



Università Politecnica delle Marche

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

Realizzazione di un rover con ESP32, Raspberry e comunicazione mediante protocollo MQTT

Corso di Sistemi Operativi Dedicati

Professore

Prof. Daniele Marcozzi

Gruppo 15

Denil Nicolosi

Anno accademico 2022-2023

Indice

1 Introduzione	2
1.1 Detttagli funzionamento	2
2 Componenti hardware	3
2.1 Telaio rover a due motori	3
2.2 Modulo batterie e circuito di protezione	4
2.3 Moduli regolatori di tensione	6
2.4 Modulo per il controllo dei motori L298N	6
2.5 ESP32	7
2.6 Modulo RTC DS3231	8
2.7 Modulo accelerometro MPU6050	9
2.8 Modulo sensore di temperatura DHT11	10
2.9 Modulo ultrasuoni HC-SR04	11
2.10 Raspberry Pi 3B	12
3 Componenti software	12
3.1 Ubuntu	12
3.2 Raspberry Pi OS	14
3.3 FreeRTOS	15
3.4 Node-RED	16
3.5 MySql	17
3.6 Broker MQTT	18
4 Implementazione	20
4.1 Realizzazione dello schema elettrico e assemblaggio del rover	20
4.1.1 Primo piano	20
4.1.2 Secondo piano	22
4.1.3 Terzo piano	24
4.2 Sviluppo su ESP32	26
4.2.1 Task invio dati	29
4.2.2 Task ricezione dati	30
4.2.3 Task modalità di funzionamento	32
4.3 Sviluppo su Raspberry	34
4.3.1 Installazione sistema operativo Raspberry Pi OS	34
4.3.2 Sviluppo software client MQTT	34
4.3.3 Creazione del servizio per l'avvio automatico	36
4.4 Sviluppo su macchina virtuale Ubuntu	37
4.4.1 Creazione macchina virtuale e installazione Ubuntu	37
4.4.2 Installazione broker MQTT	38
4.4.3 Installazione Node-RED	38
4.4.4 Installazione MySQL	39
4.4.5 Sviluppo interfaccia Node-RED	41
5 Conclusioni e implementazioni future	56

1 Introduzione

Il progetto consiste nella realizzazione di un rover con diversi sensori che verrà comandato a distanza da una interfaccia web.

Tra i componenti chiave si trovano un telaio con due motori, un ESP32, una Raspberry Pi 3B, un modulo di controllo dei motori, un modulo RTC, un sensore ultrasuoni, un accelerometro, un sensore di temperatura e umidità. Il rover avrà a bordo sia l'ESP32 sia la Raspberry. La scheda ESP32 avrà il compito di pilotare il rover e acquisire le misure dai vari sensori e comunicarle alla Raspberry mediante comunicazione seriale (connessione USB). Il coordinamento dei vari task che l'ESP32 dovrà svolgere sarà affidato a FreeRTOS. Raspberry Pi svolgerà il compito di client MQTT e dovrà occuparsi della comunicazione dei dati verso il broker MQTT che verrà implementato all'interno di una macchina virtuale Ubuntu 23.10.1. In figura 1 si visualizza lo schema di funzionamento tra i vari dispositivi.

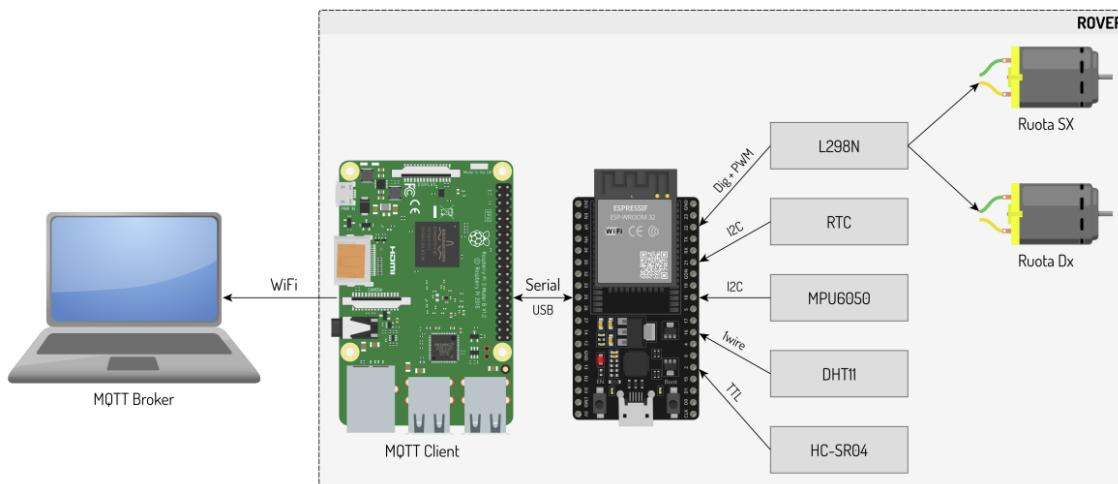


Figura 1: Schema a blocchi

1.1 Dettagli funzionamento

Il rover dovrà avere due modalità di funzionamento:

1. una autonoma in cui “pattuglierà” la zona
2. una manuale in cui la movimentazione sarà definita dai comandi impartiti dall’utente.

La selezione della modalità di funzionamento, così come i comandi di movimentazione, dovranno essere impartiti mediante interfaccia web e comunicati al rover tramite protocollo MQTT.

Indipendentemente dalla modalità di funzionamento, il robot dovrà inviare costantemente la lettura dei vari sensori corredata di timestamp al broker MQTT.

Allo stesso modo il sensore ad ultrasuoni dovrà garantire al rover di non collidere con gli ostacoli presenti nell’ambiente circostante.

La sincronizzazione del modulo RTC dovrà avvenire mediante un comando proveniente dal broker MQTT che, utilizzando il protocollo ntp, avrà accesso all’ora corretta. La sincronizzazione dovrà avvenire all’accensione del sistema e, successivamente, ad intervalli regolari (e.g. ogni ora, una volta al giorno, etc.)

L’interfaccia WEB dovrà garantire le seguenti funzionalità:

- selezione della modalità di funzionamento
- movimentazione del rover (e.g. avanti, indietro, sinistra, destra)

- visualizzazione, in tempo reale, dei dati provenienti dai sensori a bordo del rover.

Inoltre i dati provenienti dai sensori verranno salvati all'interno di un database. La comunicazione MQTT verrà gestita in modo che i dati da salvare all'interno del database non vengano persi a causa di una disconnessione (implementazione della comunicazione con un QoS almeno pari a 1).

2 Componenti hardware

Il sistema utilizzato è composto dai seguenti componenti:

- Telaio rover a due motori
- Modulo batterie e circuito di protezione
- Moduli regolatori di tensione
- Modulo per il controllo dei motori L298N
- ESP32
- Modulo RTC DS3231
- Modulo accelerometro MPU6050
- Modulo sensore di temperatura DHT11
- Modulo ultrasuoni HC-SR04
- Raspberry Pi 3B

2.1 Telaio rover a due motori

Il telaio utilizzato per il rover è un kit base per la realizzazione di piccole unità a due ruote motrici e una rotella girevole. La base è formata da una o più piastre di materiale acrilico trasparente dalla dimensioni di 135mm (L), 75mm (W), 2.7mm di spessore. In figura 2 si visualizza il telaio assemblato con una sola piastra.

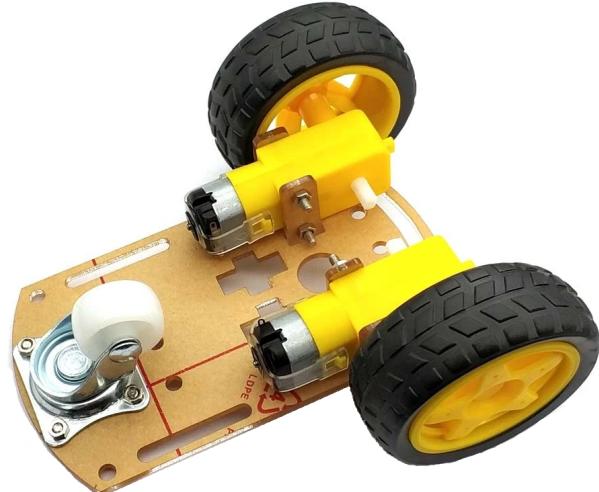


Figura 2: Telaio rover

In questo caso, sono state utilizzate 3 piastre, con 4 barre filettate M4 da 100 mm per distanziare ad altezze regolabili ciascun piano.

Il kit contiene due piccoli motori DC a magneti permanenti, spazzole di carbone, equivalenti al tipo TGP01D-A130. Il rapporto di riduzione del riduttore con ingranaggi di plastica con uscita biassiale è 1:48. Il motore presenta le seguenti specifiche tecniche:

- Gamma di tensione di funzionamento: 3 ÷ 12 V
- Tensione nominale: 3 V
- Velocità a vuoto: @ 3 V: 110 rpm
- Corrente a vuoto: @ 3 V: 120 mA (0,12 A)
- Corrente di stallo: @ 3 V 450 mA (0,45 A)
- Coppia di stallo: 0.27 kgf·cm (0.027 N·m)
- Dimensioni: 69,2 x 22,5 x 18,8 mm
- Peso circa: 35 g

2.2 Modulo batterie e circuito di protezione

Per alimentare il rover, è stato realizzato un pacco batterie formato da 3 celle al litio chiamate 18650, ciascuna con le seguenti caratteristiche:

- Tipologia: Lithium-ion
- Tensione di uscita nominale: 3,7V
- Capacità nominale: 3000 mAh
- Tensione di Ricarica: 4,2v DC
- Corrente di carica standard: 480 mA
- Peso: 49 g
- Dimensioni: 65x18mm

Per poter essere utilizzate, sono state inserite in un case apposito (mostrato in figura 3) in modo da poterle sostituire agevolmente.



Figura 3: Pacco batterie

Per ottenere una tensione adeguata al tipo di utilizzo, le tre celle vengono messe in serie, ottenendo una tensione nominale di 11,1v. Per fare questo, viene utilizzato un circuito BMS di protezione, che svolge diverse funzionalità:

- Controllo bilanciamento pacco batteria: quando una qualsiasi cella del pacco batteria supera la tensione di bilanciamento iniziale, il BMS provvede a redistribuire la carica in eccesso tra le varie celle (bilanciamento attivo) oppure utilizzando apposite resistenze (bilanciamento passivo).
- Protezione tra cui sovraccarico, sovraccarico, sovracorrente, sovratensione, protezione da corto circuito.

In particolare il circuito utilizzato, mostrato in figura 4, ha le seguenti caratteristiche:

- Tensione di carica: 12,6-13 V
- Corrente di uscita massima: 20 A
- Potenza di uscita massima: 252 W
- Dimensioni: 59 x 20 mm.

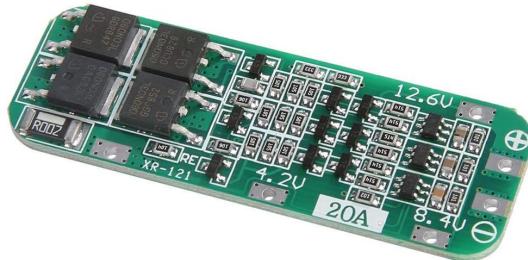


Figura 4: Modulo BMS

Il collegamento delle celle con il circuito BMS è stato realizzato come mostrato in figura 5.

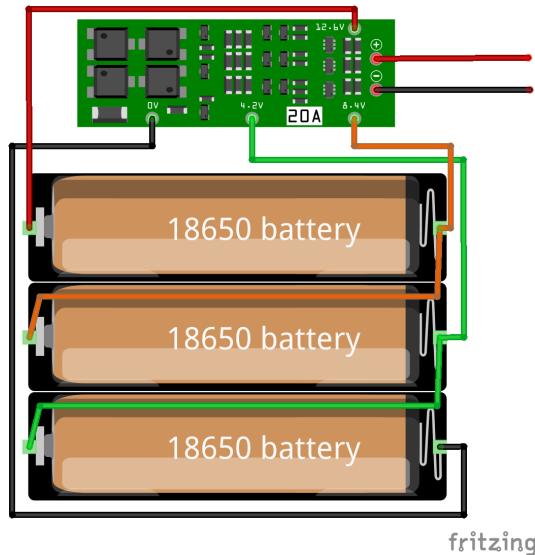


Figura 5: Schema collegamento batteria e BMS

2.3 Moduli regolatori di tensione

Per alimentare i vari dispositivi del progetto (come i sensori, l'ESP32 e la Raspberry) è necessario un dispositivo che regoli e stabilizzi la tensione in ingresso dalle batterie (che è variabile in base allo stato di carica di quest'ultime). In particolare, sono stati utilizzati due regolatori di tensione:

- Un modulo LM2596S, illustrato in figura 6, con range di tensione di ingresso DC da 3.2V a 40V e tensione di uscita DC da 1.25V a 35V, regolabile tramite apposito potenziometro a vite. La corrente massima di uscita è di 3A. In questo caso, è stato regolato su un'uscita di 5v, per l'alimentazione del modulo HC-SR04 e l'ESP32 mediante pin "VIN".



Figura 6: Modulo LM2596S

- Un modulo step-down senza denominazione, illustrato in figura 7, con range di tensione di ingresso DC da 6v a 24v e tensione di uscita fissa a 5v tramite porta usb. La corrente massima di uscita è di 3A. Questo modulo si presta perfettamente per alimentare la Raspberry tramite cavo usb.



Figura 7: Modulo step-down usb

2.4 Modulo per il controllo dei motori L298N

Questo modulo è basato sul driver Dual H-Bridge L298N e permette di pilotare con semplicità due motori DC o un motore passo-passo bipolare. Per ogni motore vi sono a disposizione tre pin di input: EN, IN1, IN2. Il pin EN (enable) attiva l'uscita relativa al motore (se controllato in pwm, è possibile regolare la velocità del motore). I pin IN1 e IN2 vengono utilizzati per controllare il verso di rotazione del motore, dando il segnale logico alto ad uno dei due. Le principali caratteristiche tecniche sono:

- Tensione operativa: 6-32 V
- Tensione logica: 5V
- Assorbimento di picco per canale: fino a 2 A
- Potenza massima erogabile: 25 W
- Dimensioni: 43 x 43 x 28mm
- Peso: 25g



Figura 8: Modulo L298N

2.5 ESP32

L'ESP32 è una serie di microcontroller SoC (System-on-Chip) a basso costo e a basso consumo con Wi-Fi e Bluetooth dual-mode integrati creata da Espressif Systems, come successore del popolare ESP8266.

Il processore è basato su un SoC dual-core Xtensa, funziona a 32 bit e ha una frequenza di clock fino a 240 MHz. Il modulo contiene 4 MB di flash e i suoi 38 pin sono disposti in modo da ridurne al minimo le dimensioni. L'assorbimento energetico in stand-by è inferiore a 5 µA, rendendolo adatto ad integrazione in progetti wearable alimentati a batteria. ESP32 supporta velocità di trasferimento dati fino a 150 Mbps, ed una potenza di uscita in antenna di 20.5 dBm che assicura una completa compatibilità con tutti i device. Un'altra caratteristica interessante dell'ESP32 è la sua capacità di funzionare come un dispositivo dual-mode, ovvero può essere utilizzato sia come un dispositivo autonomo che come un modulo slave in un sistema più grande. Ciò lo rende particolarmente utile per l'Internet of Things (IoT), poiché consente di creare dispositivi connessi in modo semplice e conveniente.

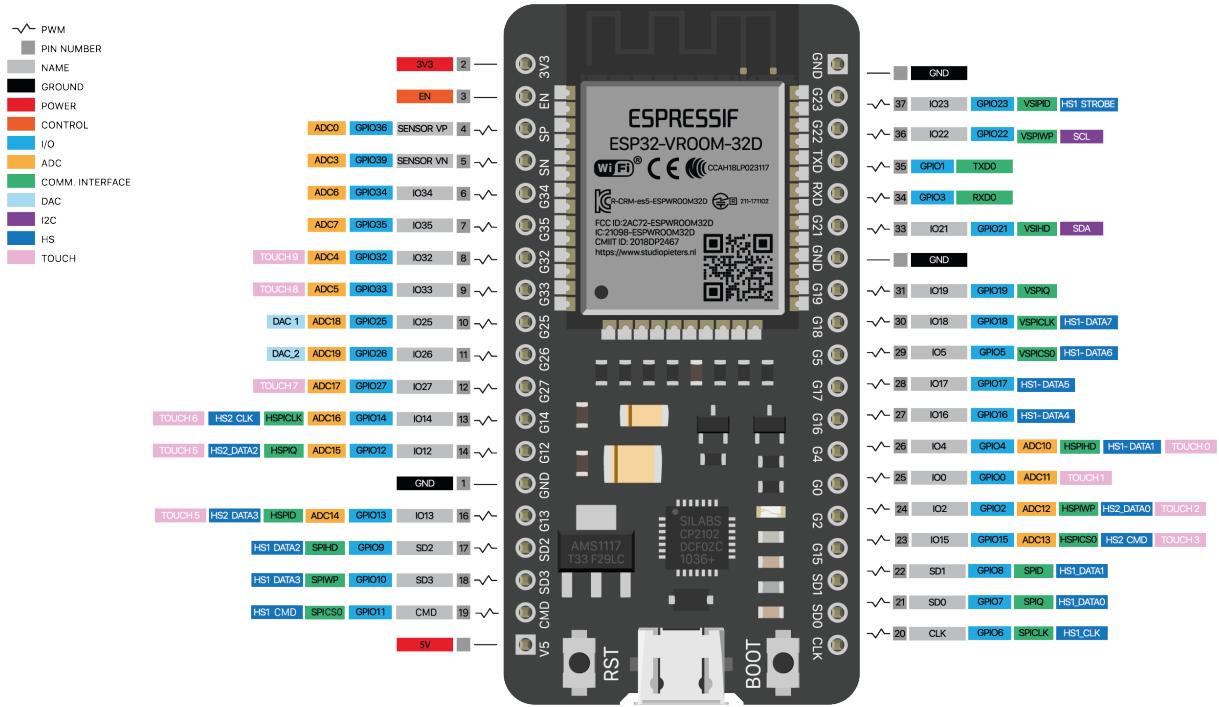
In questo caso, l'ESP32 viene utilizzato all'interno di un DevKit, ovvero una board che contiene ed estende le funzionalità del chip (come regolatore di tensione, led built-in, controller usb e altro ancora).

Nella figura 9 si può osservare l'ESP32 nel DevKit.



Figura 9: ESP32 nel DevKit

Per utilizzare i pin, si fa riferimento al pinout del DevKit in figura 10.



funzione del giorno e dell'ora .Il DS3231 ha un circuito con un comparatore di precisione che controlla lo stato dell'alimentazione e provvede a commutare sull'alimentazione a batteria quando l'alimentazione principale viene a mancare. Indicato in tulle le applicazioni dove è necessario avere un preciso riferimento temporale. Presenta le seguenti specifiche tecniche:

- Alimentazione: da 3,3 a 5,5 volt
- Precisione orologio (da 0°C a +40°C): 2 ppm
- Uscita a onda quadra: 32.768 Hz
- EEPROM: AT24C32 (capacità di memoria 32Kb)
- Interfaccia: I2C @400 kHz
- Dimensioni (mm): 38x22x14
- Peso: 8 g

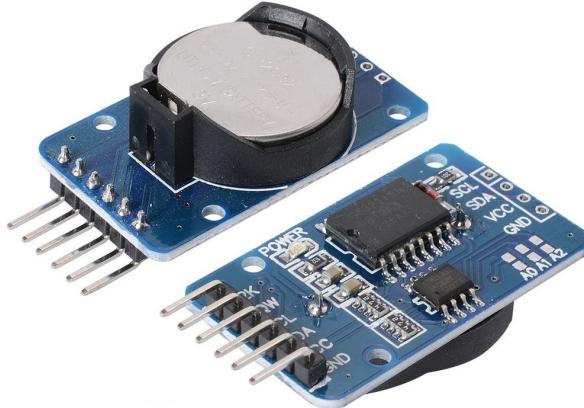


Figura 11: Modulo RTC DS3231

Come accennato, questo modulo utilizza per la comunicazione il protocollo I2C. Il protocollo permette la comunicazione di dati tra due o più dispositivi I2C utilizzando un bus (canale di comunicazione) a due fili, più uno per il riferimento comune di tensione; L'I2C è progettato per consentire la comunicazione tra dispositivi utilizzando solo due linee di segnale: SDA (Serial Data Line) e SCL (Serial Clock Line). SDA è la linea bidirezionale utilizzata per il trasferimento effettivo dei dati, mentre SCL è la linea di clock generata dal dispositivo master per sincronizzare la comunicazione tra i dispositivi collegati. Nel bus I2C, possono essere collegati più dispositivi utilizzando un'architettura a bus multi-master, il che significa che più dispositivi possono fungere da master e iniziano la comunicazione in momenti diversi. Tuttavia, nella maggior parte delle applicazioni, viene utilizzato un solo dispositivo master (come un microcontrollore) che controlla l'intera comunicazione nel bus I2C.

2.7 Modulo accelerometro MPU6050

Il modulo accelerometro MPU6050 è un dispositivo di rilevamento inerziale che combina un accelerometro triassiale e un giroscopio triassiale all'interno di un singolo chip. Questo componente, sviluppato da InvenSense, offre la capacità di misurare accelerazioni lineari lungo tre assi e velocità angolari attorno ai medesimi assi. La presenza di entrambi gli accelerometri e i giroscopi all'interno di un unico package consente una misurazione simultanea e precisa dei movimenti tridimensionali del dispositivo a cui è applicato.

L'accelerometro, in particolare, fornisce informazioni sulla forza gravitazionale e sulle accelerazioni lineari,

mentre il giroscopio misura la velocità angolare. Questi dati combinati consentono una stima accurata delle variazioni di orientamento e movimento nello spazio tridimensionale.

Il MPU6050 è noto per la sua precisione e stabilità, rendendolo un componente ideale per applicazioni che richiedono il monitoraggio accurato del movimento. È spesso impiegato in progetti di robotica, droni, veicoli autonomi e dispositivi indossabili, grazie alla sua capacità di fornire dati attendibili sulla dinamica del movimento.

Anche questo modulo, comunica grazie al bus I2C illustrato nel paragrafo precedente. Nel contesto del progetto, il modulo MPU6050 sarà utilizzato per acquisire informazioni sulle accelerazioni e le velocità angolari del rover.

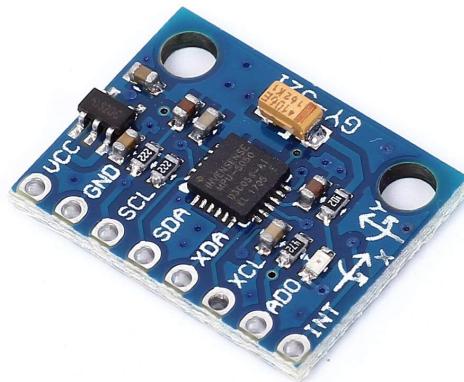


Figura 12: Modulo MPU6050

2.8 Modulo sensore di temperatura DHT11

Il sensore di temperatura e umidità DHT11 è un componente sensore economico e ampiamente utilizzato progettato per misurare sia la temperatura che l'umidità relativa nell'ambiente circostante. Prodotto da Aosong Electronics, il DHT11 è apprezzato per la sua semplicità d'uso e la precisione accettabile, rendendolo ideale per una vasta gamma di applicazioni.

Il DHT11 incorpora un sensore di umidità resistivo e un termistore per la rilevazione della temperatura. Utilizza un segnale digitale per comunicare i dati misurati al microcontrollore o al processore con cui è connesso. La sua interfaccia semplice lo rende facilmente integrabile in progetti elettronici, e la sua bassa richiesta di risorse lo rende adatto anche per sistemi con limitate capacità computazionali.

Tuttavia, è importante notare che, nonostante la sua praticità, il DHT11 potrebbe non essere adatto per applicazioni che richiedono una precisione estrema.

La comunicazione con il sensore avviene utilizzando una connessione seriale che utilizza un solo filo (Single-Wire Two-Way). Il pacchetto informativo che include i dati di temperatura ed umidità inviati dal sensore ha una lunghezza di 40 bit ed una durata di 4 ms. Nel contesto del progetto, il sensore DHT11 sarà utilizzato per rilevare la temperatura e l'umidità ambientale.

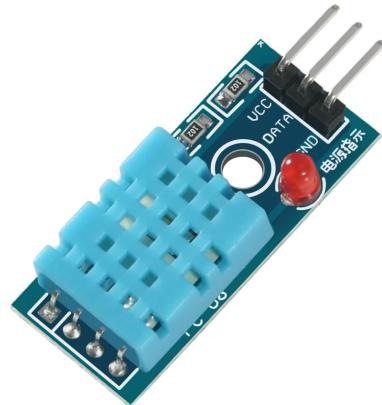


Figura 13: Modulo DHT11

2.9 Modulo ultrasuoni HC-SR04

Il modulo ultrasuoni HC-SR04 è un dispositivo di rilevamento della distanza ampiamente utilizzato, noto per la sua semplicità d'uso ed efficienza nella misurazione delle distanze in modo non invasivo. Questo modulo sfrutta il principio della propagazione degli ultrasuoni per calcolare la distanza tra il sensore e un oggetto nel suo campo di rilevamento.

Il modulo HC-SR04 è composto da un trasmettitore ultrasuoni e un ricevitore, e opera secondo il principio del “tempo di volo”. Il trasmettitore emette brevi impulsi ultrasonici, che viaggiano nello spazio e vengono riflessi da un oggetto circostante. Il ricevitore quindi rileva il segnale riflessi, e la differenza di tempo tra l'invio e la ricezione di questi impulsi viene utilizzata per calcolare la distanza tra il modulo e l'oggetto. Uno dei vantaggi principali del modulo HC-SR04 è la sua facilità di integrazione. Richiede solamente quattro pin (due per l'alimentazione e due per la trasmissione e la ricezione dei segnali ultrasonici) e può essere interfacciato direttamente con i microcontrollori.

La sua applicazione è diffusa in progetti che richiedono la misurazione della distanza in tempo reale, come ad esempio nell'evitamento degli ostacoli per veicoli autonomi o robot a guida autonoma.

Nel contesto del nostro progetto, il modulo ultrasuoni HC-SR04 sarà utilizzato per rilevare la presenza di ostacoli nell'ambiente circostante al rover. Ciò consentirà al rover di evitare collisioni.



Figura 14: Modulo HC-SR04

2.10 Raspberry Pi 3B

La Raspberry Pi 3B è una scheda computer single-board (SBC) sviluppata dalla Raspberry Pi Foundation. Questa scheda è un esempio paradigmatico di computer a basso costo e ad alte prestazioni, ideato per favorire l'apprendimento della programmazione e l'accesso alla tecnologia informatica.

Il cuore della Raspberry Pi 3B è un processore ARM Cortex-A53 quad-core con frequenza di clock di 1.2 GHz, affiancato da 1 GB di memoria RAM. Questa combinazione offre prestazioni notevoli rispetto alle precedenti iterazioni della Raspberry Pi, rendendo la scheda più adatta per una vasta gamma di applicazioni, tra cui progetti IoT, server leggeri, e centri multimediali.

Uno dei punti di forza della Raspberry Pi 3B è la sua ampia connettività. La scheda è dotata di porte USB, HDMI, e Ethernet, offrendo così la flessibilità di connettersi a diversi dispositivi e reti. La presenza di un modulo Wi-Fi integrato e di Bluetooth la rende particolarmente adatta per applicazioni senza fili e l'interfacciamento con dispositivi esterni.

Inoltre, la Raspberry Pi 3B dispone di un connettore GPIO (General Purpose Input/Output) che consente l'interfacciamento con sensori, attuatori e altri componenti elettronici. Questa caratteristica la rende una scelta popolare per progetti di prototipazione e sperimentazione hardware.

La versatilità della Raspberry Pi 3B è evidente nella sua capacità di fungere da server, stazione di lavoro o centro multimediale. È supportata da una vasta comunità di sviluppatori e offre una vasta gamma di risorse e documentazione online, facilitando l'apprendimento e lo sviluppo di progetti personalizzati.

Nel contesto del progetto, la Raspberry Pi 3B sarà utilizzata come client MQTT e controller di comunicazione per il rover.

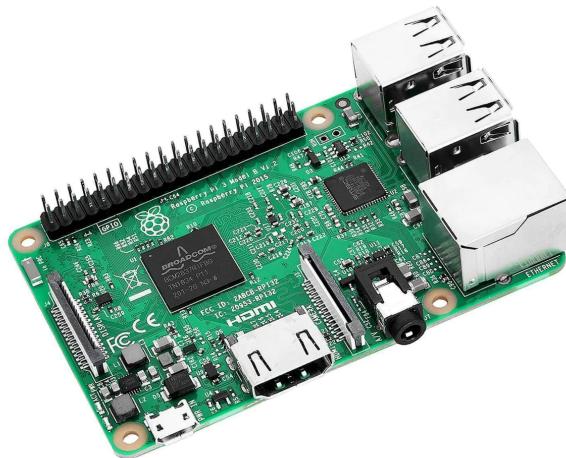


Figura 15: Raspberry Pi 3B

3 Componenti software

3.1 Ubuntu

Ubuntu è una distribuzione di sistema operativo basata su Linux, nata con l'obiettivo di fornire un'esperienza informatica completa, potente e accessibile a un vasto pubblico di utenti. Fondato sulla filosofia Debian, Ubuntu è sviluppato e mantenuto dalla società Canonical Ltd., e ha rapidamente guadagnato una vasta

popolarità grazie alla sua combinazione di stabilità, facilità d'uso e un forte focus sulla comunità open-source.

Al cuore di Ubuntu c'è il kernel Linux, un componente fondamentale che offre al sistema operativo la robustezza e la flessibilità necessarie per supportare una vasta gamma di hardware e applicazioni. La filosofia open-source di Ubuntu si estende ben oltre il kernel, abbracciando ogni componente del sistema operativo. Ciò significa che il codice sorgente è liberamente accessibile, modificabile e condivisibile, incoraggiando la collaborazione e l'innovazione.

Ubuntu offre un ambiente desktop che può essere personalizzato in base alle preferenze dell'utente. Attualmente, la versione più comune di Ubuntu utilizza l'ambiente desktop GNOME per fornire un'interfaccia pulita e intuitiva. Tuttavia, gli utenti hanno la possibilità di scegliere tra diversi desktop environments, come KDE, XFCE o LXQt, per adattare l'aspetto e il comportamento del sistema alle loro esigenze specifiche.

Il sistema di gestione dei pacchetti APT (Advanced Package Tool) semplifica notevolmente la gestione del software su Ubuntu. Gli utenti possono facilmente installare, aggiornare o rimuovere pacchetti software utilizzando comandi intuitivi come apt-get o interfacce grafiche come il Software Center. Questo rende il processo di gestione del software efficiente e accessibile anche per gli utenti meno esperti.

Ubuntu segue un ciclo di rilascio regolare, con nuove versioni programmate ogni sei mesi. Le versioni LTS (Long Term Support) sono particolarmente significative in quanto garantiscono un supporto esteso per cinque anni, offrendo una scelta stabile per gli utenti e le organizzazioni che desiderano evitare aggiornamenti frequenti.

La comunità di Ubuntu è notoriamente attiva e inclusiva. Gli utenti possono accedere a forum, documentazione e risorse online per ottenere supporto, condividere esperienze e partecipare a progetti open-source correlati. Ciò crea un ambiente collaborativo in cui sia gli utenti principianti che gli sviluppatori esperti possono trovare risposte e condividere conoscenze.

Ubuntu offre un insieme di applicazioni e software preinstallato, tra cui un browser web, suite di produttività (come LibreOffice), strumenti di sviluppo e molto altro. La vasta selezione di software disponibile attraverso il sistema di gestione dei pacchetti rende facile ampliare le funzionalità del sistema secondo le necessità.

Nel contesto del progetto, l'utilizzo di Ubuntu all'interno di una macchina virtuale svolge un ruolo cruciale, fornendo un ambiente affidabile per l'esecuzione di servizi essenziali come MQTT broker, database MySQL e l'interfaccia web con Node-RED. La combinazione di stabilità, flessibilità e un solido supporto della comunità rende Ubuntu una scelta eccellente per sostenere le operazioni critiche in un progetto IoT.

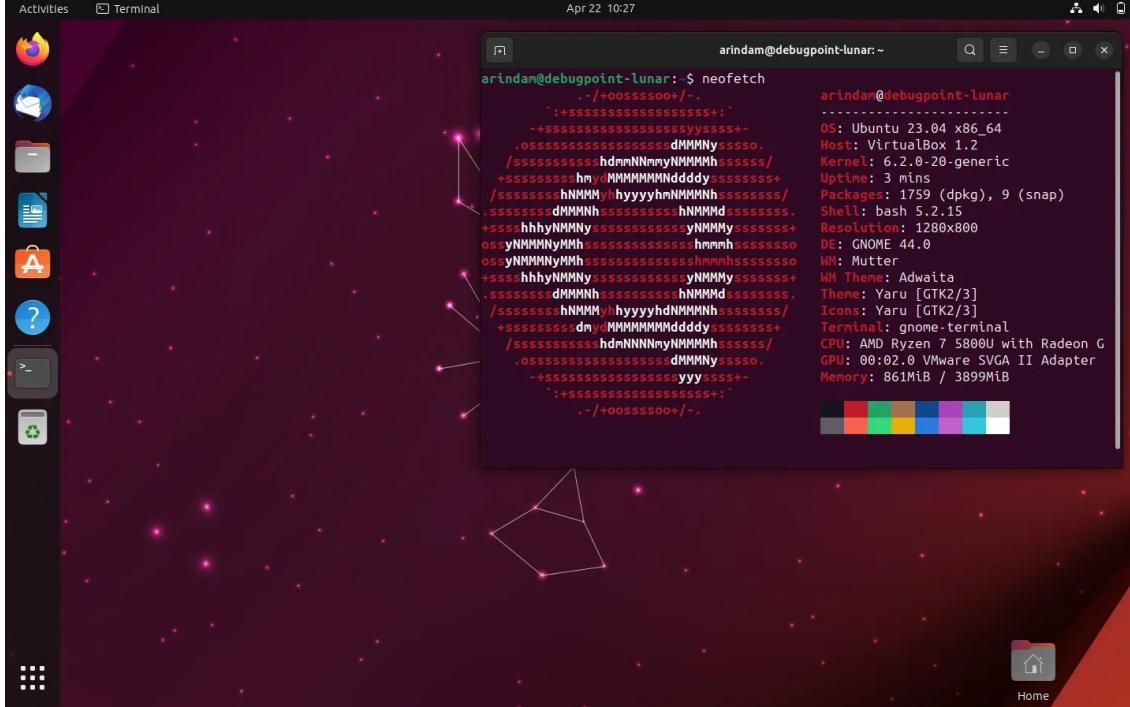


Figura 16: Ambiente desktop Ubuntu 23 con GNOME

3.2 Raspberry Pi OS

Raspberry Pi OS, precedentemente noto come Raspbian, è un sistema operativo specializzato sviluppato appositamente per le schede Raspberry Pi [1]. Questo sistema operativo è stato progettato per sfruttare al meglio le caratteristiche hardware specifiche di Raspberry Pi, fornendo un ambiente operativo stabile e ottimizzato per le applicazioni embedded e l'apprendimento della programmazione.

Raspberry Pi OS si basa sul kernel Linux e include una serie di ottimizzazioni specifiche per sfruttare appieno le risorse hardware delle schede Raspberry Pi. Ciò assicura prestazioni efficienti e stabili, nonché la compatibilità con un'ampia varietà di periferiche e componenti hardware.

Raspberry Pi OS presenta di default il desktop environment PIXEL (Pi Improved Xwindows Environment, Lightweight), progettato appositamente per offrire un'esperienza desktop reattiva e leggera sulle risorse limitate delle Raspberry Pi. PIXEL è caratterizzato da un'interfaccia pulita e intuitiva, ideale per progetti e utilizzi educativi.

Raspberry Pi OS include un set di software preinstallato che copre una gamma di funzionalità, tra cui un browser web, editor di testo, strumenti di programmazione e software per la produttività. Inoltre, grazie alla crescente popolarità di Raspberry Pi, è possibile accedere a un vasto ecosistema di applicazioni e librerie compatibili.

Anche Raspberry Pi OS, come Ubuntu, utilizza il sistema di gestione dei pacchetti APT, semplificando notevolmente il processo di installazione, aggiornamento e rimozione del software. Questo approccio rende facile e accessibile la gestione delle applicazioni e degli strumenti del sistema.

Data la popolarità delle Raspberry Pi, Raspberry Pi OS gode di una comunità attiva di utenti e sviluppatori. Forum online, documentazione e risorse didattiche offrono supporto e soluzioni a una vasta gamma di domande e problemi.

La flessibilità di Raspberry Pi OS permette di adattare il sistema operativo alle specifiche esigenze di progetti personalizzati. La presenza del terminale e la possibilità di personalizzare il sistema forniscono agli sviluppatori un ambiente ricco di possibilità.

Nel contesto del progetto, l'impiego di Raspberry Pi OS come sistema operativo per la Raspberry Pi è fondamentale per sfruttare appieno il potenziale di questa scheda. Esso consente lo sviluppo e l'esecuzione del codice utilizzato per la comunicazione MQTT fra l'ESP32 e la macchina virtuale Ubuntu.

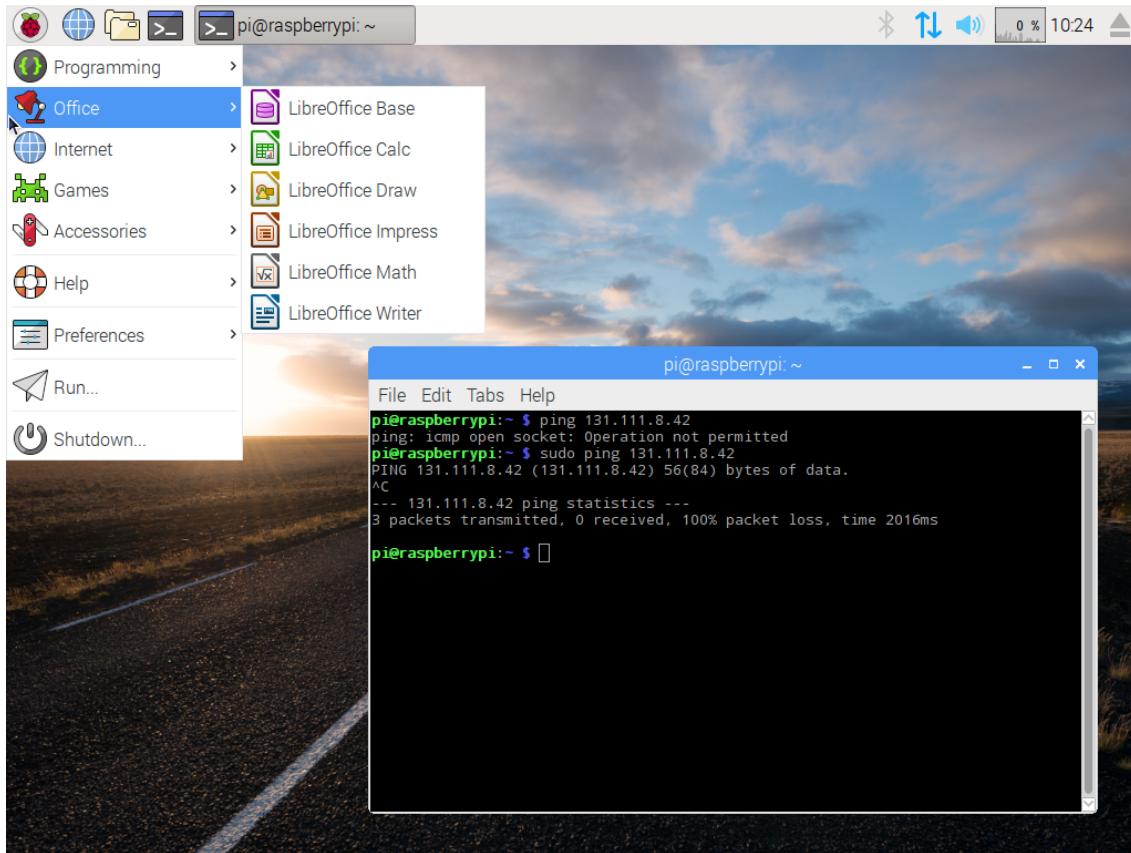


Figura 17: Ambiente desktop Raspberry Pi OS

3.3 FreeRTOS

FreeRTOS, acronimo di “Free Real-Time Operating System”, è un sistema operativo open-source progettato specificamente per sistemi embedded in tempo reale [2]. L'ESP32, un microcontrollore potente e versatile, sfrutta le caratteristiche di FreeRTOS per coordinare e gestire in modo efficiente le attività concorrenti all'interno del sistema.

FreeRTOS offre un kernel in tempo reale, il che significa che è progettato per garantire risposte predeterminate e deterministiche alle interruzioni e agli eventi temporizzati. Ciò lo rende particolarmente adatto per applicazioni in cui la gestione del tempo è cruciale, come nel caso del controllo dei motori e della raccolta dati simultanea da sensori sul rover.

FreeRTOS supporta il concetto di multithreading, consentendo l'esecuzione simultanea di più attività. Queste attività, chiamate “tasks” (task), possono essere assegnate a compiti specifici, come la gestione dei motori, l'acquisizione dati dai sensori e la comunicazione seriale con altri componenti del sistema.

Per garantire la corretta sincronizzazione delle attività e la condivisione sicura delle risorse, FreeRTOS fornisce meccanismi come semafori, mutex e code di messaggi. Questi strumenti consentono una gestione robusta e affidabile delle risorse condivise all'interno del sistema.

FreeRTOS include uno scheduler che assegna priorità alle attività in base alle loro esigenze temporali. Ciò consente di assegnare priorità più elevate a compiti critici in modo che siano eseguiti prima, garantendo una risposta rapida alle richieste del sistema.

FreeRTOS è progettato per essere altamente portabile, il che significa che può essere facilmente adattato per funzionare su diverse architetture di microcontrollore, inclusa l'architettura Xtensa utilizzata nell'ESP32. Questa portabilità è un vantaggio chiave che consente agli sviluppatori di utilizzare FreeRTOS su una varietà di dispositivi embedded.

FreeRTOS è supportato da una comunità attiva di sviluppatori e utenti, che contribuiscono all'evoluzione continua del sistema operativo. Gli sviluppatori possono beneficiare di una vasta documentazione, forum online e risorse educative per sfruttare al massimo le funzionalità di FreeRTOS.

Nel contesto del progetto, l'ESP32 sfrutta FreeRTOS per coordinare le attività del rover in modo efficiente e deterministico. La combinazione di un kernel in tempo reale, gestione delle attività multithreading e strumenti di sincronizzazione rende FreeRTOS un'ottima scelta per garantire un funzionamento affidabile del rover in modalità autonoma e durante l'interazione con il sistema di controllo remoto.



Figura 18: Logo FreeRTOS

3.4 Node-RED

Node-RED è una piattaforma di sviluppo visuale basata su browser che sfrutta il runtime Node.js per semplificare la creazione di interfacce IoT intuitive e l'orchestrazione di flussi di dati [3]. Utilizzando una interfaccia a flusso drag-and-drop, Node-RED consente agli sviluppatori di creare facilmente applicazioni IoT, integrare dispositivi hardware e orchestrare comunicazioni complesse. Caratteristiche principali:

- Runtime Node.js: Node-RED è costruito su Node.js, un runtime JavaScript server-side che consente l'esecuzione di codice JavaScript lato server. Questa scelta architettonica offre flessibilità e scalabilità nello sviluppo di applicazioni IoT e di automazione.
- Interfaccia Visuale Intuitiva: L'interfaccia grafica di Node-RED permette agli sviluppatori di costruire flussi di lavoro visivi attraverso un'interfaccia drag-and-drop, semplificando la progettazione e la gestione delle logiche applicative.
- Nodi Predefiniti e Personalizzati: Node-RED fornisce una vasta libreria di nodi predefiniti per eseguire funzioni comuni, come l'interazione con MQTT, l'analisi di dati JSON e altro ancora. Inoltre, è possibile creare nodi personalizzati per adattare il sistema alle esigenze specifiche del progetto.
- Integrazione di Protocollo MQTT: Grazie ai nodi predefiniti dedicati a MQTT, Node-RED semplifica l'integrazione del protocollo MQTT nei flussi di lavoro, facilitando la comunicazione tra dispositivi IoT, applicazioni e servizi cloud.

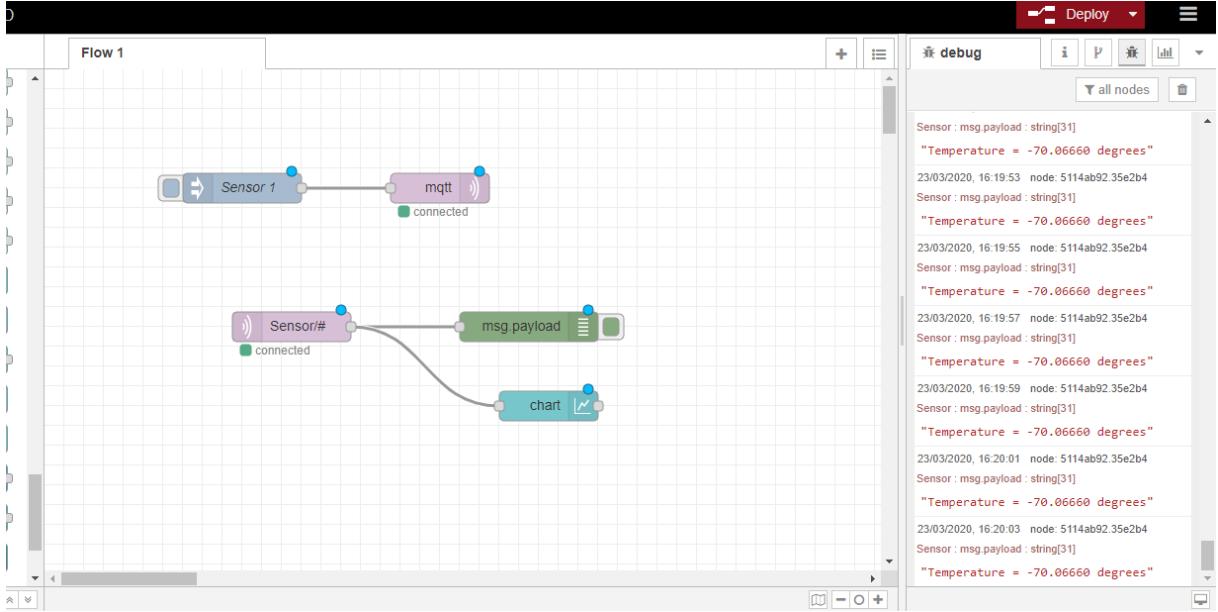


Figura 19: Interfaccia di sviluppo Node-RED

Utilizzo pratico nel progetto:

- Sviluppo del Flusso MQTT: creazione di flussi di lavoro dinamici per la comunicazione bidirezionale tra il rover e l'interfaccia web, utilizzando i nodi MQTT di Node-RED per pubblicare e sottoscrivere a topic.
- Progettazione dell'Interfaccia Web: Node-RED permette la progettazione di interfacce web interattive utilizzando nodi HTML, CSS e JavaScript integrati. Queste interfacce visualizzano dati in tempo reale e consentono all'utente di impartire comandi al rover.
- Gestione degli Eventi: Node-RED può gestire eventi in tempo reale come nuovi dati dai sensori o comandi MQTT in arrivo. Si possono definire logiche specifiche basate su questi eventi per adattare dinamicamente il comportamento dell'interfaccia web.

3.5 MySql

MySQL è un sistema di gestione di database relazionali open-source ampiamente utilizzato per la gestione efficiente e sicura di dati strutturati [4]. Con il suo approccio robusto e la comprovata affidabilità, MySQL è una scelta popolare per applicazioni di database in una vasta gamma di settori, inclusa l'Internet of Things (IoT). Caratteristiche principali:

- Struttura Relazionale: MySQL adotta un modello di database relazionale, organizzando i dati in tabelle con relazioni definite. Questa struttura facilita la gestione di dati complessi e fornisce un framework flessibile per le query.
- Affidabilità e Stabilità: MySQL è noto per la sua stabilità e affidabilità, con una lunga storia di utilizzo in applicazioni critiche. La gestione transazionale, la concorrenza e le funzionalità di backup contribuiscono a garantire l'integrità dei dati.
- Open Source e Comunità Attiva: Essendo un progetto open-source, MySQL beneficia di una vasta comunità di sviluppatori e utenti che contribuiscono a migliorare costantemente il sistema. Ciò si traduce in aggiornamenti frequenti, correzioni di bug e una ricca documentazione.

- Supporto per il Linguaggio SQL: MySQL supporta il linguaggio SQL (Structured Query Language), il che semplifica la creazione, l'interrogazione e la gestione dei dati nel database. Gli sviluppatori possono sfruttare query SQL per accedere, modificare e recuperare informazioni in modo efficiente.

Utilizzo pratico nel progetto:

- Archiviazione dei dati dei sensori: MySQL sarà utilizzato per archiviare i dati provenienti dai sensori del rover. I dati saranno memorizzati in una tabella relazionale, consentendo una gestione organizzata e strutturata dei dati.
- Integrazione con Node-RED: Node-RED, utilizzando appositi nodi, si interfaccia con il database MySQL per l'inserimento e il recupero di dati. Ciò permette di conservare e recuperare storicamente i dati raccolti dai sensori del rover.
- Gestione dell'Accesso: MySQL consente la definizione di utenti e privilegi, consentendo un controllo preciso dell'accesso ai dati. Questo è particolarmente importante in scenari in cui la sicurezza e la privacy dei dati sono prioritari.

La combinazione di MySQL, Node-RED e MQTT crea un sistema completo e scalabile per l'acquisizione, l'archiviazione e l'accesso ai dati del rover.

3.6 Broker MQTT

MQTT, acronimo di Message Queuing Telemetry Transport, è un protocollo di messaggistica leggero e basato su pubblicazione/sottoscrizione (Publish/Subscribe). È progettato per essere efficiente in termini di larghezza di banda e consumare poche risorse, il che lo rende particolarmente adatto per applicazioni IoT (Internet of Things) e reti a basso consumo.

Funzionamento di MQTT:

- Client MQTT: Un client MQTT può essere un dispositivo IoT, un'applicazione o qualsiasi entità che invia o riceve messaggi tramite il protocollo MQTT. Un client può agire sia da produttore (publish) che da consumatore (subscribe) di messaggi.
- Broker MQTT: Il broker MQTT è un intermediario che riceve, instrada e gestisce i messaggi tra i client. Il broker è responsabile di garantire che i messaggi siano recapitati ai client corretti secondo il modello di pubblicazione/sottoscrizione.

Publish/Subscribe Model:

- Publish (Pubblicare): Un client MQTT può pubblicare messaggi su un argomento (topic). Gli argomenti sono stringhe che identificano il canale o il “tema” dei messaggi. Quando un client pubblica un messaggio su un argomento, il broker inoltra quel messaggio a tutti i client che sono sottoscritti a quell'argomento.
- Subscribe (Sottoscrivere): Un client MQTT può sottoscriversi a uno o più argomenti. Quando un client è sottoscritto a un argomento, il broker inoltrerà a quel client i messaggi pubblicati su quell'argomento da altri client.

Nella figura 20 si può vedere il funzionamento in maniera schematica del protocollo MQTT.

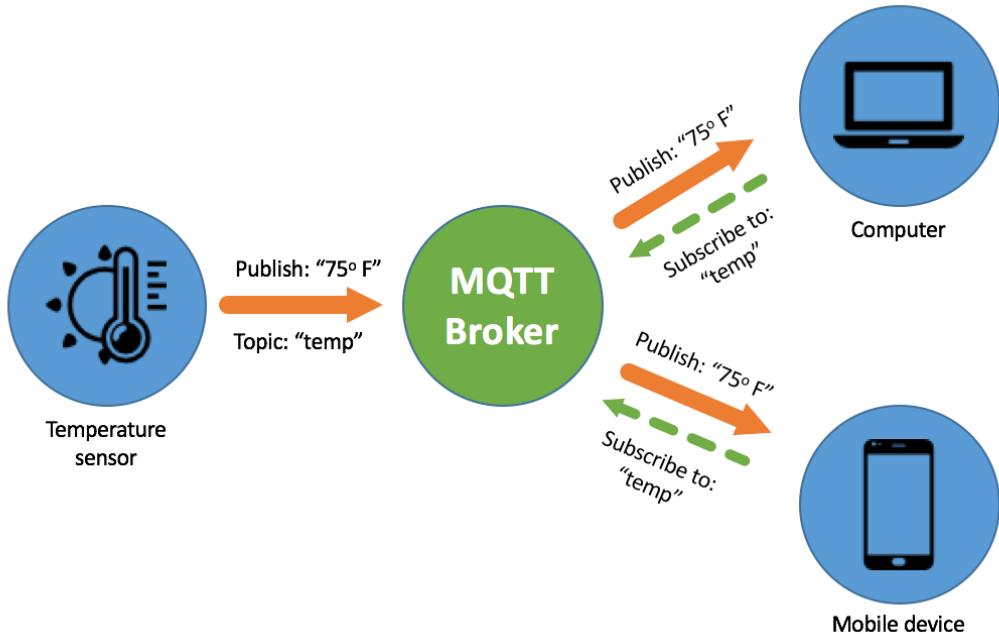


Figura 20: Schema Mqtt

Il QoS (Quality of Service) in MQTT definisce il livello di affidabilità e garanzia di consegna associato a un messaggio. MQTT offre tre livelli di QoS, ognuno dei quali si adatta a diverse esigenze di affidabilità:

1. QoS 0 (At Most Once):

- In questo livello di QoS, il messaggio viene consegnato al massimo una volta, senza alcuna conferma di ricezione.
- È il livello di QoS più leggero e veloce, ma non garantisce che il messaggio raggiunga il destinatario o che venga consegnato solo una volta. Potrebbe esserci una perdita di messaggi in situazioni di rete instabile.
- Applicabile in situazioni in cui la perdita di alcuni messaggi non è critica e il focus principale è sulla velocità e la leggerezza delle comunicazioni.
- Esempi di utilizzo: trasmissioni di dati in tempo reale in cui la perdita di un singolo dato non è cruciale.

2. QoS 1 (At Least Once):

- In questo livello di QoS, il mittente invia il messaggio e il ricevente conferma la ricezione. Se il mittente non riceve la conferma, invia nuovamente il messaggio.
- Assicura che il messaggio venga ricevuto almeno una volta dal destinatario, ma potrebbe causare la ricezione duplicata del messaggio in alcune situazioni.
- Adatto per applicazioni in cui è importante che il messaggio venga ricevuto, ma ricevere il messaggio più di una volta non provoca problemi.
- Esempi di utilizzo: monitoraggio di sensori in cui la perdita di dati è accettabile, ma è importante ricevere i dati correttamente.

3. QoS 2 (Exactly Once):

- È il livello di QoS più affidabile. In questo caso, il mittente invia il messaggio, e il destinatario conferma la ricezione. Successivamente, il mittente invia un messaggio di conferma di ricezione.
- Garantisce che il messaggio venga consegnato esattamente una volta. Tuttavia, questo livello di QoS comporta un maggiore overhead di comunicazione a causa degli scambi aggiuntivi tra mittente e destinatario.
- Applicato in situazioni in cui la consegna esattamente una volta è critica e la ricezione duplicata di un messaggio è inaccettabile.
- Esempi di utilizzo: controllo remoto di attuatori, comandi critici in cui la ricezione duplicata potrebbe causare problemi.

Nel contesto del progetto, MQTT svolge un ruolo chiave nella comunicazione tra i diversi componenti del sistema, come la Raspberry Pi (che agisce come il client MQTT), la macchina virtuale Ubuntu (che funge da broker MQTT).

La Raspberry Pi pubblicherà dati raccolti dai sensori e informazioni sullo stato del rover attraverso MQTT. Riceverà comandi e istruzioni dal sistema di controllo remoto (Interfaccia Node-RED) attraverso sottoscrizioni MQTT.

La macchina virtuale Ubuntu ospiterà il broker MQTT che gestisce la ricezione e l'instradamento dei messaggi tra la Raspberry Pi e il sistema di controllo remoto. Consentirà la sottoscrizione ai dati del rover e l'invio di comandi al rover tramite pubblicazione e sottoscrizione MQTT.

4 Implementazione

4.1 Realizzazione dello schema elettrico e assemblaggio del rover

Come primo step, viene assemblato il telaio del rover montando i vari componenti nel modo ottimale. In particolare, è stato necessario realizzati 3 piani separati, ciascuno con un ruolo chiave, che verranno descritti di seguito.

Per descrivere i collegamenti elettrici effettuati, sono stati realizzati degli schemi con il software Fritzing, importando esternamente i vari componenti non inclusi nella libreria di default.

4.1.1 Primo piano

Nel primo piano del rover, viene alloggiato il pacco batterie con relativo BMS, il modulo di controllo dei motori L298N e il regolatore di tensione LM2596S.

La scelta di posizionare questi componenti nel piano più inferiore, deriva dal peso delle batterie (e quindi l'abbassamento del baricentro), la vicinanza dei motori al modulo di controllo e l'utilizzo di cavi più corti dalle batterie per il regolatore di tensione.

Inoltre, è stato inserito un interruttore per accendere e spegnere il rover.

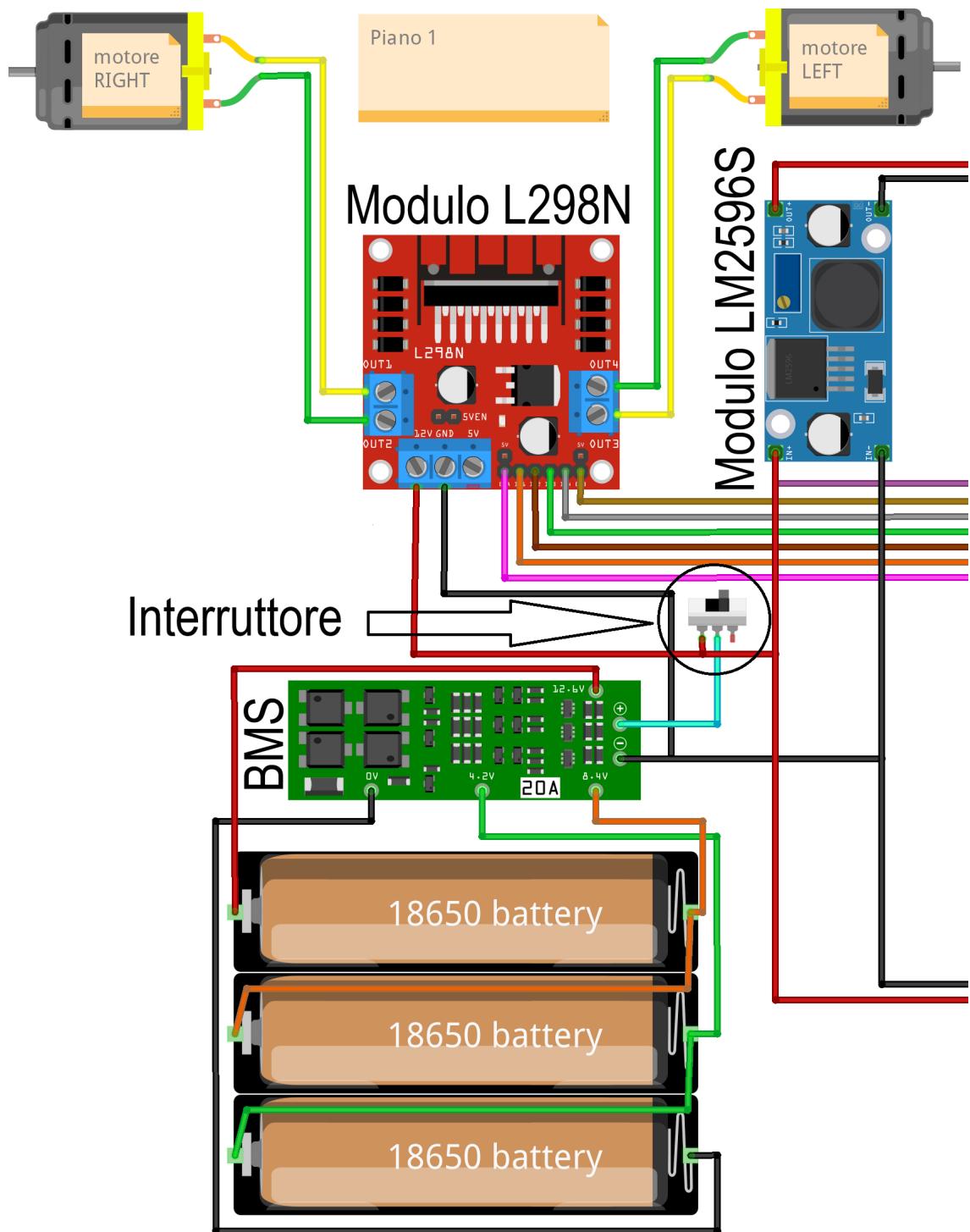


Figura 21: Schema elettrico piano 1

4.1.2 Secondo piano

Nel secondo piano, sono stati inseriti i vari sensori insieme all'ESP32 come mostrato in figura 22.

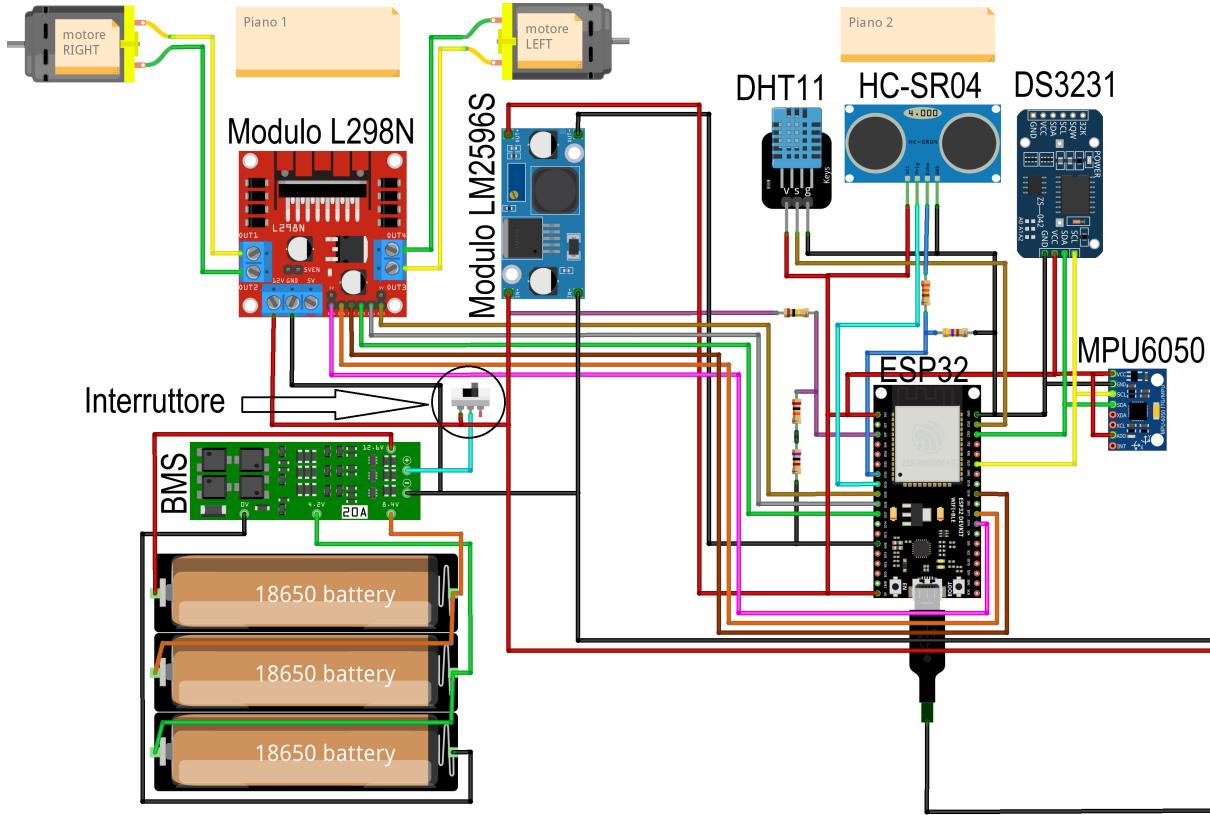


Figura 22: Schema elettrico piano 1 e 2

In questo schema elettronico, in aggiunta al precedente si può osservare che:

- Il pin *GPIO36* dell'ESP32 è stato collegato, tramite partitore di tensione, al cavo positivo della batteria al fine di misurarne la tensione. Il partitore di tensione è stato inserito perché l'ESP32 non può ricevere in input una tensione superiore a 3.3v. Inserendo quindi le resistenze come mostrato in figura 23, la tensione in input può essere compresa tra 15v e 0v. Per il dimensionamento delle resistenze, è stata utilizzata la seguente formula:

$$V_{out} = \frac{V_{source} \cdot R_2}{R_1 + R_2} \quad (1)$$

Da cui, fissati i valori $V_{out} = 3.3v$ e $V_{source} = 15v$, si trovano $R_1 = 100000\Omega$ e $R_2 = 27000\Omega$.

Non avendo a disposizione una singola resistenza da $27k\Omega$, è stata ottenuta una resistenza equivalente inserendo due resistenze in serie da $20k\Omega$ e $7k\Omega$.

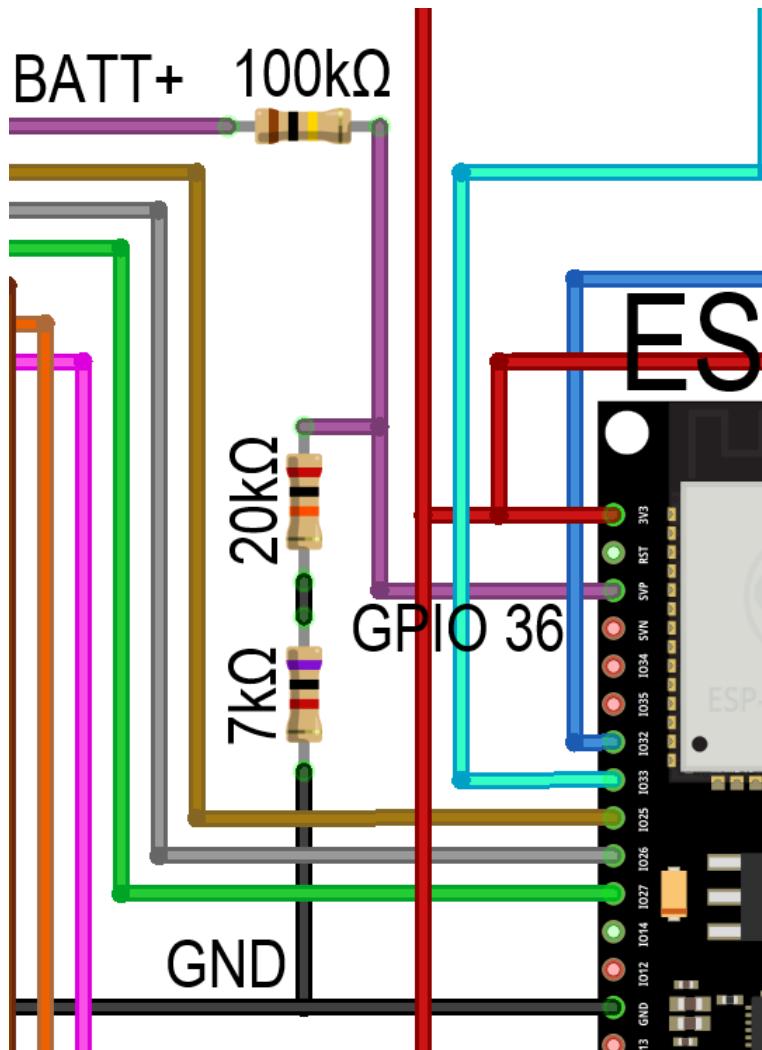


Figura 23: Partitore di tensione per batteria

- I pin positivo e negativo in uscita dal regolatore di tensione LM2596S sono stati collegati rispettivamente ai pin *VIN* e *GND* dell'ESP32. Infatti, tramite il DevKit ESP32 utilizzato, nel pin *VIN* può essere inserita una tensione di 5v che verrà poi trasformata a 3.3v grazie al regolatore di tensione interno.
 - I 6 collegamenti del modulo L298N (3 per motore, considerando il pin *EN*, *IN1* e *IN2*) che giungono dal piano 1, vengono collegati all'ESP32 nel piano 2.
 - Il modulo DHT11 viene collegato attraverso 2 pin di alimentazione, *Vcc* e *GND* collegati rispettivamente al 3.3v e al *GND* dell'ESP32, e un pin di segnale *Data* collegato al *GPIO23*.
 - Il modulo HC-SR04, come riportato nel datasheet all'indirizzo cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf, necessita di una alimentazione di 5v anzichè 3.3v. Per questo pin *Vcc* viene collegato al pin *VIN*, a cui è collegata l'alimentazione proveniente dal regolatore di tensione dedicato. La massa rimane comune, collegandola al *GND* dell'ESP32. Il pin *TRIGGER* viene collegato direttamente al pin *GPIO33* perché si considera sufficiente la tensione di 3.3v per emettere il segnale. Nel pin di risposta *ECHO*, che invece ha una tensione di 5v, necessita di un partitore di tensione per evitare di danneggiare il pin di ingresso dell'ESP32 che accetta una tensione massima

di 3.3V. Utilizzando sempre la formula espressa nell'equazione 1, si trovano i valori $R_1 = 330\Omega$ ed $R_2 = 470\Omega$ e vengono collegate come mostrato in figura 24.

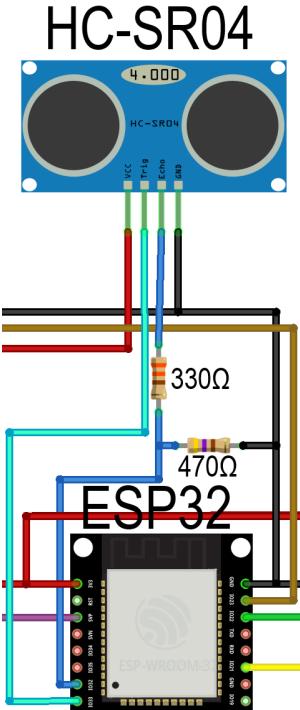


Figura 24: Partitore di tensione per pin *ECHO*

- Per il modulo DS3231, che può lavorare ad una tensione compresa tra i 3.3v e 5v, viene collegato all'alimentazione al pin *3V3* e *GND* dell'ESP32. I pin *SCL* e *SDA*, per la comunicazione I2C, vengono collegati ai pin predefiniti dell'ESP32 per questa comunicazione ovvero rispettivamente *GPIO21* e *GPIO22*.
 - Il modulo MPU6050 presenta anch'esso i 4 pin del modulo DS3231 e vengono collegati allo stesso modo. Infatti, nei pin *SCL* e *SDA* possono essere collegati più moduli contemporaneamente. In aggiunta, viene collegato il pin *AD0* a *Vcc* per cambiare l'indirizzo I2C da 0x68 a 0x69. Questo perché anche il modulo DS3231 ha come indirizzo predefinito 0x68 e questo provoca una collisione di indirizzi.

4.1.3 Terzo piano

Nel terzo piano, si trova solamente la Raspberry Pi 3B. Questa viene alimentata tramite un regolatore di tensione separato che offre una porta USB in modo da poter collegare un cavo USB - micro usb al connettore dedicato. Nonostante sia possibile alimentare la Raspberry anche attraverso la GPIO, questa viene considerata una pratica insicura e sconsigliata perché non vi è alcuna protezione in caso di malfunzionamenti all'alimentatore, danneggiando la scheda irrimediabilmente. Inoltre vi è un ulteriore cavo USB - USB-C per collegare tramite seriale l'ESP32 e la Raspberry. Lo schema elettrico completo viene mostrato in figura 25.

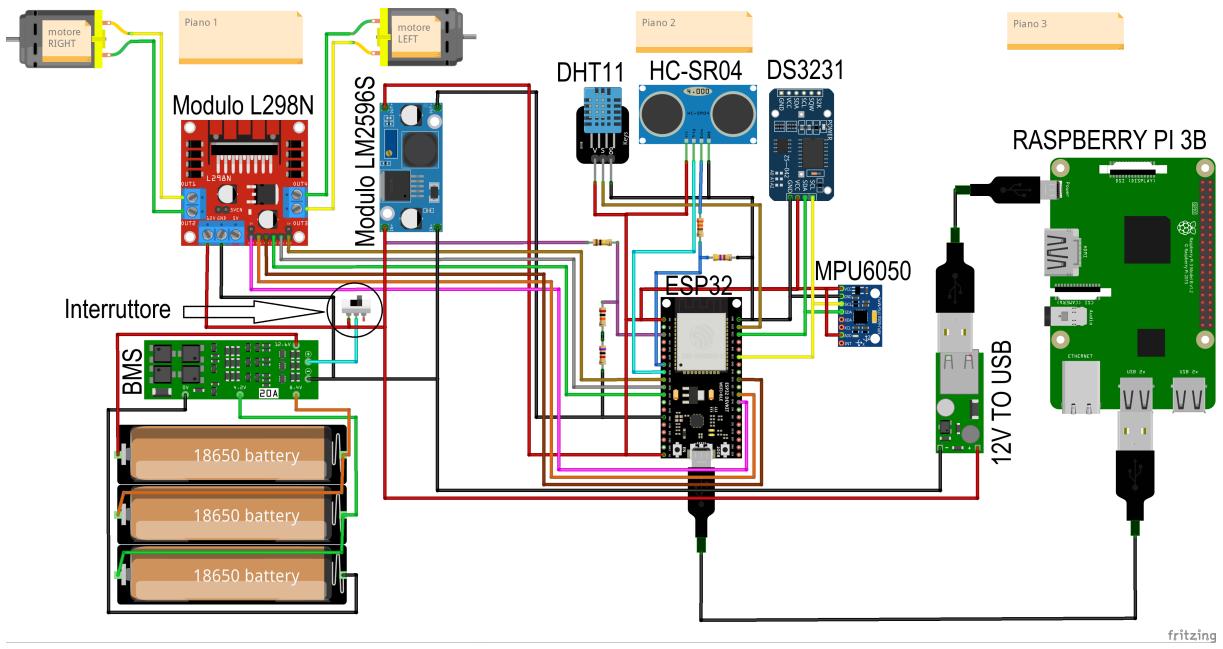


Figura 25: Schema elettrico completo

La realizzazione pratica del rover completo dei tre piani è mostrato in figura 26.

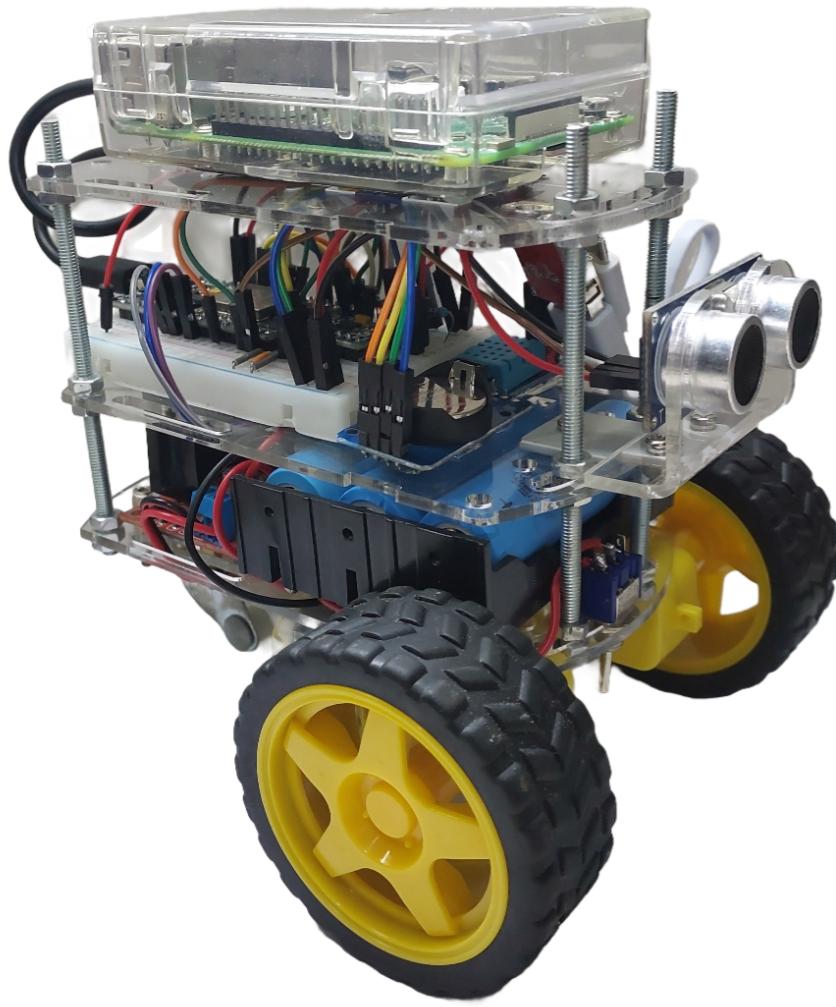


Figura 26: Foto del rover completo

4.2 Sviluppo su ESP32

La programmazione dell'ESP32 è stata eseguita tramite l'IDE di Arduino, scaricabile dal seguente link www.arduino.cc/en/software. Infatti, scaricando delle opportune librerie, è possibile programmare anche dispositivi diversi da Arduino. In questo caso è stata utilizzata la seguente libreria ufficiale scaricabile al link github.com/espressif/arduino-esp32 importandola nella sezione “Gestore schede” dell'IDE. Si seleziona quindi la scheda utilizzata (in questo caso “ESP32-WROOM-DA”) e la porta seriale in cui è collegata. Iniziando lo sviluppo, per prima cosa si importano le seguenti librerie:

```
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
#include <Wire.h>
#include <DS3231.h>
#include <HCSR04.h>
#include <L298N.h>
#include <DHT.h>
```

```
#include <ArduinoJson.h>
```

Che vengono scaricate nella sezione “Gestore librerie” dai seguenti indirizzi:

- Adafruit_MPU6050: github.com/adafruit/Adafruit_MPU6050
- Adafruit_Sensor: github.com/adafruit/Adafruit_Sensor
- DS3231:github.com/NorthernWidget/DS3231
- HCSR04: github.com/Martinsos/arduino-lib-hc-sr04
- L298N: github.com/AndreaLombardo/L298N
- DHT: github.com/adafruit/DHT-sensor-library
- ArduinoJson: arduinojson.org

Vi è una sezione in cui si definiscono gli utilizzi dei pin dell'ESP32 tramite costanti, in modo da avere un'unica parte iniziale del programma in cui questi sono definiti e possono essere cambiati.

```
// Definizione dei pin GPIO ESP32
#define HCSR04_TRIGGER_PIN 32
#define HCSR04_ECHO_PIN 33
#define DHT11_PIN 23
#define BATTERY_PIN 36
#define L_MOTOR_EN_PIN 25
#define L_MOTOR_IN1_PIN 26
#define L_MOTOR_IN2_PIN 27
#define R_MOTOR_EN_PIN 16
#define R_MOTOR_IN1_PIN 17
#define R_MOTOR_IN2_PIN 18
```

Successivamente, si definiscono alcune costanti utili nel programma, come i tempi di delay per ogni task e la distanza di sicurezza in cui il rover si deve arrestare:

```
#define DELAY_SEND_SENSOR_DATA_MS 2000
#define DELAY_FUNCTION_MODE_MS 50
#define DELAY_RECEIVE_DATA_MS 100
#define SAFETY_DISTANCE_CM 20
```

Si dichiarano i vari oggetti utilizzati, come i sensori e i due documenti json di input e output:

```
Adafruit_MPU6050 mpu;
RTClib myRTC;
DS3231 myClock;
DHT dht11(DHT11_PIN, DHT11);
L298N l_motor(L_MOTOR_EN_PIN, L_MOTOR_IN1_PIN, L_MOTOR_IN2_PIN);
L298N r_motor(R_MOTOR_EN_PIN, R_MOTOR_IN1_PIN, R_MOTOR_IN2_PIN );
UltraSonicDistanceSensor distanceSensor(HCSR04_TRIGGER_PIN, HCSR04_ECHO_PIN
    → );
StaticJsonDocument<250> docOutput;
StaticJsonDocument<250> docInput;
```

Per calcolare la dimensione del documento json, è stato utilizzato il tool disponibile alla pagina: arduinojson.org/v6/assistant

Piu avanti, vengono definite delle variabili per l'utilizzo nei vari task e degli enum per la classificazione dei comandi ricevuti e della modalità di funzionamento:

```
sensors_event_t a, g, temp;

enum function_mode{
    AUTO = 0 ,
    MAN = 1
} function_mode_selected;

enum command{
    FORWARD = 0,
    BACKWARD = 1,
    RIGHT = 2,
    LEFT = 3,
    STOP = 4
} command_received;
```

Il metodo **setup()** riportato di seguito è parte del framework di Arduino e viene chiamata una sola volta all'inizio del programma per l'inizializzazione. In questa fase, vengono configurate le impostazioni iniziali del microcontrollore e iniziano a eseguire i task o le funzioni principali del programma.

```
void setup(void) {
    Serial.begin(115200);
    Wire.begin();
    if (!mpu.begin(0x69)) {
        Serial.println("Failed_to_find_MP6050_chip");
    }else{
        Serial.println("MPU6050_Found!");
    }
    dht11.begin();

    function_mode_selected=MAN;
    command_received=STOP;
    l_motor.setSpeed(70);
    r_motor.setSpeed(70);

    xTaskCreate(taskRiceviDati, "Task_ricezione_dati", 5000, NULL, 2, NULL);
    xTaskCreate(taskFunctionMode, "Task_modalita_di_funzionamento", 5000,
                ↗ NULL, 3, NULL);
    xTaskCreate(taskInvioDati, "Task_invio_dati", 5000, NULL, 1, NULL);
}

void loop () {}
```

- `Serial.begin(115200)`: Inizializza la comunicazione seriale a una velocità di 115200 bps. Questo è utilizzato per la comunicazione tra il microcontrollore e la Raspberry tramite il monitor seriale.
- `Wire.begin()`: Inizializza la libreria di comunicazione I2C (Wire) che è comunemente usata per comunicare con sensori esterni, come l'MPU6050 (accelerometro e giroscopio) e l'RTC.

- `mpu.begin(0x69)`: Inizializza il sensore MPU6050 utilizzando l'indirizzo I2C specificato (0x69). Se la comunicazione con il sensore ha successo, viene stampato un messaggio indicando che il sensore è stato trovato.
- `dht11.begin()`: Inizializza il sensore di temperatura e umidità DHT11.
- `function_mode_selected=MAN` e `command_received=STOP`: Inizializza alcune variabili di stato. La variabile `function_mode_selected` è impostata su MAN (modalità manuale), e `command_received` è impostata su STOP.
- `l_motor.setSpeed(70)` e `r_motor.setSpeed(70)`: Imposta la velocità dei motori sinistro (`l_motor`) e destro (`r_motor`) a 70 su 255 (circa il 30% della potenza).
- `xTaskCreate(...)`: Questo blocco di codice crea e avvia tre task paralleli (multithreading). Questi task sono designati per eseguire specifiche funzioni in modo indipendente, permettendo al programma di gestire contemporaneamente diverse attività. Nella funzione vengono passati i seguenti parametri:
 - Nome della funzione che verrà utilizzata dal task
 - Nome in stringa del task
 - Dimensione dello stack per il task
 - Parametro in input al task (in questo caso non vengono utilizzati)
 - Priorità del task (un numero più piccolo indica una priorità più bassa [6])
 - Un handle che viene utilizzato per richiamare il task in parti del codice esterno al task (in questo caso non viene utilizzato)
- `void loop()`: Questa funzione è vuota () e non contiene alcun codice. In un programma Arduino, il loop viene eseguito continuamente dopo la fase di setup. Tuttavia, in questo caso, la logica principale del programma è gestita attraverso i task creati nel setup, e il loop è stato lasciato vuoto per evitare l'esecuzione di codice in modo sequenziale all'interno del loop principale.

4.2.1 Task invio dati

Il task **taskInvioDati()** svolge un ruolo cruciale nel raccogliere dati da diversi sensori e inviarli attraverso la porta seriale in un formato strutturato. La strutturazione gerarchica del documento JSON facilita la comprensione e l'interpretazione dei dati ricevuti da parte di un sistema esterno (in questo caso la Raspberry).

```
void taskInvioDati(void *pvParameters) {
    while (1) {
        mpu.getEvent(&a, &g, &temp);

        docOutput["distance"] = distanceSensor.measureDistanceCm();
        docOutput["acceleration"]["x"] = a.acceleration.x;
        docOutput["acceleration"]["y"] = a.acceleration.y;
        docOutput["acceleration"]["z"] = a.acceleration.z;
        docOutput["rotation"]["x"] = g.gyro.x;
        docOutput["rotation"]["y"] = g.gyro.y;
        docOutput["rotation"]["z"] = g.gyro.z;
        docOutput["unixtime"] = myRTC.now().unixtime();
        docOutput["temperature"] = dht11.readTemperature();
        docOutput["humidity"] = dht11.readHumidity();
        docOutput["battery"]["voltage"] = readVoltage();
        serializeJson(docOutput, Serial);
    }
}
```

```

    Serial.println();

    vTaskDelay(DELAY_SEND_SENSOR_DATA_MS / portTICK_PERIOD_MS);
}

}

float readVoltage() {
    return (float)analogRead(BATTERY_PIN) / 4096 * 17 ;
}

```

Il ciclo principale del task è strutturato come segue:

1. Acquisizione dei dati:

- Utilizza l'oggetto mpu per ottenere le letture di accelerazione (a), rotazione (g), e temperatura (temp) dal sensore MPU6050 (accelerometro e giroscopio).
- Utilizza il sensore di distanza (distanceSensor) per misurare la distanza in centimetri.
- Utilizza l'oggetto myRTC per ottenere l'ora attuale in formato unixtime.
- Legge la temperatura (temperature) e umidità (humidity) dal sensore DHT11.
- Legge la tensione della batteria tramite la funzione `readVoltage()`.

2. Assemblaggio dei dati:

- Struttura un documento JSON (docOutput) che contiene tutte le letture acquisite dai sensori. I dati vengono organizzati in un formato gerarchico.

3. Serializzazione e invio:

- Utilizza la libreria “ArduinoJSON” per serializzare il documento JSON (`serializeJson(docOutput, Serial)`) e invia il risultato sulla porta seriale (`Serial.println()`).

4. Ritardo tra le iterazioni:

- Utilizza `vTaskDelay()` per introdurre un ritardo tra le iterazioni del loop, garantendo che il task esegua ciclicamente il task con almeno un determinato ritardo. Il ritardo è specificato dal valore `DELAY_SEND_SENSOR_DATA_MS`.

4.2.2 Task ricezione dati

Il task **taskRiceviDati** gestisce la ricezione e l'interpretazione di dati provenienti dalla porta seriale, fornendo un'interfaccia per la configurazione e il controllo del rover tramite comandi JSON. L'uso di JSON facilita la trasmissione di dati strutturati e la loro elaborazione in modo ordinato nel sistema.

```

void taskRiceviDati(void *pvParameters) {
    while (1) {
        if(Serial.available()) {
            DeserializationError err = deserializeJson(docInput, Serial);
            if (err == DeserializationError::Ok)
            {
                if (!docInput["rtc_sync_unixtime"].isNull()) {
                    time_t epoch = (time_t)docInput["rtc_sync_unixtime"].as<long>();
                    myClock.setEpoch(epoch);
                }
            }
        }
    }
}

```

```

if (!docInput ["function_mode_auto"].isNull()) {
    if (docInput ["function_mode_auto"].as<String> ()=="true") {
        function_mode_selected=AUTO;
    }else{
        function_mode_selected=MAN;
    }
}
if (!docInput ["movement_command"].isNull ()) {
    command_received = (command) docInput ["movement_command"].as<int> ();
}
if (!docInput ["speed"].isNull ()) {
    int speed=docInput ["speed"].as<int> ();
    if(speed>0 && speed<=100) {
        //mapping della velocità che viene ricevuta in 0-100% al range pwm in
        ↪ 0-255
        speed = map(speed, 0, 100, 0, 255);
        l_motor.setSpeed(speed);
        r_motor.setSpeed(speed);
    }
}
else
{
    // Flush all bytes in the "link" serial port buffer
    while (Serial.available() > 0)
        Serial.read();
}
}
vTaskDelay(DELAY_RECEIVE_DATA_MS / portTICK_PERIOD_MS);
}
}

```

Il suo ciclo principale è strutturato come segue:

1. Verifica della disponibilità di dati seriali:
 - Verifica se ci sono dati disponibili sulla porta seriale utilizzando `Serial.available()`.
2. Deserializzazione dei dati JSON:
 - Se ci sono dati disponibili, utilizza la libreria ArduinoJSON per deserializzare i dati JSON dalla porta seriale (`deserializeJson(docInput, Serial)`).
 - Gestisce eventuali errori di deserializzazione tramite la variabile `err`
3. Elaborazione dei dati ricevuti:
 - Se la deserializzazione è avvenuta correttamente (`err == DeserializationError::Ok`), il task procede ad analizzare il contenuto del documento JSON (`docInput`).
 - Se è presente un campo chiamato `rtc_sync_unixtime` nel documento JSON, viene estratto il valore associato e viene utilizzato per sincronizzare l'orologio (`myClock`).
 - Analogamente, vengono letti e interpretati altri campi come `function_mode_auto`, `movement_command`, e `speed`. Questi valori vengono utilizzati per impostare la modalità di funzionamento, il comando di movimento e la velocità dei motori, rispettivamente.

4. Gestione degli errori di deserializzazione:

- In caso di errori di deserializzazione, ad esempio a causa di un formato non valido, viene eseguito un processo di pulizia del buffer seriale per evitare problemi futuri (`Serial.read()`)

5. Ritardo tra le iterazioni:

- Utilizza `vTaskDelay()` per introdurre un ritardo tra le iterazioni del loop, garantendo che il task esegua ciclicamente il task con almeno un determinato ritardo. Il ritardo è specificato dal valore `DELAY_RECEIVE_DATA_MS`.

4.2.3 Task modalità di funzionamento

Il task `taskFunctionMode` implementa la logica di controllo del rover in base alla modalità di funzionamento selezionata e ai comandi ricevuti, fornendo un meccanismo flessibile per controllare il rover sia manualmente che autonomamente.

```
void taskFunctionMode(void *pvParameters) {
    while (1) {

        if(function_mode_selected == AUTO) {

            if(distanceSensor.measureDistanceCm() <=SAFETY_DISTANCE_CM) {
                r_motor.stop();
                l_motor.stop();
                delay(500);
                //random rotate
                r_motor.forward();
                l_motor.backward();
                delay(random(300,800));
                r_motor.stop();
                l_motor.stop();
                delay(500);
            }else{
                r_motor.forward();
                l_motor.forward();
            }
        }
        else
        {
            switch(command_received) {
                case FORWARD:
                    if(distanceSensor.measureDistanceCm() <=SAFETY_DISTANCE_CM) {
                        r_motor.stop();
                        l_motor.stop();
                    }else{
                        r_motor.forward();
                        l_motor.forward();
                    }
                    break;
                case BACKWARD:
                    r_motor.backward();
                    l_motor.backward();
                    break;
            }
        }
    }
}
```

```

    case RIGHT:
        r_motor.backward();
        l_motor.forward();
        break;
    case LEFT:
        r_motor.forward();
        l_motor.backward();
        break;
    case STOP:
        r_motor.stop();
        l_motor.stop();
        break;

    default:
        r_motor.stop();
        l_motor.stop();

    }
}
vTaskDelay(DELAY_FUNCTION_MODE_MS / portTICK_PERIOD_MS);
}
}

```

Il suo ciclo principale è strutturato come segue:

1. Modalità automatica (AUTO):

- Se la modalità di funzionamento è impostata su AUTO, il rover agisce autonomamente.
- Misura la distanza dal sensore (`distanceSensor`) e, se la distanza è inferiore o uguale alla `SAFETY_DISTANCE_CM` (distanza di sicurezza), ferma il rover, esegue una rotazione casuale e poi si ferma nuovamente.
- Se la distanza è superiore alla `SAFETY_DISTANCE_CM`, il rover continua a muoversi in avanti.

2. Modalità manuale (MAN):

- Se la modalità di funzionamento è impostata su MAN (manuale), il rover risponde ai comandi ricevuti (`command_received`) tramite la porta seriale.
- I comandi possono essere FORWARD, BACKWARD, RIGHT, LEFT, STOP.
- In base al comando ricevuto, il rover si muove in avanti, indietro, a destra, a sinistra, si ferma o esegue altre azioni specifiche.
- Se la distanza è inferiore alla `SAFETY_DISTANCE_CM`, il rover si arresta e non consente di muoversi in avanti.

3. Ritardo tra le iterazioni:

- Utilizza `vTaskDelay()` per introdurre un ritardo tra le iterazioni del loop, garantendo che il task esegua ciclicamente il task con almeno un determinato ritardo. Il ritardo è specificato dal valore `DELAY_FUNCTION_MODE_MS`.

Il codice completo si trova nel seguente repository [github](#) [7].

4.3 Sviluppo su Raspberry

La scheda di sviluppo Raspberry, per poter funzionare e gestire correttamente tutte le sue risorse, ha bisogno di un sistema operativo. Esistono tanti di questi che si possono utilizzare, ciascuno per un utilizzo specifico. Verrà poi sviluppato il codice necessario per comunicare tramite protocollo MQTT le informazioni tra il rover e l'interfaccia web.

4.3.1 Installazione sistema operativo Raspberry Pi OS

In questo caso, è stato scelto di utilizzare il sistema operativo ufficiale Raspberry Pi OS, un sistema general purpose che comprende un desktop minimale e diversi software per un utilizzo comune. Per installarlo sulla memoria SD da inserire, è sufficiente scaricare dal sito ufficiale il tool Raspberry Pi Imager all'indirizzo raspberrypi.com/software. Una volta scaricato e aperto, nell'interfaccia basterà selezionare il modello di Raspberry Pi, il sistema operativo desiderato e la memoria SD. Il processo è mostrato in figura 27.

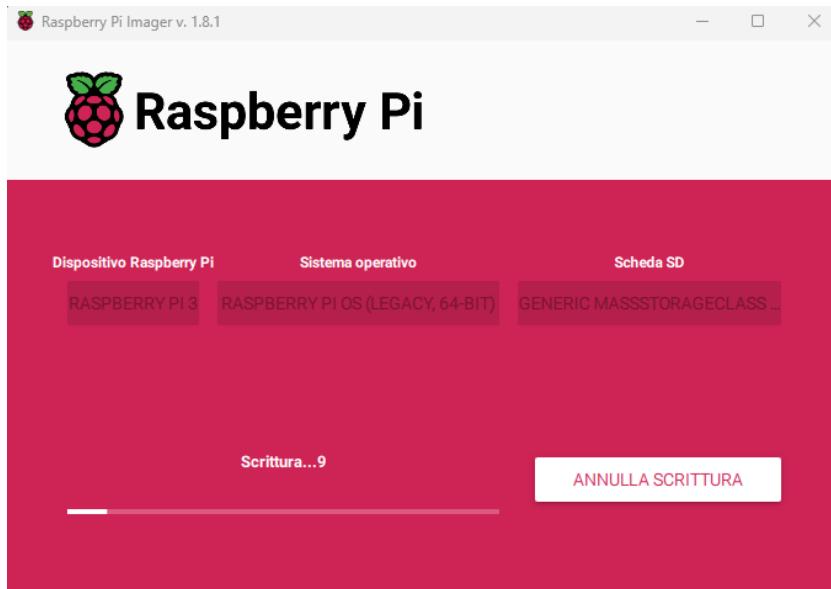


Figura 27: Scrittura del sistema operativo su memoria SD

Una volta terminato il processo di scrittura, basterà inserire la memoria SD all'interno dello slot della Raspberry Pi e avviirla. Inizieranno i primi passi di configurazione del sistema operativo (come la scelta del layout della tastiera, lingua, nome utente e password). Terminato il processo, il sistema operativo è pronto ad essere utilizzato.

4.3.2 Sviluppo software client MQTT

Per lo sviluppo del software è stato scelto il linguaggio Python. Il seguente codice implementa la comunicazione bidirezionale tra il dispositivo ESP32 e il broker MQTT su server Ubuntu, utilizzando la libreria Paho MQTT e la comunicazione seriale. La comunicazione seriale è gestita attraverso il modulo `serial`, mentre l'interazione MQTT è possibile grazie a `paho.mqtt.client`.

```
import serial
import simplejson as json
import paho.mqtt.client as mqtt
```

```

def on_connect(client, userdata, flags, rc):
    print("Connected_to_host_", client._host, "'")
    client.subscribe("movement_command")
    client.subscribe("speed")
    client.subscribe("function_mode_auto")
    client.subscribe("rtc_sync_unixtime")

def on_message(client, userdata, msg):
    #write to serial port json with key msg.topic and value msg.payload
    ser.write(json.dumps({msg.topic:msg.payload}).encode('utf-8'))
    print("Serial_out:",json.dumps({msg.topic:msg.payload}))

client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

client.connect("ubuntu", 1883, 60)
client.loop_start()
buffer=""

try:

    ser = serial.Serial('/dev/ttyUSB0',baudrate=115200)
    if ser.isOpen():
        print("Serial_port_opened:" + ser.portstr)

    ser.reset_input_buffer()

    while True:
        try:
            if ser.inWaiting()>0:
                buffer = ser.readline().decode('utf-8').rstrip()
                print ("Serial_in:",buffer)
                data=json.loads(buffer)
                client.publish(topic = "rover_output", payload = buffer, qos=1)
            except Exception as e:
                print(e)

        except KeyboardInterrupt:
            print("Stop_program...")
            ser.close()
        except serial.SerialException:
            print("Serial_error")

    client.loop_stop()
    client.disconnect()

```

Il codice sopra riportato può essere suddiviso nelle seguenti fasi:

- Configurazione iniziale:
 - Il codice stabilisce una connessione al server MQTT sulla macchina con hostname “ubuntu” alla

porta 1883.

- Vengono definite le callback `on_connect` e `on_message` per gestire gli eventi di connessione e di ricezione dei messaggi MQTT.
 - Il client MQTT viene avviato in un thread separato (`client.loop_start()`).
- Gestione dei messaggi MQTT:
 - La callback `on_message` è chiamata ogni volta che il client riceve un messaggio da uno dei topic sottoscritti.
 - I messaggi MQTT vengono serializzati in formato JSON e inviati attraverso la porta seriale utilizzando `ser.write()`.
- Inizializzazione della comunicazione seriale:
 - Viene inizializzata una connessione seriale (`ser`) sulla porta “/dev/ttyUSB0” con un baud rate di 115200.
 - Vengono gestite eccezioni per interruzioni da tastiera (`KeyboardInterrupt`) o errori nella comunicazione seriale (`serial.SerialException`).
- Lettura dati dalla seriale e pubblicazione su MQTT:
 - All’interno di un loop continuo, il programma legge i dati dalla seriale, li decodifica da UTF-8 e li pubblica su un topic MQTT chiamato “rover_output”. Si nota che la pubblicazione viene gestita con QoS pari a 1.
- Terminazione del programma:
 - Il programma si ferma in caso di interruzione da tastiera o errori nella comunicazione seriale.
 - Operazioni di chiusura del client MQTT e della connessione seriale vengono eseguite prima della terminazione.

Inoltre, per rilevare la porta seriale da utilizzare, è stato realizzato uno script secondario contenente la seguente porzione di codice:

```
import serial
import serial.tools.list_ports

#print serial port available
ports = serial.tools.list_ports.comports()
for port, desc, hwid in sorted(ports):
    print ("{}:{}[{}].format(port, desc, hwid))
```

Praticamente, grazie al tool `serial.tools.list_ports`, fornisce all’utente una lista delle porte seriali disponibili sul sistema, insieme alle relative informazioni. La stampa delle informazioni avviene in un formato che facilita la lettura e l’identificazione delle porte seriali.

Il codice sviluppato è disponibile nel repository github [7].

4.3.3 Creazione del servizio per l’avvio automatico

Per far sì che il programma client si avvi in automatico all’avvio della Raspberry è stato realizzato un servizio dedicato.

Il primo passaggio consiste nel creare un file di configurazione del servizio nel percorso indicato con il comando:

```
sudo vim /lib/systemd/system/mqtt_client.service
```

Il contenuto del file di configurazione è il seguente:

```
[Unit]
Description= Service mqtt client
After=multi-user.target

[Service]
User=denil
Restart=on-failure
RestartSec=30
Type=idle
ExecStart=/usr/bin/python /home/denil/mqtt_client/mqtt_client.py

[Install]
WantedBy=multi-user.target
```

Si nota che su ExecStart viene riportato il comando da eseguire, su User l'utente che esegue il comando e su Restart viene impostato il criterio di riavvio del servizio (al fallimento dell'operazione).

Per abilitare il servizio, vengono eseguiti in successione i seguenti comandi:

- sudo systemctl daemon-reload
- sudo systemctl enable mqtt_client.service
- sudo systemctl start mqtt_client.service
- sudo systemctl status mqtt_client.service

Con l'ultimo comando si osserva lo stato del servizio che, se è andato tutto a buon fine, dovrà essere su “Running”. In figura 28 si mostra l'output del comando.

```
denil@raspberrypi:~ $ sudo systemctl status mqtt_client.service
● mqtt_client.service - Service mqtt client
   Loaded: loaded (/lib/systemd/system/mqtt_client.service; enabled; vendor preset: enabled)
   Active: active (running) since Sat 2024-01-06 12:58:20 CET; 2s ago
     Main PID: 1511 (python)
        Tasks: 1 (limit: 779)
       CPU: 497ms
      CGroup: /system.slice/mqtt_client.service
              └─1511 /usr/bin/python /home/denil/mqtt_client/mqtt_client.py
```

Figura 28: Stato del servizio creato

4.4 Sviluppo su macchina virtuale Ubuntu

Per sviluppare l'interfaccia web e il broker MQTT, è stato scelto il sistema operativo Ubuntu 23.10.1.

4.4.1 Creazione macchina virtuale e installazione Ubuntu

Per questioni di praticità, è stata realizzata una macchina virtuale con il software “VMware workstation player”, scaricabile gratuitamente all'indirizzo vmware.com/it/products/workstation-player. Una volta installato, viene scaricata l'immagine di Ubuntu 23.10.1 dal sito ufficiale [8]. Per installare il sistema operativo e creare la macchina virtuale, basta eseguire pochi semplici passaggi guidati, come riportato nella guida ufficiale [9].

4.4.2 Installazione broker MQTT

La scelta del broker MQTT è ricaduta sul software “Mosquitto”. Di seguito, verrà illustrata la procedura di installazione di Mosquitto all’interno del sistema operativo Ubuntu [5].

1. Aprire il terminale e digitare il seguente comando per installare i pacchetti **mosquitto** e **mosquitto-clients**:

```
sudo apt install mosquitto mosquitto-clients
```

2. Una volta completata l’installazione, verificare che il servizio sia attivo con il seguente comando:

```
sudo systemctl status mosquitto
```

3. Se non dovesse risultare attivo, attraverso il seguente comando si renderebbe tale:

```
sudo systemctl enable mosquitto
```

4.4.3 Installazione Node-RED

Per installare Node-RED si può procedere semplicemente tramite lo store ufficiale di Ubuntu (Ubuntu Software Center), come mostrato in figura 29. Cercare il pacchetto “node red” quindi fare click su “installa”.

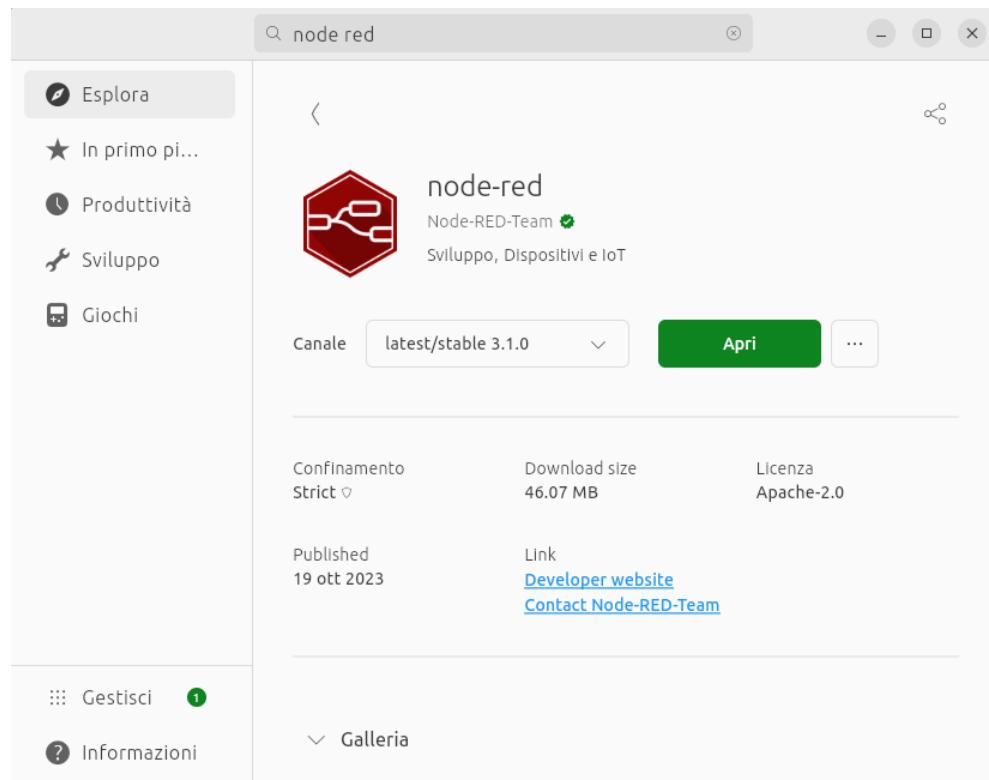


Figura 29: Installazione software Node-RED

4.4.4 Installazione MySQL

Per memorizzare i dati ricevuti dal rover, è stato scelto il DBMS MySQL. Per installarlo sulla macchina virtuale Ubuntu, sono stati eseguiti i seguenti passaggi:

1. Aprire il terminale e digitare il comando:

```
sudo apt install mysql-server
```

2. Avviare il servizio con il comando:

```
sudo systemctl start mysql.service
```

3. Avviare la prompt di MySQL con il comando:

```
sudo mysql
```

4. Utilizzare il comando ALTER USER per impostare la password per l'utente root:

```
mysql> ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY 'pass';
```

5. Uscire dal prompt:

```
mysql> exit;
```

6. Eseguire lo script per la configurazione in sicurezza:

```
sudo mysql_secure_installation
```

7. Fare il login con l'utente root e successivamente inserire la password appena impostata, con il comando:

```
mysql -u root -p
```

8. Ora creare un nuovo utente per nodered:

```
mysql> CREATE USER 'nodered'@'localhost' IDENTIFIED BY 'n0dep4assw0rd';
```

9. Assegniamo i privilegi all'utente al database desiderato:

```
mysql> GRANT PRIVILEGE ON *.* TO 'nodered'@'localhost';
```

10. Apportare le modifiche con il comando:

```
mysql> FLUSH PRIVILEGES;
```

11. Uscire quindi dalla console di MySQL:

```
mysql> exit;
```

12. Accedere con il nuovo utente:

```
mysql -u username -p
```

13. Per controllare lo stato del servizio utilizzare il comando:

```
systemctl status mysql.service
```

Per poter visualizzare ed interagire con il database con una interfaccia grafica, è stato installato anche il software “MySQL Workbench” dallo store ufficiale di Ubuntu (Ubuntu Software Center), come mostrato in figura 30.

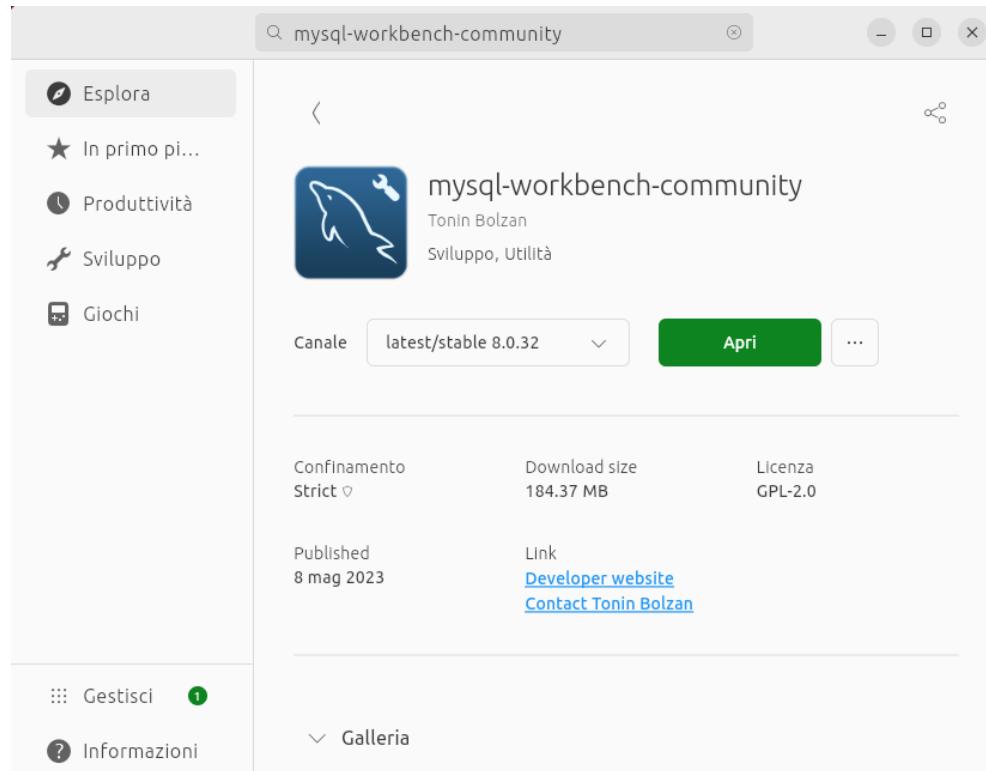


Figura 30: Installazione software MySQL Workbench

Una volta inseriti i parametri di connessione, è stato creato il database eseguendo la query:

```
CREATE DATABASE nodered;
```

Per creare poi la tabella per contenere tutte le misurazioni effettuate dal rover, viene eseguito il seguente comando DDL:

```
CREATE TABLE `nodered`.`sensors_history` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `distance` DOUBLE NULL,
  `acceleration_x` DOUBLE NULL,
  `acceleration_y` DOUBLE NULL,
  `acceleration_z` DOUBLE NULL,
  `rotation_x` DOUBLE NULL,
  `rotation_y` DOUBLE NULL,
  `rotation_z` DOUBLE NULL,
  `timestamp` DATETIME NULL,
```

```

`temperature` DOUBLE NULL,
`humidity` DOUBLE NULL,
`battery_voltage` DOUBLE NULL,
PRIMARY KEY (`id`),
UNIQUE INDEX `id_UNIQUE` (`id` ASC) VISIBLE;

```

Si otterrà quindi una tabella con il diagramma UML come mostrato in figura 31.

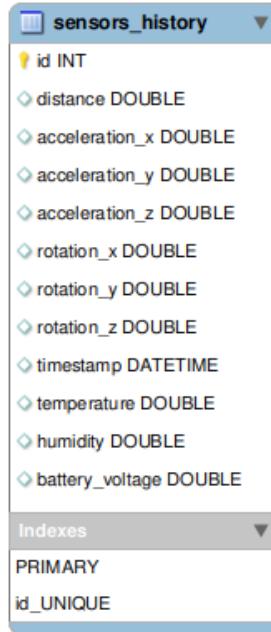


Figura 31: Diagramma UML database

4.4.5 Sviluppo interfaccia Node-RED

Una volta installato Node-RED, sarà accessibile sul browser all'indirizzo predefinito:

<http://localhost:1880>

Il primo passo, è quello di installare i componenti aggiuntivi che verranno utilizzati nel progetto, ovvero le librerie:

- node-red-dashboard
- node-red-node-mysql

Per fare questo, aprire il menu in alto a destra e cliccare su “Manage palette” come mostrato in figura 32.

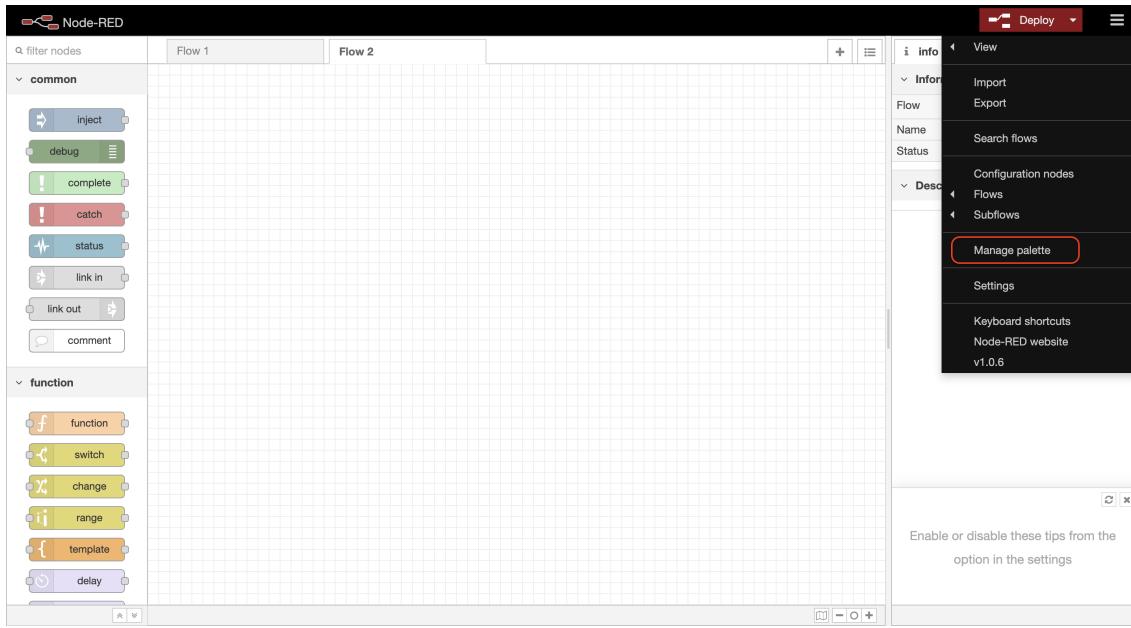


Figura 32: Pulsante “Manage palette”

Verrà visualizzata la schermata “User settings”, fare clic sulla scheda “Install” (1) e digitare “node-red-dashboard” (2) nella barra di ricerca. Verrà visualizzato il modulo node-red-dashboard e si potrà fare click su “Install” (3). Il passaggio è illustrato in figura 33

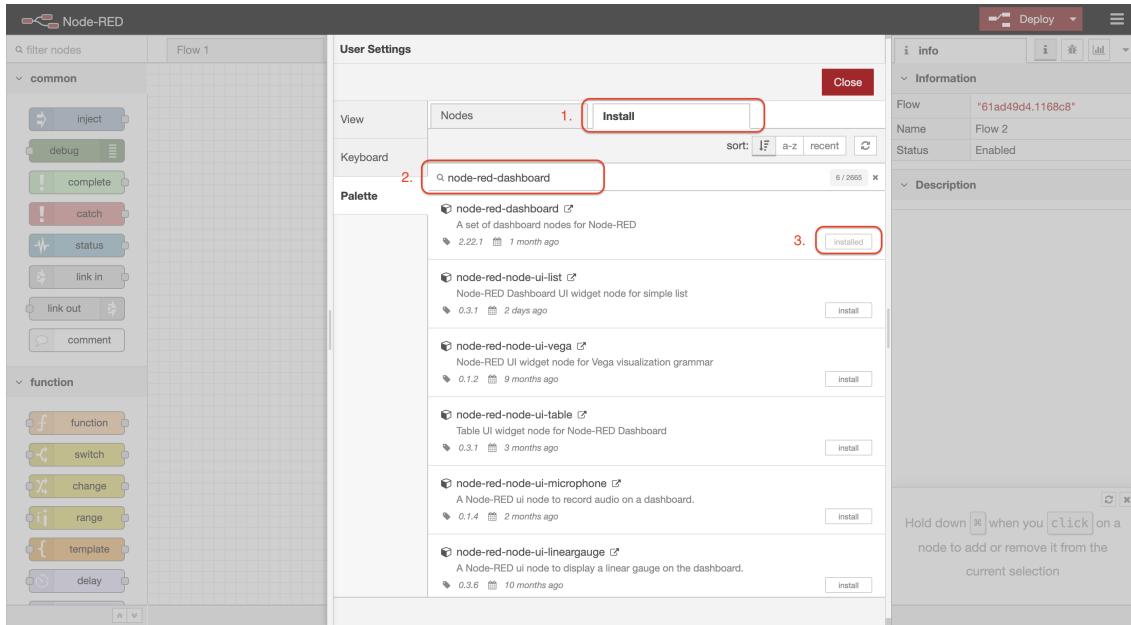


Figura 33: Installazione “node-red-dashboard”

Ripetere quest’ultimo passaggio anche per il pacchetto “node-red-node-mysql”. Adesso si potrà procedere allo sviluppo trascinando i blocchetti presenti nella barra laterale di sinistra.

La prima parte dello sviluppo, riguarda il flusso di dati che parte dalla ricezione di un messaggio MQTT. Viene mostrata l'implementazione in figura 34.

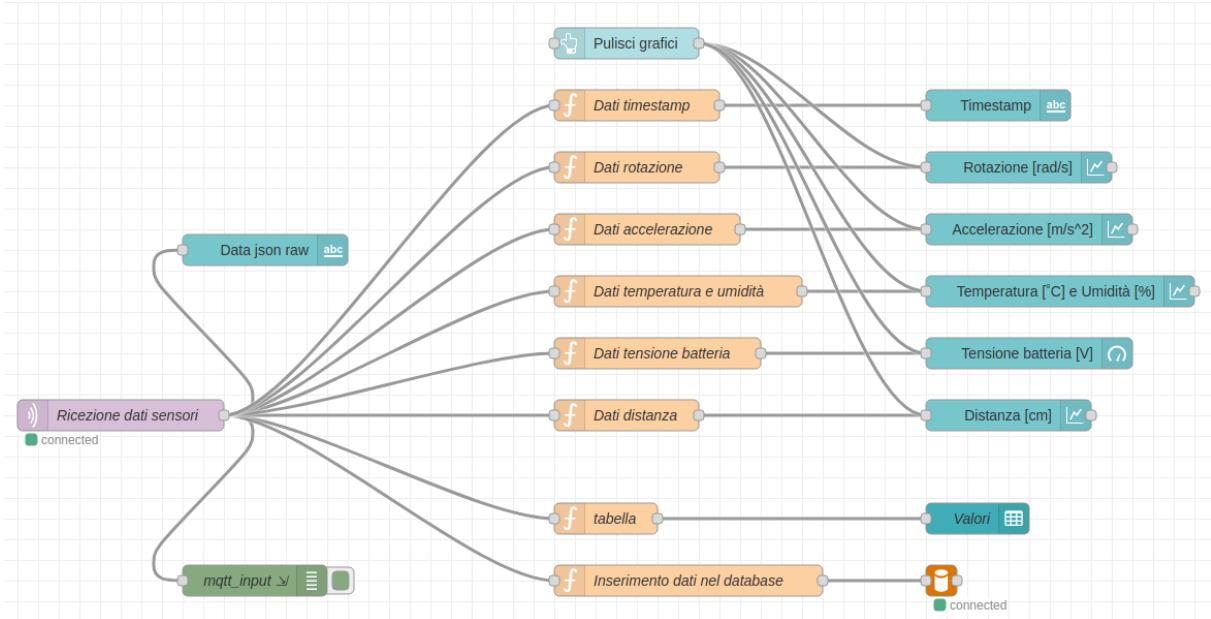


Figura 34: Sviluppo prima parte Node-RED

Viene utilizzato come nodo di partenza “mqtt in”, configurato opportunamente per potersi collegare al broker inserendo l’indirizzo del server (in questo caso localhost alla porta di default 1883) il topic a cui sottoscriversi (in questo caso “rover_output”) e il livello di QoS (in questo caso a valore 1). La configurazione è mostrata in figura 35.

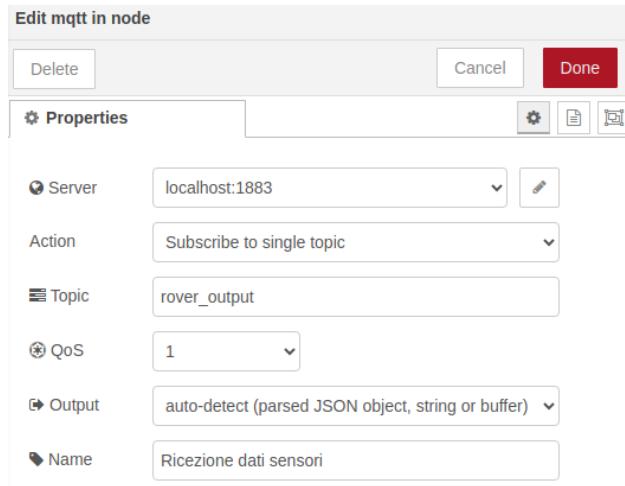


Figura 35: Configurazione del nodo “mqtt in”

Viene aggiunto poi un “debug” node collegato all’uscita del precedente nodo, utile per visualizzare nella finestra di debug l’output del “mqtt in”. Il nodo sarà configurato come mostrato in figura 36.

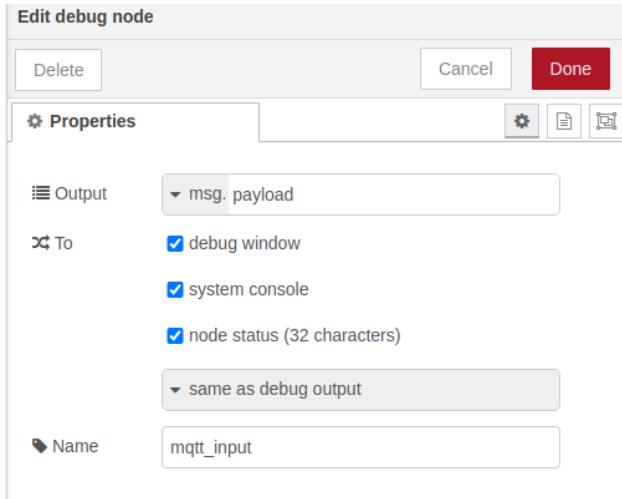


Figura 36: Configurazione del nodo “debug”

Per visualizzare l’output anche nella interfaccia utente, viene aggiunto il nodo “text” che collegato all’uscita del nodo “mqtt in” non ha bisogno di altre configurazioni. Per poter inserire i dati ricevuti all’interno del database, si utilizza il nodo “mysql”. Nella configurazione, viene aggiunto il database con i parametri di connessione come segue in figura 37.

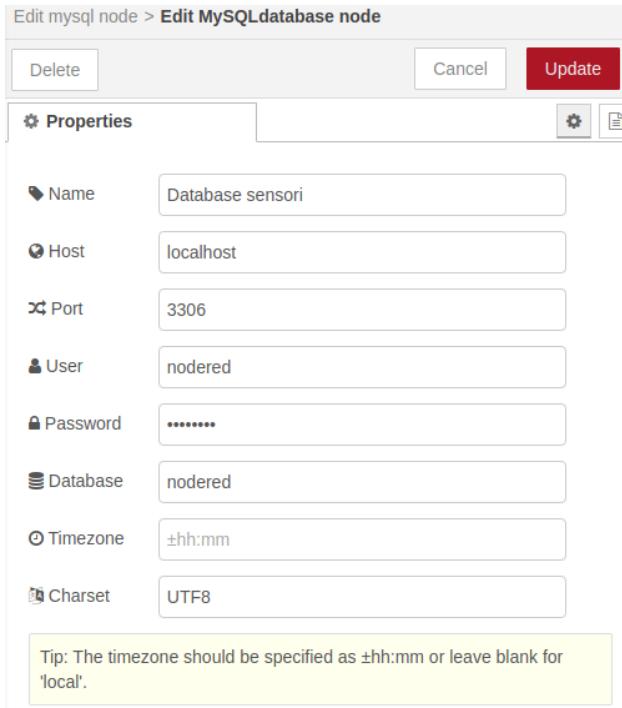


Figura 37: Configurazione della connessione al database

Fra il nodo del database e il nodo mqtt, viene aggiunto un blocco “function” per costruire la stringa della query a partire dai dati ricevuti dal rover. Infatti, nella tab “On Message” del pannello di configurazione

della funzione troviamo il seguente codice Javascript:

```
msg.topic = "INSERT INTO `nodered`.`sensors_history` (`distance`, `  
    ↪ acceleration_x`, `acceleration_y`, `acceleration_z`, `rotation_x`, `'  
    ↪ rotation_y`, `rotation_z`, `timestamp`, `temperature`, `humidity`, `'  
    ↪ battery_voltage`) " +  
" VALUES (" +  
msg.payload.distance+", "+  
msg.payload.acceleration.x + ", " +  
msg.payload.acceleration.y + ", " +  
msg.payload.acceleration.z + ", " +  
msg.payload.rotation.x + ", " +  
msg.payload.rotation.y + ", " +  
msg.payload.rotation.z + ", " +  
" FROM_UNIXTIME(" + msg.payload.unixtime + "), " +  
msg.payload.temperature + ", " +  
msg.payload.humidity + ", " +  
msg.payload.battery.voltage +  
");"  
return msg;
```

Per visualizzare i dati in formato tabellare nell’interfaccia utente, viene utilizzato il nodo “Table”. Anche qui, viene inserito un blocco “function” tra il nodo mqtt e il blocco tabella con il seguente codice per costruire la riga con le intestazioni:

```
msg.payload=[  
{  
"Timestamp (unixtime) [s]":msg.payload.unixtime,  
"Temperatura [C)": msg.payload.temperature,  
"Accelerazione X[m/s^2)": msg.payload.acceleration.x,  
"Accelerazione Y[m/s^2)": msg.payload.acceleration.y,  
"Accelerazione Z[m/s^2)": msg.payload.acceleration.z,  
"Rotazione X[rad/s)": msg.payload.rotation.x,  
"Rotazione Y[rad/s)": msg.payload.rotation.y,  
"Rotazione Z[rad/s)": msg.payload.rotation.z,  
"Umidità [%)": msg.payload.humidity,  
"Batteria [V)": msg.payload.battery.voltage,  
"Distanza [cm)": msg.payload.distance  
}];  
return msg;
```

Per visualizzare i dati in modo più interattivo, sono stati utilizzati alcuni widget, come i nodi “chart” e “gauge”. Per ogni nodo, è stata posta prima un blocco “function” per selezionare correttamente il tipo di dato da inviare al grafico. Ad esempio, nella funzione per il gauge che mostra la tensione della batteria è stato inserito il seguente codice:

```
msg.payload = msg.payload.battery.voltage;  
return msg;
```

Allo stesso modo, nel chart che visualizza con due linee diverse la temperatura e l’umidità, è stato inserito il seguente codice:

```

var msg1 = {};
var msg2 = {};

msg1.payload = msg.payload.temperature;
msg1.topic = 'Temperatura';
msg2.payload = msg.payload.humidity;
msg2.topic = 'Umidita';

return [[msg1,msg2]]

```

Viene poi inserito un pulsante per pulire i grafici dai dati storici, ovvero un nodo “button” configurato in modo che al click invi un payload con un array vuoto.

Nella seconda parte dello sviluppo, mostrato in figura 38, viene implementata la visualizzazione dello storico dei dati ricevuti dal rover.

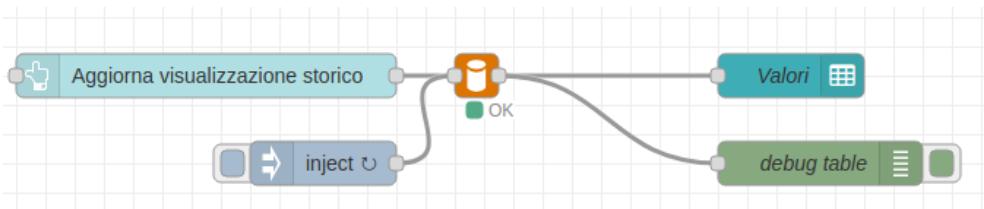


Figura 38: Sviluppo seconda parte Node-RED

Per fare questo, è stato inserito un nodo “button” che invi al database (ovvero il nodo “mysql”) la seguente query:

```

SELECT
`sensors_history`.`id` AS 'ID',
`sensors_history`.`distance` AS 'Distanza [cm]',
`sensors_history`.`acceleration_x` AS 'Accelerazione X [m/s^2]',
`sensors_history`.`acceleration_y` AS 'Accelerazione Y [m/s^2]',
`sensors_history`.`acceleration_z` AS 'Accelerazione Z [m/s^2]',
`sensors_history`.`rotation_x` AS 'Rotazione X [rad/s]',
`sensors_history`.`rotation_y` AS 'Rotazione Y [rad/s]',
`sensors_history`.`rotation_z` AS 'Rotazione Z [rad/s]',
`sensors_history`.`timestamp` AS 'Timestamp (unixtime) [s]',
`sensors_history`.`temperature` AS 'Temperatura [C]',
`sensors_history`.`humidity` AS 'Umidita [%]',
`sensors_history`.`battery_voltage` AS 'Batteria [V]'
FROM `nodered`.`sensors_history`
ORDER BY ID;

```

Il risultato della query, viene dirottato sul nodo “table” che visualizzerà tutti i dati disponibili nel database. Questa operazione viene ripetuta comunque ad intervalli regolari per aggiornare i dati visualizzati, attraverso il nodo “inject” che è configurato per inviare la stessa query al database ogni minuto. Le proprietà del nodo sono mostrate in figura 39.



Figura 39: Configurazione nodo “inject”

Nella terza e ultima parte dello sviluppo a blocchi, mostrata in figura 40, si gestisce tutta la comunicazione che va dall’interfaccia web verso il rover.

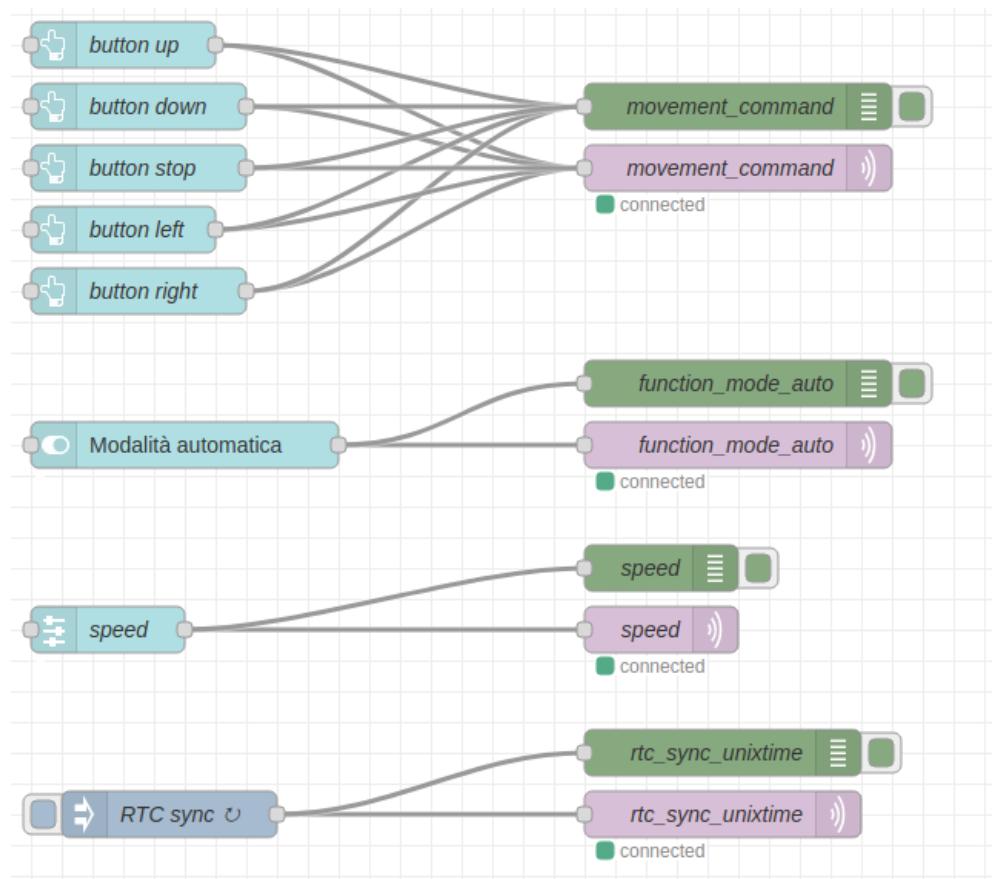


Figura 40: Sviluppo terza parte Node-RED

Per i comandi di movimento, si utilizzano 5 pulsanti (4 per le direzioni e 1 di stop) configurati per inviare un numero intero che verrà poi decifrato con un determinato mapping dall'ESP32. In figura 41 si mostra la configurazione del pulsante per voltare a sinistra.

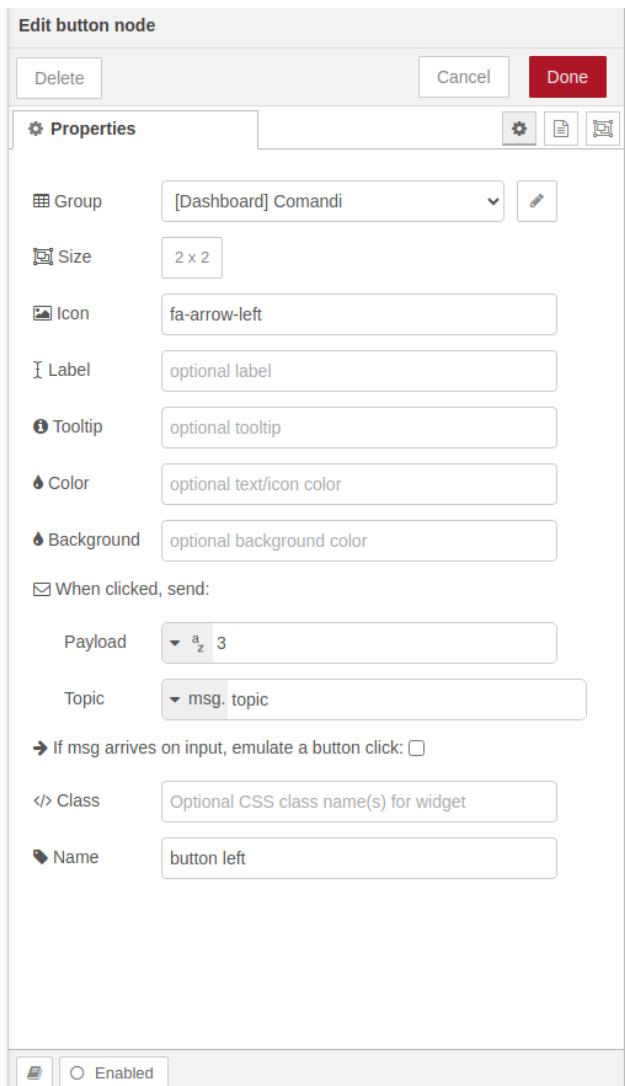


Figura 41: Configurazione pulsante movimento sinistra

L'output del pulsante viene collegato al nodo "mqtt out" che si occupa di fare una "publish" del messaggio nel topic configurato (in questo caso "movement_command"). Vengono impostati anche l'indirizzo del broker mqtt e la QoS. In figura 42 si mostra la configurazione del nodo mqtt.

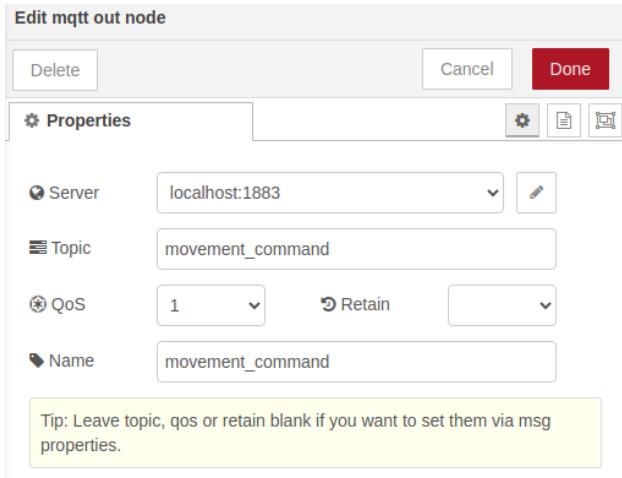


Figura 42: Configurazione del nodo “mqtt out”

Allo stesso modo, vengono implementate anche le comunicazioni per la modalità di funzionamento e la velocità. La modalità di funzionamento invia un booleano nel topic “function_mode_auto” mentre per la velocità viene inviato un intero nel topic “speed”.

Per la sincronizzazione dell’orologio, viene utilizzato un nodo “inject” impostato per inviare il timestamp in formato unixtime all’avvio del sistema e a intervalli regolari di 60 minuti. La configurazione del nodo viene mostrata in figura 43. L’uscita del blocco viene poi dirottata sempre al nodo “mqtt out” che farà una publish mqtt con topic “rtc_sync.unixtime”.

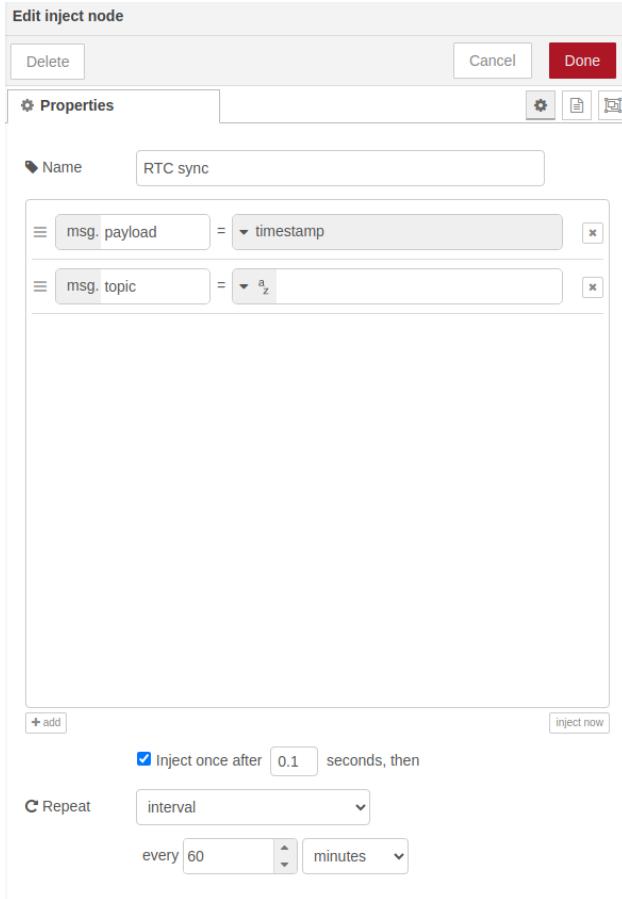


Figura 43: Configurazione del nodo “inject” per l’RTC sync

Per la progettazione del layout grafico vi è una sezione apposita nel menù a sinistra. In questo caso, come mostrato in figura 44, sono state create due pagine: una per visualizzare i dati ricevuti in real time e comandare il rover, e una in cui si visualizza lo storico dei dati. Le dashboard vengono progettate trascinando e ridimensionando gli elementi nella griglia.

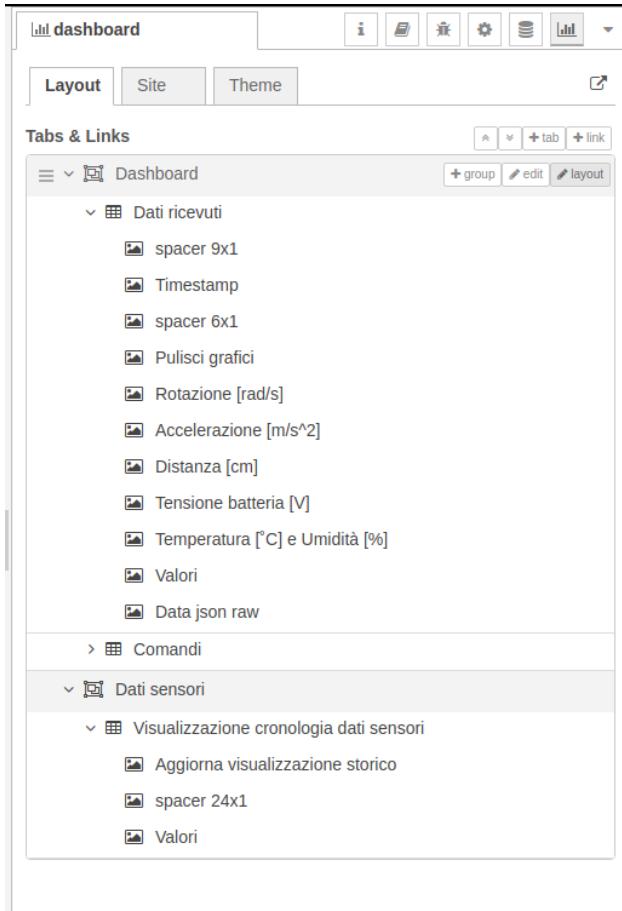


Figura 44: Tab progettazione layout

Nella prima schermata, sono state fatte due tab: una in cui si visualizzano i dati ricevuti real-time con i relativi grafici, e una in cui vi sono le azioni per comandare il rover. La progettazione è mostrata in figura 45.

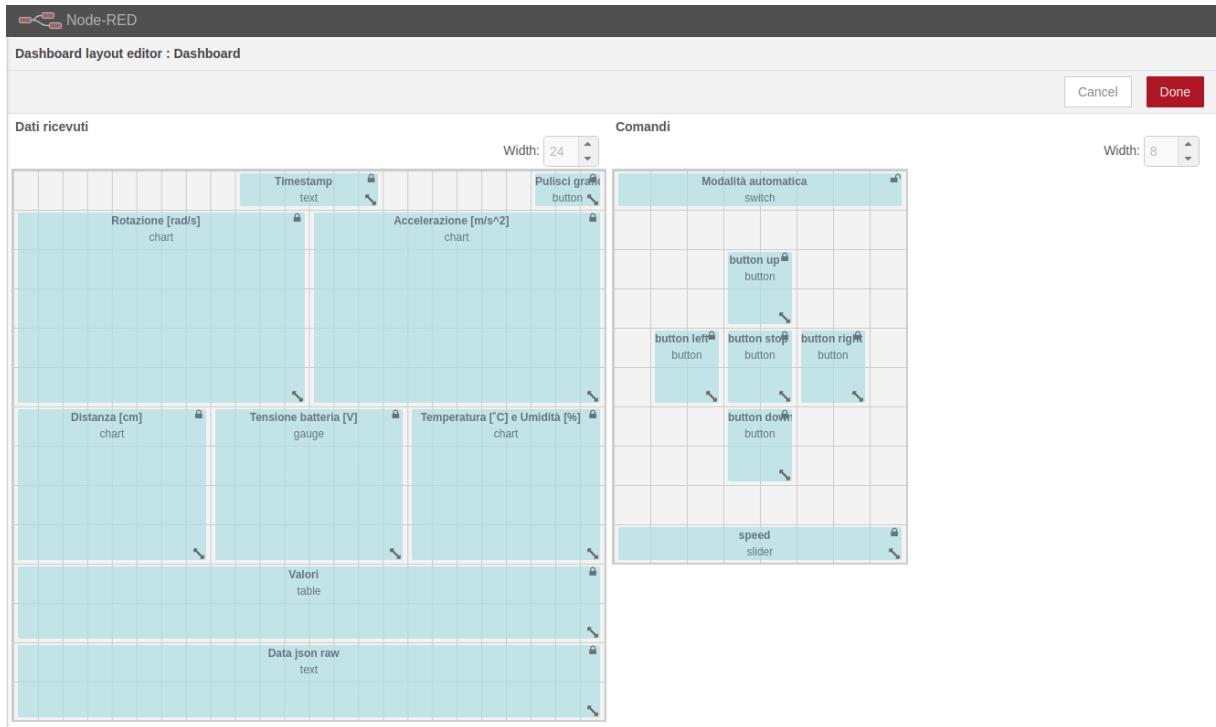


Figura 45: Progettazione layout dashboard

Nella seconda schermata, quella dedicata allo storico dei dati, si visualizza un semplice pulsante per l'aggiornamento e la tabella popolata nel database. La progettazione è mostrata in figura 46.

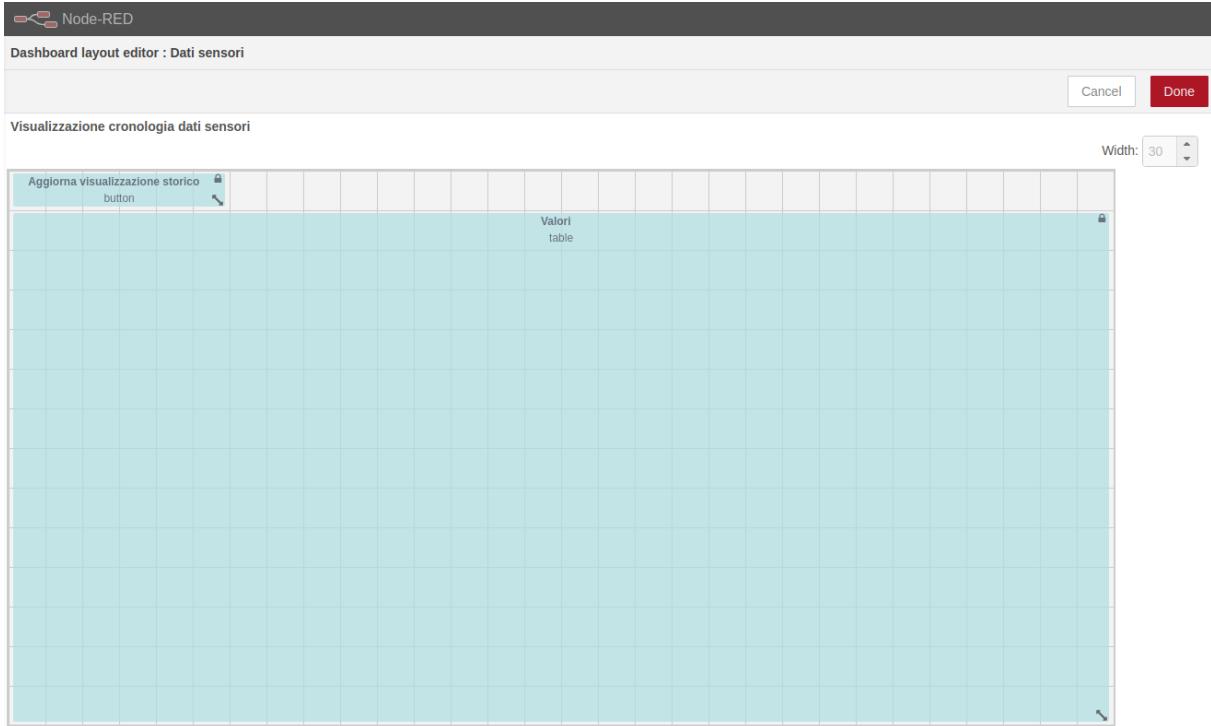


Figura 46: Progettazione layout storico dati

Le interfacce che visualizzerà l'utente nella prima e seconda schermata sono rappresentate rispettivamente nelle figure 47 e 48.

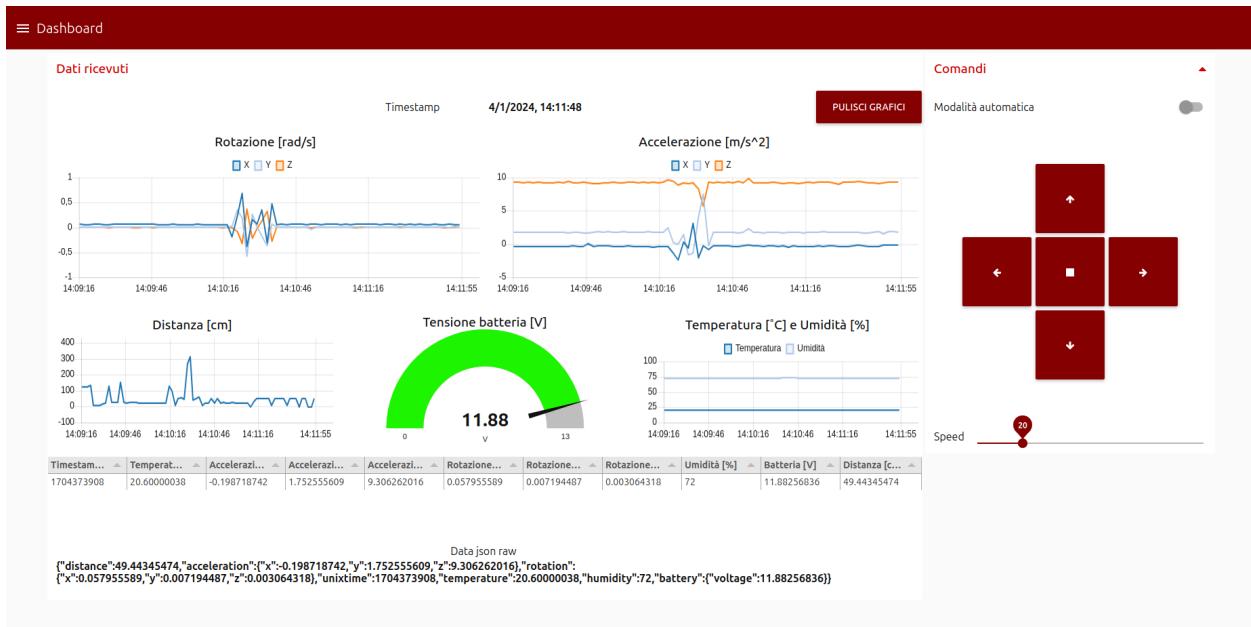


Figura 47: Prima schermata utente

☰ Dati sensori

Visualizzazione cronologia dati sensori

AGGIORNA VISUALIZZAZIONE STORICO

ID	Distanza [cm]	Accelerazion...	Accelerazion...	Accelerazion...	Rotazione X ...	Rotazione Y ...	Rotazione Z ...	Timestamp ...	Temperatura...	Umidità [%]	Batteria [V]
1	42.63490295	-0.620098233	0.268150598	9.423366661	0.059820827	0.007460949	-0.000266462	2023-12-27T16:1...	20.60000038	70	3.38671875
2	42.63490295	-0.620098233	0.268150598	9.423366661	0.059820827	0.007460949	-0.000266462	2023-12-27T16:1...	20.60000038	70	3.38671875
3	42.63490295	-0.620098233	0.268150598	9.423366661	0.059820827	0.007460949	-0.000266462	2023-12-27T16:1...	20.60000038	70	3.38671875
4	42.63490295	-0.620098233	0.268150598	9.423366661	0.059820827	0.007460949	-0.000266462	2023-12-27T16:1...	20.60000038	70	3.38671875
5	42.63490295	-0.620098233	0.268150598	9.423366661	0.059820827	0.007460949	-0.000266462	2023-12-27T16:1...	20.60000038	70	3.38671875
6	42.63490295	-0.620098233	0.268150598	9.423366661	0.059820827	0.007460949	-0.000266462	2023-12-27T16:1...	20.60000038	70	3.38671875
7	42.63490295	-0.620098233	0.268150598	9.423366661	0.059820827	0.007460949	-0.000266462	2023-12-27T16:1...	20.60000038	70	3.38671875
8	42.63490295	-0.620098233	0.268150598	9.423366661	0.059820827	0.007460949	-0.000266462	2023-12-27T16:1...	20.60000038	70	3.38671875
9	42.63490295	-0.620098233	0.268150598	9.423366661	0.059820827	0.007460949	-0.000266462	2023-12-27T16:1...	20.60000038	70	3.38671875
10	42.63490295	-0.620098233	0.268150598	9.423366661	0.059820827	0.007460949	-0.000266462	2023-12-27T16:1...	20.60000038	70	3.38671875
11	42.63490295	-0.620098233	0.268150598	9.423366661	0.059820827	0.007460949	-0.000266462	2023-12-27T16:1...	20.60000038	70	3.38671875
12	42.63490295	-0.620098233	0.268150598	9.423366661	0.059820827	0.007460949	-0.000266462	2023-12-27T16:1...	20.60000038	70	3.38671875
13	42.63490295	-0.620098233	0.268150598	9.423366661	0.059820827	0.007460949	-0.000266462	2023-12-27T16:1...	20.60000038	70	3.38671875
14	42.63490295	-0.620098233	0.268150598	9.423366661	0.059820827	0.007460949	-0.000266462	2023-12-27T16:1...	20.60000038	70	3.38671875
15	42.63490295	-0.620098233	0.268150598	9.423366661	0.059820827	0.007460949	-0.000266462	2023-12-27T16:1...	20.60000038	70	3.38671875
16	42.63490295	-0.620098233	0.268150598	9.423366661	0.059820827	0.007460949	-0.000266462	2023-12-27T16:1...	20.60000038	70	3.38671875
17	42.63490295	-0.620098233	0.268150598	9.423366661	0.059820827	0.007460949	-0.000266462	2023-12-27T16:1...	20.60000038	70	3.38671875
18	42.63490295	-0.620098233	0.268150598	9.423366661	0.059820827	0.007460949	-0.000266462	2023-12-27T16:1...	20.60000038	70	3.38671875
19	42.63490295	-0.620098233	0.268150598	9.423366661	0.059820827	0.007460949	-0.000266462	2023-12-27T16:1...	20.60000038	70	3.38671875
20	42.63490295	-0.620098233	0.268150598	9.423366661	0.059820827	0.007460949	-0.000266462	2023-12-27T16:1...	20.60000038	70	3.38671875
21	42.63490295	-0.620098233	0.268150598	9.423366661	0.059820827	0.007460949	-0.000266462	2023-12-27T16:1...	20.60000038	70	3.38671875
22	42.63490295	-0.620098233	0.268150598	9.423366661	0.059820827	0.007460949	-0.000266462	2023-12-27T16:1...	20.60000038	70	3.38671875
23	42.63490295	-0.620098233	0.268150598	9.423366661	0.059820827	0.007460949	-0.000266462	2023-12-27T16:1...	20.60000038	70	3.38671875
24	42.63490295	-0.620098233	0.268150598	9.423366661	0.059820827	0.007460949	-0.000266462	2023-12-27T16:1...	20.60000038	70	3.38671875
25	42.63490295	-0.620098233	0.268150598	9.423366661	0.059820827	0.007460949	-0.000266462	2023-12-27T16:1...	20.60000038	70	3.38671875
26	42.63490295	-0.620098233	0.268150598	9.423366661	0.059820827	0.007460949	-0.000266462	2023-12-27T16:1...	20.60000038	70	3.38671875
27	42.63490295	-0.620098233	0.268150598	9.423366661	0.059820827	0.007460949	-0.000266462	2023-12-27T16:1...	20.60000038	70	3.38671875
28	42.63490295	-0.620098233	0.268150598	9.423366661	0.059820827	0.007460949	-0.000266462	2023-12-27T16:1...	20.60000038	70	3.38671875

Figura 48: Seconda schermata utente

Lo sviluppo dell'interfaccia web tramite Node-RED è stato esportato in formato JSON ed è consultabile nel repository github [7].

5 Conclusioni e implementazioni future

In conclusione, il progetto ha portato alla realizzazione di un rover controllato da un sistema complesso, basato su ESP32 e Raspberry Pi, che interagiscono attraverso la comunicazione seriale e MQTT. La presenza di sensori e l'interfaccia web forniscono un controllo avanzato e una ricca acquisizione di dati per il rover. Durante lo sviluppo, sono state affrontate sfide significative, come la gestione dei task su FreeRTOS e l'integrazione dei vari componenti hardware e software.

I risultati ottenuti dimostrano il funzionamento coerente del rover in entrambe le modalità di funzionamento: autonoma e manuale. L'interfaccia web fornisce un mezzo intuitivo per il controllo del rover, mentre i sensori integrati permettono un monitoraggio dettagliato dell'ambiente circostante.

Per migliorare ulteriormente il progetto, alcune possibili implementazioni future potrebbero includere:

- Rendere più sicure le informazioni scambiate con il protocollo MQTT mediante crittografia TLS/SSL.
- Aggiunta di algoritmi di navigazione avanzati: implementare algoritmi di navigazione avanzati per consentire al rover di compiere decisioni più intelligenti durante la modalità autonoma.
- Espansione della quantità di sensori: integrare ulteriori sensori per migliorare la capacità di percezione dell'ambiente circostante.
- Sviluppo di funzionalità aggiuntive dell'interfaccia web: utilizzare una interfaccia che permetta di comandare il rover mediante le frecce della tastiera o di fermare il rover al rilascio del pulsante di movimento.

In definitiva, il progetto ha fornito una solida base per ulteriori sviluppi e miglioramenti. L'esperienza acquisita durante questa fase costituirà una risorsa preziosa per futuri progetti embedded e sistemi IoT.

Riferimenti bibliografici

- [1] Raspberry pi OS: raspberrypi.com/software
- [2] FreeRTOS: freertos.org
- [3] Node-Red: nodered.org
- [4] MySQL: mysql.com/it
- [5] Mosquitto broker: mosquitto.org
- [6] Freertos task priority: freertos.org/RTOS-task-priority
- [7] Repository github: github.com/denilnicolosi/Rover_IoT
- [8] Download immagine ubuntu 23: ubuntu-it.org/download
- [9] Guida ufficiale creazione macchina virtuale: kb.vmware.com/s/article/1018415