DESDE 2013
https://repository.uaeh.edu.mx/revistas/index.php/icbi/issue/archive
Pädi Boletín Científico de Ciencias Básicas e Ingenierías del ICBI
ISSN: 2007-6363

Publicación Semestral Pädi Vol. 14 No. Especial (2026) 75–83

Detección en tiempo real de las dimensiones de objetos usando cámaras RGB-D para navegación móvil
Real-time detection of object dimensions using RGB-D cameras for mobile navigation

J. C. Sernaque-Julca [a], G. Quiroz-Compean [a,*]

[a] *Facultad de Ingeniería Mecánica y Eléctrica, Universidad Autónoma de Nuevo León, 66455, San Nicolás de los Garza, Nuevo León, México.*

Resumen

Este estudio compara dos métodos para estimar dimensiones (ancho, alto) y distancia de objetos 3D usando una cámara RGB-D. El primer método utiliza datos de profundidad y el segundo agrega datos de color. Ambos procesan nubes de puntos mediante la librería *Point Cloud Library*, aplicando filtrado y segmentación. En las pruebas experimentales se evaluaron cilindros y cajas distribuidos a menos de 1.25 m desde la cámara y en condiciones controladas para evaluar cómo afectan los parámetros (resolución, diezmado y voxels) al rendimiento del sistema. Luego, con cámara en movimiento (0.15 m/s), se determinó que el segundo método es más preciso que el primero, obteniendo un error absoluto medio de: 9.60×10^{-3} m *vs.* 6×10^{-3} m (alto) y 1.17×10^{-2} m *vs.* 4.50×10^{-3} m (ancho). Ejecutándose, ambos, a 22 fotogramas por segundo, demostrando la utilidad de los métodos para percepción 3D en tiempo real usando una Raspberry Pi 4B+ y programación en C/C++.

Palabras Clave: Detección de objetos 3D, Procesamiento de nubes de puntos, Navegación autónoma.

Abstract

This study compares two methods for estimating dimensions (width, height) and distance of 3D objects using an RGB-D camera. The first method uses depth data, while the second method adds color data. Both process point clouds using the Point Cloud Library, applying filtering and segmentation (decimation, Voxel, Random Sample Consensus, and Euclidean Clustering). The tests evaluated cylinders and boxes distributed within 1.25 m of the camera and under controlled conditions to assess how the parameters (resolution, decimation, and voxels) affect system performance. Then, with a moving camera (0.15 m/s), the second method was determined to be more accurate than the first, obtaining an mean absolute error of: 9.60×10^{-3} m *vs.* 6×10^{-3} m (height) y 1.17×10^{-2} m *vs.* 4.50×10^{-3} m (width). Running both, at 22 frames per second, demonstrating the usefulness of the methods for real-time 3D perception using a Raspberry Pi 4B+ and C/C++ programming.

Keywords: 3D Object Detection, Point Cloud Processing, Autonomous Navigation.

1. Introducción

La percepción 3D es esencial en robótica y navegación autónoma, especialmente en aplicaciones que demandan precisión, robustez y eficiencia en tiempo real (Zhou, 2022; Zhou *et al.*, 2024). Entre las tecnologías disponibles, las cámaras RGB-D destacan por integrar segmentación semántica RGB con datos de profundidad, mejorando la detección 3D a un costo accesible (Mao *et al.*, 2023).

Aunque presentan limitaciones, como nubes de puntos dispersas y mediciones con cierto margen de error, su capacidad para procesar información multimodal (color + profundidad) en tiempo real las hace idóneas para entornos industriales y robóticos que priorizan la relación costo-rendimiento. Alternativas como los sistemas LiDAR/multimodales ofrecen mayor precisión, pero su elevado precio restringe su uso masivo. Mientras tanto, los detectores 3D multivista (Park *et al.*, 2022) parecen ser una solución prometedora a futuro (Mao *et al.*, 2023). En este contexto, las cámaras RGB-D se posicionan como una opción práctica actual, combinando versatilidad y accesibilidad (Wang *et al.*, 2021).

*Autor para correspondencia: griselda.quirozcm@uanl.edu.mx
Correo electrónico: juan.sernaquej@uanl.edu.mx (Juan Carlos Sernaque Julca), griselda.quirozcm@uanl.edu.mx(Griselda Quiroz Compéan).

Historial del manuscrito: recibido el 16/07/2025, última versión-revisada recibida el 21/12/2025, aceptado el 13/12/2025, publicado el 25/03/2026. **DOI:** https://doi.org/10.29057/icbi.v14iEspecial.15512

Los avances en percepción 3D aún enfrentan desafíos clave relacionados con la precisión, eficiencia, fusión multimodal e implementación en sistemas embebidos. Por un lado, lograr un equilibrio entre precisión y eficiencia resulta complicado, por ejemplo, al usar métodos basados en grids (filtro de diezmado, voxel, etc) donde mayor resolución implica mayor uso de memoria. Por otro lado, la fusión multimodal (ejemplo: color y profundidad) mejora la precisión pero aumenta significativamente el costo computacional; Además, la mayoría de los métodos para datos 3D se implementan en GPUs de forma ineficiente, limitando su uso en sistemas embebidos (Mao *et al.*, 2023). Pocos trabajos abordan directamente la optimización computacional, destacando la necesidad de mejores soluciones (Lin *et al.*, 2021; Lang *et al.*, 2019).

Este trabajo aborda el problema de la detección de dimensiones (ancho, alto) y distancia de objetos para navegación móvil que combina eficiencia y precisión mediante dos enfoques complementarios:

- Un método ligero basado exclusivamente en datos de profundidad y Clustering Euclidino (CE).
- Un método de fusión de datos (color y profundidad) que busca mejorar la precisión al corregir adecuadamente artefactos en el mapa de profundidad.

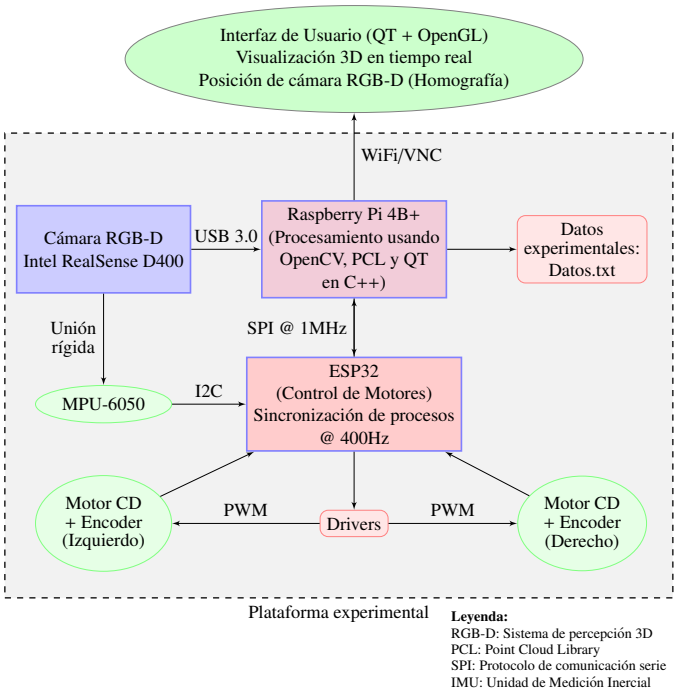
Ambos métodos utilizan un algoritmo de seguimiento de objetos 3D basado en la técnica de asociación del vecino más cercano (NNA, por las siglas en Inglés de *Nearest Neigbor Association*) y se evalúan analizando el rendimiento en condiciones estáticas (objetos y cámara estáticos) y dinámicas (cámara en movimiento), para determinar las configuraciones óptimas que equilibran rendimiento y precisión.

Para estos fines, la Sección 2 de este documento describe la plataforma experimental utilizada y detalla los algoritmos propuestos. La Sección 3 presenta el protocolo experimental y los resultados basados en métricas de desempeño y precisión. La Sección 4 analiza los resultados, mientras que sus implicaciones en navegación autónoma se discuten en la Sección 5.

2. Metodología

2.1. Descripción del Sistema

La Figura 1 muestra la arquitectura hardware/software del sistema robótico móvil para la detección en tiempo real de las dimensiones de objetos. Tal como se describe en la Figura 1a, el sistema utiliza una cámara Intel RealSense D400 con resolución máxima de 1280×720 pixeles (RGB) y 640×480 pixeles (profundidad), con un campo de visión de 69.4°×42.5° y rango operativo de 0.2 a 10 metros. Este sensor se conecta mediante USB 3.0 a una Raspberry Pi 4B+ equipada con un procesador quad-core ARM Cortex-A72 a 1.5 GHz y memoria LPDDR4. Para el procesamiento 3D, la Raspberry Pi ejecuta algoritmos basados en *Point Cloud Library* (PCL) y OpenCV. Se tiene un subsistema de control que opera a 400 Hz, utiliza un microcontrolador ESP32 y sincroniza todos los procesos mediante interrupciones. Este subsistema gestiona dos motores DC controlados por señal PWM, recibe datos de orientación (ϕ: Roll, θ: Pitch, ψ: Yaw) de una unidad MPU-6050 vía I2C unida rígidamente a la cámara y de los encoders unidos a los ejes de los motores. La comunicación entre la Raspberry Pi y el ESP32 se realiza mediante SPI a 5 MHz. Una interfaz gráfica de usuario (GUI) desarrollada en QT con renderizado OpenGL permite visualización remota vía WiFi/VNC. Además, se tiene un módulo externo compuesto por una cámara RGB fija y un algoritmo basado en OpenCV que calcula en tiempo real la posición exacta de la cámara Realsense D400. Todos los componentes están integrados en una plataforma robótica móvil que registra los datos experimentales para su posterior análisis, ver Figura 1b.



(a)

(b)

Figura 1: Plataforma experimental para detectar en tiempo real las dimensiones de objetos usando cámaras RGB-D. (a) Diagrama de bloques de la arquitectura mostrando los componentes de hardware (cámara RGB-D, Raspberry Pi 4B+, ESP32, motores y sensores), interfaces de comunicación (USB, SPI, I2C, PWM) y software (procesamiento PCL, interfaz QT/OpenGL). (b) Estructura móvil donde están montados todos los componentes necesarios para experimentos con la cámara.

La plataforma experimental, descrita en la Figura 1b, se emplea para evaluar dos métodos orientados al cálculo en tiempo real de las dimensiones (ancho, alto y distancia) de objetos. El

primer método (Método 1) hace uso exclusivo de información de profundidad con cámara y objetos fijos. El segundo método (Método 2) integra información de color y profundidad para mejorar la precisión. Los métodos se evalúan en dos arreglos experimentales bien definidos, ver Tabla 1. El primero (Arreglo experimental 1) se utiliza para establecer las condiciones básicas para la ejecución eficiente de los métodos 1 y 2 en la plataforma experimental: resolución máxima, parámetros de procesamiento y su impacto en el rendimiento; además del impacto de la forma de los objetos (cilindros y cajas) en el desempeño, aislando los efectos del movimiento relativo entre los objetos y la cámara. El segundo (Arreglo experimental 2) analiza simultáneamente: (1) el efecto del movimiento relativo cámara-objeto en la exactitud de las dimensiones y (2) el impacto de los métodos propuestos (1 y 2) en el desempeño, así como el rendimiento computacional.

Tabla 1: Arreglos experimentales para el cálculo de dimensiones usando el Método 1 (solo datos de profundidad) y el Método 2 (fusión de datos de profundidad y color).

Parámetro	Arreglo 1 (Estático)	Arreglo 2 (Dinámico)
Sensores	D+MPU	RGB-D + MPU + Encoders
Movimiento de cámara	-	Velocidad constante
Objetos	Cilindros y cajas (Estáticos)	Cilindros (Estáticos)
Control Adquisición	– Manual	ESP32 + PD Automatizada

En las siguientes secciones se detallan las técnicas propuestas para procesar los datos de profundidad y color para cada método, incluyendo los algoritmos de detección, un algoritmo de seguimiento de objetos y los métodos de calibración utilizados para garantizar la precisión métrica en ambos escenarios.

2.2. *Método 1: Detección de dimensiones usando exclusivamente información de profundidad*

Para determinar las dimensiones de objetos usando datos de profundidad (Método 1), con cámara y objetos fijos (ver Tabla 1 y Figura 4), se ha propuesto un flujo de proceso donde se hace uso exclusivo de la información de profundidad que va desde la captura de imagen de profundidad (D) hasta la extracción de las dimensiones (w_k, h_k) y distancia (d_k) de cada objeto C_j detectado, ver Figura 2. El proceso inicia con el preprocesamiento de D, eliminando pixeles con profundidad D_{ij} mayor a un umbral h_d y reduciendo su resolución mediante un filtro de diezmado para optimizar el cómputo (RealSense™, 2025a; IntelCorporation, 2024). La imagen resultante se convierte a una nube de puntos (P) utilizando la orientación de la cámara ($\boldsymbol{\Theta} = (\phi, \theta, \psi)$) en un formato que es compatible con la librería PCL (RealSense™, 2025b). La eliminación de pixeles, el filtro de diezmado y la conversión a nube de puntos se realiza usando Intel RealSense SDK 2.54.2.

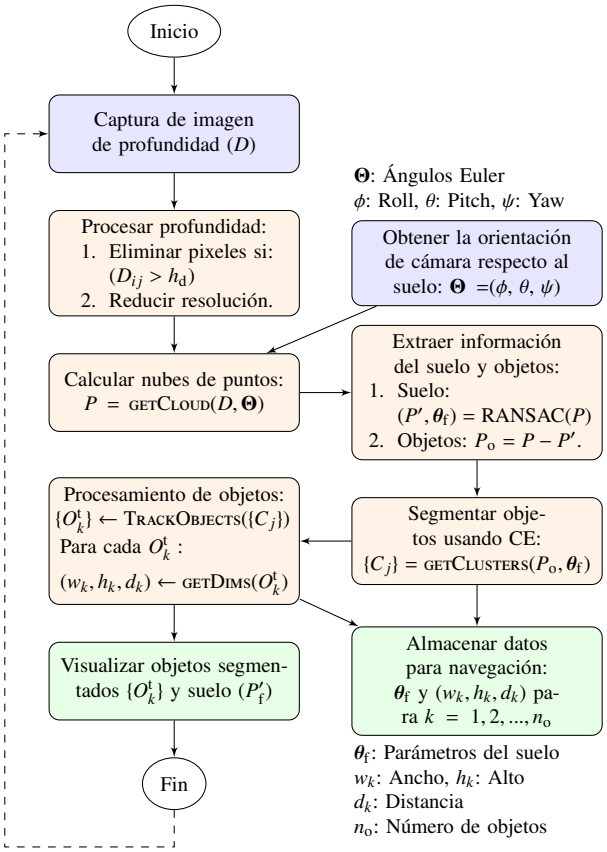


Figura 2: Diagrama de flujo del algoritmo para detección de las dimensiones de objetos usando solamente datos de profundidad (Método 1). El proceso inicia con el preprocesamiento de D, eliminando pixeles con distancia mayor a h_d y reduciendo su resolución para optimizar cómputo. La imagen resultante se convierte a una nube de puntos (P) utilizando la orientación de la cámara ($\boldsymbol{\Theta} = (\phi, \theta, \psi)$). Mediante RANSAC, se separa automáticamente el suelo (P') de los objetos ($P_o = P - P'$), estimando simultáneamente la inclinación del suelo ($\boldsymbol{\theta}_f$). Los objetos se agrupan mediante CE ($\{C_j\}$ = getClusters($P_o, \boldsymbol{\theta}_f$)), filtrando puntos anómalos bajo el suelo. Finalmente, se extraen las dimensiones (w_k, h_k) y distancia (d_k) de cada objeto $\{O_k^\mathrm{t}\}$.

El algoritmo *Random Sample Consensus* (RANSAC) (Fischler y Bolles, 1981) se emplea para estimar el plano del suelo Π a partir de la nube de puntos P, descrito por la ecuación general del plano:

$$\Pi : Ax + By + Cz + D = 0, \tag{1}$$

donde $\boldsymbol{\theta} = (A, B, C, D)^\mathrm{T}$ es el vector de parámetros del plano normalizado ($\|(A, B, C)\|_2 = 1$) y (x, y, z) son coordenadas en el sistema de referencia de la cámara RGB-D. El algoritmo genera hipótesis de un plano a partir de muestras aleatorias conformadas por 3 puntos no colineales, evaluando cada una mediante el conteo de *inliers* (puntos que cumplen $|Ax_i+By_i+Cz_i+D| < \tau_\mathrm{in}$). Tras alcanzar el criterio de convergencia, se refina la solución con mínimos cuadrados sobre los mejores *inliers*. Esto permite obtener el subconjunto $P' \subset P$ y los parámetros $\boldsymbol{\theta}$ que caracterizan la superficie del suelo. En este trabajo, se hace uso de la librería PCL para aplicar RANSAC y obtener $P' \subset P$ y $\boldsymbol{\theta}$ a través de la función pcl::SACSegmentation::segment(P, M_plano, τ_in, N_iter) (PCL, 2023), donde M_plano corresponde al modelo del

plano dado por la Ecuación (1), τ_{in} es el umbral de distancia para los *inliers* y N_{iter} el número máximo de iteraciones. La función de PCL se puede escribir de manera simplificada por $(P', \boldsymbol{\theta}_{\text{f}}) = \text{RANSAC}(P)$. A continuación, se extraen los puntos correspondientes a objetos mediante la sustracción de conjuntos $P_o = P - P'$. Estos puntos (P_{o}) se agrupan mediante CE: $\{C_j\} = \text{GETCLUSTERS}(P_{\text{o}}, \boldsymbol{\theta}_{\text{f}}, \epsilon_{\text{cluster}})$, donde $\epsilon_{\text{cluster}}$ es el radio de búsqueda; el cual ha sido seleccionado por su simplicidad, eficiencia y facilidad de implementación (Rusu y Cousins, 2011). El algoritmo busca identificar grupos de puntos, por ejemplo, $\boldsymbol{p_1} = (x_1, y_1, z_1)$ y $\boldsymbol{p_2} = (x_2, y_2, z_2) \in P_{\text{o}}$ cuya distancia $d(\boldsymbol{p_1}, \boldsymbol{p_2}) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$ sea menor a $\epsilon_{\text{cluster}}$. La librería PCL emplea una estructura *KdTree* para búsquedas espaciales eficientes, usando como parámetros: la tolerancia de cluster $\epsilon_{\text{cluster}}$ y el tamaño mínimo ($N_{\text{mín}}$) y máximo ($N_{\text{máx}}$) por cluster. El algoritmo procesa exclusivamente nubes no vacías ($P_o \neq \varnothing$), generando un conjunto de clusters $\{C_j\}_{j=1}^{n_{\text{o}}}$ donde cada C_j representa un grupo de puntos conectados espacialmente. Los valores de $\epsilon_{\text{cluster}}$ se ajustan dinámicamente mediante la interfaz de usuario, con un valor mínimo garantizado de 0.001 m para la tolerancia. También, se incluye una etapa de filtrado que elimina puntos bajo el plano del suelo (usando $\boldsymbol{\theta}_{\text{f}}$).

Para fines de evaluación, se podrían extraer datos experimentales (dimensiones y distancia) las métricas del impacto en el rendimiento del sistema de los parámetros de preprocesamiento (diezmado, voxel, umbral de profundidad, etc.) usando directamente la información de los clusters; sin embargo, para fines de navegación (con movimiento relativo de cámara-objetos) se implementa un algoritmo de seguimiento para etiquetar y seguir a lo largo del tiempo los clusters, ver Sección 2.3.1. A continuación se presenta el segundo método, que combina información de profundidad y color.

2.3. *Método 2: Detección de dimensiones usando información de profundidad y color*

Para el Método 2, se ha propuesto un proceso de detección de dimensiones que combina información de profundidad y color (ver la Figura 3). Inicialmente, se aplica un umbral de profundidad h_{d} al frame de profundidad D para eliminar pixeles fuera del rango de interés (ROI) ($D_{ij} > h_{\text{d}}$), generando una máscara binaria $M_{\text{d}} \in \{0, 1\}^{m \times n}$ con los pixeles dentro del ROI. Sobre M_{d}, se extraen contornos $\{p_{\text{d}}^{\text{d}}\}$ mediante la función FINDCONTOURS($\cdot$) implementada en OpenCV. Estos contornos, cuyas posiciones están en el espacio de profundidad, se transforman al espacio RGB ($\{p_{\text{d}}^{\text{c}}\}$) mediante (Zhang, 2000):

$$\{p_{\text{d}}^{\text{c}}\} = \pi_{\text{c}}(\mathbf{T}_{\text{d}\to\text{c}} \cdot \pi_{\text{d}}^{-1}(p_{\text{d}}^{\text{d}}, z)), \tag{2}$$

donde $\pi_d^{-1}(\cdot)$ es la proyección inversa a 3D usando parámetros intrínsecos del sensor de profundidad, $\mathbf{T}_{\text{d}\to\text{c}} \in \mathbb{R}^{4\times4}$ es la matriz extrínseca de calibración entre sensores RGB y D (IntelCorporation, 2023b), $\pi_c(\cdot)$ es la proyección perspectiva al plano RGB.

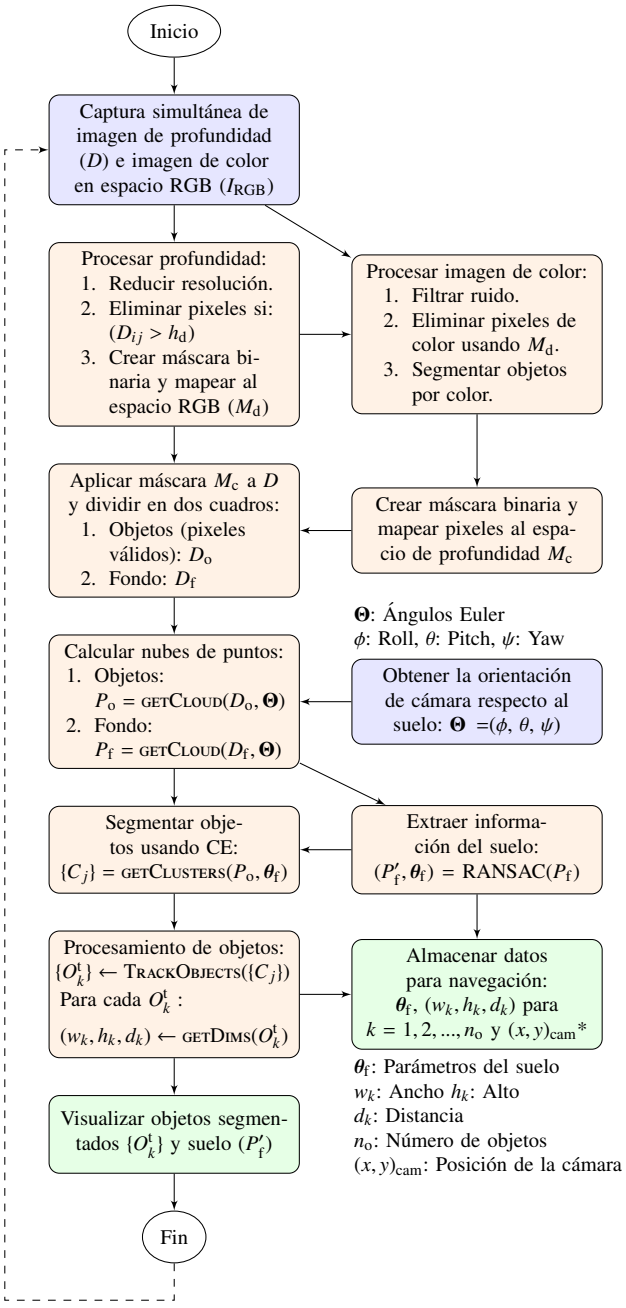


Figura 3: Diagrama de flujo del algoritmo de detección de objetos en tiempo real mediante fusión de imágenes RGB-D (Método 2). El proceso inicia con la captura simultánea de una imagen de color (I_{RGB}) y una de profundidad (D). La imagen RGB se procesa para segmentar objetos basados en color, generando una máscara binaria que se aplica a D, dividiéndola en dos cuadros: uno para objetos (D_{o}) y otro para el fondo (D_{f}). Ambos cuadros se convierten a nubes de puntos (P_{o} y P_{f}) usando la pose de la cámara ($\boldsymbol{\Theta}$). La nube de objetos (P_{o}) se segmenta mediante CE, mientras que el fondo (P_{f}) se procesa con RANSAC para estimar parámetros del suelo ($\boldsymbol{\theta}_{\text{f}}$); P_{o} es corregida mediante la eliminación puntos anómalos (que se encuentran bajo el suelo) usando $\boldsymbol{\theta}_{\text{f}}$. Finalmente, se extraen las dimensiones (w_k, h_k) y distancia (d_k) de cada objeto y se visualizan/almacenan los resultados. *La posición de la cámara $(x, y)_{\text{cam}}$ se obtiene externamente por homografía y se envía por WiFi.

Posteriormente, en el espacio RGB, se aplica filtrado mor-

fológico a la imagen de color usando $\{p_{\mathrm{d}}^{\mathrm{c}}\}$ como ROI, la cual fue calculada usando la Ecuación (2). Luego se segmenta por color, generando una máscara M_{c}, se extraen contornos $\{p_{\mathrm{c}}^{\mathrm{c}}\}$ que contiene los objetos detectados y se mapean al espacio de profundidad ($\{p_{\mathrm{c}}^{\mathrm{d}}\}$) invirtiendo la transformación usando la siguiente ecuación:

$$p_{\mathrm{c}}^{\mathrm{d}} = \pi_{\mathrm{d}}(\mathbf{T}_{\mathrm{c}\to\mathrm{d}} \cdot \pi_{\mathrm{c}}^{-1}(p_{\mathrm{c}}^{\mathrm{c}}, z)), \tag{3}$$

donde, $\mathbf{T}_{\mathrm{c}\to\mathrm{d}} = \mathbf{T}_{\mathrm{d}\to\mathrm{c}}^{-1}$, $p_{\mathrm{c}}^{\mathrm{d}}$ son los contornos mapeados del espacio de color al de profundidad.

En la implementación, la Ecuación (3) se aplica a cada pixel del contorno mediante el siguiente procedimiento: (I) obtener su valor de profundidad (z) desde la imagen de profundidad, (II) si el valor no es válido, se interpola localmente, (III) se realiza la desproyección del pixel con profundidad al espacio 3D del sistema de color (RS2_DEPROJECT_PIXEL_TO_POINT), (IV) el punto 3D se transforma al sistema de coordenadas de profundidad (RS2_TRANSFORM_POINT_TO_POINT) y (V) finalmente, se proyecta al plano de imagen de la cámara de profundidad (RS2_PROJECT_POINT_TO_PIXEL).

Luego, los contornos mapeados $p_{\mathrm{c}}^{\mathrm{d}}$ se aplican a D y se extraen dos cuadros: uno para los objetos (D_{o}) y otro para el fondo (D_{f}). Ambos cuadros se convierten a nubes de puntos (P_{o} y P_{f}) usando la orientación de la cámara $\mathbf{\Theta} = (\phi, \theta, \psi)$ para obtener las nubes de puntos alineadas con un sistema inercial $\{W(x, y, z)\}$. La nube de puntos que contiene el fondo (P_{f}) se procesa con el algoritmo RANSAC usando PCL ($(P'_{\mathrm{f}}, \boldsymbol{\theta}_{\mathrm{f}}) = \mathrm{RANSAC}(P_{\mathrm{f}})$) que toma como argumento la nube de puntos P_{f} y devuelve los parámetros que definen el plano del suelo ($\boldsymbol{\theta}_{\mathrm{f}} = (A, B, C, D)$) y los puntos que se ajustan al modelo (P'_{f}). La información del suelo $\boldsymbol{\theta}_{\mathrm{f}}$ se utiliza para corregir puntos anómalos dentro de los clusters (C_j) que erróneamente se mapean debajo del suelo. Luego se etiquetan los objetos O_k^{t} usando un algoritmo de seguimiento basado en la técnica NNA y finalmente se extraen las dimensiones (w_k, h_k) y distancia (d_k) de cada uno de ellos. Estas se visualizan/almacenan para su posterior análisis.

El CE solo retorna puntos agrupados espacialmente, pero no los etiqueta, esto es un problema en aplicaciones de navegación, pues es necesario que haya un seguimiento adecuado de los objetos a lo largo del tiempo para implementar tareas en línea como, diseño de trayectorias, evasión de obstáculos, entre otros. Como solución, se propone un esquema de asociación basado en NNA por su equilibrio entre eficiencia computacional y precisión en entornos con movimientos moderados (Bochinski *et al.*, 2017).

2.3.1. *Seguimiento de objetos 3D basado en NNA*

El algoritmo de seguimiento propuesto está diseñado específicamente para realizar el seguimiento de objetos 3D en una nube de puntos P_{o} que contiene solamente la información de los objetos frente a la cámara y que previamente se le ha extraído la información del suelo (RANSAC) y el fondo (pixeles Dij fuera de rango h_{d}), ver Algoritmo 1. El procedimiento propuesto opera en cuatro etapas secuenciales. Primero, calcula los centroides $\mathbf{c}_i = \frac{1}{N_{C_i}} \sum_{\mathbf{p}\in C_i} \mathbf{p}$ para cada cluster detectado C_i, donde N_{C_i} es el número de puntos del cluster y $\mathbf{p} \in \mathbb{R}^3$ representa un punto en la nube. Luego, se resuelve el problema de asociación mediante NNA definiendo un umbral h_c e implementando una función $id\_map[j] = \textsc{NearestNeighbor}(\{\mathbf{c}_j\}, \{O_k^{\mathrm{t}}.\boldsymbol{c}\}, h_c)$ que evalúa los nuevos objetos y los relaciona con los previamente detectados en función de la probabilidad de correspondencia bajo la restricción $\|\mathbf{c}_j - O_k^{\mathrm{t}}.\boldsymbol{c}\|_2 \le h_c$. Para aquellos objetos no asociados ($id\_map[j] = 0$), el sistema genera nuevos IDs. La etapa de actualización modifica dinámicamente los parámetros de los tracks existentes ($O_k^{\mathrm{t}}.\boldsymbol{c}, O_k^{\mathrm{t}}.C) \leftarrow (\mathbf{c}_j, C_j$) y actualiza un registro de tiempo (última vez que fue detectado) T^{last} al tiempo actual. Finalmente, elimina tracks inactivos cuando $\textsc{CurrentTime}() - T^{last} > \tau$, donde τ determina el tiempo máximo de oclusión tolerable. Este enfoque ha sido diseñado para funcionar de manera eficiente en la Raspberry Pi 4B+, ya que no requiere mucha potencia computacional.

Algoritmo 1 Seguimiento de Objetos 3D mediante NNA.

Require:
DetectedObjects: Conjunto de clusters detectados $\{C_i\}$
τ: Umbral temporal para eliminación de tracks (segundos)
h_c: Umbral de distancia máxima para asociación (metros)
trackedObjects: Conjunto de objetos seguidos $\{O_k^{\mathrm{t}}\}_{k=1}^{n_{\mathrm{o}}}$
donde $O_k^{\mathrm{t}} = (ID, \boldsymbol{c}, C, T^{last})_k$ con:
ID: Identificador único
$\boldsymbol{c}$: Centroide del objeto en $\mathbb{R}^3$
C: Nube de puntos del objeto
T^{last}: Último tiempo de detección

Ensure:
trackedObjects: Conjunto actualizado de objetos seguidos

procedure TrackObjects
  Cálculo de centroides
  for cada $C_i \in$ *DetectedObjects* **do**
    $\mathbf{c}_i \leftarrow \textsc{ComputeCentroid}(C_i)$
  end for
  Asociación de objetos
  $id\_map \leftarrow \textsc{NearestNeighbor}(trackedObjects, \{\mathbf{c}_i\}, h_c)$
  ▷ $id\_map[j] = \begin{cases} k \in [1, ..., n_{\mathrm{o}}] & \text{si se asocia a algún } O_k^{\mathrm{t}} \\ 0 & \text{sin asociación} \end{cases}$
  Actualización de estado
  for cada par $(C_j, \mathbf{c}_j) \in$ *DetectedObjects* **do**
    if $id\_map[j] = \emptyset$ **then** ▷ Nuevo objeto
      $ID \leftarrow \textsc{GenerateUniqueID}()$
      $trackedObjects \leftarrow trackedObjects \cup \{(ID, \mathbf{c}_j, C_j, \textsc{CurrentTime}())\}$
    else ▷ Objeto existente
      $O_{id\_map[j]}^{\mathrm{t}}.\boldsymbol{c} \leftarrow \mathbf{c}_j$
      $O_{id\_map[j]}^{\mathrm{t}}.C \leftarrow C_j$
      $O_{id\_map[j]}^{\mathrm{t}}.T^{last} \leftarrow \textsc{CurrentTime}()$
    end if
  end for
  Eliminación de objetos perdidos
  for cada $O_k^{\mathrm{t}} \in$ *trackedObjects* **do**
    if $\textsc{CurrentTime}() - O_k^{\mathrm{t}}.T^{last} > \tau$ **then**
      $trackedObjects \leftarrow trackedObjects \setminus \{O_k^{\mathrm{t}}\}$
    end if
  end for
end procedure

2.4. *Calibración de la cámara RealSense D400*

La calibración de la cámara Intel RealSense D400 se realizó siguiendo los protocolos recomendados por

el fabricante (IntelCorporation, 2023b), combinando el firmware integrado (OnChip-Calibration) con ajustes manuales (Tare-Calibration) mediante la herramienta `realsense-viewer` proporcionada con Intel RealSense SDK 2.54.2. Inicialmente, se ejecutó la calibración intrínseca mediante el método OnChip-Calibration del Intel RealSense SDK 2.54.2 (IntelCorporation, 2023a), que ajusta automáticamente los parámetros de distorsión radial y tangencial del sensor de profundidad mediante un proceso de optimización no lineal basado en patrones de luz infrarroja. Este método, diseñado para corregir desalineaciones entre los sensores infrarrojos izquierdo y derecho, reporta una precisión de ±0.10 % del rango de profundidad (equivalente a ±2 mm a 2 metros) según la documentación técnica. Para la Tare-Calibration, se utilizó como referencia un plano rígido (60 × 60 cm) posicionado perpendicularmente al eje óptico a una distancia nominal de 1.50 metros. Dado que no se disponía de un láser de clase II, la distancia de referencia se verificó con un flexómetro calibrado (precisión certificada de ±1.50 mm/m según norma ISO 9001), asegurando que las mediciones de la cámara estuvieran dentro de este rango de tolerancia. Luego de la calibración de la cámara, se implementan los Métodos 1 y 2 en lenguaje C/C++ utilizando una interfaz de usuario, para extraer métricas de rendimiento y precisión métrica. En la siguiente sección se detallan dichos los resultados.

3. Experimentos

Los algoritmos de detección de dimensiones (Método 1 y 2, Figuras 2–3) se evaluaron mediante dos arreglos experimentales: una con objetos y cámara estáticos, otra con cámara en movimiento a velocidad constante (ver Figuras 4 y 5). El Arreglo experimental 1 utiliza cuatro objetos de referencia (cilindros de ∅ 5-10 cm y cajas de 23×25×7 cm) posicionados estáticamente dentro del campo visual de la cámara, con separación mínima de 5 cm para prevenir errores de segmentación por profundidad. El arreglo experimental emplea iluminación artificial controlada con fuente centrada sobre la cámara y una superficie negra no reflectante para minimizar interferencias ópticas, ver en la Figura 4. Este diseño aísla los efectos de la geometría del objeto en las mediciones (ancho, alto y distancia), eliminando variables de movimiento y ambientales. Este arreglo experimental analizó el rendimiento del sistema midiendo la tasa de fotogramas (FPS) promedio en 15 ejecuciones para cada caso, variando sistemáticamente parámetros modificables a través de una interfaz gráfica (resolución: 1280×720 a 156×144; factor de diezmado: 1–8; tamaño de voxel: $0 - 3.50 \times 10^{-2}$ m) bajo condiciones controladas (iluminación constante, objetos fijos), adicionalmente se capturaron 30 muestras de las mediciones (ancho, alto y distancia) de cuatro objetos (2 cajas y 2 cilindros) para cuantificar la precisión métrica usando el Error Absoluto Medio (MAE, del inglés *Mean Absolute Error*) como métrica y la relación entre la forma y la precisión.

El Arreglo experimental 2 considera el sistema en condiciones de movimiento controlado, utilizando dos cilindros (∅ 5-10 cm) estáticos sobre una superficie negra no reflectante. La cámara, montada en la plataforma móvil (Figura 1b), se desplaza a velocidad constante (v_{cam}) mientras procesa los datos de profundidad y color, ver Figura 5.

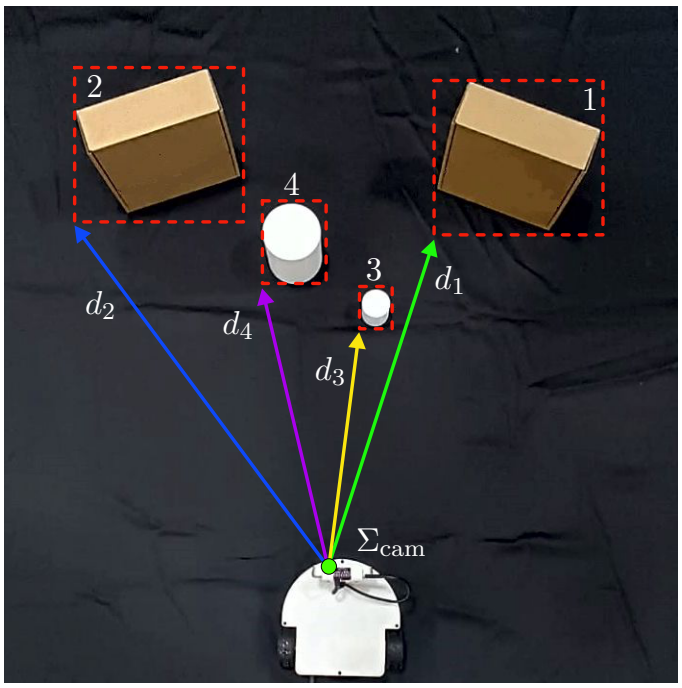


Figura 4: Distribución espacial de objetos del Arreglo experimental 1 (estático). La cámara y los objetos (cilindros: ∅ 5-10 cm; cajas: 23×25×7 cm) permanecen fijos, con cada objeto ($i = 1, ..., 4$) posicionado a una distancia d_i de la cámara (Σ_{cam}).

Este diseño permitió evaluar el desempeño mediante 15 repeticiones con la cámara moviéndose en trayectorias lineales a $v_{\text{cam}} = 0.15$ m/s durante 30 s, extrayendo estadísticas de precisión en la estimación de las dimensiones bajo condiciones de movimiento.

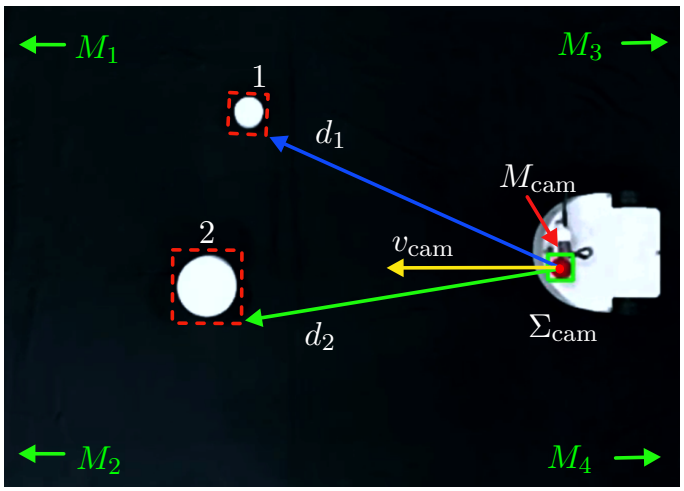


Figura 5: Distribución de objetos para el Arreglo experimental 2 (dinámico). Los objetos (cilindros: ∅ 5-10 cm) permanecen estáticos y están posicionados a una distancia d_i, $i = 1, 2$, respecto a la cámara (Σ_{cam}). La cámara se desplaza a velocidad constante (v_{cam}). Se usan marcas M_j, $j = 1, ..., 4$ en los bordes del espacio de trabajo para determinar la posición de la cámara (M_{cam}) mediante homografía.

3.1. Evaluación del rendimiento del sistema

La Tabla 2 muestra la tasa de adquisición en FPS de la imagen de profundidad D para diferente resolución. Para la captura de datos, solamente se realiza la adquisición, no el procesamiento de cuadro.

Tabla 2: Frecuencia de muestreo de la adquisición de datos de profundidad con diferentes resoluciones.

Resolución	Promedio (FPS)	σ
1280×720	20.46	±0.41
640×480	29.81	±0.62
480×270	**31.29**	±0.40
156×144	31.22	±0.18

Mientras que la Tabla 3 compara el rendimiento para diferentes parámetros de diezmado y voxel con una resolución fija (480×270). Se utiliza el Método 1 para todos los casos de la tabla, con la finalidad de que los resultados sean, comparables.

Tabla 3: Rendimiento combinado por diezmado y tamaño de voxel.

Voxel (mm)	FPS (Promedio ±σ)					
	Diezmado: 8		Diezmado: 4		Sin diezmado	
35	22.29	±1.86	17.88	±2.73	9.45	±0.19
15	**21.63**	±1.96	18.48	±1.35	8.87	±0.31
5	20.59	±1.77	12.20	±1.79	7.08	±0.16
0	15.63	±0.45	0.65	±0.03	0.65	±0.03

3.2. Detección de dimensiones con objetos de diferente forma

Las figuras 6 y 7 muestran respectivamente la imagen de profundidad y los clusters detectados mediante el Método 1 (ver Figura 2) con cámara y objetos estáticos (Arreglo experimental 1, ver la Figura 4). La Tabla 4 reporta las mediciones de cuatro objetos (dos cilindros: ∅ 5-10 cm; dos cajas: 23×25×7 cm), incluyendo distancia, ancho y alto, con sus correspondientes errores absolutos y relativos. Los objetos segmentados se muestran en colores verde (Objeto 1), azul (Objeto 2), amarillo (Objeto 3) y rosa (Objeto 4), ver Figura 4.

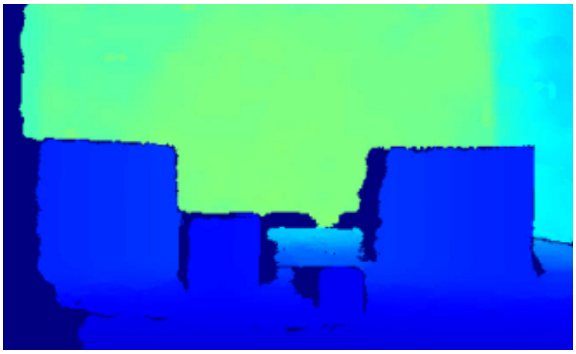


Figura 6: Imagen de profundidad (D) utilizada para determinar las dimensiones de objetos usando el Método 1 (ver la Figura 2) y considerando el Arreglo experimental 1 (ver la Figura 4). Los pixeles más cercanos se muestran en color azul y los más lejanos en verde.

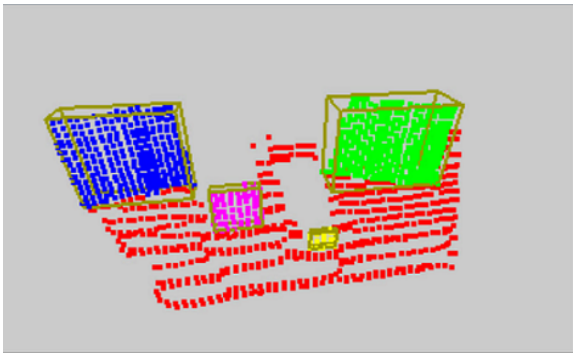


Figura 7: Objetos detectados en la nube de puntos usando el Método 1. Los clusters se muestran en diferentes colores para diferenciarlos. Los puntos del suelo se muestran en rojo.

3.3. Detección de dimensiones usando información de profundidad y color

Las Figuras 8 y 9 muestran respectivamente una captura instantánea de los datos de entrada para el Método 2: la imagen de color (I_{RGB}) y el mapa de profundidad (D). La Figura 10 ilustra el proceso de fusión RGB-D, comparando los artefactos presentes en los datos brutos de profundidad (izquierda) con el resultado filtrado (derecha). Finalmente, la Figura 11 presenta los objetos detectados y etiquetados en el escenario dinámico del Arreglo 2, donde se aplicó seguimiento combinando información de color y profundidad con la cámara en movimiento.

Tabla 4: Resultados de mediciones (m) 3D con objetos de forma cilíndrica y cubica: precisión y métricas de error.

Obj.	Distancia			Ancho			Alto			Error relativo (%)			MAE
	R	M	E	R	M	E	R	M	E	Dist.	Ancho	Alto	
1	0.70	0.74	-3.60×10^{-2}	0.23	0.25	$\mathbf{-2 \times 10^{-2}}$	0.21	0.19	1.20×10^{-2}	5.10	8.90	5.90	2.30×10^{-2}
2	0.76	0.83	$\mathbf{-7.50 \times 10^{-2}}$	0.23	0.23	-6×10^{-3}	0.21	0.19	1.20×10^{-2}	9.90	2.70	5.90	3.10×10^{-2}
3	0.54	0.59	-1.20×10^{-2}	0.10	0.09	3×10^{-3}	0.10	0.03	$\mathbf{1.90 \times 10^{-2}}$	2.20	3	19	1.10×10^{-2}
4	0.48	0.49	-5.30×10^{-2}	0.05	0.05	-4×10^{-2}	0.05	0.08	1.80×10^{-2}	11	80	36	3.70×10^{-2}
									Promedio	7.10	23.40	16.70	2.5×10^{-2}

Notas: R = Valor real, M = Valor medido promedio, E = Error promedio (R - M), MAE = Error Absoluto Medio, del inglés *Mean Absolute Error* Valores destacados indican errores significativos (>15 % relativo o >0.05 m absoluto).

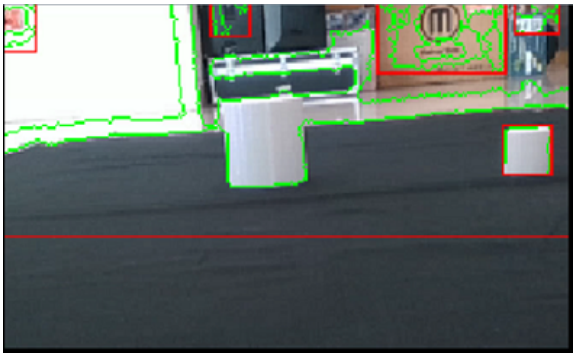


Figura 8: Imagen de color (I_{RGB}) utilizada por el Método 2 para segmentar objetos, ver la Figura 3.

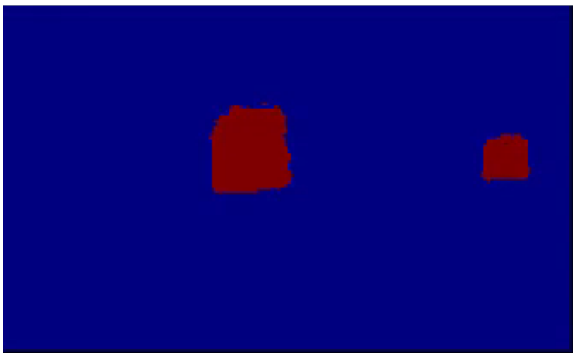


Figura 9: Imagen de profundidad segmentada (D_{o}) empleando información de color (I_{RGB}). El Método 2 separa la información de los objetos D_{o} del el suelo, ver la Figura 3.

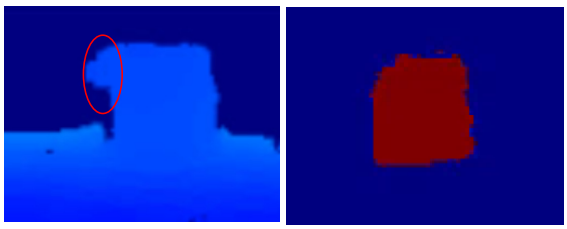


Figura 10: Efectos de la fusión de datos de profundidad (D) y color (I_{RGB}). Los artefactos en D (Izquierda) se han eliminando y se conservan solo los pixeles válidos (Derecha).

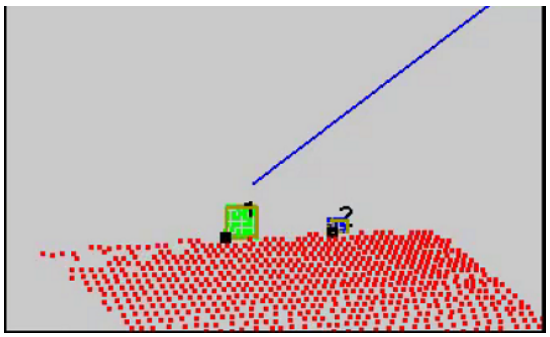
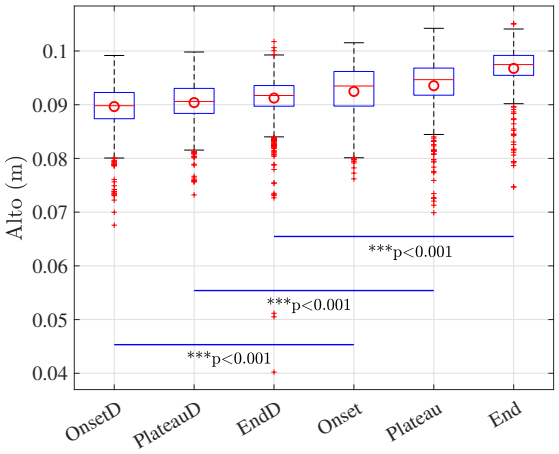
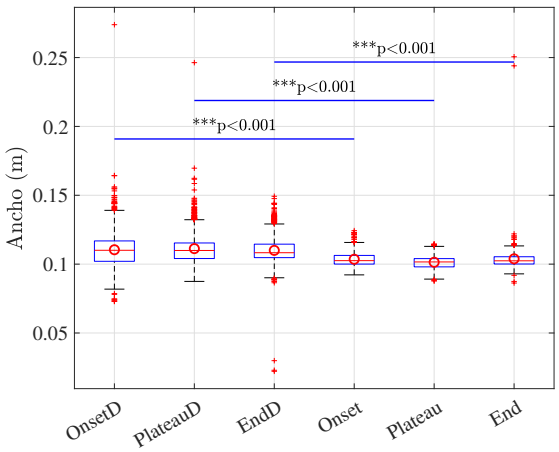


Figura 11: Objetos detectados con el Método 2 (ver la Figura 3) y cámara moviéndose a velocidad constante (ver la Figura 5).

En la Figura 12, se presentan las estadísticas asociadas a la detección de las dimensiones de uno de cilindro ($w = h = 0.10$ m) con la cámara en movimiento a velocidad constante $v_{\text{cam}} = 0.15$ m/s. Los datos experimentales se segmentaron en tres fases: *(i) Onset* (movimiento inicial a partir de $d(t_0) \approx 1$ m), *(ii) Plateau* (movimiento uniforme a mitad de trayectoria), y *(iii) End* (movimiento final hasta $d(t_n) \approx 0.40$ m). La Figura 12 compara los resultados del Método 1 (izquierda) y Método 2 (derecha) para ambas dimensiones: ancho (gráfico superior) y alto (gráfico inferior). En términos cuantitativos, el error absoluto medio (MAE) para la estimación del alto fue de 9.60×10^{-3} m con el Método 1 y 6×10^{-3} m con el Método 2; para la ancho, el MAE fue de 1.17×10^{-2} m y 4.50×10^{-3} m, respectivamente.



Algoritmo: (Izquierda) Método 1, (Derecha) Método 2



Algoritmo: (Izquierda) Método 1, (Derecha) Método 2

Figura 12: Estadísticas sobre la detección del ancho, $w = 0.10$ m, (arriba) y alto, $h = 0.10$ m, (abajo) de un cilindro con la cámara en movimiento a velocidad constante $v_{\text{cam}} = 0.15$ m/s. Se han dividido los datos experimentales en tres secciones *Onset* (con la cámara empezando a moverse desde el punto de partida a una distancia $d(t_0) \approx 1$ m), *Plateu* (cámara a mitad del recorrido), y *End* (último tramo del recorrido hasta alcanzar una mínima distancia $d(t_n) \approx 0.40$ m) del experimento. Se comparan los Métodos 1 (izquierda) y 2 (derecha). Los errores absolutos medios (MAE) fueron, para el Método 1: 9.60×10^{-3} m (alto) y 1.17×10^{-2} m (ancho); y para el Método 2: 6×10^{-3} m (alto) y 4.50×10^{-3} m (ancho).

4. Discusión

Para aplicaciones de robótica móvil con tareas de navegación, es necesario contar con sensores que proporcionen información del entorno. En este trabajo nos hemos enfocado en abordar la detección de las dimensiones (ancho y alto) y distancia de objetos 3D usando una cámara RGB-D. Hemos abordado dos métodos: El primero (Método 1) consiste en utilizar unicamente datos de profundidad para la obtención de mediciones y el segundo (Método 2) combina datos de profundidad y de color. El Método 1 es más simple y requiere menos operaciones y recursos de hardware, ya que sólo utiliza la imagen de profundidad (D) para estimar las mediciones. El Método 2 es computacionalmente más exigente, pues requiere adquirir de manera simultánea, la información de profundidad y color (I_{RGB}); pero es conceptualmente más robusto a aberraciones ópticas, al fusionar información de color y profundidad. Ambos métodos realizan detección de objetos por CE, la cual agrupa puntos espacialmente relacionados pero no los etiqueta; por ello, para propósitos de navegación se propone un algoritmo de seguimiento de objetos 3D basado en la técnica de NNA, similar al propuesto por (Bochinski *et al.*, 2017) y que se ha adaptado para funcionar con nubes de puntos asociados a objetos individuales. La técnica propuesta considera que hay pocos traslapes.

Los métodos han sido utilizados en dos arreglos experimentales, uno con objetos y cámara fijos (Arreglo experimental 1) y otro con cámara en movimiento y objetos estáticos (Arreglo experimental 2). El Arreglo experimental 1 permitió evaluar que para la adquisición, la resolución 156×144 ofrece el mayor rendimiento (180 % más rápido que 1280×720 (Full HD)), pero reduce el campo visual efectivo a solo 25°×16° según (IntelCorporation, 2023b). Por lo tanto, la resolución de 480×270 es la óptima al mantener el 100 % del campo visual original (69°×42°) con un rendimiento 83 % más rápido que Full HD, ver Tabla 2. Se observó que se puede mejorar el rendimiento sacrificando precisión al utilizar técnicas de diezmado con un factor de [4 - 8] y tamaño de voxel (1.50×10^{-3} m) logrando 18-22 FPS, ver Tabla 3. También se determinó que, dado que el algoritmo propuesto mide las dimensiones en función de los puntos ubicados en los extremos mínimos y máximos, la perspectiva del objeto puede generar errores en el algoritmo y entregar medidas muy grandes. Los errores más grandes en la estimación del ancho y la distancia, se presentaron al evaluar cajas (MAE= 7.50×10^{-2}m para la distancia y MAE= 0.02 m para el ancho), ver Tabla 4. Con base en estos resultados, para la evaluación de la precisión de los algoritmos con cámara en movimiento, es mejor usar cilindros, pues estos no tienen el problema de la perspectiva.

El Arreglo experimental 2, con cámara en movimiento, permitió validar los métodos 1 (sólo profundidad) y 2 (fusión de color y profundidad). Los datos revelan que el Método 1 no presenta cambios considerables en la precisión al estimar las dimensiones conforme cambia la distancia y siempre presentan un offset constante (MAE = 9.60×10^{-3} m para la alto y MAE = 1.17×10^{-2} m para el ancho). Mientras que se aprecia una mejora significativa del Método 2 respecto al Método 1 en la estimación de las dimensiones (valor de significancia $p < 10^{-3}$ y MAE = 6×10^{-3} m para la alto y MAE = 4.50×10^{-3} m para el ancho); Además, la estimación del alto mejora conforme la cámara se acerca al objeto, ver Figura 12. Esta mejora se asocia a que el Método 2 logra reducir artefactos en la imagen de profundidad mediante la fusión de información de color, ver Figura 10.

5. Conclusiones

Este trabajo desarrolla un sistema para la estimación de dimensiones y seguimiento de objetos en robótica móvil mediante cámaras RGB-D, usando dos enfoques. Uno basado en profundidad, que ofrece eficiencia computacional (18-22 FPS, MAE $< 1.17 \times 10^{-2}$ m) ideal para sistemas embebidos; y otro que fusiona datos de profundidad y color, que mejora significativamente ($p < 10^{-3}$) la precisión (MAE $< 6 \times 10^{-3}$ m) al corregir artefactos ópticos. Si bien el Método 2 requiere conceptualmente más recursos, no se observó una diferencia notable. Estos resultados son alentadores y, aunque solo se ha probado con cajas y cilindros, podría ampliarse a estructuras más complejas mediante la integración con SLAM visual-inercial, adaptación dinámica de parámetros de filtrado y resolución; además de aprendizaje profundo, combinando así eficiencia y precisión según necesidades de aplicación.

Referencias

Bochinski, E., Eiselein, V., y Sikora, T. (2017). High-speed tracking-by-detection without using image information. En *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pp. 1–6.

Fischler, M. A. y Bolles, R. C. (1981). Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395.

IntelCorporation (2023a). Depth camera calibration tools. Accessed: 23-06-2025.

IntelCorporation (2023b). Intel realsense d400 series datasheet. Accessed: 23-06-2025.

IntelCorporation (2024). Post-processing filters. Accessed: 23-06-2025.

Lang, A. H., Vora, S., Caesar, H., Zhou, L., Yang, J., y Beijbom, O. (2019). Pointpillars: Fast encoders for object detection from point clouds.

Lin, Y., Zhang, Z., Tang, H., Wang, H., y Han, S. (2021). Pointacc efficient point cloud accelerator. En *MICRO54 54th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 449–461. ACM.

Mao, J., Shi, S., y et al., X. W. (2023). 3d object detection for autonomous driving: A comprehensive survey. *International Journal of Computer Vision*, 131:1909–1963.

Park, J., Xu, C., Yang, S., Keutzer, K., Kitani, K., Tomizuka, M., y Zhan, W. (2022). Time will tell: New outlooks and a baseline for temporal multi-view 3d object detection.

PCL (2023). Point cloud library (pcl): Sacsegmentation class. Accessed: 23-06-2025.

RealSense™, I. (2025a). Depth image compression by colorization for intel® realsense™ depth cameras.

RealSense™, I. (2025b). Pcl (point cloud library).

Rusu, R. B. y Cousins, S. (2011). 3D is here: Point Cloud Library (PCL). En *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China.

Wang, C., Wang, C., Li, W., y Wang, H. (2021). A brief survey on rgb-d semantic segmentation using deep learning. *Displays*, 70:102080.

Zhang, Z. (2000). A flexible new technique for camera calibration. *IEEE TPAMI*, 22(11).

Zhou, J. (2022). A review of lidar sensor technologies for perception in automated driving. *Academic Journal of Science and Technology*, 3:255–261.

Zhou, L., Wu, G., Zuo, Y., Chen, X., y Hu, H. (2024). A comprehensive review of vision-based 3d reconstruction methods. *Sensors*, 24(7).