# **📘 Documento Comparativo – Librerías lib\_irobot vs lib\_robot\_roomba**

## **🟦 1. Descripción General de Cada Librería**

### **lib\_irobot**

Librería **sencilla y directa**, pensada para:

* Facilitar la conexión con el robot mediante Create2.
* Proveer funciones independientes para:  
  + **Lectura de sensores:** batería, movimiento, proximidad.
  + **Control de movimiento:** drive, spin, manual.
  + **Control de limpieza y LEDs.**

✅ Muy útil para proyectos simples.  
 ✅ Alta compatibilidad con communication\_node\_v4.

### **lib\_robot\_roomba**

Librería **avanzada y modular**, diseñada para:

* Encapsular un **estado interno de todos los sensores** en un objeto Robot.
* Permitir:  
  + Activar/desactivar sensores dinámicamente (enabled).
  + Leer **solo los sensores habilitados** mediante filtros.
  + Actualizar automáticamente los datos de sensores con update\_sensors().
  + Conectar por **Serial o TCP** (vía socat).
* Proveer un modelo de robot extensible y preparado para futuras ampliaciones.

✅ Mucho más flexible.  
 ✅ Ideal para proyectos con filtrado dinámico, conexión remota o múltiples subsistemas.

## **🟩 2. Tabla Comparativa de Funcionalidades**

| **Funcionalidad** | **lib\_irobot** | **lib\_robot\_roomba** |
| --- | --- | --- |
| **Conexión Serial** | Sí (Create2) | Sí |
| **Conexión TCP (socat)** | ❌ | ✅ |
| **Lectura de sensores** | Funciones sueltas | Acceso unificado vía Robot.read\_sensors() |
| **Filtros dinámicos de sensores** | ❌ | ✅ Activar/desactivar por enabled |
| **Actualizar solo sensores activos** | ❌ | ✅ update\_sensors() |
| **Control de movimiento y limpieza** | Sí | Parcial (se puede integrar) |
| **Control de LEDs y música** | Sí | Parcial (requiere implementar métodos) |
| **Modelo de robot persistente** | ❌ | ✅ (estado interno de sensores) |
| **Compatibilidad inmediata con v4** | ✅ | ⚠️ Requiere ajustes |

## **🟨 3. ¿Qué ventajas aporta lib\_robot\_roomba?**

**Ventajas principales al migrar:**

1. **Filtrado dinámico de sensores:**
   * Solo publicas datos de sensores habilitados.
   * Ahorra recursos en temas con alta frecuencia.
2. **Modelo de datos estructurado:**
   * Todos los sensores agrupados en categorías:  
     + movement
     + security
     + power
     + misc
   * Mucho más fácil mapearlos a tus mensajes ROS.
3. **Conexión TCP y Serial:**
   * Puedes operar el robot remotamente vía red.
4. **Escalabilidad:**
   * Preparada para expandir sensores, botones y estados.
5. **Consistencia en la actualización:**
   * update\_sensors() sincroniza todo de una vez.

## **🟪 4. Ejemplo de uso de lib\_robot\_roomba en un nodo**

Para que te quede claro, aquí tienes un **esquema de cómo se usaría**:

from lib\_robot\_roomba import Robot

self.robot = Robot()

# Si usas TCP

self.robot.connect\_robot(connection\_type='TCP', address='192.168.1.10', port=9000)

# Para activar solo algunos sensores

self.robot.\_sensors['movement']['distance']['enabled'] = True

self.robot.\_sensors['movement']['angle']['enabled'] = False

# Leer sensores habilitados

data = self.robot.read\_sensors()

distance = data['movement']['distance']

# Actualizar datos

self.robot.update\_sensors(self.robot)

## **🟥 5. ¿Cómo implantar lib\_robot\_roomba en Communication\_Node\_v0.0.5?**

### **5.1 Cambios necesarios en communication\_node\_v0.0.5**

✅ **1. Sustituir connect\_robot()**

* En lugar de lib\_irobot.connect\_robot(), deberás crear un objeto Robot y llamar connect\_robot() de la clase.
* Ejemplo:

self.robot = Robot()

self.robot.connect\_robot(

connection\_type='Serial',

serial\_port='/dev/roomba',

connection\_mode='FULL'

)

✅ **2. Sustituir las funciones de lectura de sensores**

* Cambiar de:

data = leer\_estado\_movimiento(self.robot)

* A:

self.robot.update\_sensors(self.robot)

data = self.robot.read\_sensors()

* **Nota importante:** La estructura de data es diferente, ahora tendrás diccionarios anidados por categoría.

✅ **3. Adaptar publicación de topics**

* Ejemplo: Publicar movimiento

msg = Movement()

msg.distance = data['movement']['distance']

msg.angle = data['movement']['angle']

msg.left\_encoder\_counts = data['movement']['left\_encoder\_counts']

msg.right\_encoder\_counts = data['movement']['right\_encoder\_counts']

self.movement\_pub.publish(msg)

✅ **4. Implementar nuevos métodos si los necesitas**

* Si quieres seguir usando control de LEDs, música o motores, tendrás que crear métodos en Robot o importarlos de lib\_irobot.

## **🟫 6. Plan recomendado de migración**

Si decides pasarte a lib\_robot\_roomba, mi propuesta es:

1️⃣ Crear un branch experimental:  
 **feature/migracion\_lib\_robot\_roomba**

2️⃣ Adaptar el initialize\_connection():

* Configurar Serial y TCP como parámetros.

3️⃣ Modificar todos los métodos publish\_\* para usar read\_sensors().

4️⃣ Probar topic por topic con ros2 topic echo.

5️⃣ Validar que la cola de prioridad sigue funcionando igual.

6️⃣ (Opcional) Añadir métodos faltantes (LEDs, música).

## **✅ 7. Conclusión**

**¿Por qué te interesa migrar a lib\_robot\_roomba?**

✔ Si necesitas **filtros dinámicos**.  
 ✔ Si quieres conexión por TCP.  
 ✔ Si planeas tener un robot con sensores opcionales o configuración en tiempo real.

**¿Por qué seguir con lib\_irobot?**

✔ Si prefieres simplicidad y compatibilidad inmediata.  
 ✔ Si no necesitas TCP ni filtrado dinámico.