Guía Inicial: Entendiendo los Paquetes de ROS 2

En el mundo de ROS 2, los **paquetes** son la unidad fundamental de organización del código. Piensa en ellos como **"cajas de herramientas" autocontenidas y bien organizadas** para una funcionalidad específica de tu robot. Cada paquete agrupa todo lo necesario para que un componente de tu sistema robótico funcione de forma independiente y se integre con el resto.

¿Por qué usar Paquetes en ROS 2?

Aunque al principio puedas ejecutar scripts de Python sueltos, los paquetes se vuelven **indispensables** a medida que tus proyectos crecen y se vuelven más complejos. Aquí te explico por qué:

1. **Organización y Modularidad:**
   - Un paquete agrupa tu código fuente (nodos), archivos de configuración, scripts de lanzamiento, modelos 3D, etc., en una estructura lógica y estándar. Esto hace que tu proyecto sea más fácil de entender y mantener.
   - Cada paquete puede ser un módulo independiente (por ejemplo, un paquete para el control del Tello, otro para la detección de objetos, otro para el SLAM).
2. **Reusabilidad y Compartibilidad:**
   - Una vez que tienes un paquete, es muy fácil reutilizarlo en otros proyectos o compartirlo con otros desarrolladores. Es una unidad de software que "simplemente funciona" si se cumplen sus dependencias.
3. **Gestión de Dependencias Automática:**
   - Cada paquete declara explícitamente de qué otras librerías o paquetes de ROS 2 depende. Cuando compilas tu espacio de trabajo, ROS 2 se encarga de asegurar que todas esas dependencias estén disponibles.
4. **Integración con Herramientas de ROS 2:**
   - Herramientas clave de ROS 2 como colcon build (para compilar), ros2 run (para ejecutar nodos) y ros2 launch (para iniciar múltiples nodos y configuraciones complejas a la vez) están diseñadas para trabajar con la estructura de paquetes.
5. **Estándar de la Comunidad:**
   - Es la forma estándar y recomendada de desarrollar en ROS. Seguir este estándar facilita la colaboración y el uso de recursos y paquetes existentes de la enorme comunidad de ROS.

Anatomía de un Paquete ROS 2 Básico (Enfocado en Python)

Cuando creas un paquete ROS 2 para Python, se genera una estructura de carpetas y archivos clave. Vamos a verlos:

```
mi_proyecto_ros/
└── src/
    └── mi_paquete_tello_vision/  <-- Esta es la carpeta raíz de tu paquete
        ├── package.xml         <-- El "carnet de identidad" del paquete
        ├── setup.py            <-- El "instalador" de Python
        ├── resource/
        │   └── mi_paquete_tello_vision
        ├── mi_paquete_tello_vision/  <-- Carpeta con el código fuente Python (módulo)
        │   ├── __init__.py          <-- Archivo esencial para Python
        │   └── nodo_tello_camara.py  <-- Tu código de nodo Python
        │   └── nodo_deteccion_puertas.py
        └── launch/             <-- (Opcional) Scripts para lanzar nodos
            └── mi_lanzamiento.launch.py
        └── config/             <-- (Opcional) Archivos de configuración (YAML)
            └── calibracion_camara.yaml
```

1. package.xml (El Carnet de Identidad)

Este archivo XML es crucial. Contiene metadatos sobre tu paquete y, lo más importante, sus **dependencias**.

- **Metadata:** Nombre del paquete, versión, descripción, autor, licencia.
- **Dependencias (<depend>):** Aquí declaras de qué otros paquetes o librerías de ROS 2 depende tu código. Cuando ejecutas colcon build, ROS 2 lee este archivo para asegurarse de que tienes todo lo necesario instalado.
    - **Ejemplo:**

```
<package format="3">
  <name>mi_paquete_tello_vision</name>
  <version>0.0.0</version>
  <description>Paquete para la visión y control del Tello</description>
  <maintainer email="tu_email@ejemplo.com">Tu Nombre</maintainer>
  <license>Apache-2.0</license>

  <depend>rclpy</depend>          <!-- Para escribir nodos ROS 2 en Python -->
  <depend>sensor_msgs</depend>    <!-- Para mensajes de imagen -->
  <depend>cv_bridge</depend>      <!-- Para convertir entre imágenes ROS y
OpenCV -->
```

```
  <depend>std_msgs</depend>       <!-- Para mensajes de datos simples -->
  <depend>djitellopy</depend>     <!-- La librería para el Tello -->
  <depend>opencv-python</depend>  <!-- OpenCV para procesamiento de
imagen -->
  <depend>numpy</depend>          <!-- Para arrays numéricos con OpenCV -->

  <test_depend>ament_copyright</test_depend>
  <test_depend>ament_flake8</test_depend>
  <test_depend>ament_pep257</test_depend>
  <test_depend>python3-pytest</test_depend>

  <export>
    <build_type>ament_python</build_type>
  </export>
</package>
```

2. setup.py (El Instalador de Python)

Este script le dice a Python y a ROS 2 cómo encontrar y ejecutar tus nodos Python. Es donde "registras" tus scripts Python para que ros2 run los reconozca.

- **entry_points:** Esta sección es la más importante para los ejecutables Python. Define un "punto de entrada" para tus nodos.
  - **Ejemplo:**

```
from setuptools import find_packages, setup

package_name = 'mi_paquete_tello_vision'

setup(
    name=package_name,
    version='0.0.0',
    packages=find_packages(exclude=['test']),
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
        # Aquí puedes añadir archivos de lanzamiento o configuración
        (os.path.join('share', package_name, 'launch'),
glob('launch/*.launch.py')),
        (os.path.join('share', package_name, 'config'), glob('config/*.yaml')),
```

```
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='Tu Nombre',
    maintainer_email='tu_email@ejemplo.com',
    description='Paquete para la visión y control del Tello',
    license='Apache-2.0',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'nodo_tello_camara = mi_paquete_tello_vision.nodo_tello_camara:main',
            'nodo_deteccion_puertas =
mi_paquete_tello_vision.nodo_deteccion_puertas:main',
            # Puedes añadir más nodos aquí
        ],
    },
)
```

- 'nodo_tello_camara = mi_paquete_tello_vision.nodo_tello_camara:main' significa:
  - nodo_tello_camara: Es el nombre que usarás en la terminal con ros2 run mi_paquete_tello_vision nodo_tello_camara.
  - mi_paquete_tello_vision.nodo_tello_camara: Indica la ruta del módulo Python (la subcarpeta mi_paquete_tello_vision, y dentro de ella el archivo nodo_tello_camara.py).
  - main: Es el nombre de la función (generalmente main()) dentro de nodo_tello_camara.py que inicia tu nodo ROS 2.

3. mi_paquete_tello_vision/ (la subcarpeta con el código Python)

Esta carpeta, con el mismo nombre que tu paquete, es donde resides tu código fuente Python. Se convierte en un módulo de Python cuando se instala el paquete.

- **__init__.py:** Este archivo, aunque a menudo está vacío, es crucial. Le dice a Python que la carpeta en la que se encuentra es un "paquete" de Python. Sin él, Python no podría importar tus módulos desde otras partes de tu proyecto.
- **nodo_tello_camara.py y nodo_deteccion_puertas.py:** Aquí es donde escribes el código real de tus nodos ROS 2, definiendo la clase de tu nodo, sus publicadores, suscriptores, timers, y la función main() que lo inicializa.

4. launch/ (Scripts de Lanzamiento)

Esta carpeta es opcional, pero muy útil para proyectos complejos. Contiene archivos de lanzamiento (generalmente *.launch.py en ROS 2) que te permiten iniciar múltiples nodos con un solo comando, pasarles parámetros y remapear topics.

- **Ejemplo de uso:** ros2 launch mi_paquete_tello_vision mi_lanzamiento.launch.py

5. config/ (Archivos de Configuración)

También opcional. Aquí puedes guardar archivos de configuración (comúnmente .yaml) con parámetros para tus nodos, como la calibración de la cámara, umbrales de detección, etc. Estos parámetros se pueden cargar en tus nodos al iniciar.

El Workflow Básico con Paquetes ROS 2

Aquí tienes los pasos generales para trabajar con paquetes en tu proyecto:

1. Crear un "Workspace" (Espacio de Trabajo):
   Un workspace es una carpeta donde guardas uno o más paquetes ROS 2. Es tu área de desarrollo.
   mkdir -p ~/mi_proyecto_ros/src # Crea la carpeta principal y la subcarpeta 'src'
   cd ~/mi_proyecto_ros/src

2. Crear un Paquete ROS 2:
   Usa el comando ros2 pkg create.
   ros2 pkg create --build-type ament_python mi_paquete_tello_vision --dependencies rclpy sensor_msgs cv_bridge std_msgs djitellopy opencv-python numpy

   - --build-type ament_python: Especifica que es un paquete Python.
   - --dependencies ...: Lista las dependencias directas de tu código Python.
3. **Añadir y Modificar tu Código:**
   - Mueve tus scripts Python (ej. nodoTello.py, camara.py, nodoImagen.py, nodoLogica.py) a la subcarpeta de código de tu paquete: ~/mi_proyecto_ros/src/mi_paquete_tello_vision/mi_paquete_tello_vision/.
   - Edita el archivo setup.py dentro de la carpeta raíz de tu paquete (~/mi_proyecto_ros/src/mi_paquete_tello_vision/setup.py) para registrar tus nodos en la sección entry_points.
   - Asegúrate de que tus nodos Python tengan la estructura básica de un nodo ROS 2 (clase Node, __init__, main).
4. Construir (Compilar) el Workspace:
   Este paso es donde colcon procesa todos los paquetes en tu workspace, resuelve dependencias y prepara los ejecutables.

cd ~/mi_proyecto_ros # ¡Importante: ve a la raíz de tu workspace, NO a la carpeta 'src'!
colcon build # Construye todos los paquetes que encuentre

- Si solo quieres construir paquetes específicos (útil si tienes muchos y solo modificas algunos):
  colcon build --packages-select mi_paquete_tello_vision ORB_SLAM3_ROS2

- Si la compilación es exitosa, se crearán carpetas como install, build, log en la raíz de tu *workspace*.

5. "Sourcear" el Entorno (Cargar Variables de Entorno):
   Después de construir, necesitas decirle a tu terminal dónde encontrar tus nuevos paquetes y ejecutables. Debes hacer esto en CADA TERMINAL NUEVA que abras para trabajar con tu workspace.
   source install/setup.bash

   También puedes añadir esto a tu ~/.bashrc para que se cargue automáticamente (pero ten cuidado si trabajas con múltiples versiones de ROS o *workspaces*).
6. Ejecutar tus Nodos:
   Una vez que el entorno está "sourceado", puedes ejecutar tus nodos:
   ros2 run mi_paquete_tello_vision nodo_tello_camara
   ros2 run mi_paquete_tello_vision nodo_deteccion_puertas

7. Usar Archivos de Lanzamiento (.launch.py):
   Para iniciar varios nodos a la vez, o nodos con configuraciones complejas, usas archivos de lanzamiento.
   ros2 launch mi_paquete_tello_vision mi_lanzamiento.launch.py

Una Nota sobre Paquetes C++ (como el Wrapper de ORB-SLAM3)

Aunque no sepas C++, es útil saber que los paquetes C++ tienen una estructura similar, pero usan:

- CMakeLists.txt en lugar de setup.py para definir cómo se compila el código C++ y sus dependencias.
- ament_cmake como tipo de construcción en package.xml.
- El código fuente está en archivos .cpp y .hpp.

Cuando compiles el *wrapper* de ORB-SLAM3, colcon build usará su CMakeLists.txt para compilarlo. Tú solo necesitas asegurarte de que el CMakeLists.txt del *wrapper*

sepa dónde está tu instalación de ORB-SLAM3 (como hablamos antes).