

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE ELECTRÓNICA
VALPARAÍSO - CHILE



**"EVALUACIÓN DE MODELO DE SEGUIMIENTO DE OBJETOS EN ROBOT iCub
MEDIANTE CREACIÓN DE GROUND TRUTH CON COMPUTER VISION"**

DIEGO EDUARDO SEPÚLVEDA SILVA

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL ELECTRÓNICO

PROFESOR GUÍA : DRA. MARÍA-JOSÉ ESCOBAR
PROFESOR CORREFERENTE : DR. MAURICIO ARAYA

Marzo 2024

AGRADECIMIENTOS

En primera instancia, quiero agradecer a mi papá, mi mamá y mi hermano por haberme apoyado en estos años y creer en mis capacidades para sobrevivir en otra ciudad y apoyarme a la distancia en cada una de las locuras de esta experiencia.

También quiero agradecer a los amigos que he hecho a lo largo de mi paso por esta universidad. En primera lugar, agradecer a Debate UTFSM, que más que debate, siempre fue un círculo de apoyo y contención. No puedo dejar fuera a cada persona que conocí en el Coro UTFSM y del Coro Comunitario, espacio que me hizo reencontrarme una vez más con mi amor por la música y donde pude conocer grandes músicos y amistades maravillosas.

Quiero agradecer a la profesora Maria José Escobar y al profesor Mauricio Araya por permitirme explorar un poco de lo que se trata la investigación y la robótica y confiar en mis capacidades para lograr este proyecto. Sin duda, espero poder seguir desarrollando conocimiento en estas áreas.

Quiero agradecer personalmente a Mireya, Betsabé, Ignacio, Sebastián, Paula, Brit, Sofía, Laau y Mati, las risas definitivamente no faltaron.

Finalmente, quiero agradecer a Mati y Brit por enseñarme lo que se puede lograr con el trabajo en equipo, con la buena comunicación y el gran cariño que me han aportado. Las juntas para llorar valieron la pena.

RESUMEN



En el contexto de la robótica, la simulación y evaluación de algoritmos y modelos complejos es una tarea necesaria que permite obtener resultados de rendimiento dada una tarea específica a cumplir. Este proyecto se ha enfocado principalmente al desarrollo de una plataforma de evaluación que cuente con las herramientas necesarias para la toma de datos de estudio sobre tareas de detección de objetos implementadas en el robot humanoide iCub. El diseño de la plataforma planteada se decide en busca de una solución de bajo costo, por lo que se utiliza una única cámara web para la toma de datos. Esta cuenta con un módulo de estimación de coordenadas que funciona como generador de información de confianza o Ground Truth frente a la posición de un objeto observado, desarrollado utilizando 3 soluciones para el seguimiento, detección mediante segmentación de color, a través de su representación en formato HSV. Los otros 2 algoritmos se han desarrollado a partir de la arquitectura de YOLOv8 para las tareas de detección y para segmentación, debido a las características de funcionamiento que permiten un mejor rendimiento en tiempo real. Posterior al desarrollo de algoritmos de estimación se realiza la integración de herramientas de ROS y Gazebo para diseñar la simulación final que permite la comunicación entre el sensor que simula la cámara web, la mesa de trabajo y el objeto a observar, que corresponde a una pelota roja, así como también la presencia del robot iCub. Esta simulación se evalúa a través de pruebas enfocadas en la detección de posición de un objeto en movimiento, por lo que se le designa una trayectoria a la pelota. Los análisis de resultados indican que existe un porcentaje de error bajo para los modelos basados en segmentación del objeto, indicando que la cámara permite la extracción de información suficiente para la estimación de posición dentro del campo visible. Dadas estas conclusiones, sumado a la simulación de un experimento comparando los datos obtenidos por el iCub con los de la cámara, es posible visualizar el desempeño del robot con dicha información.

Palabras clave

Aprendizaje profundo, visión por computadora, modelo pinhole, Gazebo, YOLO, detección de objetos.

ABSTRACT



In the field of robotics, the simulation and evaluation of complex algorithms and models is a necessary task that allows obtaining performance results given a specific task to accomplish. This project has focused primarily on the development of an evaluation platform equipped with the necessary tools for data collection on object detection tasks implemented in the iCub humanoid robot. The design of the proposed platform is aimed at a low-cost solution, hence a single webcam is used for data collection. It features a coordinate estimation module that serves as a confidence information generator or Ground Truth regarding the position of an observed object, developed using three solutions for tracking: the first one using color segmentation detection, represented in HSV format. The other two algorithms have been developed based on the YOLOv8 architecture for detection and segmentation tasks, due to their operational characteristics allowing better real-time performance. Following the development of estimation algorithms, the integration of ROS and Gazebo tools is carried out to design the final simulation enabling communication between the webcam-simulated sensor, the workspace, the observed object, and the presence of the iCub robot. This simulation is evaluated through tests focused on detecting the position of a moving object, for which a trajectory is assigned to the ball. Result analyses indicate a low error percentage for object segmentation-based models, suggesting that the camera allows sufficient information extraction for position estimation within the visible field. With these conclusions, coupled with the simulation of an experiment comparing data obtained by the iCub with those from the camera, it is possible to visualize the robot's performance with such information.

Keywords

Deep Learning, Computer Vision, pinhole model, Gazebo, YOLO, object detection.

GLOSARIO

- 
- **YARP:** Yet Another Robot Platform
 - **ROS:** Robot Operating System
 - **YOLO:** You Only Look Once
 - **Ground Truth:** Corresponde a la información confiable y validada, que describe la naturaleza del problema de aprendizaje que se quiere describir.
 - **Modelo anchor free:** Asociado a un modelo de detección de objetos que aprende directamente a predecir la posición y dimensión de un objeto en la imagen analizada.
 - **segColor:** Referencia a algoritmo de segmentación de color.
 - **detYOLO:** Referencia a algoritmo de detección y clasificación YOLO.
 - **segYOLO:** Referencia a algoritmo de segmentación de YOLO.

Índice de Contenidos

1. Introducción	1
1.1. Objetivos del trabajo	1
1.2. Trabajo a desarrollar y resultados esperados	2
2. Estado del arte	3
2.1. Visión por computadora	3
2.1.1. Proyección de coordenadas	3
2.1.2. Detección de objetos	4
2.2. Simulación de iCub	7
3. Estimación de posición 3D	8
3.1. Detección de objetos en tiempo real	9
3.1.1. Detección con arquitectura YOLO	9
3.1.1.1. Detección y bounding box	9
3.1.1.2. Segmentación de objetos	11
3.1.2. Detección basado en segmentación de color	13
3.2. Estimación de profundidad	15
3.3. Coordenadas en el espacio	17
4. Entorno de simulación	20
4.1. Simulación mediante Gazebo	20
4.1.1. Interacción con el mundo simulado	20
4.1.2. Construcción de simulación modular	22
4.2. Integración con ROS	24
4.2.1. Creación archivo de lanzamiento	25
5. Integración de modelo de estimación 3D a plataforma de simulación	27
5.1. Transformación de coordenadas estimadas	28
5.2. Parámetros de visión para simulación	29
5.2.1. Cámara Gazebo sim	29
5.2.2. Visión iCub	29
6. Simulaciones y resultados	30
6.1. Validación de estimación de coordenadas	30
6.1.1. Métricas de evaluación	31
6.1.2. Trayectoria sin variación de profundidad	31
6.1.3. Trayectoria con variación de profundidad	38
6.2. Evaluación tarea detección en robot iCub	45
7. Conclusiones y trabajo futuro	54
7.1. Comparación de resultados y conclusiones	54
7.2. Trabajo futuro	55

Índice de Tablas

3.1. Especificaciones cámara web utilizada para estimación de posición.	8
3.2. Especificaciones PC utilizado para el entrenamiento de modelos	8
3.3. Métricas de rendimiento para detección de objetos.	11
3.4. Métricas de rendimiento para bounding box y detección de máscara.	12
6.1. Métricas obtenidas en el experimento	38
6.2. Métricas obtenidas en el experimento	45
6.3. Métricas obtenidas en el experimento	45

Índice de Figuras

2.1. Modelo de cámara pinhole[1].	3
2.2. Arquitectura YOLOv8[4].	5
2.3. Algoritmo de Non-Maximum Suppression[5]. Establece el pseudo código con los pasos que se llevan a cabo para su implementación.	6
2.4. Capa Fully Convolutional Network[6].	6
2.5. Comparación entre segmentación semántica y segmentación de instancia[7].	7
3.1. La figura 3.1a corresponde al objeto a detectar, mientras que la figura 3.1b corresponde a la cámara a utilizar.	8
3.2. La figura 3.2a muestra el formato de anotación de Robloflow, mientras que la figura 3.2b muestra el dataset anotado[12].	10
3.3. Entrenamiento modelo de detección.	10
3.4. Resultados de entrenamiento y validación para detección de objetos	10
3.5. Detección en tiempo real	11
3.6. La figura 3.6a muestra el formato de anotación de Robloflow, mientras que la figura 3.6b muestra el dataset anotado[13].	11
3.7. Entrenamiento modelo de segmentación.	12
3.8. Resultados entrenamiento para segmentación de objetos	12
3.9. Segmentación y máscara en tiempo real	12
3.10. La figura 3.10a[14] muestra la variación de colores en formato RGB, mientras que la figura 3.10b[15] muestra la variación de colores en formato HSV	13
3.11. Diagrama de métodos CVZone para obtener los rangos HSV y encontrar la máscara necesaria, controlado por flag booleana.	13
3.12. Pestaña de CVZone que ilustra el proceso de obtención de rangos HSV.	14
3.13. Detección y segmentación de color HSV.	14
3.14. Relación distancia focal	15
3.15. La figura 3.15a representa el código para estimar la longitud focal y la figura 3.15b la estimación de distancia[17].	16
3.16. Calibración para obtener parámetros intrínsecos de webcam. Fueron utilizadas 40 imágenes para este procedimiento.	18
4.1. Modelo iCub v2.5 <i>visuomanip</i>	20
4.2. La figura 4.2a corresponde al modelo básico de cámara existente en Gazebo, y la figura 4.2b representa la mesa de trabajo utilizada en proyectos asociados al iCub.	21
4.3. Visualización imagen desde ojo derecho de robot iCub	21
4.4. Descripción de archivo platform.world	22
4.5. Descripción de la visualización de objeto red-ball en archivo <i>.sdf</i>	23
4.6. Descripción de las coordenadas y el instante en que se debe posicionar el objeto <i>red-ball</i> . La trayectoria considera 5 puntos de movimiento y el objeto elabora una trayectoria basada en esos puntos.	24
4.7. Plataforma de evaluación construida modularmente	24
4.8. Descripción de archivo icub_platform.launch	25
4.9. Conectividad entre plataformas.	25

4.10. Imagen capturada por cámara en Gazebo visualizada en rviz	26
5.1. Comunicación entre módulos de estimación de coordenadas y simulaciones del robot iCub y cámara al obtener el frame de entrada del objeto.	27
6.1. Obtención de posición real de modelo <i>red-ball</i>	30
6.2. Las figuras 6.2a y 6.2b corresponden a la trayectoria real y estimada del objeto utilizando la técnica de detección por segmentación de color	32
6.3. Las figuras 6.3a y 6.3b representan corresponden a la trayectoria real y estimada del objeto utilizando la técnica detección y clasificación con YOLO	33
6.4. Las figuras 6.4a y 6.4b representan corresponden a la trayectoria real y estimada del objeto utilizando la técnica de segmentación semántica y detección con YOLO	34
6.5. Comparación de estimación de trayectoria para cada técnica de reconstrucción, en torno al eje x	35
6.6. Comparación de estimación de trayectoria para cada técnica de reconstrucción, en torno al eje y	36
6.7. Comparación de estimación de trayectoria para cada técnica de reconstrucción, en torno al eje z	37
6.8. Trayectoria establecida para variación de profundidad.	38
6.9. Las figuras 6.9a y 6.9b corresponden a la trayectoria real y estimada del objeto utilizando la técnica de detección por segmentación de color	39
6.10. Las figuras 6.10a y 6.10b corresponden a la trayectoria real y estimada del objeto utilizando la técnica de detección y clasificación utilizando YOLO	40
6.11. Las figuras 6.11a y 6.11b corresponden a la trayectoria real y estimada del objeto utilizando la técnica de segmentación utilizando YOLO	41
6.12. Comparación de estimación de trayectoria para cada técnica de reconstrucción, en torno al eje x	42
6.13. Comparación de estimación de trayectoria para cada técnica de reconstrucción, en torno al eje y	43
6.14. Comparación de estimación de trayectoria para cada técnica de reconstrucción, en torno al eje z	44
6.15. Trayectoria visualizada por iCub con segmentación de color , y cámara.	46
6.16. Trayectoria visualizada por iCub con detección y clasificación de YOLO , y cámara.	47
6.17. Trayectoria visualizada por iCub con segmentación de instancia YOLO , y cámara	48
6.18. Histogramas de residuo en coordenadas x.	50
6.19. Histograma de residuo en coordenadas y.	51
6.20. Histograma de residuo en coordenadas z.	52

1 | Introducción



El estudio de algoritmos en el ámbito de la robótica ha ido evolucionando en conjunto con la complejidad de las herramientas tecnológicas que se tienen a disposición. Dentro de estas herramientas se encuentra el *iCub*, robot humanoide desarrollado por el Instituto Italiano de Tecnología, para la investigación de algoritmos de aprendizaje profundo, ideal para establecer un campo de investigación en torno a la interacción con el entorno, a través de sus sensores y actuadores, emulando sentidos humanos.

En la implementación de tecnologías con este nivel de complejidad se hace presente la necesidad de disminuir la posibilidad de fallo en el desarrollo de aplicaciones basados en modelos sofisticados, por lo que se hace presente el uso de herramientas de simulación que emulen el contexto real al cual se enfrentará el robot. Es por ello que se propone la elaboración de una plataforma de simulación, en conjunto con la incorporación de herramientas de Computer Vision, para la evaluación del robot iCub en la tarea de detección de un objeto en movimiento, para el cual se incorpora un punto de vista utilizando una webcam, como propuesta de bajo costo económico, el que se comportará como Ground Truth para algoritmos implementados en el iCub.

1.1. Objetivos del trabajo

El trabajo a desarrollar se divide en 3 objetivos generales que describen las grandes etapas a superar.

1) Detección de objetos y extracción de posición 3D:

Se lleva una fase de evaluación sobre las diferentes técnicas de Computer Vision y geometría proyectiva para la creación del Ground Truth utilizando información obtenida de la imagen de la webcam.

2) Estructuración plataforma de simulación:

Exploración de de herramientas computacionales para simulación del iCub, que permita la integración de algoritmos externos y sincronización de procesos para la toma de datos.

3) Integración de herramientas para evaluación:

Fase de edición de algoritmos de seguimiento y de simulación para el correcto funcionamiento. Se establecen las condiciones necesarias para el proceso de evaluación, generando datos que permitan demostrar la posibilidad de evaluación del iCub con esta herramienta.

1.2. Trabajo a desarrollar y resultados esperados

Se parte con la definición del objeto a detectar, en el que se utilizará una pelota roja. A partir de ello, se estudia las herramientas de computer vision en torno a la detección de objetos, donde se toma 2 enfoques. El primero utilizando una detección basada en aprendizaje profundo, mientras que el segundo enfoque busca una implementación basada en el procesamiento de imágenes y detección de color. El primer enfoque requiere la creación de un Dataset del objeto a detectar, por lo que en esta fase se incorporan herramientas como Roboflow, con el que se lleva a cabo las tareas de etiquetado. Además, se explora la aumentación de datos utilizando la misma plataforma web como una solución que entregue una mayor cantidad de datos de entrenamiento. Para el segundo enfoque, se exploran los métodos que ofrece la librería de OpenCV en torno a la segmentación utilizando procesamiento de imágenes y dibujando un bounding box al objeto en función del valor HSV detectado, el cual describe el color en términos de matiz, saturación y valor.

La siguiente fase de trabajo comienza con la exploración de las herramientas actuales de simulación. Para ello, se estudian las facilidades y dificultades en el uso del simulador Gazebo, el cual es utilizado comúnmente en la implementación de modelos y aplicaciones del robot iCub y que interactúa de buena manera con códigos escritos en C++. Por otro lado, se evalúa la integración de las dependencias de ROS, buscando una forma sencilla de interacción utilizando códigos en lenguaje Python con el entorno simulado. Para esta fase, se desarrollan archivos en formatos .sdf y .world que describen los modelos de cada objeto puesto en simulación, las características que tendrá el entorno, como la posición de la luz, la gravedad y el punto de vista del observador al iniciar simulación. Además de esto, se incorpora un archivo de inicialización .launch, que incorpora a la simulación las dependencias de ROS, incluyendo el servidor de ROS.

En la fase final, se incorporan una fase de modificación a las herramientas desarrolladas de Computer Vision, enfocado en calibrarlas para su correcto funcionamiento dentro de la simulación. En este proceso es importante desarrollar una evaluación de concepto, en que se estudia el error de precisión en la tarea de detectar la posición en 3 dimensiones utilizando una sola cámara. Luego de poder validar el Ground Truth desarrollado, se hace una evaluación al robot iCub en la tarea de detección del objeto. Para ello, se utiliza la información obtenida por este modelo, enfocando la evaluación en determinar el desempeño al predecir la posición del objeto, mediante el proceso de determinar la distancia a la que se encuentra. La validación de detección del iCub se definirá por el error de estimación.

Al finalizar este proyecto, se espera lograr una solución a la estimación de profundidad utilizando una sola cámara, permitiendo una detección confiable de la posición de los objetos a detectar. En este mismo contexto, se espera lograr un algoritmo de fácil modificación que permita trabajar con otro tipo de objetos.

En el contexto de la simulación, se espera obtener una plataforma que se adecúe a las necesidades que presenten estudios de tareas más complejas en la interacción con objetos, teniendo acceso a variadas herramientas de simulación que ofrece ROS para elaborar un entorno robusto, controlado y reproducible.

Finalmente, se espera que este enfoque entregue una ayuda a los estudios algoritmos de aprendizaje profundo al entregar una herramienta de validación para las tareas de interacción con objetos. Por otro lado, se espera que esta plataforma sea reproducible en la realidad, dando así una herramienta de evaluación en tiempo real, de bajo costo económico y simple de construir.

2 | Estado del arte

2.1. Visión por computadora

La Visión por computadora, o **Computer Vision** es un área de la Inteligencia Artificial centrada en desarrollar métodos y técnicas que permitan a los computadores extraer, analizar e interpretar información de imágenes y videos, siendo capaces de realizar tareas asociadas a los patrones y características encontradas, siendo implementado ampliamente en tareas de detección, clasificación y segmentación.

En este contexto, es posible encontrar el *Ground Truth* en tareas como la detección de objetos, donde el término se aplica principalmente a las coordenadas en las que se puede visualizar un objeto, particularmente representado a través de los bounding boxes o cajas delimitadoras, que indican su posición.

2.1.1. Proyección de coordenadas

La estimación de coordenadas en el espacio a través de la detección de objetos es una tarea enfocada en utilizar relaciones matemáticas entre la información de la posición en píxeles de un objeto con la posición real de este mismo. Para establecer dicha relación se hace uso del modelo más simple de cámara, el modelo pinhole.

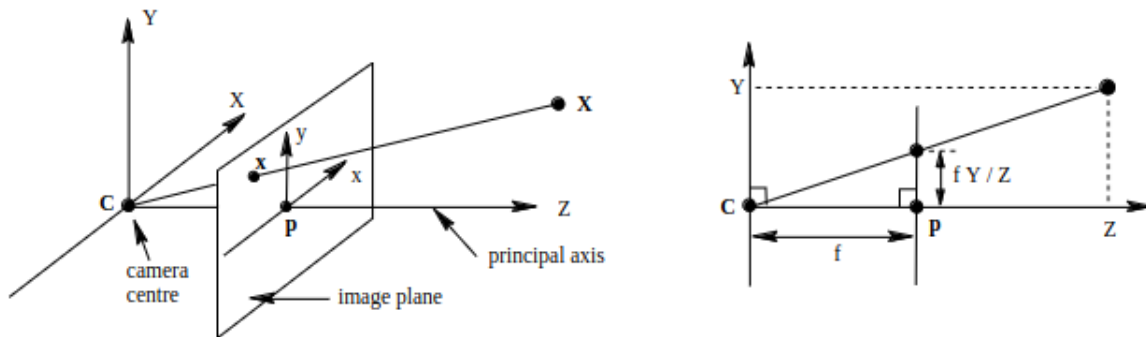


Figura 2.1: Modelo de cámara pinhole[1].

El modelo, visualizado en la figura 2.1, ubica el centro del espacio Euclidean en la posición de la cámara o centro de proyección C . Por otro lado, el plano de la imagen se establece con $z = f$, donde este valor es considerado la longitud focal. Bajo este modelo, es posible establecer un mapeo entre coordenadas euclidianas en $\mathbb{R}^3$ y $\mathbb{R}^2$

$$\mathbf{E} : \mathbb{R}^3 \rightarrow \mathbb{R}^2 \quad (2.1)$$

$$(x, y, z) \rightarrow \left(\frac{x}{z}f, \frac{y}{z}f \right) \quad (2.2)$$

Debido a la división que se genera por la coordenada en z , la proyección resulta ser no lineal, para lo cual se realiza una transformación a las coordenadas euclidianas, buscando una representación a través de coordenadas homogéneas con la construcción de una matriz de proyección, dando como resultado un mapeo lineal.

$$E : \mathbb{P}^3 \rightarrow \mathbb{P}^2 \quad (2.3)$$

$$P' = \begin{pmatrix} \mathbf{f}x \\ \mathbf{f}y \\ z \end{pmatrix} = \begin{pmatrix} \mathbf{f} & 0 & 0 & 0 \\ 0 & \mathbf{f} & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \quad (2.4)$$

Por convención, y debido a la forma en como se suelen trabajar las imágenes, se suele caracterizar la coordenada $(0,0)$ en la esquina superior izquierda y no en el centro de la imagen. La representación de esta convención afecta directamente en la ecuación 2.2, en donde se agrega un offset que se representa por la siguiente relación:

$$(x, y, z) \rightarrow \left(\frac{x}{z}\mathbf{f} + c_x, \frac{y}{z}\mathbf{f} + c_y \right) \quad (2.5)$$

El mapeo lineal mostrado en la ecuación 2.4 asume que la imagen obtenida será siempre cuadrada. Es debido a esto que para un modelo generalizado que permita adaptarse a diferentes escalas para cada eje, se define la longitud focal $\mathbf{f}_x$ y $\mathbf{f}_y$, representando la dimensionalidad en pixeles en torno al eje x e y .

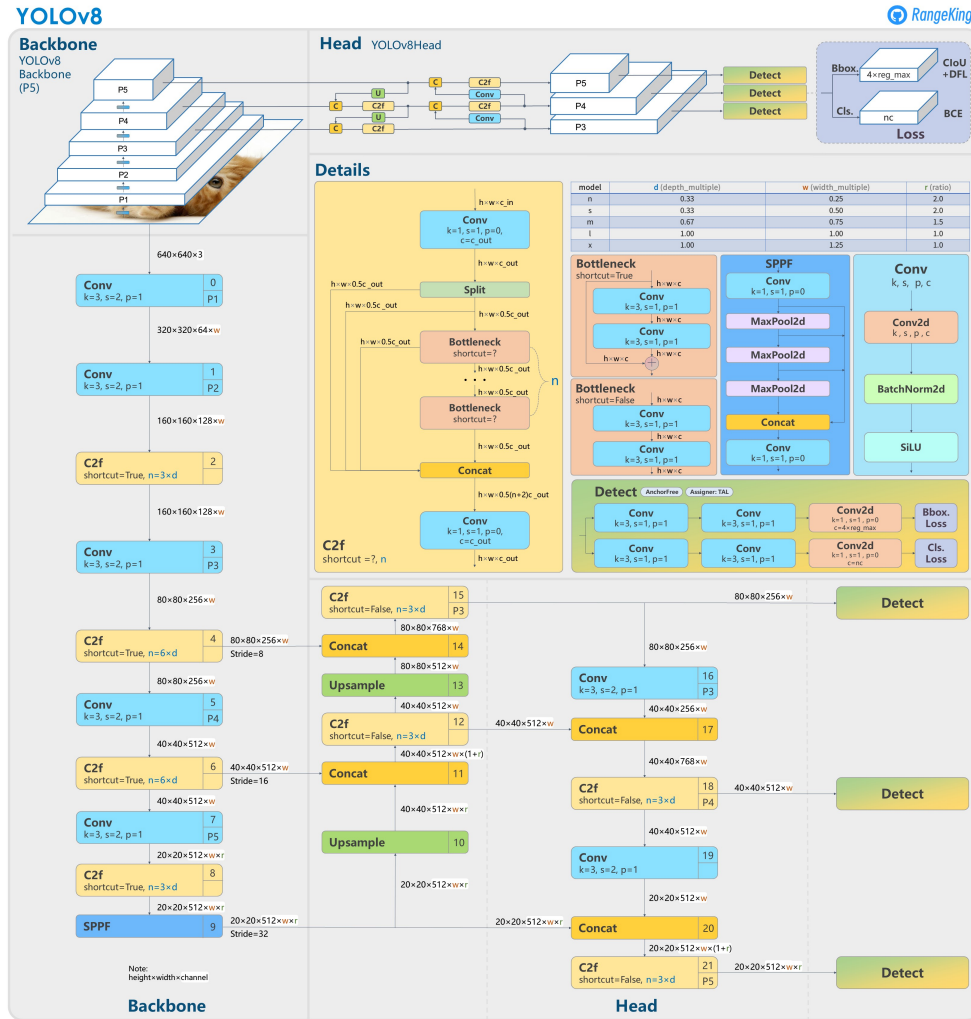
La relación final que permite la proyección con mapeo lineal se establece de la siguiente manera:

$$P' = \begin{pmatrix} \mathbf{f}_x x + z \cdot c_x \\ \mathbf{f}_y y + z \cdot c_y \\ z \end{pmatrix} = \begin{pmatrix} \mathbf{f}_x & 0 & c_x & 0 \\ 0 & \mathbf{f}_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \quad (2.6)$$

2.1.2. Detección de objetos

La detección de objetos en tiempo real consiste en una tarea que requiere una alta velocidad a la hora de extraer información sobre un frame de video. Esta consiste en establecer la localización del objeto, y en algoritmos más complejos, establecer una clasificación de este mismo, con la finalidad de obtener una salida de información más clara sobre lo que se está trabajando. Existen diversas herramientas desarrolladas a partir de la necesidad de establecer una detección refinada, basadas en el análisis de imagen en 2 etapas, una para la detección y otra para la clasificación. Sin embargo, en 2016 se presenta la primera versión de YOLO[2], un modelo basado en redes neuronales convolucionales que, a través de una sola etapa de procesamiento, era capaz de hacer las dos tareas, disminuyendo el costo computacional requerido y permitiendo un mayor desarrollo con enfoque en implementaciones en tiempo real, donde la velocidad de detección es crucial. YOLO divide la imagen en regiones y realiza predicciones simultáneas de cuadros de identificación y probabilidades para cada región, donde luego son ponderadas por cada probabilidad predicha.

Este modelo ha evolucionado con los años, estableciendo en 2023 por la empresa Ultralytics, la versión de YOLOv8[3], siendo una optimización de sus versiones anteriores.



Algorithm 1 Non-Max Suppression

```

1: procedure NMS( $B, c$ )
2:    $B_{nms} \leftarrow \emptyset$  Initialize empty set
3:   for  $b_i \in B$  do  $\Rightarrow$  Iterate over all the boxes
4:      $discard \leftarrow \text{False}$  Take boolean variable and set it as false. This variable indicates whether b(i) should be kept or discarded
5:     for  $b_j \in B$  do Start another loop to compare with b(i)
6:       if  $\text{same}(b_i, b_j) > \lambda_{nms}$  then If both boxes having same IOU
7:         if  $\text{score}(c, b_j) > \text{score}(c, b_i)$  then
8:            $discard \leftarrow \text{True}$  Compare the scores. If score of b(i) is less than that of b(j), b(i) should be discarded, so set the flag to True.
9:         if not  $discard$  then Once b(i) is compared with all other boxes and still the discarded flag is False, then b(i) should be considered. So
10:           $B_{nms} \leftarrow B_{nms} \cup b_i$  add it to the final list.
11:   return  $B_{nms}$  Do the same procedure for remaining boxes and return the final list

```

Figura 2.3: Algoritmo de Non-Maximum Suppression[5]. Establece el pseudo código con los pasos que se llevan a cabo para su implementación.

La optimización en este proceso establece una mejora significativa en la detección en tiempo real, donde se disminuye el sesgo producto de los cuadros predefinidos, obteniendo una detección eficiente.

Adicionalmente a la arquitectura establecida en la figura 2.2, se ha elaborado una variación para realizar la tarea de segmentación, el modelo YOLOv8-seg. La segmentación semántica es un proceso en que se asocia un grupo de píxeles de la imagen de entrada con una clase existente, donde es posible extraer información de la forma del objeto. Esta variación resulta de la implementación de una capa *Fully Convolutional Network*, en el módulo *head*, que corresponde a una arquitectura con varias capas convolucionales, sumado a una capa de sampling, utilizada para obtener una máscara con la misma resolución que la imagen de entrada.

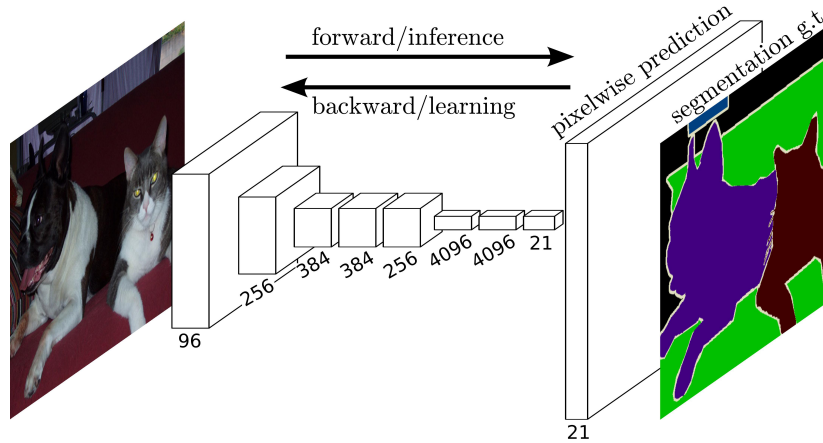


Figura 2.4: Capa Fully Convolutional Network[6].

El modelo YOLOv8-seg propone una variación denominada **segmentación de instancia**, que corresponde a una variación de la segmentación semántica. Esto implica asignar una clasificación a cada objeto segmentado, al mismo tiempo que se distingue cada instancia individualmente de los objetos. Esta técnica proporciona mayor información sobre la imagen procesada y las clases presentes.

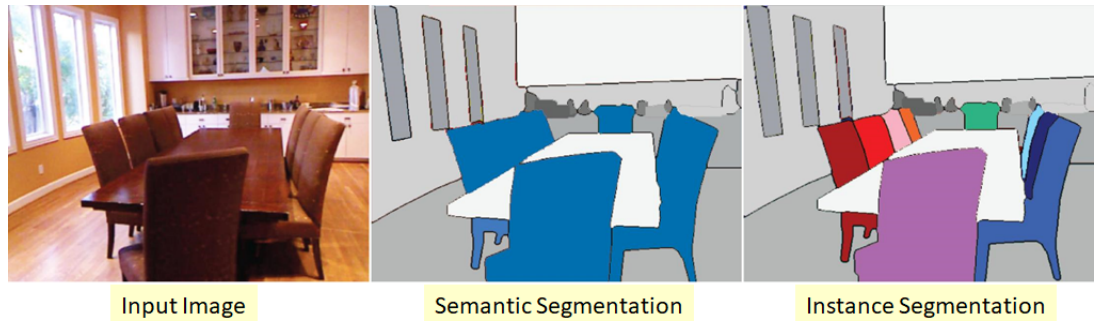


Figura 2.5: Comparación entre segmentación semántica y segmentación de instancia[7].

Las aplicaciones de esta variación van desde la extracción de información en datos como resonancias magnéticas, hasta la incorporación en el campo de la robótica en tareas de visión.

2.2. Simulación de iCub

El robot humanoide iCub[8] fue desarrollado por el Instituto Italiano de Tecnología (IIT) principalmente para el estudio y desarrollo de algoritmos de Inteligencia Artificial que le permiten aprender a interactuar con el entorno a través de técnicas de aprendizaje profundo. Utilizando sensores que emulan los sentidos humanos, como la visión o el tacto, se estudia la interacción de manera similar al ser humano. El iCub permite trabajar con distintos lenguajes de programación, así como la implementación de librerías externas para la investigación de nuevas implementaciones, mediante el uso de YARP[9] y las dependencias de robotology-superbuild[10].

YARP es un framework open-source utilizado para facilitar el desarrollo de aplicaciones para el control de robots. Establece conexiones con librerías y herramientas de simulación que permiten un trabajo modular y la integración de otras componentes. Las librerías de YARP para el iCub están inicialmente desarrolladas en C++, sin embargo, se cuenta con soporte para la traducción de códigos a lenguajes como Python o MATLAB, lo que facilita la integración de modelos de Machine Learning.

Las dependencias de YARP también permiten la simulación a través de herramientas como Gazebo[11], software open-source desarrollado para la simulación en el área de la robótica. Gazebo permite modelar los movimientos de las distintas versiones de modelos 3D disponibles para el iCub y controlar parámetros del entorno para obtener escenarios complejos. Además, ofrece la posibilidad de integrar ROS, que proporciona diversas herramientas para el desarrollo de aplicaciones en el ámbito de la robótica y permite conectar el entorno de Gazebo a modelos desarrollados en lenguajes como Python.

Los desarrollos en torno al iCub y a las tareas asociadas con la visión se han llevado a cabo mediante algoritmos complejos, como la segmentación para detectar la posición de un objeto, con el fin de establecer una trayectoria de movimiento. Esta tarea es relevante para los estudios de modelos, por lo que las etapas de análisis y comparativa de resultados son cruciales para la validación. Es por esto que se espera que el desarrollo de la plataforma propuesta por este trabajo resulte un aporte significativo en la evaluación del robot iCub para tareas en las que sea necesario rescatar información sobre la posición de los objetos.

3 | Estimación de posición 3D

En este capítulo se desarrolla la base teórica y las herramientas necesarias para la estimación de posición en 3 dimensiones de un objeto. Las herramientas para llevar a cabo este capítulo son 3 principales:

- Cámara para toma de datos en tiempo real.

Característica	Especificación
WEBCAM	Logitech C505
Resolución máxima	720p/30 fps
Campo visual diagonal (dFoV)	60°

Tabla 3.1: Especificaciones cámara web utilizada para estimación de posición.

- Computador utilizado para entrenamiento de modelos.

Característica	Especificación
Modelo GPU	NVIDIA GeForce RTX 3070, 8GB
Procesador	Intel Core i7
Memoria RAM	16GB
Sistema operativo	Ubuntu 20.04.6 LTS

Tabla 3.2: Especificaciones PC utilizado para el entrenamiento de modelos

- Objeto a detectar: Pelota roja de diámetro 6.2cm aproximadamente



(a)



(b)

Figura 3.1: La figura 3.1a corresponde al objeto a detectar, mientras que la figura 3.1b corresponde a la cámara a utilizar.

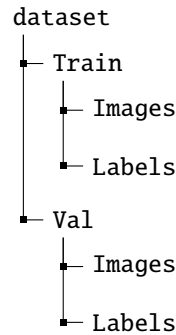
3.1. Detección de objetos en tiempo real

En el campo del Computer Vision, la detección de objetos permite el reconocimiento y/o clasificación utilizando una entrada de imagen o video, que permite el rastreo basado en algoritmos de procesamiento de imagen o utilizando aprendizaje profundo. En este ámbito, se toma 2 caminos paralelos para la detección.

3.1.1. Detección con arquitectura YOLO

Debido a las ventajas proporcionadas por la versión YOLOv8, que implica una prioridad significativa en la velocidad de detección, sin perder la cualidad de clasificación, a través de la extracción de información de entradas de imagen, se vuelve relevante su implementación para esta tarea, teniendo predicciones con alta confiabilidad, permitiendo la diferenciación de los distintos objetos que se pueden presentar en la mesa de trabajo.

Para poder recurrir a las herramientas que ofrece la arquitectura de YOLO, es necesario contar con los datos requeridos en la etapa de entrenamiento y validación a efectuar. La estructura de almacenamiento en la que estará el *dataset* es crucial, ya que se cuenta con un archivo *data.yaml*, que recopile la ruta hacia cada carpeta de información, así como también una lista con las clases presentes para entrenamiento. El formato del *dataset* debe estar regido bajo la siguiente estructura:



Dada las herramientas que ofrece YOLOv8, se elaboran 2 caminos para lograr el objetivo buscado mediante este modelo.

3.1.1.1. Detección y bounding box

Para elaborar un modelo de YOLO enfocado en la detección del objeto a través del bounding box, se elabora un *dataset* para el objeto trabajado, en este caso, la pelota roja. Para esta tarea se realiza una recopilación de datos a través del repositorio de libre uso disponible en la página web Roboflow. A esto, se le suman imágenes realizadas con la pelota roja disponible, que se presenta en la figura 3.1a. Con esto, se elabora un *dataset* de 240 imágenes. Las anotaciones e los datos creados manualmente se efectúan a través de la misma página web.

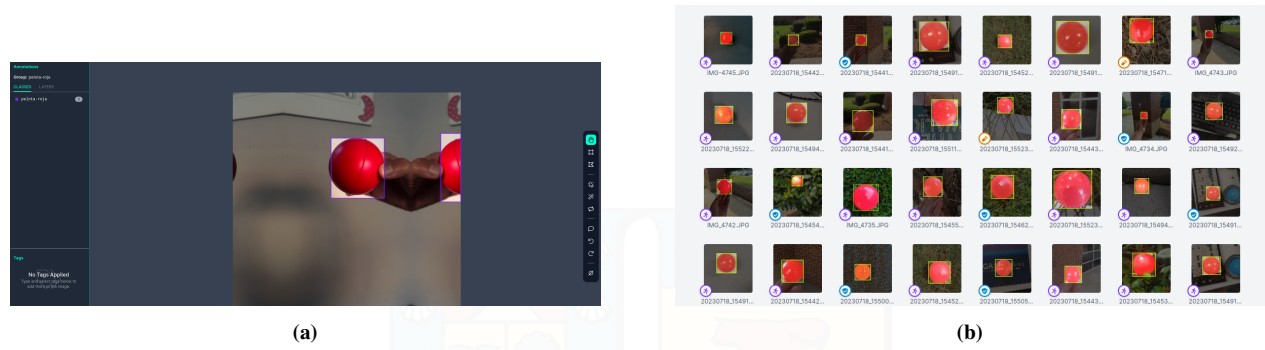


Figura 3.2: La figura 3.2a muestra el formato de anotación de Roboflow, mientras que la figura 3.2b muestra el dataset anotado[12].

El entrenamiento de este modelo se realiza con un *dataset* correspondiente a 88 % de datos para entrenamiento, 8 % para validación y 4 % para *testing*. Para la tarea de seguimiento y detección en tiempo real se ejecuta para 100 épocas, utilizando las siguientes condiciones:

```

1 model = YOLO('yolov8n.pt') #Modelo pre-entrenado de yolo.
2
3 result = model.train(data=dataPath, imgsz=640, epochs=100, device='cuda',
    batch=8, patience=0, task='detect', workers=8)

```

Figura 3.3: Entrenamiento modelo de detección.

Se hace uso de *transfer learnig*, proceso en el cual se incorpora un modelo pre-entrenado con el fin de disminuir los tiempos de entrenamiento con menos datos disponibles, teniendo los siguientes resultados:

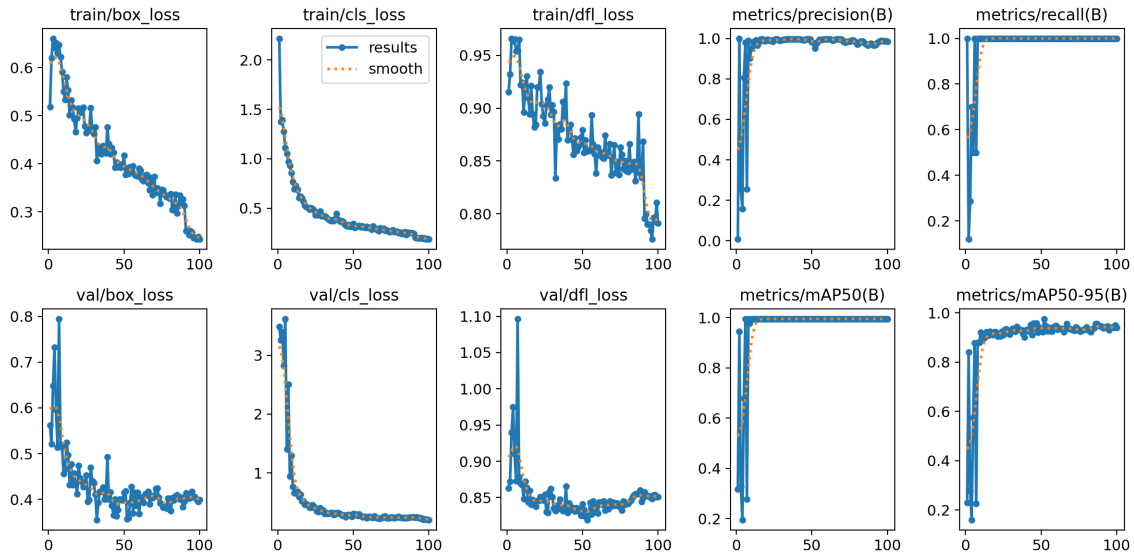


Figura 3.4: Resultados de **entrenamiento** y **validación** para detección de objetos

Bajo las métricas obtenidas durante esta fase, se puede visualizar una convergencia que presupone un correcto funcionamiento en el modelo trabajado. Para poder validar dicha suposición se tiene los resultados para los valores de *precisión media promedio (mAP)*, *precisión* y *recuperación*.

mAP50	0.995
mAP50-95	0.94023
precision	1
recall	0.995

Tabla 3.3: Métricas de rendimiento para detección de objetos.

Tomando en consideración las métricas mostradas en la tabla 3.3, enfocado principalmente en el rendimiento del modelo para la detección de objetos, es posible validar el buen funcionamiento, que se puede visualizar en la siguiente figura.



Figura 3.5: Detección en tiempo real

3.1.1.2. Segmentación de objetos

Se desarrolla un algoritmo de YOLO para segmentar el objeto a trabajar, con lo que se efectúa una etapa de etiquetado y creación del dataset, en donde se busca anotar al objeto, encerrando y detectando el área que utiliza en la imagen, lo que se diferencia del bounding box tradicional. Con este modelo se pretende acceder a mayor información sobre su tamaño equivalente en píxeles. Utilizando las herramientas de Roboflow, se desarrolla un dataset de 426 imágenes, que corresponden a imágenes utilizadas previamente en el modelo de detección. En este caso, se crean anotaciones que describen el contorno del objeto en lugar de un recuadro.

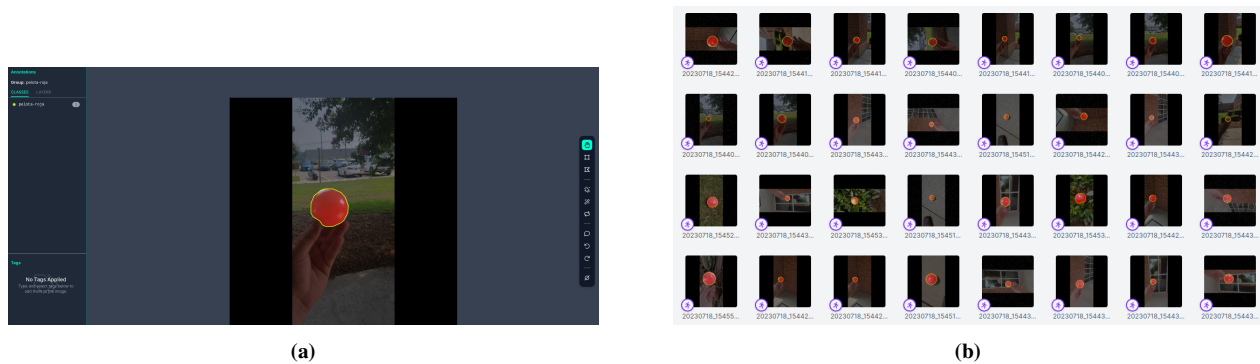


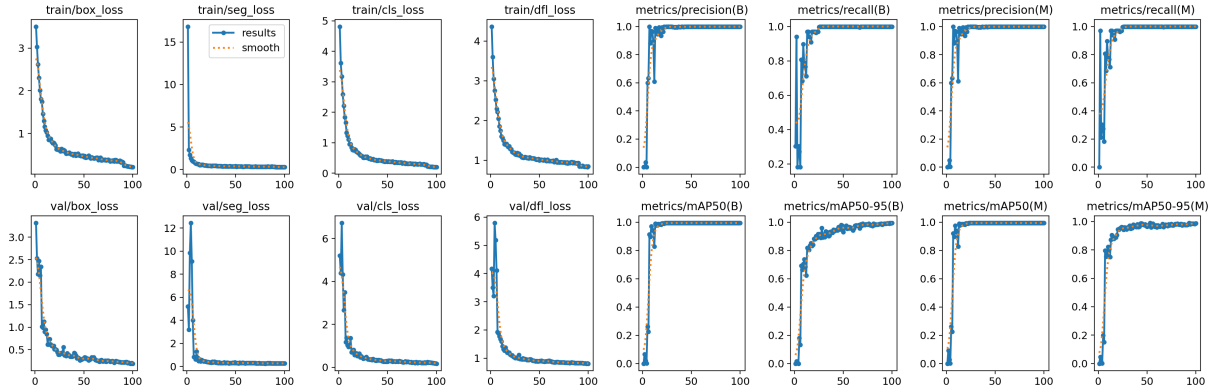
Figura 3.6: La figura 3.6a muestra el formato de anotación de Roboflow, mientras que la figura 3.6b muestra el dataset anotado[13].

El entrenamiento de este modelo se realiza con un *dataset* corresponde a 90 % de datos para entrenamiento, 8 % para validación y 2 % para *testing*. Para la tarea de segmentación en tiempo real se ejecuta para 100 épocas, utilizando las siguientes condiciones:

```

1 model = YOLO('yolov8n-seg.pt') #Modelo pre-entrenado de yolo.
2
3 result = model.train(data=dataPath, imgsz=640, epochs=100, device='cuda',
    batch=8, patience=0, task='segment', workers=8)

```

Figura 3.7: Entrenamiento modelo de segmentación.**Figura 3.8:** Resultados entrenamiento para segmentación de objetos

Las métricas obtenidas durante la fase de entrenamiento para el modelo de segmentación de la figura 3.8 muestran una convergencia que presupone un funcionamiento correcto en el modelo. Es posible validar esta suposición revisando los datos de salida de estas curvas.

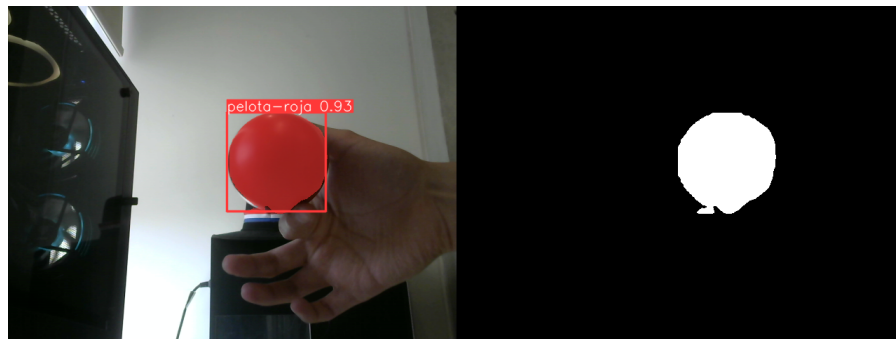
mAP50(B)	0.995
mAP50-95(B)	0.995
precision(B)	0.99844
recall(B)	1

(a) Métricas de rendimiento para bounding box.

mAP50(M)	0.995
mAP50-95(M)	0.98864
precision(M)	0.99844
recall(M)	1

(b) Métricas de rendimiento para detección de máscara.**Tabla 3.4:** Métricas de rendimiento para bounding box y detección de máscara.

Mientras que la tabla 3.4a muestra los resultados para la detección obteniendo un bounding box, la tabla 3.4b muestra los resultados para el proceso de segmentación. Ambas tareas muestran buenos resultados., lo que es posible visualizar en la siguiente figura.

**Figura 3.9:** Segmentación y máscara en tiempo real

3.1.2. Detección basado en segmentación de color

La detección de color es una forma de rastrear un patrón de bajo costo computacional. Para ello se recurre a la representación de color HSV, por las siglas de Hue, Saturation and Value o Matiz, que modela el tipo de color, Saturación, que cuantifica qué tan fuerte se visualiza el color, y Valor, que describe el brillo o intensidad de color. Los parámetros de esta representación en OpenCV varían según los siguientes valores:

$$[H : 0 - 179, S : 0 - 255, V : 0 - 255]$$

Es un modelo de color que similar al RGB, con una representación cilíndrica y que se utiliza como una forma de robustecer la detección frente a las variaciones de luminosidad, característica importante en la detección, en tiempo real, de objetos.

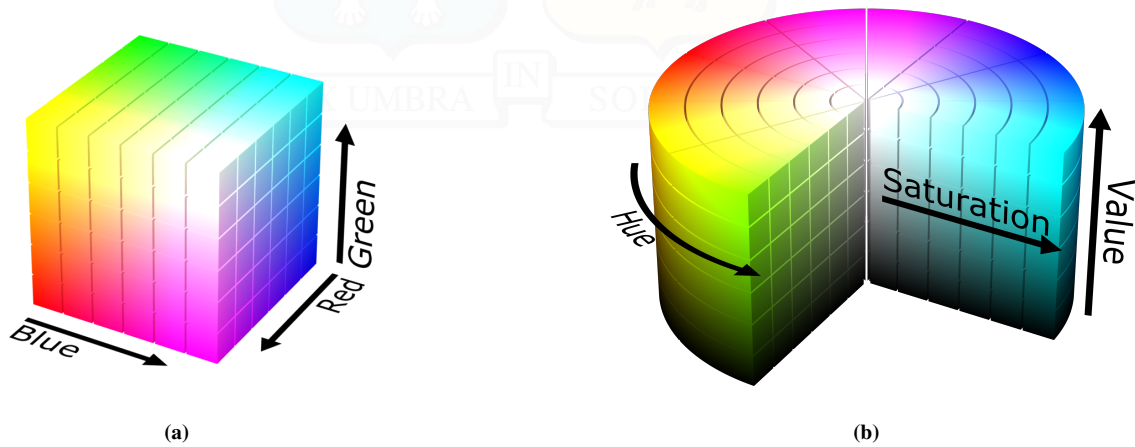


Figura 3.10: La figura 3.10a[14] muestra la variación de colores en formato RGB, mientras que la figura 3.10b[15] muestra la variación de colores en formato HSV

A pesar de que las tecnologías que reproducen imágenes utilizan el formato de representación de color RGB, este espacio de color reproduce un color específico a partir de los 3 parámetros que son la mezcla de rojo, azul y verde. Sin embargo, la utilización de la representación HSV, se utiliza un solo parámetro para esto. Los otros parámetros son importantes para la intensidad y visualización del color.

Por esto, es importante encontrar los parámetros óptimos para el trabajo con la imagen a analizar. Para construir la herramienta de detección basado en segmentación de color se implementa los métodos de CVZone[16], librería basada en OpenCV.

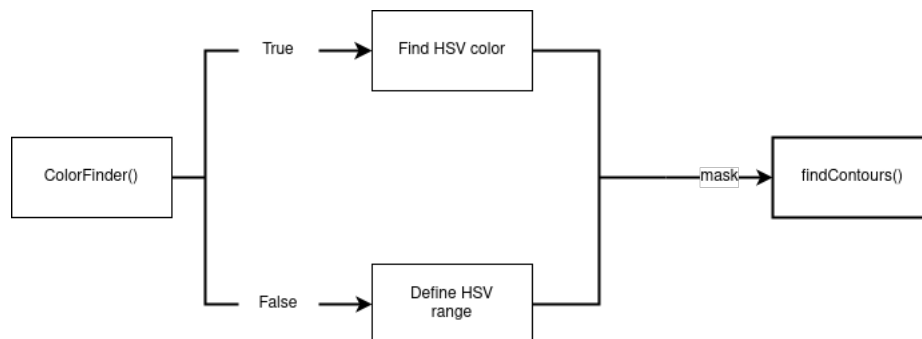


Figura 3.11: Diagrama de métodos CVZone para obtener los rangos HSV y encontrar la máscara necesaria, controlado por flag booleana.

Las imagen de la figura 3.11 corresponde al proceso necesario para encontrar la máscara. El método **ColorFinder(True)** despliega una pantalla con controles deslizantes para simplificar la búsqueda de los rangos necesarios.

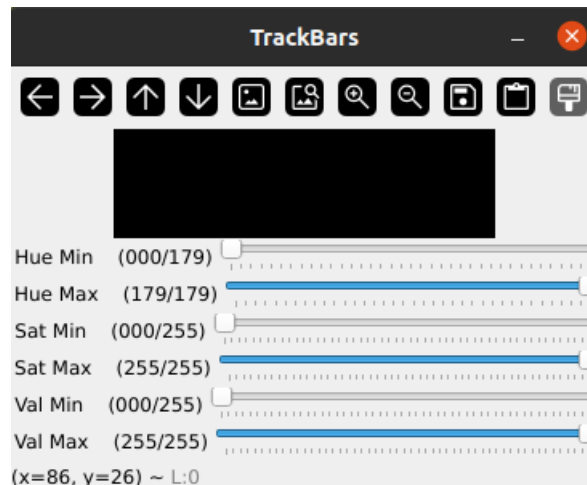


Figura 3.12: Pestaña de CVZone que ilustra el proceso de obtención de rangos HSV.

Encontrando los valores necesarios de HSV mínimo y máximo, se utiliza la flag *false* para el método, que permite trabajar con valores definidos en el código. Luego es posible utilizar este rango como entrada para el método **findContours()**, que permite dibujar el bounding box alrededor de cada parte de color observable en la imagen, retornando el ancho, largo y área de la segmentación.

Finalmente, la segmentación y detección basándose en el color definido por un rango HSV se puede visualizar de la siguiente manera:

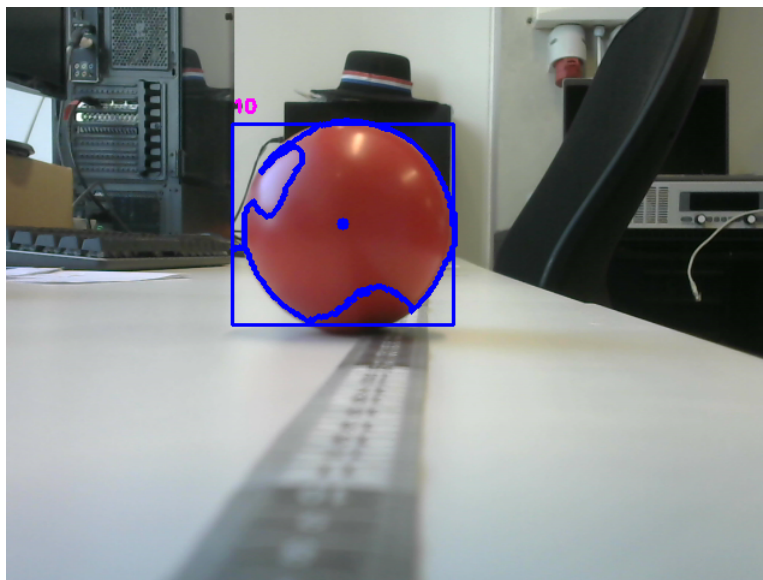


Figura 3.13: Detección y segmentación de color HSV.

3.2. Estimación de profundidad

La limitación más grande que existe en la naturaleza del proyecto está en la estimación de profundidad utilizando una única cámara como medio. Al no utilizar visión estéreo, es necesario utilizar otros medios para captar la profundidad del entorno observado. En primera instancia, se busca la forma de estimar el parámetro de longitud focal. Esta información permite crear una relación entre el tamaño real del objeto, el tamaño percibido en píxeles y la distancia a la que se encuentra dicho objeto, logrando una estimación de la percepción que tiene la cámara utilizada de la imagen que capta. Esta estimación se basa principalmente en la proyección de un triángulo que relaciona la distancia del objeto a la cámara y la representación de la imagen.

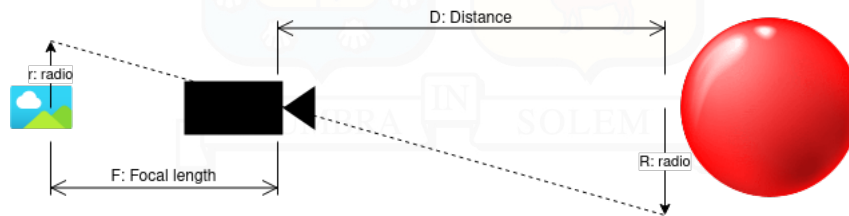


Figura 3.14: Relación distancia focal

Dado el diagrama presentado en la figura 3.14, es posible crear una relación de estimación para el valor de longitud focal[17] bajo la siguiente fórmula:

$$\frac{F}{D} = \frac{r}{R} \quad (3.1)$$

$$F = D \cdot \frac{r}{R} \quad (3.2)$$

Teniendo información sobre la forma real del objeto, en este caso el radio real de la pelota y el radio medido en píxeles, es posible estimar el parámetro F. Con lo que es posible estimar la distancia, en tiempo real, asociando los parámetros conocidos con la imagen de entrada y el área detectada.

Considerando los algoritmos de detección desarrollados en el inciso anterior, es posible establecer lo siguiente:

- Tanto para la segmentación de color como la segmentación semántica establecida por YOLO-seg, se puede realiar una relación entre el área y el radio de la pelota.
- Para la detección de YOLO, es posible usar el ancho del bounding box como forma de obtener el radio de la pelota.

El cálculo, tanto de la longitud focal como de la distancia del objeto, basándose en las ecuaciones 3.1 y 3.2, se traduce a la siguiente implementación en python:

```
1 def FocalLength(measured_distance, real_width, width_in_rf_image):  
2     focal_length = (width_in_rf_image * measured_distance) / real_width  
3     return focal_length
```

(a)

```
1 def Distance_finder(Focal_Length, real_face_width, face_width_in_frame):  
2     distance = (real_face_width * Focal_Length) / face_width_in_frame  
3     return distance
```

(b)

Figura 3.15: La figura 3.15a representa el código para estimar la longitud focal y la figura 3.15b la estimación de distancia[17].

3.3. Coordenadas en el espacio

Dada la necesidad de establecer una relación entre la profundidad obtenida y las coordenadas en el eje x e y , se utiliza un modelo matemático que permite establecer una relación directa entre las coordenadas en píxeles y las coordenadas reales del objeto. Esta relación se basa en el modelo generalizado *pinhole* de una cámara, en que se obtiene un mapeo lineal a través de las características de esta. Este modelo se establece de la siguiente manera:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \quad (3.3)$$

Donde, las coordenadas (u,v) corresponden a la posición en píxeles, mientras que las coordenadas (X,Y,Z) son la posición real del objeto en el espacio.

Desarrollando la relación matricial de las coordenadas para trabajar en función de las coordenadas reales, se puede establecer el siguiente sistema de ecuaciones:

$$\begin{cases} f_x \cdot X + c_x \cdot Z = u \\ f_y \cdot Y + c_y \cdot Z = v \\ Z = 1 \end{cases}$$

Donde es posible establecer una relación dependiente de las coordenadas en píxeles de la siguiente forma:

$$X = \frac{u - c_x}{f_x} \quad (3.4)$$

$$Y = \frac{v - c_y}{f_y} \quad (3.5)$$

En esta relación se aplica una convención que permite relacionar las coordenadas en el eje X e Y con el valor detectado en la coordenada Z , donde:

$$(x, y, z) \Rightarrow (k \cdot x, k \cdot y, k) \quad (3.6)$$

Reestructurando las relaciones establecidas previamente utilizando la convención cuando $Z \neq 1$, se escala las ecuaciones 3.4 y 3.5 según el valor de profundidad detectado.

$$X = \frac{u - c_x}{f_x} \cdot k \quad (3.7)$$

$$Y = \frac{v - c_y}{f_y} \cdot k \quad (3.8)$$

La matriz presentada en la ecuación 3.3 se denomina **matriz de parámetros intrínsecos**. Los parámetros corresponden a la siguiente información:

- **C:** Corresponde al **centro óptico**, parámetro que ubica el centro de la imagen.

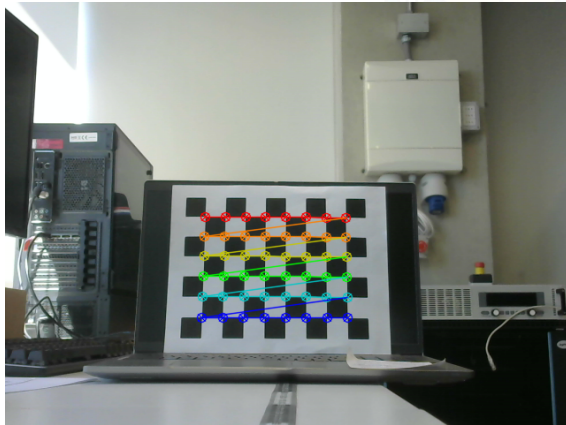
$$C = (c_x, c_y) \quad (3.9)$$

- F: Corresponde al **Focal length**, parámetro que contiene el ancho y largo de la imagen obtenida por la cámara.

$$F = (f_x, f_y) \quad (3.10)$$

La información que caracteriza la cámara utilizada para esta matriz se puede calcular mediante rutinas de calibración y evitan la distorsión que da resultado de la estructura interna de esta herramienta y las características de la lente. La calibración permite disminuir los errores en las imágenes recolectadas, además de entregar mayor confiabilidad. Los parámetros son específicos de cada cámara, por lo que de no contar con ellos previamente, se debe realizar esta rutina cada vez que se implemente un nuevo hardware recolección de imágenes.

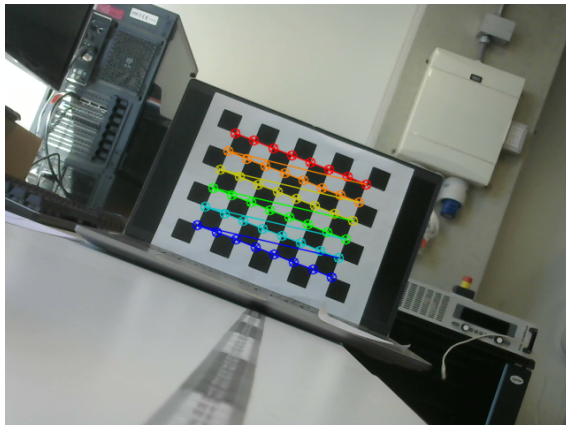
Para la aplicación necesaria mediante la webcam, se desarrolla una rutina de calibración utilizando las herramientas de la librería de OpenCV[18]. Para ello, se utiliza un tablero de ajedrez e imágenes tomadas por la misma cámara.



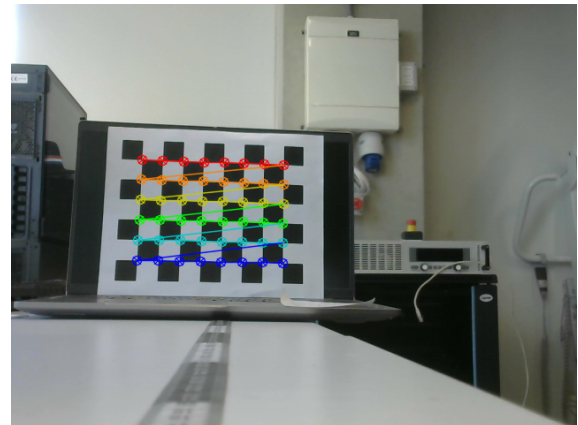
(a)



(b)



(c)



(d)

Figura 3.16: Calibración para obtener parámetros intrínsecos de webcam. Fueron utilizadas 40 imágenes para este procedimiento.

Basándose en las dependencias de OpenCV y teniendo imágenes del tablero de ajedrez, tomadas por la webcam, es posible aplicar los métodos de calibración.

Con esto en mente, es posible caracterizar la cámara y utilizar sus parámetros para estimar valores. A esta matriz se le suele representar con la letra **K**, cuyos parámetros intrínsecos se pueden visualizar en la siguiente matriz:

$$\mathbf{K} = \begin{pmatrix} 823,2638336 & 0 & 320,55874298 \\ 0 & 830,00741099 & 305,66295207 \\ 0 & 0 & 1 \end{pmatrix} \quad (3.11)$$

Dada la matriz de parámetros intrínsecos y la relación presente entre las coordenadas en píxeles y las reales, es posible visualizar un pipeline que conlleva desde la toma de datos mediante la cámara, pasando por un bloque de seguimiento, que se realiza mediante YOLO o mediante segmentación de color, estimando una distancia focal entre la posición de la cámara y el objeto observado. Finalizando este circuito, se tiene un bloque de estimación de coordenadas, entregando como salida la posición en el espacio de la pelota roja.

El primer cálculo necesario para estimar las coordenadas es la posición en el eje Z, que se tomará como la distancia, por lo tanto, el eje Z^+ se considera como la dirección a la que apunta la cámara. Para obtener las coordenadas en los ejes (x,y), asumiendo Z como parámetro conocido, es posible utilizar las fórmulas 3.7 y 3.8. Las coordenadas obtenidas se escriben bajo el sistema internacional, por lo que se representa en metros, con 3 cifras significativas de redondeo. La estimación de coordenadas asume que el eje de coordenadas se encuentra en la posición de la cámara. Esto elimina la posibilidad de tener cálculos indeterminados dentro de este sistema, que estén asociados a los valores estimados.

4 | Entorno de simulación

En este capítulo se detalla el proceso de elaboración del ambiente simulado para la plataforma de evaluación del robot iCub.

4.1. Simulación mediante Gazebo

La simulación es una etapa de experimentación controlada que posee variadas ventajas como la disminución de errores y mejorar las implementaciones reales. En este proyecto se utiliza Gazebo como entorno de simulación, siendo factible la comunicación entre los puertos del iCub y algoritmos desarrollados en C++ o Python mediante YARP. La elección de este entorno de simulación se fundamenta en la trayectoria de proyectos y estudios previos que utilizan Gazebo.

4.1.1. Interacción con el mundo simulado

El entorno de simulación de Gazebo se inicializa en su formato más simple como un espacio vacío, en el cual se interactúa arrastrando los modelos disponibles dentro de las librerías y frameworks instalados en el programa. Las dependencias de *robotology-superbuild* permiten el acceso al robot iCub a través de Gazebo, ofreciendo una variedad de modelos que representan diferentes versiones del mismo. Estas versiones varían en las extremidades y articulaciones disponibles para la interacción, lo que permite acceder a los puertos que las controlan.

El modelo del iCub seleccionado para este proyecto es el iCub v2.5 *visuomanip*, que corresponde a una versión que tiene disponibles las articulaciones del tren superior de su cuerpo, dejando las piernas de forma estática. Dado que las tareas plantificadas en el contexto de la implementación de la plataforma no requieren movimientos en el tren inferior, este modelo satisface los requerimientos principales. Es relevante mencionar que la incorporación del modelo del iCub y la implementación de aplicaciones para este están sujetas a la inicialización del servidor *yarpserver*, que reconocerá la incorporación de cualquier modelo del robot en el momento que se posicione en la simulación.

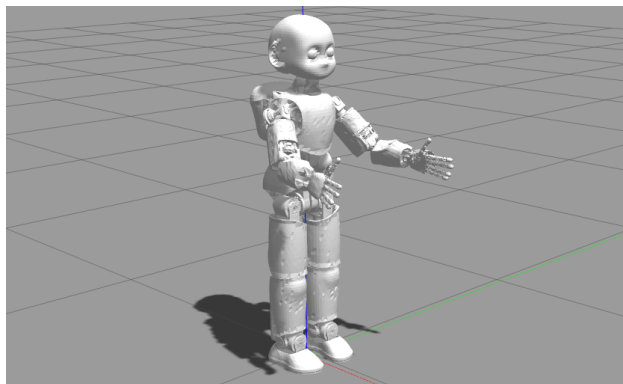


Figura 4.1: Modelo iCub v2.5 *visuomanip*

Por otro lado, para la construcción de la plataforma de trabajo, es necesario acceder a los modelos 3D disponibles en Gazebo, entregando como herramientas la mesa de trabajo y la cámara. Esta última representa la versión más básica de simulación de una webcam para la estimación de parámetros de posición.

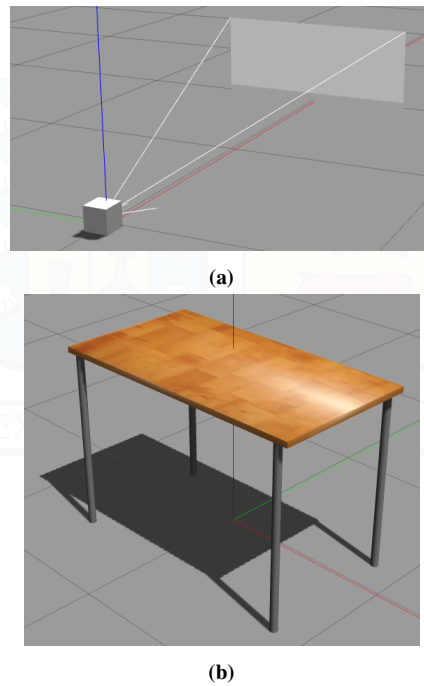


Figura 4.2: La figura 4.2a corresponde al modelo básico de cámara existente en Gazebo, y la figura 4.2b representa la mesa de trabajo utilizada en proyectos asociados al iCub.

Para validar la conectividad entre la simulación del robot y poder trabajar posteriormente utilizando la plataforma, se establece una lectura a la información capturada por el ojo derecho mediante Python, obteniendo lo siguiente:

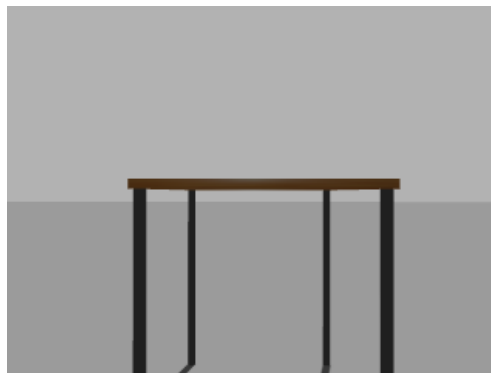


Figura 4.3: Visualización imagen desde ojo derecho de robot iCub

Las simulaciones desarrolladas se guardan como un archivo con la extensión **.sdf**, que se refiere a *Simulation Description Format*. Corresponde a un formato XML que describe objetos y entornos de simulación, visualización y control enfocado en robots. También es posible visualizar características como gravedad o la posición del sol.

4.1.2. Construcción de simulación modular

La importancia del trabajo modular en un proyecto radica en la capacidad de gestionar diferentes partes de manera independiente, entregando una estandarización y facilitando la edición de parámetros o estados. En el contexto de la creación de simulaciones en Gazebo, es posible estructurar el proyecto de manera más organizada, separando la configuración de parámetros del entorno y condiciones iniciales en archivos distintos. Para lograr esta estructura, se construye la simulación mediante la integración del archivo con extensión **.world**[19]. Con este archivo se define el formato de la plataforma, los modelos 3D que se requieren importar y las condiciones de inicialización, como las coordenadas de posición y el estado que se encuentra cada objeto.

```

1  <?xml version="1.0" ?>
2  <sdf version="1.7">
3    <world name="default">
4      <include>
5        <uri>model://sun</uri>
6      </include>
7      <light name='user_point_light_0' type='point'>
8        <pose frame=''>0.197302 -0.215077 1 0 -0 0</pose>
9        <diffuse>0.5 0.5 0.5 1</diffuse>
10       <specular>0.1 0.1 0.1 1</specular>
11       <attenuation>
12         <range>20</range>
13         <constant>0.5</constant>
14         <linear>0.01</linear>
15         <quadratic>0.001</quadratic>
16       </attenuation>
17       <cast_shadows>0</cast_shadows>
18       <direction>0 0 -1</direction>
19     </light>
20     <state world_name='default'>
21       <light name='sun'>
22         <pose frame=''>0.530772 0 10 0 -0 3.14</pose>
23       </light>
24       <light name='user_point_light_0'>
25         <pose frame=''>0.197302 -0.215077 5 0 -0 0</pose>
26       </light>
27     </state>
28     <include>
29       <uri>model://ground_plane</uri>
30     </include>
31     <include>
32       <uri>/ruta_a_modelo/mesa</uri>
33       <pose>0 0 0 0 -1.57</pose>
34     </include>
35     <include>
36       <uri>/ruta_a_modelo/camera</uri>
37       <pose>0 0 1.315000 -3.11 1.555 0</pose>
38       <scale>1 1 1</scale>
39       <link name='link'>
40         <velocity>0 0 0 0 -0 0</velocity>
41         <acceleration>0 0 0 0 -0 0</acceleration>
42         <wrench>0 0 0 0 -0 0</wrench>
43       </link>
44     </include>
45     <model name="iCub">
46       <include>
47         <uri>model://iCubGazeboV2_5_visuomanip</uri>
48         <pose>0.75 -0.034068 0.63 0.0 0.0 0.0</pose>
49       </include>
50     </model>
51   </world>
52 </sdf>

```

Figura 4.4: Descripción de archivo **platform.world**

Adicionalmente a los modelos mencionados en la figura 4.2, se agrega el archivo de descripción para el modelo de la pelota roja. Como este modelo no existe naturalmente en las dependencias de Gazebo, es necesario el campo de descripción del formato que tendrá la visual del objeto, estableciendo la forma, color y tamaño del objeto.

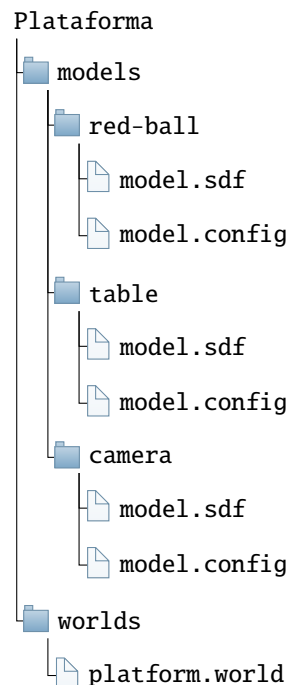
```

1  <link name="link">
2    <visual name="visual">
3      <geometry>
4        <sphere>
5          <radius> 0.035 </radius>
6        </sphere>
7      </geometry>
8      <material>
9        <script>
10         <uri>file://media/materials/scripts/gazebo.material</uri>
11         <name>Gazebo/Red</name>
12        </script>
13      </material>
14    </visual>
15  </link>

```

Figura 4.5: Descripción de la visualización de objeto **red-ball** en archivo *.sdf*

La plataforma de simulación para el robot iCub se organiza bajo 2 carpetas principales, una para la recopilación de los modelos que se incorporan al entorno, así como otra carpeta que posee la descripción y características del entorno. La jerarquía se establece de bajo el siguiente diagrama:



En conjunto a la construcción de la plataforma, se agrega movimiento al modelo *red-ball*. Para ello, se edita el archivo *model.sdf* cambiando el atributo del objeto al formato de tipo *actor*[20]. Con esto, es posible definir posiciones en el espacio en un cierto instante de tiempo, logrando que el modelo pueda trazar una trayectoria entre esos puntos.

```

1 <trajectory id="0" type="square">
2   <waypoint>
3     <time>0.0</time>
4     <pose>0.130772 0.25 0.51 0 0 0</pose>
5   </waypoint>
6   <waypoint>
7     <time>1.0</time>
8     <pose>0.130772 -0.25 0.51 0 0 0</pose>
9   </waypoint>
10  <waypoint>
11    <time>2.0</time>
12    <pose>-0.069228 -0.25 0.51 0 0 0</pose>
13  </waypoint>
14  <waypoint>
15    <time>3.0</time>
16    <pose>-0.069228 0.25 0.51 0 0 0</pose>
17  </waypoint>
18  <waypoint>
19    <time>4.0</time>
20    <pose>0.130772 0.25 0.51 0 0 0</pose>
21  </waypoint>
22 </trajectory>

```

Figura 4.6: Descripción de las coordenadas y el instante en que se debe posicionar el objeto *red-ball*. La trayectoria considera 5 puntos de movimiento y el objeto elabora una trayectoria basada en esos puntos.

Al ejecutar el archivo *platform.world* con los comandos de Gazebo, es posible cargar los modelos y establecer las posiciones deseadas para cada uno. La cámara se encuentra rotada para proporcionar una vista superior de la mesa, asegurando que la pelota permanezca en todo momento dentro del campo de visión de la cámara. Además, la mesa ha sido ajustada en altura para adecuarse a la posición y tamaño del robot iCub.

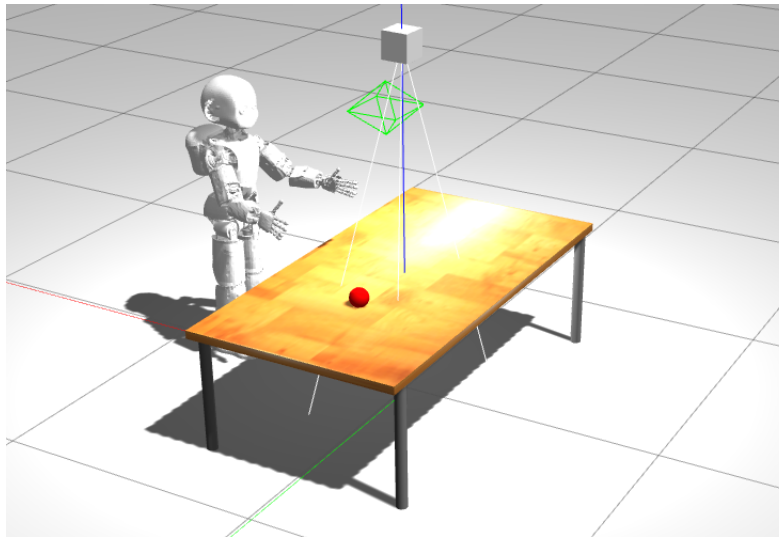


Figura 4.7: Plataforma de evaluación construida modularmente

4.2. Integración con ROS

La estructuración del proyecto, utilizando algoritmos desarrollados en Python, requiere que la simulación completa pueda ser controlada mediante este lenguaje de programación. Sin embargo, se detecta una dificultad debido a la falta de librerías, compatibilidad y conectividad entre las versiones de *Gazebo11* y *Python3*, lo que dificulta la

comunicación de los datos obtenidos por la cámara sobre la plataforma. Ante esta situación, surge la necesidad de integrar ROS como una solución factible.

4.2.1. Creación archivo de lanzamiento

Para incorporar las dependencias de ROS, es importante contar con los requerimientos necesarios. En este caso se utiliza la distribución *ROS noetic*[21], que es compatible con la versión de *Ubuntu 20.04* y *Gazebo11*. Luego de las instalaciones pertinentes, se requiere un archivo de inicialización de simulación de Gazebo con las librerías de ROS, este archivo tiene la extensión *.launch*[22] y se puede ejecutar con el comando *roslaunch*.

```

1 <launch>
2
3 <!-- these are the arguments you can pass this launch file, for example
   paused:=true -->
4 <arg name="paused" default="false"/>
5 <arg name="use_sim_time" default="true"/>
6 <arg name="gui" default="false"/>
7 <arg name="headless" default="false"/>
8 <arg name="debug" default="false"/>
9
10 <!-- We resume the logic in empty_world.launch, changing only the name of
   the world to be launched -->
11 <include file="$(find gazebo_ros)/launch/empty_world.launch">
12   <arg name="world_name" value="/ruta_platform/worlds/platform.world"/>
13   <arg name="debug" value="$(arg debug)" />
14   <arg name="gui" value="$(arg gui)" />
15   <arg name="paused" value="$(arg paused)" />
16   <arg name="use_sim_time" value="$(arg use_sim_time)" />
17   <arg name="headless" value="$(arg headless)" />
18 </include>
19
20 </launch>

```

Figura 4.8: Descripción de archivo *icub_platform.launch*

El archivo de lanzamiento, mostrado en la figura 4.8, consta de varias características de inicialización. Describe si la simulación inicia en pausa, si debe iniciar en modo *debugging*, o si se quiere visualizar la *gui*, entre otras cosas. Además, llama un mundo vacío de Gazebo e importa el archivo **platform.world** desarrollado en la etapa previa. A través del comando *roslaunch* es posible iniciar automáticamente el servidor *rostopic*, requerido para que las librerías funcionen correctamente.

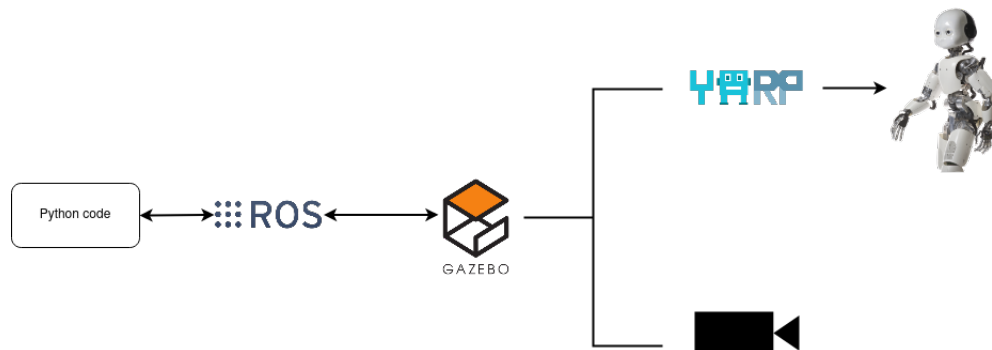


Figura 4.9: Conectividad entre plataformas.

La figura 4.9 muestra la comunicación entre las distintas dependencias y sensores importantes. A través de algoritmos desarrollados en Python, es posible acceder a los datos obtenidos por la cámara estableciendo una conexión incorporando un nodo de ROS. Por otro lado, se tiene la conectividad con el robot iCub utilizando las librerías de YARP como medio.

A través de las herramientas de ROS, se inicializa un nodo con el nombre **'camera_subscriber'**. Este nodo se suscribe al tópico **/my_camera/image_raw**, permitiendo la recepción de imágenes publicadas por la cámara en Gazebo, las que son procesadas para que estén en un formato que OpenCV pueda manejar y ser utilizadas en etapas posteriores.

Las herramientas de ROS permiten confirmar esta conectividad entre Gazebo y ROS, mostrando una visualización de la información captada por los sensores que están publicados y disponibles. Debido a que la cámara se considera un sensor de imagen, a través de la herramienta *rviz* es posible recibir las imágenes que captura.

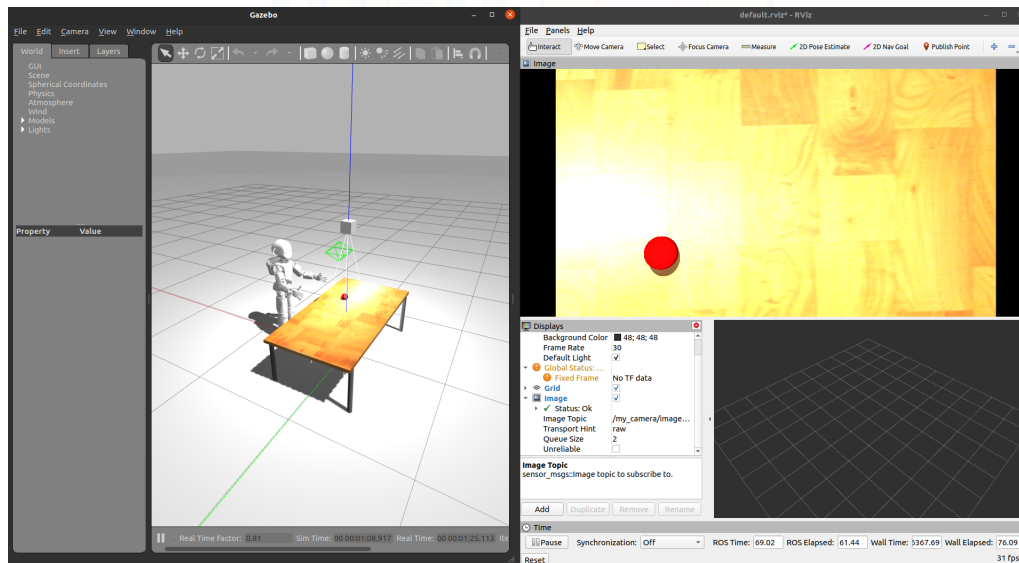


Figura 4.10: Imagen capturada por cámara en Gazebo visualizada en rviz

La figura 4.10 muestra la interacción entre Gazebo y la interfaz obtenida mediante el comando *roslaunch rviz rviz*. A través de esta visualización, es posible rastrear la imagen capturada por la cámara simulada. Es importante destacar que, aunque la simulación en Gazebo puede experimentar retardos debido a los recursos computacionales utilizados, en *RViz* se puede observar movimientos fluidos de la pelota, indicando que los retardos percibidos no presentan un problema real para la toma de datos.

5 | Integración de modelo de estimación 3D a plataforma de simulación

Para elaborar una simulación coherente con el proyecto, es necesario realizar ajustes y calibraciones requeridos para las herramientas del entorno con el que se trabaja. Para ello, se busca la forma de adaptar los módulos de estimación de coordenadas en función de compatibilizarlos con la simulación de la plataforma desarrollada para el iCub.

Para obtener una simulación adecuada en la evaluación de modelos, es necesario desarrollar un sistema que cumpla con los criterios de comparación. Es fundamental asegurarse de que las coordenadas estimadas, tanto del iCub como de la cámara, se encuentren mapeadas bajo el mismo eje, por lo que se asume que el centro del sistema de coordenadas se encuentra en la posición de la cámara. Luego, utilizando transformaciones lineales, se proyectan las coordenadas del iCub a esta referencia. Se establece así un sistema de interacción entre las simulaciones y los modelos de Python, donde es posible hacer comparaciones entre la posición estimada por la cámara y la construcción realizada por el iCub.

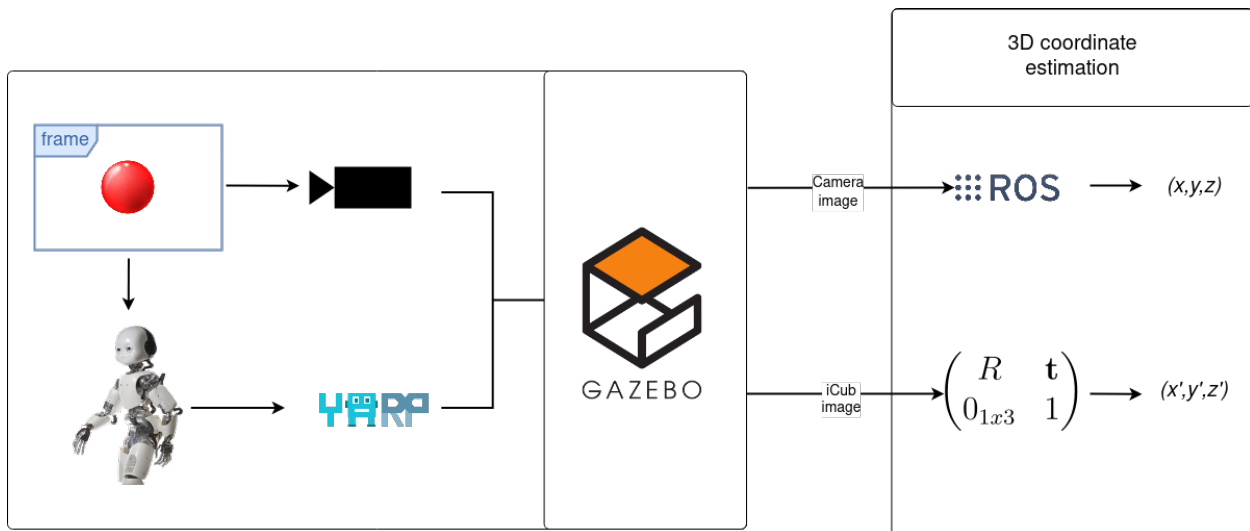


Figura 5.1: Comunicación entre módulos de estimación de coordenadas y simulaciones del robot iCub y cámara al obtener el frame de entrada del objeto.

Dentro del sistema establecido en la figura 5.1 es posible notar la existencia de una matriz que interactúa directamente con la imagen obtenida por el robot iCub, la cual se conoce como **matriz de parámetros extrínsecos** y corresponde a la descripción de rotación y traslación aplicadas a las coordenadas que se estimen. Bajo estas condiciones, es posible realizar los ajustes necesarios.

5.1. Transformación de coordenadas estimadas

Para poder utilizar métricas de comparación y realizar simulaciones confiables de la posición estimada por ambos observadores, es necesario definir un único sistema. En este sentido es que se trabaja con la matriz de parámetros extrínsecos aplicado a la visión del iCub.

La matriz de parámetros extrínsecos[23] que describe estas transformaciones se conforma por 2 procesos fundamentales:

- Matriz de rotación: Describe una rotación según un ángulo dado alrededor de alguno de los ejes de coordenadas.
- Vector de traslación: Este vector representa la dirección en la que se mueve un punto en el espacio y puede utilizarse para mover el eje de coordenadas local a una posición dada.

La matriz de parámetros extrínsecos se puede desglosar de la siguiente manera:

$$\mathbf{P} = \begin{pmatrix} R_{11} & R_{12} & R_{13} & T_x \\ R_{21} & R_{22} & R_{23} & T_y \\ R_{31} & R_{32} & R_{33} & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (5.1)$$

Para poder obtener esta matriz, se evalúan las posiciones actuales en las que se encuentran tanto el iCub como la cámara, tomando en cuenta este último modelo como el punto de referencia para el resto de cálculos. Adicionalmente a esto, es importante tener en cuenta que para que el iCub pueda observar directamente el lugar donde ocurre el movimiento del objeto, es necesario aplicar una rotación de -40° al cuello. Dada esta información, y teniendo en cuenta que la posición de la cámara es sobre la mesa de trabajo, se toma como referencia que el robot debe *rotar* su visión -50° extra, para completar una rotación de 90° hacia abajo. Esto implica que la rotación se aplica en torno al eje y en la referencia local del iCub, por lo que la matriz de rotación que describe este movimiento se describe bajo la siguiente estructura:

$$\mathbf{Rot}(50^\circ) = \begin{pmatrix} \cos(50^\circ) & 0 & \sin(50^\circ) \\ 0 & 1 & 0 \\ -\sin(50^\circ) & 0 & \cos(50^\circ) \end{pmatrix} \quad (5.2)$$

Por otro lado, es posible calcular el vector de traslación teniendo las coordenadas de la posición del ojo del iCub, así como también las de la cámara. En este caso, se trabaja particularmente con el ojo del robot para tener una simplificación del problema, ya que es particularmente el eje de referencia local del ojo el que se intenta proyectar al momento de aplicar la traslación.

En este contexto, la relación para obtener el vector de traslación es a través de la siguiente forma:

$$\mathbf{t} = (x_2 - x_1, y_2 - y_1, z_2 - z_1) \quad (5.3)$$

Donde las coordenadas se pueden interpretar como:

- (x_1, y_1, z_1) : Coordenadas de origen. Corresponden a las coordenadas del ojo del iCub.
- (x_2, y_2, z_2) : Coordenadas de destino. Corresponden a las coordenadas de la cámara.

Considerando las posiciones obtenidas en la simulación, es posible obtener el siguiente vector de traslación:

$$\mathbf{t} = (0,693799, 0,000005, -0,338206) \quad (5.4)$$

A partir de los parámetros obtenidos en las ecuaciones 5.2 y 5.4 es posible determinar la equivalencia de las coordenadas estimadas en la referencia de la cámara, bajo la siguiente relación:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} \cos(50^\circ) & 0 & \sin(50^\circ) \\ 0 & 1 & 0 \\ -\sin(50^\circ) & 0 & \cos(50^\circ) \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} 0,693799 \\ 0,000005 \\ -0,338206 \end{pmatrix} \quad (5.5)$$

Donde:

- (x', y', z') : Corresponde a la coordenada vista por el iCub, según el eje de referencia de la cámara.
- (x, y, z) : Corresponde a la coordenada vista por el iCub en su eje de referencia local.

5.2. Parámetros de visión para simulación

Otro aspecto particular del trabajo de simulación que debe ser ajustado es la información sobre las características de las cámaras con las que se trabaja, particularmente, los parámetros intrínsecos de esta.

5.2.1. Cámara Gazebo sim

Para obtener las características de la cámara necesarias en la simulación de la webcam en la plataforma, es posible aplicar técnicas de calibración dentro de la simulación. Sin embargo, para el modelo utilizado, estos parámetros son fácilmente obtenidos a partir de la información a la que se puede acceder del modelo utilizando comandos propios de Gazebo. Dado que esta cámara viene calibrada, solo se requiere obtener estos valores para poder aplicar las técnicas de estimación anteriormente descritas.

En función de esto, se obtiene la siguiente matriz de parámetros intrínsecos que describen las características de la cámara de simulación con la que se trabaja.

$$\mathbf{K} = \begin{pmatrix} 1663,1481384679323 & 0 & 960,5 \\ 0 & 1663,1481384679323 & 540,5 \\ 0 & 0 & 1 \end{pmatrix} \quad (5.6)$$

5.2.2. Visión iCub

Las características principales del iCub en el modelo de simulación no vienen preestablecidas como el modelo de cámara. Para esto se decide utilizar la matriz de parámetros intrínsecos que corresponde a la cámara del modelo de robot real con el que es posible trabajar. El motivo de esta decisión es poder emular las características de la realidad en la simulación y tener una forma de replicar el entorno real.

Dada esta decisión, los parámetros establecidos en la calibración[24] del iCub, según los repositorios del mismo, determinan la siguiente matriz de parámetros para trabajar.

$$\begin{pmatrix} 215,483 & 0 & 174,868 \\ 0 & 214,935 & 105,63 \\ 0 & 0 & 1 \end{pmatrix} \quad (5.7)$$

6 | Simulaciones y resultados

Dando por finalizada la etapa de construcción de la plataforma de simulación, es importante realizar las pruebas y experimentos pertinentes que validen el correcto funcionamiento del sistema. Para ello, se realizan simulaciones controladas que consideren algunos casos relevantes de estudio.

Previo a la toma de datos en los experimentos, es necesario contar con la sincronización adecuada entre los distintos modelos que se implementarán para esta tarea. Es por esto que, dada la existencia de las herramientas de ROS dentro del proyecto, se inicializan 2 puertos adicionales, que servirán como medio de sincronización, permitiendo establecer una frecuencia de toma de datos, en función de tener la certeza de que los datos medidos sean comparables en las instancias que se guardan.

La sincronización se realiza mediante el método **publisher()** de ROS, donde se inicializan 2 puertos a utilizar.

- `/datos_sensor1`
- `/datos_sensor2`

Dados estos tópicos, se desarrolla un código de sincronización que se suscribe a cada uno de estos y se establece una toma de datos cada 0.05 segundos. Con esto, se disminuye la cantidad de datos repetidos que puedan tomarse debido a las tasas de envío de datos particular de cada modelo.

6.1. Validación de estimación de coordenadas

Dada la dificultad que supone estimar la posición de un objeto en el espacio a través de una imagen monovisual, es importante realizar una validación del modelo que permita verificar qué tan confiable resulta ser la información rescatada. En este contexto, se plantea realizar 2 experimentos para la toma de datos. Debido a que esta validación corresponde a la estimación de coordenadas que se utilizará posteriormente para ser implementada en la webcam, es necesario recopilar los datos reales del objeto observado dados por la simulación. Es por esto que, a través de las herramientas proporcionadas por la librería *rospy*, es posible obtener la posición publicada por la simulación.

```

1 def get_model_position(model_name):
2     rospy.wait_for_service('/gazebo/get_model_state')
3     try:
4         get_model_state = rospy.ServiceProxy('/gazebo/get_model_state',
5             GetModelState)
6         model_state = get_model_state(model_name, 'world')
7         position = model_state.pose.position
8         return [position.x, position.y, position.z]
9     except rospy.ServiceException as e:
10        print("Service call failed:", e)

```

Figura 6.1: Obtención de posición real de modelo *red-ball*.

6.1.1. Métricas de evaluación

Dada la importancia en la precisión con la que el modelo realiza la estimación de coordenadas, surge la necesidad de calcular parámetros que permitan evaluar el funcionamiento. Para ello, se requieren de métricas que establezcan una comparación cuantificable entre las posiciones estimadas y reales, por lo que las métricas de evaluación serán las siguientes:

- *Error cuadrático medio*: Dada la naturaleza de esta herramienta, permite evaluar el modelo, penalizando en mayor medida los errores de mayor grado.

$$ECM = \frac{1}{N} \sum_{i=1}^N (Estimado - Real)^2 \quad (6.1)$$

- *Error porcentual*: Permite el cálculo del error existente entre una posición real y su estimación, en cada coordenada por separado.

$$E \% = \left| \frac{Estimado - Real}{Real} \right| * 100 \quad (6.2)$$

- *Error porcentual total*: Para establecer un error porcentual asociado a la totalidad de los datos en cada coordenada, se calcula el promedio de los errores.

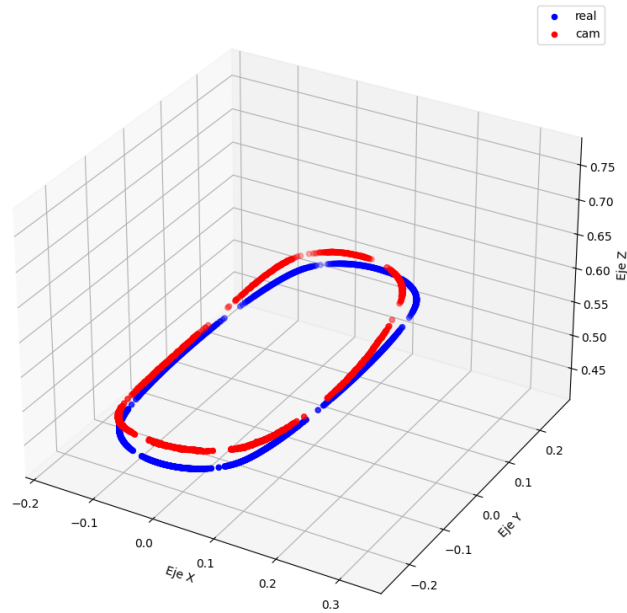
$$E \%_{total} = \frac{1}{N} \sum_{i=1}^N E \% \quad (6.3)$$

- *Error del modelo*: El porcentaje de error que se obtiene en el modelo completo se obtiene a través del promedio de los errores porcentuales en cada coordenada.

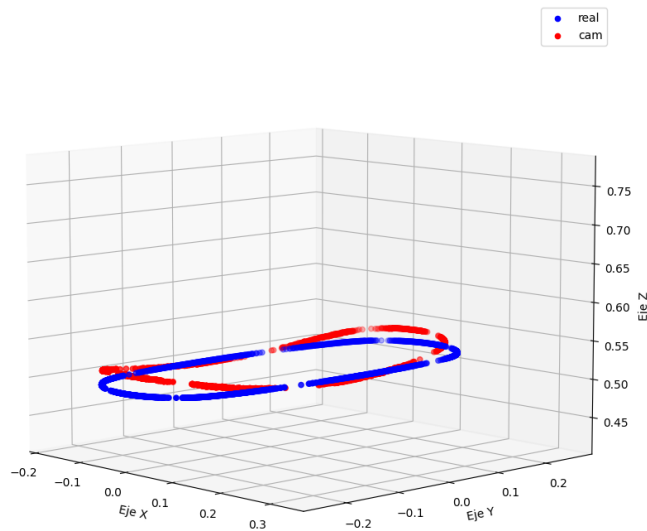
$$E_{modelo} = \frac{E \%_{total_x} + E \%_{total_y} + E \%_{total_z}}{3} \quad (6.4)$$

6.1.2. Trayectoria sin variación de profundidad

En este experimento se establece una trayectoria que consta de 4 posiciones establecidas cada 1 segundo, por lo que el modelo establece un movimiento pasando por cada uno de estas posiciones. Además, se pondrá en contraste la posición real del modelo con cada uno de los algoritmos de estimación implementados en secciones previas. Para establecer una comparación entre los datos teóricos y los estimados, se grafican ambos grupos de datos, mostrando la trayectoria del objeto. El color azul representa la trayectoria real del objeto, mientras que el color rojo representa la trayectoria estimada.



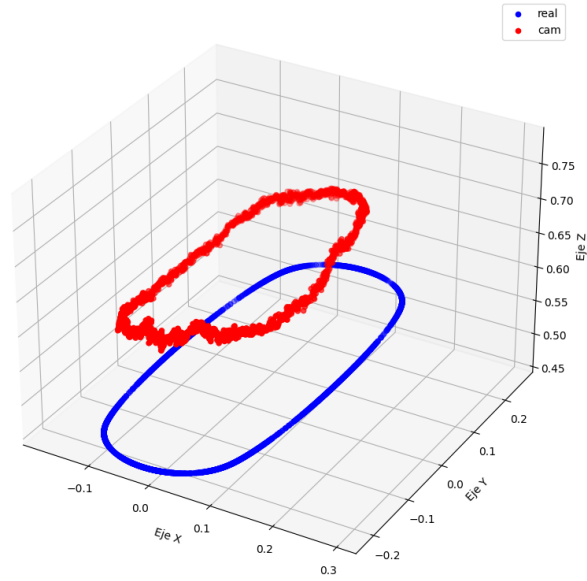
(a)



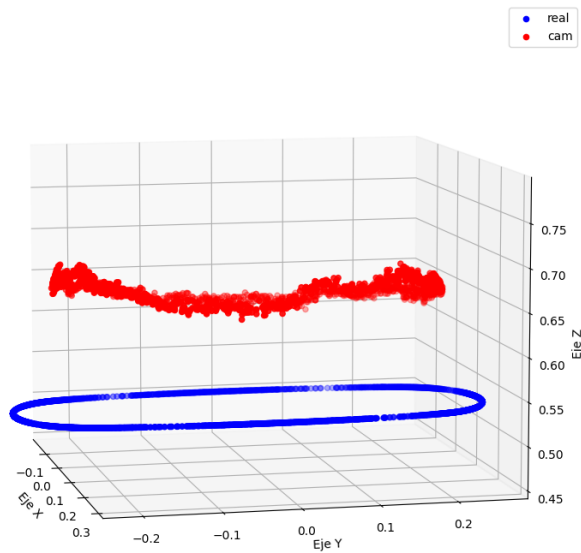
(b)

Figura 6.2: Las figuras 6.2a y 6.2b corresponden a la trayectoria real y estimada del objeto utilizando la técnica de **detección por segmentación de color**.

En los gráficos de la figura 6.2 es posible visualizar una trayectoria estable, donde la variación más notoria se presenta en el eje z. La dificultad de visualizar profundidad es una situación esperada, sin embargo, es posible visualizar un buen funcionamiento.



(a)



(b)

Figura 6.3: Las figuras 6.3a y 6.3b representan corresponden a la trayectoria real y estimada del objeto utilizando la técnica **detección y clasificación con YOLO**.

En los gráficos de la figura 6.3 se observa una diferencia considerable en la estimación del valor de la coordenada en z . Esto supone una pérdida de información en el proceso de estimar la distancia entre la cámara y el objeto a través de las dimensiones del bounding box visible.

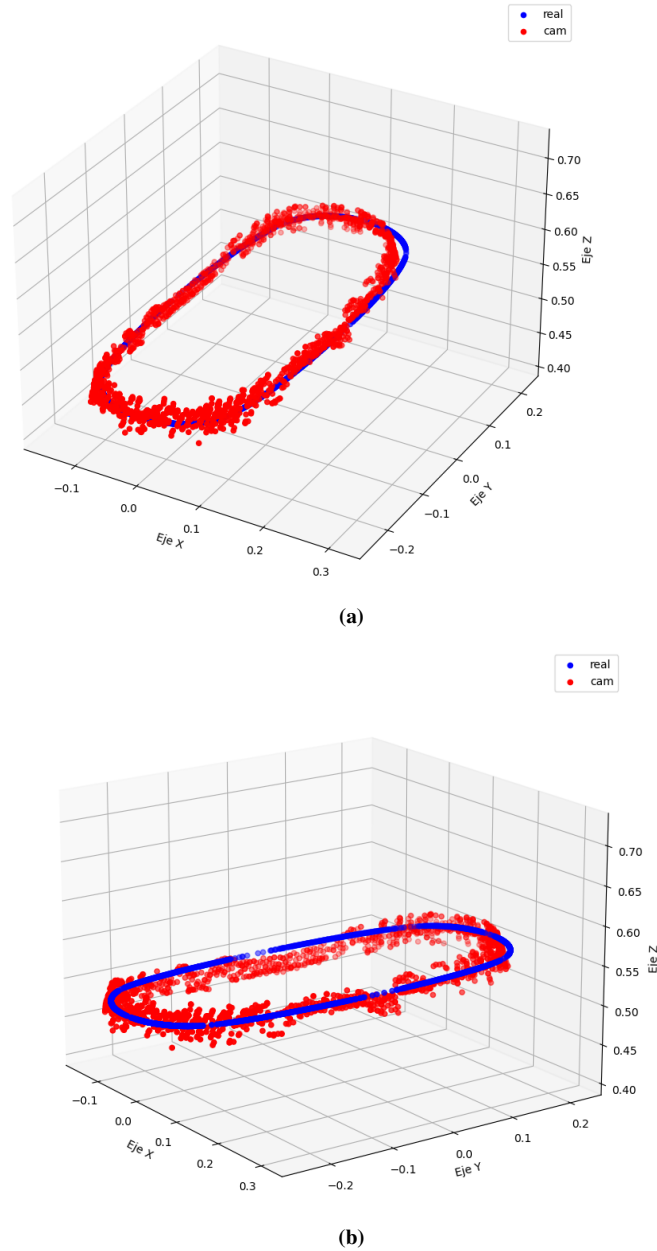
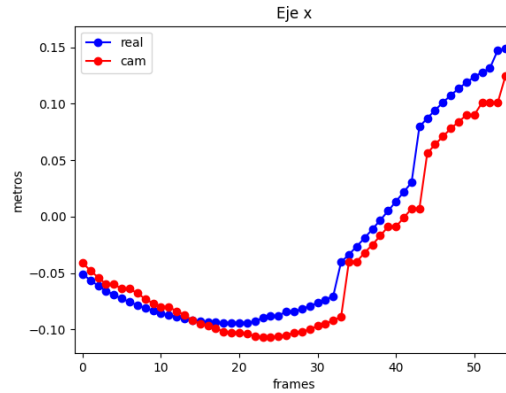


Figura 6.4: Las figuras 6.4a y 6.4b representan corresponden a la trayectoria real y estimada del objeto utilizando la técnica de **segmentación semántica y detección con YOLO**.

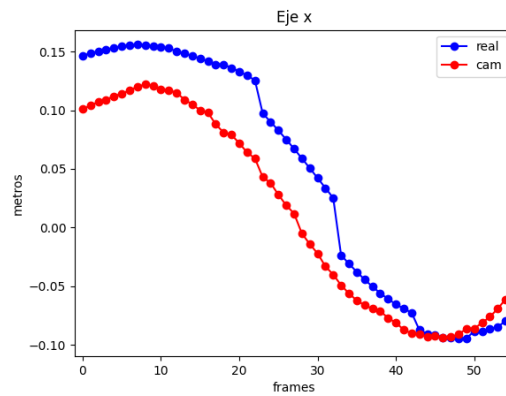
En los gráficos de la figura 6.4 se puede visualizar que es posible hacer un seguimiento a la trayectoria real, sin embargo, se presentan algunas inestabilidades en el valor de profundidad, lo que explica la dispersión visual de los puntos estimados alrededor de la trayectoria real.

Dado el comportamiento de la trayectoria estimada por los 3 modelos, se puede observar que el mejor comportamiento relativo se puede obtener a través de la estimación de coordenadas utilizando segmentación de color, mientras que el modelo con un peor desempeño parece ser utilizando el modelo de detección de YOLO.

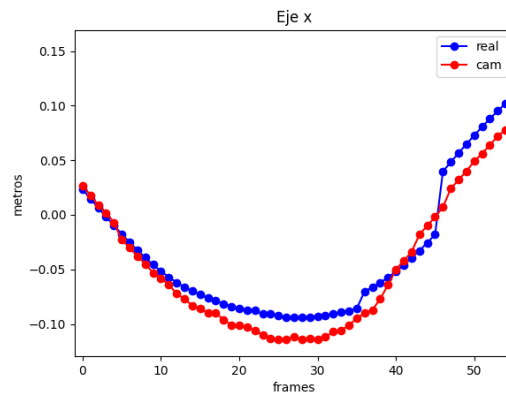
En función de poder evaluar este experimento y tener una forma visual del comportamiento en la estimación, se realizan gráficos para cada eje de coordenadas utilizando los modelos desarrollados.



(a) Resultados teóricos y experimentales segmentación de color

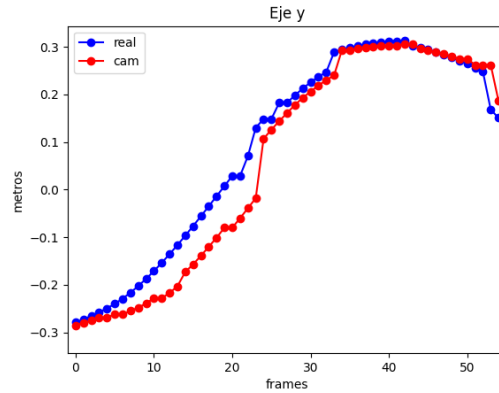


(b) Resultados teóricos y experimentales detección YOLO

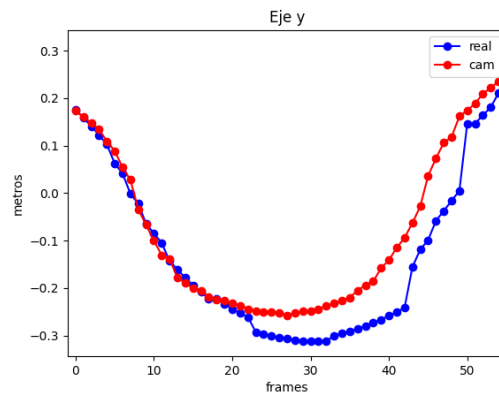


(c) Resultados teóricos y experimentales segmentación YOLO

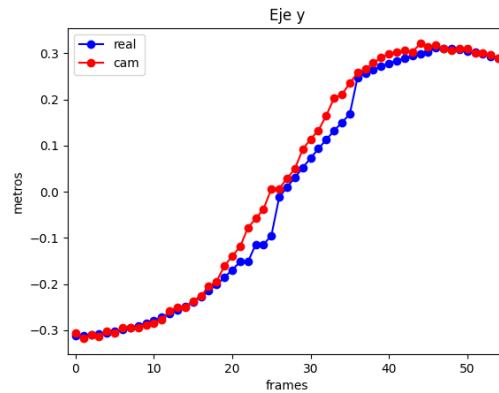
Figura 6.5: Comparación de estimación de trayectoria para cada técnica de reconstrucción, en torno al eje x.



(a) Resultados teóricos y experimentales segmentación de color

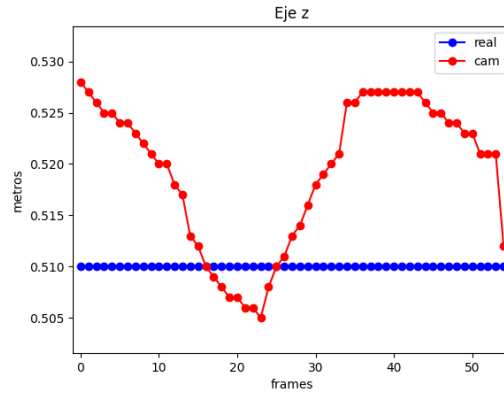


(b) Resultados teóricos y experimentales detección YOLO

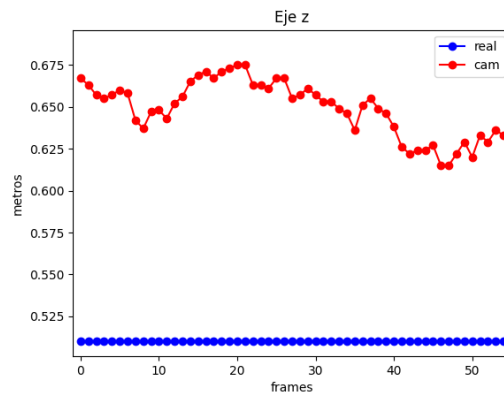


(c) Resultados teóricos y experimentales segmentación YOLO

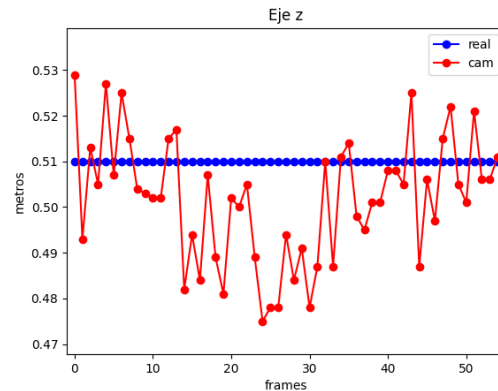
Figura 6.6: Comparación de estimación de trayectoria para cada técnica de reconstrucción, en torno al eje y.



(a) Resultados teóricos y experimentales segmentación de color



(b) Resultados teóricos y experimentales detección YOLO



(c) Resultados teóricos y experimentales segmentación YOLO

Figura 6.7: Comparación de estimación de trayectoria para cada técnica de reconstrucción, en torno al eje z.

Con esto, es posible notar en las figuras 6.5 y 6.6 que, tanto las coordenadas x como las y, los modelos basados en segmentación tienden a presentar un mejor seguimiento de la trayectoria. Particularmente en los gráficos de la figura 6.7 es posible notar la dificultad de estimar un único valor dado un objeto en movimiento.

Utilizando las métricas de evaluación, se construye la siguiente tabla de comparación que resume el funcionamiento:

Modelo	ECM x [cm]	ECM y [cm]	ECM z [cm]	ECM total [cm]	Error % x	Error % y	Error % z	Error % total
SegColor	0.05142	0.13608	0.0157	0.2035	1.76 %	2.38 %	1.0918 %	1.744 %
DetYOLO	0.12188	0.51966	1.9075	2.5488	2.84 %	6.12 %	13.7 %	7.55 %
SegYOLO	0.07236	0.19595	0.019595	0.28737	2.212 %	3.2469 %	1.1133 %	2.1909 %

Tabla 6.1: Métricas obtenidas en el experimento

A través de la tabla 6.1, es posible resumir los resultados obtenidos de las métricas, entregando datos sobre el error de propagación que existe en este proceso de estimación. Se observa un comportamiento que entrega un orden en el funcionamiento de cada modelo, donde la segmentación por color se posiciona como el modelo más efectivo en este experimento, seguido por la segmentación de YOLO, y finalmente, la detección de YOLO. Es importante destacar que la estimación de coordenadas en el eje z muestra un comportamiento inestable en comparación con los otros dos ejes, lo que refleja la complejidad de la tarea que se enfrentan los modelos respecto al cálculo de profundidad mediante una cámara. Es relevante mencionar que el modelo de segmentación por color está diseñado específicamente para las condiciones de simulación, lo que se traduce en un rendimiento uniforme y estimaciones más precisas. A pesar de ello, se puede tener un alto nivel de confianza en la segmentación de YOLO.

6.1.3. Trayectoria con variación de profundidad

Para establecer un análisis en el funcionamiento de la estimación de coordenadas en 3 dimensiones, es necesario estudiar los resultados que se pueden obtener al variar las 3 coordenadas. Para ello, se realiza un experimento utilizando la trayectoria anteriormente descrita, pero variando los parámetros del eje z.

```

1  <trajectory id="0" type="square">
2      <waypoint>
3          <time>0.0</time>
4          <pose>0.12 0.25 0.51 0 0 0</pose>
5      </waypoint>
6      <waypoint>
7          <time>1.0</time>
8          <pose>0.12 -0.25 0.6 0 0 0</pose>
9      </waypoint>
10     <waypoint>
11         <time>2.0</time>
12         <pose>-0.05 -0.25 0.51 0 0 0</pose>
13     </waypoint>
14     <waypoint>
15         <time>3.0</time>
16         <pose>-0.05 0.25 0.6 0 0 0</pose>
17     </waypoint>
18     <waypoint>
19         <time>4.0</time>
20         <pose>0.12 0.25 0.51 0 0 0</pose>
21     </waypoint>
22 </trajectory>

```

Figura 6.8: Trayectoria establecida para variación de profundidad.

Con esto, es posible obtener las siguientes trayectorias para los modelos de estimación:

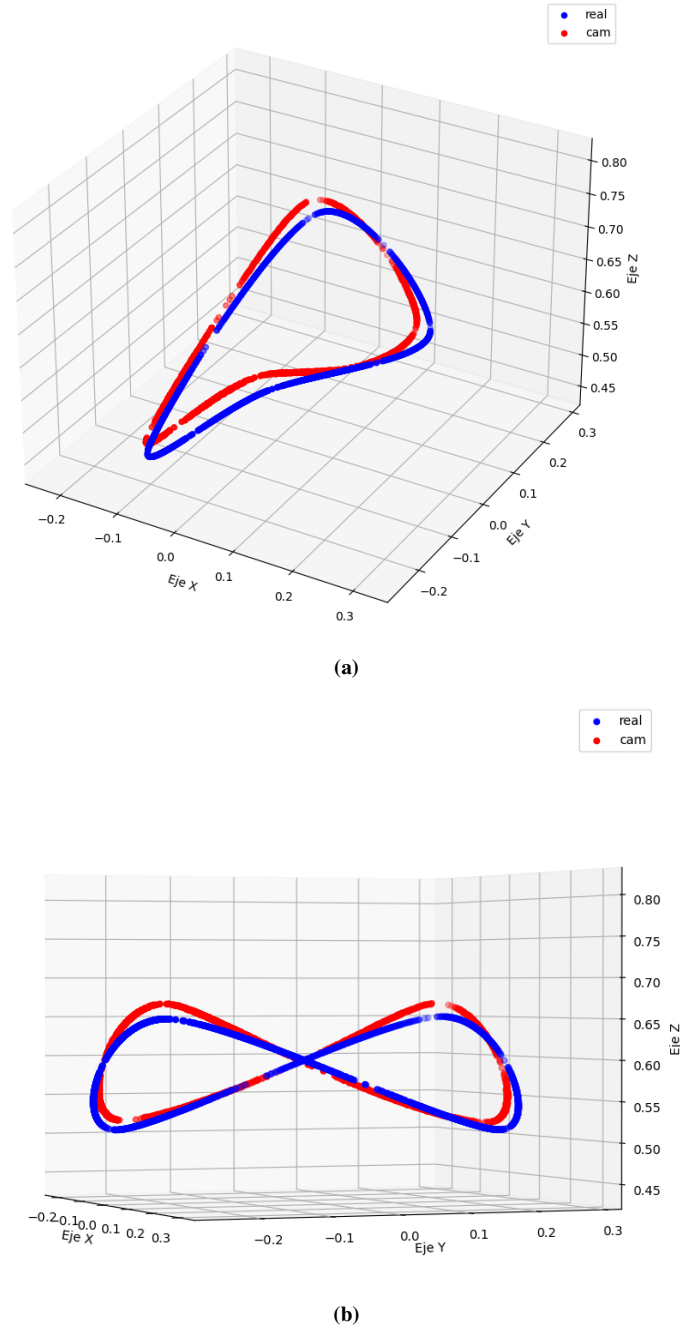
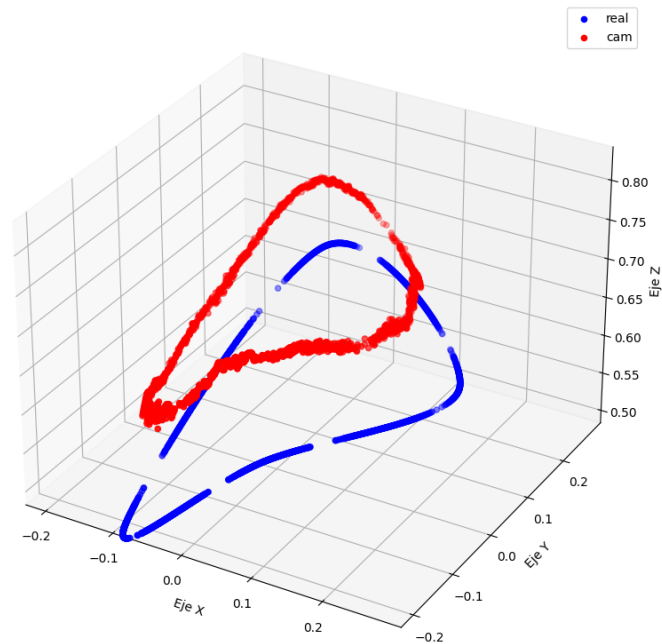
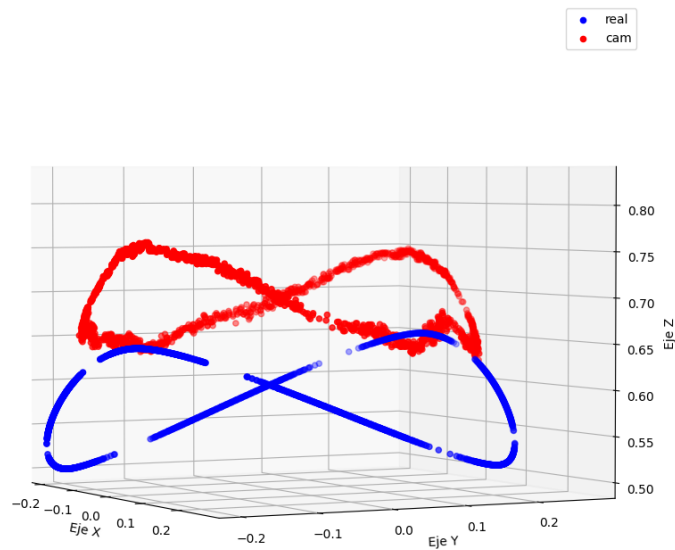


Figura 6.9: Las figuras 6.9a y 6.9b corresponden a la trayectoria real y estimada del objeto utilizando la técnica de **detección por segmentación de color**.

Es posible visualizar en los gráficos de la figura 6.9 que, si bien la estimación parece estar ligeramente corrida, sigue presentando un comportamiento uniforme y muy similar a la trayectoria real del objeto.



(a)



(b)

Figura 6.10: Las figuras 6.10a y 6.10b corresponden a la trayectoria real y estimada del objeto utilizando la técnica de **detección y clasificación utilizando YOLO**.

Para los gráficos de la figura 6.10 es posible notar el gran error existente en la coordenada z . Lo que presupone un mal funcionamiento en detección de profundidad particularmente para esta herramienta.

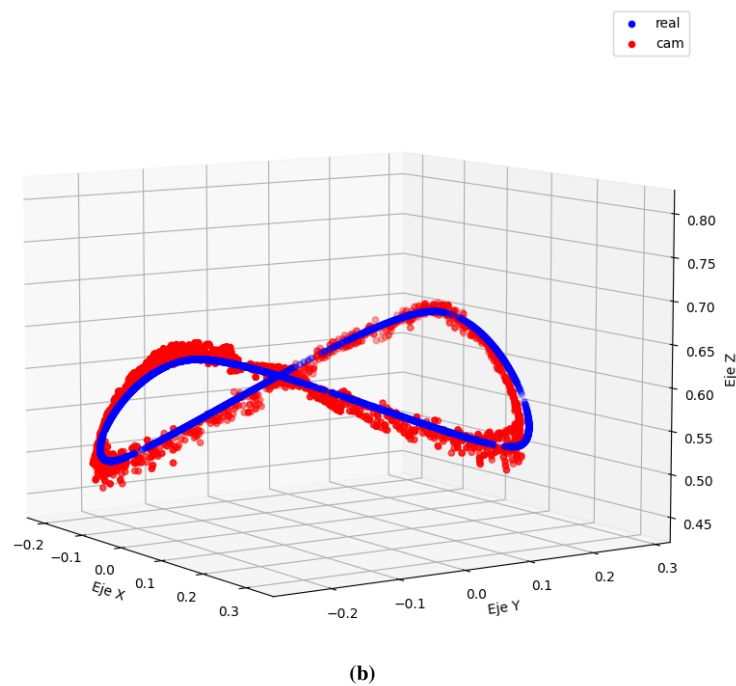
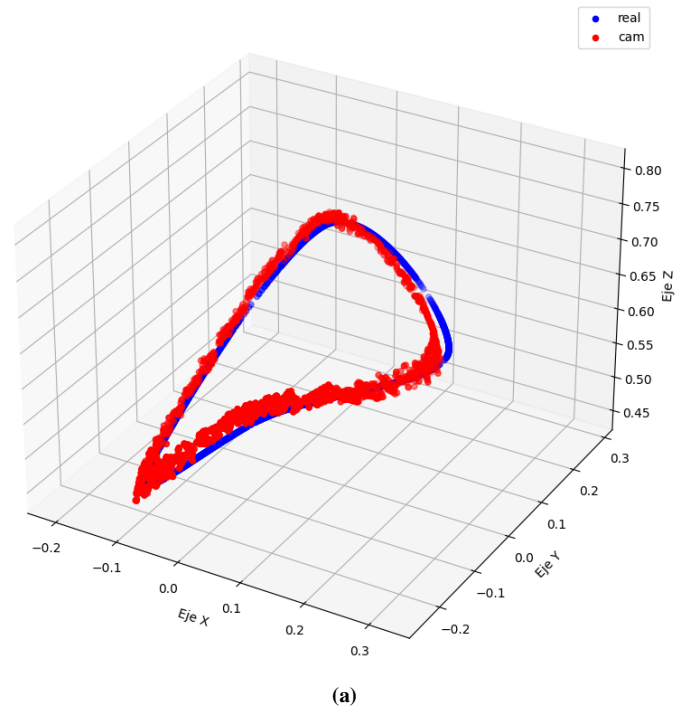
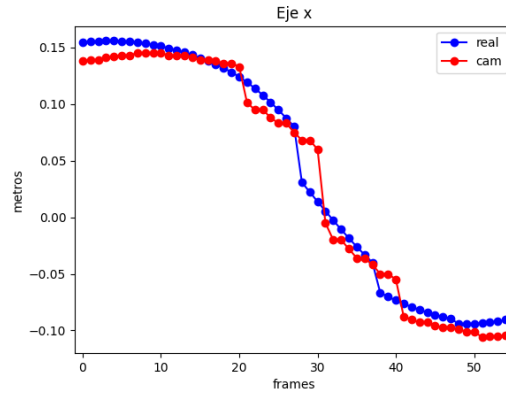


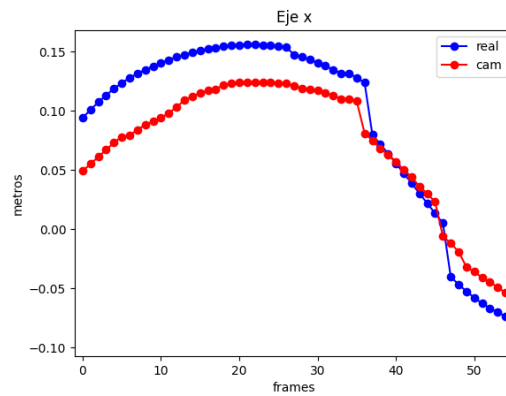
Figura 6.11: Las figuras 6.11a y 6.11b corresponden a la trayectoria real y estimada del objeto utilizando la técnica de **segmentación utilizando YOLO**.

En los gráficos de la figura 6.11 se puede visualizar un seguimiento cercano la trayectoria real del objeto, aunque presenta una dispersión de los datos ligeramente mayor a la segmentación de color.

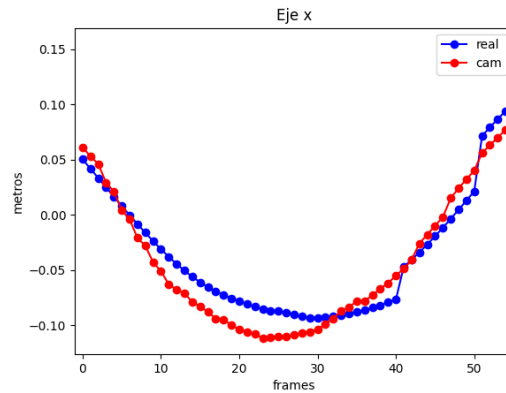
Se busca el análisis por coordenada en torno a la variación existente entre las coordenadas reales y las estimadas. Debido a esto, se construyen las siguientes gráficas para cada eje del comportamiento durante el experimento.



(a) Resultados teóricos y experimentales segmentación de color

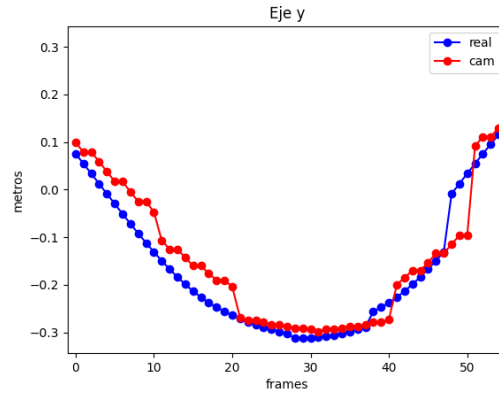


(b) Resultados teóricos y experimentales detección YOLO

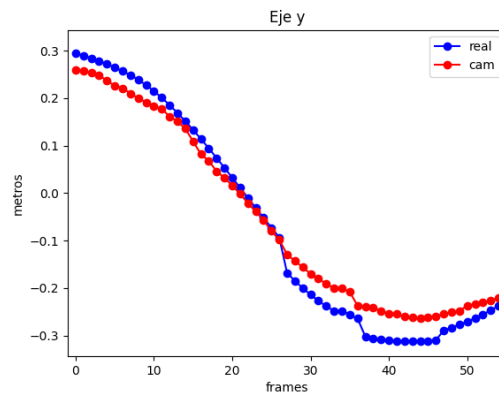


(c) Resultados teóricos y experimentales segmentación YOLO

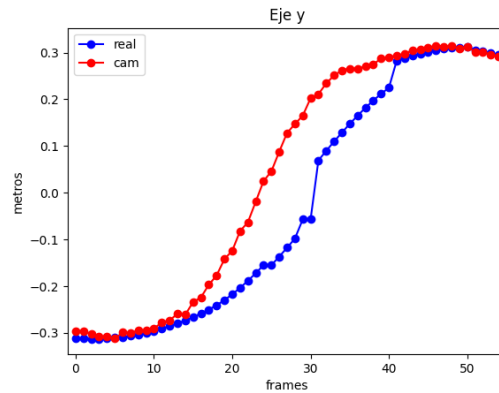
Figura 6.12: Comparación de estimación de trayectoria para cada técnica de reconstrucción, en torno al eje x.



(a) Resultados teóricos y experimentales segmentación de color

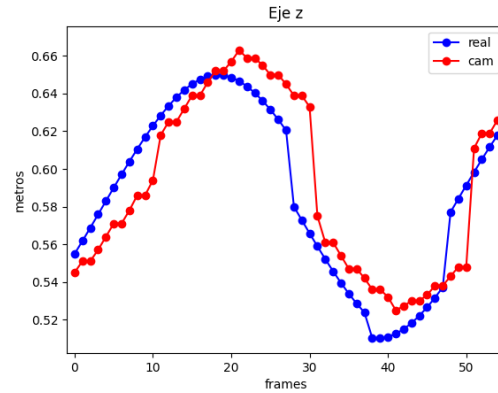


(b) Resultados teóricos y experimentales detección YOLO

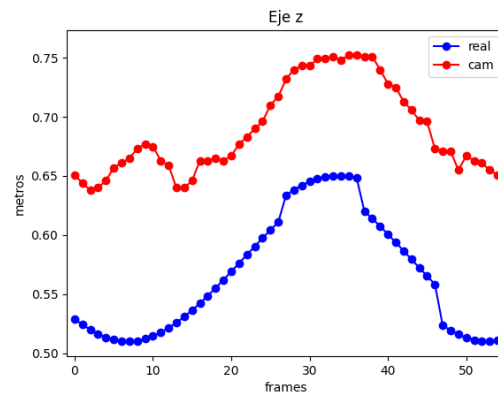


(c) Resultados teóricos y experimentales segmentación YOLO

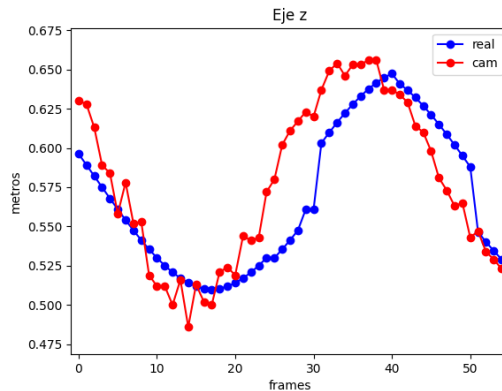
Figura 6.13: Comparación de estimación de trayectoria para cada técnica de reconstrucción, en torno al eje y.



(a) Resultados teóricos y experimentales segmentación de color



(b) Resultados teóricos y experimentales detección YOLO



(c) Resultados teóricos y experimentales segmentación YOLO

Figura 6.14: Comparación de estimación de trayectoria para cada técnica de reconstrucción, en torno al eje z.

Para los gráficos presentes en la figura 6.12 es posible notar un correcto funcionamiento en el seguimiento de la trayectoria para los modelos de segmentación. Por otro lado, en la figura 6.13 es posible notar que los 3 modelos solo varían ligeramente, siendo el caso en que más se parecen visualmente sus comportamientos. Finalmente, en la figura 6.14 se puede notar nuevamente que, para los modelos que utilizan segmentación, se les facilita la tarea de estimar profundidad, a diferencia del modelo de detección de YOLO.

Utilizando las métricas de evaluación, se construye la siguiente tabla de comparación que resume el funcionamiento:

<i>Modelo</i>	<i>ECM x [cm]</i>	<i>ECM y [cm]</i>	<i>ECM z [cm]</i>	<i>ECM total [cm]</i>	<i>Error % x</i>	<i>Error % y</i>	<i>Error % z</i>	<i>Error % total</i>
<i>SegColor</i>	0.04241	0.13847	0.04698	0.2278	1.648 %	2.532 %	1.637 %	1.939 %
<i>DetYOLO</i>	0.08822	0.29653	1.30264	1.6874	2.45 %	4.91 %	11.125 %	6.162 %
<i>SegYOLO</i>	0.06736	0.21772	0.06185	0.3469	2.097 %	3.38 %	2.09 %	2.525 %

Tabla 6.2: Métricas obtenidas en el experimento

A través de la tabla 6.2, se presentan los resultados de las métricas establecidas, los cuales indican que el método más efectivo para la tarea de estimación es la segmentación por color, seguido por el modelo YOLO para segmentación. Por otro lado, el rendimiento más bajo se observa en el modelo YOLO para detección. Estos resultados sugieren que, debido a la variabilidad en el eje z, los modelos basados en segmentación muestran un mejor seguimiento de la trayectoria en comparación con la detección a través de YOLO, que nuevamente enfrenta dificultades para estimar la profundidad. Es importante notar también que el error total de los modelos que aplican segmentación no presentan una gran diferencia, donde la arquitectura de YOLO presenta una ventana de mejora, a través de los datos de entrenamiento.

6.2. Evaluación tarea detección en robot iCub

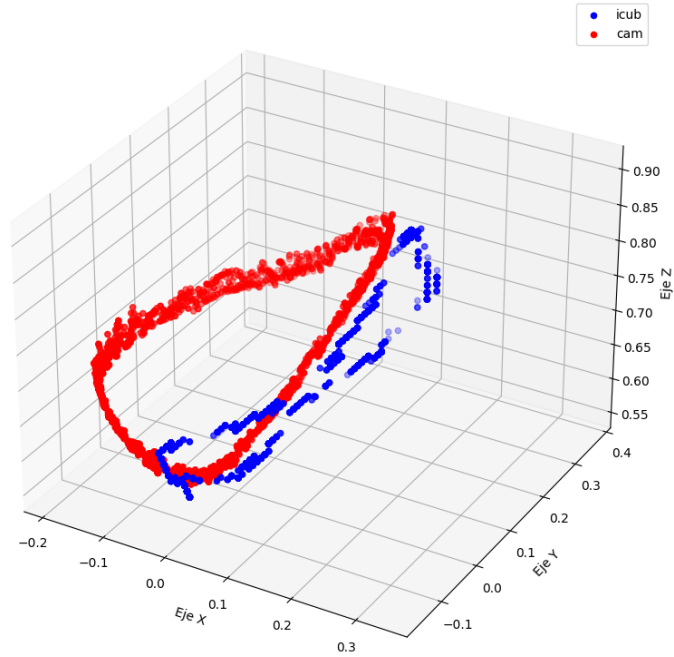
Una característica importante que se espera de la plataforma es establecer un criterio de comparación que pueda proporcionar información sobre el desempeño del iCub en tareas relacionadas con la visión. En este caso, como se ha mencionado en incisos anteriores, el punto de comparación se establece bajo la referencia a la cámara, lo que implica evaluar el comportamiento del robot para obtener la posición en relación a esta referencia. Para llevar a cabo esta estudio de comparación, se realiza un experimento en el que ambos observadores recopilarán datos de estimación. La cámara implementará el modelo que utiliza la arquitectura YOLO para segmentación, dado que su rendimiento es muy cercano al del mejor modelo encontrado, pero que también proporciona información adicional relevante, como la diferenciación de instancias y la clasificación del objeto. Por otro lado, el iCub será evaluado utilizando los tres modelos disponibles, con la esperanza de evaluar su desempeño mediante criterios de error. Además, dado que es importante establecer un criterio asociado al comportamiento de la observación, se puede emplear el diagrama de residuos para analizar la tendencia.

Este experimento se realiza a través de la medición de trayectoria variando los 3 ejes de coordenadas, similar al inciso anterior, con lo que se obtienen los siguientes resultados del experimento reflejados en la siguiente tabla:

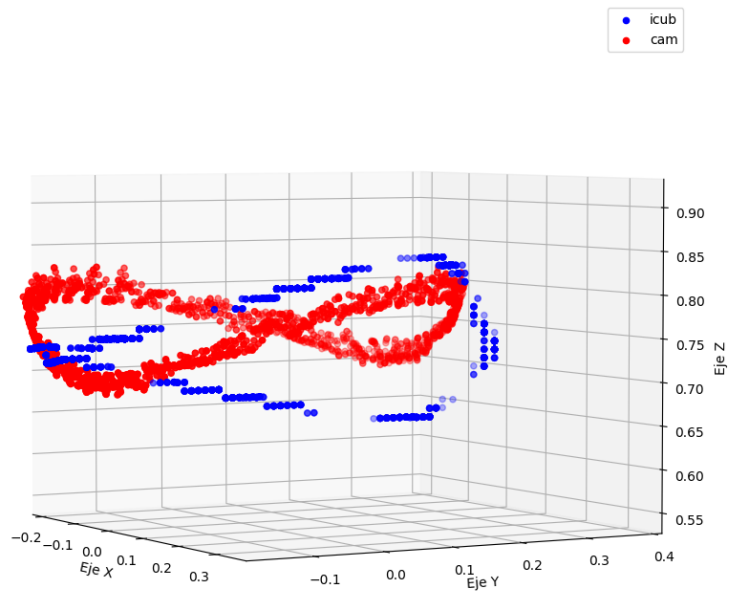
<i>Modelo</i>	<i>ECM x [cm]</i>	<i>ECM y [cm]</i>	<i>ECM z [cm]</i>	<i>ECM total [cm]</i>	<i>Error % x</i>	<i>Error % y</i>	<i>Error % z</i>	<i>Error % total</i>
<i>SegColor</i>	2.7764499	0.4472841	0.57303385	3.796767	13.768 %	5.211 %	5.94 %	8.306 %
<i>DetYOLO</i>	2.38336	1.126599	0.457074	3.96704	13.506 %	8.5301 %	4.790 %	8.9425 %
<i>SegYOLO</i>	2.801	1.92477	0.2620	4.9887	14.32 %	11.809 %	4.0714 %	10.07 %

Tabla 6.3: Métricas obtenidas en el experimento

Para representar la visualización de los datos obtenidos que se usaron para las métricas calculadas, se realizan los siguientes gráficos que dibujan la trayectoria estimada por cada observador.

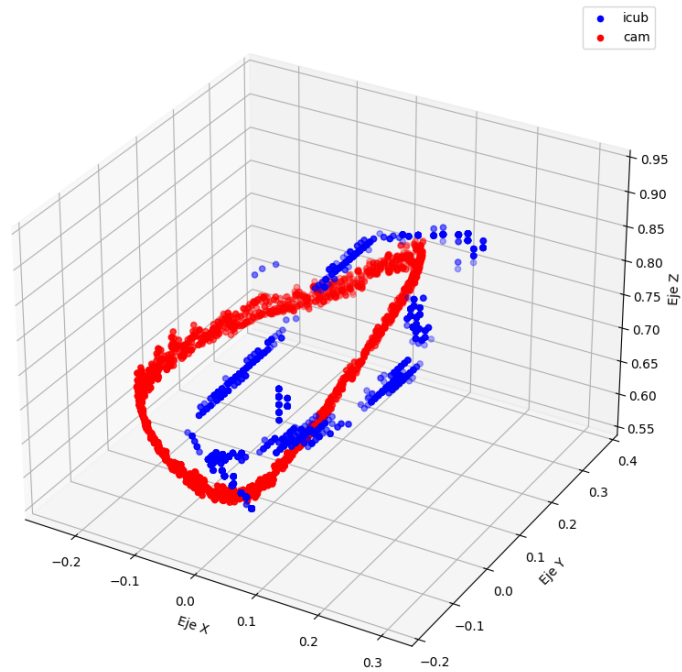


(a)

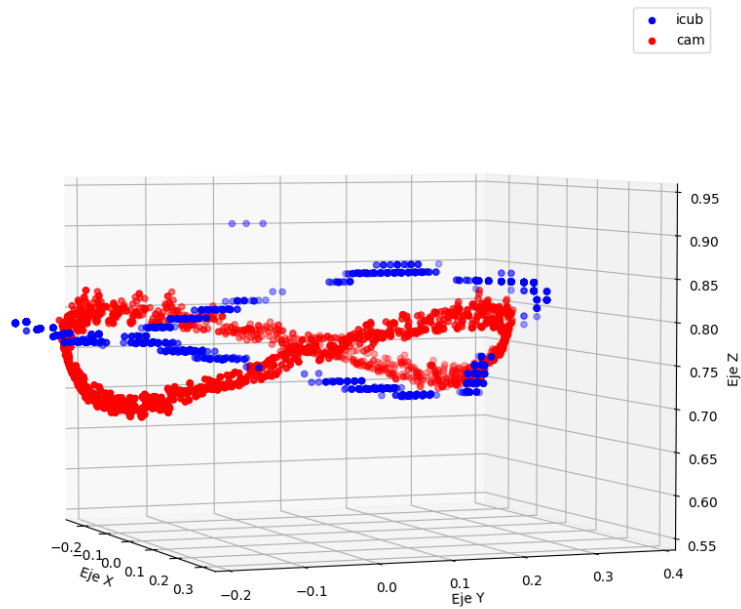


(b)

Figura 6.15: Trayectoria visualizada por iCub con **segmentación de color**, y cámara.



(a)



(b)

Figura 6.16: Trayectoria visualizada por iCub con **detección y clasificación de YOLO**, y cámara.

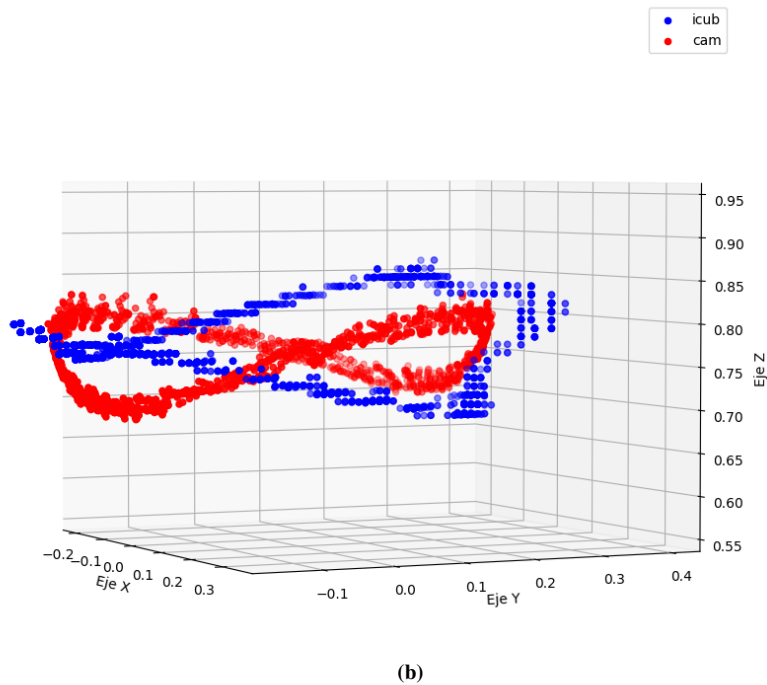
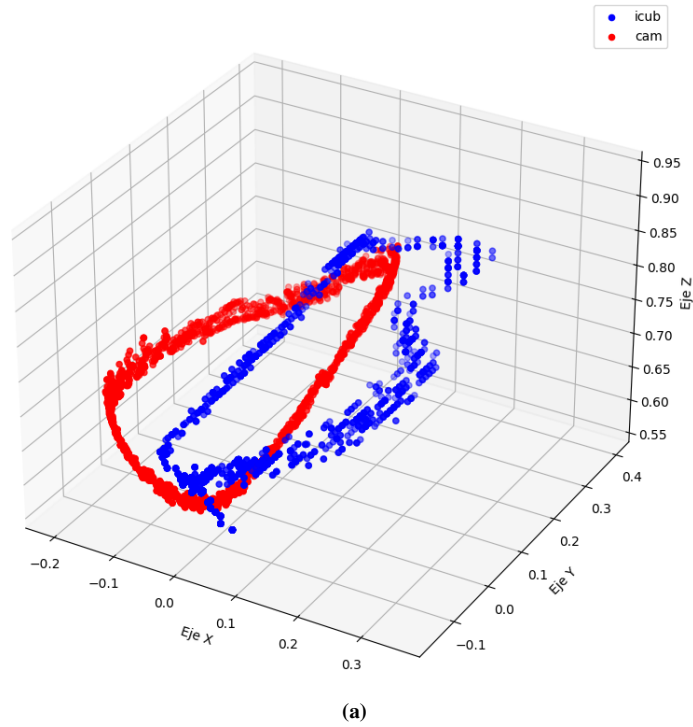


Figura 6.17: Trayectoria visualizada por iCub con **segmentación de instancia YOLO**, y cámara

Dadas las gráficas presentadas, se pueden observar las trayectorias trazadas tanto para el Ground Truth proporcionado por la cámara como para la posición estimada por el iCub, la cual se proyecta a la referencia de la cámara.

Luego de llevar cabo los experimentos implementando los modelos para la estimación de coordenadas en el iCub, se puede observar que la segmentación por color logra el error más bajo y un seguimiento más estable. Por otro lado, es importante notar que, en este caso, la segmentación generada por la arquitectura YOLO obtiene el error más alto y por consecuencia, el peor rendimiento. Este resultado es importante de mencionar para el análisis ya que, al utilizar un conjunto de datos que no incluye imágenes desde la perspectiva del iCub durante la simulación y al tener una dependencia directa del área observada en la segmentación, el modelo experimenta una pérdida de información debido a la distancia a la que se encuentra la pelota en la imagen. Sin embargo, este resultado abre una posibilidad de mejora en la reestructuración del conjunto de datos.

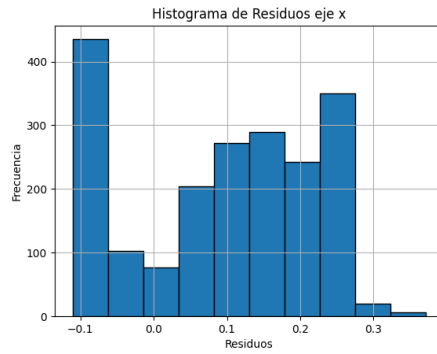
Por otro lado, el modelo basado en la detección de la arquitectura YOLO demuestra una mayor estabilidad en la estimación de las coordenadas en comparación con la segmentación de la misma arquitectura. Esto resulta interesante debido que, al simplificar el problema para trabajar únicamente en una detección que analiza todo en función del tamaño del bounding box observado, la tarea de seguimiento del objeto se vuelve más sencilla, lo que posiciona a este modelo en una posición más favorable que la segmentación de YOLO.

Dada la importancia de analizar el desempeño del iCub, y como se presentó previamente, se generan gráficos de residuos con el propósito de contar con una herramienta visual que muestre la distribución de los errores obtenidos en cada dato estimado, por cada eje de coordenadas. Esto se aplica para cada modelo implementado en el iCub.

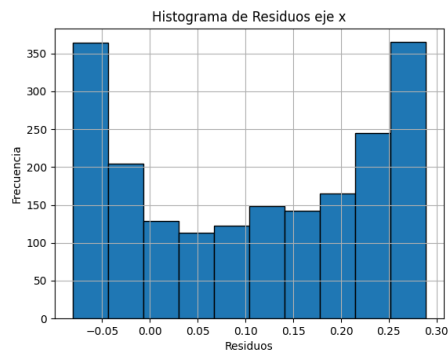
$$r = \hat{x} - x \quad (6.5)$$

- r : Residuo
- $\hat{x}$: Valor estimado. Se considera el valor del iCub.
- x : Valor real. Se considera el valor de la cámara.

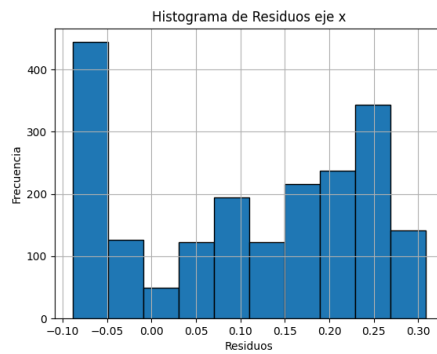
Los gráficos de residuos se generan mediante la representación de los errores residuales de cada estimación realizada por el iCub en relación con las coordenadas reales. Este residuo es resultado de la resta entre el dato estimado y el dato real. La interpretación de estos datos se realizará con el fin de analizar el patrón de distribución de los residuos, buscando una distribución cercana a la distribución normal, lo que sugiere que los errores asociados en dichas coordenadas son bajos. Es importante tener en cuenta que, al ser un gráfico de distribución de errores, se espera que la mayoría de los errores estén concentrados alrededor del centro en 0.



(a) Residuos eje x segColor

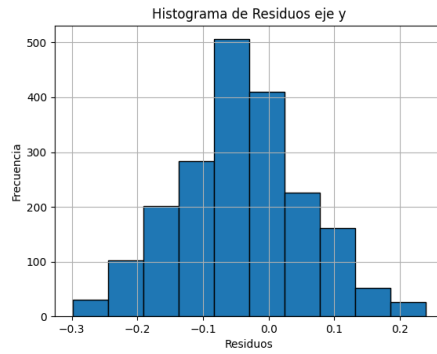


(b) Residuos eje x detYOLO

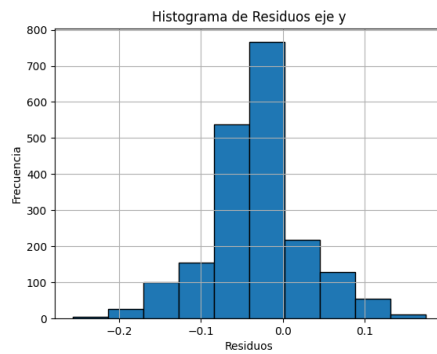


(c) Residuos eje x segYOLO

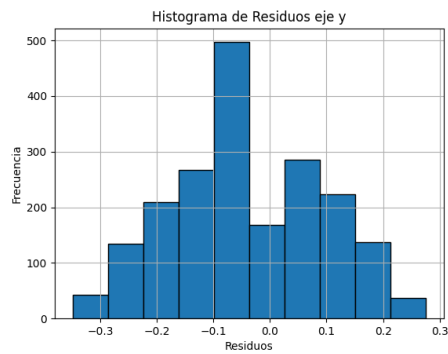
Figura 6.18: Histogramas de residuo en coordenadas x.



(a) Residuos eje y segColor

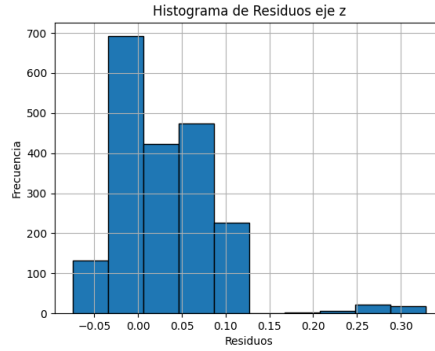


(b) Residuos eje y detYOLO

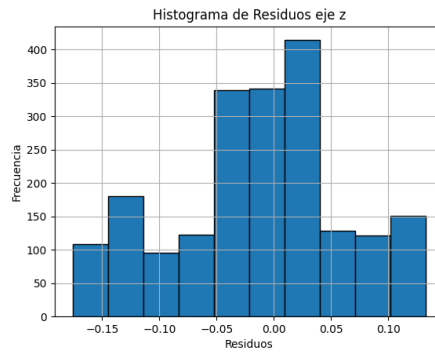


(c) Residuos eje z segYOLO

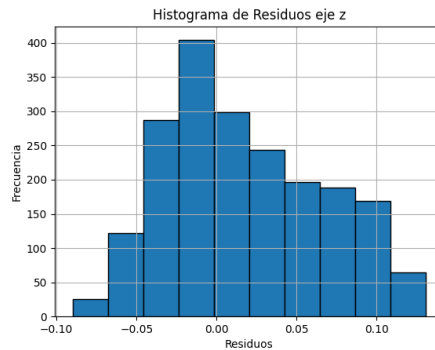
Figura 6.19: Histograma de residuo en coordenadas y.



(a) Residuos eje z segColor



(b) Residuos eje z detYOLO



(c) Residuos eje z segYOLO

Figura 6.20: Histograma de residuo en coordenadas z.

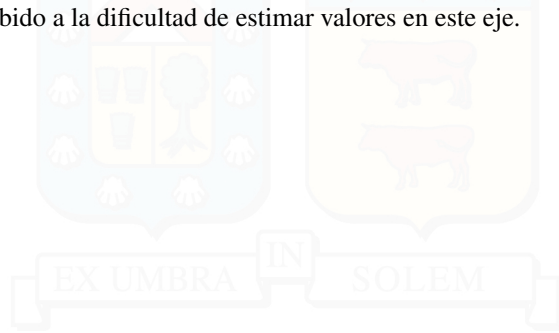
A través de la tabla 6.3 es posible suponer los resultados que se obtienen en las gráficas, sin embargo es interesante observar sus patrones.

En torno a las gráficas del eje x de coordenadas dadas en la figura 6.18, en primera instancia se puede observar que no presentan la característica principal esperada, la cual es estar centradas en 0. Esto es importante debido a que, idealmente, se espera que el error tienda a ser mínimo, sin embargo, se puede visibilizar cómo los datos estimados tienden a ser mayores a los datos captados por la cámara, por lo que se agrupan mayormente en los valores positivos de la gráfica.

Por otro lado, es posible notar en los gráficos de la figura 6.19 que, tanto la segmentación de color como la detección de YOLO presentan una distribución similar, que agrupa sus resultados en valores cercanos al mínimo. Es importante mencionar que, a pesar de esto, presenta una distribución que no se asemeja lo suficiente a una distribución

normal, esto puede reflejarse en los valores de error porcentual en la tabla 6.3, donde estos 2 algoritmos presentan valores cercanos. Finalmente, los residuos obtenidos por la segmentación de YOLO, en torno al eje y, resulta tener bastante presencia en valores positivos lejanos a 0, lo que se explica con el alto error porcentual obtenido previamente para este eje.

En torno al eje z, y como es presentado previamente en los resultados de error porcentual, los datos se agrupan bajo un comportamiento similar, aunque ninguno presenta estar centrado en 0. Esto se explica debido a que los errores para este eje en los 3 algoritmos se encuentran cercanos entre si, por lo que tienen un comportamiento similar. Es importante notar que, para 2 modelos desarrollados bajo la arquitectura de YOLO, presentan su menor error de estimación en este eje, lo cual es curioso debido a la dificultad de estimar valores en este eje.



7 | Conclusiones y trabajo futuro

7.1. Comparación de resultados y conclusiones

En el presente trabajo, se propone una solución ante la necesidad de contar con una herramienta de evaluación mediante la generación de datos con la confiabilidad suficiente para poder ser utilizados como Ground Truth en tareas de detección para el robot iCub. Para lograr esto, se planteó una solución basada en la utilización de una sola cámara para la obtención de información de posición en el espacio, en busca de una implementación de bajo costo monetario y que cumpla con los requerimientos necesarios. Dado los resultados obtenidos, es posible hacer las siguientes conclusiones:

- Dada la fase de análisis de precisión en los modelos obtenidos, se observa que las soluciones en torno a la segmentación ofrecen la información necesaria para poder hacer estimaciones más certeras, lo que sugiere que, en el contexto de la estimación de parámetros de posición utilizando una sola cámara, tiene sentido implementar algoritmos que realicen esta tarea de manera eficiente.
- En una comparativa entre ambos algoritmos de segmentación, debido a la baja diferencia existente entre ambos modelos, es posible concluir que una solución basada en una segmentación de instancia proporcionada por YOLO es la mejor decisión. El respaldo de esto viene de la mano con la cantidad de información del contexto que entrega el modelo ya que, no solo detecta su posición, sino que es posible detectar la clase del objeto y la cantidad de instancias del objeto que están presentes. Esta característica es particularmente interesante de abordar en el contexto del desarrollo de tareas más complejas asociadas al robot iCub, donde se espera poder utilizar esta plataforma para el desarrollo de aplicaciones en este robot. En este mismo sentido, si bien la solución basada en segmentación de color resulta ser una solución más sencilla y de bajo costo computacional, es importante destacar que el caso de fallo más común será en cuanto se trabaje con más de un objeto, donde no es posible hacer una diferenciación entre estos. Además, al depender de un rango de color visible, es altamente sensible a las condiciones del entorno, como la posición de la fuente de luz. Esto es principalmente complejo en experimentos que se hagan con el robot real y no en una simulación.
- Finalmente, se puede concluir que es posible evaluar al iCub a través de estos datos, proporcionando no solo información como agente externo, sino que también incorporar estos datos como fuente de entrenamiento para tareas más complejas desarrolladas para este robot. Dado los datos obtenidos por las métricas y los experimentos, es posible tener datos confiables para evaluación. Sin embargo, las métricas y experimentos fueron presentados bajo tareas muy simplificadas para el iCub, por lo que se estima prudente validar este modelo, buscando complejizar los movimientos e interacciones que realice el robot.

Dados los resultados de evaluación que fueron recopilados a través de esta plataforma, se puede afirmar que soluciona la falta de un entorno modular de evaluación para tareas de visión, además de ser una plataforma de bajo costo económico y sencilla de implementar.

7.2. Trabajo futuro

Debido a los análisis de resultados, se abren nuevas ventanas de mejora respecto a la plataforma obtenidas, además de la utilización de estos datos como entrada a otros experimentos asociados a la visión del robot iCub. En este contexto, se presentan los siguientes trabajos a futuro.

- Aumentación de los datos para la segmentación de instancia en simulación y comparación con otros métodos de segmentación semántica.
- Incorporación de tecnologías más sofisticadas como cámaras con sensor de profundidad o la utilización de un modelo de visión estéreo. Con la finalidad de robustecer los datos recopilados del entorno.
- Desarrollo de entorno estandarizado utilizando ROS para trabajos de aprendizaje profundo, en función de establecer una plataforma de entrenamiento simulado utilizando los datos recopilados como información fundamental.
- Realizar pruebas con el robot real, para validar el funcionamiento de la estimación de coordenadas en un entorno real con más variables asociadas a la luz.

Bibliografía

- [1] Richard Hartley y Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2003.
- [2] Joseph Redmon, Santosh Divvala, Ross Girshick y Ali Farhadi. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. arXiv: [1506.02640](https://arxiv.org/abs/1506.02640) [cs.CV].
- [3] G. Jocher, A. Chaurasia y J. Qiu. *Ultralytics YOLO (Version 8.0.0)*. Computer software. Consultado el 23 de febrero de 2024. 2023. URL: <https://github.com/ultralytics/ultralytics>.
- [4] Range King. *Brief summary of YOLOv8 model structure - Issue #189*. <https://github.com/ultralytics/ultralytics/issues/189>. 2023.
- [5] Sambasivarao. K. *Non-Maximum Suppression (NMS)*. 1 Octubre, 2019. URL: <https://towardsdatascience.com/non-maximum-suppression-nms-93ce178e177c>.
- [6] Evan Shelhamer, Jonathan Long y Trevor Darrell. *Fully Convolutional Networks for Semantic Segmentation*. 2016. arXiv: [1605.06211](https://arxiv.org/abs/1605.06211) [cs.CV].
- [7] Mrinal Walia. *Semantic Segmentation vs. Instance Segmentation: Explained*. <https://blog.roboflow.com/difference-semantic-segmentation-instance-segmentation/>. 2022.
- [8] iCub Tech Docs, Istituto Italiano Di Tecnologia. *iCub*. <https://icub.iit.it>.
- [9] Yet Another Robot Platform. *YARP*. <http://wiki.icub.org/wiki/YARP>.
- [10] The Robotology Community. *Robotology Superbuild Repository*. <https://github.com/robotology/robotology-superbuild>.
- [11] Open Source Robotics Foundation. *Gazebo*. <https://classic.gazebosim.org/>.
- [12] Roboflow. *Pelota Roja iCub*. <https://universe.roboflow.com/datasets-proyectos/pelota-roja-icub-5/dataset/3>.
- [13] Roboflow. *Pelota Roja Segmentation*. <https://app.roboflow.com/datasets-proyectos/pelota-roja-segmentation-2/3>.
- [14] Wikimedia Commons. *RGB color solid cube.png*. https://commons.wikimedia.org/wiki/File:RGB_color_solid_cube.png#mw-jump-to-license.
- [15] Wikimedia Commons. *HSV color solid cylinder.png*. https://en.wikipedia.org/wiki/File:HSV_color_solid_cylinder.png.
- [16] cvzone. *cvzone GitHub Repository*. <https://github.com/cvzone/cvzone>.
- [17] AiPhile. *Distance Estimation with Single Camera (OpenCV Python)*. <https://medium.com/mlearning-ai/distance-estimation-with-single-camera-opencv-python-298a96383c2b>. 18 diciembre, 2021.
- [18] *OpenCV Tutorial: Camera Calibration using OpenCV*. https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html.
- [19] A. Rocchi et al. *Simulation and Reality: Experience with Gazebo in the iCub Facility*. <http://www.icub.org/other/files/rochi-gazebo.pdf>. 18 noviembre, 2014.
- [20] *Gazebo Tutorial: Actor Simulation*. <https://classic.gazebosim.org/tutorials?tut=actor>.

- [21] *Gazebo Tutorial: Installing Gazebo with ROS*. https://classic.gazebosim.org/tutorials?tut=ros_installing&cat=connect_ros.
- [22] *Gazebo Tutorial: ROS Launch*. https://classic.gazebosim.org/tutorials?tut=ros_roslaunch&cat=connect_ros.
- [23] Aqeel Anwar. *What are Intrinsic and Extrinsic Camera Parameters in Computer Vision?* <https://towardsdatascience.com/what-are-intrinsic-and-extrinsic-camera-parameters-in-computer-vision-7071b72fb8ec>. 1 marzo, 2022.
- [24] iCub Tech Docs, Istituto Italiano Di Tecnologia. *iCub Tech Documentation*. https://icub-tech-iit.github.io/documentation/icub_robot_calibration/icub-stereo-calib/.

