UNIVERSIDAD DE GUADALAJARA

MANUAL DE

CONTROL DE TurtleBot

LABORATORIO DE CONTROL

Emmanuel Cruz Zavala
Carlos Iván Aldana Lopéz
Emmanuel Nuño Ortega
Alberto Carlos Aguirre González

Manual de Usuario

Manual de uso de los TurtleBot Waffle Pi 3
con Integración OptiTrack

Laboratorio de Control
Universidad de Guadalajara

29 de abril de 2026

Índice

1 Introducción

El presente manual describe el procedimiento completo para la instalación, configuración y puesta en marcha del robot TurtleBot Waffle Pi 3, integrado con el sistema de captura de movimiento OptiTrack.

El objetivo principal es utilizar la pose global proporcionada por el sistema OptiTrack para implementar control en lazo cerrado del robot móvil. El controlador corre directamente en C++ sobre la Raspberry Pi, comunicándose con los motores Dynamixel via OpenCR **sin depender de ROS** para el ciclo de control.

1.1 Cómo usar este manual

Este manual está organizado en dos partes:

- **Cuerpo principal (Secciones 1–9):** contiene todo lo necesario para instalar, configurar y operar el sistema paso a paso. Se recomienda seguirlo en orden la primera vez.
- **Apéndices (A–D):** documentación técnica de referencia sobre el protocolo de comunicación, las estructuras de datos C++, la ley de control y la estructura del proyecto. Se consultan cuando se necesita entender o modificar el sistema en profundidad.

2 Arquitectura General del Sistema

El sistema está compuesto por los siguientes subsistemas:

- TurtleBot3 Waffle Pi (base móvil + motores Dynamixel XM430)
- Raspberry Pi 3/4 (ejecuta el controlador C++)
- Controlador OpenCR 1.0 (interfaz entre Raspberry Pi y motores)
- Sistema OptiTrack (cámaras IR + PC con Motive)
- Servidor personalizado de comunicación (PC con Motive)

El flujo de información es el siguiente, un diagrama se puede visualizar en la Figura 1:

OptiTrack → Servidor Personalizado → `ControlClient` (C++) → Controlador → OpenCR (USB) → Dynamixel XM430

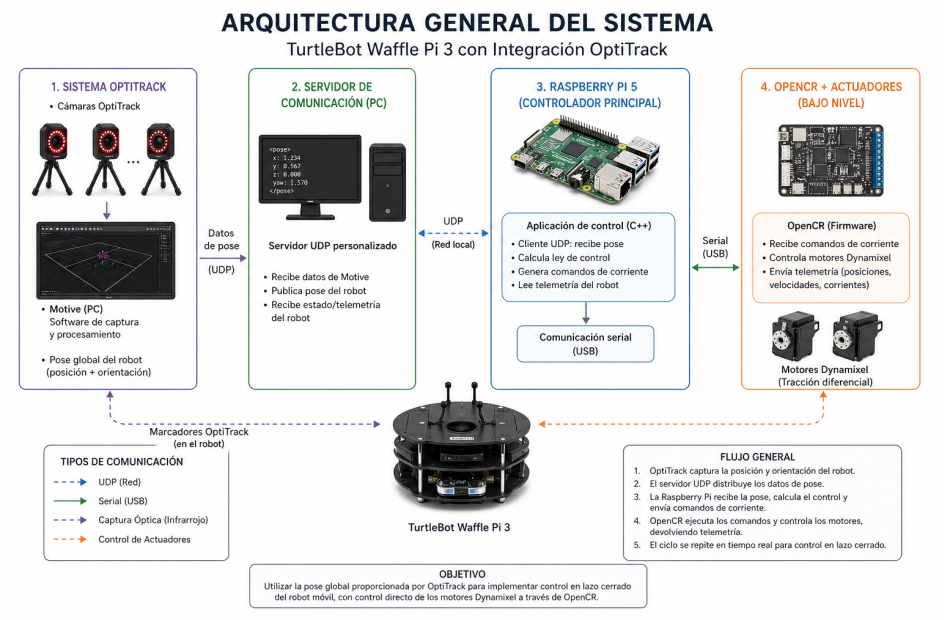

Figura 1: Flujo de información del sistema de captura de movimiento

3 Requisitos

3.1 Hardware

- Raspberry Pi 3 o 4 (recomendado 4 GB o superior)
- MicroSD mínimo 32 GB (clase 10)
- Fuente de alimentación 5 V, 3 A
- Base móvil TurtleBot Waffle Pi 3
- Controlador OpenCR 1.0
- Motores Dynamixel XM430-W210-T
- Batería LiPo 11.1 V
- Sistema de cámaras OptiTrack con Hubs USB
- Software Motive
- Marcadores reflectivos
- PC con Motive para OptiTrack
- Red WiFi común para todos los dispositivos

3.2 Software

- Raspberry Pi OS (64-bit)
- Boost $\geq$ 1.71 (`libboost-all-dev`)
- Compilador C++17 o superior (`g++` o `clang++`)
- CMake $\geq$ 3.10
- Dynamixel SDK (para comunicación directa con los motores XM430)
- Software Motive (en la PC con OptiTrack)

4 Instalación del Sistema Operativo en la Raspberry Pi

4.1 Grabado de la imagen con Raspberry Pi Imager

1. Descargar Raspberry Pi Imager desde el sitio oficial.
2. Insertar la microSD en la computadora.
3. Abrir Raspberry Pi Imager.

Seleccionar:

- **Device:** Raspberry Pi 3 o 4 (según el modelo) (Figura 2)
- **Operating System:** Raspberry Pi OS (64-bit) (Figura 3)
- **Storage:** microSD (Figura 4)

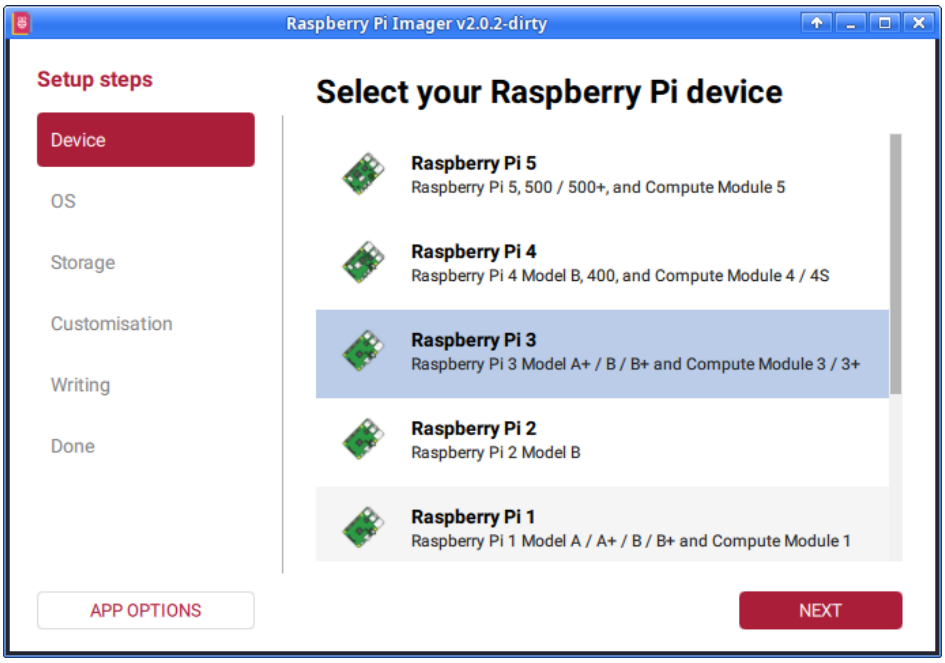

Figura 2: Seleccionar Dispositivo en Raspberry Pi Imager

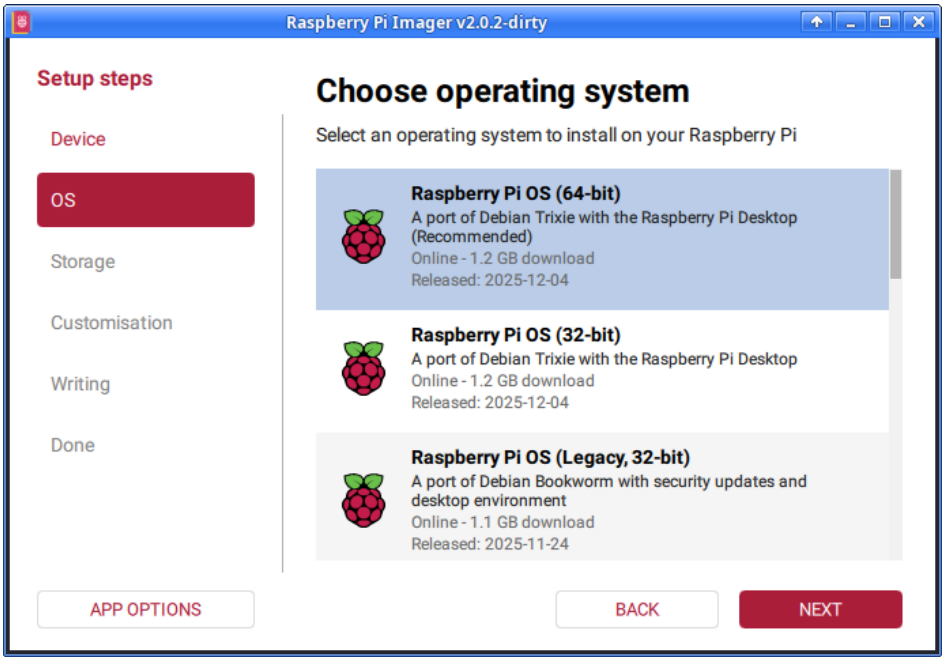

Figura 3: Seleccionar Sistema Operativo

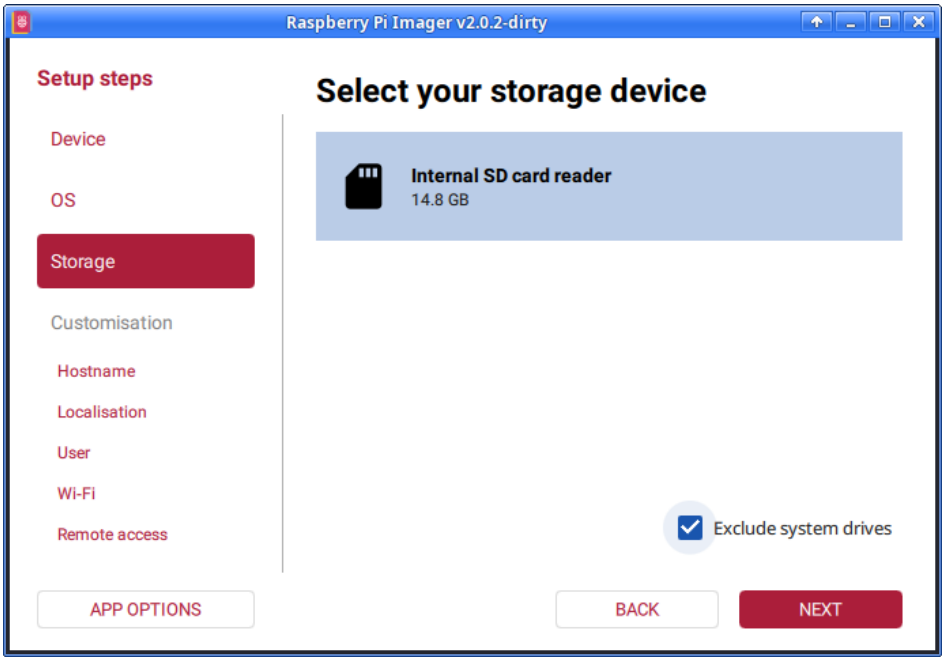

Figura 4: Seleccionar almacenamiento

4.2 Configuración avanzada (antes de grabar)

Antes de escribir la imagen, abrir la configuración avanzada y configurar:

- **Hostname:** p.ej. `turtlebot01` (Figura 5)
- **SSH:** habilitado (Figura 9)

- **Usuario y contraseña** (Figura 7)
- **Red WiFi:** SSID y contraseña de la red del laboratorio (Figura 8)
- **Localización:** zona horaria y distribución de teclado (Figura 6)

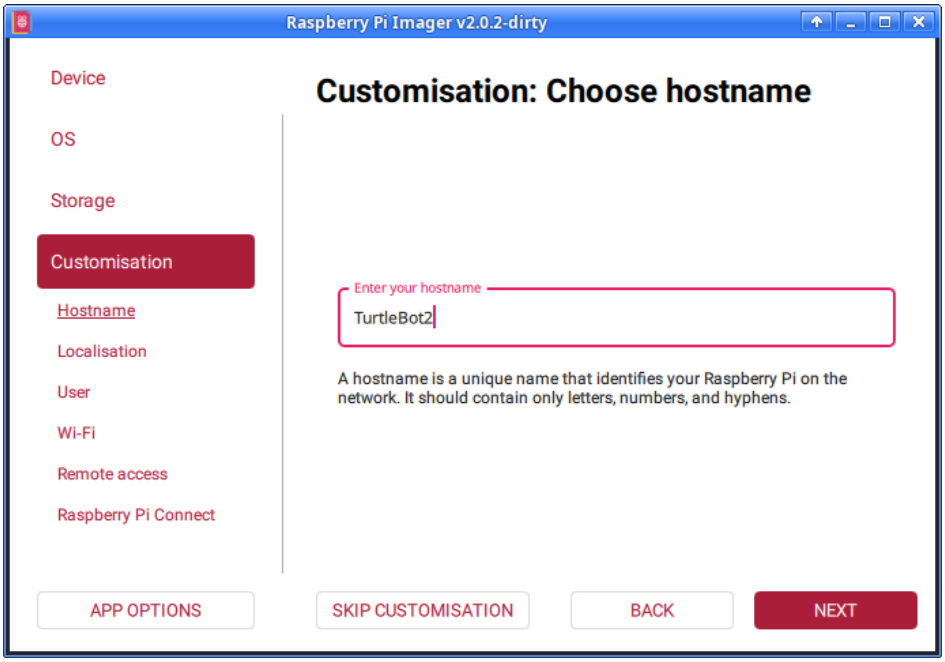

Figura 5: Configurar Hostname

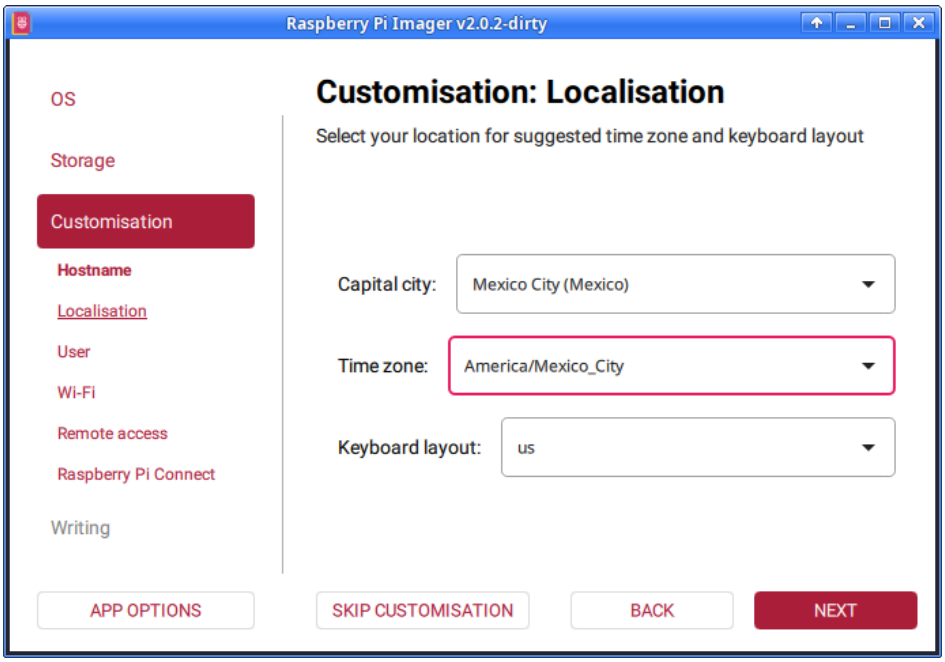

Figura 6: Configurar Localización

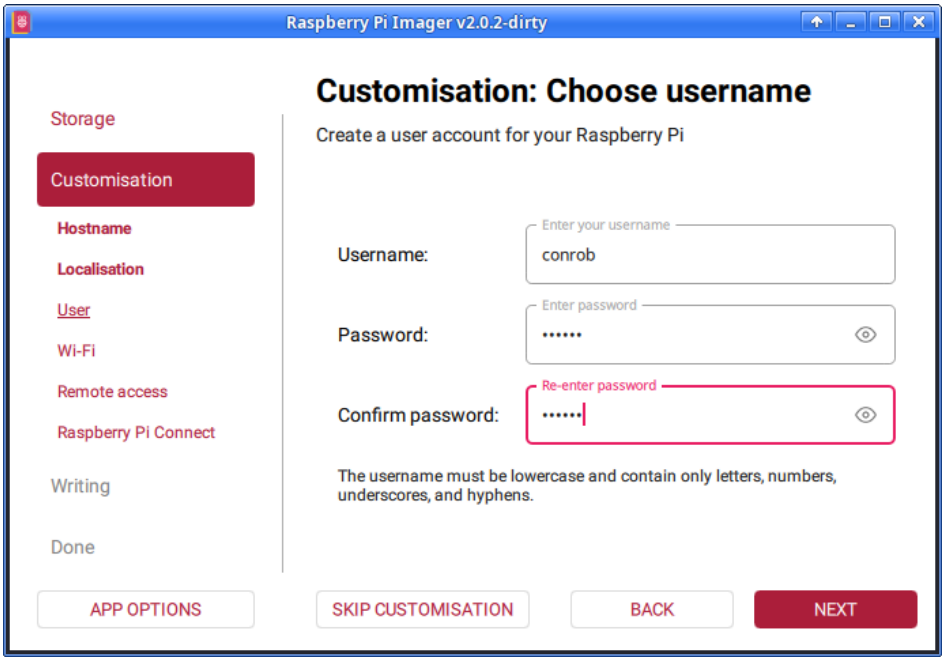

Figura 7: Configurar Usuario

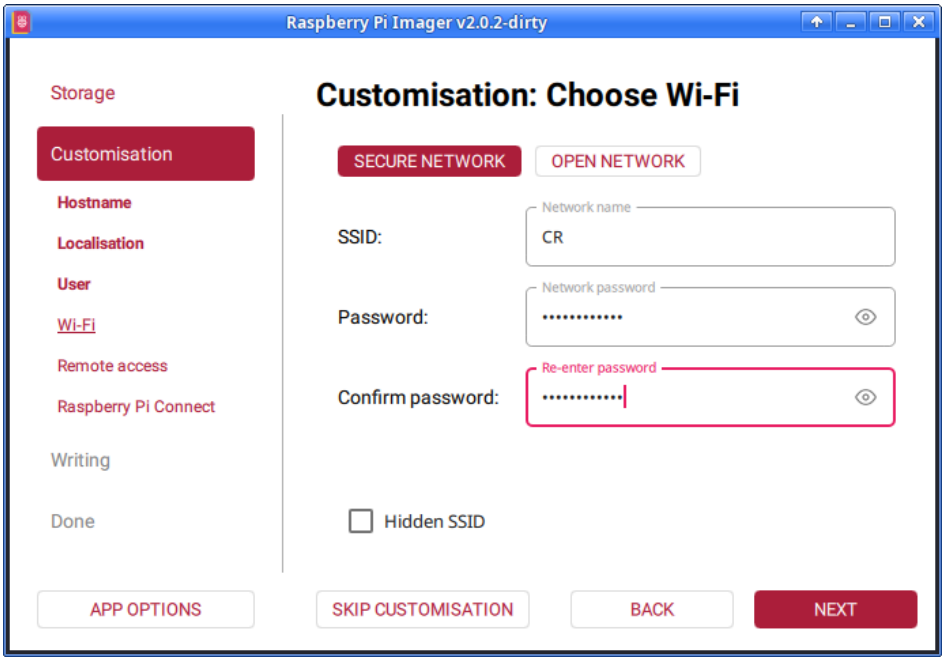

Figura 8: Configurar WiFi

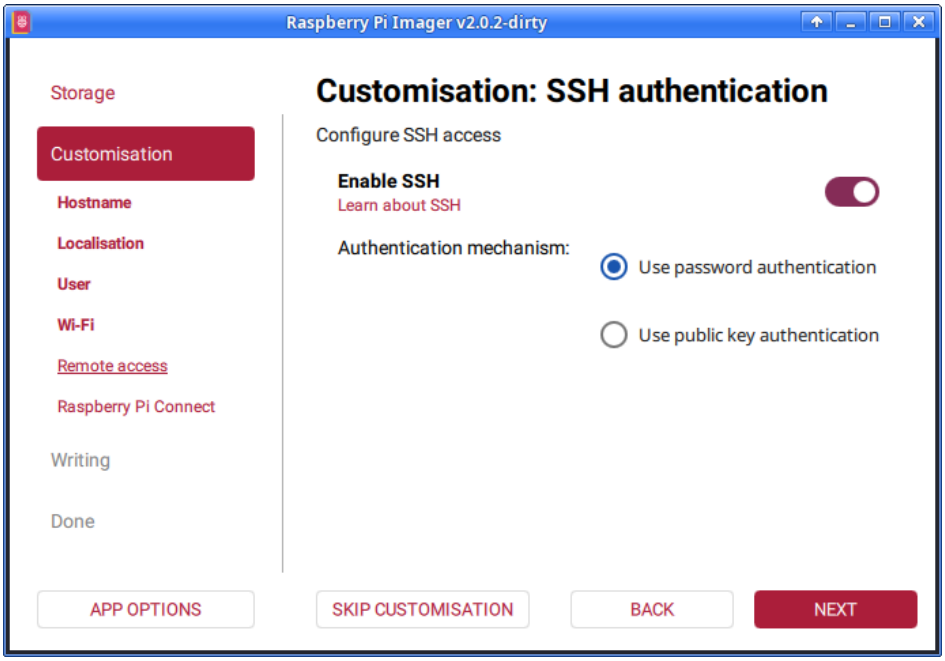

Figura 9: Habilitar SSH

Finalmente, presionar **Write** (Figura 10) y esperar a que finalice el proceso como se ve en la Figura 11.

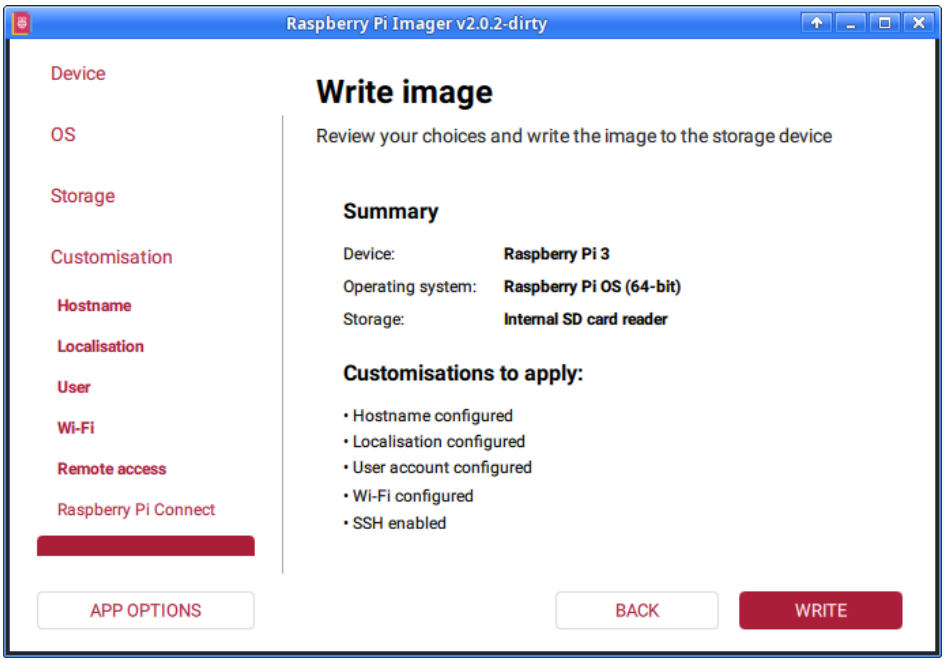

Figura 10: Escribir imagen en microSD

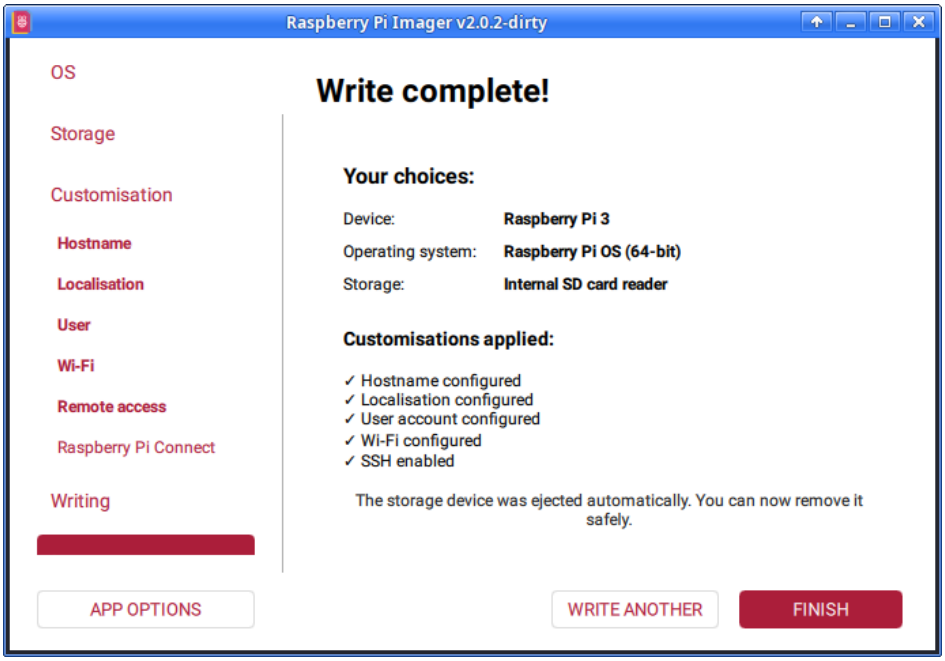

Figura 11: Proceso de escritura completado

4.3 Primer arranque y acceso SSH

Insertar la microSD en la Raspberry Pi y conectar la alimentación. El sistema arrancará automáticamente en unos 30 segundos.

4.3.1. Conocer la dirección IP de la Raspberry Pi

Para conectarse remotamente es necesario conocer la IP que el router asignó a la Raspberry Pi. Hay dos formas de obtenerla:

- **Con monitor conectado:** abrir una terminal en la Raspberry Pi y ejecutar:

  ```
  hostname -I
  ```

  El comando imprimirá algo como `192.168.1.105`. Ese número es la dirección IP.
- **Sin monitor:** revisar la lista de dispositivos conectados en la interfaz web del router. La Raspberry Pi aparecerá con el hostname configurado durante la instalación (p.ej. `turtlebot01`).

4.3.2. Conectarse por SSH desde la PC

SSH permite controlar la Raspberry Pi desde la PC sin necesidad de monitor ni teclado. Según el sistema operativo de la PC:

- **Windows:** abrir **PowerShell** o **Símbolo del sistema** (buscar “cmd” en el menú inicio).
- **Mac / Linux:** abrir una **Terminal**.

En cualquiera de los casos, ejecutar el siguiente comando, sustituyendo `usuario` por el nombre de usuario configurado durante la instalación, y `192.168.1.105` por la IP obtenida en el paso anterior:

```
ssh usuario@192.168.1.105
```

La primera vez preguntará si se confía en el host; escribir `yes` y presionar Enter. Luego pedirá la contraseña configurada durante la instalación.

Si la conexión es exitosa, el mensaje cambiará a algo como:

```
usuario@turtlebot01:~$
```

A partir de aquí, todos los comandos se ejecutan **en la Raspberry Pi** a través de esta terminal.

4.3.3. Actualizar el sistema operativo

Una vez conectado por SSH, actualizar el sistema:

```
sudo apt update
sudo apt upgrade -y
```

Cuando pregunte la contraseña, ingresar la misma del usuario. El proceso puede tardar varios minutos dependiendo de la conexión a internet.

5 Instalación de Dependencias del Proyecto

Todos los comandos de esta sección se ejecutan **en la Raspberry Pi**, ya sea con monitor y teclado conectados o a través de SSH desde la PC.

El controlador **no usa ROS**. Las dependencias necesarias son:

- **build-essential:** compilador de C++ y herramientas básicas de compilación.
- **cmake:** sistema de construcción del proyecto.
- **git:** control de versiones, necesario para descargar algunos paquetes.
- **libboost-all-dev:** biblioteca Boost, usada para la comunicación UDP y la integración numérica del controlador.
- **libboost-numeric-odeint-dev:** módulo de Boost para la integración Runge-Kutta 4 de la dinámica virtual.

Instalar todo con los siguientes comandos:

```
sudo apt update
sudo apt install -y build-essential cmake git
sudo apt install -y libboost-all-dev
sudo apt install -y libboost-numeric-odeint-dev
```

El proceso puede tardar varios minutos. El flag `-y` acepta automáticamente todas las confirmaciones.

5.1 Biblioteca matrix

La biblioteca `matrix` se usa para las operaciones de álgebra lineal del controlador (multiplicación de matrices, vectores de estado, etc.). No se instala con `apt` — debe descargarse manualmente y colocarse en una carpeta accesible al proyecto.

Una vez descargada, verificar que la ruta a sus headers esté correctamente indicada en el `CMakeLists.txt` (ver Apéndice D):

```
target_include_directories(controller PRIVATE
    include
    /ruta/a/matrix/include)   # ajustar según donde se descargó
```

Si esta ruta es incorrecta, la compilación fallará con un error de tipo `fatal error: matrix.h: No such file or directory`.

6 Conexión y Verificación del Hardware

6.1 Conexión del OpenCR a la Raspberry Pi

Conectar la base móvil (OpenCR) a la Raspberry Pi mediante un cable USB. El OpenCR aparece como dispositivo ACM:

```
ls /dev/ttyACM*
```

Dar permisos de acceso al puerto serial. La forma permanente es agregar el usuario al grupo `dialout`:

```
sudo usermod -aG dialout $USER
```

O bien, asignar permisos temporalmente (se pierden al reiniciar):

```
sudo chmod 666 /dev/ttyACM0
```

6.2 Alimentación

Verificar antes de encender:

- La batería LiPo 11.1 V está cargada.
- No existe caída de voltaje.
- Todos los conectores están firmes.

6.3 Firmware del OpenCR

El OpenCR **no usa el firmware estándar** del TurtleBot3. Debe cargarse el firmware personalizado `TorqueControl.ino`, que implementa control por corriente sobre los motores Dynamixel XM430.

El firmware se carga desde el Arduino IDE. Seguir los siguientes pasos:

6.3.1. Instalación del board de OpenCR

1. Instalar el Arduino IDE desde el sitio oficial (`https://www.arduino.cc/en/software`).
2. Abrir Arduino IDE y ir a **File** → **Preferences**. ver Figura 12

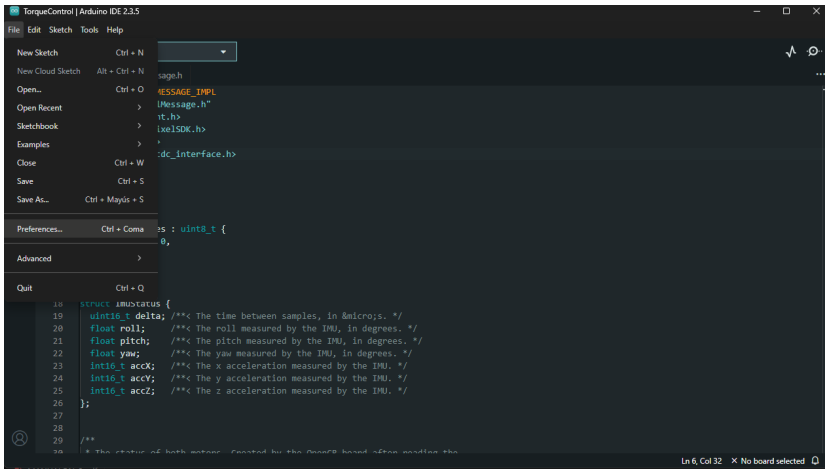

Figura 12: Preferencias

3. En el campo **"Additional boards manager URLs"**, pegar la siguiente URL (Figura 13):

   `https://raw.githubusercontent.com/ROBOTIS-GIT/OpenCR/master/arduino/opencr_release/package_opencr_index.json`

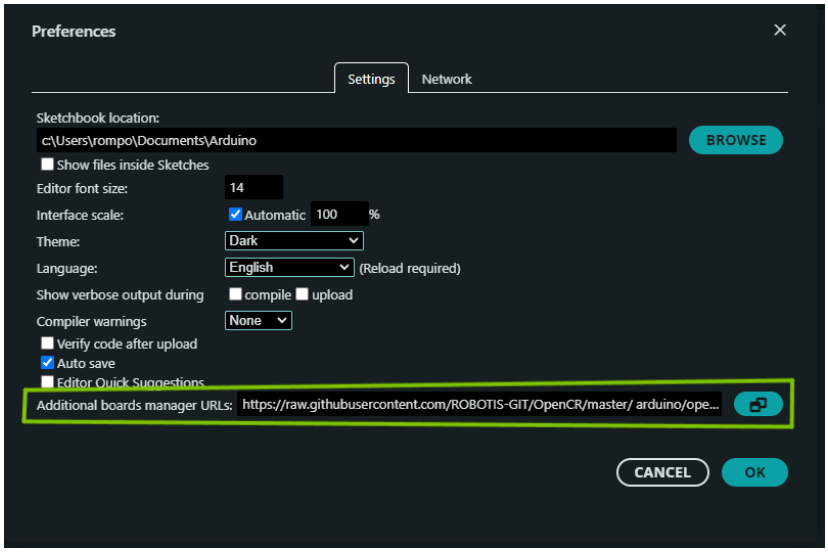

Figura 13: Pegar URL

4. Click en **OK**.
5. Ir a **Tools** → **Board** → **Boards Manager**.
6. Buscar **OpenCR** e instalarlo.

6.3.2. Cargar el firmware

1. Conectar el OpenCR a la computadora mediante USB.
2. En Arduino IDE, ir a **Tools** → **Board** y seleccionar **OpenCR**.
3. Ir a **Tools** → **Port** y seleccionar el puerto correspondiente al OpenCR.
4. Abrir el archivo `TorqueControl.ino`.
5. Presionar **Upload** (flecha hacia la derecha) y esperar a que finalice la carga.

Nota: Este proceso solo es necesario una vez, o cuando se realicen cambios al firmware. Durante la operación normal del robot, el OpenCR corre el firmware de forma autónoma.

Los detalles técnicos del firmware (protocolo serial, formato de mensajes, parámetros Dynamixel) se describen en el Apéndice A.

7 Configuración del Sistema OptiTrack

7.1 Configuración en Motive

1. Encender las cámaras OptiTrack.
2. Abrir Motive y verificar que todas las cámaras estén activas.
3. Crear un **Rigid Body** asignando los marcadores reflectivos del TurtleBot. El nombre del Rigid Body debe coincidir exactamente con el nombre usado en el código del controlador (p.ej. `TurtleBot`).
4. Verificar la reconstrucción 3D del Rigid Body.
5. **Cerrar Motive** antes de iniciar el servidor personalizado.

7.2 Servidor personalizado de comunicación

La comunicación con OptiTrack **no** usa el protocolo NatNet estándar. Se utiliza un servidor personalizado desarrollado internamente. Para iniciarlo:

```
Ejecutar el .exe llamado TestV2
```

Este ejecutable se encuentra en la computadora conectada a las camaras OptiTrack.

El servidor escucha en el **puerto 56789 UDP** y envía la pose del Rigid Body a la Raspberry Pi a $\sim$1000 Hz.

Los detalles del protocolo de comunicación (estructuras de datos, parámetros de suscripción, lógica de envío) se describen en el Apéndice B.

8 Compilación y Ejecución del Controlador

Todos los comandos de esta sección se ejecutan **en la Raspberry Pi** a través de SSH o con terminal directa.

8.1 Compilar el proyecto

Navegar hasta el directorio del proyecto y crear una carpeta de compilación:

```
cd robot_controller
mkdir build && cd build
cmake ..
make -j$(nproc)
```

Qué hace cada comando:

- `cd robot_controller`: entra a la carpeta del proyecto.
- `mkdir build && cd build`: crea una carpeta llamada `build` y entra en ella. Aquí se guardarán los archivos de compilación para no mezclarlos con el código fuente.
- `cmake ..`: lee el `CMakeLists.txt` del proyecto y prepara la compilación. Los dos puntos (`..`) indican que el `CMakeLists.txt` está en la carpeta anterior.
- `make -j$(nproc)`: compila el proyecto usando todos los núcleos disponibles del procesador (`nproc` devuelve el número de núcleos). En una Raspberry Pi 4 esto puede tardar entre 1 y 3 minutos.

Si la compilación es exitosa, aparecerá al final algo como:

```
[100%] Built target controller
```

Si hay errores, lo más común es que la biblioteca `matrix` no esté en la ruta correcta (ver Apéndice D).

8.2 Ejecutar el controlador

Antes de ejecutar, verificar que el OpenCR esté conectado por USB. El OpenCR aparece en Linux como un dispositivo serial llamado `/dev/ttyACM0` — ACM es el nombre que Linux asigna a dispositivos USB que se comunican como puerto serial, como es el caso del OpenCR.

Para verificar que el sistema lo detecta:

```
ls /dev/ttyACM*
```

Si aparece `/dev/ttyACM0`, el OpenCR está correctamente conectado. Si no aparece nada, revisar el cable USB.

Asignar permisos de acceso al puerto (si no se hizo de forma permanente en la Sección 6):

```
sudo chmod 666 /dev/ttyACM0
```

Ejecutar el controlador:

```
cd ~/robot_controller/build
./controller
```

Al arrancar, el programa realiza los siguientes pasos en orden:

1. Se conecta al servidor UDP en `192.168.5.109:56789` para recibir la pose del robot desde OptiTrack.

2. Se suscribe al tópico `PosTurtleBot` (ID 1), que corresponde al Rigid Body configurado en Motive.
3. Intenta abrir `/dev/ttyACM0` para comunicarse con el OpenCR. **Si el OpenCR no está conectado, el programa se queda esperando en este paso** hasta que se conecte el cable USB — en ese momento continúa automáticamente sin necesidad de reiniciar.
4. Lanza el hilo de monitoreo, que imprime el estado del sistema en consola cada 16 ms (IMU, motores, pose).
5. Entra al loop de control, que se ejecuta cada 1 ms enviando comandos de corriente al OpenCR.

Si todo funciona correctamente, la consola comenzará a mostrar los valores del IMU y los motores actualizándose continuamente. Para detener el programa, presionar `Ctrl+C`.

9 Operación del Sistema

Esta sección describe los pasos completos para operar el sistema. Es el núcleo del manual — las secciones anteriores cubren la instalación y configuración que solo se realiza una vez, mientras que esta sección se sigue **en cada sesión de trabajo**.

9.1 Verificaciones Previas

Antes de encender, confirmar que:

- La batería LiPo está cargada (voltaje nominal 11.1 V).
- Todos los cables USB están conectados firmemente.
- Los marcadores reflectivos del TurtleBot están bien colocados y tienen línea de vista libre hacia las cámaras OptiTrack.
- La PC con Motive y la Raspberry Pi están conectadas a la misma red WiFi.
- El ejecutable `TestV2.exe` está disponible en la PC.

9.2 Secuencia de Encendido

Seguir el orden exacto — cada paso depende del anterior.

9.2.1. En la PC: OptiTrack y servidor

1. **Encender las cámaras OptiTrack.** Esperar unos segundos a que inicialicen.
2. **Abrir Motive** y verificar que:
   - Todas las cámaras aparecen activas.
   - El Rigid Body del TurtleBot es visible y su reconstrucción 3D es estable (sin saltos ni marcadores faltantes).
3. **Cerrar Motive.** Este paso es obligatorio — el servidor personalizado y Motive no pueden correr al mismo tiempo porque ambos acceden al mismo hardware.

4. **Iniciar el servidor personalizado:** ejecutar el archivo `TestV2.exe`. La ventana del servidor debe quedar abierta durante toda la sesión. Si se cierra, el robot perderá la pose y se detendrá automáticamente.

9.2.2. En el TurtleBot: encendido y conexión

5. **Conectar la batería** al TurtleBot para encenderlo. Esperar a que el OpenCR termine de inicializar (el LED del OpenCR dejará de parpadear).
6. **Conectarse a la Raspberry Pi por SSH** desde la PC. Abrir una terminal (PowerShell en Windows, Terminal en Mac/Linux) y ejecutar:

   ```
   ssh usuario@<IP_RASPBERRY>
   ```

   Sustituir `<IP_RASPBERRY>` por la dirección IP de la Raspberry Pi (ver Sección 4.3).
7. **Verificar que el OpenCR es detectado:**

   ```
   ls /dev/ttyACM*
   ```

   Debe aparecer `/dev/ttyACM0`. Si no aparece, revisar el cable USB entre el OpenCR y la Raspberry Pi.
8. **Asignar permisos al puerto serial:**

   ```
   sudo chmod 666 /dev/ttyACM0
   ```

9. **Ejecutar el controlador:**

   ```
   cd ~/robot_controller/build
   ./controller
   ```

   Si todo está correcto, la consola comenzará a mostrar los valores del IMU y los motores actualizándose continuamente. Si la pantalla no muestra datos, revisar la sección de Resolución de Problemas.

9.2.3. Activar el movimiento

10. **Desde la PC**, una vez que el controlador esté corriendo, activar el robot enviando `should_run = 1` a través del servidor `TestV2`. El robot comenzará a moverse hacia el punto de referencia configurado.

9.3 Durante la Operación

Mientras el sistema está activo, tener en cuenta:

- La consola SSH mostrará el estado del sistema en tiempo real (IMU, motores, pose). Si los valores dejan de actualizarse, hay un problema de comunicación.
- **No mover las cámaras OptiTrack** durante la sesión.

- **No desconectar cables USB** con el sistema encendido — el firmware detectará la pérdida de comunicación y detendrá los motores en menos de 10 ms, pero una reconexión brusca puede dañar el hardware.
- Si el robot se detiene inesperadamente, revisar la sección de Resolución de Problemas antes de reiniciar.

9.4 Secuencia de Apagado

Seguir el orden inverso al encendido para evitar daños.

1. **Detener el movimiento:** desde la PC, enviar `should_run = 0` a través del servidor. Esperar a que el robot se detenga completamente antes de continuar.
2. **Detener el controlador** en la Raspberry Pi presionando `Ctrl+C` en la terminal SSH.
3. **Apagar la Raspberry Pi** de forma segura (nunca desconectar la alimentación directamente, ya que puede corromper la microSD):

   ```
   sudo shutdown now
   ```

   Esperar a que los LEDs de la Raspberry Pi se apaguen antes de desconectar la alimentación.
4. **Desconectar la batería** del TurtleBot.
5. **Cerrar el servidor** `TestV2` en la PC.
6. **Apagar las cámaras OptiTrack.**

10 Resolución de Problemas

10.1 No llega información desde el servidor OptiTrack

- Verificar que la IP del servidor esté correctamente configurada en `main.cpp`.
- Verificar que el firewall no bloquee el puerto 56789 UDP:

  ```
  sudo ufw allow 56789/udp
  ```

- Confirmar que el servidor personalizado esté activo.
- Revisar que todos los dispositivos estén en la misma red WiFi.

10.2 Pose incorrecta o errática

- Verificar la calibración en Motive.
- Confirmar la orientación del Rigid Body (el eje frontal del robot debe coincidir con el eje del Rigid Body en Motive).
- Revisar el sistema de coordenadas: OptiTrack usa Y hacia arriba; el plano del suelo es XZ. Verificar que la conversión sea correcta en el controlador (ver Apéndice C).
- Confirmar que los marcadores reflectivos tengan línea de vista libre.

10.3 Error de comunicación con OpenCR

- Verificar que el cable USB entre Raspberry Pi y OpenCR esté conectado.
- Verificar permisos del puerto serial:

```
sudo chmod 666 /dev/ttyACM0
```

- Si el programa se queda bloqueado esperando el puerto, conectar el USB y el programa continuará automáticamente.
- Verificar que el baudrate en `main.cpp` sea `B1152000` y que el firmware del OpenCR use `Serial.begin(1152000)`.
- Si los paquetes llegan corruptos, verificar que ambos lados usen la misma versión de `SerialMessage.h`.

10.4 Los motores no responden (sin movimiento)

- Verificar que el OpenCR recibe el comando: en la consola del controlador debe aparecer `Last command: X Y` si `hasLastCommand` es verdadero.
- Confirmar que han pasado menos de 10 ms entre comandos (el firmware silencia los motores si pasan más de 10 ms).
- Verificar que el firmware esté en modo corriente (registro 11 = 0) y con torque habilitado (registro 64 = 1).
- Revisar que los valores de corriente no estén en cero por ganancias muy bajas en el controlador (ver Apéndice C).

10.5 Los motores se frenan solos durante la operación

El firmware tiene un mecanismo de seguridad: si no llega un nuevo `CUR_COMMAND` en 10 ms, la corriente se pone a cero. Esto puede ocurrir si:

- El loop de control en la Raspberry Pi tarda más de 10 ms (verificar carga del sistema con `top`).
- La pose de OptiTrack deja de llegar (el controlador envía corriente cero cuando `poses_by_id` está vacío).
- El cable USB se desconectó momentáneamente.

10.6 El robot se mueve cuando `should_run` es 0

- Verificar que el ciclo de control lee correctamente `commands_by_id` y envía corriente cero antes de aplicar la ley de control.
- Verificar que el nombre del Rigid Body en Motive coincida *exactamente* con el tópico registrado (si el tópico es `PosRobot1`, el Rigid Body debe llamarse `Robot1`).
- Confirmar que el *magic number* del cliente y del servidor sea el mismo (`12345`).

10.7 Datos descartados por antigüedad (dato obsoleto)

Si `poses_by_id` se vacía durante la operación, es posible que los datos de OptiTrack lleguen con más de 20 ms de retraso. Verificar:

- Latencia de red entre la PC con Motive y el servidor.
- Que `update_body()` se llame con suficiente frecuencia.
- Aumentar `max_sample_time` en `handle_data_response()` si la red es lenta.

11 Buenas Prácticas

- Verificar la calibración de OptiTrack antes de cada sesión de trabajo.
- Mantener línea de vista limpia entre cámaras y marcadores reflectivos.
- No mover las cámaras OptiTrack durante la operación.
- No desconectar dispositivos USB con el sistema energizado.
- Mantener baterías correctamente cargadas antes de cada uso.
- Documentar cualquier cambio en parámetros o en el protocolo personalizado.
- No modificar el protocolo de comunicación personalizado sin documentar los cambios.
- Verificar integridad de cables antes de cada uso.

A Firmware del OpenCR: Referencia Técnica

A.1 Funcionamiento general

El firmware personalizado `TorqueControl.ino` corre un loop continuo con tres tareas:

- **pollSerial():** lee y escribe paquetes seriales hacia/desde la Raspberry Pi.
- **pollIMU():** lee el IMU cada ∼1 ms y envía `ImuStatus` a la Raspberry Pi.
- **pollMotors():** aplica el comando de corriente recibido a los motores y envía `MotorStatus` a la Raspberry Pi cada ∼1 ms.

A.2 Seguridad: expiración de comandos

Si la Raspberry Pi deja de enviar comandos por más de **10 ms** (`maxCommandTime = 10000 µs`), el OpenCR pone automáticamente la corriente de ambos motores a cero. Entre 1.5 ms y 10 ms sin comando, la corriente se reduce proporcionalmente hasta cero. Esto protege al robot ante una pérdida de comunicación.

A.3 Configuración de los motores Dynamixel

El firmware configura los motores en **modo corriente (modo 0)** al arrancar. Los parámetros relevantes son:

Parámetro	Valor	Descripción
Operating Mode (reg 11)	0	Control de corriente
Drive Mode (reg 9)	0	Modo normal
Torque Enable (reg 64)	1	Motores habilitados
Baudrate Dynamixel	3,000,000 bps	Bus interno OpenCR
Baudrate Serial (USB)	1,152,000 bps	Comunicación con RPi

A.4 Formato del comando de corriente

La Raspberry Pi envía un paquete de tipo `CUR_COMMAND` con:

- `lCur`: corriente deseada motor izquierdo (`int16_t`, unidades de 2.69 mA)
- `rCur`: corriente deseada motor derecho (`int16_t`, unidades de 2.69 mA)

Rango válido: $[-300, +300]$ unidades ($\approx \pm 807$ mA).

A.5 Datos que envía el OpenCR a la Raspberry Pi

ImuStatus (cada ∼1 ms):

- `delta`: tiempo entre muestras en µs
- `roll`, `pitch`, `yaw`: ángulos en grados
- `accX`, `accY`, `accZ`: aceleración (escala $\pm 2g$, raw `int16_t`)

MotorStatus (cada ∼1 ms):

- `delta`: tiempo entre muestras en µs

- `lCur`, `rCur`: corriente medida en unidades de 2.69 mA
- `lVel`, `rVel`: velocidad en unidades de 0.229 RPM
- `lPos`, `rPos`: posición en unidades de rev/4096

A.6 Protocolo serial Raspberry Pi – OpenCR

La comunicación usa el puerto `/dev/ttyACM0` a **1,152,000 bps** con un protocolo de paquetes binarios personalizado (`SerialMessage.h`).

Estructura de un paquete:

1. Bytes de inicio: `turl\n` (5 bytes)
2. Longitud del payload: `uint32_t` (4 bytes)
3. Datos intercalados con bytes de separación `b`, en bloques de 4 bytes
4. Checksum XOR de 32 bits al final

Tipos de mensaje (primer byte del payload):

Tipo	Valor	Dirección
`OCR_MESSAGE`	0	Bidireccional (texto debug)
`IMU_STATUS`	1	OpenCR → RPi
`MOTOR_STATUS`	2	OpenCR → RPi
`CUR_COMMAND`	3	RPi → OpenCR

B Protocolo de Comunicación UDP: Referencia Técnica

B.1 Visión general

Motive → Servidor Personalizado → `ControlClient` (C++)

El servidor y el cliente están implementados en C++ moderno usando **Boost.Asio** para operaciones de red asíncronas sobre UDP. Todos los mensajes son **binarios con tamaño fijo**, validados en tiempo de compilación con `static_assert`.

Instalación de Boost en Raspberry Pi OS:

```
sudo apt install libboost-all-dev
```

B.2 Parámetros de configuración

Parámetro	Valor por defecto	Dónde cambiarlo
Puerto servidor	56789	Constructor de `ControlServer`
Magic number	12345	Atributo `expected_magic`
Intervalo de envío	1 ms	Atributo `send_interval`
Intervalo suscripción	10 ms	Atributo `send_interval` del cliente
Duración suscripción	1,000,000 µs (1 s)	`subscribe_for` en el cliente
Máx tópicos por paquete	4	Constante `max_topics`
Frescura máxima	20 ms	`max_sample_time` en cliente

B.3 Estructuras de datos del protocolo

B.3.1. Solicitud (Cliente → Servidor)

`SubscriptionRequest` (136 bytes):

- `topics[4][16]`: nombres de tópicos (máx 4, cada uno máx 16 chars)
- `num_topics`: cuántos tópicos se suscriben en esta solicitud
- `ids[4]`: identificadores locales asignados por el cliente (`uint64_t`)
- `subscribe_for[4]`: duración de la suscripción en microsegundos

`Request` (152 bytes): envoltorio general de solicitud.

- `magic`: número de validación (`uint64_t`, valor esperado: `12345`)
- `type`: tipo de solicitud (`RequestType::Subscription = 1`)
- union con `SubscriptionRequest`

B.3.2. Datos de pose y comando

`PoseData` (32 bytes):

- `timestamp`: microsegundos desde epoch (`uint64_t`, 8 bytes)
- `xyz[3]`: posición en metros (floats, 12 bytes)
- `ypr[3]`: orientación en **yaw, pitch, roll** en radianes (floats, 12 bytes)

Nota: la orientación se envía como **YPR**, no como cuaterniones.
`CommandData` (16 bytes):

- `timestamp`: microsegundos desde epoch (`uint64_t`)
- `should_run`: 1 si el robot debe moverse, 0 si debe detenerse (`uint8_t`)

B.3.3. Respuesta (Servidor → Cliente)

`DataResponse`: puede contener hasta 4 tópicos por paquete.

- `ids[4]`: IDs de los tópicos
- `num_topics`: cuántos tópicos vienen en este paquete
- `types[4]`: tipo de cada dato (`Pose = 1`, `Command = 2`)
- `data[4]`: union de `PoseData` o `CommandData`

`Response` (192 bytes): envoltorio general de respuesta.

- `magic`: número de validación (`uint64_t`)
- `timestamp`: tiempo del servidor al momento del envío
- `type`: tipo de respuesta (`ResponseType::Data = 1`)
- union con `DataResponse`

B.4 Clase ControlServer

El servidor escucha en el puerto **56789 UDP** y gestiona suscriptores.

B.4.1. Atributos principales

- `bodies`: mapa nombre → `PoseData` con las poses actuales
- `subscribers`: mapa endpoint → `SubscriberInfo`

- `should_run`: bandera global de marcha
- `send_interval`: intervalo de envío (por defecto 1 ms)
- `expected_magic`: valor de validación (por defecto 12345)

Llamado para actualizar la pose de un Rigid Body:

```
server.update_body("Robot1", {x, y, z}, {yaw, pitch, roll});
```

B.4.2. Lógica de envío: handle_send()

Cada 1 ms el servidor: elimina suscripciones expiradas; para cada suscriptor activo construye un `Response` con sus tópicos (prefijo `Pos` → `PoseData`; prefijo `Cmd` → `CommandData`); y envía el paquete UDP. Si hay más de 4 tópicos, envía múltiples paquetes.

B.4.3. Expiración de suscripciones

La condición de expiración es:

```
timestamp_actual - subscribed_at > subscribed_for
```

El cliente renueva por defecto cada 10 ms con duración de 1 segundo.

B.5 Clase ControlClient

B.5.1. Construcción y suscripción a tópicos

```
io_context ctx;
ControlClient client(ctx, "192.168.1.100", 56789);
client.add_subscription("PosRobot1", 1);  // pose, id=1
client.add_subscription("CmdRobot1", 2);  // comando, id=2
ctx.run();
```

El prefijo `Pos` indica datos de pose; el prefijo `Cmd` indica comandos. El nombre tras el prefijo debe coincidir exactamente con el nombre del Rigid Body en Motive.

B.5.2. Lectura de datos recibidos

```
// Pose: xyz en metros, ypr en radianes
PoseData& p = client.poses_by_id[1];
float x   = p.xyz[0];
float y   = p.xyz[1];
float z   = p.xyz[2];
float yaw = p.ypr[0];

// Comando de marcha
CommandData& c = client.commands_by_id[2];
bool run = c.should_run;
```

B.5.3. Validación de frescura de datos

El cliente descarta datos cuyo timestamp sea más antiguo que **20 ms** respecto al timestamp del servidor:

```
max_sample_time = 20 ms
if (resp.timestamp - data.timestamp > max_sample_time)
    // dato descartado (demasiado viejo)
```

B.5.4. Integración en el ejecutable principal

El cliente y el controlador comparten el mismo `io_context` de Boost.Asio. El patrón típico:

```
io_context ctx;
ControlClient client(ctx, "192.168.1.100", 56789);
client.add_subscription("PosRobot1", 1);
client.add_subscription("CmdRobot1", 2);

std::thread io_thread([&ctx]() { ctx.run(); });

while (true) {
    auto it = client.poses_by_id.find(1);
    if (it != client.poses_by_id.end()) {
        // usar it->second para calcular control
    }
    std::this_thread::sleep_for(std::chrono::milliseconds(10));
}
io_thread.join();
```

C Ley de Control: Referencia Técnica

C.1 Sistema de coordenadas

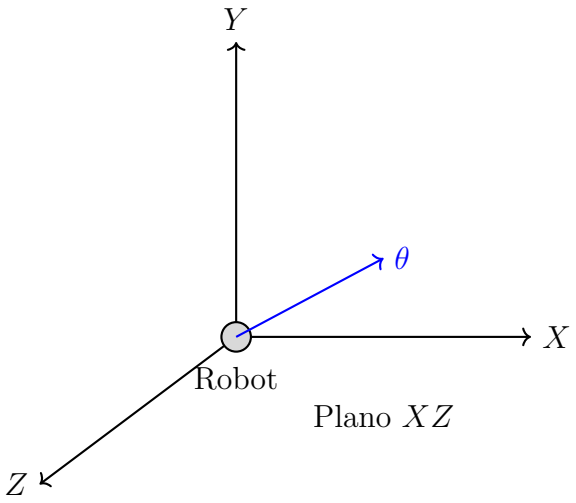

Figura 14: Sistema de coordenadas de OptiTrack. El eje Y apunta hacia arriba; el plano del suelo es XZ.

C.2 Extracción de pose desde OptiTrack

El controlador se suscribe al Rigid Body `TurtleBot` con ID 1:

```
ControlClient client(ctx, "192.168.5.109");
client.add_subscription("PosTurtleBot", 1);
```

En cada ciclo:

```
auto& pose = client.poses_by_id[1];
pos   = mat<2,1>{ pose.xyz[0], pose.xyz[2] }; // plano XZ
theta = pose.ypr[0] / 180 * pi;               // yaw en radianes
```

Se usan `xyz[0]` y `xyz[2]` (no `xyz[1]`) porque el eje Y de OptiTrack apunta hacia arriba. El yaw viene en grados y se convierte a radianes multiplicando por $\pi/180$.

C.3 Modelo cinemático diferencial

$$\dot{x} = v\cos\theta, \quad \dot{y} = v\sin\theta, \quad \dot{\theta} = \omega$$

C.4 Ley de control con dinámica virtual

El controlador implementa una ley de control basada en **dinámica virtual** con integración numérica Runge-Kutta 4 (via Boost.Odeint).

Variable	Valor actual	Descripción
`kv`	1.0	Ganancia de velocidad lineal
`kw`	1.0	Ganancia de velocidad angular
`dv`	1.0	Amortiguamiento dinámica lineal
`dw`	1.0	Amortiguamiento dinámica angular
`pv`	1.0	Ganancia proporcional lineal
`pw`	1.0	Ganancia proporcional angular
`kalpha`	1.0	Ganancia de acoplamiento angular
`kM`	0.5	Masa del robot
`kI`	1.0	Inercia del robot
`kR`	0.15	Radio de la base diferencial
`kr`	0.066	Radio de la rueda

Las señales de control calculadas son:

```
float Uv = -kv * (phi_t(theta) * (z_bar - dynv))[0][0];
float Uw = -kw * (theta - dynw);
```

donde `z_bar = pos - delta` es la posición relativa al offset `delta` (permite desplazar el punto de referencia del robot).

C.5 Conversión a comandos de corriente

```
float Tl = kr / 2 * (Uv * kM + (kI * Uw) / (2 * kR));
float Tr = kr / 2 * (Uv * kM - (kI * Uw) / (2 * kR));

// Saturado a [-300, 300] unidades Dynamixel
int16_t lCur = clamp<short>(Tl * 20, -300, 300);
int16_t rCur = clamp<short>(Tr * 20, -300, 300);
WriteCurrentCommand(CurrentCommand(lCur, rCur));
```

El factor `* 20` es la ganancia de corriente (ajustable). El rango $[-300, +300]$ corresponde a $\approx \pm 807\,\text{mA}$ en los XM430.

C.6 Flujo completo del ciclo de control

El loop principal en `main.cpp` se ejecuta cada **1 ms**:

1. `ctx.poll_one()`: procesa un evento de red UDP (actualiza `poses_by_id`).
2. Verifica si hay pose disponible en `poses_by_id[1]`.
3. Si hay pose: extrae posición y yaw, ejecuta la dinámica virtual, calcula torques y envía `CurrentCommand` al OpenCR via serial.
4. Si no hay pose: envía corriente cero a ambos motores.
5. `pollSerial()`: procesa los bytes seriales pendientes del OpenCR (recibe `ImuStatus` y `MotorStatus`).

C.7 Sincronización sin bloqueo: LockFreeLatest

Para evitar condiciones de carrera entre el hilo de impresión y el loop de control, los datos del IMU y los motores se intercambian usando la estructura `LockFreeLatest<T>`, que mantiene 3 buffers (libre, compartido, último) y permite escritura y lectura concurrentes sin mutex:

```
LockFreeLatest<ImuStatus>   imuStatus({});
LockFreeLatest<MotorStatus> motorStatus({});

// Escritura (hilo serial):
imuStatus.getFree() = nuevosDatos;
imuStatus.submit();

// Lectura (hilo de impresión):
auto& latest = imuStatus.getLatest();
```

D Estructura del Proyecto y Compilación

D.1 Archivos del proyecto

```
robot_controller/
  main.cpp              - Loop principal, control, serial
  LockFreeLatest.h      - Buffer triple lock-free
  SerialMessage.h       - Protocolo serial RPi <-> OpenCR
  SerialOpenCR.cpp      - Clase de conexión serial
  udp_server.h          - ControlClient (UDP / OptiTrack)
  include/
    current_command.h   - Estructura CurrentCommand
    imu_status.h        - Estructura ImuStatus
    motor_status.h      - Estructura MotorStatus
  TorqueControl/
    TorqueControl.ino   - Firmware del OpenCR
    SerialMessage.h     - (mismo protocolo, versión Arduino)
```

D.2 CMakeLists.txt

```
cmake_minimum_required(VERSION 3.10)
project(robot_controller)
set(CMAKE_CXX_STANDARD 17)

find_package(Boost REQUIRED COMPONENTS system)
find_package(Threads REQUIRED)

add_executable(controller main.cpp)

target_include_directories(controller PRIVATE
```

```
    include
    /path/to/matrix/include)   # ajustar según instalación

target_link_libraries(controller
    Boost::system
    Threads::Threads)
```

D.3 Compilación y ejecución

```
cd robot_controller
mkdir build && cd build
cmake ..
make -j$(nproc)

# Ejecutar:
./controller
```