Universidad Politécnica
de Madrid

Escuela Técnica Superior de
Ingenieros Informáticos

Máster Universitario en Inteligencia Artificial

Trabajo Fin de Máster

Exploración Autónoma en Interiores para el robot Spot basado en la red Yolo

Autor: José David Loja Romero

Tutor: Javier de Lope Asiaín

Madrid, mayo 2022

Este Trabajo Fin de Máster se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Máster
Máster Universitario en Inteligencia Artificial
Título: Exploración Autónoma en Interiores para el robot Spot basado en la red Yolo
mayo 2022

Autor: José David Loja Romero

Tutor:
Javier de Lope Asiaín
Departamento de Inteligencia Artificial
ETSI Informáticos
Universidad Politécnica de Madrid

Resumen

La exploración autónoma en robots móviles es un área de estudio interesante, debido a las distintas problemáticas como lo es la navegación sobre entornos accidentados, decisión de rutas a tomar y la capacidad de identificar zonas de interés.

El presente trabajo describe el desarrollo de un módulo de exploración autónoma para interiores desplegada sobre una plataforma dinámica, como lo es el robot Spot de Boston Dynamics de la empresa Alisys.

Mediante el uso de la red Yolo como bloque de detección para la detección de esquinas a nivel del suelo, se despliega el sistema para identificar zonas de libre tránsito en tiempo real mediante el uso de las cinco cámaras a bordo del robot, obteniendo una perspectiva global del entorno.

El documento detalla la arquitectura planteada que contiene: una clase de alto nivel desarrollada para acceder a las funcionalidades de Spot, la generación de la base de datos, el modelo entrenado, los algoritmos de exploración y las pruebas ejecutadas.

Abstract

Autonomous exploration in mobile robots is an interesting area of study, it consists of different problems such as navigation over rugged environments, decision of routes to take and the ability to identify areas of interest.

The present work describes the development of an autonomous exploration module for interiors deployed on a dynamic platform, such as the Spot robot from Boston Dynamics of the Alisys company.

Using the Yolo network as an inference engine for the detection of corners at ground level, the system is deployed to identify free transit areas in real time using the five cameras on board the robot, obtaining a global perspective of the environment.

The document details the proposed architecture that contains: a high-level class developed to access the Spot functionalities, the generation of the database, the trained model, the exploration algorithms, and the tests executed.

Tabla de contenidos

1 Introducción

Hoy en día el sector de la industria y la defensa están muy interesados en implementar soluciones a corto plazo desde la robótica para distintas necesidades como: aplicaciones de inspección, monitoreo, instrumentación o incluso desactivación de bombas o ingreso a zonas de alto riesgo. Por lo que, un robot es ideal para este tipo de trabajos en lo que intervienen riegos de salud para un operario o el costo de mantenimiento es más barato.

A excepción de los drones, por lo general los robots utilizados en estos campos son de tipo autónomos o semiautónomos con ruedas u orugas y tienen la desventaja de que en gran parte de los casos no son capaces de desplazarse por entornos accidentados y dinámicos. Es aquí donde se integran robots con algún mecanismo de articulaciones como patas, basados en la naturaleza. Estos tienen la capacidad de liberar pendientes, subir escalones y moverse sobre los obstáculos.

El robot de referencia de tipo cuadrúpedo es Spot de Boston Dynamics, es catalogado por el fabricante como una mula de carga o robot dinámico capaz de desplazarse por terrenos accidentados y responder a cambios del entorno.

Cabe recalcar, que este tipo de robot cuadrúpedo no es más que un dispositivo que tiene la capacidad de trasladarse de un lugar a otro y posee un desarrollo base para su funcionamiento interno. Es decir, que aún queda un largo trayecto para integrar hardware y software para generar un caso de uso real. Funcionalidades como de exploración, monitoreo, control a distancia y otras son actualmente líneas de investigación en constante desarrollo y de vital interés en el mercado.

De esta manera Alisys Digital S.L.U ha venido trabajando estos últimos 4 años en desarrollar soluciones para la industria y defensa por lo que actualmente posee una Plataforma Robótica de Teleoperación con una infraestructura dedicada que permite operar, de manera autónoma o semiautónoma, robots y dispositivos a distancia a través de internet con tecnología 5G. Es un sistema muy potente debido a que coloca una capa de usabilidad muy accesible y fácil de usar para operarios no expertos. Además, estandariza el control sobre los robots u otros dispositivos, ya que gestiona desde una misma interfaz sensores, robots dinámicos, AGV y drones. Cabe recalcar que Alisys es partner de Boston Dynamics hace dos años por lo que actualmente posee los dos modelos de robot: Explorer y Enterprise. Este último con un brazo de 8 grados de libertad.

En consecuencia, constantemente se busca desarrollar nuevas funcionalidades que permitan extender las capacidades de los robots y generar aplicaciones reales. Una de estas funcionalidades de interés es la Exploración Autónoma en Interiores.

1.1 Descripción del problema

Se plantea desarrollar un Módulo de Exploración Autónoma o MEA para robots de distinto propósito. No todos los robots dinámicos poseen la misma tecnología o hardware de fábrica por lo que el sistema debe funcionar con unos requerimientos básicos.

En consecuencia, se define que el MEA es un proyecto a largo plazo que contempla generar un servicio para procesamiento en la nube y en local, capaz de suministrar el cómputo necesario para que robots dinámicos de distintas marcas posean la capacidad de exploración autónoma.

Por lo tanto, para el presente trabajo se plantea desarrollar una prueba de concepto restringiendo el alcance a exploración en interiores definida como un Módulo de Exploración Autónoma en Interiores en el que se dispone de Spot Explorer para generar una primera iteración y evaluación de sus capacidades, tomando en cuenta los siguientes criterios de aceptación:

- Utilizar las 5 cámaras de una banda que posee Spot a bordo.
- No se puede utilizar un LIDAR.
- El sistema debe ser capaz de ejecutarse en el computador a bordo o en uno externo a Spot.
- Spot debe ser capaz de decidir de manera autónoma la ruta de exploración.
- El punto de inicio en el espacio interior a explorar es aleatorio.

1.2 Objetivos

Es así como se define los siguientes objetivos:

- Generar una clase de alto nivel con Python que permita acceder a las funcionalidades del SDK de Spot.
- Definir la arquitectura del módulo de exploración autónoma.
- Generar un modelo de inteligencia artificial capaz de identificar zonas de libre tránsito para la exploración autónoma.
- Determinar el criterio de generación de una base de datos a utilizar para la construcción del modelo.

1.3 Estructura del documento

El documento parte describiendo los robots cuadrúpedos y aplicaciones de estos en entornos reales seguido de un breve resumen sobre modelos para la detección de objetos ya que, el bloque de detección para el sistema es un detector de esquinas a nivel del suelo. En el capítulo 3 se detalla las herramientas utilizadas como el robot y el hardware disponible para el entrenamiento. En el capítulo 4 se detalla la propuesta de arquitectura empleada para solventar el problema con cada uno de sus bloques funcionales. En el capítulo 5 se desglosa el desarrollo de los elementos que conforman el sistema. Posteriormente en el capítulo 6 se muestra los resultados al experimento realizado. En el capítulo 7 se presentan las conclusiones del trabajo seguido del capítulo 8 con los trabajos futuros relacionados al trabajo presentado. Las referencias utilizadas en el trabajo se encuentran en el capítulo 9. Finalmente, en el capítulo 10 se describe la documentación de la clase en python implementada para controlar Spot y un diagrama de la red implementada.

2 Estado del arte

2.1 Robots cuadrúpedos

En años pasados se han creado robots con distintas capacidades o mecanismos que les permiten desplazarse por entornos definidos. Generalmente la opción mas inmediata es un robot con ruedas u orugas que dependiendo de las dimensiones de los obstáculos, son medianamente eficientes. Sin embargo, desde que Raibert en 1986 [1] propuso un robot con una pata bajo un sistema de péndulo invertido y accionamiento por pistón, han surgido varias propuestas de este tipo de sistemas dinámicos cuya principal característica es desplazarse por terrenos accidentados.

En 1996 se presenta TITAN-VIII [2], un robot cuadrúpedo con motores de corriente continua y de bajo costo que es capaz de desplazarse a 0.9 m/s. Posteriormente en 2002 H. Kimura [3] presenta un robot más complejo con dos grados de libertar por pata y un sistema de retroalimentación para tomar en cuenta los cambios a nivel del suelo y adatar el movimiento de desplazamiento. De esta forma se da un primer acercamiento a estos sistemas cuadrúpedos. En 2008 la empresa Boston Dynamics [4] da a conocer su robot cuadrúpedo BigDog-V2 que es capaz de desplazarse por entornos accidentados con capacidad de carga de 154 kg y una velocidad de trote de 1.6 m/s.

Claramente un robot cuadrúpedo inspirado en la naturaleza está mejor adaptado para solventar obstáculos o pendientes. En [5] diferencian cinco características claves que permiten evaluarlos con respecto a plataformas con ruedas. Primero, son capaces de liberar obstáculos y agujeros. Segundo, tienen una configuración estable sobre sus cuatro patas. Tercero, se pueden desplazar en todas las direcciones. Cuarto, son estables en terrenos accidentados o dinámicos. Quinto, pueden llevar cargas útiles sobre el lomo.

Actualmente, existen en el mercado muchas opciones de robots cuadrúpedos con distintas capacidades de carga y funcionalidades. Esta el V60 de Ghost Robotics [6] cuya plataforma está orientada para el sector de la defensa ya que puede ser equipado con armamento letal y no letal. Otro modelo es el ANYmal C inspirado en su hermano ANYmal B [7] que posee un lidar, cámara de exploración incorporada y un sistema de carga autónoma. El robot cuadrúpedo más conocido es el Spot de Boston Dynamics [8] cuya tecnología de percepción y reacción al entorno es por el momento la más avanzada disponible a nivel comercial.

2.1.1 Casos de uso

Estos robots mencionados a nivel de aplicación no son más que mulas de carga capaces de desplazarse por entornos dinámicos. Actualmente, sobre estos robots se busca montar sistemas mas dedicados para cada caso de uso que se requiera construir. Por ejemplo, [9] usa a Spot para el monitoreo de signos vitales mediante visión artificial, mediante una cámara térmica analiza la temperatura del paciente y con una tableta a bordo lo comunica con el médico a distancia. En [10] usan a BigDog para seguimiento de un objetivo como un soldado mediante un lidar y visión artificial, el robot es capaz de desplazarse por el entorno de manera autónoma al sistema de seguimiento. En [11] utilizan un robot cuadrúpedo denominado SCalf-III y una cámara estéreo para el seguimiento de personas entrenada con una red Yolo-V3.

En todos estos casos el robot se comporta como una plataforma móvil que recibe ordenes de velocidad y dirección a la cual tiene que dirigirse, por lo que los sistemas desplegados son independientes al procesamiento del movimiento.

Adicionalmente, cabe señalar que para un caso de uso real a nivel comercial o industrial se busca que estos robots colaboren con sistemas mas complejo u otro tipo de plataformas como drones o AGV. En [12] proponen un sistema multi-robot autónomo en el que el robot Spot colabora con un AGV y un drone en el cual, el sistema posee una plataforma que administra las misiones a generar por los equipos y un panel de monitoreo para un operario humano. Es decir, que el sistema accede a las funcionalidades de cada robot mediante su SDK para de esta forma controlar a muy alto nivel las acciones. De manera similar en [13] utilizan el V60 para la exploración múltiple de túneles de minas con un sistema de comunicación XBee, lidar y cámaras de profundidad para detectar zonas de riego o identificar por visión artificial objetos en específico o personas que requieren soporte. Otro tipo de aplicación colaborativa con robots la realiza en [14] que utilizan robots Unitree A1 con una cámara térmica y una red Yolo para la identificación de víctimas en búsqueda y rescate.

Las aplicaciones descritas parten de la base de uso del robot como una plataforma móvil en entornos accidentados sobre la cual se construye el hardware y software necesario para ejecutar accionas más completas como lo es la detección de objetos de interés.

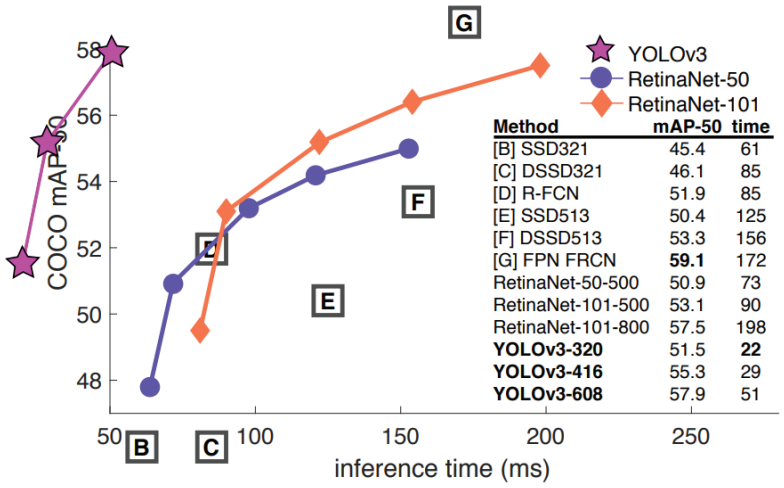
2.2 Detección de objetos

La detección de objetos es una extensión a la clasificación de imágenes ya que, en que en esta no solo interesa conocer la clase a la que pertenece un objeto identificado sino también la ubicación de este en la imagen observada. [15]

Actualmente gran parte de los algoritmos para la detección de objetos utilizan las redes neuronales convolucionales o CNN las cuales funcionan como un extractor de características muy eficiente. Basan su funcionamiento en el proceso matemático de convolución, que consiste en realizar productos punto entre los frames y distintos filtros o kernels. Estos de manera unitaria son capaces de identificar ciertas características. Con el proceso de entrenamiento y profundidad de la red, pueden identificar patrones o características más complejas en la imagen. [16]. En el 2012 A. Krizhevsky, I. Sutskever, and G. E. Hinton propusieron AlexNet [16] en la competencia de ImageNet que utiliza como eje troncal este tipo de redes convolucionales.

Desde entonces han surgido nuevas arquitecturas como R-CNN que parte de primero detectar regiones de interés previa a la detección [17]. En base a a esta nueva arquitectura surgió Faster R-CNN que mejora los tiempos de detección y obtiene un mayor porcentaje en el promedio de las predicciones para la base de datos de COCO. [18]

Otra red muy conocida es Yolo que tiene tiempos de detección bajos por lo que es ideal para realizar procesamiento en tiempo real. La característica que le permite ejecutar tiempos muy bajos es que evalúa toda la imagen mediante celdas en las que analiza si existe alguna clase de interés. Este proceso genera varias detecciónes por objeto detectado, por lo que, es necesario realizar un proceso denominado *non_max_suppresion* que consiste en tomar la detección con mayor confiabilidad. [19]



Method	mAP-50	time
[B] SSD321	45.4	61
[C] DSSD321	46.1	85
[D] R-FCN	51.9	85
[E] SSD513	50.4	125
[F] DSSD513	53.3	156
[G] FPN FRCN	**59.1**	172
RetinaNet-50-500	50.9	73
RetinaNet-101-500	53.1	90
RetinaNet-101-800	57.5	198
YOLOv3-320	51.5	**22**
YOLOv3-416	55.3	29
YOLOv3-608	57.9	51

Figura 1. Comparativa de tiempos de detección con la base de datos COCO de YoloV3 contra otras arquitecturas. [20]

La versión de yolov3 que contiene 53 capas de convolución partiendo desde 32 filtros hasta 1024; lo que le proporciona la capacidad de identificar patrones complejos en su última capa de convolución. Seguida a esta capa se realiza la operación de Average Pooling o Agrupación Promedio con el objetivo de identificar características nítidas. Al final agrega una capa full_conected seguida de una capa softmax que retorna un vector de salida con las confiabilidad, ubicación y clase de las predicciones.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figura 2. Arquitectura de la red convolucional Darknet-53. [20]

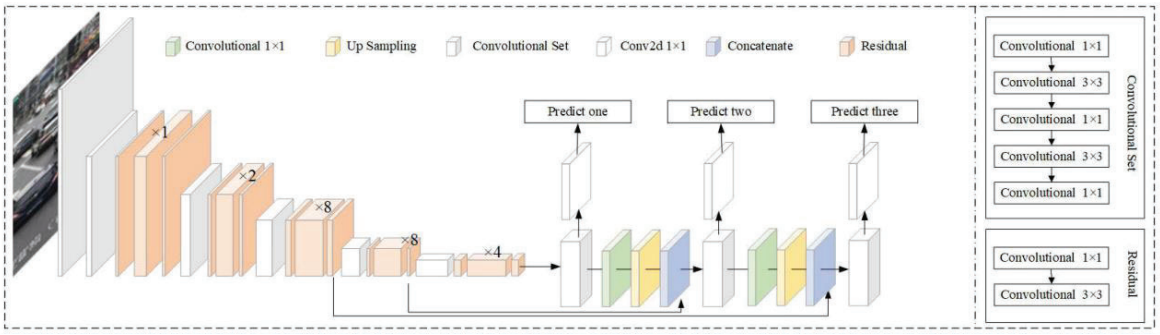


Figura 3. Arquitectura YoloV3 con Darknet-53 como eje central. [21]

La versión más actual de la red Yolo es la V5 que mejora significativamente los tiempos de detección en comparación a sus predecesores.

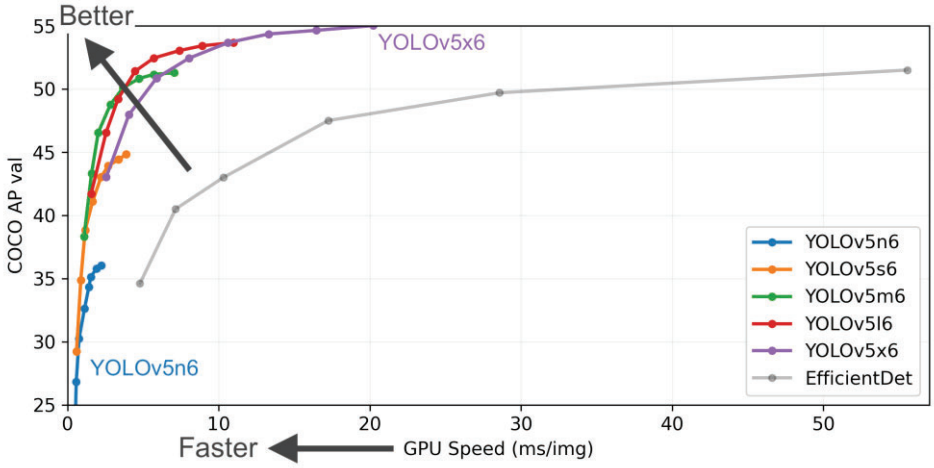


Figura 4. Comparativa de modelos disponibles de YoloV5. [22]

3 Herramientas utilizadas

3.1 Robot Spot

Spot es un robot cuadrúpedo capaz de desplazarse por entornos accidentados y reaccionar de forma dinámica a los cambios. Posee cinco pares de cámaras estéreo que proporcionan imágenes en una banda, blanco y negro, que le sirven para percibir su entorno. Sus características más importantes se describen en la Tabla 1.

Tabla 1. Principales características de robot Spot Boston Dynamics. [8]

Autonomía	90 min
Capacidad de batería	564 Wh
Protección	IP54
Velocidad máxima	1.6 m/s
Inclinación máxima	+-30º
Máxima altura de obstáculo	300 mm
Peso	32.7 Kg
Carga máxima	14 Kg
Conectividad	Wifi 2.4Ghz 802.11ac 1 puerto Gigabit Ethernet 2 puertos GXP

Spot procesa información en tiempo real sobre sí mismo, los objetos que le rodea y la percepción del entorno; por lo que es capaz de calcular su movimiento y trayectoria de sus articulaciones en caso de algún obstáculo o si se le aplica una fuerza externa.

Es importante resaltar que toda la computación de sus sistemas la realiza internamente en su sistema interior, al cual se puede acceder únicamente mediante el SDK y no es posible cargar ni manipular el software que lleva.

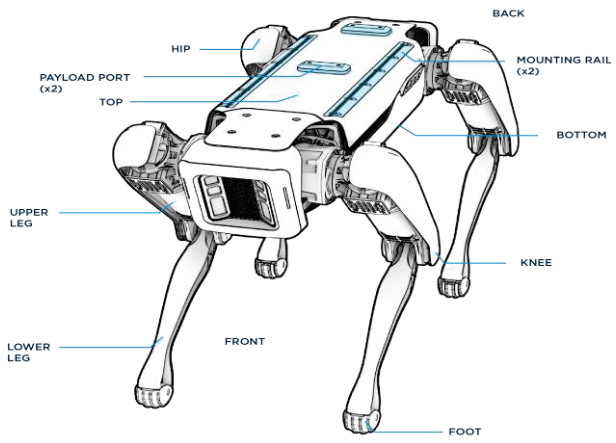


Figura 5. Anatomía de robot Spot Boston Dynamics. [8]

Actualmente se fabrica únicamente el modelo Enterprise que tiene capacidad de carga autónoma a diferencia del modelo Explorer. Sin embargo, las características son similares salvo actualizaciones en componentes y capacidades de integración futuras. Cabe recalcar que Spot viene de fábrica con un mando como se observa en la Figura 6. Con el cual se pueden realizar todas las funcionalidades que trae el SDK.

Figura 6. Controladora de serie para Spot. [8]

3.1.1 SDK

Boston Dynamics proporciona un SDK con el cual es posible obtener toda la información que la cataloga en marcos descritos en la Tabla 2 y servicios descritos en la Tabla 3. Actualmente el SDK se encuentra en la versión 3.1.0.

Tabla 2. Descripción de los marcos disponibles en el SDK de Spot. [8]

Marcos	Descripción
odom	Recolecta la información inercial que estima la ubicación del robot en el mundo.
vision	Recolecta el mismo tipo de información que odom pero mediante un análisis visual.
body	Describe la orientación y posición del cuerpo del robot.
flat_body	Describe la información de body pero en relación a la gravedad.
gpe	Estima el plano del suelo sobre la que el robot se encuentra
fiducial	Detecta un fiducial en el entorno.
filtered_fiducial	Agrega una capa de filtrado a fiducial para una detección más estable.
camera	En este marco permite acceder a la información de los 5 pares de cámaras que posee el robot.

Tabla 3. Descripción de los servicios disponibles en el SDK de Spot. [8]

Servicio	Descripción
Robot State Service	Este servicio permite concentrar el estado del robot, así como su odometría, información de batería, información de red, cinemática, información de suelo y errores.
Image Service	Permite acceder ágilmente a la información de las distintas cámaras y a una estimación de profundidad de estas.
Local Grid Service	Genera una malla con la información que recoge del entorno para estimar su posición en el mundo y obstáculos alrededor. Considera cualquier objeto, pared u otro, como obstáculo si es mayor a 40cm de altura.
World Object Service	Con este servicio es capaz de almacenar la información de la ubicación de los objetos a su alrededor, es decir, rastrea la ubicación.
E-Stop Service	Es el paro de emergencia por software del robot, por lo que es indispensable que siempre este activado para su funcionamiento.
Power Service	Permite encender y apagar los motores del robot.
Robot Comand Service	Permite acceder a las funciones de movimiento del robot.

Adicionalmente, Spot posee un paro de emergencia físico en la parte posterior, por lo que para cualquier uso de los servicios descritos en la Tabla 3 es indispensable que se encuentre activado.

Esta descripción descrita es la básica tanto para el modelo Explorer y Enterprise. Adicionalmente, se dispone de funciones específicas para el control del brazo.

Además, posee funciones de serie las cuales se describen en la Tabla 4.

Tabla 4. Funcionalidades de serie de Spot. [8]

Funcionalidades	Descripción
Choreography	Permite generar coreografías de movimiento. Requiere de una licencia adicional para acceder a esta funcionalidad.
Autowalk	Permite grabar misiones manualmente sobre entornos y luego Spot es capaz de reproducirlo. El proceso de grabación se realiza con el mando de fábrica o con la Plataforma Robótica de Alisys.

3.2 Hardware disponible

Para el desarrollo del sistema y entrenamiento del modelo se dispuso de una torre proporcionada por Alisys con características de alto rendimiento como se describe en la Tabla 5.

Tabla 5. Características torre de entrenamiento.

Característica	Descripción
Procesador	Intel(R) Core(TM) i7-9700K CPU @ 3.60GHz
Tarjeta Gráfica	NVIDIA GeForce RTX 2080 SUPER
Sistema operativo	Ubuntu 20.04
RAM	32 GiB

Adicionalmente, se dispuso de una tarjeta de desarrollo, descrita en la Tabla 6, para Edge Computing con la finalidad de validar el sistema en este tipo de dispositivos y determinar las configuraciones adecuadas para el correcto funcionamiento.

Tabla 6. Características de la placa de desarrollo JetsonNano. [23]

Característica	Descripción
Procesador	CPU Quad-core ARM A57 @ 1.43 GHz
GPU	128-core Maxwell
Sistema operativo	Jetpack SDK 5.0 NVIDIA - Ubuntu
RAM	4 GiB

4 Arquitectura propuesta

Como describe el SDK de Spot, es posible acceder a las funcionalidades de movimiento básico y cámaras a bordo externamente mediante conexión wifi o vía ethernet en sus conectores abordo. Por lo que se plantea inicialmente usar conectividad inalámbrica con la torre proporcionada en la que se ejecuta el sistema por completo. La arquitectura planteada para el Módulo de Exploración Autónoma en Interiores se compone de tres bloques funcionales como se muestra en la Figura 7.

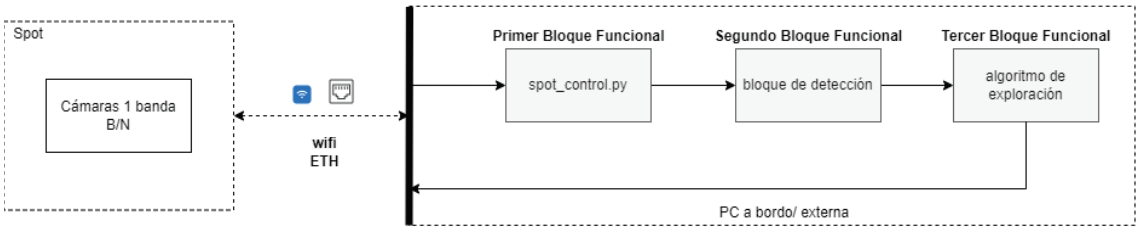


Figura 7. Arquitectura simplificada del Módulo de Exploración Autónoma en Interiores.

Cada uno de estos bloques funcionales comparten información y se ejecutan de manera simultánea.

4.1 Primer bloque funcional: spot_control

El primer bloque funcional corresponde a una clase en python que contiene las configuraciones básicas para el funcionamiento y conectividad con Spot, así como todas las funciones para el movimiento y adquisición de imágenes de sus 5 módulos de cámaras.

Esta clase denominada *spot_control.py* fue desarrollada tomando en consideración los criterios de escalabilidad de codificación para futuros despliegues en el Departamento de Robótica de Alisys. Además, se implementaron funcionalidades que son compatibles con el brazo robótico que posee el modelo Enterprise, relacionadas al movimiento y misiones autónomas.

El primer requisito implementado en esta clase es la conectividad con Spot detallado en el SDK, que además se usuario y contraseña proporcionado por el fabricante, requiere que se levante el servicio de paro de emergencia E-Stop en segundo plano a la ejecución del script troncal. Esta implementación está realizado con hilos de ejecución en python y sintetizados en una sola función de la clase denominada *connect( ).*

Con la conectividad asegurada al robot, se dispone de otra función que permite iniciar los motores de Spot accediendo al servicio de Power Servirce del SDK. En la clase esto se encapsula en la función *power_on( ).*

La funcionalidades descritas anteriormente aseguran la conectividad y habilitan para comandar el robot por lo que la función desarrollada para el movimiento es denominada *move( ).* Esta función recibe parámetros de velocidad para trasladar al robot sobre su plano XY y velocidad de rotación sobre su eje Z descrito en la Figura 8.

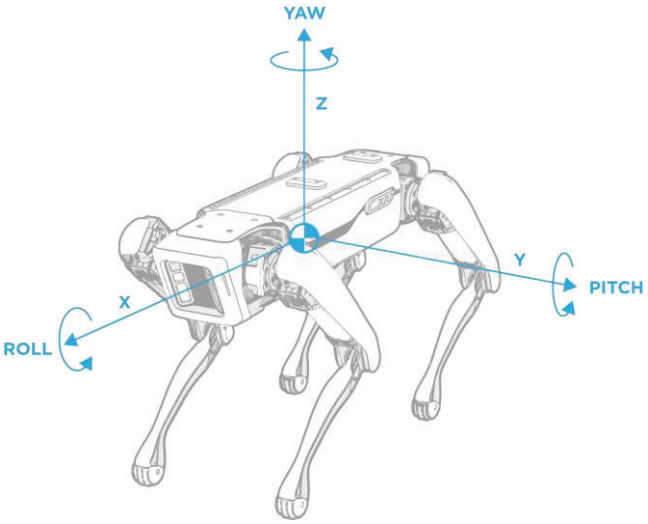


Figura 8. Ejes troncales del robot Spot. [8]

Finalmente, es indispensable obtener la información de las 5 cámaras ya que estas se usarán para la detección de libres zonas de tránsito. El servicio de Image Service del SDK proporciona fotogramas sin codificar por lo que la función *get_video( )* se encarga de codificar en un arreglo de pixeles para su posterior procesamiento.

En el Anexo 1 se detalla cada una de las funcionalidades desarrolladas y sus respectivas llamadas.

4.2 Segundo bloque funcional: bloque de detección

El segundo bloque funcional contiene el modelo entrenado para la detección de esquinas a nivel del suelo. Este proporciona la ubicación de las esquinas en cada una de las 5 cámaras de Spot.

La hipótesis planteada es generar un polígono a nivel del suelo que indique si es una zona de libre tránsito o no. Inicialmente se planteó identificar objetos como puertas, paredes, sillas, mesas u otro objeto común que se pueda encontrar en interiores, pero una característica que tienen en común estos objetos es que están a nivel del suelo por lo que generan esquinas claramente diferenciables.

De esta forma se construyó un modelo que sea capaz de identificar dichas esquinas de objetos cercanos y lejanos al observador que formen una zona de libre tránsito. Como se observa en la Figura 9 las esquinas a detectar son únicamente las más próximas a un espacio de libre tránsito independientemente de la naturaleza del objeto que forme dicha esquina.

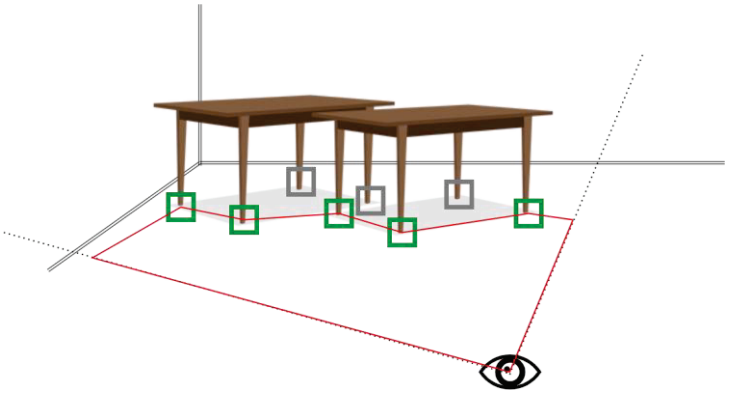


Figura 9. Ejemplo de observador identificando esquinas de interés en color verde y gris esquinas de no interés. Zona de libre tránsito en rojo.

Es importante recalcar que el proceso de detección se realiza sobre los 4 lados del robot para obtener una concepción global de zonas de libre tránsito en cada instante.

En consecuencia, este bloque funcional contiene el modelo entrenado que retorna las coordenadas de las esquinas de interés detectadas en cada una de las 5 cámaras de spot en forma de lista para su posterior procesamiento en el tercer bloque funcional.

4.3 Tercer bloque funcional: algoritmo de exploración

Por último, está el tercer bloque funcional que contiene el algoritmo de exploración que proporciona la lógica necesaria para que el robot sea capaz de decidir la dirección a tomar en base al análisis global de las zonas de libre tránsito.

Una vez detectado las esquinas con el modelo entrenado, se procede a identificar la zona de libre tránsito. Esta zona está definida por el polígono que forma cada una de las *corner_floor* identificadas y a su vez forman una superficie virtual que se asemeja al suelo como se observa en la Figura 9 de ejemplo.

Por si solas las esquinas dan únicamente información sobre la existencia y ubicación de objetos cercanos, sin embargo, ya que no se calcula una distancia como tal hacia el objeto, es necesario segmentar cada detección y determinar si se encuentra en una zona en la que se defina como transitable o no transitable.

Por lo tanto, cada identificación la podemos situar sobre un plano 2D de alto y ancho igual a la resolución de la cámara usada en cada lado del robot. Sobre este plano se definieron tres zonas para catalogar si las esquinas se sitúan en una zona transitable, de exploración o no transitable. Estas zonas están definidas por dos umbrales, uno en X o ancho y otro en Y o alto. Cada umbral depende de la posición de las cámaras y su rango de visión.

De esta forma los umbrales permitirán discriminar si existe algún obstáculo que posteriormente será usada por el algoritmo de exploración para decidir la acción a tomar por el robot.

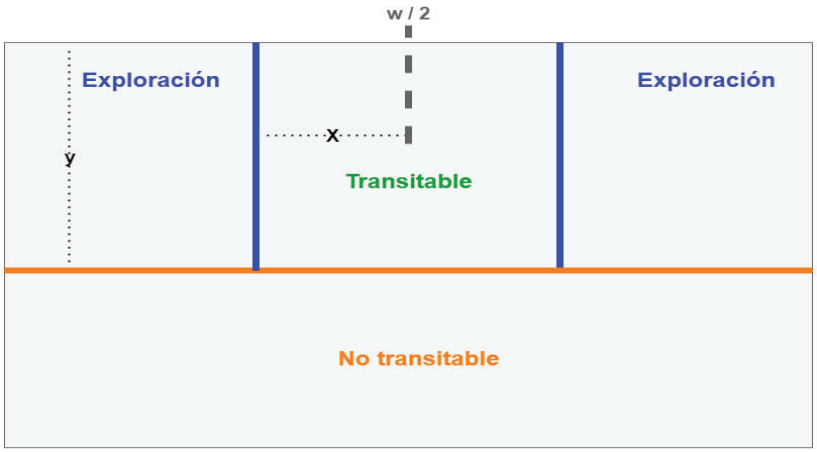


Figura 10. Definición de zonas transitables en 2D.

Para obtener un único sensor que permita determinar si el entorno se puede transitar o no, se calculó la media de las esquinas detectadas obteniendo un único punto o vector. En consecuencia, esta categorización permite definir con un valor de 1 al sensor resultante cuando se ubica sobre la zona transitable o de exploración y en 0 si se sitúa sobre la zona no transitable.

De esta forma Spot es capaz de explorar visualmente el entorno identificando zonas transitables, en la Figura 11 se observa una representación de tránsito en un tiempo T1 a Tn con objetos a su alrededor.

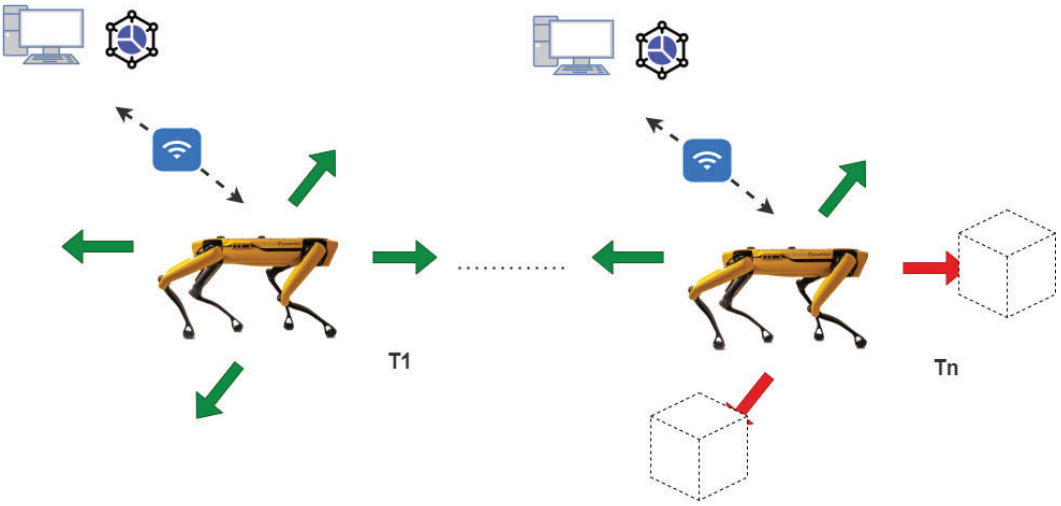


Figura 11. Representación de detección de zonas transitables de Spot.

5 Desarrollo

5.1 Generación del dataset

El modelo debe ser capaz de identificar los puntos de interés a nivel del suelo, por lo que, es necesario construir una base de datos propia con las imágenes de cada una de las cinco cámaras de Spot en la que se realizará un proceso de etiquetado sobre las mismas.

Como primer paso se realizó la captura de las imágenes mediante la clase *spot_control.py* para guardarlas con un identificador único en el almacenamiento local de una computadora externa. Las imágenes fueron capturadas tanto con Spot en movimiento como estático, en la Figura 12 se puede observar una composición de las cinco cámaras de Spot en un instante de tiempo.

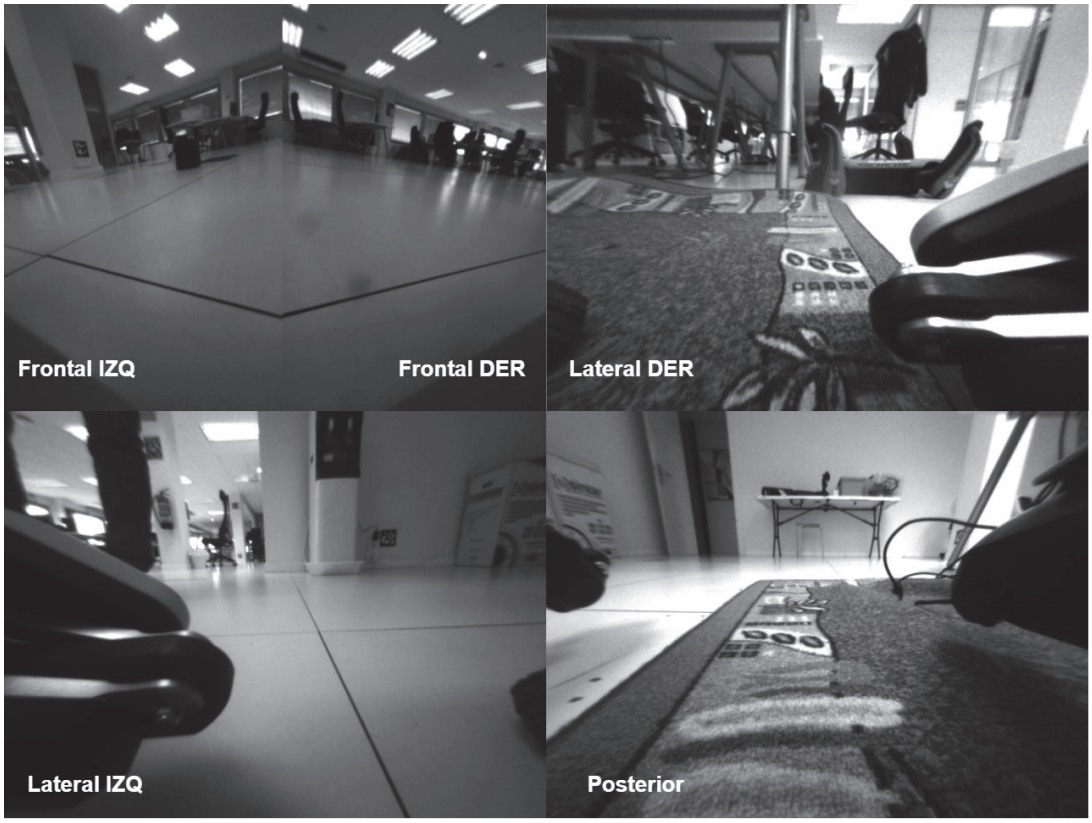


Figura 12. Ejemplo de imágenes de las cámaras abordo de Spot.

Para el proceso de etiquetado se utilizó la herramienta labelImg [24] que permite generar bases de datos sobre imágenes de manera sencilla y ágil. En la Figura 13 se puede observar un ejemplo de etiquetado con esta herramienta de los puntos de interés definidos como esquinas a nivel del suelo.

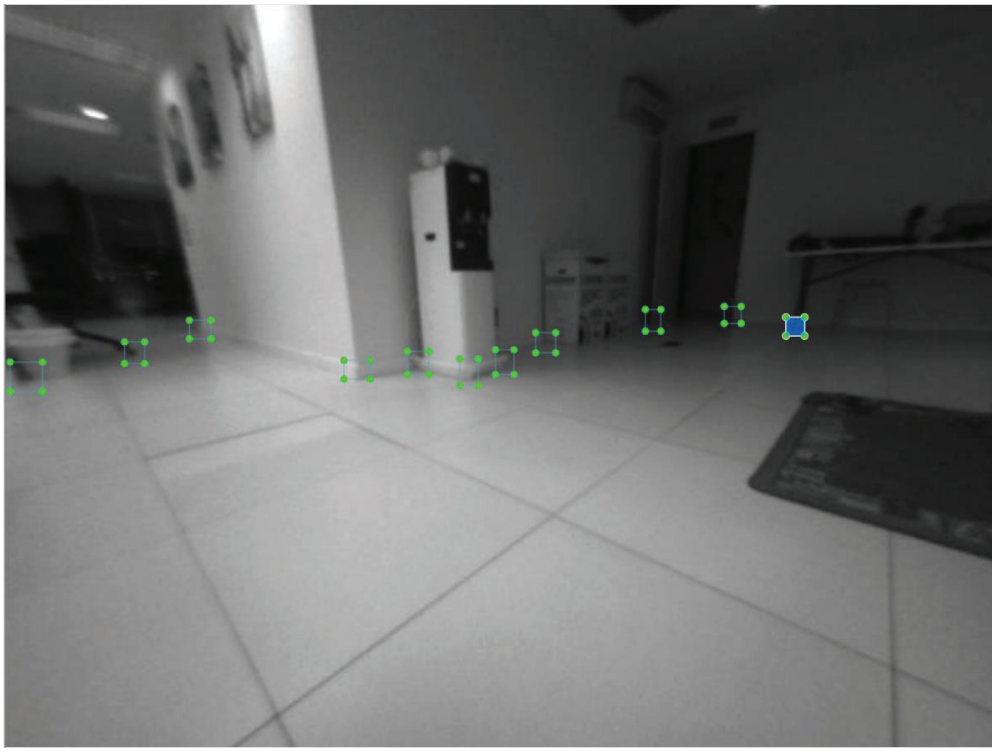


Figura 13. Ejemplo de etiquetado con labelImg de los puntos de interés definidas como esquinas de suelo.

En el etiquetado se marcan las esquinas a nivel del suelo con cuadrados señalando dos puntos sobre la zona de interés, superior izquierdo denominado A y otro inferior derecho denominado B; posteriormente se indica a la clase que pertenece, definida como *corner_floor* indexada en cero. En la Tabla 7 se pueden observar las etiquetas generadas de la Figura 13. Los datos de cada imagen se guardaron en un archivo *.txt* con el mismo nombre de la imagen.

Tabla 7. Ejemplo de formato de etiquetado.

Clase	**cood. X punto A**	**cood. Y punto A**	**cood. X punto B**	**cood. Y punto B**
0	0.021094	0.501042	0.032813	0.039583
0	0.130469	0.468750	0.020313	0.029167
0	0.196875	0.437500	0.021875	0.025000
0	0.355469	0.491667	0.026562	0.025000
0	0.417187	0.482292	0.021875	0.031250
0	0.468750	0.494792	0.018750	0.035417
0	0.504687	0.481250	0.018750	0.033333
0	0.546094	0.455208	0.020313	0.027083
0	0.654687	0.425000	0.015625	0.029167
0	0.735156	0.417708	0.017188	0.022917
0	0.798438	0.433333	0.018750	0.025000

Se generaron 500 imágenes para evaluar en el entrenamiento. En adición, se dividió cada base de datos en 90% para entrenamiento y 10% para validación en el proceso de entrenamiento. Las imágenes de prueba se realizaron directamente en el funcionamiento del sistema en entornos fisicos que no hayan formado parte de la captura inicial de la base de datos.

Posteriormente, cada imagen y etiqueta se convirtieron en tensores para ser utilizados en el entrenamiento. Se utilizó la herramienta de torchvision, *transforms.ToTensor()* [25] la cual permite convertir la información de la imagen en el rango [0,255] a tensores en el rango [0.0, 0.1]. De igual manera se realiza para las etiquetas mediante *torch.from_numpy*. [25] Por lo que se tiene finalmente una base de datos iterable con la ruta de cada imagen, el tensor de la imagen y el tensor de sus etiquetas.

5.2 Modelo

Se plantearon dos implementaciones con pytorch de Yolo con la V3 y V5 para evaluar los tiempos de detección y confiabilidad en tiempo real.

Para la codificación con YoloV3 se utilizó la configuración de YoloV3 descrita en [20], que utiliza como columna vertebral la arquitectura de la red convolucional Darknet-53.

Con el fin de acortar el proceso de entrenamiento y obtener mejores resultados se siguió la técnica de Transfer Learning, que consiste en tomar las capas ocultas de una red y transferirlo a una nueva arquitectura con el objetivo de utilizar el aprendizaje obtenido. [26] Para este PoC se tomó la red de YoloV3 entrenada con la base de datos COCO. [20]

De igual manera para la implementación de YoloV5 se desplego el modelo entrenado yolov5s con la implementación descrita en [22].

5.3 Entrenamiento

Para la implementación con YoloV3 se tomó la base de datos previamente generada se invocó con la utilidad de pytorch DataLoader [25], el cual permite cargar un conjunto de datos iterables, para posteriormente pasarlo a la red para ser entrenados por una cantidad de épocas definida. Adicionalmente, al finalizar cada época se realiza una validación del modelo con la base de datos de validación.

Tabla 8. Configuración realizada para este entrenamiento con YoloV3.

Parámetro	Valor	Descripción
dataset	Base de datos generada tipo lista	Contiene las imágenes y clases como tensores
batch_size	4	Número de imágenes por lote
shuffle	True	Reorganiza los datos en cada época
num_workers	8	Número de subprocesos en CPU para el entrenamiento
pin_memory	True	Copia los tensores al CUDA

Para finalizar la configuración del entrenamiento se utilizó el optimizador Adam [27] que permite la optimización basada en gradientes. En pythorch se tiene disponible con *torch.optim.Adam.* [25]

El entrenamiento se realizó utilizando CUDA con *torch.device("cuda").* [25]

En la Figura 10 se observa un diagrama del proceso de entrenamiento para este PoC.

Adicionalmente, se utilizó tensorboard [28] para guardar las métricas de entrenamiento que se describen en la Tabla 9.

Tabla 9. Métricas de evaluación para detección de objetos. [29] [30]

Métricas	**Descripción**
precision	Refleja la confiabilidad del modelo para realizar predicciones positivas tomando en cuenta todas las predicciones.
recall	Refleja la capacidad de realizar predicciones positivas sin tomar en cuenta las predicciones negativas clasificadas como positivas.
f1	Mide el equilibrio entre precisión y recall.
AP	Representa el promedio de todas las precisiones.
loss	Tasa de pérdida del proceso de aprendizaje.
model	Arquitectura del modelo.

Para el entrenamiento con YoloV5 se tomó la misma base de datos con las imágenes y sus las etiquetas organizadas en ficheros *.txt* y encapsularlos en el formato *.yaml* compatible para el entrenamiento del modelo. [22]

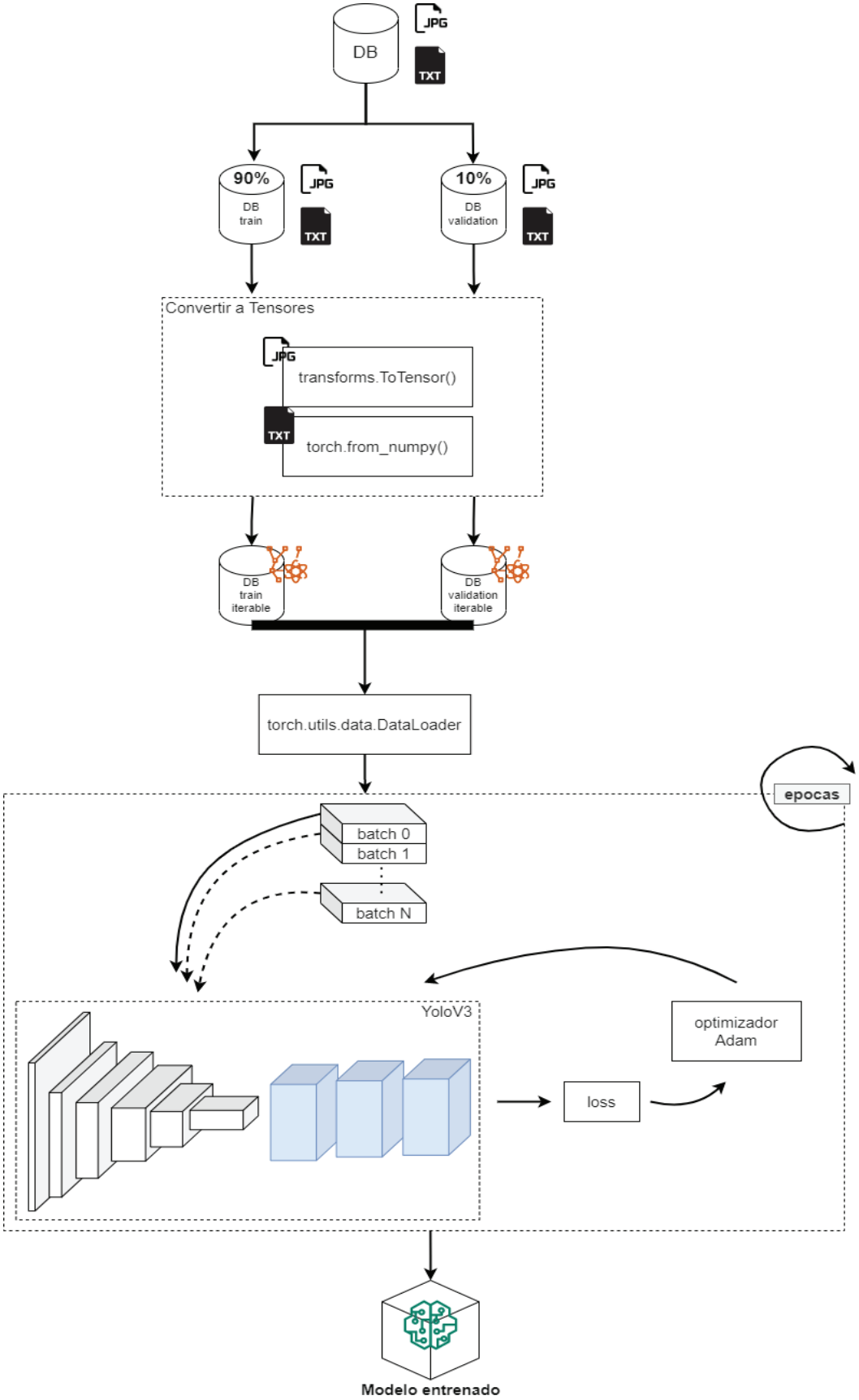


Figura 14. Proceso simplificado de entrenamiento para el MEAI con YoloV3.

5.4 Detección

La detección se realizó sobre las cinco fuentes de imágenes de Spot al mismo tiempo; debido a que se requiere conocer en cada instante de tiempo los espacios de libre tránsito, para posteriormente pasar esta información al algoritmo de exploración.

Cada imagen es convertida a tensores y luego se realiza el proceso de detección con el modelo entrenado. La red retorna todas las predicciones por objeto detectado, por lo que, se realiza el proceso de *non_max_suppresion* que consiste en seleccionar la mejor detección del conjunto generado por la red. [19] Al final se obtiene la confiabilidad de la predicción, la clase, alto, ancho y punto central de la caja que encierra a la predicción.

El proceso de detección se desplego sobre la torre de entrenamiento. Sin embargo, se realizaron pruebas adicionales con la placa de desarrollo JetsonNano agregando 4GiB de memoria *swap* y eliminando la interfaz visual para aumenta espacio en la memoria RAM.

5.5 Algoritmo de exploración

Para este PoC se implementó dos algoritmos de exploración, uno basado movimiento aleatorio y evasión de obstáculos, y otro basado en la cantidad de esquinas de exploración. Lo algoritmos se describen en Algoritmo 1 y 2.

Algoritmo 1: Exploración

```
Input: S = [s_front, s_back, s_left, s_right]
estado = explorar
if estado == explorar then
        if S > 0 then
                estado = evadir
        else
                estado = girar X grados
        end
else
        estado = girar X grados
        estado = explorar
end
```

Algoritmo 2: Cantidad de esquinas de exploración

Input: **S = [s_front, s_back, s_left, s_right, num_front, num_back, num_left, num_right]**

Input**: umbral = [rand_front, rand_back, rand_left, rand_right]**

if **S** then

    if s_front == 1 then

        estado = mover adelante

    else

        evaluar S

        estado = girar X grados hacia lado con más esquinas

    end

else

    rand umbral

    evaluar u**mbral**

    estado = girar X grados hacia lado con mayor probabilidad

end

6 Resultados

Se plantean realizar las pruebas de funcionamiento en la plata 7 de las oficinas de Alisys que se muestra en la Figura 15 en la que se define un punto de inicio A y un punto final B de exploración. Además, se divide el espacio de experimentación en 63 celdas virtuales.

Figura 15. Entorno de prueba para exploración. Oficina de Robótica de Alisys.

Como primer apartado es validar la capacidad de las 2 implementaciones generadas, con yoloV3 y yoloV5s, en detectar las esquinas a nivel del suelo cercanas al observador.

Posteriormente, registrar información relevante en relación con los tiempos de detección ya que el análisis de la libre zona de tránsito es en tiempo real y sobre las imágenes capturadas por las 5 cámaras de Spot.

Finalmente, levantar todo el sistema con los tres bloques funcionales y evaluar el comportamiento del sistema en un entorno real, distinto al utilizado para la generación de las bases de datos y ajustar cada uno de los umbrales en cada cámara para delimitar las zonas de exploración, transitable y no transitable.

Adicionalmente, se plantea realizar una prueba de implementación del sistema con las imágenes capturas de una cámara 360 a bordo del robot Spot Enterprise con la finalidad de observar la capacidad de funcionamiento.

Sobre este entorno descrito anteriormente se validaron los dos modelos generados con yoloV3 y yolov5s obteniendo resultados similares en relación con la cantidad de esquinas que pueden detectar en una imagen.

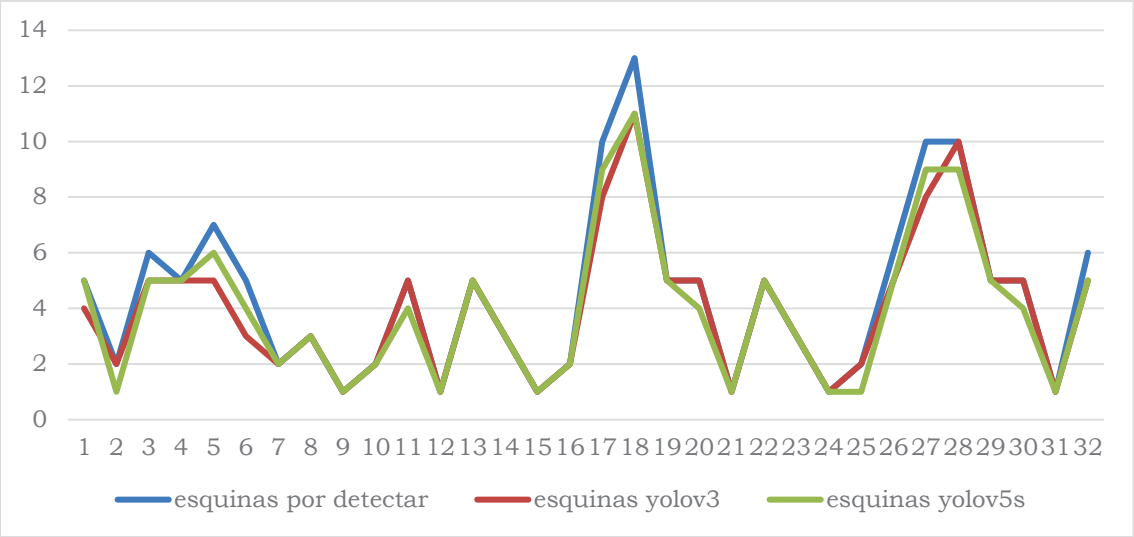


Figura 16. Comparación de los modelos sobre la cantidad de esquinas detectadas en 32 muestras de imágenes.

Por otro lado, los tiempos de detección en promedio para el modelo con arquitectura de yolov3 es de 0.06 segundos y para el modelo yolov5s el promedio es de 0.02 segundos.

El modelo yolov3 tiene una tasa de perdida en entrenamiento de 0.013 en la época 100 como se observa en la Figura 17.

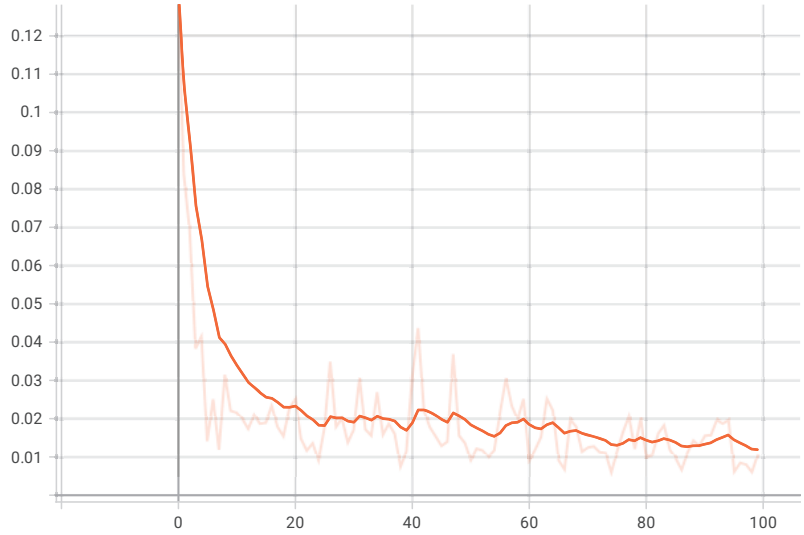


Figura 17. Tasa de pérdida del proceso de aprendizaje vs épocas.

La *precisión* y el *recall* en la época 100 es de 0.26 y 0.56 respectivamente con un f1 de 0.35 y un AP de 0.256 como se observan en las Figuras 18, 19, 20 y 21.

Esto describe que el modelo posee una gran capacidad de predecir muestras positivas, por lo que para este caso de uso en el que se tiene una sola clase es funcional para su propósito de detección de esquinas del suelo.

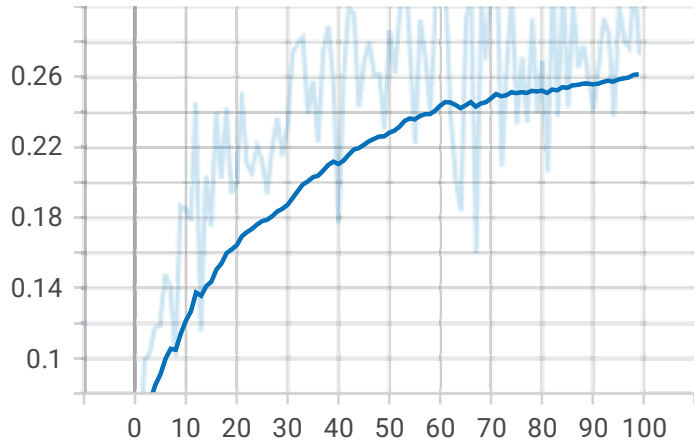


Figura 18. Métrica de precisión en entrenamiento del modelo vs épocas.

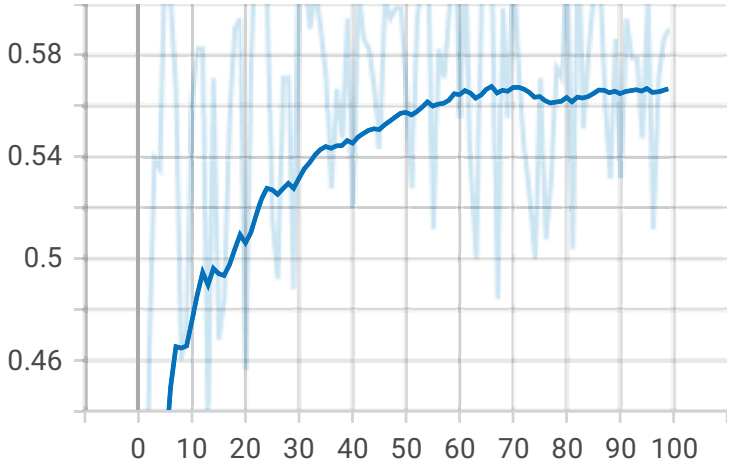


Figura 19. Métrica de recall en entrenamiento del modelo vs épocas.

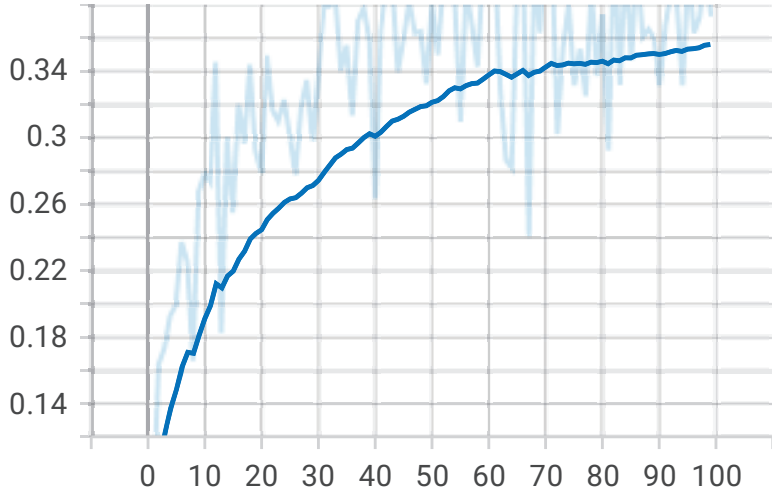


Figura 20. Métrica f1 en entrenamiento del modelo vs épocas.

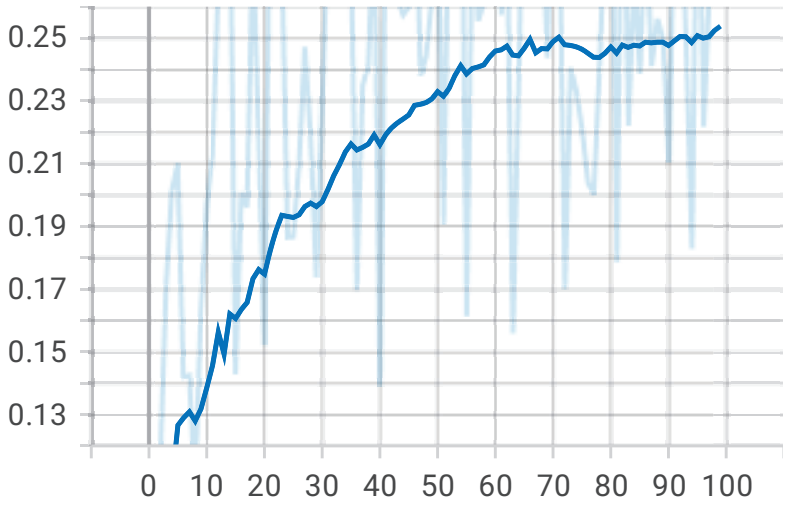


Figura 21. Métrica AP en entrenamiento del modelo vs épocas.

De igual manera con el modelo de yolov5s se obtuvo una mayor *precision* de 0.483, un *recall* de 0.275 y un AP de 0.22 en la época 100 del entrenamiento.

Ambas implementaciones sin capaces de identificar las esquinas a nivel del suelo sin embargo la implementación con yolov5s es la más rápida.

En la Figura 22 se puede observar el proceso de detección desplegado sobre las cinco cámaras de Spot al mismo tiempo.

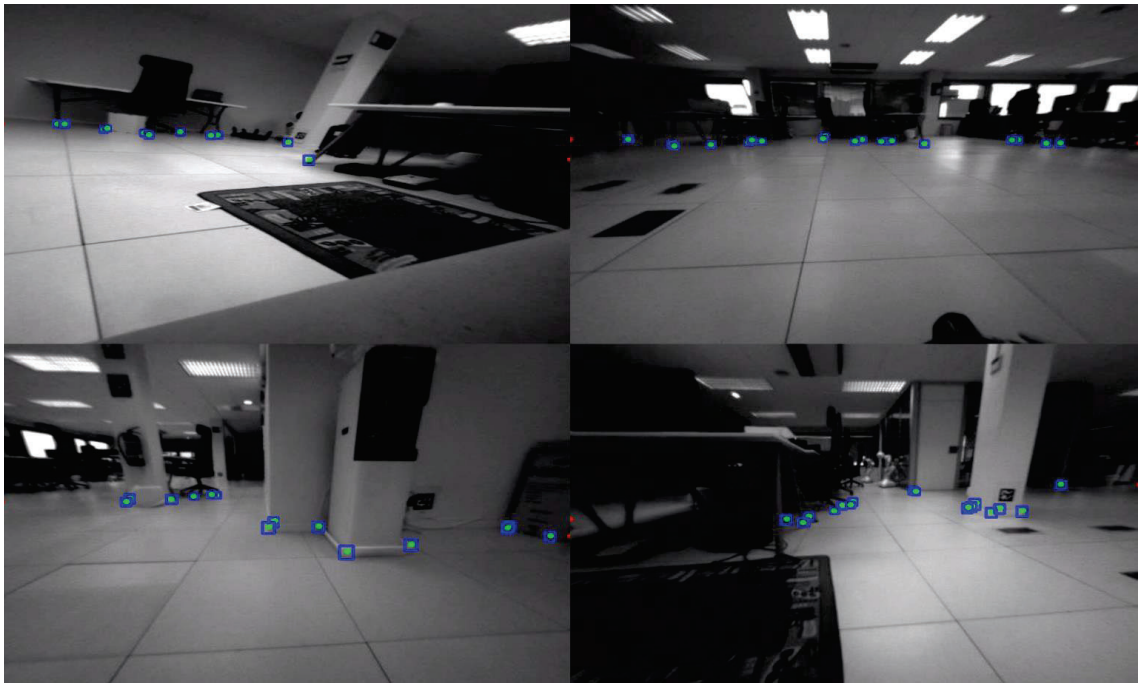


Figura 22. Detección en tiempo real sobre las 5 cámaras de Spot.

Por otro lado, con la JetsonNano el tiempo de detección con yolov3 es de 1 segundo, esto debido a las limitaciones de hardware.

Es importante recalcar que las dos implementaciones son capaces de detectar únicamente las esquinas a nivel del suelo, es decir, que otro tipo de esquinas como las formadas por mesas, esquinas superiores de pared y puertas no son detectadas, por lo que cumplen con el objetivo de marcar los puntos claves para determinar los espacios de libre tránsito. En las Figuras 23, 24 y 25 se observan más pruebas de detección.

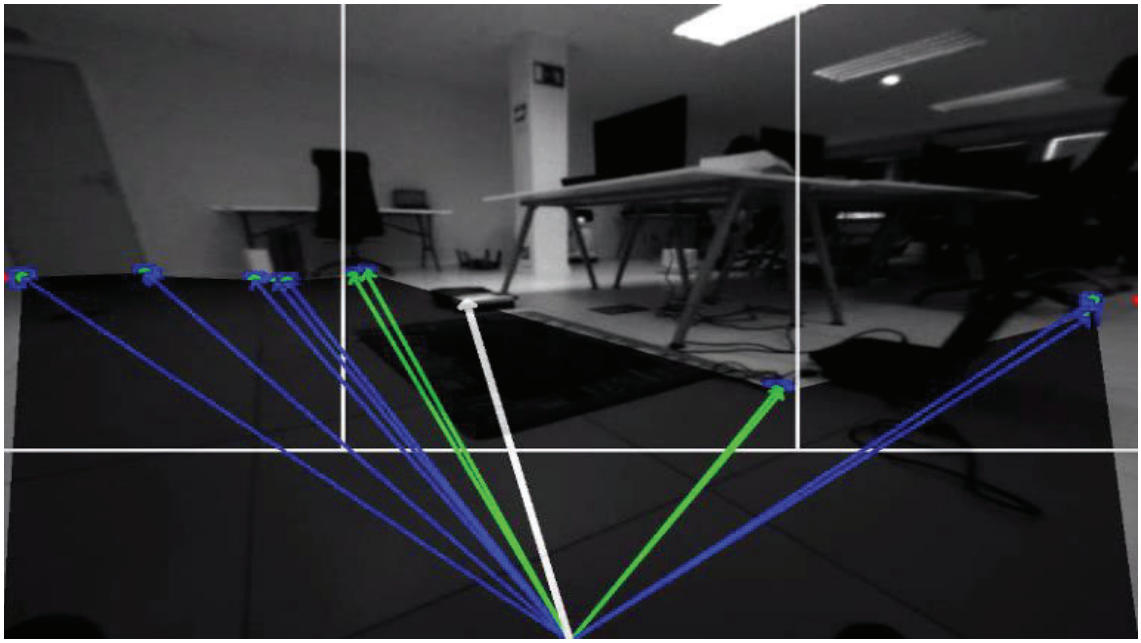


Figura 23. Prueba de detección con objetos cercanos sobre el suelo.

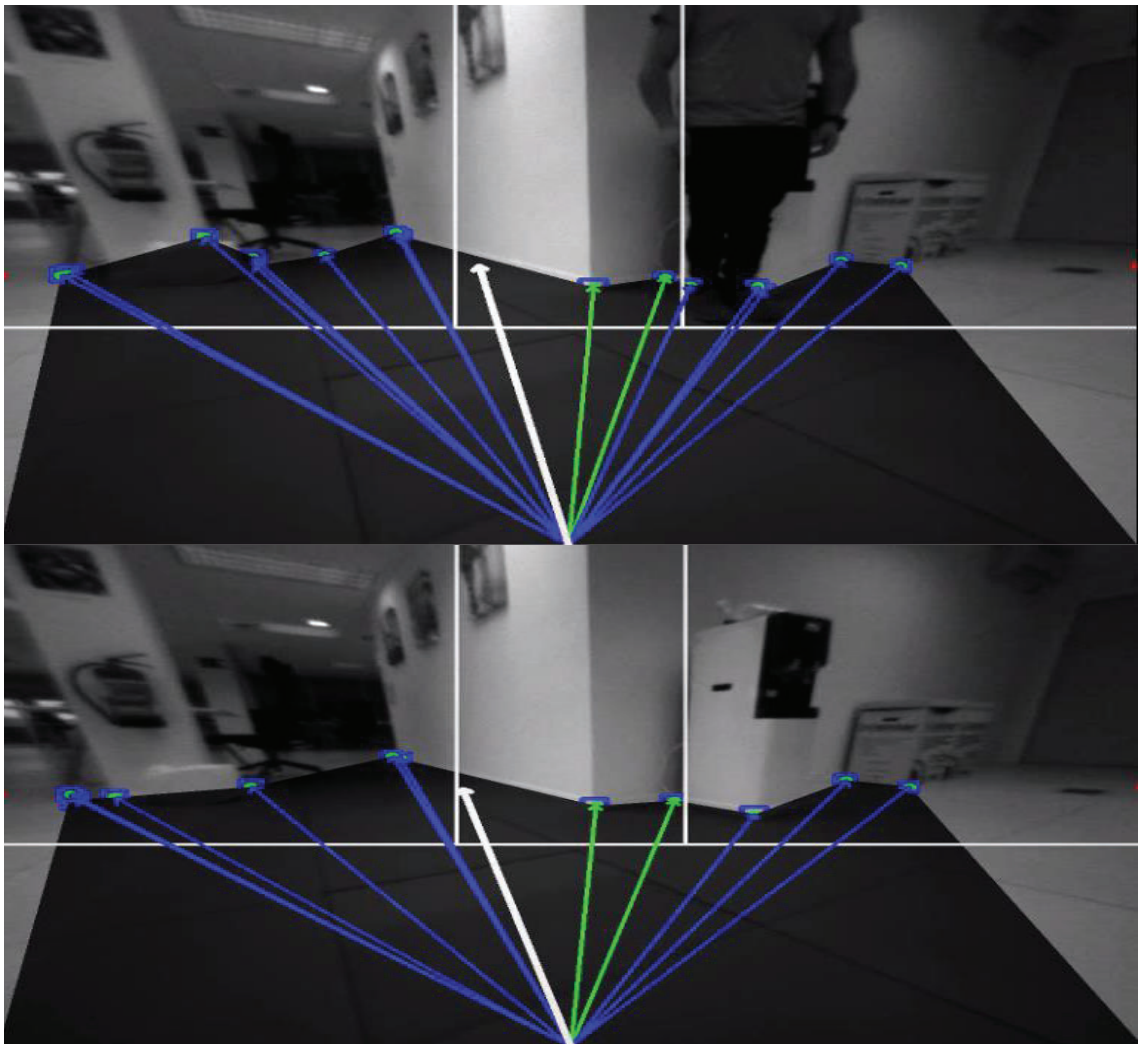


Figura 24. Prueba de detección con personas paradas frente a las esquinas detectadas.

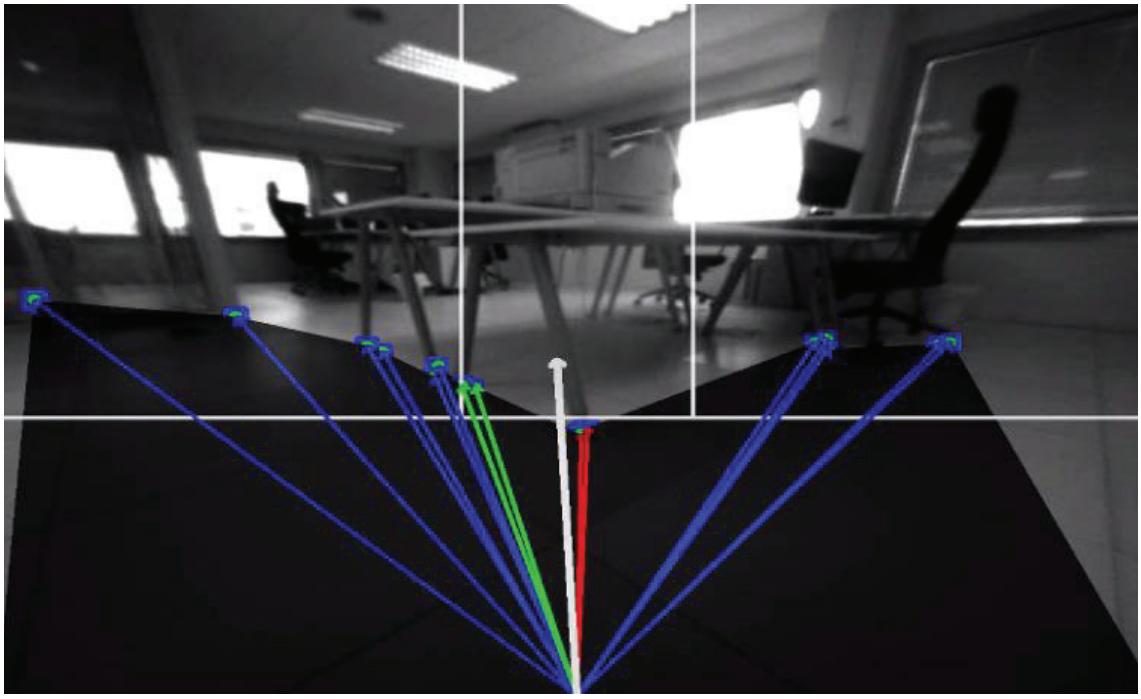


Figura 25. Prueba de detección con esquinas cercanas y lejanas.

Otra característica interesante del funcionamiento de las implementaciones es la capacidad de detectar únicamente las esquinas más próximas al robot, por lo que esquinas que se encuentras por detrás de esquinas cercanas no son detectadas. Esto asegura la identificación de las zonas de libre tránsito y en casos como en la Figura 25, solo detecta las esquinas más cercanas.

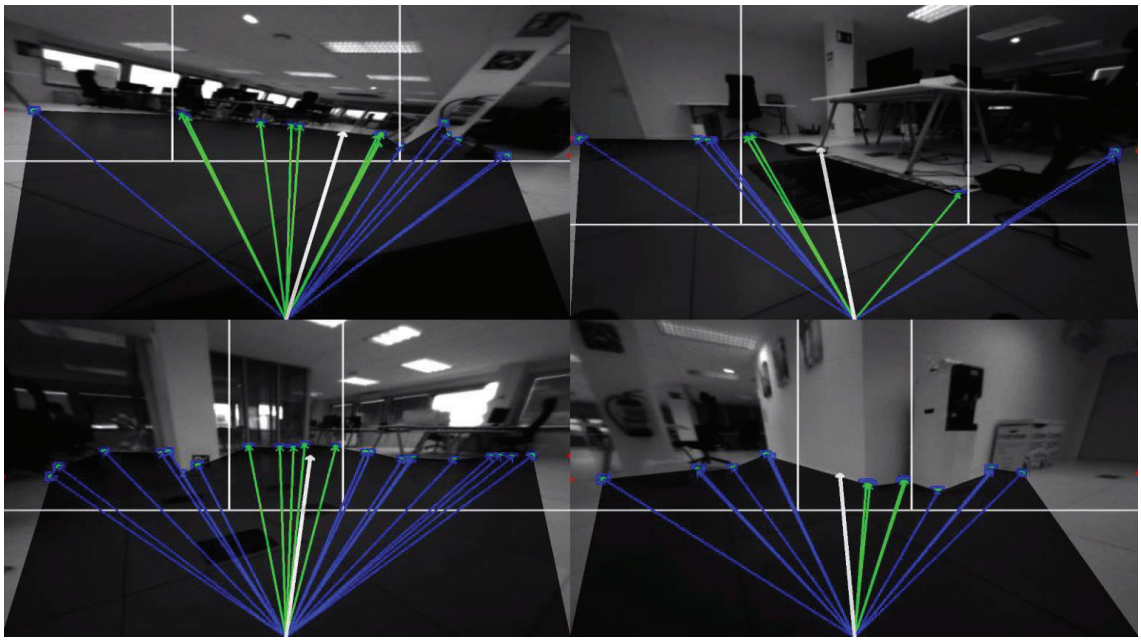


Figura 26. Vectores resultantes para la definición de zonas de libre tránsito.

Con la implementación del Algoritmo 1, Spot fue capaz de explorar 31 celdas del mapa en un tiempo de 3 minutos. Por otro lado, con la implementación del

Algoritmo 2 aumentó las celdas exploradas a 54 en un tiempo de 5 minutos. Esto debido a que la lógica del algoritmo es dar mayor peso al lado que tenga más esquinas de exploración y por lo tanto se refleja en mayor cantidad de celdas exploradas.

Tabla 10. Métricas de evaluación de los algoritmos implementados.

Algoritmo	Tiempo de exploración	Celdas exploradas
1	3 min	31
2	5 min	54

Figura 27. Exploración de A a B con la implementación del Algoritmo 1.

Figura 28. Exploración de A a B con la implementación del Algoritmo 2.

La prueba adicional sobre el video 360 fue satisfactoria ya que claramente es capaz de detectar las esquinas a nivel del suelo como se observa en la Figura 29.

Figura 29. Prueba de detección de esquinas sobre un video 360 RGB.

7 Conclusiones

La arquitectura planteada cumple con los criterios de escalabilidad, principalmente la clase desarrollada para el acceso a alto nivel del SDK de spot, denominada *spot_control*, que permite realizar despliegues en menor tiempo y proporciona una base de desarrollo fácil de mantener y actualizar por el equipo de Robótica.

En relación con la identificación de libres zonas de tránsito se solventó detectando las esquinas cercanas o puntos de interés que forman las paredes u otros objetos a nivel del suelo. Esto permite construir una zona desde la perspectiva de visión de la cámara, por lo que se puede inferir la superficie del suelo. Mediante este razonamiento fue posible generar una base de datos en la que se etiquetó manualmente los puntos de interés a ser entrenados.

El proceso de detección para las pruebas realizas se desplegaron desde un computador externo a Spot obteniendo tiempos de detección bajos sobre las dos implementaciones de yolo. Sin embargo, en las pruebas adicionales con la JetsonNano, este tiempo aumenta debido a sus limitaciones de hardware, pero de todas formas Spot es capaz de explorar.

La prueba adicional realizada con la cámara 360 permitió comprobar la capacidad del sistema de detectar esquinas con imágenes distintas a las utilizadas en el entrenamiento. Sin embargo, la lógica de exploración cambia relativamente ya que, al tener una visión global del entorno sobre una misma imagen, se debe tomar en consideración de igual manera los cuatro lados del robot para ser diferenciados.

8 Trabajos futuros

El presente desarrollo permitió evaluar la prueba de concepto de un Módulo de Exploración en Interiores accediendo a las funcionalidades básicas de Spot por lo que, para futuros desarrollos en el Departamento de Robótica se plantean integrar datos de odometría, para hacer el seguimiento de las celdas exploradas, identificación de zonas de interés, integraciones de mapeo por LIDAR, generación de misiones autónomas en base a la exploración generada, prueba de algoritmos más complejos de exploración, integraciones de hardware dedicado para comunicaciones malladas y punto a punto e integración de hardware para Edge Computing.

Adicionalmente, se encuentra en la planificación del Departamento contar con un servicio de kubernetes en el cual desplegar soluciones como la del presente trabajo que permita escalar los despliegues en robots de distintas marcas y con distintas capacidades de movilidad.

Otra línea de investigación del presente trabajo es utilizar el sistema para capturar el recorrido explorado y generar una reconstrucción volumétrica del entorno en tiempo real.

9 Referencias bibliográficas

[1] M. H. Raibert, *Legged robots that balance*. MIT press, 1986.

[2] K. Arikawa and S. Hirose, "Development of quadruped walking robot TITAN-VIII," in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems. IROS'96*, 1996, vol. 1, pp. 208–214.

[3] H. Kimura, Y. Fukuoka, Y. Hada, and K. Takase, "Three-dimensional adaptive dynamic walking of a quadruped-rolling motion feedback to cpgs controlling pitching motion," in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*, 2002, vol. 3, pp. 2228–2233.

[4] M. Raibert, K. Blankespoor, G. Nelson, and R. Playter, "Bigdog, the rough-terrain quadruped robot," *IFAC Proceedings Volumes*, vol. 41, no. 2, pp. 10822–10825, 2008.

[5] H. Chai *et al.*, "A survey of the development of quadruped robots: Joint configuration, dynamic locomotion control method and mobile manipulation approach," *Biomimetic Intelligence and Robotics*, vol. 2, no. 1, p. 100029, 2022.

[6] Ghost Robotics, "VISION 60 | Ghost Robotics," *Ghost Robotics*, 2022. https://www.ghostrobotics.io/vision-60 (accessed May 18, 2022).

[7] M. Hutter *et al.*, "Anymal-toward legged robots for harsh environments," *Advanced Robotics*, vol. 31, no. 17, pp. 918–931, 2017.

[8] Boston Dynamics, "SPOT SDK Boston Dynamics," *Boston Dynamics*, 2022. https://dev.bostondynamics.com/ (accessed Apr. 01, 2022).

[9] H.-W. Huang *et al.*, "Mobile Robotic Platform for Contactless Vital Sign Monitoring," *Cyborg and Bionic Systems*, vol. 2022, 2022.

[10] K. Cho, S. H. Baeg, and S. Park, "3D pose and target position estimation for a quadruped walking robot," in *2013 10th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, 2013, pp. 466–467.

[11] Z. Jiang, H. Zhang, J. Xu, G. Tian, and X. Rong, "Real-Time Target Detection and Tracking System Based on Stereo Camera for Quadruped Robots," in *2019 Chinese Control And Decision Conference (CCDC)*, 2019, pp. 2949–2954.

[12] S. C. Nandakumar *et al.*, "Bio-inspired Multi-robot Autonomy," *arXiv preprint arXiv:2203.07718*, 2022.

[13] I. D. Miller *et al.*, "Mine tunnel exploration using multiple quadrupedal robots," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2840–2847, 2020.

[14] C. Cruz Ulloa, G. Prieto Sánchez, A. Barrientos, and J. del Cerro, "Autonomous thermal vision robotic system for victims recognition in search and rescue missions," *Sensors*, vol. 21, no. 21, p. 7346, 2021.

[15] S. S. A. Zaidi, M. S. Ansari, A. Aslam, N. Kanwal, M. Asghar, and B. Lee, "A survey of modern deep learning based object detection models," *Digital Signal Processing*, p. 103514, 2022.

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Adv Neural Inf Process Syst*, vol. 25, 2012.

[17] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[18] H. Jiang and E. Learned-Miller, "Face detection with the faster R-CNN," in *2017 12th IEEE international conference on automatic face & gesture recognition (FG 2017)*, 2017, pp. 650–657.

[19] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[20] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," *arXiv*, 2018.

[21] Q.-C. Mao, H.-M. Sun, Y.-B. Liu, and R.-S. Jia, "Mini-YOLOv3: real-time object detector for embedded applications," *Ieee Access*, vol. 7, pp. 133529–133538, 2019.

[22] G. Jocher *et al.*, "ultralytics/yolov5: v6.1 - TensorRT, TensorFlow Edge TPU and OpenVINO Export and Inference," Feb. 2022, doi: 10.5281/ZENODO.6222936.

[23] NVIDIA, "Jetson Nano," *NVIDIA Corporation*, 2022. https://elinux.org/Jetson_Nano (accessed Apr. 01, 2022).

[24] D. Tzutalin, "LabelImg," *GitHub Repository*, vol. 6, 2015.

[25] N. Ketkar and J. Moolayil, "Introduction to pytorch," in *Deep learning with python*, Springer, 2021, pp. 27–91.

[26] F. Zhuang *et al.*, "A comprehensive survey on transfer learning," *Proceedings of the IEEE*, vol. 109, no. 1, pp. 43–76, 2020.

[27] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[28] D. Mané and others, "TensorBoard: TensorFlow's visualization toolkit," *Retrieved October*, vol. 8, p. 2021, 2015.

[29] R. Padilla, S. L. Netto, and E. A. B. da Silva, "A survey on performance metrics for object-detection algorithms," in *2020 international conference on systems, signals and image processing (IWSSIP)*, 2020, pp. 237–242.

[30] R. Padilla, W. L. Passos, T. L. B. Dias, S. L. Netto, and E. A. B. da Silva, "A Comparative Analysis of Object Detection Metrics with a Companion Open-Source Toolkit," *Electronics (Basel)*, vol. 10, no. 3, 2021, doi: 10.3390/electronics10030279.

10 Anexos

Anexo 1. Documentación de clase desarrollada spot_control.py

10.1.1 Invocación

La clase desarrollada se invoca de la siguiente manera:

import spot_control

spot = spot_control.Control(...)

10.1.2 Configuración

Contiene 11 parámetros de configuración accesibles en la función __init__.

Tabla 11. Descripción de los parámetros de configuración de la clase *spot_control.py*.

Parámetro	Valor por defecto	Rango	Descripción
user	user	Definido por el usuario	Nombre de usuario para autenticar con Spot
password	**********	Definido por el usuario	Contraseña de autenticación
host	192.168.50.3	Definido por el usuario	IP en la que se encuentra conectado Spot
VELOCITY_BASE_SPEED	0.5	0.0 – 1.6 ms	Velocidad base de movimiento
VELOCITY_BASE_ANGULAR	0.8	0.0 – 1 rad/s	Velocidad angular
VELOCITY_CMD_DURATION	0.6	0.0 – 1 s	Tiempo de duración para la velocidad
HEIGHT_MAX	0.3	0.0 – 1 m	Altura máxima
ROLL_OFFSET_MAX	0.4	0.0 – 1 rad/s	Giro máximo para Roll
YAW_OFFSET_MAX	0.7805	0.0 – 1 rad/s	Giro máximo para Yaw
PITCH_OFFSET_MAX	0.7805	0.0 – 1 rad/s	Giro máximo para Ptch
HEIGHT_CHANGE	0.1	0.0 – 0.3 m	Pasos para cambio de altura

10.1.3 Funciones Disponibles

10.1.3.1 connect

Realiza la conexión con Spot, así también permite el servicio paro de emergencia *e-stop*. Además, esta función toma el control de Spot mediante *self.lease_client.take()* sin importar que otro cliente se encuentre conectado. Retorna el estado de la conexión True/False.

state = spot.connect()

10.1.3.2 stop

Inicia la parada de emergencia de Spot el cual es necesario para el funcionamiento y operación. Se realiza la llamada dentro de la función *connect,* por lo que, no es necesario su utilización de manera externa.

spot.stop()

10.1.3.3 power_on

Prepara Spot para el encendido de sus motores. Debe estar presionado el pulsador físico de la parte posterior que corresponde a habilitar la sección de potencia de los motores.

spot.power_on()

10.1.3.4 power_off

Realiza el apagado de motores de manera segura. Si Spot está caminando o realizando alguna acción con el brazo, al activar esta función, recogerá el brazo y apagará de manera segura.

spot.power_off()

10.1.3.5 standing

Activa el modo *standing* de Spot el cual permite mover su cuerpo en el mismo sitio, es decir, permite realizar movimientos en roll, pitch y yaw.

spot.standing()

10.1.3.6 stairs_mode

Activa el modo escaleras. El movimiento es más lento y cuidadoso.

spot.stairs_mode()

10.1.3.7 run_docking

Disponible para Spot Enterprise. Al activar esta función, ejecuta la secuencia automática para ubicarse sobre el Dock de carga automática. En el caso que no tenga visible ningún fiducial de referencia, no ejecutará ninguna acción.

spot.run_docking()

10.1.3.8 run_undocking

Disponible para Spot Enterprise. Al activar esta función, ejecuta la secuencia automática para levantarse del Dock de carga automática. En el caso que no se encuentre sobre el Dock no ejecutará ninguna acción.

spot.run_undocking()

10.1.3.9 tall

Mantiene una altura definida.

Tabla 12. Descripción del parámetro de la función *tall*.

Parámetro	Valor por defecto	Rango	Descripción
body_height_	0.1	0 - 1	Altura del cuerpo de Spot

spot.tall(0.1)

10.1.3.10 unstow_hand

Disponible para Spot con brazo articulado. Extiende el brazo a una posición estándar.

spot.unstow_hand()

10.1.3.11 stow_hand

Disponible para Spot con brazo articulado. Recoge el brazo a una posición estándar.

spot.stow_hand()

10.1.3.12 move_body

Permite mover el cuerpo de Spot en sus 3 ejes. Se debe tener cuidado al mover sobre roll en el caso que lleve carga extra sobre el lomo.

Tabla 13. Descripción de parámetros de la función *move_body*.

Parámetro	Valor por defecto	Rango	Descripción
yaw	0.0	0.0 – 1.0	Movimiento sobre Yaw
roll	0.0	0.0 – 1.0	Movimiento sobre Roll
pitch	0.0	0.0 – 1.0	Movimiento sobre Pitch

spot.move_body(yaw=0,roll=0,pitch=0)

10.1.3.13 move

Permite mover Spot por el entorno.

Tabla 14. Descripción de los parámetros de la función *move*.

Parámetro	Valor por defecto	Rango	Descripción
x	0.0	-1.0 – 1.0	Velocidad de movimiento hacia adelante (1) y atrás (-1)
y	0.0	-1.0 – 1.0	Velocidad de movimiento hacia izquierda (1) y derecha (-1)
z	0.0	-1.0 – 1.0	Velocidad de rotación sobre yaw, sentido horario (1) y antihorario (-1)

spot.move(x=0,y=0,z=0)

10.1.3.14 get_video

El acceso a las cámaras por el servicio del SDK entrega sin codificar, por lo que era complejo trabajar directamente con ello. Esta función codifica con cv2 para retornar la información de los frames y lista para procesar.

Tabla 15. Descripción del parámetro de la función *get_video*.

Parámetro	Valor por defecto	Rango	Descripción
name_cam	hand_color_image	Cámaras disponibles	Nombre de la cámara que se desea acceder

Lista de cámaras disponibles:

- back_depth
- back_depth_in_visual_frame
- back_fisheye_image
- frontleft_depth
- frontleft_depth_in_visual_frame
- frontleft_fisheye_image
- frontright_depth
- frontright_depth_in_visual_frame
- frontright_fisheye_image
- hand_color_image
- hand_color_in_hand_depth_frame

- hand_depth
- hand_depth_in_hand_color_frame
- hand_image
- left_depth
- left_depth_in_visual_frame
- left_fisheye_image
- right_depth
- right_depth_in_visual_frame
- right_fisheye_image

frame = spot.get_video('hand_color_image')

10.1.3.15 hand_control

Permite mover el brazo de Spot mediante punto final creando un plano XYZ para el griper. El eje X corre a lo largo del cuerpo, el eje Y es perpendicular a X y va de izquierda a derecha y Z es perpendicular a XY.

Por seguridad del movimiento por punto final se recomienda no superar el Rango de los parámetros especificados, ya que el brazo puede efectuar movimientos bruscos que pueden afectar a la integridad del brazo.

Tabla 16. Descripción de los parámetros de la función *hand_control*.

Parámetro	Valor por defecto	Rango	Descripción
x	0.75	0.75 – 1.1	Coordenada de punto final sobre el eje X, movimiento hacia atrás y adelante
y	0.0	-0.5 – 0.5	Coordenada de punto final sobre el eje Y, movimiento de izquierda a derecha
z	0.5	-0.25 – 0.8	Coordenada de punto final sobre el eje Z, movimiento de abajo hacia arriba
griper	0.0	0.0 – 1.0	Apertura del griper
time_sample	0.5	0.1 – 5	Tiempo de muestreo en segundos para la ejecución de los comandos

spot.hand_control(x=0.75,y=0,z=0.25,grip=0,time_sample=0.5)

10.1.3.16 hand_control_join

Permite mover de madera independiente cada una de las junturas que incluyen el brazo, incluyendo el griper.

Tabla 17. Descripción de los parámetros de la función *hand_control_join*.

Parámetro	Valor por defecto	Rango	Descripción
cmd_sh0	0.692	--	Velocidad de movimiento de motor base de 330º
cmd_sh1	-1.882	--	Velocidad de movimiento de motor base de 210º
cmd_el0	1.652	--	Velocidad de movimiento primera sección de motor de 180º
cmd_el1	-0.0691	--	Velocidad de movimiento primera sección de motor de 320º
cmd_wr0	0.0	--	Velocidad de movimiento segunda sección de motor de 210º
cmd_wr1	0.0	--	Velocidad de movimiento segunda sección de motor de 330º
cmd_grip	0.0	--	Velocidad de movimiento de griper de motor de 90º
time_sample	0.5	--	Tiempo de muestreo en segundos para la ejecución de los comandos

spot.hand_control(cmd_sh0=0.0692,cmd_sh1=-1.882,cmd_el0=1.652,cmd_el1=-0.0691,cmd_wr0=0,cmd_wr1=0,cmd_grip=0,time_sample=0.5)

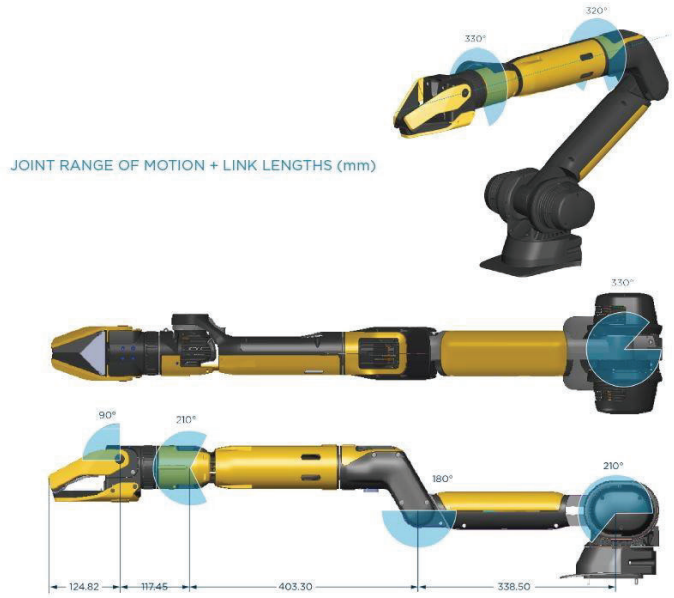


Figura 30. Descripción del brazo articulado compatible con el robot Spot. [8]

10.1.3.17 carry

Permite mover una carga tomada por el brazo de Spot y trasladarla por el entorno en conjunto.

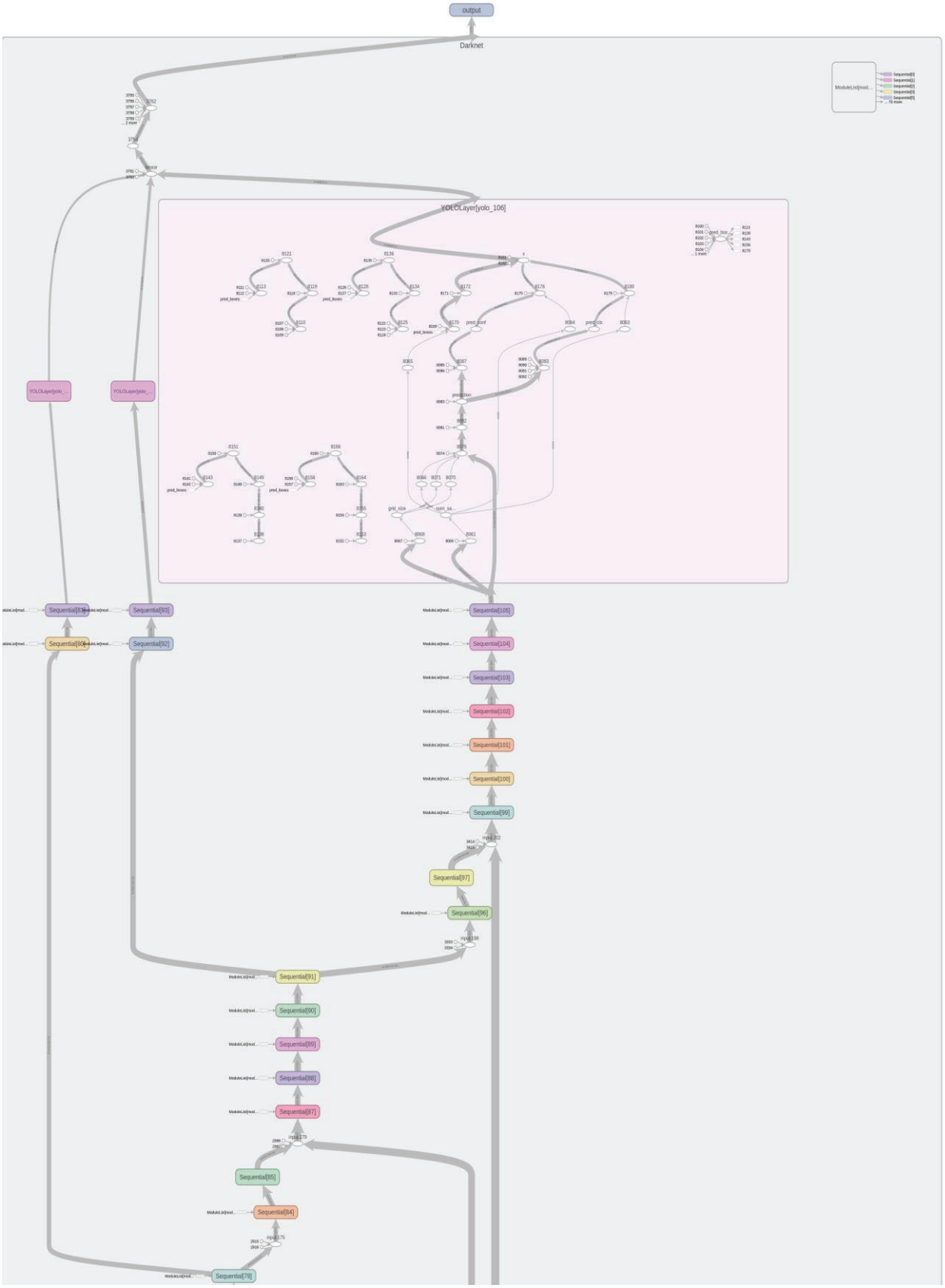
spot.carry()

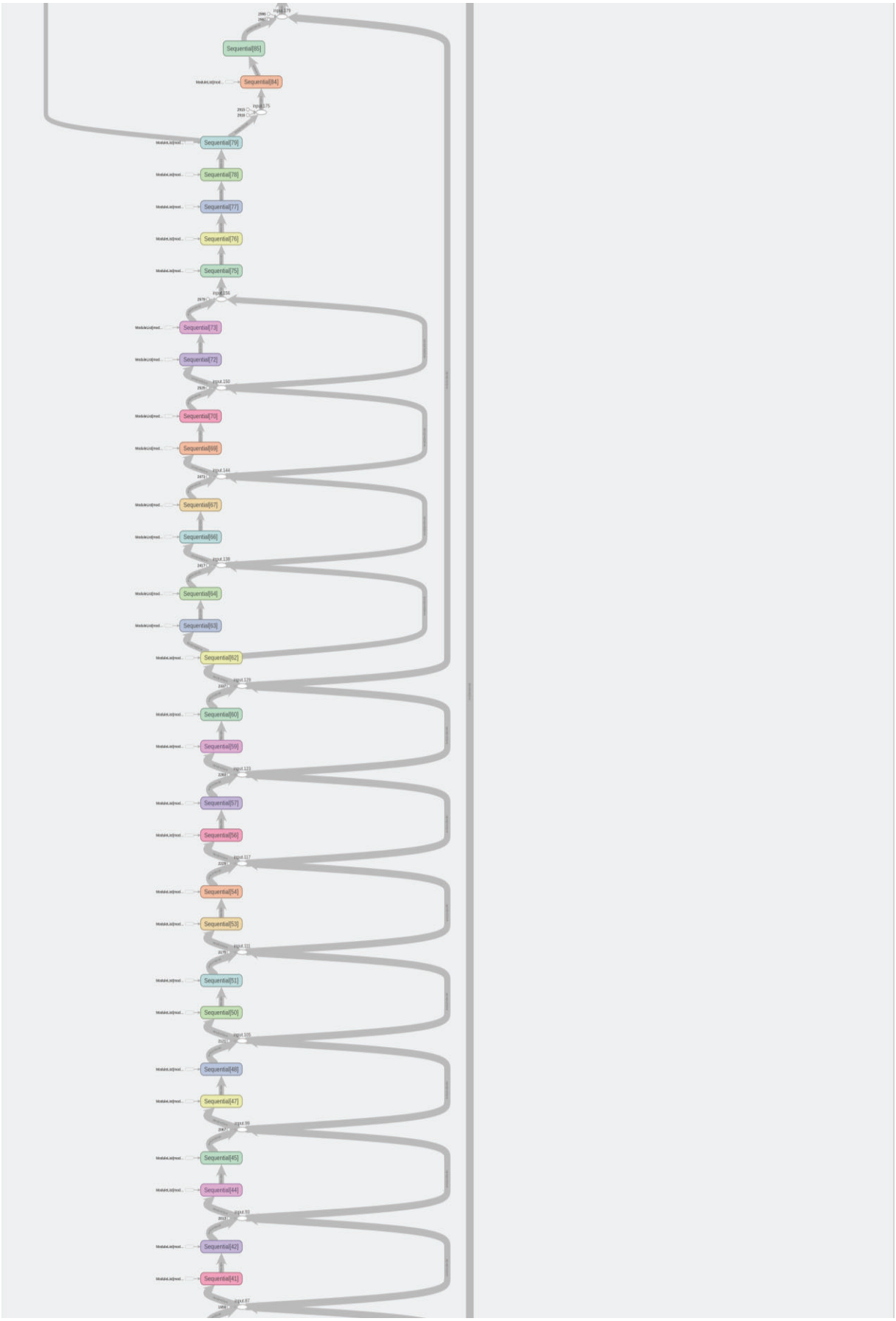
10.1.3.18 misión_hand

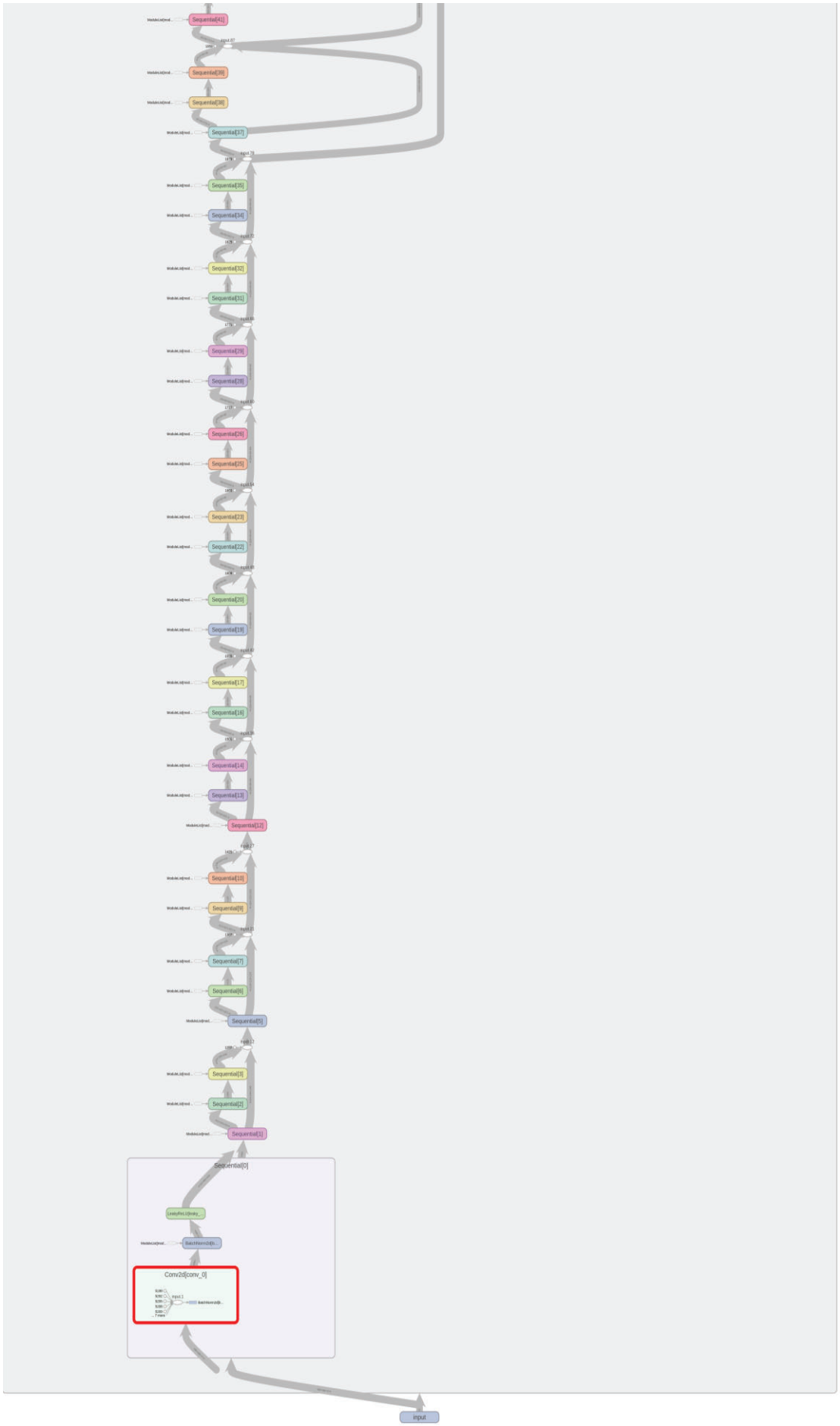
Permite grabar misiones con el brazo robótico para luego ser reproducidas a discreción del usuario. Estas se graban en un formato *.csv,* por lo que es de fácil manipulación y permanecen almacenadas en disco.

spot.mision()

Anexo 1. Representación del modelo entrenado generado por tensorboard.




Sequential[85]
Sequential[84]
Sequential[79]
Sequential[78]
Sequential[77]
Sequential[76]
Sequential[75]
Sequential[73]
Sequential[72]
Sequential[70]
Sequential[69]
Sequential[67]
Sequential[66]
Sequential[64]
Sequential[63]
Sequential[62]
Sequential[60]
Sequential[59]
Sequential[57]
Sequential[56]
Sequential[54]
Sequential[53]
Sequential[51]
Sequential[50]
Sequential[48]
Sequential[47]
Sequential[45]
Sequential[44]
Sequential[42]
Sequential[41]


Sequential[41]
Sequential[39]
Sequential[38]
Sequential[37]
Sequential[35]
Sequential[34]
Sequential[32]
Sequential[31]
Sequential[29]
Sequential[28]
Sequential[26]
Sequential[25]
Sequential[23]
Sequential[22]
Sequential[20]
Sequential[19]
Sequential[17]
Sequential[16]
Sequential[14]
Sequential[13]
Sequential[11]
Sequential[10]
Sequential[9]
Sequential[7]
Sequential[6]
Sequential[5]
Sequential[3]
Sequential[2]
Sequential[1]
Sequential[0]
Conv2d[conv_0]
input