.27 - 7.32 muestran el comportamiento de los datos con la caja posicionada de forma horizontal con un giro de -90° con el algoritmo BRISK. El comportamiento es parecido a las pruebas anteriores, es decir, un error de orden bajo en X y Z, un error pronunciado en Y, y una gran dispersión en los giros roll, pitch y yaw.

TABLA 7.13. ERRORES ABSOLUTOS PROMEDIO DE LA CAJA VERTICAL (ROLL = 0°) Y HORIZONTAL (ROLL = ±90°).

Algoritmo	Posición	X [cm]	Y [cm]	Z [cm]	Roll [°]	Pitch [°]	Yaw [°]
BRISK	Vertical (roll = 0°)	-4.37	8.01	3.80	-1.85	16.68	5.42
	Horizontal (roll = -90°)	-5.10	7.46	2.08	8.98	39.64	-13.03
	Horizontal (roll = +90°)	-4.97	8.53	1.71	9.06	21.78	-11.01

TABLA 7.14. DESVIACIONES ESTÁNDAR DE LA CAJA VERTICAL (ROLL = 0°) Y HORIZONTAL (ROLL = ±90°).

Algoritmo	Posición	X [cm]	Y [cm]	Z [cm]	Roll [°]	Pitch [°]	Yaw [°]
BRISK	Vertical (roll = 0°)	0.12	0.08	0.12	3.41	3.39	1.04
	Horizontal (roll = -90°)	0.17	1.07	0.37	7.79	29.52	13.89
	Horizontal (roll = +90°)	0.52	3.18	1.72	24.90	37.39	24.12

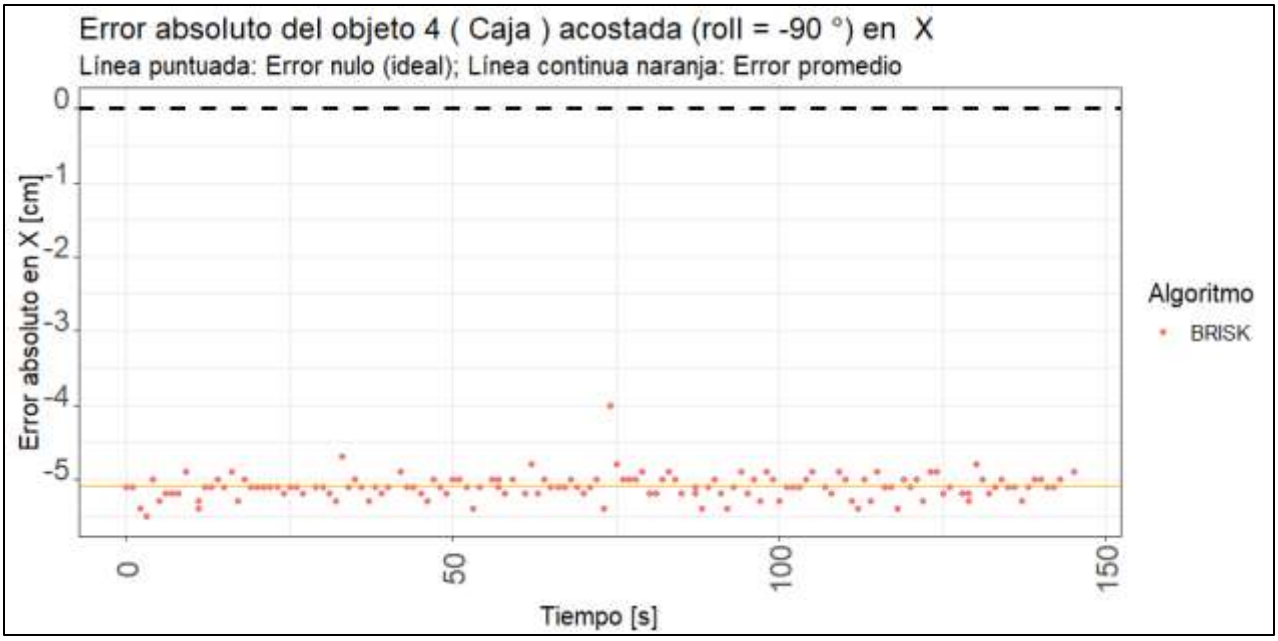

Figura 7.27. Comparación del error absoluto en X. Caja horizontal (roll = -90°).

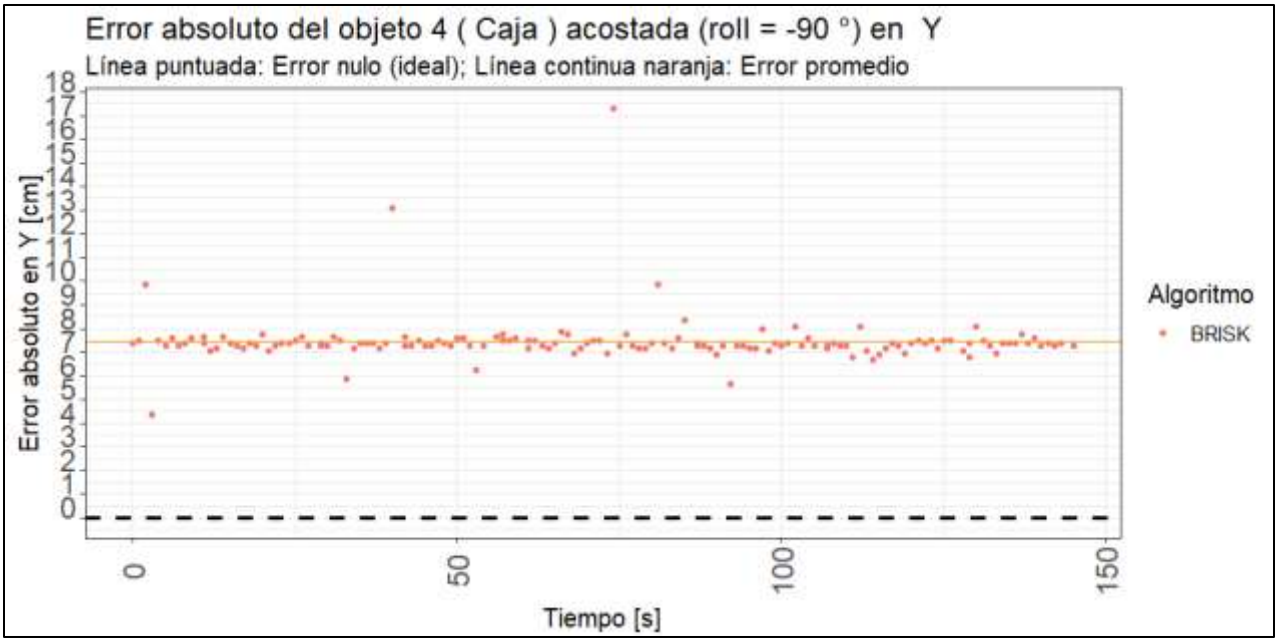

Figura 7.28. Comparación del error absoluto en Y. Caja horizontal (roll = -90°).

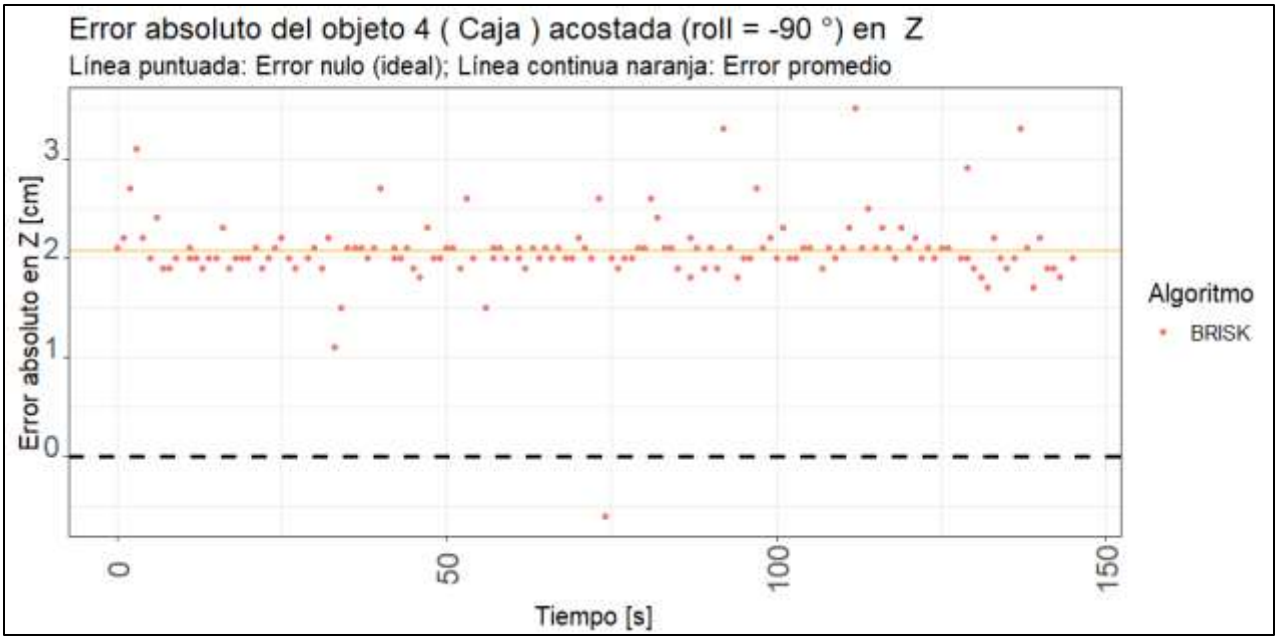

Figura 7.29. Comparación del error absoluto en Z. Caja horizontal (roll = -90°).

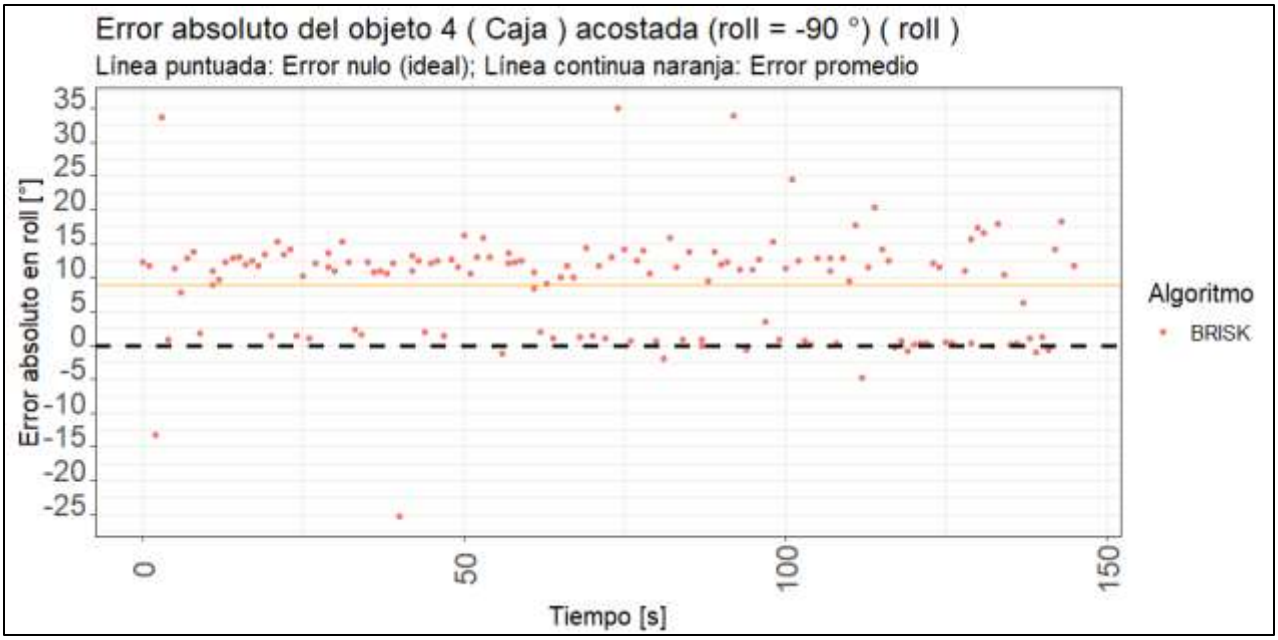

Figura 7.30. Comparación del error absoluto en roll. Caja horizontal (roll = -90°).

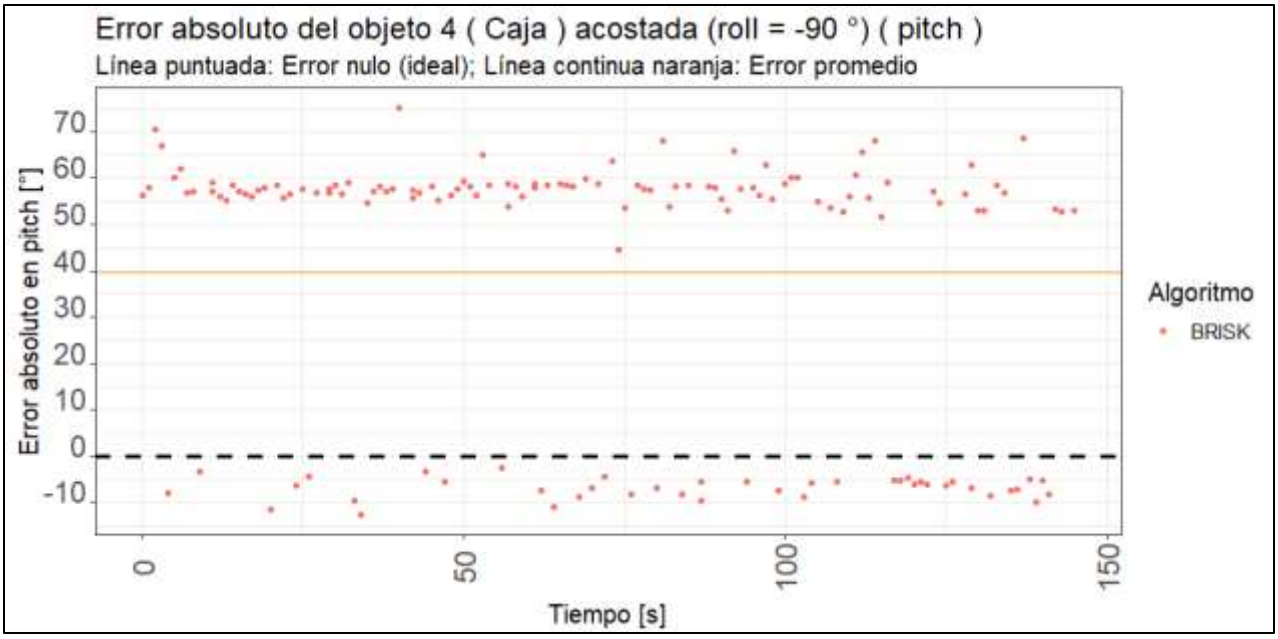

Figura 7.31. Comparación del error absoluto en pitch. Caja horizontal (roll = -90°).

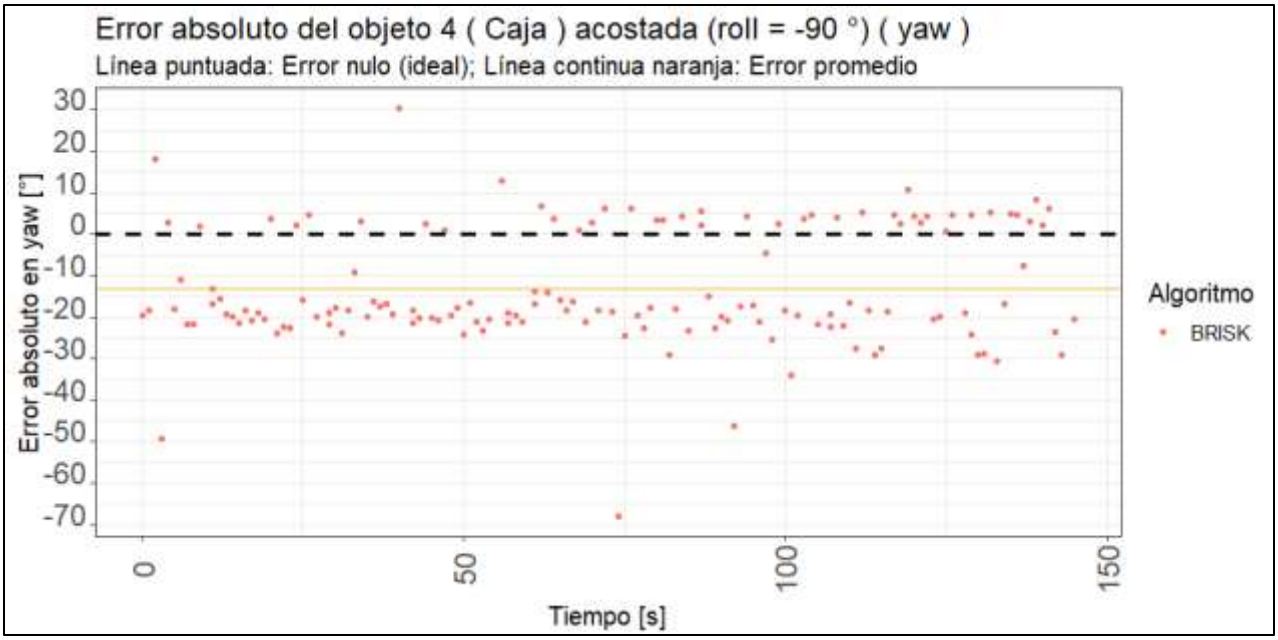

Figura 7.32. Comparación del error absoluto en yaw. Caja horizontal (roll = -90°).

Capítulo 8 Conclusiones y trabajo a futuro

En primera instancia, al obtener el promedio de todas las mediciones de pitch se obtiene un valor de 16.23°, el cual es parecido a la inclinación del Kinect (0.3 rad = 17.19°). Esto indica que los algoritmos toman como referencia la inclinación del Kinect. Para solucionar esto en pruebas futuras se deben ajustar los algoritmos para que el cálculo de la orientación en pitch tome en cuenta la inclinación del Kinect.

En general, se aprecia que hay una baja desviación estándar en los datos, lo que indica que los algoritmos son **precisos** para calcular la **posición** de los objetos, y que esta última no disminuye significativamente ante cambios de la orientación. Así mismo, se observó que la **exactitud** de los algoritmos para calcular el mismo parámetro fue buena para las coordenadas X y Z ya que el error fue bajo (3 - 4 [cm]) en comparación con el eje Y, ya que este presentó un error con un rango de 6 - 8 [cm] y que el gripper no es capaz de manejar.

En este caso, se descarta un error en los algoritmos debido a que los valores se mantuvieron constantes durante todas las pruebas, por lo que los errores se asocian a errores humanos en la estabilización del Kinect, por ejemplo, movimientos en el trípode que ocasionaron cambios en la orientación. Sin embargo, no se descartan errores en la calibración del Kinect o en la medición de las coordenadas reales. Para futuras pruebas, se recomienda utilizar una base más sólida o anclar el Kinect a una superficie completamente inmóvil como una pared.

Aunado a esto, se observó que la **exactitud** de los algoritmos para calcular la **orientación** de los objetos fue inestable en la mayoría de los casos, pero con algunas excepciones en las que sí lo fue, por ejemplo, al determinar los giros sobre el eje X (roll). Por otro lado, la **precisión** fue el apartado que más fallas presentó, particularmente en las pruebas donde hubo cambios en la orientación de los objetos. En este apartado se puede concluir que los algoritmos no son precisos al calcular la orientación y esta imprecisión se exacerba ante cambios de orientación.

Cabe resaltar que BRISK fue el algoritmo que presentó el menor desempeño, pero fue el único algoritmo que pudo reconocer objetos horizontales. Por otro lado, el algoritmo con mayor estabilidad y que presentó menos errores en el cálculo de la **posición** fue el RANSAC-PCA, sin embargo, este presentó mayor inestabilidad al calcular la orientación

de objetos completamente simétricos (esfera y cubo) y que está asociado a que la forma de estos no indica una dirección clara de los ejes. Por ello, se recomienda agregar al algoritmo un elemento de reconocimiento para obtener resultados estables en la orientación.

En el caso de FAST-ORB, su desempeño fue mejor en las pruebas con objetos sobrepuestos, aunque este disminuyó en pruebas con objetos de frente, además de que no fue capaz de reconocer objetos horizontales. En este sentido, el desempeño fue considerado ligeramente menor al de RANSAC-PCA. La tabla 8.1 resume las ventajas y desventajas de cada algoritmo.

TABLA 8.1. VENTAJAS Y DESVENTAJAS DE LOS ALGORITMOS ESTUDIADOS.

Algoritmo	Ventajas	Desventajas
FAST-ORB	• Reconocimiento de varios objetos a la vez. • Reconocimiento de objetos sobrepuestos.	• Incapaz de reconocer objetos horizontales.
BRISK	• Reconocimiento de varios objetos a la vez. • Reconocimiento de objetos horizontales.	• Menor precisión y exactitud • Problemas al reconocer objetos sobrepuestos.
RANSAC-PCA	• Mejor precisión y exactitud.	• Solo reconoce un objeto a la vez. • Inestable con objetos completamente simétricos.

Finalmente, se lograron realizar simulaciones y obtener gráficas de los movimientos del robot en condiciones físicas parecidas a las reales a partir del uso de *Gazebo* y la librería *Moveit* de ROS, ambos softwares libres y de código abierto. Posterior y para complementar el presente trabajo, se sugiere la realización de pruebas que involucren el movimiento del robot vía remota desde una computadora, en tiempo real y con trayectorias generadas con *Moveit*, para así comparar los resultados reales con los calculados.

Referencias

[1] Bundenministerium für Bildung und Forschung, ”Industrie 4.0”. 2011. [Online]. Disponible en: https://www.bmbf.de/de/zukunftsprojekt-industrie-4-0-848.html. [Último acceso: Abril 2020].

[2] *Información estadística de invenciones, signos distintivos y protección a la propiedad intelectual: Patentes por Nacionalidad del Titular*. Instituto Mexicano de la Propiedad Intelectual, enero 2020. Disponible en: https://datos.gob.mx/busca/dataset/informacion-estadistica-de-invenciones-signos-distintivos-y-proteccion-a-la-propiedad-intelectu/resource/91d10508-2162-4c59-8bf4-e9f514826e08

[3] D. Henke Dos Reis, D. Welfer, M. A. De Souza Leite Cuadros and D. F. Tello Gamarra, “Mobile Robot Navigation Using an Object Recognition Software with RGBD Images and the YOLO Algorithm,” in *Applied Artificial Intelligence,* Vol. 33, pp. 1-16. 2019. DOI: 10.1080/08839514.2019.1684778

[4] J. Redmon, A. Farhadi, S. Divvala, R. Girshick, “You Only Look Once: Unified, Real-Time Object Detection” in *Conference on Computer Vision and Pattern Recognition*, 2016, pp. 779-788. 10.1109/CVPR.2016.91.

[5] J. Y. Lee and C. Lee, “Path planning for SCARA robot based on marker detection using feature extraction and, labelling,” *International Journal of Computer Integrated Manufacturing*, Vol. 31, No. 8, pp. 769-776, South Korea, 2018. doi: 10.1080/0951192X.2018.1429669

[6] D. G. Lowe, "Object recognition from local scale-invariant features," *Proceedings of the Seventh IEEE International Conference on Computer Vision*, Kerkyra, Greece, 1999, pp. 1150-1157 vol.2, doi: 10.1109/ICCV.1999.790410.

[7] P. Andhare and S. Rawat, "Pick and place industrial robot controller with computer vision," *2016 International Conference on Computing Communication Control and automation (ICCUBEA)*, Pune, 2016, pp. 1-4, doi: 10.1109/ICCUBEA.2016.7860048.

[8] Esa Apriaskar et al (2020) “Robotic technology towards industry 4.0: Automatic object sorting robot arm using kinect sensor”, *Journal of Physics*: Conf. Ser.1444 012030

[9] O. Glaufe, O. Gladstone, E. Ciro, C. C. A. T. and L. José, "Development of Robotic Arm Control System Using Computational Vision," in IEEE Latin America Transactions, vol. 17, no. 08, pp. 1259-1267, August 2019, doi: 10.1109/TLA.2019.8932334.

[10] E. Vázquez, “Desarrollo de un Sistema de detección y manipulación de objetos para un robot de servicio,” Tesis de Licenciatura, Universidad Nacional Autónoma de México (UNAM), México, 2018.

[11] Laboratorio de Biorobótica (UNAM), “Justina”, 2018, [Online] Disponible en: https://github.com/RobotJustina/JUSTINA. [Último acceso: abril 2020]

[12] J. J. Craig, *Introduction to robotics. Mechanics and control.* Tercera edición. EUA: Pearson Education, 2005, pp 19, ISBN 0-13.123629-6

[13] M. W. Spong, S. Hutchinson and M. Vidyasagar, *Robot modeling and control*. Primera edición. EUA: John Wiley & Sons, Inc, 2005, pp 36, ISBN 0471649902

[14] International Organization for Standardization, “ISO 8373:2012”, 2012 [Online]. DIsponible en: https://www.iso.org/obp/ui/#iso:std:iso:8373:ed-2:v1:en [Último acceso: abril 2020]

[15] Hägele, Martin & Nilsson, Klas & Pires, J. Norberto. (2007). Industrial Robotics. 10.1007/978-3-540-30301-5_43.

[16] Golnazarian, Wanek & Hall, Ernest. (2002). Intelligent Industrial Robots. 10.1201/9780203908587.ch6.5.

[17] A. Barrientos, L. F. Peñin, C. Balager, R. Aracil, *Fundamentos de robótica*. Madrid, España: Mc-Grawll, 1997, pp 97, ISBN 84-481-0815-9.

[18] R. Szeliski, *Computer Vision. Algorithms and Applications*. Berlin, Alemania: Springer, 2011, pp 812.

[19] J. F. Peters, *Foundations of computer vision. Computational geometry, visual image structures and object shape detection*. Canada: Springer, 2017, pp 2, ISBN 978-3-319-52481-8

[20] M. Treiber, *An introduction to Object Recognition. Selected algorithms for a wide variety of applications*. Londres: Springer, 2010. ISBN 978-1-84996-234-6

[21] T. Tuytelaars, K. Mikolajczyk, et al., “Local invariant feature detectors: a survey,” *Foundations and trends in computer graphics and vision*, vol. 3, no. 3, pp. 177–280, 2008.

[22] S. Ansari, "A Review on SIFT and SURF for Underwater Image Feature Detection and Matching," *2019 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, Coimbatore, India, 2019, pp. 1-4.

[23] T. Lindberg. *Scale-Space Theory in Computer Vision.* Stockholm: Kluwer Academic Publishers, 1994, pp. 8-11. 10.1007/978-1-4757-6465-9.

[24] M. A. Fischler and R. C. Bolles, “Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography,” *Commun of the ACM 24*, pp. 381–395, 1981. 10.1145/358669.358692

[25] A. Tharwat, “Principal component analysis - a tutorial,” International Journal of Applied Pattern Recognition, 3, pp. 197, 2016. 10.1504/IJAPR.2016.079733.

[26] P. White and R. Ingalls, “Introduction to simulation,” in *Proceedings - Winter Simulation Conference,* pp. 12-23, 2009. 10.1109/WSC.2009.5429315.

[27] *KUKA KR5 arc HW*. [Online] Disponible en: https://www.kuka.com/-/media/kuka-downloads/imported/6b77eecacfe542d3b736af377562ecaa/pf0012_kr_5_arc_hw_en.pdf [Último acceso: abril 2020]

[28] KUKA Robot Group, “KR 5 arc HW, KR 5 arc HW-2. Specification”, 2011. [Online] Disponible en: http://supportwop.com/IntegrationRobot/content/2-Robots/Petites_charges(5-16kg)/KR_5_arc/English_Manual_KR_5_arc_HW_en.pdf [Último acceso: abril 2020]

[29] PZN-plus 100-2. [Online]. Disponible en: https://schunk.com/mx_es/tecnologia-de-sujecion/product/2020-0303412-pzn-plus-100-2/ [Último acceso: abril 2020]

[30] Robot Operating System, “urdf,” [Online] Disponible en: http://wiki.ros.org/urdf [Último acceso: abril 2020]

[31] Ioan A. Sucan and Sachin Chitta, "MoveIt", [Online] Disponible en: moveit.ros.org. [Último acceso: abril 2020]

[32] Orocos Kinematics and Dynamics. “KLD Documentation”. [Online] Disponible en: https://www.orocos.org/kdl [Último acceso: abril 2020]

[33] R. Diankov, “Automated Construction of Robotic Manipulation Programs”. Tesis. Carnegie Mellon University, Robotics Institute, 2010. Disponible en: http://www.programmingvision.com/rosen_diankov_thesis.pdf

[34] E. Lachat, H. Macher, T. Landes and P. Grussenmeyer, “Assessment and Calibration of a RGB-D Camera (Kinect v2 Sensor) Towards a Potential Use for Close-Range 3D Modeling”, *Remote sensing,* 2015. 10.3390/rs71013070

[35] A. Islam, M. A. Hossain, Y. M. Jang, “Interference mitigation technique for Time-of-Flight (ToF) cameras”, *Eight international*, Computer science, 2016. 10.1109/ICUFN.2016.7537001

[36] Robot Operating System (ROS), “Documentation”. [Online]. Disponible en: http://wiki.ros.org/es. [Último acceso: abril 2020]

[37] L. Xiang, F. Echtler, C. Kerl *et al*, “libfreenect2: Release 0.2”. Zenodo, 2016. http://doi.org/10.5281/zenodo.50641

[38] T. Wiedemeyer, “IAI Kinect2”, Institute for Artificial Intelligence, University Bremen, 2015. [Online] Disponible en: https://github.com/code-iai/iai_kinect2

[39] M. Labb, “Find-Object”, Introlab, 2011. [Online]. Disponible en: https://github.com/introlab/find-object

[40] Rosten E., Drummond T. (2006) “Machine Learning for High-Speed Corner Detection”. In: Leonardis A., Bischof H., Pinz A. (eds) Computer Vision – ECCV 2006. ECCV 2006. Lecture Notes in Computer Science, vol 3951. Springer, Berlin, Heidelberg. doi: 10.1007/11744023_34

[41] Y. Biadgie and K. Sohn, “Feature Detector Using Adaptative Accelareted Segment Test,” *2014 International Conference on Information Science & Applications* (ICISA), Seúl, 2014. pp. 1-4. doi: 10.1109/ICISA.2014.6847403.

[42] M. Calonder, V. Lepetit, C. Strecha and P. Fua, “BRIEF: Binary Robust Independent Elementary Features,” European Conference on Computer Vision, 6314, 2010, pp. 778-792. doi: 10.1007/978-3-642-15561-1_56.

[43] E. Rublee, V. Rabaud, K. Konolige and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," *2011 International Conference on Computer Vision*, Barcelona, 2011, pp. 2564-2571, doi: 10.1109/ICCV.2011.6126544.

[44] K. Yang, D. Yin, J. Zhang, H. Xiao and K. Luo, "An Improved ORB Algorithm of Extracting Features Based on Local Region Adaptive Threshold," *2019 6th International Conference on Systems and Informatics (ICSAI)*, Shanghai, China, 2019, pp. 1212-1217.

[45] R. Sun *et al*., "A Flexible and Efficient Real-Time ORB-Based Full-HD Image Feature Extraction Accelerator," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 2, pp. 565-575, Feb. 2020.

[46] S. Leutenegger, M. Chli and R. Y. Siegwart, "BRISK: Binary Robust invariant scalable keypoints," *2011 International Conference on Computer Vision*, Barcelona, 2011, pp. 2548-2555, doi: 10.1109/ICCV.2011.6126542.

[47] E. Mair, G. D. Hager, D. Burschka, M. Suppa, and G. Hirzinger, "Adaptive and generic corner detection based on the accelerated segment test". *Proceedings of the European Conference on Computer Vision (ECCV)*, 2010. 2, 5. doi: *10.1007/978-3-642-15552-9_14.*

[48] Robot Operating System (ROS). "rviz", [Online] Disponible en: http://wiki.ros.org/rviz [Último acceso: abril 2020]

[49] Open Source Robotics Foundation. "Gazebo. Simulation made easy", 2014. [Online] Disponible en: http://gazebosim.org/ [Último acceso: abril 2020]

[50] Robot Operating System (ROS). "Solidworks to URDF Exporter". [Online] Disponible en: http://wiki.ros.org/sw_urdf_exporter [Último acceso: abril 2020]

[51] Willow Garage and ROS community, "ORK: Object Recognition Kitchen". [Online] Disponible en: https://github.com/wg-perception/object_recognition_core [Último acceso: octubre 2020]