



MACHINE LEARNING CON NVIDIA JETSON Y MATLAB

Despliegue de Modelos de ML con ROS y Jetson

Diego Banda R.
Universidad Católica San Pablo
diego.banda@ucsp.edu.pe





AGENDA

- Repaso de lo visto.
- ¿Qué es ROS?
- Código de MATLAB para el despliegue.
- Código de Jetson para ROS.
- ¡Desplegar el modelo y a jugar!



Repasemos lo trabajado

Y un vistazo rápido de lo que va a pasar...



Entorno Local



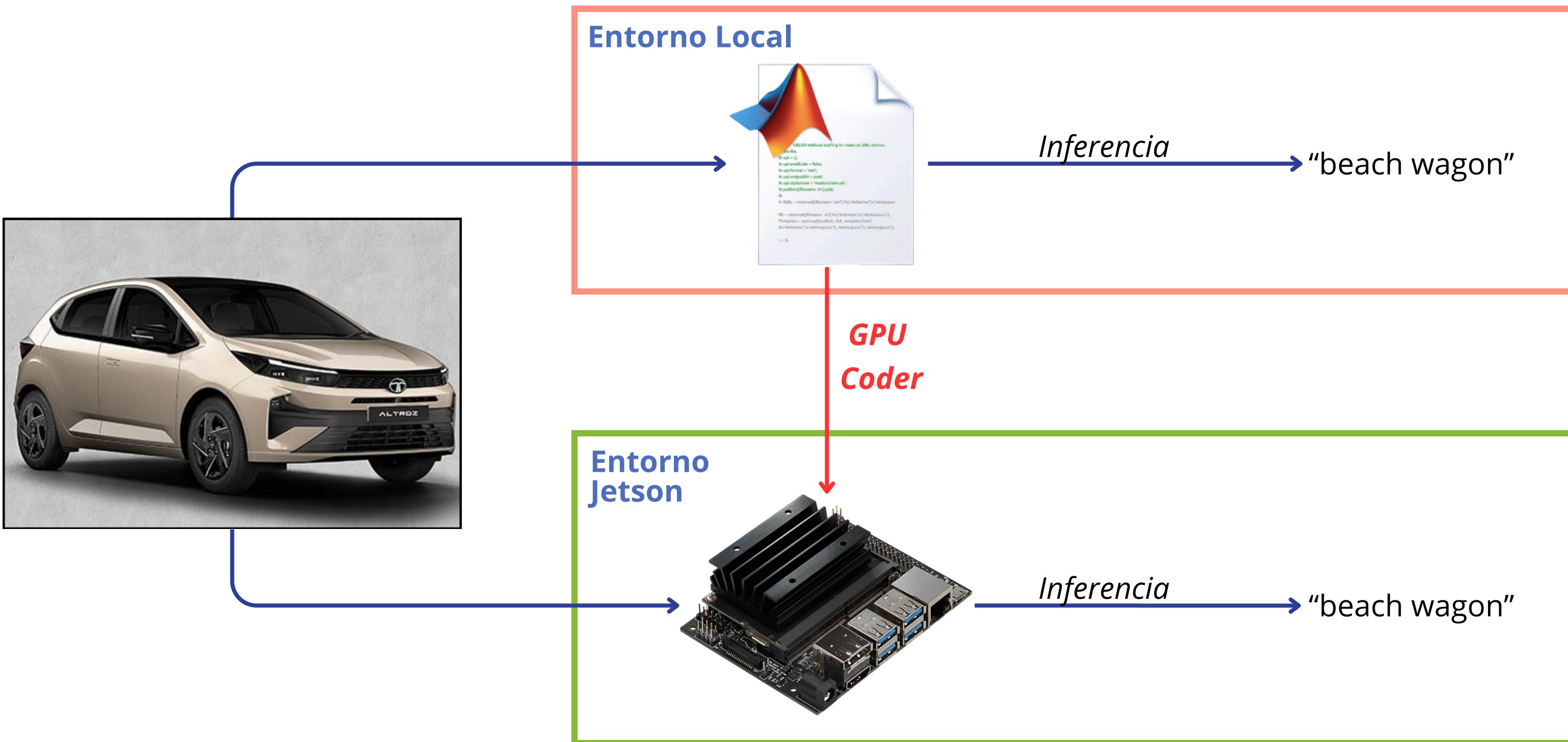
Inferencia

"beach wagon"



Repasemos lo trabajado

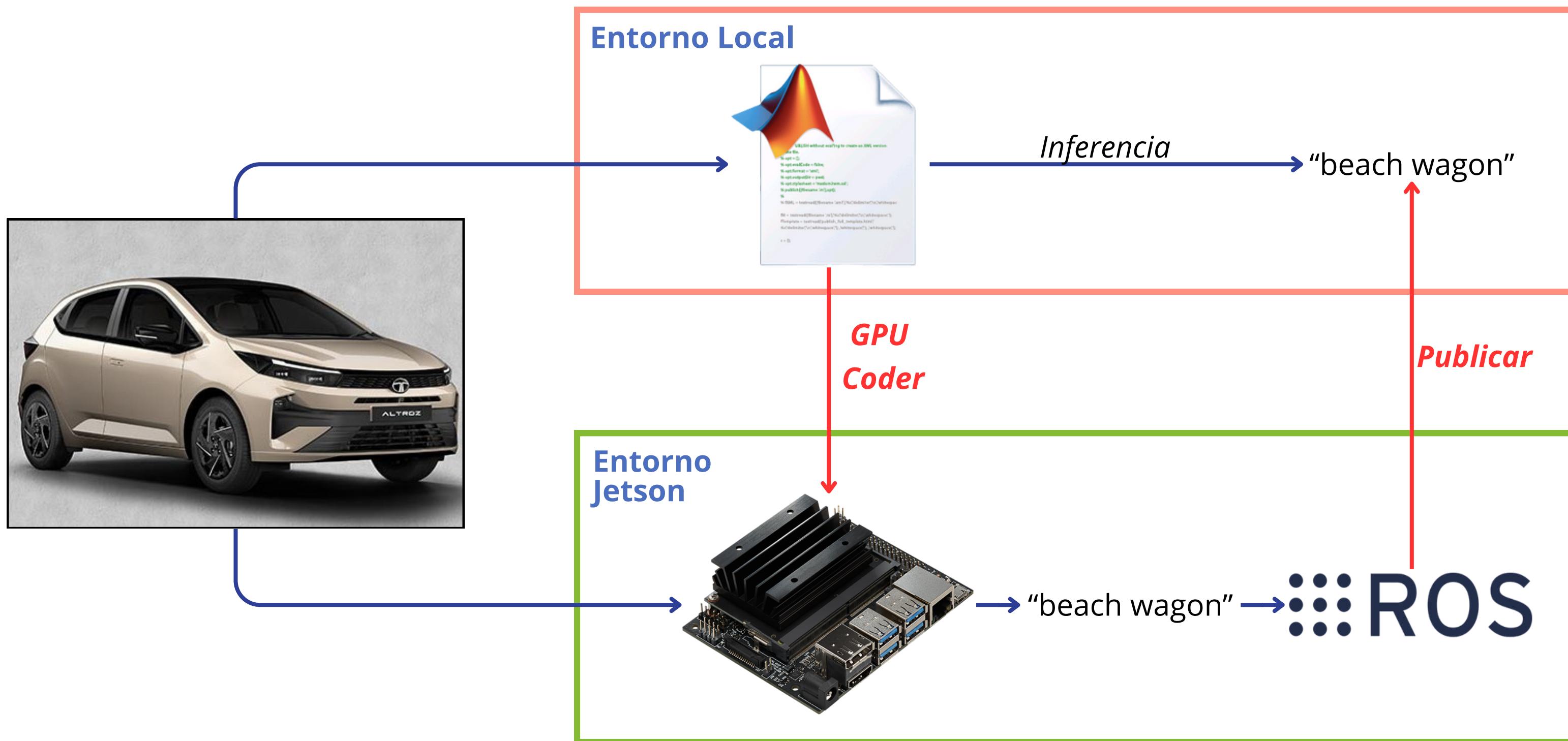
Y un vistazo rápido de lo que va a pasar...





Repasemos lo trabajado

Y un vistazo rápido de lo que va a pasar...

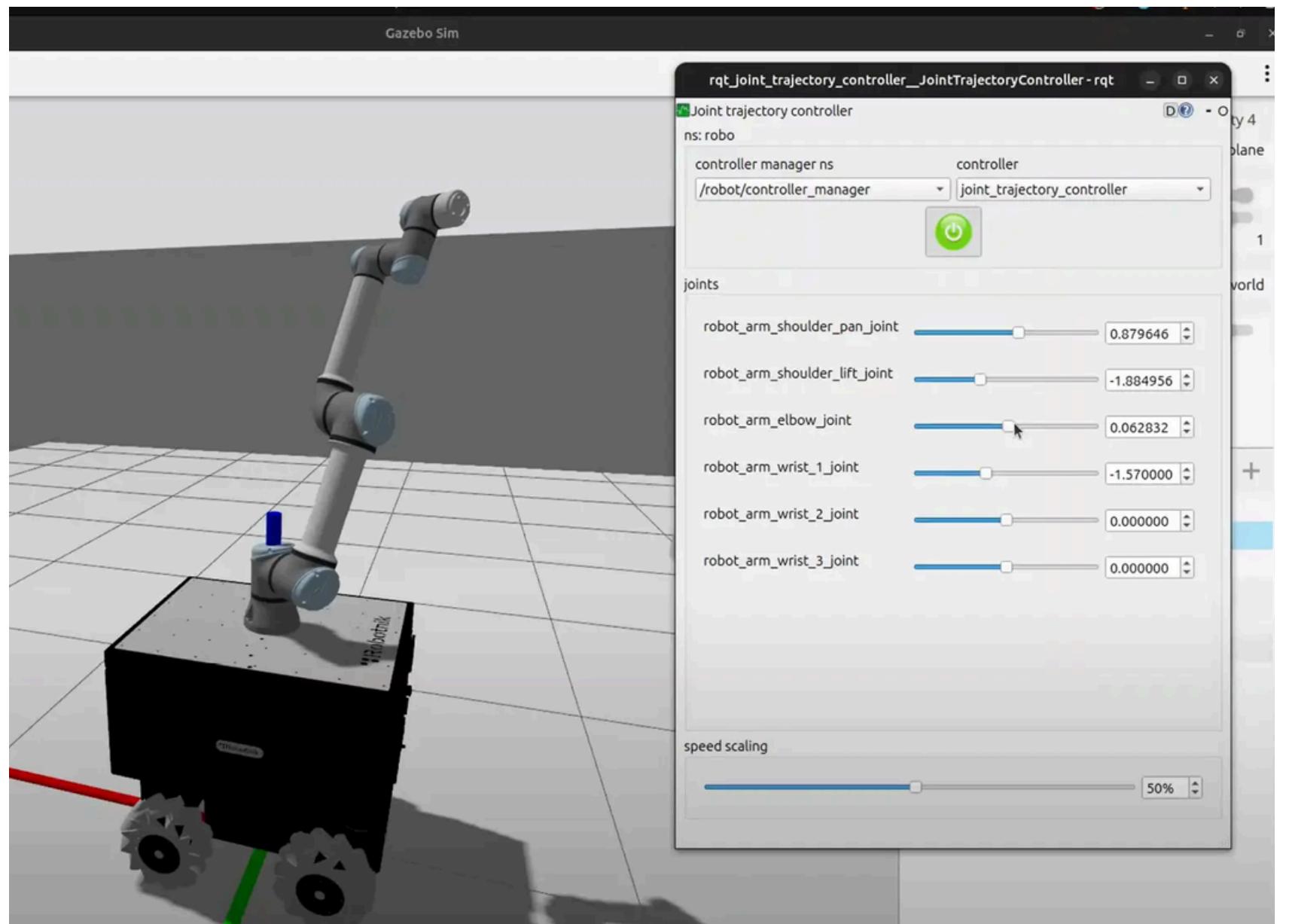




¿Qué es ROS?

- ROS es un **sistema operativo de código abierto** para la **robótica**.
- Conjunto de **bibliotecas de software** y **herramientas** que facilitan el desarrollo de **aplicaciones robóticas** compatibles con una **amplia variedad de plataformas**.
- Fue desarrollado originalmente en **2007** en el **Laboratorio de Inteligencia Artificial de la Universidad de Stanford**, y su desarrollo continuó en Willow Garage.
- Desde 2013 es gestionado por la **Open Source Robotics Foundation (OSRF)**.

ROS





Filosofía de ROS

- **Punto a punto (Peer to Peer):** Los sistemas ROS están compuestos por numerosos programas pequeños que se conectan entre sí y mantienen un intercambio continuo de mensajes.
- **Basado en herramientas:** Existen múltiples programas pequeños y genéricos que realizan tareas como visualización, registro, graficado de flujos de datos, entre otros.
- **Multilingüe:** Los módulos de software en ROS pueden desarrollarse en cualquier lenguaje para el cual exista una biblioteca cliente. Actualmente, hay bibliotecas disponibles para C++, Python, LISP, Java, JavaScript, MATLAB, Ruby, entre otros.
- **Ligero (Thin):** Las convenciones de ROS fomentan que los colaboradores creen bibliotecas independientes y que posteriormente las integren mediante envoltorios (wrappers) que permitan enviar y recibir mensajes con otros módulos ROS.
- **¡Libre y de código abierto!**

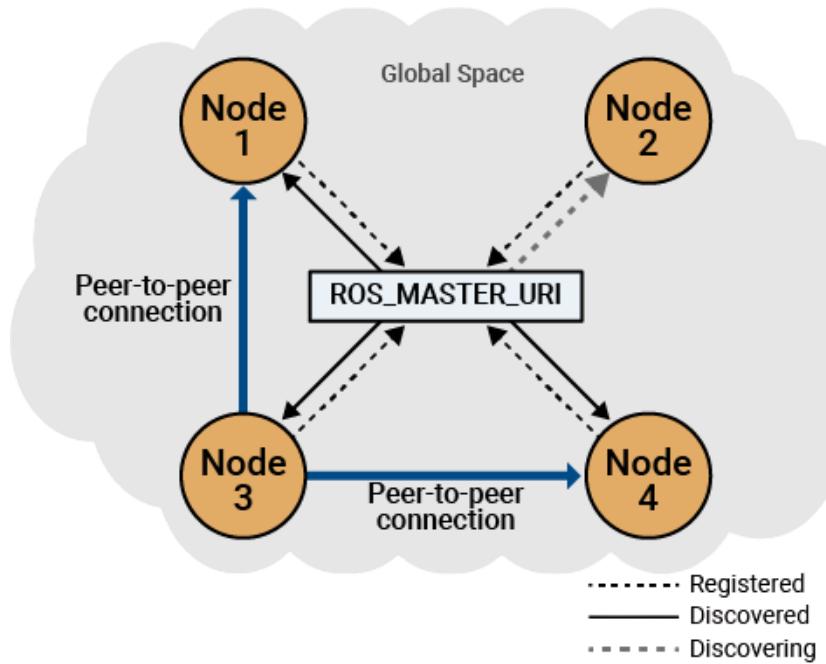


Las dos caras de ROS

Sistema Operativo

Proporciona servicios estándar de un sistema operativo, tales como:

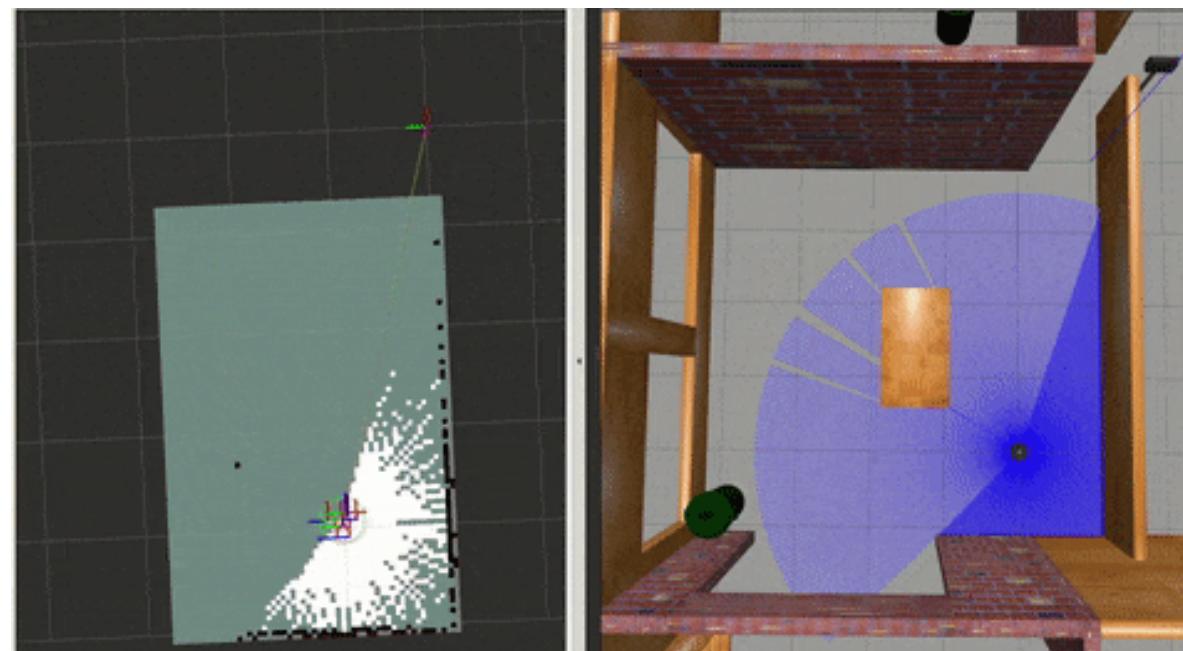
- Abstracción de hardware.
- Control de dispositivos de bajo nivel.
- Funcionalidades de uso común.
- Intercambio de mensajes entre procesos.
- Gestión de paquetes....



Paquetes contribuidos por la comunidad

Conjunto de paquetes desarrollados por usuarios que implementan funcionalidades robóticas habituales, como:

- Localización y Mapeo Simultáneo (SLAM).
- Planificación.
- Percepción.
- Manipulación...





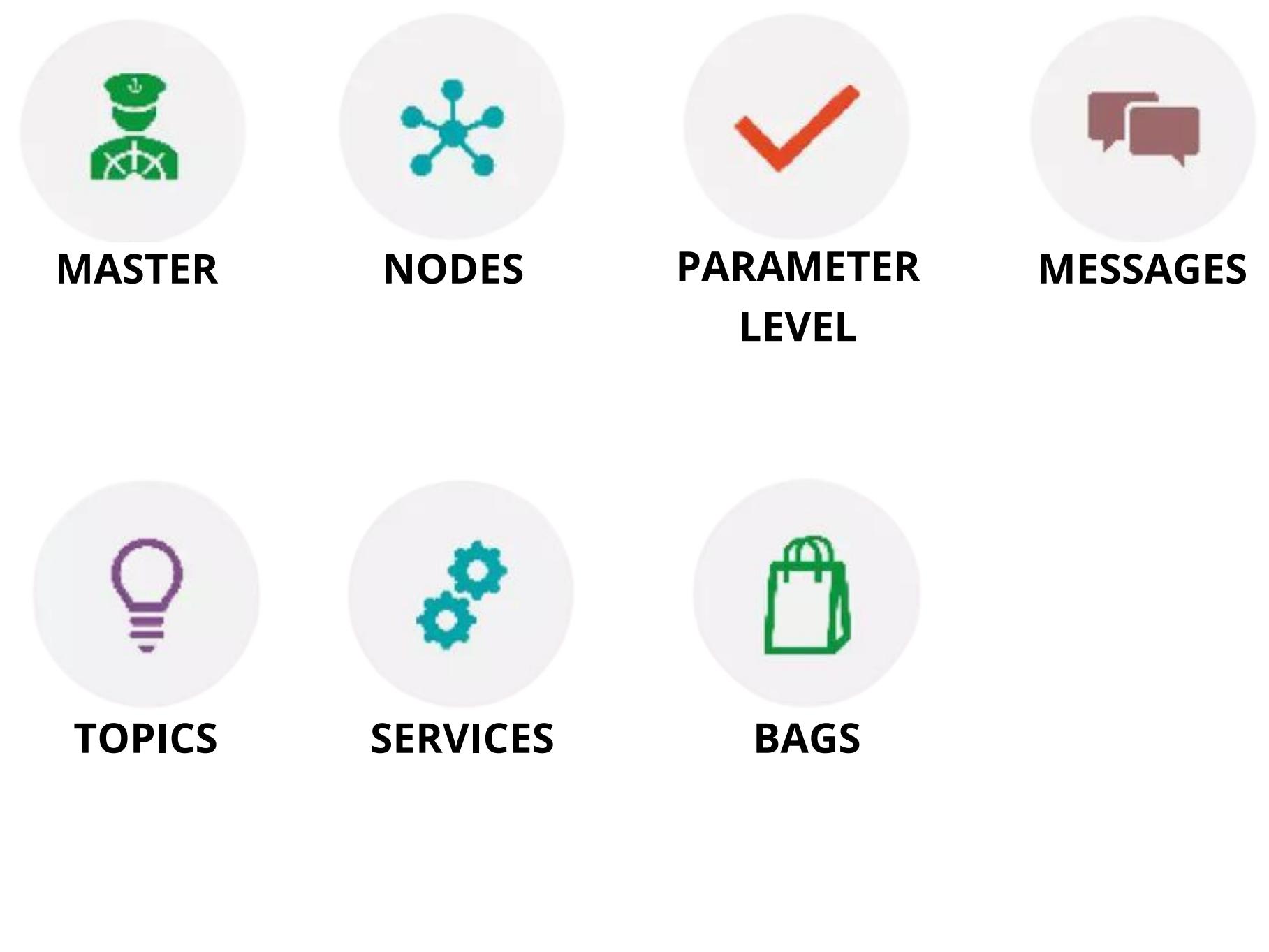
Conceptos Esenciales de ROS

NIVEL GRÁFICO COMPUTACIONAL DE ROS (CGL)

El Gráfico Computacional de ROS es la red peer-to-peer de procesos de ROS que colaboran en el procesamiento de datos.

Los componentes básicos del Gráfico Computacional de ROS son:

- Los Nodos.
- El Maestro.
- El Servidor de Parámetros.
- Los Mensajes.
- Los Servicios.
- Los Tópicos.
- Las Bolsas (Bags).





Conceptos Esenciales de ROS



MASTER

- Servidor que registra y gestiona las direcciones IP de los nodos, permitiendo su descubrimiento e interconexión.
- Proporciona la información necesaria para que los nodos establezcan conexiones peer-to-peer y puedan intercambiar mensajes.



NODES

- Procesos que realizan tareas específicas dentro del sistema de control de un robot (ej.: control de motores, sensores, localización, planificación).
- Se implementan mediante bibliotecas cliente de ROS (roscpp en C++, rospy en Python) y pueden publicar/suscribirse a tópicos o proveer/usar servicios.



Conceptos Esenciales de ROS



**PARAMETER
LEVEL**

- Diccionario compartido accesible a través de APIs de red, utilizado para almacenar parámetros estáticos de configuración.
- Se ejecuta dentro del Maestro.



MESSAGES

- Estructuras de datos tipadas que permiten la comunicación entre nodos.
- Admiten tipos primitivos (enteros, booleanos, flotantes) y estructuras anidadas, similares a structs en C.



TOPICS

- Canales de comunicación que siguen el modelo publish/subscribe: un nodo publica mensajes y otros se suscriben para recibirlas.
- Permiten desacoplar la producción y el consumo de datos; múltiples publicadores y suscriptores pueden coexistir en un mismo tópico.



Conceptos Esenciales de ROS



SERVICES

- Comunicación síncrona request/response entre nodos, útil cuando se requiere una transacción directa en lugar de transmisión continua de datos.
- Un nodo ofrece un servicio y otro actúa como cliente solicitando la operación, similar a una llamada a procedimiento remoto (RPC).



BAGS

- Formato para almacenar y reproducir datos de mensajes de ROS.
- Esencial para registrar información (p. ej., de sensores) y reutilizarla en pruebas y desarrollo de algoritmos.



Creando una Aplicación ROS

- ROS Workspace.
- ROS Package.
- ROS Nodes.





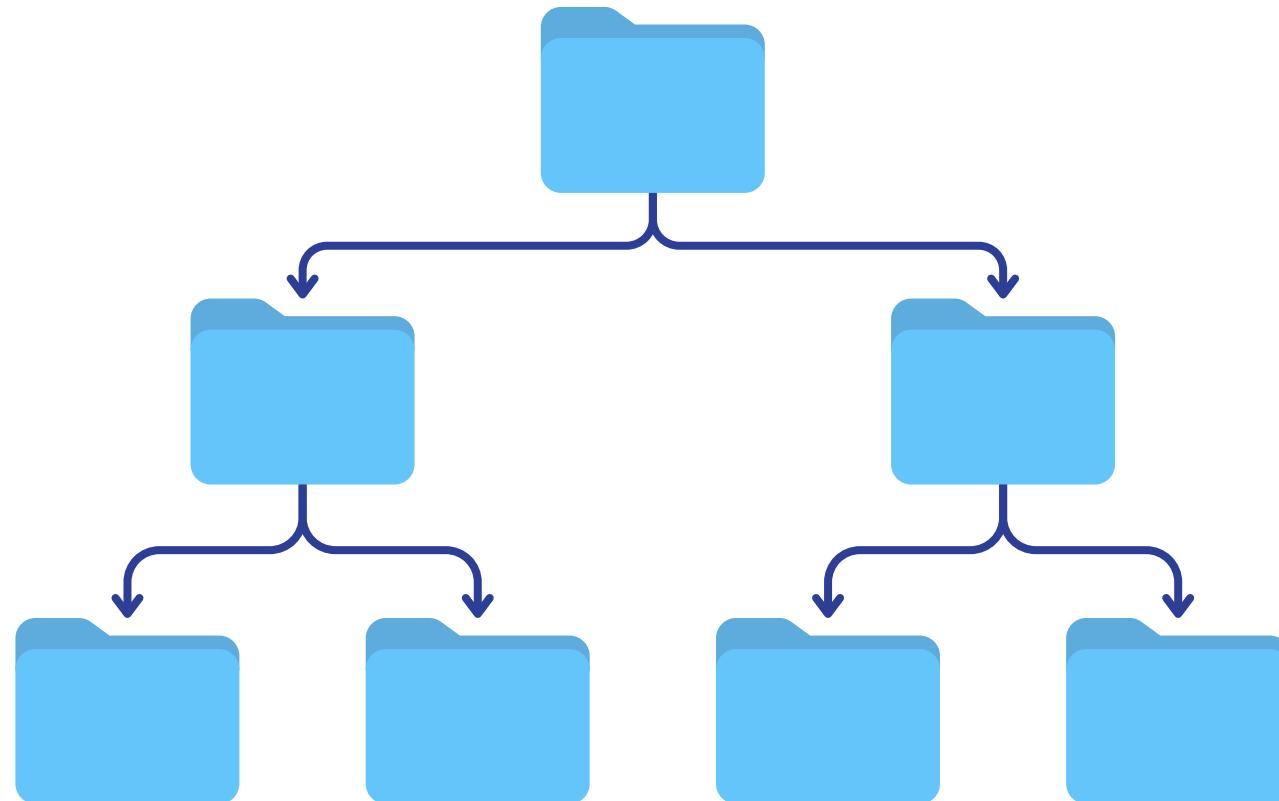
Creando una Aplicación ROS

- ROS Workspace.
- ROS Package.
- ROS Nodes.





ROS Workspace



- El primer paso en el desarrollo con ROS es la creación del espacio de trabajo (workspace), que es donde se almacenan los paquetes de ROS.
- En él se pueden crear nuevos paquetes, instalar paquetes existentes, compilar y generar nuevos ejecutables.
- Un espacio de trabajo es, sencillamente, un conjunto de directorios en los que reside un conjunto relacionado de código de ROS.



ROS Workspace

3 carpetas para dominar ROS...

SRC Folder

Código fuente y paquetes

- Es el lugar donde se crean o clonian nuevos paquetes desde repositorios.
- Facilita la organización del código fuente y la gestión de versiones mediante sistemas como Git.
- Los paquetes de ROS solo se compilan y generan ejecutables cuando se encuentran dentro de esta carpeta.

Build Folder

Archivos de compilación

- La herramienta catkin genera en esta carpeta los archivos de compilación y las cachés intermedias de CMake.
- Estos archivos evitan la recompilación completa de todos los paquetes al ejecutar el comando `catkin_make`.
- Permite un proceso de compilación más eficiente, reduciendo el tiempo de desarrollo.

Devel Folder

Ejecutables y entorno local

- Aquí se almacenan los ejecutables resultantes de la compilación de los paquetes.
- Contiene scripts de Shell para añadir el espacio de trabajo actual a la ruta de trabajo de ROS.
- Actúa como un entorno de desarrollo local donde se prueban los paquetes antes de ser instalados en el sistema.



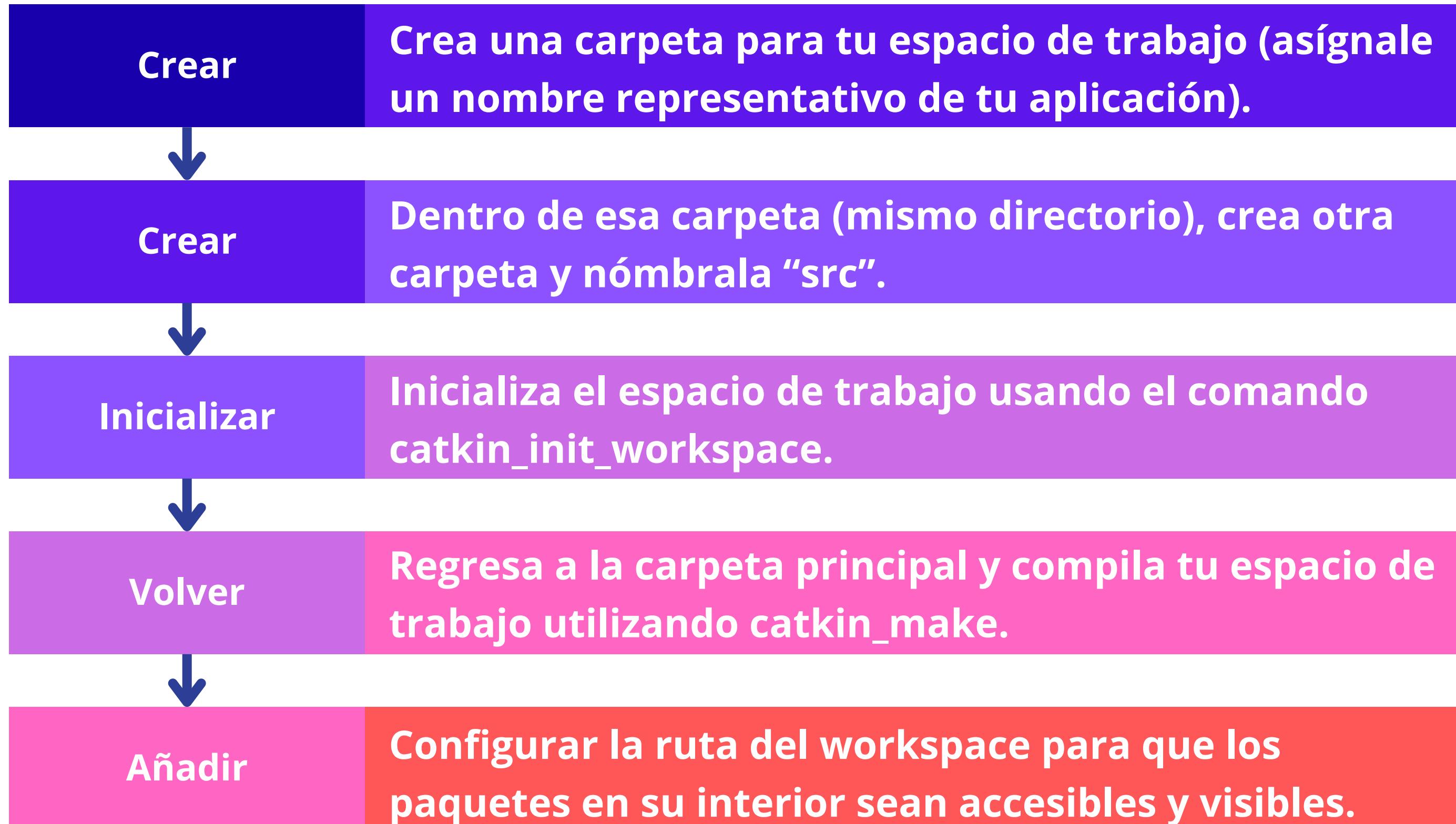
ROS Workspace

Estructura común del workspace de ROS

```
/home/ucspjason/catkin_ws/
└ src/
    └ msra_deployed_nn/
        ├── CMakeLists.txt      # Build system del paquete
        ├── package.xml         # Metadatos del paquete ROS
        └── launch/
            └── alexnet_usb.launch # Launch file para lanzar el nodo con cámara USB
        └── src/
            └── alexnet.cpp       # Nodo C++ que invoca la biblioteca generada
            └── alexnet.hpp        # Interfaz de la clase que maneja la inferencia
```

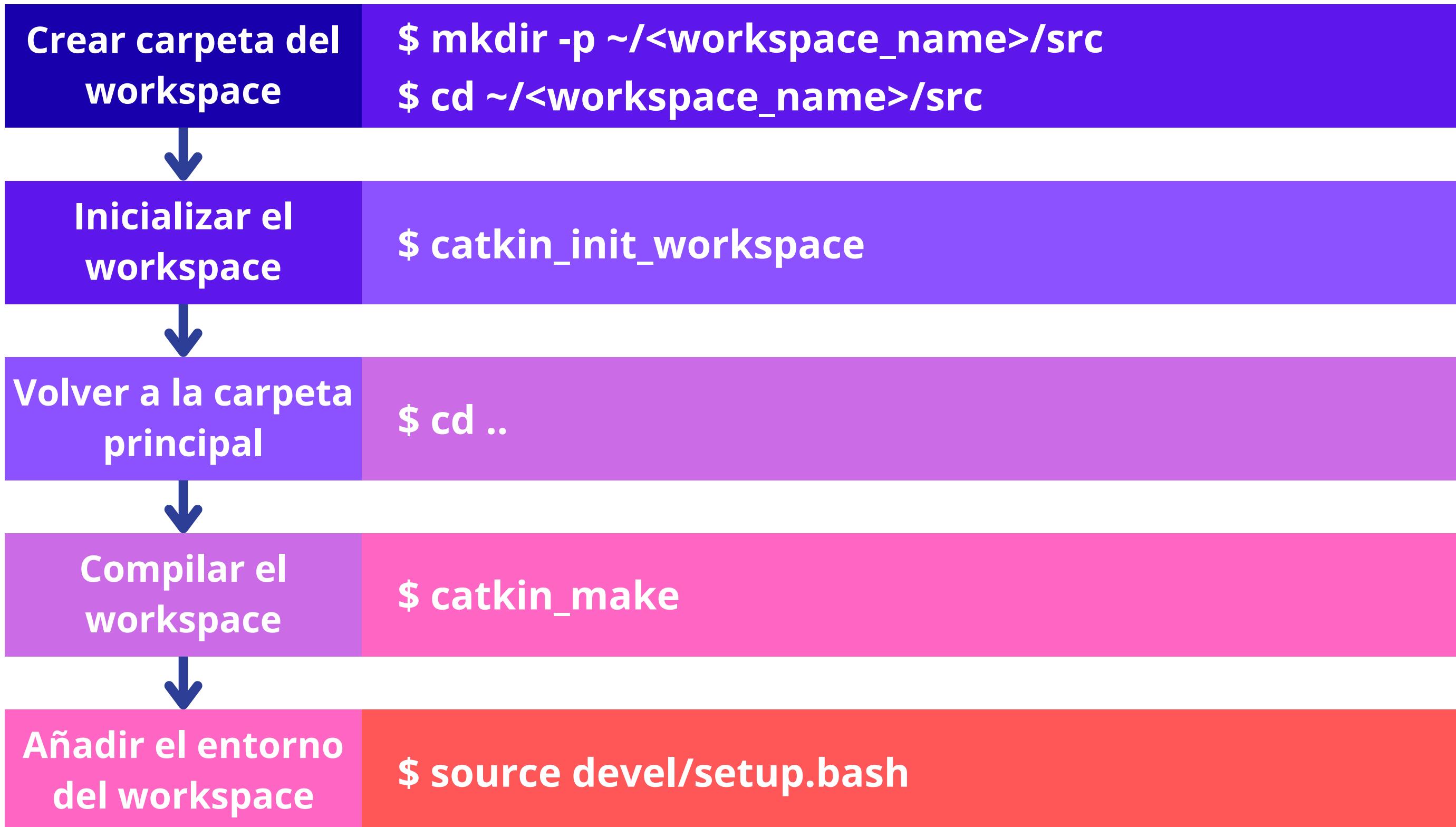


¿Cómo crear un ROS Workspace?





¿Cómo crear un ROS Workspace?





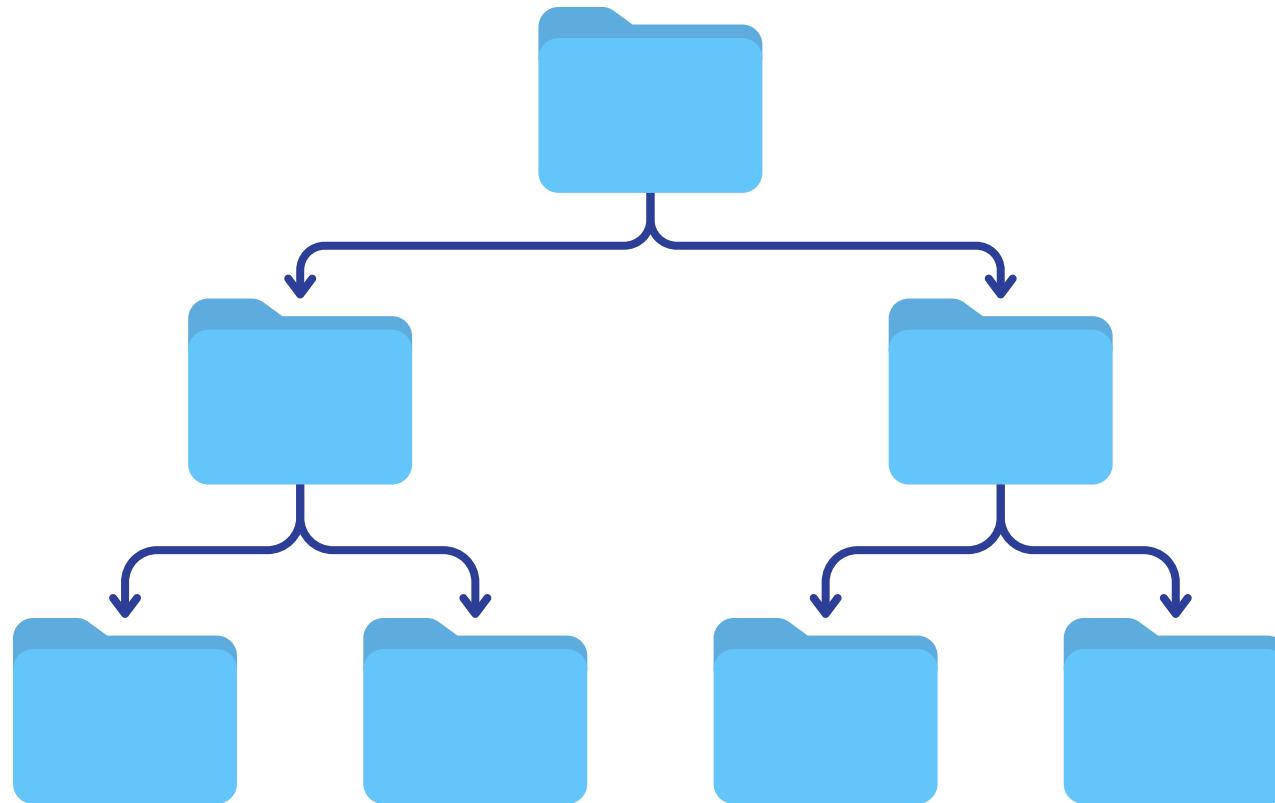
Creando una Aplicación ROS

- ROS Workspace.
- ROS Package.
- ROS Nodes.



ROS Package

¿Qué es un paquete en ROS?



- El software de ROS está organizado en paquetes, cada uno de los cuales contiene alguna combinación de código, datos y documentación.
- El paquete de ROS es el lugar donde se organizan los nodos de ROS, así como las bibliotecas y demás componentes asociados.



¿Cuál es el propósito de estos archivos y carpetas ROS?

CMakeLists.txt

- Contiene todas las instrucciones necesarias para compilar el código fuente del paquete de ROS.
- Define cómo deben construirse los ejecutables y las bibliotecas asociadas, gestionando la integración con catkin y CMake.

package.xml

- Archivo en formato XML que describe la información esencial del paquete (nombre, versión, autor, licencia, etc.).
- Especifica las dependencias con otros paquetes o bibliotecas, lo que permite la correcta compilación e instalación del software.

src

- Carpeta que contiene el código fuente principal del paquete, generalmente implementado en C++ o Python.
- Aquí se desarrollan los nodos y demás componentes funcionales que forman parte de la aplicación robótica.

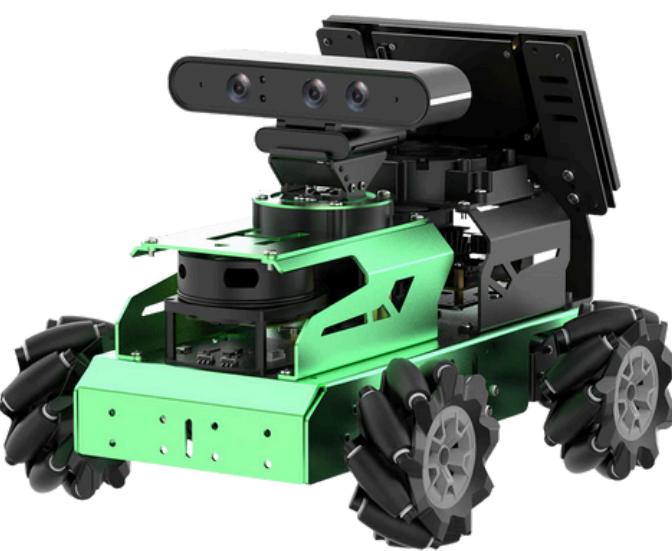
include

- Carpeta que almacena los archivos de cabecera (header files) del paquete, principalmente utilizados en C++.
- Facilita la reutilización de funciones, clases y estructuras, promoviendo una programación modular y organizada.



Creando una Aplicación ROS

- ROS Workspace.
- ROS Package.
- ROS Nodes.

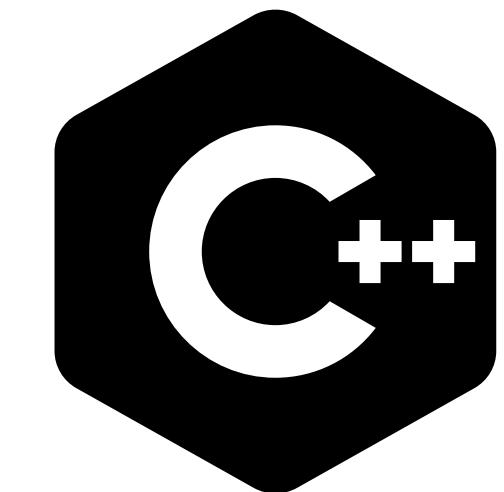




Archivos de cabecera y módulos de ROS

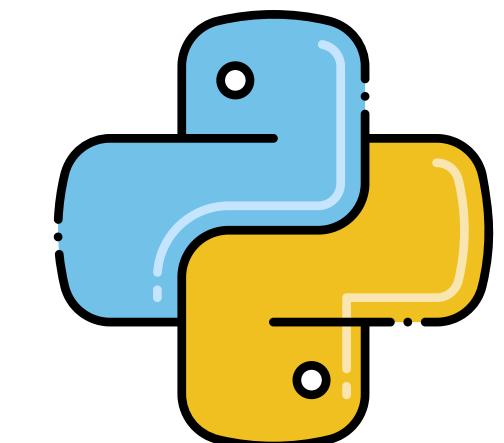
C++ → Inclusión de archivos de cabecera

- Para crear un nodo en ROS con C++, es necesario incluir archivos de cabecera.
- Ejemplo:
#include "ros/ros.h"
- El archivo ros.h contiene todas las definiciones necesarias para implementar las funcionalidades básicas de ROS.



Python → Importación de módulos

- En Python, en lugar de incluir cabeceras, se importan módulos.
- Para crear un nodo en ROS, se requiere importar el módulo principal:
import rospy





Archivos de cabecera y módulos de ROS para mensajes

Cabecera de mensajes ROS

- El paquete std_msgs proporciona definiciones de mensajes con tipos de datos estándar como enteros, flotantes y cadenas de texto.
- Ejemplos en C++:

```
#include "std_msgs/String.h"  
#include "std_msgs/Int32.h"  
#include "std_msgs/Int64.h"
```

- Para definir un mensaje personalizado, se utiliza la siguiente sintaxis:

```
#include <nombre_paquete_mensajes/NombreMensaje.h>
```



Inicialización de un nodo en ROS

Iniciar Nodo

- La inicialización es un paso obligatorio para cualquier nodo ROS.
- En C++, se realiza con el siguiente código:

```
int main(int argc, char **argv)
{
    ros::init(argc, argv, "nombre_del_nodo");
    ...
}
```

- En Python, se inicializa con:

```
rospy.init_node('nombre_del_nodo')
```



Creación de un manejador de nodo (NodeHandle)

Manejador de Nodo

- Tras la inicialización, en C++ es necesario crear una instancia de ros::NodeHandle, que permite iniciar el nodo y realizar operaciones como publicar o suscribirse a tópicos.
- Ejemplo en C++:
ros::NodeHandle nh;
- El resto de las operaciones del nodo se gestionan a través de esta instancia (nh).
- En Python, no es necesario crear un manejador explícito, ya que el módulo rospy se encarga de manejar internamente estas operaciones.

¿PREGUNTAS?

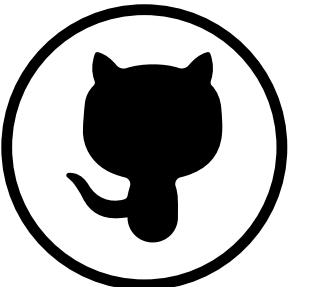


Diego Banda R.
Universidad Católica San Pablo
diego.banda@ucsp.edu.pe





Página de recursos

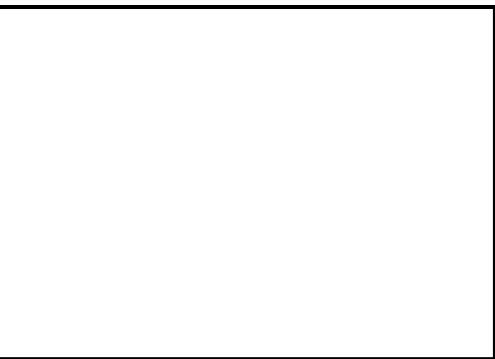


[DiegoABR07/Deep-Learning-Deployment...](#)



Implementation of object detection and classification models on NVIDIA Jetson Nano Developer Kit, using MATLAB R2024a, GPU Coder, cuDNN, TensorRT...

1 Contributor 0 Issues 0 Stars 0 Forks



[matlab-deep-learning/MATLAB-Deep-Learning...](#)



Discover pretrained models for deep learning in MATLAB

4 Contributors 12 Issues 533 Stars 117 Forks

