

Università degli Studi di Padova – Dipartimento di Ingegneria dell'Informazione
Corso di Laurea in Ingegneria dell'Automazione e dei Sistemi

Relazione per la prova finale
***«Sviluppo di un bridge di comunicazione Micro-ROS
tra Unitree Go2 Pro e dispositivi esterni»***

Relatore: Prof. Pozzebon Alessandro
Correlatori: Prof. Trivellin Nicola, Dott. Manfrin Manuel

Laureando: De Pazzi Mattia
Matricola n.: 2106455

Padova, 23/09/2026

<u>Introduzione</u>	3	<u>Protocollo di Comunicazione</u>	11
<u>Contesto di sviluppo</u>	4	<u>Funzionamento del Sistema</u>	14
<u>Unitree Go2 Pro</u>	5	<u>Video Simulazione</u>	15
<u>STM32</u>	6	<u>Conclusioni</u>	16
<u>ROS 2 e Micro-ROS</u>	8	<u>Bibliografia</u>	17

Il progetto di tesi descritto in questa presentazione riguarda lo sviluppo di un ponte di comunicazione tra il sistema operativo del cane robot Unitree Go2 modello Pro e i dispositivi necessari alla realizzazione del suo scopo ultimo, ovvero la dissuasione dei lupi in aree ben definite.

Lo scopo del progetto è far sì che il robot si rechi in autonomia nel luogo indicato da dei sensori posizionati nel perimetro della zona da controllare; una volta sul posto, grazie alla telecamera integrata, dovrà riconoscere il lupo e procedere con l'attività di dissuasione.

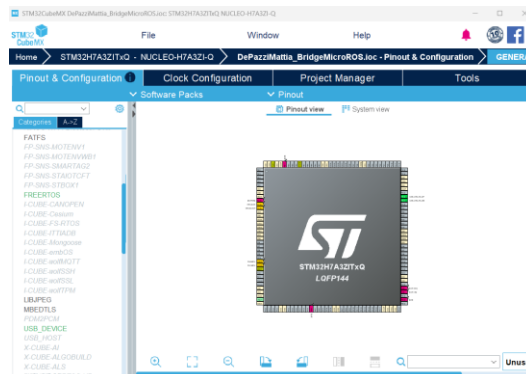
Il computer interno del robot comunica con i dispositivi esterni (es. GPS e dissuasore) collegati ad una scheda STM32 modello NUCLEO-H7A3ZI-Q grazie ai protocolli ROS2 e Micro-ROS e un protocollo di comunicazione USB.



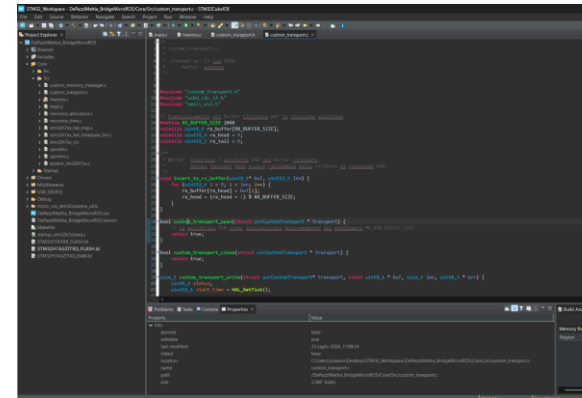
CONTESTO DI SVILUPPO



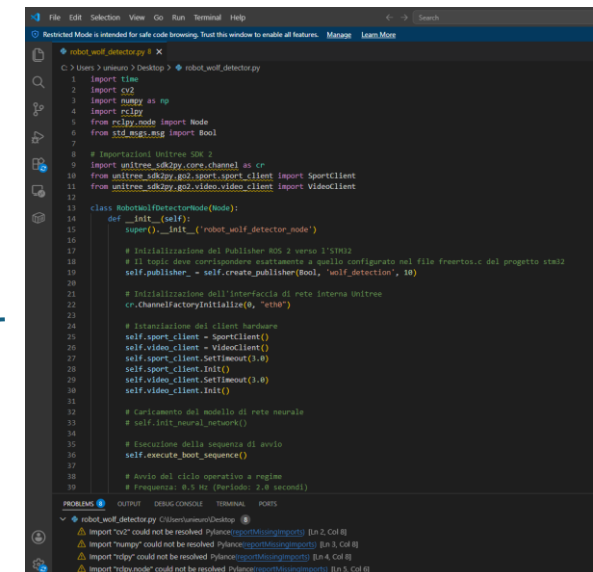
1 - Configurazione scheda STM32 con STM32CubeMX



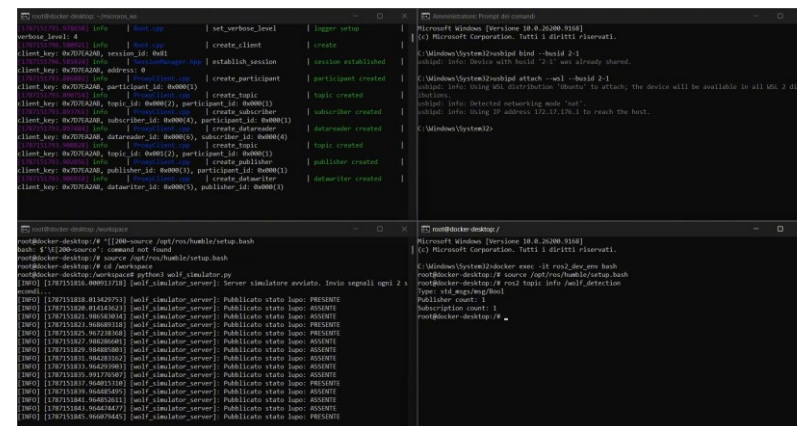
2 - Sviluppo del firmware per la scheda STM32 con STM32CubeIDE



3 - Sviluppo del nodo ROS2 da scaricare nel robot



4 - Simulazione della comunicazione tra robot e scheda STM32



UNITREE Go2 PRO

Il modello Unitree Go2 Pro è una piattaforma robotica quadrupede destinata all'uso civile, dimostrativo e di ispezione, il quale presenta alcuni sensori integrati che rendono possibile la sua interazione con il mondo esterno:

- Il sensore principale è un LiDAR emisferico proprietario che offre un campo visivo di 360° orizzontali e 90° verticali. Il sistema presenta un punto cieco ridotto al minimo, con una distanza di rilevamento che parte da 0,05 metri.
- Il comparto ottico include una fotocamera HD grandangolare con risoluzione di 1280x720 pixel e un campo visivo di 120°, utilizzata principalmente per la trasmissione video a bassa latenza all'operatore (RTT 2.0, tecnologia proprietaria Unitree).
- Il robot integra un sistema di posizionamento vettoriale wireless (Wireless Vector Positioning Tracker), moduli Wi-Fi 6 (dual-band), Bluetooth 5.2 e un modulo 4G tramite eSIM.

Il calcolo cinematico e l'elaborazione dei dati sensoriali sono affidati a un processore a 8 core ad alte prestazioni. Questo modulo gestisce in tempo reale il bilanciamento dinamico e i calcoli per il tracciamento ambientale.



STM32 NUCLEO-H7A3ZI-Q

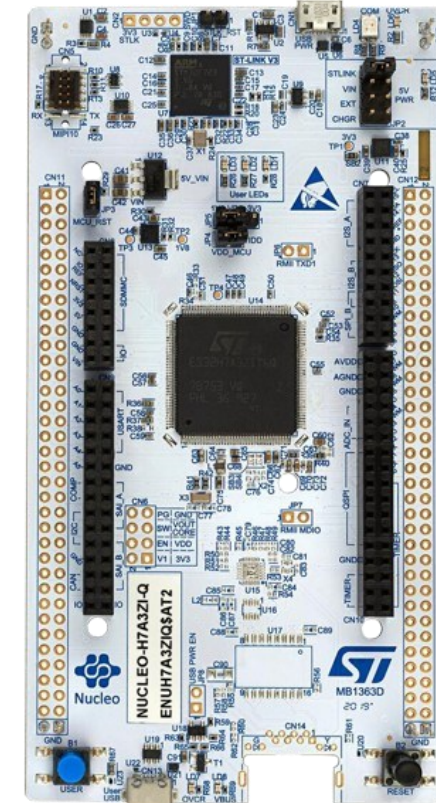
La scheda di sviluppo NUCLEO-H7A3ZI-Q fa parte della famiglia STM32 Nucleo-144 prodotta da STMicroelectronics. Si tratta di una piattaforma hardware concepita per la prototipazione e lo sviluppo di sistemi embedded, fornendo accesso diretto alle periferiche di un microcontrollore STM32H7A3ZIT6Q.

Microcontrollore

Il core di questo microcontrollore è basato sull'architettura ARM Cortex-M7, con frequenza di clock massima di 280 MHz. L'inclusione di una FPU (Floating Point Unit) in doppia precisione permette l'esecuzione efficiente di algoritmi matematici complessi. Il chip dispone di 2 MB di memoria Flash per l'archiviazione del firmware e di circa 1.4 MB di RAM (suddivisa in aree funzionali per l'esecuzione critica del codice senza stati di attesa e la SRAM per la gestione dei buffer e delle periferiche).

Gestione Energetica

Il suffisso logico "-Q" nel codice identificativo indica l'inclusione di un convertitore SMPS (Switched-Mode Power Supply) integrato per la generazione della tensione di alimentazione del core logico (V_{core}). Questa scelta architettonica è cruciale per la riduzione dei consumi in modalità di esecuzione.



STM32 NUCLEO-H7A3ZI-Q



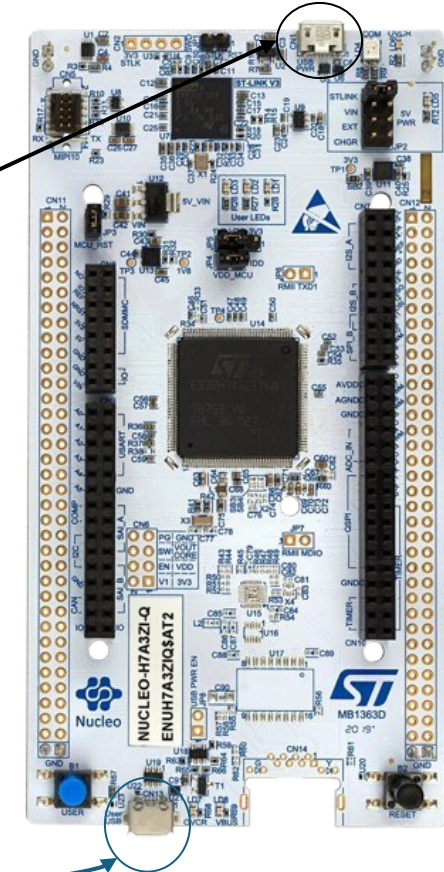
Programmatore e debugger

La scheda integra a bordo un circuito programmatore e debugger STLINK-V3E (con porta USB dedicata). Tale strumento esonera l'utente dall'utilizzo di sonde JTAG/SWD separate. L'STLINK supporta la programmazione flash del microcontrollore e fornisce una porta COM virtuale collegata all'UART della MCU.

Porte USB integrate

Sulla scheda sono fisicamente presenti due interfacce USB: mentre la prima è adibita in modo esclusivo al circuito STLINK per il debugging e la programmazione del firmware, la porta contrassegnata come User USB è collegata direttamente ai pin di ingresso/uscita del microcontrollore centrale STM32H7A3.

Lo scopo della porta User USB è fornire allo sviluppatore l'accesso fisico diretto alla periferica hardware USB OTG FS (On-The-Go Full Speed) integrata nel silicio del microcontrollore, permettendo di implementare e testare protocolli di comunicazione USB personalizzati.



ROS 2 e Micro-ROS

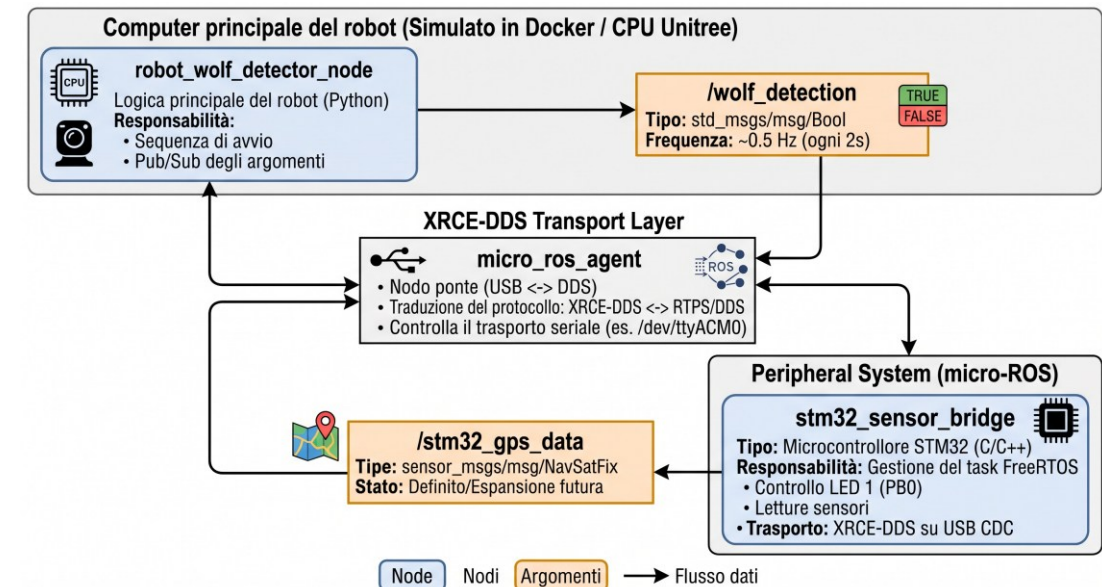
ROS 2

ROS 2 (Robot Operating System) è un middleware open-source che fornisce un livello di astrazione hardware, driver per dispositivi, librerie, strumenti di visualizzazione e un framework per il passaggio di messaggi tra processi.

Il concetto fondamentale alla base di ROS è l'architettura a nodi distribuiti: un nodo è un processo eseguibile che svolge un compito specifico, i nodi comunicano tra loro scambiandosi strutture dati chiamate messaggi attraverso canali nominati definiti Topic, utilizzando un paradigma di tipo *Publisher/Subscriber*.

Il sistema di comunicazione è stato standardizzato utilizzando il protocollo DDS (Data Distribution Service), progettato per reti di computer industriali.

Problema: questa architettura richiede elevate risorse di memoria non disponibili in un sistema embedded come la scheda STM32 descritta in precedenza.



Micro-ROS

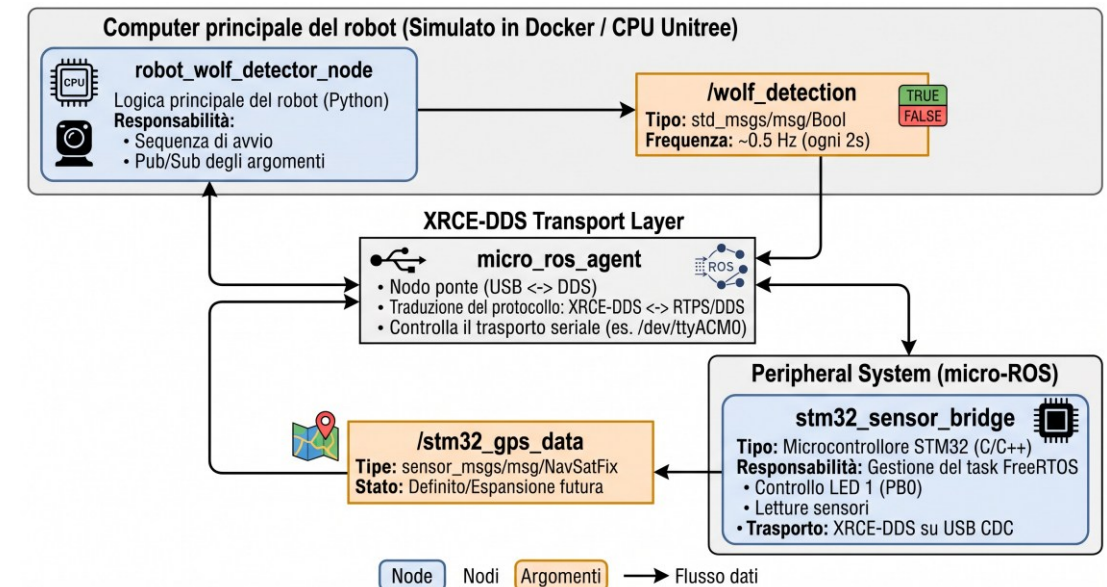
Micro-ROS nasce come estensione ufficiale di ROS 2, concepita specificatamente per colmare il divario tra i computer ad alte prestazioni e i sistemi embedded estremamente vincolati.

Metodo di riduzione delle risorse: il paradigma client-agent

Un microcontrollore STM32 tipico possiede una memoria RAM limitata (da poche decine di KB fino a circa 1-2 MB nei modelli di fascia altissima). Il protocollo DDS standard genera un traffico di rete continuo per mantenere il grafo dei nodi e gestire la Quality of Service (QoS), esaurendo rapidamente la memoria heap di un MCU.

Micro-ROS risolve questo problema delegando il carico computazionale:

- Sulla scheda STM32 viene eseguito solo il micro-ROS Client che comunica tramite un trasporto seriale (UART, USB o UDP/TCP minimale) con un Agent in esecuzione su un computer più potente.
- L'Agent funge da ponte, traducendo i pacchetti minimi del microcontrollore nel formato DDS standard, rendendo la scheda STM32 visibile al resto della rete ROS 2 come se fosse un nodo convenzionale.



ROS 2 e Micro-ROS



Caratteristica	ROS 2 Standard	Micro-ROS
Sistema Operativo (OS)	Richiede un OS completo con standard POSIX (Linux, Windows, macOS).	Eseguito su Real-Time Operating Systems (RTOS come FreeRTOS).
Risorse di Memoria	Ordine dei Gigabyte o centinaia di Megabyte (RAM e Storage).	Ordine dei Kilobyte (10 – 100 KB di RAM).
Protocollo di Rete (Middleware)	Implementazione DDS completa (CycloneDDS). Comunicazione Peer-to-Peer con discovery automatica intensiva.	Utilizza Micro XRCE-DDS (eXtremely Resource Constrained Environments). Architettura basata su Client-Agent.

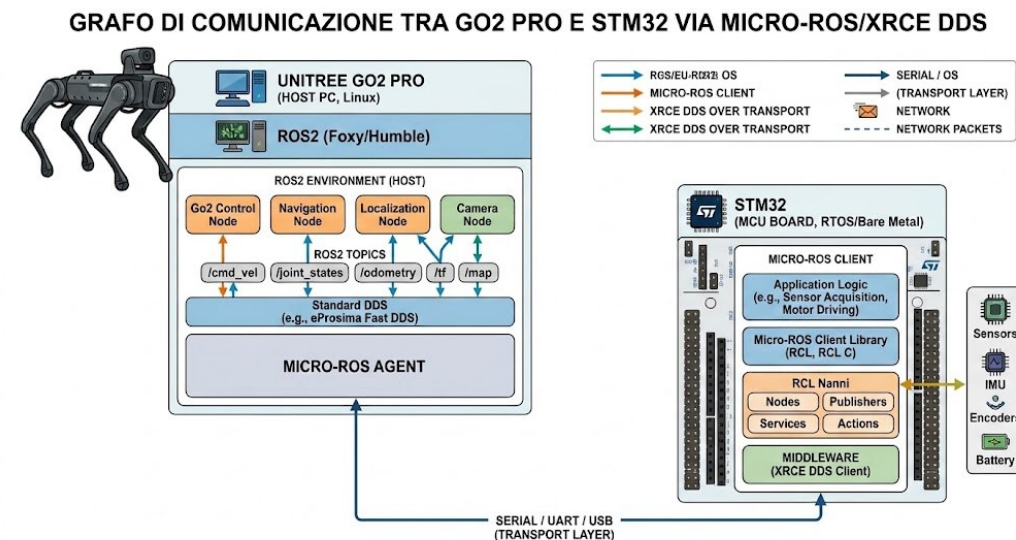
Architettura del protocollo

Il protocollo di comunicazione impiegato tra l'unità di calcolo del robot (su cui risiede il micro-ROS agent) e la scheda STM32 è il **Micro XRCE-DDS**.

Tale protocollo, standardizzato dal consorzio OMG (Object Management Group), è incapsulato a livello di trasporto fisico all'interno di un flusso seriale **USB CDC** (Communication Device Class).

Il contesto di questa architettura deriva dalla necessità di far dialogare un microcontrollore, intrinsecamente privo di memoria sufficiente per ospitare stack di rete complessi, con il protocollo RTPS (Real-Time Publish Subscribe) nativo di ROS 2.

Il Micro XRCE-DDS agisce in modalità architetturale *Client-Server*: la scheda STM32 opera come Client (generando e consumando dati), mentre il robot opera come Agent (ricevendo i frame seriali, decodificandoli e iniettandoli nel grafo ROS 2 globale).



Handshake

Discovery: Il Client STM32 invia un pacchetto logico di PING sul bus USB. Questo processo è governato dalla direttiva `rmw_uros_ping_agent()`. L'esecuzione del firmware viene sospesa finché l'Agent sul robot non risponde con un pacchetto di acknowledgment (PING_ACK), confermando l'apertura del canale trasmissivo e la disponibilità delle risorse.

Creazione della Sessione (Create Client): A seguito della validazione fisica del canale, la scheda trasmette un payload CREATE_CLIENT. Tale pacchetto contiene un identificatore hardware univoco (`client_key`) e definisce il limite massimo della MTU (Maximum Transmission Unit) accettabile per la memoria RAM dell'STM32. L'Agent registra l'hardware e risponde assegnando un `session_id`.

Allocazione delle Entità: Il Client STM32 invia in sequenza i frame necessari per generare i costrutti logici previsti dallo standard ROS 2:

- Creazione del Participant (l'identità del nodo nel dominio DDS).
- Creazione dei Topic (la definizione del canale `wolf_detection` e `stm32_gps_data`).
- Creazione del Subscriber e del DataReader per finalizzare l'ascolto. L'Agent decodifica queste richieste, alloca istanze speculari all'interno del sistema operativo Linux del robot, e restituisce un ack di conferma per ogni singola entità istanziata.

```
// 2. Allocazione e definizione del Nodo
allocator = rcl_get_default_allocator();

// Barriera di sincronizzazione: il sistema si ferma qui finché l'Agent non è attivo
while(rmw_uros_ping_agent(100, 1) != RMW_RET_OK) {
    osDelay(100);
}

// Costruzione delle entità ROS 2 protetta dalla macro RCCHECK
RCCHECK(rclc_support_init(&support, 0, NULL, &allocator));
RCCHECK(rclc_node_init_default(&node, "stm32_sensor_bridge", "", &support));

// 3. Creazione del Subscriber (Ricezione input dal Robot)
RCCHECK(rclc_subscription_init_default(
    &wolf_subscriber,
    &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Bool),
    "wolf_detection"
));

// 4. Creazione del Publisher (Trasmissione dati sensoriali al Robot)
RCCHECK(rclc_publisher_init_default(
    &gps_publisher,
    &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(sensor_msgs, msg, NavSatFix),
    "stm32_gps_data"
));
```

L'intero processo di handshake è protetto dalla macro RCCHECK che si occupa di verificare che ogni operazione eseguita al suo interno termini senza errori. In caso contrario blocca il codice in un costrutto while(1) e un led inizia a lampeggiare ad alta frequenza indicando un errore nell'esecuzione che richiede il reset hardware della scheda.

Inoltre, per evitare differenze di *Endianness*, i messaggi vengono convertiti in array secondo lo standard CDR prima di essere immessi nel cavo USB.

Una volta attraversato il bus USB verso la scheda STM32, i byte in ingresso vengono temporaneamente immagazzinati nel buffer circolare allocato nell'Heap di FreeRTOS durante la chiamata di inizializzazione dell'Executor.

```
size_t custom_transport_read(struct uxrCustomTransport* transport, uint8_t* buf, size_t len, int timeout, uint8_t* err) {
    size_t bytes_read = 0;
    uint32_t start_time = HAL_GetTick();

    // lettura bloccante con limite temporale (timeout)
    while ((bytes_read < len) && ((HAL_GetTick() - start_time) < timeout)) {
        // Verifica la presenza di dati non letti confrontando i puntatori del buffer
        if (rx_head != rx_tail) {
            buf[bytes_read] = rx_buffer[rx_tail];
            rx_tail = (rx_tail + 1) % RX_BUFFER_SIZE;
            bytes_read++;
        } else {
            // Il buffer è vuoto: si attende l'arrivo di nuovi pacchetti USB dall'interrupt
            osDelay(1);
        }
    }

    return bytes_read;
}
```

Il sistema opera seguendo tre fasi di processo principali:

1) **Inizializzazione dell'hardware e del protocollo di rete.**

Una volta avviato, il robot innesca il micro-ROS agent e alimenta la scheda STM32 che a sua volta carica in memoria il kernel FreeRTOS. In seguito il microcontrollore abilita la porta USB interna e la configura secondo gli standard CDC.

La prima fase termina con l'handshake tra i computer interno del robot e microprocessore della scheda (secondo le modalità descritte in precedenza) e l'allocazione dello spazio di memoria per il Publisher del topic `stm32_gps_data` e del Subscriber per il topic `wolf_detection`.

2) **Sequenza di avvio cinematica.**

Conclusa la prima fase, il robot attende 3 secondi prima di iniziare la movimentazione dei motori in modo da allineare i servizi interni di Unitree e per dare il tempo all'utente di allontanarsi.

Utilizzando le funzioni predefinite dello SportClient il robot si alza e si stabilizza sulle 4 zampe, attendendo poi 2 secondi prima di inizializzare la telecamera e il lidar.

3) **Elaborazione dati.**

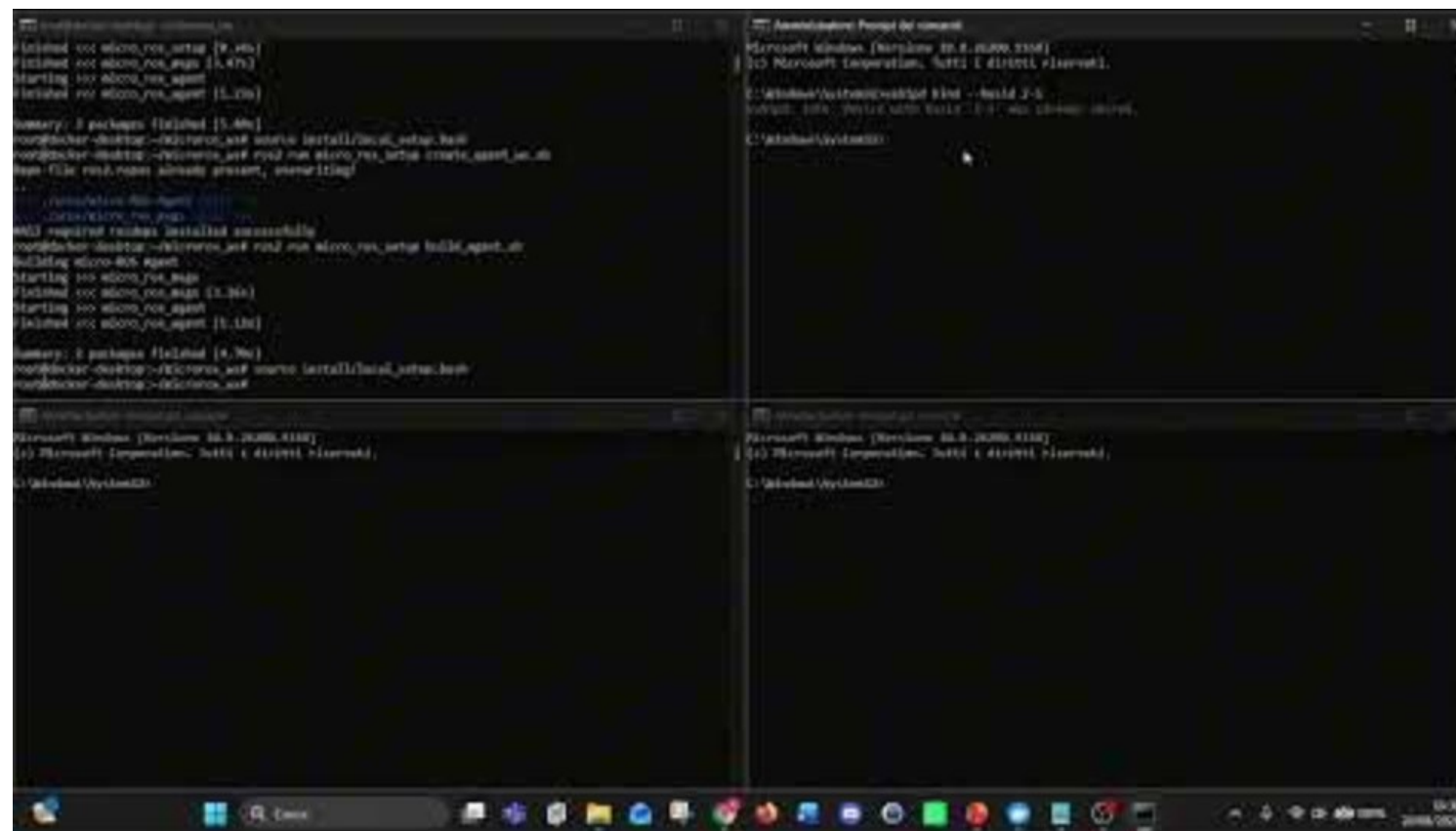
Nella terza fase viene racchiuso il vero e proprio ciclo di lavoro del sistema, ovvero lo scambio dei dati GPS e di presenza del lupo tra il robot e la scheda STM32.

Grazie alle funzioni predefinite dello SportClient il robot salva un frame proveniente dal flusso video della telecamera ogni due secondi: questo frame sarà analizzato da un modello di rete neurale per verificare se nel campo visivo del robot sia presente un lupo (al momento il dato è simulato). Infine il dato booleano contenente l'informazione di presenza viene pubblicato sul topic `wolf_detection`.

Dall'altro lato del sistema, la scheda STM32 ogni 200 ms elabora i dati provenienti dal GPS ad essa collegato e li pubblica sul topic `stm32_gps_data` per renderli disponibili al robot (anche questi dati sono simulati).

Questo scambio dati è mediato dall'architettura agent/client di micro-ROS che traduce i messaggi da standard DDS a XRCE e viceversa.

VIDEO SIMULAZIONE DELLA COMUNICAZIONE



[Link al video YouTube](#)

Il lavoro descritto ha condotto allo sviluppo e alla validazione di un'architettura di comunicazione bidirezionale tra il sistema operativo del robot quadrupede Unitree Go2 Pro e la periferica STM32 NUCLEO-H7A3ZI-Q. L'obiettivo di stabilire un ponte dati funzionale al raggiungimento dell'area di interesse e alla dissuasione dei lupi è stato implementato attraverso le seguenti strategie:

- L'impiego del framework Micro-ROS ha permesso di superare i rigidi vincoli di memoria del microcontrollore. Delegando il carico computazionale del protocollo DDS standard al computer del robot tramite un paradigma Client-Agent, è stato possibile instaurare una comunicazione efficiente basata sullo standard Micro XRCE-DDS.
- La sequenza di avvio è resa deterministica dall'implementazione di un handshake di rete protetto dalla macro diagnostica RCCHECK, la quale previene blocchi di memoria (HardFault) e assicura la corretta allocazione delle entità ROS 2 all'interno del kernel FreeRTOS. Inoltre, il ritardo di 3 secondi imposto all'avvio garantisce la corretta stabilizzazione cinematica del robot prima dell'attivazione dei sensori.
- Il layer di trasporto fisico, operante su bus USB CDC, garantisce la corretta elaborazione dei flussi asincroni. L'architettura software gestisce parallelamente la ricezione delle coordinate GPS ogni 200 ms (sul topic `stm32_gps_data`) e la trasmissione dell'esito dell'analisi visiva ogni 2 s (sul topic `wolf_detection`).

Prospettive e Sviluppi Futuri

Attualmente i dati che vengono scambiati nel sistema sono simulati. I successivi step evolutivi per il completamento del sistema prevedono il collegamento fisico del modulo GPS e del sistema di dissuasione alla scheda STM32, sostituendo i vettori di test con la telemetria reale, nonché la realizzazione di una rete neurale in grado di analizzare i frame catturati dal robot.

BIBLIOGRAFIA



1. Specifiche cane robot Unitree Go2 Pro: <https://www.unitree.com/go2>
2. Dati tecnici STM32 Nucleo-H7A3ZI-Q: <https://www.st.com/en/evaluation-tools/nucleo-h7a3zi-q.html>
3. Basi del protocollo di comunicazione: <https://www.youtube.com/watch?v=phkBljoJKj4>
4. ROS2 e Micro-ROS: https://roscon.ros.org/2019/talks/roscon2019_microros.pdf
<https://www.generationrobots.com/blog/it/ros2-cosa-cambia-rispetto-a-ros/>
<https://micro.vulcanexus.org/>

Grazie