**Seguimiento 3D Teleoperado de un Objeto Móvil Basado en Fusión Sensorial usando ROS 2**

Estudiante: Ocampo Yuca Josue

Docente: Elmer Alan Cornejo



Universidad Católica Boliviana San Pablo

Departamento de Ciencias de la Tecnología e Innovación

Carrera de Ingeniería Mecatrónica

Tarija

IMT 342 - Robótica

29 de mayo de 2025

TABLA DE CONTENIDO

[I. Introducción 1](#_Toc178344523)

[A. Objetivos 1](#_Toc178344524)

[1) Objetivo General: 1](#_Toc178344525)

[2) Objetivos Específicos del Proyecto: 1](#_Toc178344526)

[B. Caso de Estudio 1](#_Toc178344527)

[C. Requerimientos 1](#_Toc178344528)

[II. Descripción Técnica 3](#_Toc178344529)

[A. Variable Física: 3](#_Toc178344530)

[B. Instrumentos de Medición: 3](#_Toc178344531)

[C. Acondicionamiento de Señal: 3](#_Toc178344532)

[III. Ingeniería del proyecto 4](#_Toc178344533)

[A. Selección 4](#_Toc178344534)

[B. Esquematización 4](#_Toc178344535)

[C. Dimensionamiento 4](#_Toc178344536)

[D. Simulación 4](#_Toc178344537)

[E. Montaje 4](#_Toc178344538)

[IV. Conclusiones 5](#_Toc178344539)

[A. Conclusiones 5](#_Toc178344540)

[B. Recomendaciones 5](#_Toc178344541)

[V. Referencias 6](#_Toc178344542)

LISTA DE FIGURAS

**No se encuentran elementos de tabla de ilustraciones.**

# Introducción

## Objetivos

### Objetivo General:

Desarrollar un Vehículo Guiado Automáticamente (AGV) capaz de ser teleoperado, integrando un sistema robótico de código abierto (ROS 2) con un microcontrolador (ESP8266) para la adquisición de comandos físicos y su posterior visualización en un entorno simulado en tiempo real.

### Objetivos Específicos del Proyecto:

Establecer un entorno de desarrollo robusto para ROS 2 Humble Hawksbill sobre Ubuntu 22.04 LTS en una máquina virtual.

Implementar una comunicación bidireccional confiable entre el sistema ROS 2 y la placa ESP8266 utilizando el protocolo MQTT.

Diseñar y modelar la estructura cinemática y visual del AGV mediante archivos URDF/XACRO. Visualizar el modelo 3D del AGV en el entorno de RViz2, mostrando su configuración y, idealmente, su estado de movimiento.

Integrar el sistema de control de movimiento (ros2\_control) con el modelo del AGV para permitir la teleoperación y reflejar los comandos de los pulsadores físicos en el simulador.

Demostrar la interacción entre los comandos físicos (pulsadores en ESP8266) y la respuesta del AGV simulado.

## Caso de Estudio

El presente proyecto se centra en la creación de un prototipo funcional de un AGV teleoperado. Este caso de estudio aborda la necesidad creciente en la automatización industrial y logística de soluciones flexibles y de bajo costo para el transporte de materiales. La teleoperación, en este contexto, permite una intervención humana directa para tareas de precisión o para entornos donde la autonomía total no es factible o deseada. La integración de un microcontrolador como el ESP8266 facilita una interfaz física accesible y distribuida, mientras que ROS 2 proporciona la infraestructura robótica necesaria para la planificación, control y visualización del vehículo en un entorno dinámico y escalable [1].

.

## Requerimientos

Requerimientos de Software:

\* Sistema Operativo: Ubuntu 22.04 LTS.

\* Entorno de Robótica: ROS 2 Humble Hawksbill.

\* Plataforma de Desarrollo Integrado (IDE): Código VS.

\* Corredor MQTT: Mosquitto.

\*Herramientas de Visualización: RViz2.

\* Herramientas de Modelado Robótico: XACRO.

\* Marco de Control Robótico: ros2\_control

Requerimientos de Hardware:

\* Microcontrolador: ESP8266 (o similar, como NodeMCU).

\* Componentes de Interfaz Física: Pulsadores para teleoperación.

\* Actuador Demostrativo: LED integrado en la ESP8266 (o externo).

\* Equipo de Cómputo: PC con capacidad para virtualización (VirtualBox).

Requerimientos de Comunicación:

\* Conectividad de Red: Wi-Fi para la ESP8266 y comunicación de red entre la máquina host y la máquina virtual. \* Protocolo de Mensajería: MQTT para la comunicación entre ROS 2 y ESP8266.

\* Protocolo de Comunicación ROS 2: DDS (Servicio de Distribución de Datos).

Requerimientos Funcionales:

\* Transmisión de comandos desde pulsadores ESP8266 a ROS 2.

\* Control de actuadores simples (LED) desde ROS 2 vía MQTT a ESP8266.

\* Modelado 3D del AGV en ROS 2 para visualización.

\* Visualización en tiempo real del AGV en RViz2.

\* Habilitación de control de movimiento del AGV simulado mediante ros2\_control.

# Descripción Técnica

## Variable Física:

La principal variable física controlada y observada en este proyecto, más allá de la teleoperación de un LED, es el movimiento del AGV. Esta se descompone en:

\* Velocidad Lineal: El desplazamiento del AGV hacia adelante o hacia atrás, expresado en metros por segundo (m/s).

\* Velocidad Angular: La capacidad del AGV para girar sobre su propio eje o curvar, expresada en radianes por segundo (rad/s).

\* Estados de Pulsadores: La condición de pulsado o liberado de los botones físicos (digital: ON/OFF) que representan los comandos de teleoperación (adelante, atrás, izquierda, derecha). La relevancia en un proceso industrial radica en la capacidad de un operador humano para guiar de forma intuitiva el AGV, ideal para tareas de posicionamiento fino, navegación en entornos cambiantes o ante imprevistos, donde la automatización completa aún no es óptima [2].

## Instrumentos de Medición:

**Pulsadores Físicos:** Utilizados en la placa ESP8266 para generar comandos de teleoperación manual. Estos "instrumentos" convierten la acción mecánica del operador en una señal digital que el microcontrolador interpreta.

**ESP8266:** Actúa como el puente inicial de "medición", leyendo el estado de los pulsadores y encapsulando esta información en mensajes MQTT.

**Broker Mosquitto MQTT:** No es un instrumento de medición directo, pero es esencial como infraestructura de comunicación, permitiendo el enrutamiento de los "datos medidos" (estados de pulsadores) desde el ESP8266 hacia los nodos de ROS 2.

**Temas ROS 2:** Aunque no son instrumentos físicos, los tópicos de ROS 2 ( /cmd\_velpara comandos de velocidad, y tópicos personalizados para el estado de los pulsadores) actúan como canales donde se "miden" y distribuyen las intenciones de control y los datos de estado. [3]

**robot\_state\_publisher:** Convierte la descripción del modelo (URDF/XACRO) en transformaciones espaciales (TF) que pueden ser "medidas" y visualizadas por RViz2.

**RViz2:** Es el instrumento de "medición" y visualización por excelencia en el entorno ROS. Permite observar el modelo 3D del robot y, idealmente, su movimiento y el efecto de las transformaciones publicadas. [3].

## Acondicionamiento de Señal:

**Desde Pulsadores a ESP8266:** La lectura de los pulsadores se realiza directamente a través de pines GPIO de la ESP8266. Para evitar el "rebote" (múltiples lecturas falsas por una única pulsación), se implementaría un acondicionamiento digital a nivel de software (debouncing) en el código de la ESP8266.

**Desde ESP8266 a MQTT:** Los estados lógicos de los pulsadores (HIGH/LOW) se convierten en mensajes de texto o booleanos ( "ON", "OFF"o 1/ 0) y se publican en temas específicos de MQTT. Esta es una forma de "acondicionamiento" para adaptarse al protocolo de mensajería.

**Desde MQTT a Nodos ROS 2:** El nodo puente de ROS 2 se suscribe a los temas MQTT, recibe los mensajes de texto/booleanos, y los "acondiciona" (parsea) para convertirlos en tipos de mensajes ROS 2 apropiados (ej., std\_msgs/Boolo mensajes personalizados) para su uso interno.

**Generación de Comandos de Velocidad:** Los estados de los pulsadores condiciones se traducen en comandos de velocidad ( geometry\_msgs/Twist) que el diff\_drive\_controller esperaría. Esto implica una lógica de control en el nodo puente de ROS 2 para mapear las pulsaciones a velocidades lineales y angulares.

# Ingeniería del proyecto

## Selección

La selección de tecnologías se basó en una evaluación de sus capacidades y relevancia para la robótica moderna y el IoT. Se optó por **ROS 2 (Humble Hawksbill)** por su arquitectura distribuida, su madurez y su comunidad activa, siendo el estándar de facto en robótica. La **ESP8266** [4] fue elegido por su relación costo-beneficio, capacidad Wi-Fi y facilidad de programación para sistemas embebidos. **MQTT con Mosquitto** [5] se seleccionó por ser un protocolo ligero y eficiente para la comunicación IoT, ideal para la integración entre el microcontrolador y la PC. El **modelado URDF/XACRO** se elige por ser el formato estándar de ROS para la descripción de robots, y **RViz2** por su versatilidad en la visualización. El framework **ros2\_control** se seleccionó por su capacidad para gestionar el control de bajo nivel del hardware del robot, tanto en simulación como en un futuro robot físico.

## Esquematización

El sistema se esquematiza como una arquitectura de capas y subsistemas interconectados: \* **Capa Inferior (Hardware Físico):** ESP8266 con pulsadores y LED.

\* **Capa de Comunicación IoT:** Protocolo MQTT y Mosquitto Broker, actuando como intermediario.

\* **Capa de Control Robótico (ROS 2):**

\* **Nodos Puente MQTT-ROS:** Nodos Python encargados de comandos públicos ROS basados ​​en mensajes MQTT recibidos de la ESP8266, y publicar mensajes MQTT basados ​​en comandos ROS.

\* **Nodos de Sistema ROS 2:** robot\_state\_publisher para el modelo, ros2\_control\_nodecomo gestor de controladores, joint\_state\_broadcasterpara estados de articulaciones y diff\_drive\_controllerpara el movimiento.

\* **Entorno de Simulación/Visualización:** RViz2 para la representación gráfica del AGV. \* La interacción se esquematiza como un flujo de datos:

Pulsador (ESP8266) -> MQTT -> Nodo Puente ROS 2 -> Tópico ROS 2 (/cmd\_vel) -> diff\_drive\_controller -> ros2\_control\_node -> Modelo en RViz2.

## Dimensionamiento

## \* **Dimensiones del AGV Virtual:** El modelo del AGV se dimensionó en el model.urdf.xacrocon medidas realistas para un robot móvil básico... base de 0.70x0.39x0.20 metros, ruedas de 0.06m de radio.

## \* **Masas e Inercias:** Se asignan masas e inercias aproximadas a cada enlace del robot (por ejemplo, base de 10 kg, ruedas de 0,5 kg) para una simulación física más realista (aunque esta no se activó en Gazebo en esta fase).

## \* **Offsets y Distancias:** Las posiciones de las ruedas y sensores (IMU, caster) se definieron con offsets precisos respecto al base\_linken el URDF.

## \* **Parametrización de Controladores:** El diff\_drive\_controllerrequiere parámetros como wheel\_separation, wheel\_radius, y nombres de las articulaciones ( left\_wheel\_names, right\_wheel\_names), que debían ser dimensionados en el archivo YAML.

## Simulación

La simulación se realizó principalmente a través de la **visualización en RViz2** . Se creó un modelo 3D detallado del AGV utilizando XACRO, lo que permitió su representación precisa en el entorno virtual. El robot\_state\_publisher se encargó de animar la pose del robot en función de las transformaciones. La intención era que el diff\_drive\_controllerutilizara los comandos de velocidad para actualizar la odometría y las poses de las ruedas, reflejando el movimiento del AGV en tiempo real en RViz. La ausencia del simulador Gazebo ( use\_simulator:="False") se decidió para enfocar los esfuerzos en la visualización en RViz y la integración del ros2\_control.

## Montaje

## El montaje del proyecto se dividió en dos esferas interconectadas:

## \* **Montaje Lógico/Software:** Implicó la configuración de todos los componentes de software: \* Instalación y configuración de ROS 2 Humble.

## \* Creación y compilación de paquetes ROS 2 ( basic\_mobile\_robot).

## \* Edición de archivos launchpara orquestar el inicio de nodos.

## \* Configuración del Mosquitto Broker.

## \* Creación de scripts Python para los nodos puente MQTT-ROS.

## \* Desarrollo del archivo model.urdf.xacroy el archivo de parámetros basic\_mobile\_bot\_controllers.yaml.

## \* Verificación del CMakeLists.txtpara asegurar la instalación de los archivos de configuración.

## \* **Montaje Físico (ESP8266):** Consistió en la conexión de los pulsadores y un LED a los pines GPIO de la placa ESP8266, y la carga del firmware que maneja la comunicación MQTT y la lectura de los estados de los pulsadores. Se aseguró que el ESP8266 estaba conectado a la misma red Wi-Fi que la máquina host para facilitar la comunicación con el corredor.

# Conclusiones

## Conclusiones

## El proyecto de AGV teleoperado ha sido una experiencia de aprendizaje intensivo y desafiante, logrando avances significativos en la integración de múltiples tecnologías robóticas y de IoT. Se ha demostrado con éxito la comunicación bidireccional entre ROS 2 y un microcontrolador ESP8266 utilizando MQTT, un hito crucial para la teleoperación. El modelado del robot en URDF/XACRO y su visualización estática en RViz2, junto con la correcta publicación de transformadas (TF), son logros sólidos. Sin embargo, la integración completa del control de movimiento simulado se vio obstaculizada por dificultades imprevistas en la carga de los parámetros de los controladores de ros2\_control, específicamente el diff\_drive\_controller, debido a la falta de reconocimiento del archivo YAML de configuración. Esta situación subraya la complejidad intrínseca de los sistemas ROS 2 y la necesidad de una comprensión profunda de sus mecanismos de empaquetamiento y gestión de recursos para una implementación exitosa.

## Recomendaciones

## Para futuras iteraciones o proyectos de similar complejidad, se recomienda encarecidamente: \* **Fortalecer la formación en ros2\_control:** Incluir módulos dedicados a la depuración avanzada de archivos YAML de configuración y la integración de controladores con modelos URDF. \* **Profundizar en la gestión de paquetes ROS 2:** Enfatizar cómo CMakeLists.txt(o setup.py) afecta la instalación de archivos de configuración y otros recursos en la carpeta install. \* **Progresión Pedagógica Gradual:** Introducir proyectos de esta magnitud con una secuencia de aprendizajes más gradual, incluyendo módulos dedicados a cada tecnología (ROS 2, MQTT, microcontroladores) antes de requerir una integración compleja. \* **Entorno de Debugging Distribuido:** Proporcionar herramientas y metodologías específicas para depurar sistemas que involucran hardware embebido, brokers de red y software distribuido. \* **Documentación y ejemplos completos:** Acceso a ejemplos más detallados y documentación clara sobre la configuración de los controladores de ros2\_controlpara diferentes tipos de robots.

# Referencias

1. Bibliografía

|  |  |
| --- | --- |
| [1] | M. y. K. A. Maciejewski, ROS 2 en 7 Días . The ROS 2 Cookbook, Mastering, 2020. |
| [2] | A. Koubaa, Sistema operativo de robot (ROS): la referencia completa (volumen 2), 2018. |
| [4] | Documentación ROS 2, «Tutoriales, conceptos y API de ROS 2 .,» ROS 2, [En línea]. Available: https://docs.ros.org/en/humble/. |
| [5] | D. Monslave, ESP8266: Proyectos de Desarrollo, México: HAWSILL, 2019. |
| [6] | P. Hiersche, MQTT para IoT: El Internet de las cosas con MQTT, CreateSpace, 2015. |

1. Webgrafía

|  |  |
| --- | --- |
| [3] | R. Comunnity, «Documentación de control de ROS 2,» [En línea]. Available: <https://control.ros.org/master/doc/ros2_control/controller_manager/doc/userdoc.html> |

Foros y Comunidades:

Respuestas de la comunidad ROS. *ROS responde al foro de preguntas y respuestas* Disponible en: <https://answers.ros.org/>

ANEXOS