

TESI DI LAUREA IN
INTERNET OF THINGS, BIG DATA E MACHINE LEARNING

Progettazione e sviluppo di un UGV a controllo remoto per la sorveglianza urbana

CANDIDATO

Marco Catanzaro

RELATORE

Prof. Ivan Scagnetto

CORRELATORE

Dott. Denis Tavaris

CONTATTI DELL'ISTITUTO

Dipartimento di Scienze Matematiche, Informatiche e Fisiche

Università degli Studi di Udine

Via delle Scienze, 206

33100 Udine — Italia

+39 0432 558400

<https://www.dmif.uniud.it/>

Ai miei cari.

Sommario

Uno dei temi più sentiti nella società contemporanea è quello della sicurezza. Le persone desiderano sentirsi protette, specialmente durante la notte o nei periodi di assenza dalla propria abitazione o attività commerciale. Inoltre in caso di effrazione, si aspettano un rapido intervento da parte delle autorità e la possibilità di rintracciare il responsabile con la refurtiva.

Tuttavia, le forze dell'ordine non riescono sempre a garantire una presenza capillare sul territorio, e non sono rari i casi di furti e rapine in cui il colpevole rimanga ignoto. Questa difficoltà è spesso legata all'assenza di un adeguato sistema di videosorveglianza pubblica, fondamentale per raccogliere elementi utili come l'identificazione del volto, le caratteristiche dei veicoli coinvolti e il percorso seguito durante la fuga.

Coprire ampie aree urbane, come interi quartieri, richiederebbe un elevato numero di dispositivi statici, con costi significativi, alta vulnerabilità a danneggiamenti e una copertura non sempre efficace e completa. Per questo motivo, ho scelto di analizzare e sperimentare un'alternativa mobile: la realizzazione di un drone terrestre, noto anche come UGV (Unmanned Ground Vehicle), in grado di operare tramite controllo remoto e, in prospettiva futura, gestito tramite intelligenza artificiale. L'obiettivo è creare un sistema di sorveglianza dinamico, capace di pattugliare autonomamente determinate aree, fornire dati in tempo reale, segnalare tempestivamente la situazione alle autorità e raccogliere tracce utili alle indagini.

Nella fase finale del progetto invece, mi dedicherò allo studio di tecnologie più sofisticate, in particolare l'integrazione del sistema Navio2 abbinato al firmware ArduRover, con obiettivo finale il confronto dei pregi e difetti dei due approcci.

Indice

1 Stato dell'arte	1
1.1 Sistemi IoT nel quotidiano e per la difesa	1
1.1.1 Applicazione nel settore pubblico	1
1.1.2 Applicazioni nel campo militare	2
1.2 Sistemi IoT per la sorveglianza urbana	5
1.2.1 Sicurezza collettiva vs libertà individuali	6
1.2.2 Crime Detection	6
1.3 Veicoli senza equipaggio	8
1.3.1 Nomenclatura ufficiale per gli <i>UV</i> (Unmanned Vehicle):	8
1.3.2 UGV – Veicoli terrestri senza equipaggio	8
1.4 Aspetti normativi legati agli UGV	13
1.4.1 Veicoli aerei vs veicoli terrestri	13
1.4.2 Normativa sulla privacy e registrazione tramite telecamere	13
1.4.3 Normativa sull'utilizzo della comunicazione wireless	13
2 Progettazione di UV	15
2.1 Progettazione dell' UGV	17
2.2 Componenti Principali	18
2.2.1 Struttura esterna	18
2.2.2 Motori	20
2.2.3 Driver Motori	22
2.2.4 Alimentazione	26
2.3 Sensoristica	30

2.3.1	Rilevatore di tensione e convertitore analogico-digitale	30
2.3.2	Accelerometro	32
2.3.3	Sensore a ultrasuoni	34
2.3.4	Buzzer acustico	36
2.3.5	Streaming audio e video	38
2.3.6	GPS	44
2.4	Microcontrollore (MCU)	47
2.4.1	Criteri di selezione	47
2.4.2	Valutazione dei microcontrollori	47
2.4.3	Valutazione dei microcomputer	49
2.4.4	Raspberry pi 3B +	50
2.4.5	GPIO e Pinout del Raspberry Pi	52
2.4.6	Circuito finale	53
2.5	Stack di Comunicazione	55
2.5.1	Livello Fisico e Datalink	55
2.5.2	Livello di Rete	57
2.5.3	Livello di Trasporto	58
2.5.4	Livello di Applicazione	60
2.5.5	Mantenimento e rappresentazione del dato	64
2.6	Codice e Architettura di sistema	68
2.6.1	Architettura di sistema	68
2.6.2	Interfaccia Grafica (GUI)	69
2.6.3	ManagerDati	71
2.6.4	Lettura e visualizzazione dei dati	73
2.6.5	Movimento del veicolo – lato PC	74
2.7	Analisi del Codice Lato Raspberry	76
2.7.1	Movimento del Veicolo	76
2.7.2	Codice Sensori	78

3.0.1	Cos'è Navio2	81
3.0.2	ArduRover: il software	81
3.0.3	Considerazioni finali	82
4	Conclusione	83
	Bibliografia	84

1

Stato dell'arte

1.1 Sistemi IoT nel quotidiano e per la difesa

Come ben sappiamo, moltissime delle tecnologie che utilizziamo quotidianamente sono state inizialmente sviluppate per scopi bellici. È questo il caso di molti sistemi IoT, particolarmente utili quando è necessario coordinare e gestire in modo efficiente grandi reti di dispositivi, risorse, persone e veicoli. L'enorme quantità di dati generata, se opportunamente analizzata e visualizzata, può offrire un vantaggio strategico determinante contro il nemico, contribuendo a ridurre i costi operativi e, soprattutto perdite umane.

L'Internet delle Cose (IoT) è descritto come: *"un sistema distribuito che crea valore a partire dai dati, permettendo a oggetti fisici eterogenei di condividere informazioni e coordinare decisioni"* [1].

1.1.1 Applicazione nel settore pubblico

Concentrandosi più in generale sull'impiego pubblico attuale di queste tecnologie, i vari Stati nel mondo cercano, ciascuno con le proprie risorse, di implementare diverse soluzioni tecnologiche (*illustrate in Figura 1.1*), con l'obiettivo di migliorare l'efficienza non solo in ambito difensivo, ma anche nella gestione delle **catastrofi ambientali**, nel controllo dello **spazio aereo**, nella **sicurezza pubblica** e nella prevenzione degli **incendi**, nella **gestione energetica**, nella logistica delle **catene di approvvigionamento** e nel **settore sanitario**.

In particolare, gli scenari più interessanti emergono quando, oltre alla raccolta continua e capillare di dati provenienti da sensori distribuiti sul territorio, viene integrato un sistema di elaborazione avanzata anche tramite **algoritmi di intelligenza artificiale**. Questo permette alle autorità di prendere decisioni rapide, accurate e basate su evidenze oggettive.

Caso di studio significativo: [4] un sofisticato sistema di allerta per **incendi boschivi**, che sfrutta sensori di temperatura, umidità, velocità del vento e immagini satellitari per prevenire disastri su larga scala.

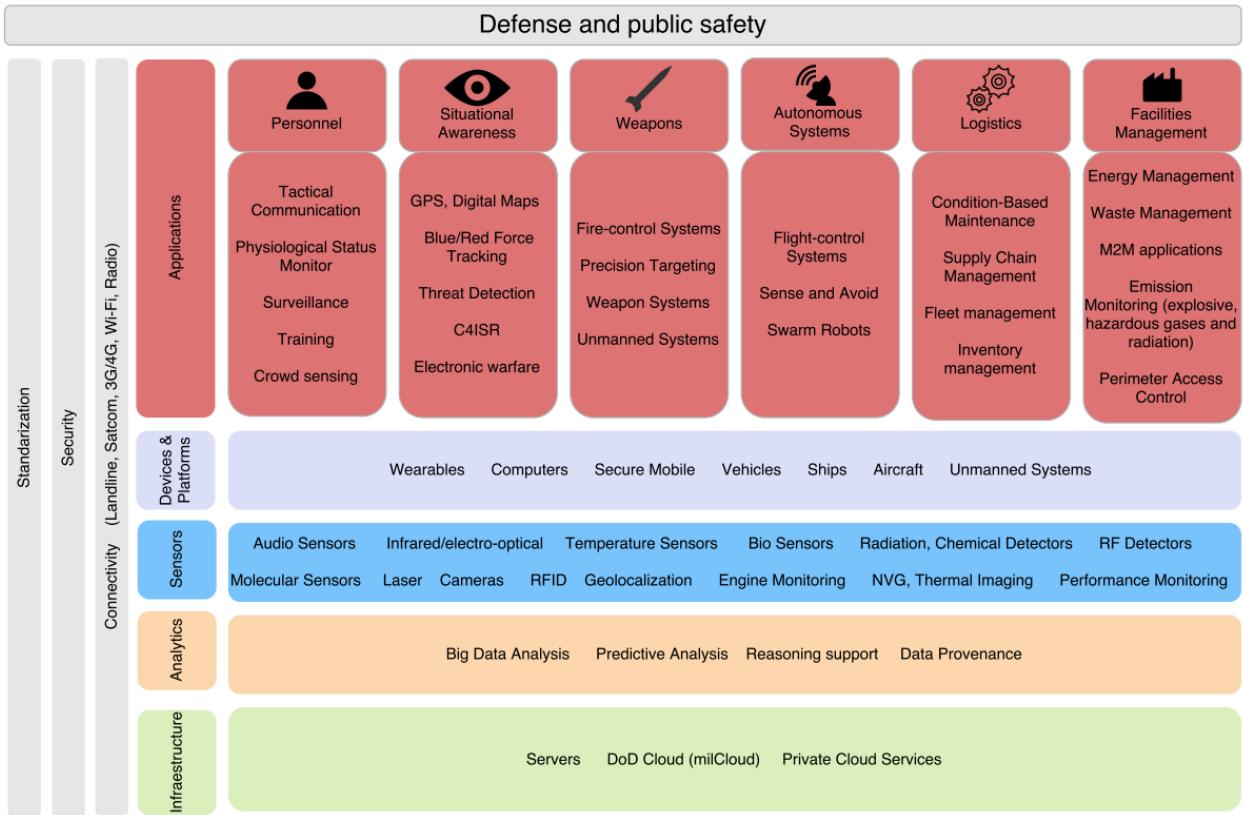


Figura 1.1: Tecnologie IoT applicate al settore pubblico [1]

1.1.2 Applicazioni nel campo militare

Possiamo affermare che, dopo la polvere da sparo, una delle innovazioni che ha realmente rivoluzionato il modo di fare la guerra siano state le **telecomunicazioni**. La possibilità di trasmettere informazioni in tempo reale, anche da territori nemici, ha rappresentato un vero *game changer* strategico. Questo ha permesso di abbandonare metodi obsoleti come i piccioni viaggiatori che, pur offrendo una buona "capacità di banda", soffriva di delay troppo elevati e scarsa affidabilità.

La prima applicazione significativa della radio in ambito bellico risale alla **Guerra russo-giapponese** del 1904-1905 [5], quando i russi utilizzarono comunicazioni via onde radio per coordinare le proprie operazioni navali. Tuttavia, fu durante la Prima Guerra Mondiale che la radio divenne uno strumento cruciale per il controllo delle truppe.

Le singole unità militari, composte da alcune decine di uomini, erano dotate di uno o più **addetti alle comunicazioni radio**, il cui compito consisteva nel trasportare e utilizzare l'apparato ricetrasmettente. All'epoca, questi dispositivi **pesavano diversi chilogrammi** e risultavano troppo ingombranti per essere distribuiti individualmente a ciascun soldato.

Questi operatori radio, seppur rudimentali, svolgevano un **ruolo analogo a quello dei sensori IoT** odierni: raccoglievano informazioni direttamente dal campo di battaglia e le trasmettevano ai comandi superiori. I generali, una volta ricevuti i dati, li visualizzavano su **mappe cartacee**, elaborando di conseguenza strategie e decisioni operative.

Al giorno d'oggi, grazie all'invenzione del transistor e alla **miniaturizzazione dell'elettronica**, per via delle ingenti risorse economiche investite nel settore militare, si è giunti allo sviluppo di sistemi estremamente complessi integrati direttamente su ciascun operatore. Attraverso **dispositivi indossabili** è possibile rilevare in tempo reale i **parametri biomedici** e le condizioni psicofisiche dei soldati, consentendo di identificare tempestivamente condizioni critiche come emorragie interne o attacchi di panico. L'intervento rapido basato su questi dati ha già permesso di salvare numerose vite umane.

Sono inoltre impiegati dispositivi integrati nell'equipaggiamento o nel casco che **potenziano i cinque sensi** dell'operatore, come telecamere termiche, sensori di movimento e microfoni ambientali. Quando applicati a **veicoli autonomi aerei o terrestri** (*figura 1.2*), tali sensori possono rilevare la presenza e la posizione di nemici sul campo di battaglia.



Figura 1.2: Cane robot, esercitazione congiunta Cina-Cambogia 2024 © Repubblica.it

Grazie a moderni sistemi di raccolta e analisi dei dati, è possibile monitorare l'**inventario individuale** del soldato, incluse provviste, acqua e munizioni, così da garantire rifornimenti di emergenza rapidi e mirati.

Non si può infine ignorare l'uso sempre più diffuso delle cosiddette armi intelligenti, come i **missili guidati** da crociera Tomahawk in dotazione agli Stati Uniti, o i sofisticati sistemi di intercettazione e **difesa antimissilistica** come l'Iron Dome israeliano.

Infine, anche la fase di **addestramento** beneficia delle tecnologie IoT: sistemi come lo **I-MILES** (*Instrumented Multiple Integrated Laser Engagement System*) [6] consentono di simulare campi di battaglia realistici tramite sensori, suoni, fumo e raggi laser, basandosi su dati raccolti da scenari reali, e fornendo statistiche post-esercitazione per migliorare l'efficacia formativa.

Per concludere, la Figura 1.3 confronta le tecnologie attualmente in dotazione ai soldati con quelle previste per un futuro prossimo. Questo schema evidenzia chiaramente l'evoluzione verso **una guerra sempre più digitalizzata** e guidata dai dati.

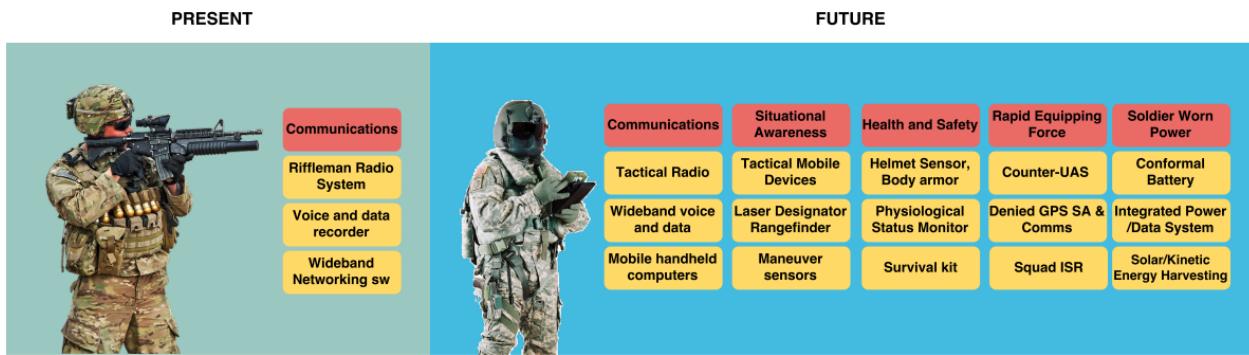


Figura 1.3: Soldati oggi vs. futuro [1]

1.2 Sistemi IoT per la sorveglianza urbana

I primi sistemi di sorveglianza, così come li intendiamo oggi, risalgono a un'epoca ben precedente alla nascita di Internet e dei computer. Già nel 1927, il celebre inventore sovietico **Léon Theremin**, noto anche per aver ideato il predecessore dei moderni sintetizzatori musicali, sviluppò per il Cremlino un sistema di sorveglianza analogico [8].

Il sistema si basava su una **telecamera meccanica**, nella quale la luce dell'immagine passava attraverso un disco rotante (Figura 1.4) dotato di fori disposti a spirale. A seconda della quantità di luce che attraversava ciascun foro, una **cellula fotoelettrica** posta dietro al disco generava un segnale elettrico proporzionale, che veniva trasmesso via radio verso un punto di osservazione sicuro.

Il ricevente disponeva di un sistema analogo, dotato di un disco gemello e di una **lampada al neon** al posto della cellula fotoelettrica. Questo secondo dispositivo era in grado di riprodurre l'immagine originaria ricreando l'intensità luminosa trasmessa.

Integrando il sistema con un semplice microfono radio, tecnologia già diffusa ai tempi, fu realizzato il primo prototipo di sistema **CCTV** (*Closed Circuit Television*), seppur con una qualità d'immagine decisamente lontana dagli standard odierni.

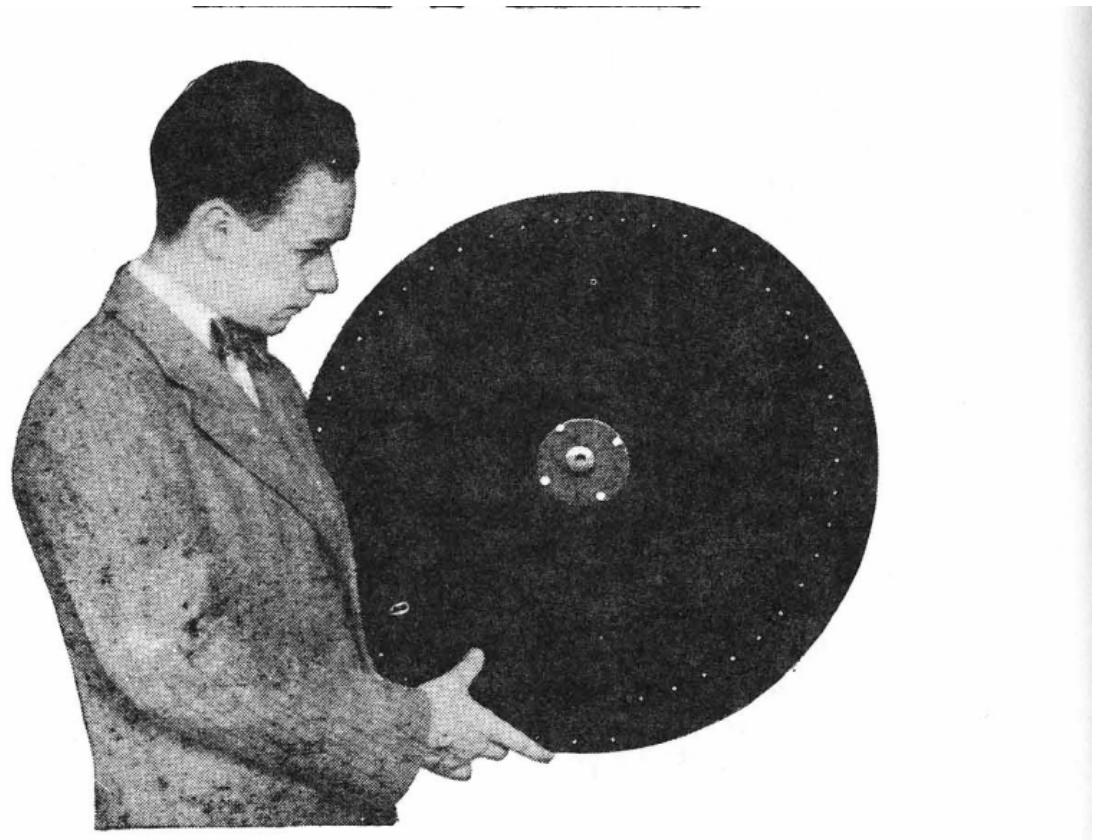


Figura 1.4: Theremin e il disco Nipkow [4]

1.2.1 Sicurezza collettiva vs libertà individuali

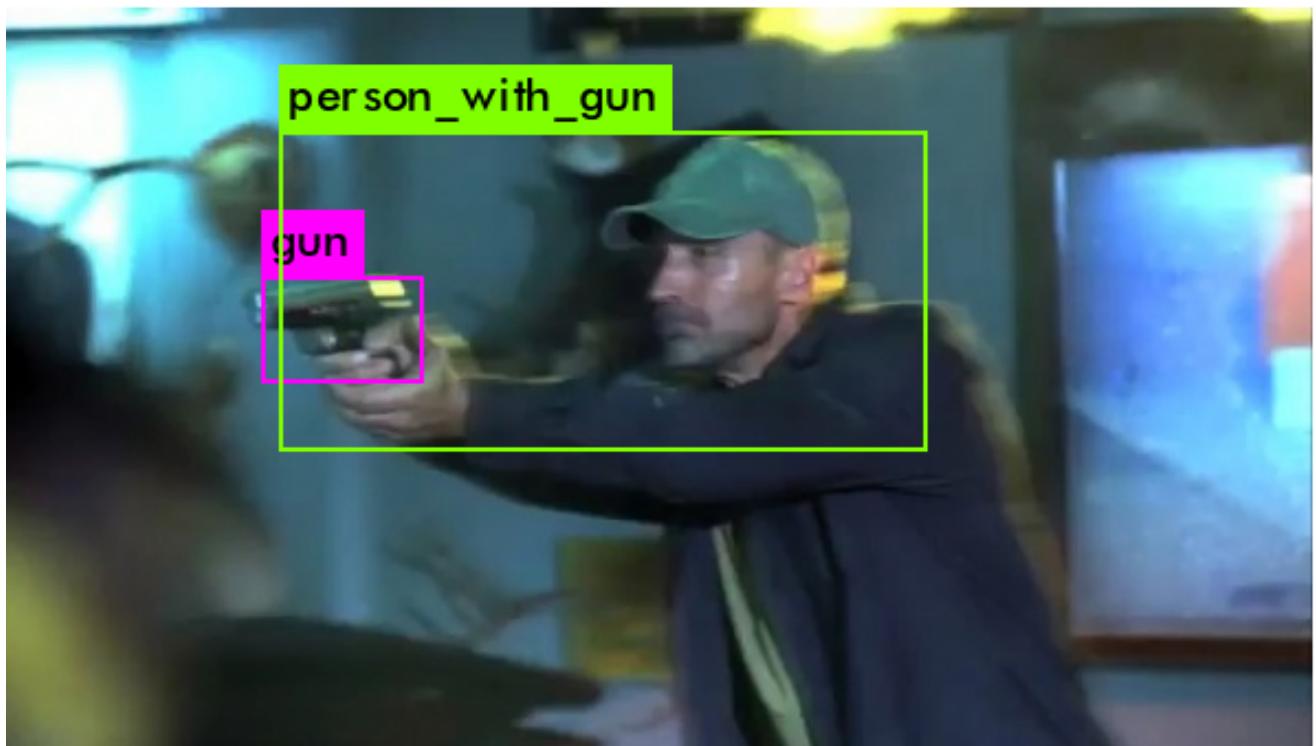
Facendo un salto nel futuro, è necessario riflettere sui rischi connessi al controllo diffuso della popolazione. Comprendere i limiti di questa zona grigia tra sicurezza collettiva e tutela dei diritti e delle **libertà individuali** è da sempre uno degli argomenti politici più discussi a livello globale. Il modo in cui viene applicata la legge può variare notevolmente tra i vari Paesi.

L'esempio più impressionante è certamente quello della **Cina**: chiunque abbia visitato il Paese si rende subito conto della capillare diffusione di telecamere, installate a ogni angolo della strada. Tuttavia, non è tanto la quantità dei dati raccolti a sollevare preoccupazioni, quanto piuttosto il fatto che questi sistemi utilizzano algoritmi di **riconoscimento facciale** per identificare i cittadini. Questa pratica, ormai normalizzata nella società cinese, è integrata con sistemi di controllo sociale come il discusso **Social Credit System** [9]. Alcune fonti scientifiche, tuttavia, suggeriscono che tale sistema non sia così esteso e repressivo come spesso viene descritto nei media occidentali [10].

1.2.2 Crime Detection

In conclusione, ritengo che una possibile soluzione per gestire questa zona grigia, già in fase di sperimentazione in alcuni Paesi occidentali, consista nel garantire una copertura quanto più estesa possibile della sorveglianza, anche grazie all'uso di droni e veicoli autonomi terrestri.

La differenza cruciale rispetto ai modelli invasivi risiede nel tipo di dati raccolti e nel loro trattamento. I dati dovrebbero essere **automaticamente eliminati in assenza di comportamenti illeciti**. Inoltre, è essenziale evitare ogni forma di **riconoscimento identitario**, puntando invece su algoritmi avanzati di *machine learning* in grado di rilevare in tempo reale **situazioni sospette**, ad esempio una rapina o un'aggressione, e **allertare tempestivamente le forze dell'ordine** o i servizi di emergenza. In questo modo, si garantirebbe una maggiore rapidità d'intervento e un'elevata probabilità di catturare il responsabile, preservando al contempo la privacy dei cittadini onesti.



Algoritmo computer vision di riconoscimento azione [10]

1.3 Veicoli senza equipaggio

Si tratta di veicoli di varia natura che possono essere controllati a distanza manualmente o pilotati da un sistema a controllo autonomo.

1.3.1 Nomenclatura ufficiale per gli **UV** (Unmanned Vehicle):

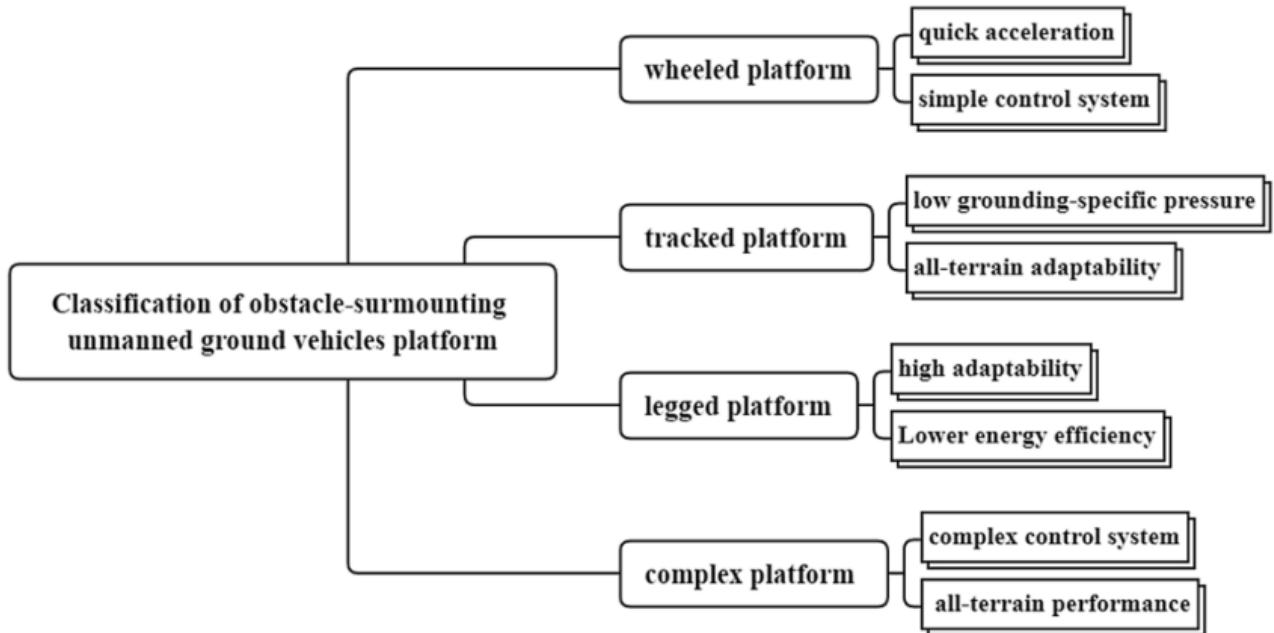
- **UAV** – *Unmanned Aerial Vehicle*: veicolo aereo senza pilota, comunemente noto come drone. Utilizzato in ambito militare, agricolo, per monitoraggio ambientale e consegna merci.
- **UGV** – *Unmanned Ground Vehicle*: veicolo terrestre autonomo o controllato da remoto. Può essere dotato di ruote, cingoli o zampe meccaniche, ed è impiegato in scenari militari, industriali o di soccorso.
- **USV** – *Unmanned Surface Vehicle*: veicolo di superficie non presidiato, che naviga su mari, fiumi o laghi. Utilizzato per sorveglianza marittima, esplorazione, operazioni scientifiche o militari.
- **UUV** – *Unmanned Underwater Vehicle*: veicolo subacqueo autonomo, impiegato per missioni sottomarine come il rilevamento di mine, la mappatura del fondale marino, la sorveglianza o la raccolta dati scientifici.

1.3.2 UGV – Veicoli terrestri senza equipaggio

Negli ultimi anni, l'interesse verso i veicoli UGV è cresciuto in modo significativo, coinvolgendo non solo l'ambito accademico, ma anche quello industriale. A differenza dei veicoli autonomi impiegati su **strade urbane e infrastrutture ben definite** come le automobili, l'attenzione si sta progressivamente spostando verso lo sviluppo di UGV capaci di operare in **ambienti non strutturati**, come foreste, montagne o scenari di emergenza legati a disastri naturali. In questi contesti, la capacità di muoversi su terreni difficili, aggirare ostacoli e adattarsi dinamicamente all'ambiente circostante rappresenta un requisito fondamentale.

La mobilità di un UGV è strettamente legata al tipo di **sistema di locomozione** adottato, che influenza direttamente la sua adattabilità e le sue prestazioni operative. Le principali configurazioni includono:

- UGV dotati di ruote;
- UGV cingolati;
- UGV con arti robotici o zampe meccaniche;
- Sistemi complessi e sperimentali (es. monoruota, guida su rotaie, ecc.).



Classificazione dei principali tipi di UGV [11]

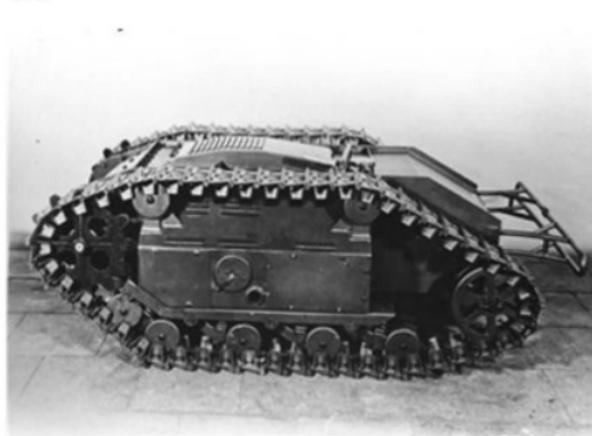
Le origini degli UGV risalgono agli anni '30 del Novecento, quando l'Unione Sovietica sviluppò il carro armato radiocomandato *T-26* e la Germania progettò il *Goliath*. Questi primi **prototipi radiocomandati** erano già in grado di disinnescare esplosivi, consegnare armi o fungere da esca sul campo di battaglia [11]. Tuttavia, le tecnologie dell'epoca non consentivano un controllo totalmente affidabile e questi veicoli non furono utilizzati su larga scala durante la guerra.

(a)



T-26

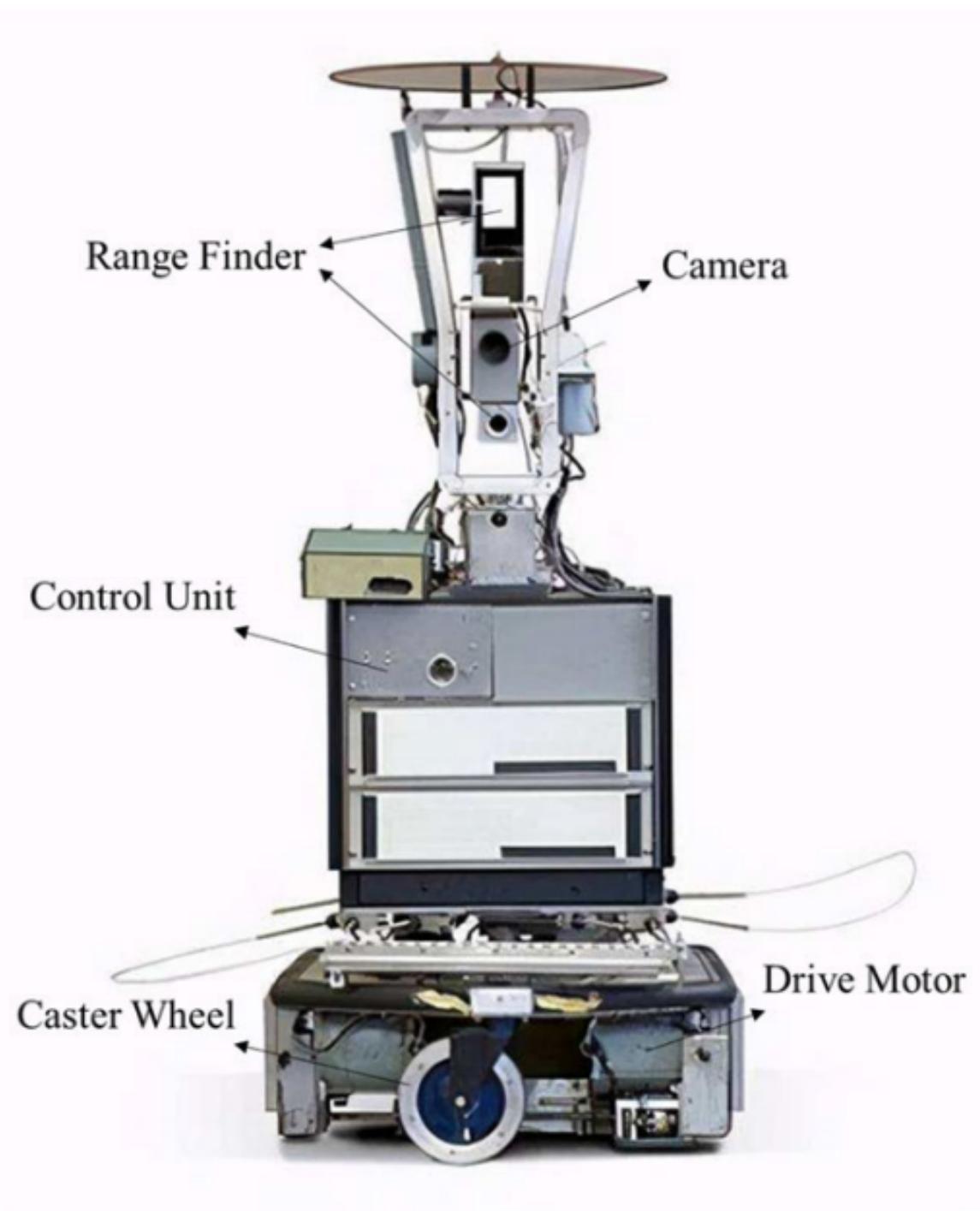
(b)



Goliath

Primi prototipi di UGV degli anni '30 [11]

Con l'evoluzione dell'elettronica e dell'informatica, alla fine degli anni '60 l'Università di Stanford diede avvio al progetto **SHAKEY**, finanziato dalla DARPA (Defence Advanced Research Projects Agency). SHAKEY è considerato il primo UGV della storia a impiegare sensori a ultrasuoni, videocamere e dispositivi tattili per percepire l'ambiente circostante. Il sistema era inoltre in grado di elaborare tali dati tramite un processore, pianificare percorsi e attuare strategie di elusione degli ostacoli in ambienti chiusi.



SHAKEY, il primo vero UGV della storia [11]

Il successo del progetto SHAKEY stimolò notevolmente l'interesse della comunità scientifica e delle istituzioni verso i veicoli terrestri autonomi, aprendo la strada a nuovi finanziamenti. Tra questi si distingue il progetto **ALV** (*Autonomous Land Vehicle*), sviluppato negli anni '90. ALV fu il primo veicolo terrestre autonomo, delle **dimensioni di un automobile**, in grado di muoversi all'aperto, anche su strade sterrate, raggiungendo una velocità massima di 10 km/h. Dotato di otto ruote e numerosi sensori, era in grado di spostarsi da un punto A a un punto B.

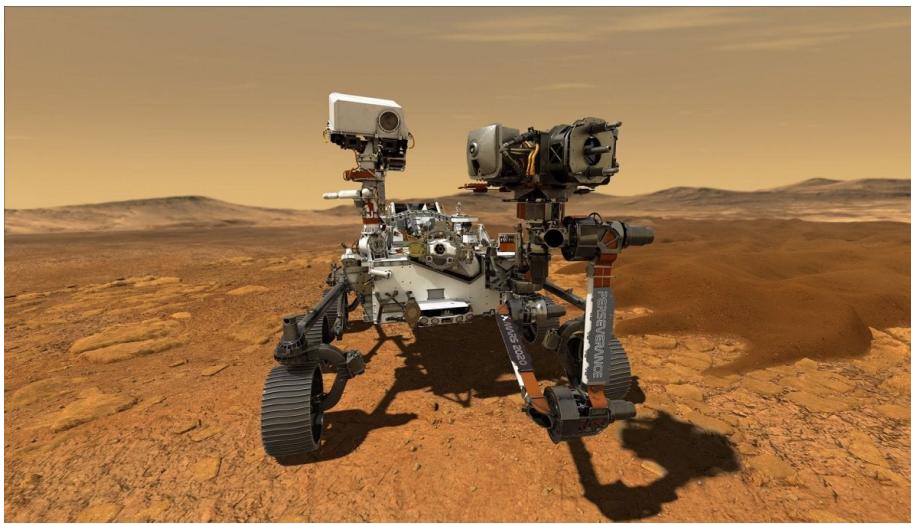


ALV, la prima vera automobile a guida autonoma [11]

Oggi, gli Stati Uniti continuano a rappresentare il principale polo di sviluppo per gli UGV, seguiti dalla Cina. Oltre all'impiego in ambito difensivo, questi veicoli vengono progettati anche per applicazioni specifiche come il **disinnesco di ordigni esplosivi** a distanza. Uno dei più famosi è il **TALON**, impiegato dalle forze armate americane per missioni ad alto rischio. Inoltre un altro utilizzo importante è l'**esplorazione spaziale**, come le missioni robotiche inviate su Marte.



UGV Talon, disinnesca esplosivi [11]



Rover NASA Perseverance [www.repubblica.it]

1.4 Aspetti normativi legati agli UGV

In questa sezione verranno analizzati gli aspetti normativi che entrerebbero in gioco qualora si volesse portare questo progetto nel **mondo reale**, ad esempio per scopi commerciali o di pubblica utilità. L'introduzione di un sistema autonomo nel contesto urbano, rurale o industriale comporta infatti una serie di considerazioni legali, etiche e procedurali che devono essere attentamente valutate.

1.4.1 Veicoli aerei vs veicoli terrestri

Partiamo dalla premessa che, a differenza dei droni aerei, non esiste un regolamento ufficiale specifico per gli UGV, in quanto sono ancora **poco diffusi** e potenzialmente meno pericolosi.

Basti pensare che i droni aerei vanno significativamente più veloci e muovendosi su 3 assi possono raggiungere facilmente zone sensibili in cui l'incidente è molto più grave. Inoltre, avendo bisogno di più capacità di scarica e quindi batterie più grosse e instabili come le LIPO, se costruiti male possono anche prendere fuoco più facilmente e fare danni seri.

Di conseguenza per i droni volanti esiste un regolamento preciso a livello nazionale presisposto da **ENAC**, in cui per i droni **a partire dai 250g** ci sono anche obblighi precisi di registrazione all'app D-flight e sono stati applicati dei regolamenti di altitudine su determinate zone, come le no flight zone.

Quindi, per l'impiego degli UGV (Unmanned Ground Vehicle) esistono due ambiti normativi fondamentali su cui è necessario focalizzarsi e che saranno illustrati nelle prossime sezioni.

1.4.2 Normativa sulla privacy e registrazione tramite telecamere

Secondo quanto stabilito dal Regolamento Generale sulla Protezione dei Dati (GDPR), in Italia è lecito riprendere persone su suolo pubblico soltanto per uso strettamente personale e privato. Tali riprese **non devono essere diffuse** o pubblicate, e devono essere eliminate immediatamente se non rilevanti, potendo essere giustificate come “riprese accidentali”.

Qualora invece le riprese documentino un reato o un comportamento illecito, è consentito conservarle e utilizzarle come **prova a supporto di eventuali indagini** o procedimenti giudiziari. Il mancato rispetto di tali disposizioni può comportare sanzioni amministrative anche di rilevante entità.

1.4.3 Normativa sull'utilizzo della comunicazione wireless

Per quanto riguarda la comunicazione senza fili, l'estensione del raggio d'azione di un dispositivo può teoricamente essere ottenuta aumentando la potenza del segnale. Tuttavia, la legge italiana impone limiti precisi in materia di trasmissione radio, soprattutto per le **bande di frequenza ad uso civile**.

Nello specifico, i limiti di potenza consentiti senza autorizzazioni sono generalmente compresi tra i 25 e i 100 mW per le bande a 2,4 GHz e 5,8 GHz (Wi-Fi), e 25 mW per la banda a 868 MHz utilizzata da tecnologie come LoRa. L'utilizzo della banda a 433 MHz (tipica dei radiocomandi) è anch'esso soggetto a restrizioni.

L'impiego di dispositivi che superano tali limiti o operano in bande non liberalizzate può causare interferenze nelle comunicazioni di altri utenti o apparati (es. sistemi aeronautici, militari o di emergenza), con conseguenti sanzioni e responsabilità legali. È inoltre possibile ottenere autorizzazioni superiori conseguendo apposite licenze da radioamatore.

Va infine sottolineato che, anche operando entro i limiti legali, il Wi-Fi alla massima potenza ammessa è in grado di raggiungere una distanza teorica di circa 300–400 metri in campo aperto.

2

Progettazione di UV

In questo capitolo verranno affrontate le **principali architetture** alla base della progettazione dei veicoli senza equipaggio e veicoli autonomi, valide trasversalmente per tutte le categorie analizzate nel capitolo precedente.

La struttura fondamentale di un sistema autonomo si basa su un'interazione continua e ciclica tra il mondo reale e quello virtuale. Come rappresentato nella *Figura 2.1*, il modello base di riferimento mostra un ciclo di elaborazione con cadenza dell'ordine dei nanosecondi.

Il processo inizia con l'acquisizione di dati dal mondo esterno attraverso un insieme eterogeneo di sensori. La percezione dell'ambiente varia in funzione delle caratteristiche di ciascun sensore e viene tradotta in una rappresentazione interna sotto forma di dati e valori formattati da protocolli specifici standardizzati o personalizzati.

Tali dati vengono elaborati autonomamente o, nel caso di controllo da remoto, supervisionati da un essere umano esterno, per la generazione di un **comportamento** ("behavior"), tenendo conto della missione assegnata, dei vincoli operativi e dei parametri configurati, il sistema lo traduce in azioni fisiche nel mondo reale tramite **motori e attuatori**. Tuttavia, per via di limiti meccanici o **imprecisioni tecniche**, l'azione eseguita può differire da quella pianificata. Per questo motivo, una progettazione efficace prevede l'utilizzo di sensori aggiuntivi per monitorare e verificare la corretta esecuzione delle azioni, trasmettendo eventuali feedback di errore e supporto.

Il ciclo si ripete in modo continuo, aggiornando in tempo reale lo stato della missione e le variabili operative. La qualità dell'implementazione, in assenza di malfunzionamenti hardware, dipende in larga misura dalla robustezza del codice e dalla **configurazione dei parametri**. Per questo motivo, nel contesto dell'Internet of Things (IoT) e dei sistemi autonomi in generale, risulta sempre più cruciale il contributo di professionisti esperti in informatica e automazione.

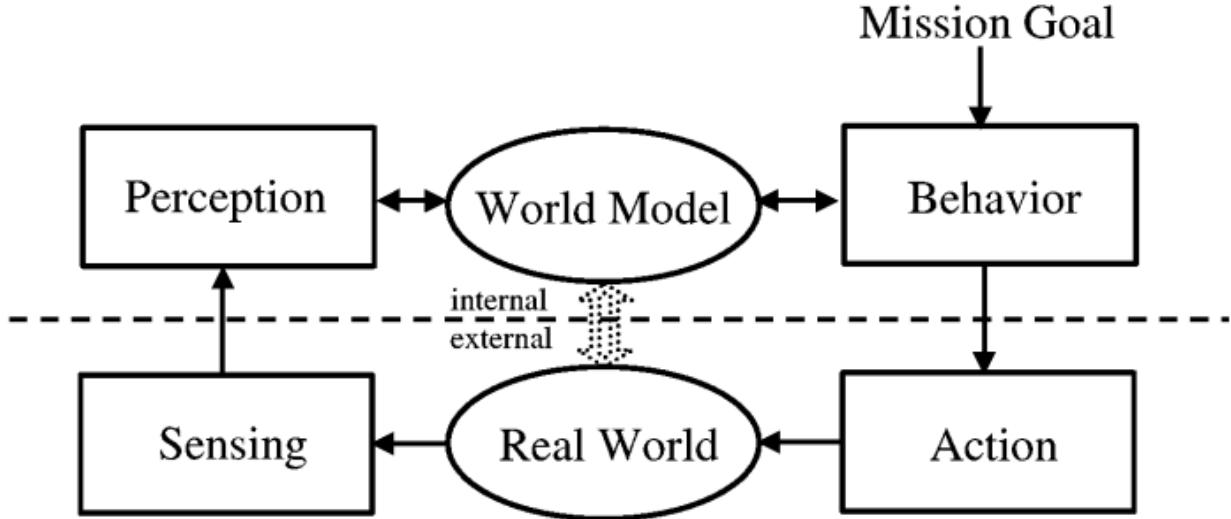


Figura 2.1: Modello base di funzionamento di un sistema autonomo [12]

Restando sempre a un livello astratto, approfondiamo meglio le quattro macro-fasi del ciclo operativo. Ciascuna di esse delimita, non sempre in modo netto, le seguenti sottofasi più specifiche:

- 1. Acquisizione (Sensing):** i sensori (telecamere, LIDAR, IMU, ecc.) raccolgono dati grezzi sull’ambiente circostante e sullo stato interno del veicolo.
- 2. Percezione (Perception):** questi dati vengono interpretati e elaborati per estrarre informazioni rilevanti, compatte e facilmente utilizzabili dal resto del sistema (es. NMEA per il GPS).
- 3. Cognizione/Pianificazione (Cognition/Planning):** sfruttando le informazioni percepite e gli obiettivi di missione, il sistema genera strategie operative e traiettorie ottimali, tenendo conto dei vincoli ambientali e dinamici. In questa fase si possono implementare sistemi di controllo qualità, verifica e validazione dei requisiti.
- 4. Controllo (Control):** la strategia definita viene trasformata in comandi precisi per motori e attuatori, tramite algoritmi specifici (*es. PID, MPC che non sono affrontati direttamente in questa tesi*).
- 5. Attuazione (Actuation):** i comandi vengono trasmessi ai motori e agli attuatori, generando le azioni fisiche. Sensori di feedback permettono di monitorare l’efficacia e correggere eventuali errori.

2.1 Progettazione dell' UGV

In questa sezione verranno analizzate nel dettaglio tutte le **scelte progettuali** effettuate per la realizzazione dell' UGV oggetto di tesi. Per ciascun componente verranno presentate le principali **alternative** disponibili e le motivazioni che hanno guidato le decisioni adottate, alla luce delle specifiche esigenze tecniche, funzionali e operative.

Per facilitare la comprensione e rendere più concreta l'implementazione rispetto ai modelli astratti presentati nei capitoli precedenti, si è fatto riferimento a una rappresentazione semplificata del sistema, più aderente all'architettura effettivamente implementata.

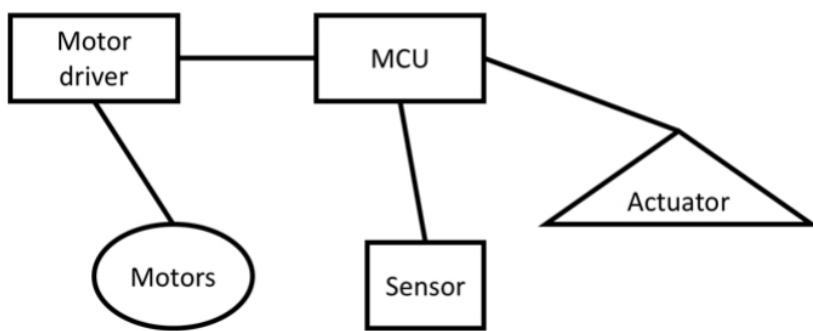


Figura 2.2: Modello architetturale semplificato del sistema sviluppato [42]

Nel dettaglio, i principali blocchi funzionali del modello sono così descritti:

- **MCU (Microcontroller Unit)**: rappresenta il cervello del sistema, ovvero l'unità di elaborazione centrale responsabile della gestione del flusso dati, della logica di controllo e del coordinamento tra i vari sottosistemi.
- **Motor driver**: si tratta di un componente elettronico, a volte integrato nella MCU, che permette di interfacciare i segnali di controllo con i motori. Include circuiti di regolazione della tensione, protezioni da sovraccarico e dispositivi per la dissipazione termica.
- **Sensori**: sono dispositivi deputati all'acquisizione di informazioni sull'ambiente circostante. Esistono molte tipologie di sensori, ciascuna specializzata nel rilevamento di particolari fenomeni fisici (es. distanza, temperatura, luminosità), che convertono in tempo reale in segnali digitali o analogici processabili dal sistema.
- **Attuatori**: includono tutti quei dispositivi in grado di generare azioni fisiche sull'ambiente esterno (*output* del *MCU*), come bracci meccanici, luci, altoparlanti e dispositivi audiovisivi.
- **Motori**: in questa rappresentazione vengono trattati come una particolare classe di attuatori, dedicata esclusivamente alla locomozione del veicolo.

2.2 Componenti Principali

2.2.1 Struttura esterna

Per questo progetto si è deciso di ricondizionare una struttura esterna già esistente, comprensiva del sistema di locomozione, utilizzando una **vecchia macchina radiocomandata** degli anni '80, modello La Rossa (figura 2.3), appartenente alla categoria "Formula Super Buggy" in scala 1:14. Il veicolo nasce dalla collaborazione tra la casa di giocattoli giapponese Nikko e l'azienda italiana GIG [13].

Il sistema di locomozione adottato è basato su **ruote**, con un motore posteriore dedicato alla trazione e uno anteriore per lo sterzo. All'epoca, questi motori erano controllati da un semplice chip (figura 2.4) a radiofrequenze, che lavorava con **segnali analogici grezzi**, operando tipicamente su bande da 27 a 40 MHz, modulati in AM o FM. Attraverso l'utilizzo di **porte logiche**, il sistema era in grado di codificare in maniera essenziale quattro comandi principali: avanti, indietro, sinistra e destra. Nonostante la semplicità del sistema, i tempi di reazione erano molto rapidi, permettendo al veicolo di raggiungere una velocità massima di circa 20 km/h [14].

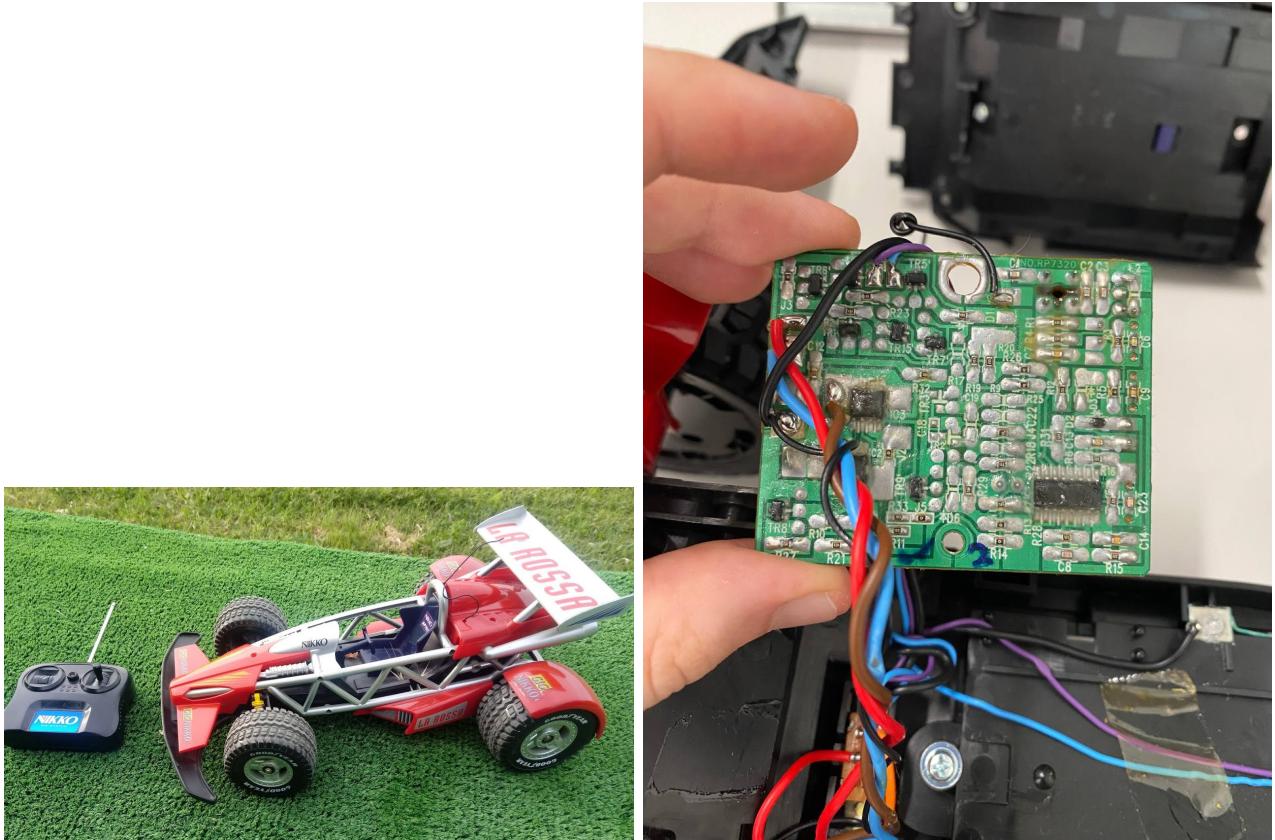


Figura 2.3: A sinistra: Gig-Nikko "La Rossa" A destra: chip di controllo originale.

Per adattare il veicolo al nuovo sistema di controllo, si è provveduto alla rimozione della scocca originale in plastica e del chip elettronico, mantenendo i collegamenti diretti ai motori. È stata quindi progettata e realizzata, tramite **stampa 3D**, una struttura di supporto semplificata, in grado di ospitare in modo ordinato tutte le nuove componenti elettroniche e i relativi cablaggi.

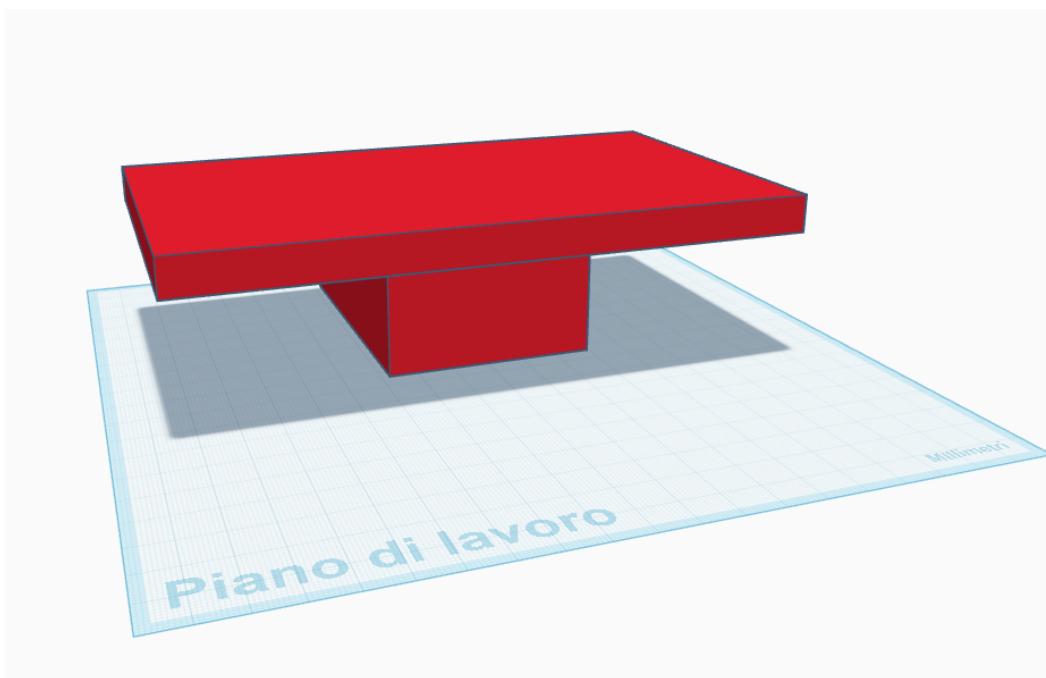


Figura 2.4: Modello 3D della struttura di supporto per componenti e cablaggi.

2.2.2 Motori

Dalla struttura originale sono stati ereditati i due motori principali, rispettivamente la trazione posteriore e lo sterzo anteriore. Dopo un'attenta analisi e ricerca, è stato possibile identificare i modelli e valutarne le caratteristiche tecniche.

Motore di trazione – Mabuchi RC-280

Il motore utilizzato per la trazione è un **Mabuchi RC-280**, prodotto da una delle più note aziende giapponesi specializzate in attuatori elettrici per modellismo, giocattoli ed elettrodomestici. Il modello RC-280 era molto diffuso negli anni '80 e '90, grazie all'ottimo rapporto tra **dimensioni compatte e prestazioni elevate**. È stato infatti adottato anche in molti altri modelli, come la Freccia Rossa (scala 1:16) e la Tamiya QD Thunder Shot (scala 1:14). In alternativa, per veicoli più pesanti, veniva impiegato il modello RS-360SH, leggermente più grande e meno veloce, ma dotato di coppia maggiore.

E' un motore a **corrente continua** (DC), l'inversione della polarità determina la **direzione di rotazione**. E' un motore brushed, dotato di spazzole in grafite che trasferiscono corrente al rotore. Le spazzole, **soggette a usura**, possono generare elettricità statica; per questo è spesso presente un terzo **cavo di massa**, fissato alla scocca, per la scaricare l'energia in eccesso.

Caratteristiche tecniche del motore RC-280, come da sito ufficiale Mabuchi Motors [15] :

- **Tensione nominale:** da 3,5V a 9V (nel progetto modulerò la potenza in PWM al 50 %).
- **Velocità a vuoto:** 14.000 giri/min, con assorbimento di 0,28A.
- **Corrente di stallo:** fino a 1,60A.
- **Potenza meccanica massima:** 5,56W.
- **Coppia di stallo:** 29,9mN·m.

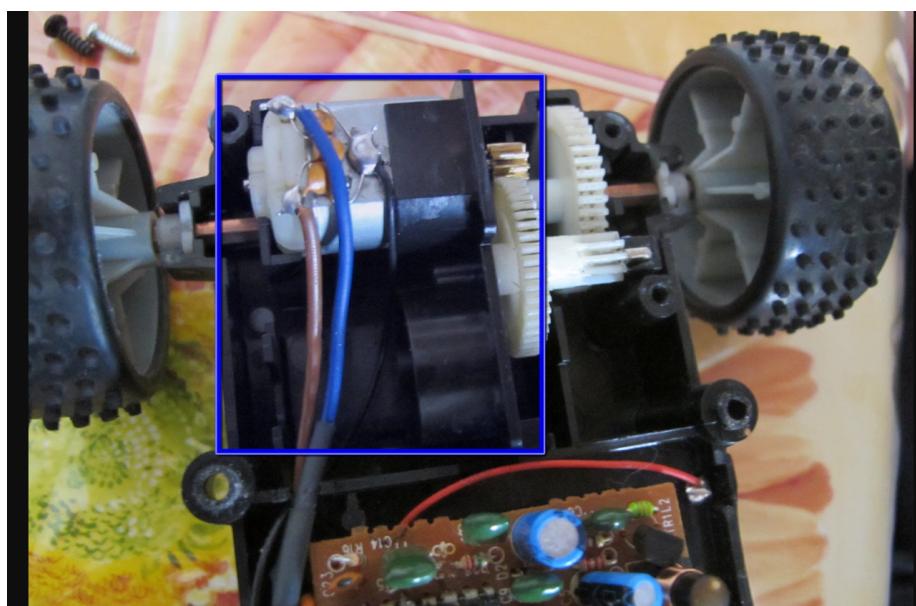


Figura 2.5: Motore Mabuchi RC-280 per la trazione posteriore.

Motore dello sterzo – FN130

Il motore impiegato per il controllo dello sterzo è un **FN130**, un piccolo motore DC solitamente abbinato a un sistema di **molle e ingranaggi**, tale da permettere una funzione simile a quella di un servosterzo, pur senza richiedere un controllo angolare elettronico sempre attivo.

Il meccanismo (figura 2.5) si basa su una molla di frizione di dimensione precisa, in modo da essere compressa dal motore fino a un certo limite. Il movimento provoca la **rotazione di un perno in plastica** che agisce direttamente sulle ruote anteriori, limitando l'angolo massimo di sterzata per prevenire danni meccanici. Quando la **corrente viene rimossa** dal motore, la molla riporta automaticamente il sistema in posizione centrale.

In fase di sviluppo software, è emersa una criticità: **interrompere bruscamente l'alimentazione** al motore non è tecnicamente corretto e può generare comportamenti indesiderati. Una possibile soluzione adottabile potrebbe essere quella di inviare un impulso inverso, cioè con polarità opposta, al termine della sterzata, per facilitare il ritorno al centro senza interruzioni nette di corrente.

Le principali caratteristiche tecniche del FN130 sono [16]:

- **Tensione nominale:** da 3V a 6V (nel progetto modulerò la potenza in PWM al 50%).
- **Corrente di stallo:** fino a 2,10A,
(evitare di mantenere lo sterzo troppo a lungo, data la natura bloccante del meccanismo a molla)
- **Potenza meccanica media:** 2,5W.

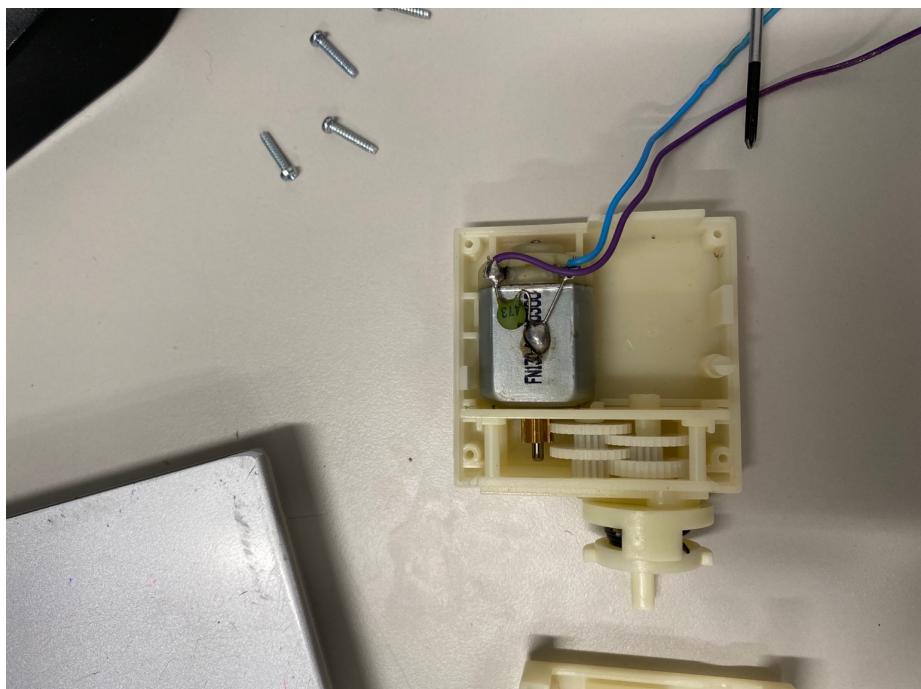


Figura 2.6: Motore FN130 utilizzato per lo sterzo.

2.2.3 Driver Motori

Il primo nuovo componente da analizzare è il driver, un chip fondamentale per la gestione dell'**alimentazione elettrica dei motori**. L'obiettivo è fornire una tensione adeguata, superiore a quella insufficiente erogata dalla MCU. Spesso, questa proviene da una **batteria esterna** di cui però bisogna **regolarne il flusso** di corrente in modo sicuro e controllato.

In particolare, durante l'avvio o lo spegnimento di un motore possono verificarsi improvvisi picchi di corrente (**current spikes**) che se non vengono accuratamente gestiti, potrebbero danneggiare la MCU o altri componenti sensibili del circuito. Per prevenire questi problemi, molti driver vengono dotati di circuiti di protezione, tra cui i **diodi di flyback** [17], che assorbono e dissipano l'energia in eccesso generata dal collasso del campo magnetico nel momento in cui il motore viene spento.

Un'altra caratteristica fondamentale del driver è la presenza di una rete di transistor controllabili e programmabili direttamente dalla MCU. Questi permettono di aprire o chiudere i circuiti in modo selettivo, e quindi manipolare il flusso di corrente della batteria esterna, tramite **impulsi controllati a basso voltaggio**. Attraverso un circuito in particolare chiamato **ponte H** (figura 2.7) [18], è possibile invertire la polarità della corrente, consentendo così la rotazione del motore in entrambe le direzioni, senza la necessità di intervenire fisicamente sul cablaggio, ma via segnale digitale.

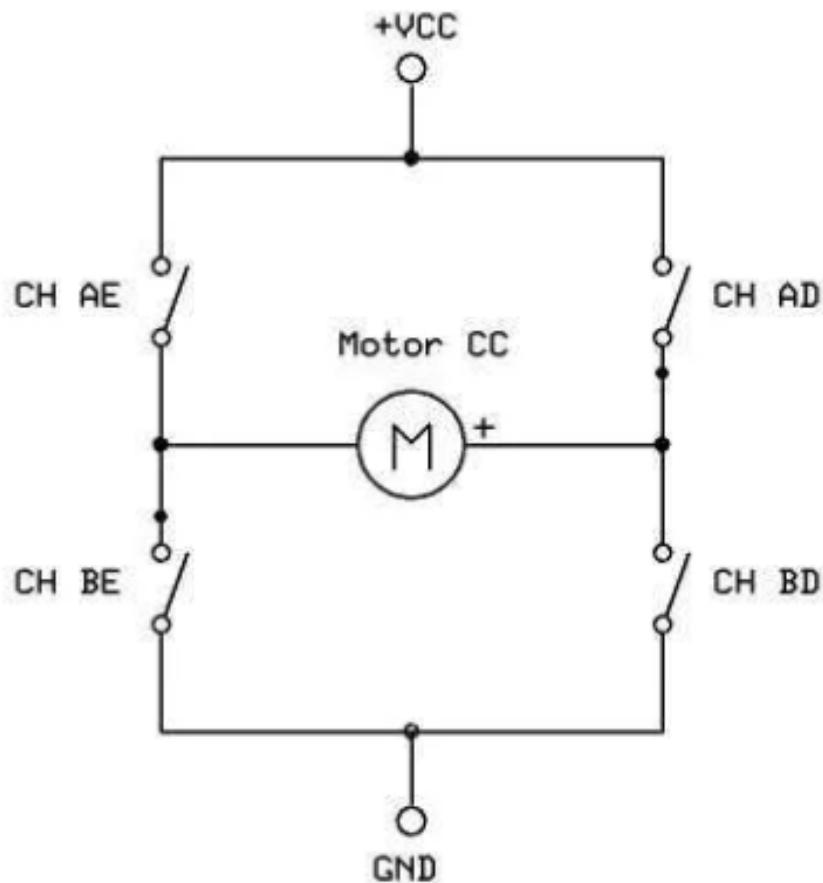


Figura 2.7: Schema elettronico di un ponte H

Molti componenti elettronici, come motori DC e LED, operano solo con **segnali analogici**. Tuttavia, molte MCU moderne non dispongono nativamente di uscite analogiche. Per questo motivo si utilizza la tecnica della **PWM (Pulse Width Modulation)** [19]: un metodo che permette di simulare un segnale analogico tramite impulsi digitali ad alta frequenza.

In breve, la **differenza tra un segnale analogico e uno digitale** [20] è che il segnale analogico varia in modo continuo e fluido nel tempo, assumendo un valore stabile o al massimo che cambia gradualmente, per esempio 5v. Al contrario, un segnale digitale varia bruscamente in scatti discreti, rappresentando tipicamente due stati: 0 e 1, tramite una tensione bassa e una alta, per esempio 0V e 5V.

Il PWM è un segnale digitale a frequenza costante, di cui è possibile modificare il **duty cycle**, ovvero la percentuale di tempo in cui la tensione rimane alta durante ciascun ciclo. Praticamente, ogni volta che **il segnale sarà acceso, verrà aperto il transistor** e quindi chiuso il circuito tra la batteria esterna e i motori. Decidendo per quanto tempo far scorrere la corrente, è possibile addirittura **regolare la potenza media erogata al carico**, quindi nel nostro caso controllare la **velocità del motore**. In questo modo gli attuatori percepiscono una tensione media proporzionale al duty cycle, proprio come se ci fosse un flusso analogico costante.

- **Frequenza:** indica quante volte il ciclo on/off viene ripetuto in un secondo (Hz). Ad esempio, 1000 Hz significa che ci sono 1000 cicli al secondo.
- **Duty cycle:** indica per quanto tempo il segnale resta attivo all'interno di ciascun ciclo. Ad esempio, un duty cycle del 50% significa che il segnale resta acceso per metà del tempo e spento per l'altra metà.

$$DutyCycle = \frac{T_{on}}{T_{on} + T_{off}} \times 100 \quad (2.1)$$

dove T_{on} è la durata del segnale acceso e T_{off} la durata del segnale spento.

Quindi $T_{on} + T_{off}$ equivale alla durata di un ciclo.

Nota: un segnale a 5V modulato al 50% tramite PWM ha una **tensione istantanea** sempre pari a 5V, ma una **potenza media equivalente** a circa 2.5V. Questo comporta, ad esempio, una velocità di rotazione ridotta nei motori o una luminosità inferiore nei LED.

A livello meccanico quindi, come già detto, il **duty cycle controlla** quanta energia viene erogata e quindi la velocità del motore, mentre **la frequenza influisce** su aspetti come: fluidità del movimento, rumore e vibrazioni, surriscaldamento, e tempi di risposta del motore, in quanto a frequenze troppo alte, le componenti del circuito potrebbero non rispondere bene ed essere più lente.

Nel progetto, si è scelto di usare una frequenza di **1000 Hz** per entrambi i motori, che ha mostrato buoni risultati nella riduzione di rumore e vibrazioni. Generalmente i LED necessitano di frequenze molto più alte per evitare sfarfallii visibili.

Infine, i motori utilizzati nel progetto presentano un range operativo di tensione inferiore rispetto all'alimentazione fornita. Per prevenire danni e surriscaldamenti, è stata impostata da software una potenza PWM fissa al **50%** per entrambi i motori. Dai test effettuati, questo valore ha garantito una **buona reattività e sicurezza**, in particolare per il motore dello sterzo, che avendo un **meccanismo bloccante** è più incline a problemi meccanici.

Driver L298N

Una delle prime proposte considerate per il driver motori è stata l'utilizzo del chip **L298P** prodotto da KeyStudio (figura 2.8), integrato in uno *shield* per Arduino. Questo modulo è pensato per essere direttamente montato sopra la scheda Arduino, utilizzando soltanto i pin necessari per il controllo, lasciando liberi gli altri. Si trattava di una soluzione interessante per motivi di **compattezza** (non richiedeva cavi aggiuntivi per collegare la MCU) e di praticità, poiché permetteva il controllo simultaneo di **due motori contemporaneamente**. Questa caratteristica è stata particolarmente apprezzata e ricercata anche nelle alternative successive, in quanto semplificava notevolmente la programmazione e l'integrazione nel sistema.

Tuttavia, l'utilizzo dell'L298P **vincolava la scelta della MCU all'arduino**, e il riadattamento ad altri microcontrollori risultava complesso e poco affidabile. È stata anche **tentata una riconfigurazione** dei pin tra un Raspberry Pi e lo shield, ma senza risultati soddisfacenti.

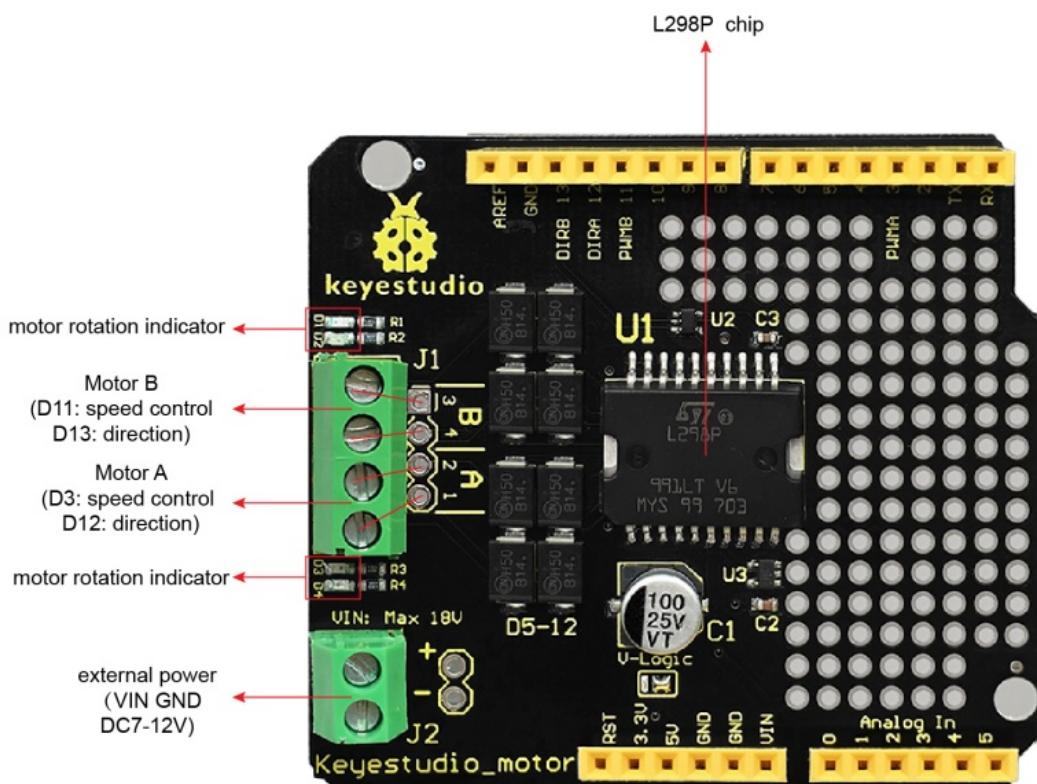


Figura 2.8 – L298P KeyStudio

La scelta finale è ricaduta sul **L298N** [21], un **driver standalone**, adattabile a qualsiasi MCU, basato su architettura *dual full-bridge* (doppio ponte H), capace di controllare due motori in corrente continua (DC) oppure, in alternativa, un motore stepper (non trattato in questo progetto).

Il collegamento del circuito è semplice:

- I **pin laterali** dedicati ai due poli di ciascun motore: motore A (a sinistra) e motore B (a destra).
- I pin di **alimentazione** comprendono:
 - +12V e GND per collegare la batteria esterna
(range supportato da 5V a 46V anche se le istruzioni consigliano di non superare i 12V).
 - 5V **in output** opzionale, che può essere usato per alimentare la MCU.
Tuttavia, nel presente progetto non è stato utilizzato, per evitare instabilità dovuta all'inefficienza del circuito interno, che introduce una **caduta di tensione di circa 1.5–2V**.
- I **pin centrali** (tre per ogni motore) comprendono:
 - Il pin ENA/ENB, collegato a un'uscita PWM della MCU, per la modulazione della potenza.
 - I due pin IN1/IN2 (per motore A) e IN3/IN4 (per motore B), collegati a pin digitali per il controllo della direzione tramite la gestione del ponte H.

Come visto in precedenza, il ponte H consente l'inversione della polarità del motore senza necessità di intervenire fisicamente sul circuito. Quando entrambi i pin direzionali sono **entrambi spenti** (LOW) o accesi (HIGH), non c'è passaggio di corrente utile e il **motore resta fermo**. L'alternanza tra HIGH e LOW sui pin di direzione è ciò che consente la rotazione in un senso o nell'altro.

Nota: se entrambi i pin sono accesi, tutti e due i transistor nel ponte H sono chiusi, questo significa che la tensione positiva arriva a entrambi i capi del motore, **non generando una differenza di potenziale**, la corrente non passa e il motore non gira.

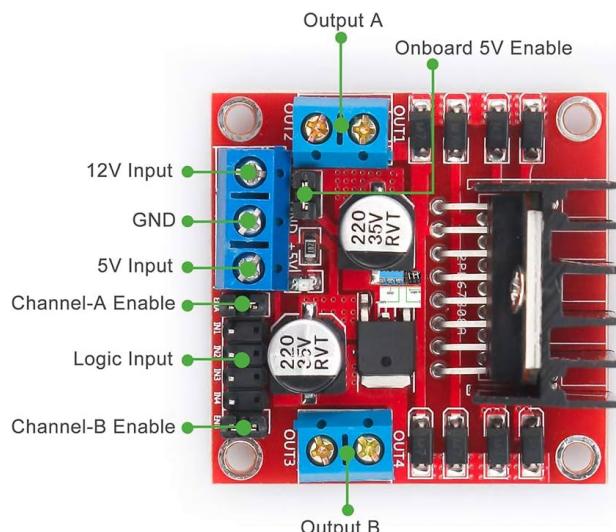


Figura 2.9 – Modulo driver L298N con dissipatore

Il chip L298N è economico, facilmente reperibile e generalmente affidabile. Tuttavia, ha un limite di corrente di circa 2A per canale: se i motori rimangono bloccati troppo a lungo o richiedono una corrente elevata, il chip potrebbe surriscaldarsi ed entrare in **protezione termica**. Per questa ragione, è stato installato un dissipatore di calore sul modulo per aumentare la stabilità operativa.

2.2.4 Alimentazione

L'alimentazione rappresenta uno degli aspetti più critici nella progettazione di UV, in quanto le batterie moderne che riescono a eccellere in tutte le caratteristiche desiderate risultano spesso scarse o molto costose. Una soluzione tecnologicamente valida deve saper bilanciare correttamente le principali prestazioni richieste: **autonomia operativa, capacità di scarica, stabilità e sicurezza**, il tutto mantenendo **dimensioni e peso** contenuti. Le batterie rilasciate per questo tipo di applicazioni forniscono corrente continua (DC).

Tali caratteristiche dipendono fortemente dalla tipologia di veicolo sviluppato. Ad esempio, nei droni aerei si tende spesso a sacrificare la durata della batteria e la sicurezza in favore della leggerezza e di una maggiore capacità di scarica, necessarie per garantire manovre rapide e reattive in volo. Al contrario, un drone terrestre, che consuma energia in modo più costante e prevedibile, richiede meno potenza istantanea, ma una maggiore autonomia e stabilità nel tempo. Basti pensare che quando un UGV deve restare fermo non consuma batteria per i motori al contrario del drone.

Le batterie sono generalmente costituite da **celle elettrochimiche**, ognuna delle quali fornisce una tensione compresa tra 3,V e 4,2,V. Per ottenere tensioni maggiori, più celle vengono collegate in serie.

Le principali caratteristiche da considerare sono:

- **Capacità**: espressa in mAh (milliampere/ora), indica la quantità di carica che la batteria può erogare nel tempo.
- **Fattore di scarica (C-rate)**: rappresenta il tasso massimo di corrente che una batteria può erogare in **condizioni di sicurezza**. C è un valore moltiplicativo, che indica la corrente massima continua erogabile **in funzione della capacità** nominale della batteria.

Ad esempio, una batteria da $2000\text{ mAh} = 2\text{ Ah}$ con un C-rate di 10C può fornire fino a:

$$I_{max} = C \times \text{capacità} = 10 \times 2\text{ Ah} = 20\text{ A}$$

Quindi, se il circuito, per esempio un **grossso motore DC stallato richiede 20A** di corrente, la batteria **sarà in grado di erogarli** senza subire danni o problemi tecnici, perchè tale assorbimento rientra nei limiti della sua capacità di scarica.

Resta quindi da valutare: per quanto tempo la batteria sarà in grado di sostenere questa erogazione prima di scaricarsi completamente?

Il tempo di scarica varia in base alla corrente assorbita. È calcolabile con la formula:

$$TempoDiScarica = \frac{CapacitàBatteria}{CorrenteAssorbita}$$

dove:

- tempo di scarica in ore [h]
- capacità della batteria [Ah]
- corrente assorbita [A]

Nel caso precedente, assorbendo 20 A, la batteria si scaricherebbe completamente in:

$$t = \frac{2 \text{ Ah}}{20 \text{ A}} = 0,1 \text{ h} = 6 \text{ minuti}$$

Quindi, maggiore è il valore di corrente assorbita, minore sarà la durata della batteria. Viceversa, per farla durare più a lungo, si deve **ridurre il carico o aumentare la capacità** per esempio aumentando il numero di celle. La scarica dipende spesso anche dalla carica della batteria, **più la batteria è scarica meno corrente riesce a fornire**.

- **Sicurezza:** dipende dalla chimica interna. Le **batterie LiPo** (polimeri di litio) [22] sono molto performanti, ma anche instabili e pericolose se mal gestite (sottocarica, sovraccarica, urti). Richiedono attenzione nello stoccaggio e nella ricarica: non devono essere caricate oltre i 4,2 V per cella né scaricate sotto i 3V. Possiedono grossa capacità di scarica, che viene usata dai droni volanti per brevi periodi, consumando molta battaria, è difficile volare per più di 30 minuti.

Considerando che il progetto sviluppa un drone terrestre, è stato possibile optare per una soluzione più sicura, scegliendo batterie agli **ioni di litio** (Li-ion) [23], nello specifico le comuni **celle 18650** (figura 2.10) (18,mm di diametro, 65,mm di lunghezza). Queste batterie sono stabili, sicure anche in caso di urti e facilmente reperibili (es. power bank, laptop, sigarette elettroniche).



Figura 2.10 - batterie 18650 ioni di litio

Nel progetto sono state utilizzate due celle singole 18650 collegate in serie tramite un apposito supporto, incollato nella struttura del vecchio vano batteria della macchina. Ogni cella ha una tensione media di 3,7,V e una tensione massima di 4,2,V. In serie, la tensione massima è di 8,4,V. Il caricabatterie impiegato interrompe automaticamente la ricarica una volta raggiunta questa soglia per evitare danni.

La **capacità di scarica** delle celle Li-ion è modesta (1C–10C), quindi in grado di fornire pochi ampere in sicurezza. Tuttavia, dai test effettuati, è emerso che i valori **risultano adatti** per alimentare MCU, sensori e motori, a condizione di evitare picchi improvvisi di corrente, dovuti allo stallo dei motori.

Quando la richiesta di corrente supera la capacità della batteria (**soprattutto se scarica**), si verifica un particolare fenomeno chiamato **caduta di tensione**, più volte osservato durante il progetto a causa di un caricabatterie difettoso. Cioè il valore della tensione crolla in pochi secondi, non riuscendo più ad alimentare nessun componente, se la tensione scende sotto i 3,3V per cella, si rischia di **danneggiare permanentemente** la batteria. Per questo motivo, si è deciso di interrompere l'uso e di rimettere le celle sotto carica, quando la tensione complessiva scende sotto i 6,5V (valore 0% stato batteria).

Durante test prolungati, con celle da 2600mAh ciascuna, il sistema ha mostrato un'autonomia compresa tra 40 e 60 minuti e tensione compresa tra 8,4,V e 6,5V.

Sono state considerate anche alternative, come:

- Utilizzare **due pacchi batteria separati**: uno per la MCU e i sensori, l'altro per i motori.
- Aggiungere una terza cella in serie (per un totale di 12,6,V) per aumentare stabilità e autonomia.

Il circuito è stato progettato in modo modulare, consentendo futuri upgrade nel sistema di alimentazione, in modo da essere più sicuri sulla distribuzione della corrente e per aumentare la durata di utilizzo.

Un'altra criticità osservata è stata la **disparità di carica tra le celle**. In presenza di due batterie in serie, quella più scarica tende a degradarsi più velocemente, causando cadute di tensione e instabilità. Un componente utile, non implementato nei test iniziali ma consigliato per le fasi successive, è il **Battery Management System (BMS)**. Questo modulo hardware monitora e protegge il pacco batteria da sottotensione, sovratensione, corto circuito, surriscaldamento e gestisce appunto il bilanciamento tra le varie celle evitando cadute di tensione.

Distribuzione dell'alimentazione nel circuito

Analizzando come verranno alimentate le diverse componenti del sistema, si osserva che:

I sensori sono alimentati direttamente dalla MCU, che fornisce in uscita tensioni regolate a 5V o 3,3V.

Il modulo L298N, come visto in precedenza, supporta un'alimentazione fino a 35V. In questo caso, non è tanto il driver a doverci preoccupare quanto i motori, che hanno un range operativo compreso tra 3V e 6V. Considerando che il modulo introduce una caduta di tensione interna di circa 2V, si è deciso

di collegare direttamente la batteria (con tensione massima pari a 8,4,V) all'ingresso del driver, regolando successivamente la potenza tramite modulazione PWM al 50%. Dai test eseguiti, questo approccio si è dimostrato sicuro e non sono stati rilevati surriscaldamenti né danni ai motori.

In parallelo, la stessa alimentazione viene fornita a un **convertitore buck** (figura 2.11) [24], necessario per alimentare la MCU con una **tensione stabile** e costante di 5V. Questo regolatore di tensione, regolabile tramite vite, è dotato di un piccolo display integrato per il monitoraggio in tempo reale e include protezione da sovrattensione e sovraccorrente, grazie alla presenza di condensatori, induttori e dei già citati diodi di flyback. Il convertitore è progettato per erogare una tensione di uscita sempre inferiore o uguale a quella in ingresso; pertanto, anche durante una fase di scarica, la tensione in uscita resterà stabile finché quella in ingresso non scenderà sotto i 5V.

Infine, è importante ricordare che in un circuito in parallelo la **tensione è uguale per tutti i rami, mentre la corrente si divide**. Ciò significa che, in condizioni di stress, i motori possono assorbire una quantità elevata di corrente, **sottraendola alle altre componenti**. La MCU, essendo molto più sensibile, potrebbe risentirne, anche perché il convertitore buck regola solo la tensione e non stabilizza la corrente. Un modo per sistemare questo problema è stato già citato in precedenza e corrisponde ad utilizzare un' alimentazione separata tra motori e MCU.

Nei test svolti, è emerso che in condizioni particolari, come batterie molto scariche e motori sotto carico elevato, si possono verificare cali di corrente tali da **compromettere il funzionamento di parti della MCU**. Un caso osservato è stato il disaccoppiamento del modulo Wi-Fi, uno degli elementi a più alto consumo della scheda, causato appunto da instabilità dell'alimentazione.



Figura 2.11 - convertitore buck

2.3 Sensoristica

Sono stati scelti una serie di sensori essenziali per rilevare l'ambiente esterno e consentire il controllo dell'UGV anche a distanza, senza la necessità di una visuale diretta sul mezzo.

Si includeranno inoltre alcuni sensori opzionali, che potrebbero essere integrati in futuro per migliorare ulteriormente il sistema.

2.3.1 Rilevatore di tensione e convertitore analogico-digitale

L'obiettivo è monitorare in tempo reale la tensione della batteria, analogamente a quanto avviene con lo schermo del convertitore buck, ma acquisendo il dato tramite la MCU per elaborarlo e utilizzarlo sia a scopo informativo che per il controllo del veicolo.

Oltre a fornire un'**indicazione da remoto sul livello di carica della batteria**, utile per determinare quando è necessario ricorrere alla base di ricarica, il monitoraggio permette anche di analizzare quali **azioni provocano cali di tensione** e come questi variano nel tempo.

Il componente utilizzato per questa funzione è composto da due parti:

Il primo modulo utilizzato è un **sensore di tensione resistivo** [25], basato su un partitore di tensione. È costituito da due resistenze ad alta precisione da $30\text{ k}\Omega$ e $7,5\text{ k}\Omega$, collegate in serie. Il principio di funzionamento si basa sulla legge di Ohm, secondo la quale la caduta di tensione su una resistenza è data da $V = R \cdot I$, con I corrente costante nel circuito.

Quando due resistenze sono collegate in serie, la tensione totale si ripartisce proporzionalmente tra di esse: la resistenza più alta comporta una caduta di tensione maggiore. Il nodo centrale tra le due resistenze fornisce una tensione ridotta, proporzionale a quella di ingresso. Questo valore, chiamato V_{out} , è calcolabile con la formula:

$$V_{out} = V_{in} \cdot \frac{R_2}{R_1 + R_2}$$

Nel nostro caso:

$$\frac{7,5}{30 + 7,5} = \frac{1}{5} \Rightarrow V_{out} = \frac{V_{in}}{5}$$

In pratica, questo modulo consente di **ridurre una tensione di ingresso** (fino a 25V) a un valore compatibile con i convertitori analogico-digitali, che tipicamente supportano un massimo di 5V in ingresso. In breve sta trasformando un segnale in ingresso, massimo 25v, **in un segnale tra 0v e 5v** in modo preciso e **proporzionato** riducendolo esattamente di 5 volte.

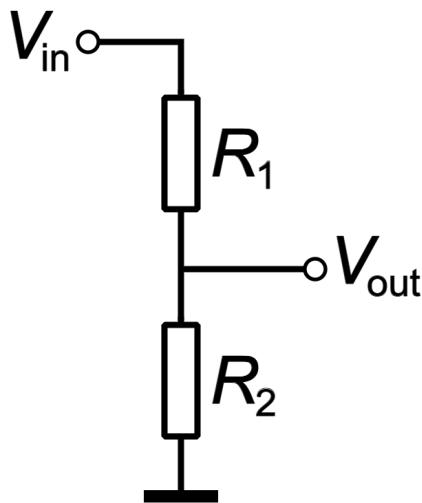


Figura 2.12 - circuito partitore di tensione e dimostrazione

$$(1) V = R \cdot I \quad (2) I = \frac{V}{R}$$

$$(3) V_{in} = V_{R_1} + V_{R_2} = I \cdot (R_1 + R_2)$$

$$(4) V_{out} = V_{R_2} = I \cdot R_2 = V_{in} \cdot \frac{R_2}{R_1 + R_2}$$

Il secondo componente utilizzato è il convertitore da analogico a digitale **ADS1115** [26], al quale viene inviato un segnale di tensione continua compreso tra 0 e 5 V. Questo modulo è in grado di trasformare un segnale analogico, cioè continuo nel tempo, in un valore digitale discreto che una qualsiasi MCU è in grado di interpretare.

In molti casi, come nel nostro progetto o nell’impiego di sensori di temperatura o di rilevamento della luce ambientale (fotocellule), **i sensori non generano direttamente segnali digitali**, ma restituiscono in uscita una tensione analogica proporzionale alla grandezza fisica misurata.

Il convertitore ADS1115 acquisisce questo segnale analogico e lo discretizza, assegnandogli un **valore intero ben definito**: ad esempio, una tensione di 2,73 V viene convertita in un numero intero (es. 35800), perdendo ovviamente gli **infiniti valori intermedi** non rappresentabili nel dominio digitale. La misura così ottenuta viene inviata alla MCU tramite l’interfaccia di comunicazione **I2C** (pin SDA e SCL), che sarà approfondita in una sezione successiva.

Il modello ADS1115 fornisce una risoluzione a 16 bit, permettendo quindi di distinguere tra \$2^{16} = 65\,536\$ **livelli distinti** all’interno del range massimo accettato (0–5 V). È importante notare che questo componente può essere utilizzato con altri sensori solo se il segnale in ingresso rientra nel range di tensione supportato; in caso contrario, è necessario inserire un **partitore di tensione** tra il sensore e il convertitore per adattare l’ampiezza del segnale.



Figura 2.26: voltage sensor

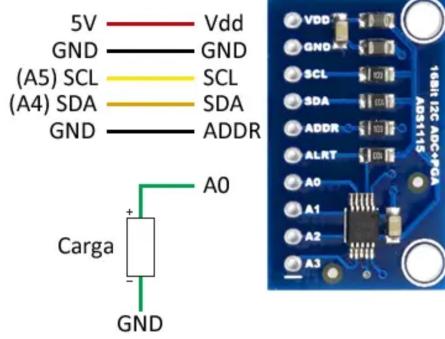


Figura 2.27: ASD1115

2.3.2 Accelerometro

L'**ADXL345** [27] è un accelerometro digitale a **tre assi**, in grado di misurare l'accelerazione applicata lungo gli assi *X*, *Y* e *Z*. L'asse *Z*, in particolare, è quello verticale e registra costantemente l'**accelerazione gravitazionale** terrestre (circa $1g = 9,81 \text{ m/s}^2$), valore che dovrà essere compensato a livello software per ottenere misurazioni accurate. Tale compensazione è spesso già integrata nelle principali librerie software disponibili per il dispositivo.

Il principio di funzionamento dell'ADXL345 si basa sulla tecnologia **MEMS** (Micro-Electro-Mechanical Systems) [28], una struttura meccanica su **scala micrometrica** realizzata in silicio. In particolare, all'interno del chip è presente una **massa mobile** (*proof mass*) sospesa da microscopiche molle (*spring beams*) che reagiscono ai movimenti del dispositivo. Le leggi della fisica quotidiana funzionano anche alle dimensioni di un acaro della polvere, quando avviene un'accelerazione, la **massa tende a restare in quiete** (per inerzia), deformando le molle nella direzione opposta rispetto al movimento. Lo spostamento viene rilevato tramite variazioni di capacità elettrica nei condensatori integrati, e convertito in un **segnale digitale proporzionale all'accelerazione** subita secondo la legge $F = m \cdot a$. Questo processo avviene internamente al sensore, quindi non è necessario un convertitore analogico-digitale esterno.

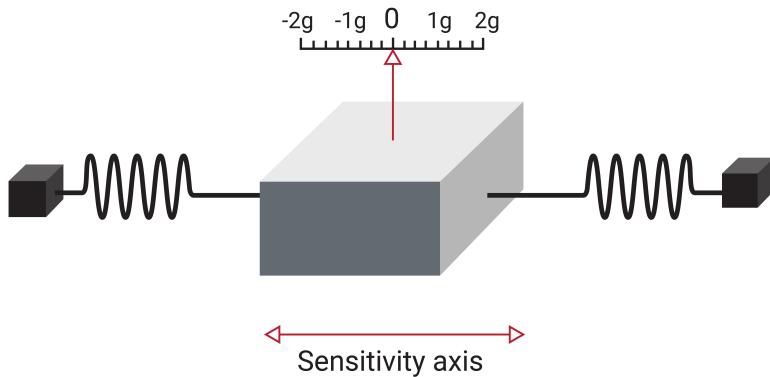


Figura 2.8: Sistema MEMS integrato in un accelerometro digitale

Specifiche tecniche principali [27]:

- **Intervallo di misura:** $\pm 2g$, $\pm 4g$, $\pm 8g$, $\pm 16g$ — selezionabile tramite software.
Rappresenta l'accelerazione massima misurabile per ciascun asse (valori negativi per spostarsi in entrambe le direzioni).
- **Risoluzione:** da 13 a 16 bit. Ad esempio, con un range impostato a $\pm 16g$ (intervallo totale di $32g$) e risoluzione a 13 bit (8192 livelli), si ottiene una sensibilità di $32/8192 = 0,0039g$ per livello. Se invece si impone un range inferiore come $\pm 2g$, la sensibilità aumenta a $0,0005g$, permettendo di rilevare **accelerazioni molto piccole**.

Quindi, considerando che il nostro veicolo ha una velocità massima di circa 20km/h (ossia $\approx 5,5m/s$), se raggiunge tale velocità in 1 secondo, si ha un'accelerazione di $5,5 m/s^2 \approx 0,56g$, oppure $0,28g$ in 2 secondi. Quindi, un range di $\pm 2g$ o $\pm 4g$ è perfettamente sufficiente.

Anche questo sensore comunicherà i suoi dati tramite il **collegamento I2C**, con indirizzo differente rispetto al sensore precedente. La **frequenza di campionamento** standard è dai 50 ai 200 Hz, anche se può essere spinta fino a 3200Hz tramite il software. Il dispositivo inoltre, è in grado di **calcolare le rotazioni sugli assi** tramite semplici calcoli di algebra lineare, che però nel nostro sistema non sono stati sviluppati.

Calibrazione e offset iniziale.

Durante i test iniziali, quando il veicolo era fermo, ci si attendeva di leggere valori prossimi a $(0, 0, 1)g$. In realtà, sono stati rilevati valori come $(0,12g, -0,07g, 0,98g)$. Queste deviazioni sono dette *offset iniziali* e sono dovute al rumore **intrinseco del sensore**, comune nei modelli non professionali.

Per correggerli, è stata implementata una **procedura di calibrazione**: nelle fasi iniziali il veicolo resta immobile per alcuni secondi, durante i quali vengono raccolti più campioni di cui si ottiene la media.

L'offset così calcolato viene sottratto ai valori futuri. Per l'asse Z è inoltre necessario compensare la forza gravitazionale sottraendo $1g$ dal valore iniziale.

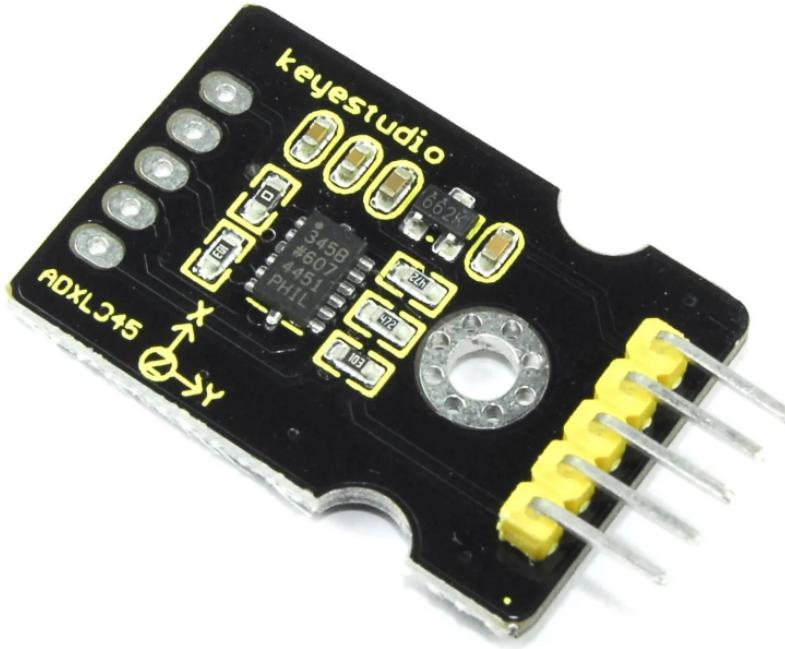


Figura 2.9: Accelerometro ADXL345 montato sul veicolo

2.3.3 Sensore a ultrasuoni

Il sensore a ultrasuoni rappresenta un componente centrale nello sviluppo del progetto, in quanto permette di **misurare la distanza** tra il veicolo e un eventuale ostacolo frontale, utilizzando una tecnologia economica, leggera e facilmente integrabile con la MCU.

Non potendo impiegare soluzioni più avanzate come LiDAR o laser scanner, è stata adottata una soluzione più semplice e accessibile: il sensore **HC-SR04** [29] a ultrasuoni. Questo dispositivo è composto da un emettitore e un ricevitore di onde sonore ad alta frequenza, generalmente intorno ai 40 kHz. Il suo funzionamento si basa sullo stesso principio dell'**ecolocalizzazione** utilizzata da alcuni animali, come i pipistrelli.

In particolare, alla ricezione di un segnale di comando (*trigger*), il sensore emette un'onda sonora per la durata di 10 microsecondi. Se questa onda colpisce un ostacolo, rimbalza e torna indietro, venendo rilevata dal microfono (*echo*) integrato. **La velocità del suono in aria** a temperatura ambiente (20°C) è di circa 343 m/s . Conoscendo questa velocità e misurando via software il tempo impiegato dal segnale per compiere il tragitto andata e ritorno, è possibile calcolare la distanza dell'ostacolo nel modo seguente:

$$\text{Distanza} = \frac{\text{VelocitàDelSuono} \times \text{TempoDiRimbalzo}}{2}$$

Esempio pratico: Se il tempo di ritorno del segnale è di $10\text{ ms} = 0,01\text{ s}$, la distanza sarà:

$$d = \frac{343 \cdot 0,01}{2} = 1,7\text{ m}$$

Data la semplicità e il basso costo del dispositivo, esistono alcune limitazioni tecniche. Il range operativo è compreso tra 5 cm e 400 cm , con un errore medio di circa $0,5\text{ cm}$, che può considerarsi accettabile per il nostro scopo.

Tuttavia, si riscontrano alcune criticità:

- **Frequenza di aggiornamento:** finché il sensore non riceve il segnale di ritorno (o non scade il timeout interno per segnale perso), non è possibile emettere un nuovo impulso. Questo introduce una latenza tra una misurazione e la successiva, limitando la frequenza di aggiornamento a circa 100–150 ms. In scenari di guida rapida o controllo autonomo, ciò potrebbe essere insufficiente.
- **Interferenze:** l'utilizzo di onde sonore lo rende suscettibile a disturbi provenienti da motori, altri sensori vicini o rumore ambientale, portando a misurazioni errate o falsi positivi.
- **Angolo di rilevamento:** il fascio ultrasonico ha un'ampiezza di soli 15° , pertanto il sensore è in grado di rilevare ostacoli solo in una direzione ben definita. Se la superficie dell'ostacolo non è perfettamente perpendicolare al sensore, il segnale rimbalza in direzione diversa e non viene recepito dal microfono, portando alla perdita del segnale, viene misurata una distanza infinita.

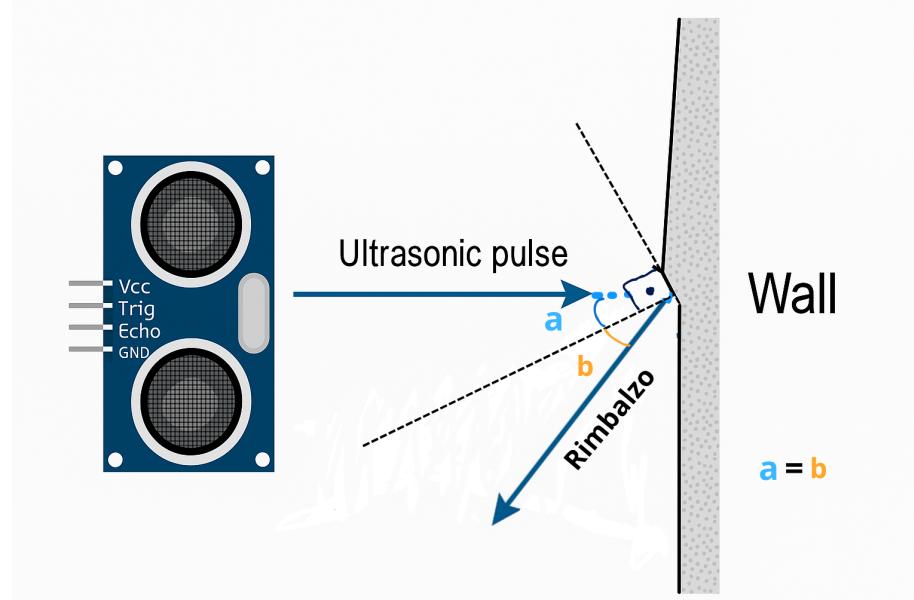


Figura 2.10: Teoria dei rimbalzi acustici

Nell'immagine (figura 2.12) è illustrato il principio della **riflessione speculare**: l'angolo di incidenza a tra l'impulso emesso e la **normale** (cioè il vettore perpendicolare alla superficie) è uguale

all’angolo b tra la normale e il **vettore di riflessione**. Affinché il segnale rientri nel cono di ricezione di 15° , la condizione da rispettare è $a + b \leq 15^\circ$, e quindi $a \leq 7,5^\circ$. Questo limita fortemente la capacità del sensore di rilevare **ostacoli inclinati** oltre quella angolazione.

Strategie di mitigazione: molti dei limiti sopra descritti non possono essere risolti direttamente. La soluzione adottata nel progetto prevede quindi l’impiego di **più sensori disposti in posizioni strategiche**, in grado di comunicare tra loro per eseguire triangolazioni e fornire una percezione più accurata dell’ambiente. In particolare, i sensori sono posizionati frontalmente, sul lato destro e sul lato sinistro, in modo da rilevare ostacoli in fase di avanzamento o svolta.

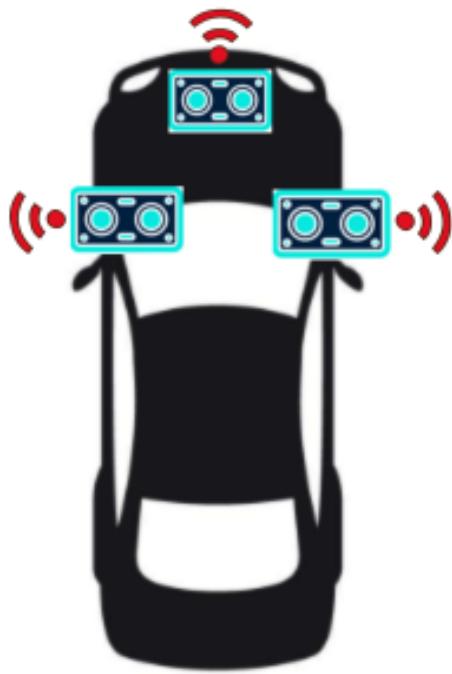


Figura 2.11: Disposizione dei sensori a ultrasuoni sul veicolo

Dai test effettuati, tale configurazione si è dimostrata sufficiente per identificare ostacoli in condizioni di guida manuale. Tuttavia, si segnala che in condizioni ideali sarebbe stato opportuno montare **ulteriori sensori**: ad esempio, due frontali con angolature diverse per aumentare il campo visivo, e almeno uno posteriore per supportare le manovre in retromarcia.

2.3.4 Buzzer acustico

Dopo aver trattato i sensori a ultrasuoni, è opportuno introdurre brevemente un componente di tipo attuatore, scelto per integrare e rendere più interattivo il sistema di rilevamento ostacoli: il **buzzer acustico** [30], noto anche come *cicalino*.

Questo dispositivo non rileva dati, ma emette un suono che può essere utilizzato come segnale di allarme o avviso per l’utente in modo interattivo. Esistono due principali tipologie:

- **Buzzer passivo**, che può produrre suoni articolati variando la frequenza attraverso un segnale PWM. È in grado di generare **melodie** simili a quelle dei vecchi telefoni cellulari, ma richiede una gestione più complessa via software.
- **Buzzer attivo**, che emette un suono fisso (generalmente 2.048 Hz) appena riceve un segnale elettrico continuo, sia a 5 V sia a 3.3 V. Questa versione è stata scelta nel progetto per la sua semplicità di utilizzo e per l'efficacia nella segnalazione di eventi, pur producendo un suono monotono e meno gradevole.

Il principio fisico alla base è quello del *trasduttore piezoelettrico*, il quale **oscilla a una frequenza costante** nel campo dell'udibile, convertendo l'energia elettrica in onde sonore percepibili dall'**orecchio umano**.

Nel progetto, il buzzer è stato integrato con i sensori a ultrasuoni: viene attivato ogni volta che un **ostacolo viene rilevato** a meno di 20 cm da uno dei sensori. In tal modo, l'allarme acustico fornisce un immediato feedback sonoro all'operatore, migliorando la consapevolezza durante la guida da remoto a brevi distanze e per eventuali test.

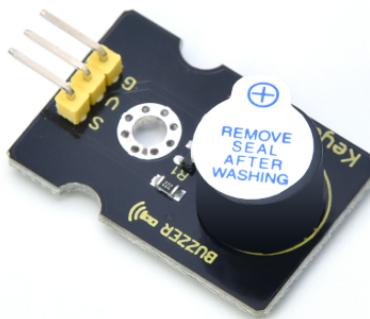


Figura 2.12: Buzzer acustico utilizzato nel progetto

2.3.5 Streaming audio e video

L'ultimo componente sensoristico implementato nel progetto è una semplice webcam, da sfruttare per lo streaming audio e video durante il pilotaggio da remoto o per l'analisi in tempo reale da parte di futuri agenti AI. Dopo aver valutato diverse opzioni, come la *Raspberry Pi Camera* — che presentava problemi legati alla collocazione fisica, all'assenza di microfono integrato e alla necessità di un supporto stampato in 3D — è stato deciso di optare per una normale **webcam USB**.

Il modello scelto è la **Logitech C270 HD**, dotata di microfono integrato. Sebbene non sia un dispositivo di fascia alta, presenta una **qualità video sufficiente** per gli scopi del progetto, **dimensioni contenute** e un peso ridotto che non incide significativamente sulla massa complessiva del veicolo. Collegata tramite USB-A, è in grado di catturare video in risoluzione 1280x720 (720p HD) fino a 30 fps. Tale risoluzione influenza sia sulla definizione delle immagini, sia sulla latenza percepita, in quanto l'**ampiezza del flusso video** determina la quantità di dati da trasmettere e processare, quindi una risoluzione ottima rispetto alle limitate capacità di banda che abbiamo.

La webcam ha un **campo visivo** (FOV) di circa 60°, che durante i test si è rivelato **limitante** nella guida remota: non consente una visione completa dei bordi del paraurti anteriore, rendendo difficoltose le manovre come le curve strette. La videocamera è posizionata frontalmente, al centro del veicolo, e non permette una **visuale periferica sufficiente** per un controllo preciso in ambienti ristretti.

Come introdotto nell'introduzione storica sulla sorveglianza, le prime telecamere (CCTV) erano basate su segnali analogici, ma oggi possiamo sfruttare tecnologie digitali per convertire le immagini in dati binari. Le webcam moderne utilizzano un sistema **optoelettronico** [31] che cattura la luce attraverso una lente e la proietta su una griglia di *fotodiodi*, organizzati secondo un *filtro di Bayer*. Ogni fotodiodo è sensibile a una **componente cromatica** (rosso, verde o blu) e produce un segnale elettrico proporzionale all'**intensità** della luce ricevuta.

Il processo di ricostruzione dell'immagine si chiama **demosaicing** [32], e consiste nell'**interpolazione delle informazioni acquisite** da ciascun filtro cromatico per ottenere l'intero spettro di colore in ogni pixel. Le intensità registrate vengono quindi discetizzate e digitalizzate, ogni tonalità viene mappata in uno dei **256 valori possibili**, ogni colore viene rappresentato da 8 bit, quindi ogni pixel peserà esattamente 24 bit (almeno nella codifica base RGB).

La webcam dispone di un processore interno che gestisce operazioni di bilanciamento del bianco, riduzione del rumore, correzione colore e infine **compressione del video**. Il formato di compressione adottato è il noto **H.264** [33], o *Advanced Video Coding* (AVC), ampiamente utilizzato in piattaforme di streaming come YouTube e Netflix, ma anche nei sistemi di videosorveglianza, trasmissioni televisive e videoconferenze (es. Zoom, Microsoft Teams).

La compressione H.264 opera con due modalità:

- **Intraframe**, dove ogni fotogramma viene compresso indipendentemente (simile a una fotografia);
- **Interframe**, dove vengono analizzate le **differenze tra fotogrammi** consecutivi, salvando solo le variazioni per ridurre drasticamente lo spazio occupato.

Senza compressione, un video in 720p può raggiungere circa 1 GB al minuto, mentre con H.264 può essere ridotto a **meno di 100 MB** senza perdita percettibile di qualità, grazie alla ricostruzione dei frame durante la decodifica.

Audio

Oltre al video, la webcam integra un **microfono** che è stato utilizzato per acquisire l'audio ambientale. Il funzionamento del microfono si basa su una *membrana* che vibra al passaggio delle onde sonore, trasferendo l'energia a dei condensatori che generano un **segnale elettrico proporzionale** alla pressione sonora. Questo segnale viene quindi amplificato, digitalizzato e compresso per la trasmissione.

Sviluppi futuri

Tra gli sviluppi futuri si ipotizza l'utilizzo di tecniche di intelligenza artificiale per il **filtraggio in tempo reale del rumore** generato dal movimento del veicolo, mantenendo però attenzione a non eliminare informazioni vocali potenzialmente rilevanti per le indagini (es. urla, parole chiave). Dal punto di vista video, si potrebbe implementare un sistema di **riconoscimento automatico di comportamenti illeciti** (es. rapine, aggressioni), aumentando le capacità di intervento preventivo del veicolo.



Figura 2.13: Webcam Logitech C270 HD utilizzata per streaming video e acquisizione audio

USB e porta seriale

Una **porta seriale** [34] è una tecnologia di comunicazione che consente la trasmissione dei dati *bit per bit*, in modo sequenziale. La comunicazione avviene tramite due linee: TX (trasmissione) e RX (ricezione), in modalità asincrona, cioè **senza bisogno di un segnale di clock condiviso**.

Le prime porte seriali classiche, di grandi dimensioni, come la **RS-232**, erano diffuse nei computer tra gli anni '80 e i primi 2000. Questi connettori (tipicamente a 9 o 25 pin, spesso con cavi grigi o blu) utilizzavano un protocollo molto semplice chiamato **UART** [35] (Universal Asynchronous Receiver/Transmitter). Da qui derivano anche le denominazioni di sistema come COM1, COM2, rimaste nei moderni sistemi operativi.

Con l'evoluzione tecnologica, le moderne comunicazioni seriali si sono spostate su protocolli più avanzati ed efficienti, come lo **USB** (Universal Serial Bus), oggi standard per il collegamento di dispositivi come tastiere, webcam, microfoni, memorie esterne. Lo standard USB implementa un protocollo di comunicazione strutturato, con gestione di pacchetti, indirizzi, negoziazione priorità e controllo degli errori, spesso deve interagire direttamente con i driver proprietari dei dispositivi.

È importante notare che, sebbene entrambi i sistemi trasmettano dati in modo seriale, l'**USB non è direttamente compatibile** con UART né elettricamente (5 V vs ± 12 V) né logicamente: per comunicare tra un dispositivo UART e una porta USB è necessario un **chip/adattatore** di conversione (es. FTDI, CH340, CP2102).

La comunicazione via USB, è oggi lo standard principale per collegare periferiche esterne a un computer, come mouse, tastiere, webcam, microfoni, memorie esterne, microcontrollori, ecc. Tuttavia, il modo in cui i dispositivi USB vengono **gestiti dal sistema operativo** dipende da numerosi fattori: la presenza o meno di driver specifici, il tipo di dispositivo e il sistema operativo stesso.

Gestione interna del sistema operativo

Quando un dispositivo viene collegato a una porta USB fisica della scheda madre, il **kernel** del sistema operativo rileva l'evento e avvia una procedura di enumerazione del dispositivo salvando le informazioni principali :

- la classe del dispositivo (es. memoria di massa, dispositivo HID, CDC, ecc.);
- il produttore e l'identificativo univoco;
- se è richiesto un **driver generico** (standard) oppure un **driver specifico** proprietario.

Se il sistema dispone già di un driver compatibile (come accade per mouse, tastiere, Arduino, ecc.), il dispositivo viene riconosciuto e montato automaticamente. In caso contrario, può essere richiesta l'**installazione manuale** di un driver.

Rappresentazione dei dispositivi USB in Linux e macOS

In ambienti UNIX-like come **Linux** e **macOS**, i dispositivi collegati via USB vengono rappresentati come **file speciali** all'interno della directory `/dev/`. Alcuni esempi comuni includono:

- `/dev/ttyUSB0, /dev/ttyUSB1` — vecchi dispositivi UART che tramite **adattatori USB-seriale**, come vecchi modelli gps seriali, interfacce macchinari industriali, telecamere termiche, strumenti da laboratorio...;
 - `/dev/ttyACM0, /dev/ttyACM1` — vecchi dispositivi UART appartenti alla classe CDC (Communication Device Class) collegati direttamente tramite usb **senza adattatore** ma grazie al protocollo **ACM** simulano la porta seriale, come Arduino o altri microcontrollori simili.
 - `/dev/sda1, /dev/sdb1` — **memorie di massa** USB, come pen drive e hard disk esterni;
 - `/dev/input/eventX, /dev/hidraw1` — dispositivi moderni **HID (human interface device)** come mouse, tastiere, gamepad. Generalmente questi non hanno bisogno di driver specifici, se compatibili vengono riconosciuti immediatamente.
- `/dev/audi01, /dev/Video01` invece sono per i **dispositivi audio e video**, spesso necessitano dell'installazione di loro driver proprietari.

Quindi da console è possibile:

- **Vedere tutti i dispositivi connessi:**

```
ls /dev/tty*
```

- **Inviare dati** come se fosse un file:

```
echo "dati" > /dev/ttyUSB0
```

- **Leggere dati** come se fosse un file:

```
cat /dev/ttyUSB0
```

Tramite Python è possibile gestire le porte seriali utilizzando la libreria `pyserial`:

```
import serial

ser = serial.Serial('/dev/ttyUSB0') # Apri la connessione con la porta
ser.write(b'dati')
risposta = ser.readline().decode()
ser.close() # Chiudi la connessione con la porta
```

Gestione dei dispositivi USB in Windows

Nel sistema operativo **Windows**, i dispositivi USB non sono rappresentati come file nel filesystem. La gestione delle periferiche avviene tramite il *Device Manager* e il *Kernel Plug and Play*.

Quando un dispositivo USB viene collegato:

- COM1, COM2, COM3... — dispositivi seriali che simulano le vecchie porte UART, collegati tramite **adattatori USB-seriale** (es. convertitori USBRS232). Il sistema operativo assegna loro un numero crescente (COM3, COM4...) man mano che vengono connessi. Sono tipici di: vecchi GPS, strumenti da laboratorio, interfacce industriali, microcontrollori.

A differenza di Linux, **anche i dispositivi** collegati direttamente via USB che utilizzano il **proto-collo ACM** per simulare una porta seriale (come Arduino) ricevono comunque un numero COM (**COMx**) e sono gestiti allo stesso modo dei dispositivi con adattatore.

- **Memorie di massa USB** — come chiavette USB e hard disk esterni. Su Windows non sono rappresentate da file speciali, ma vengono assegnate **lettere di unità** (es. E:, F:, ecc.). Possono essere visualizzate tramite **Gestione disco** o **Esplora file**.
- **Dispositivi HID (Human Interface Device)** — come mouse, tastiere, controller. Sono riconosciuti automaticamente da Windows grazie ai driver HID standard integrati nel sistema operativo. **Non viene loro assegnato un nome** di file visibile all’utente, ma possono essere gestiti e visualizzati nella sezione “**Gestione dispositivi**”, dove sono ordinati per **categoria** (es. dispositivi di acquisizione audio/video, mouse, tastiere, gamepad...).

Per **comunicare con le porte COM in Windows**, il funzionamento è molto simile a quanto avviene in Linux: si utilizzano gli stessi comandi della libreria `pyserial` in Python. In effetti, l’interazione tramite PowerShell risulta più macchinosa rispetto a quella tramite terminale Unix, motivo per cui è spesso preferibile utilizzare Python per una gestione diretta e multipiattaforma.

Diversamente, i dispositivi USB con driver generici, come quelli appartenenti alla classe **HID** (Human Interface Device), **non vengono mappati su un file visibile** (come avviene con `/dev/hidraw*` su Linux) e non sono associati direttamente a un nome di dispositivo accessibile. L’interazione con questi dispositivi richiede l’accesso al processo specifico collegato alla categoria hardware (mouse, tastiera, gamepad, ecc.).

Per farlo, si può scegliere tra due approcci:

- utilizzare le **system call** di basso livello fornite dalle **API di Windows**, come `CreateFile()`, `ReadFile()`, `WriteFile()`;
- utilizzare librerie di alto livello come `hidapi`, disponibile anche in Python, che si appoggia internamente alle **Windows API** per interagire con i dispositivi HID.

Un esempio base in Python con hidapi:

```
import hid

# Stampa tutti i dispositivi HID collegati
for device in hid.enumerate():
    print(device)

# Apre il dispositivo specifico (es. Logitech C270)
dev = hid.device()
dev.open(0x046D, 0xC52B) # Vendor ID e Product ID

# Legge fino a 64 byte
data = dev.read(64)
print("Dati ricevuti:", data)

dev.close()
```

In questo esempio, i valori 0x046D e 0xC52B corrispondono rispettivamente al **Vendor ID** e al **Product ID** del dispositivo. Tali identificativi possono essere recuperati tramite il comando `hid.enumerate()` o usando strumenti come *Gestione dispositivi* di Windows.

È importante non confondere le **porte seriali UART con le porte VGA**: le prime servono per la trasmissione di dati digitali un bit alla volta (tipicamente per comunicazione tra dispositivi), mentre le porte VGA sono destinate alla trasmissione di segnali video analogici (RGB) verso un monitor.

Un'altra porta storica è la **porta parallela**, usata soprattutto per collegare stampanti e scanner. A differenza della seriale (1 bit alla volta), la porta parallela permetteva la trasmissione **simultanea di 8 bit**, risultando inizialmente più veloce, ma meno affidabile su lunghe distanze.

Nel tempo, queste interfacce sono state progressivamente sostituite da tecnologie più veloci e versatili, come le porte USB. A partire dai connettori USB-A e USB-B (oggi quasi obsoleti), si è giunti alle moderne USB-C e Thunderbolt, capaci di supportare non solo alimentazione e dati, ma anche segnali video/audio, con velocità che superano i **gigabit al secondo**.

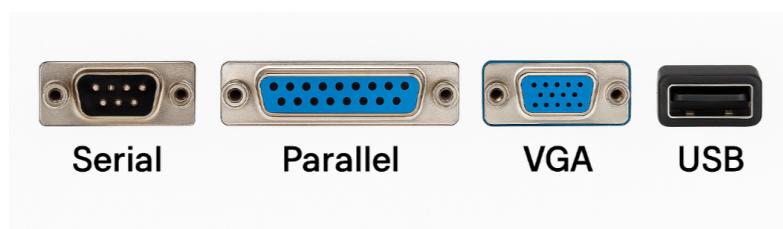


Figura 2.14: Porte per il trasferimento dei dati

2.3.6 GPS

Come ultimo sensore, è opportuno trattare il modulo **GPS** [36], anche se non è stato integrato fisicamente nel progetto nativo. Tuttavia, esso è presente di default all'interno del *kit Navio2*, il sistema per l'automazione di veicoli autonomi che verrà analizzato nella parte finale della tesi.

Il termine **GNSS** [37] (Global Navigation Satellite System) è una denominazione generica che comprende tutti i sistemi satellitari attualmente esistenti per la navigazione globale, tra cui:

- **NAVSTAR GPS** (Stati Uniti)
- **GLONASS** (Russia)
- **Galileo** (Unione Europea)
- **BeiDou** (Cina)

Questi sistemi si basano su costellazioni di satelliti posizionati in **orbita geosincrona** (precisamente in *MEO*, Medium Earth Orbit) a un'altitudine di circa 20.000 km. Ogni satellite compie due orbite complete ogni 24 ore. Il numero di satelliti è elevato (oltre 30 per ciascun sistema), e la loro distribuzione è tale da garantire la visibilità simultanea di **almeno quattro satelliti da qualsiasi punto sulla superficie terrestre**.

Il principio di funzionamento si basa sulla trasmissione continua di **due dati fondamentali** da parte dei satelliti:

- la loro **posizione orbitale** (ephemeris);
- l'**orario preciso**, misurato tramite orologi atomici a bordo.

Il ricevitore GPS sulla superficie, confrontando il **tempo** di invio e ricezione del segnale, può **calcolare la distanza da ciascun satellite**. **Intersecando le quattro sfere** definite dalle rispettive distanze, è possibile determinare la **posizione tridimensionale** del dispositivo (longitudine, latitudine e altitudine).

Il sistema GPS statunitense ha un errore medio di pochi metri, ma grazie a tecniche di **correzione differenziale** (*Differential GPS*), è possibile ridurre tale errore a pochi centimetri (sub-meter accuracy).

Una volta connesso il modulo GPS tramite porta seriale (USB o UART), è possibile leggere le informazioni provenienti dai satelliti accedendo direttamente alla porta (es. `/dev/ttyUSB0` o COM3).

Il formato standard utilizzato è l'**NMEA 0183** (National Marine Electronics Association), originariamente concepito per l'ambito nautico, ma oggi ampiamente diffuso anche nei veicoli terrestri. I messaggi sono scritti in ascii, quindi leggibili anche dal uomo, iniziano tutte con una parola nel formato \$GPRMC,

di cui le **prime 2 lettere** rappresentano la **costellazione** GP(USA) , GL(RUSSIA), GA(Europa)...
Ogni messaggio iniziale corrisponde a una struttura diversa di dati:

Frase	Nome	Contenuto
\$GPGGA	Fix Data	Ora, latitudine, longitudine, tipo di fix, n. satelliti, altitudine, HDOP
\$GPRMC	Min. data	Ora, stato (A/V), latitudine, longitudine, velocità (nodi), direzione, data
\$GPGSV	Sat View	N. satelliti visibili, elevazione, azimut, SNR di ciascun satellite
\$GPGSA	Sat Active	Tipo fix (2D/3D), ID satelliti usati, PDOP, HDOP, VDOP
\$GPVTG	Track Speed	Direzione di movimento, velocità in nodi e km/h

Tabella 2.1: Principali frasi NMEA e loro contenuto

Tra queste, la frase \$GPRMC è la più comunemente utilizzata, poiché contiene le informazioni essenziali per la navigazione. Dopo l'intestazione \$GPRMC, i dati sono organizzati in una sequenza di campi separati da virgole:

Campo	Significato	Esempio	Note
1	Header	\$GPRMC	Identifica il tipo di messaggio
2	Orario UTC	144326.00	Formato hhmmss.ss
3	Status	A	A = valido, V = non valido
4	Latitudine	5107.001773	Formato DDmm.mmmm
5	Direzione latitudine	N	N = nord, S = sud
6	Longitudine	11402.3291611	Formato DDDmm.mmmm
7	Direzione longitudine	W	E = est, W = ovest
8	Velocità sopra il suolo (knots)	0.080	1 nodo = 1.852 km/h
9	Rotta vera (True)	323.3	In gradi rispetto al Nord vero
10	Data	210307	Formato ddmmyy
11	Variazione magnetica	0.0	In gradi (opzionale)
12	Direzione var. magnetica	E	E o W (opzionale)
13	Indicatore modalità (opz.)	A	Autonomo, Differenziale, Stimato, Nessuno
14	Checksum	*20	Verifica di integrità

Tabella 2.2: Campi della frase NMEA \$GPRMC

È importante notare che le frasi NMEA contengono solo i dati ricevuti dal dispositivo GPS a terra: **non vi sono informazioni dettagliate sui satelliti**, come le loro orbite individuali o frequenze.

L'utilizzo del modulo GPS presenta alcune criticità:

- **ostacoli fisici** (edifici, alberi, tunnel) o condizioni meteo avverse possono causare perdita o degradazione del segnale;
- un **orario errato** all'interno del ricevitore può compromettere la corretta ricezione dei segnali;
- alla **prima accensione**, il modulo può impiegare da 1 a 2 minuti per ottenere un fix stabile, producendo dati non accurati inizialmente.

Infine, un semplice script in Python consente di leggere i dati da una porta seriale, filtrando solo \$GPRMC:

```
import serial

ser = serial.Serial('/dev/ttyUSB0', 9600) # o 'COM3' su Windows

while True:
    riga = ser.readline().decode('ascii', errors='replace')
    if riga.startswith('$GPRMC'):
        print(riga)
```



Figura 2.15: Modulo GPS economico USB

2.4 Microcontrollore (MCU)

La scelta della microcontrollore (MCU) rappresenta un momento cruciale nello sviluppo del sistema, poiché costituisce il “cervello” dell’intero veicolo. La MCU ha il compito di interfacciarsi con tutti i **sensori e attuatori**, elaborare i dati in tempo reale, gestire le **comunicazioni di rete**, e fornire una risposta coerente e tempestiva agli input ricevuti. Oltre a ciò, il sistema deve supportare la **trasmissione audio/video** in tempo reale, lavorare in **multithread** ed essere sufficientemente performante, pur mantenendo consumi energetici contenuti, essendo **alimentato a batteria**.

2.4.1 Criteri di selezione

I criteri principali che hanno guidato la scelta dell’MCU sono:

- Capacità computazionale (RAM, numero di core, clock).
- Supporto al multithreading e gestione contemporanea di più task.
- Presenza di connettività Wi-Fi integrata.
- Disponibilità di pin GPIO per interfacciare sensori e attuatori.
- Bassi consumi in idle e operativi.
- Supporto a linguaggi ad alto livello (es. Python) e sistemi operativi Linux-based.

2.4.2 Valutazione dei microcontrollori

Sono stati inizialmente analizzati microcontrollori di fascia bassa, apprezzati per i loro consumi ridotti ma **penalizzati in termini di potenza computazionale**.

STM32 (es. STM32F103)

- CPU: 1 core a 72 MHz.
- RAM: 20 kB.
- GPIO: 37.
- Consumo in standby: $\approx 2 \mu\text{A}$.
- Wi-Fi: assente.

Questa soluzione è stata esclusa per l'**assenza di connettività** wireless integrata e per l'insufficiente capacità computazionale.

ESP8266

- CPU: 1 core a 80 MHz.
- RAM: 80 kB utilizzabili.
- GPIO: 17.
- Wi-Fi: integrato (2.4 GHz).
- Consumo in standby: $\approx 20 \mu\text{A}$.

Nonostante l'integrazione Wi-Fi, l'ESP8266 non dispone della **potenza necessaria** per elaborazioni in parallelo, trasmissione audio/video e gestione sensori in tempo reale.

ESP32

- CPU: 2 core a 240 MHz.
- RAM: fino a 520 kB.
- GPIO: ≈ 34 .
- Wi-Fi: integrato (2.4 GHz).
- Consumo in standby: $\approx 10 \mu\text{A}$.

L'ESP32 rappresenta un significativo passo avanti rispetto all'ESP8266 e sarebbe adatto per progetti IoT di media complessità. Tuttavia, l'**assenza di un sistema operativo completo** e la limitata capacità RAM/CPU rendono complessa la gestione di video/audio streaming e multitasking avanzato.

2.4.3 Valutazione dei microcomputer

Per applicazioni più complesse è opportuno orientarsi verso sistemi basati su microcomputer (SBC – Single Board Computer), in grado di eseguire un sistema operativo (es. Linux), offrendo maggiore flessibilità, multitasking avanzato e supporto a linguaggi di alto livello come Python.

Raspberry Pi Zero W

- CPU: 1 core a 1 GHz.
- RAM: 512 MB.
- GPIO: 40.
- Wi-Fi: 2.4 GHz.
- Consumo medio: ≈ 100 mA.

Soluzione molto **compatta e a basso consumo**. Tuttavia, la CPU single-core e la RAM limitata possono rappresentare un **collo di bottiglia** nelle operazioni intensive, come lo streaming video/audio.

Raspberry Pi Zero 2 W

- CPU: 4 core (ARM Cortex-A53 a 1 GHz).
- RAM: 512 MB.
- GPIO: 40.
- Wi-Fi: 2.4 GHz.
- Consumo medio: ≈ 100 mA.

Miglioramento significativo rispetto al modello precedente, mantenendo consumi ridotti. **Potenzialmente adatto al progetto**, anche se servirebbero ottimizzazioni specifiche per ridurre il carico della CPU.

2.4.4 Raspberry pi 3B +

- CPU: 4 core (ARM Cortex-A53 a 1.4 GHz).
- RAM: 1 GB.
- GPIO: 40.
- Wi-Fi: dual band 2.4 GHz / 5 GHz.
- Consumo medio: ≈ 400 mA.

Il Raspberry Pi 3B+ [38] è stato selezionato per via delle sue elevate prestazioni, che permettono di gestire:

- Server UDP per il controllo remoto.
- Invio di molti dati sensoriali in poco tempo.
- Elaborazione dati da 7 sensori in multithread.
- Streaming video e audio in tempo reale.

Il maggior consumo rispetto ai microcontrollori è **giustificato dalla flessibilità, affidabilità e semplicità** di sviluppo offerte dal sistema operativo.

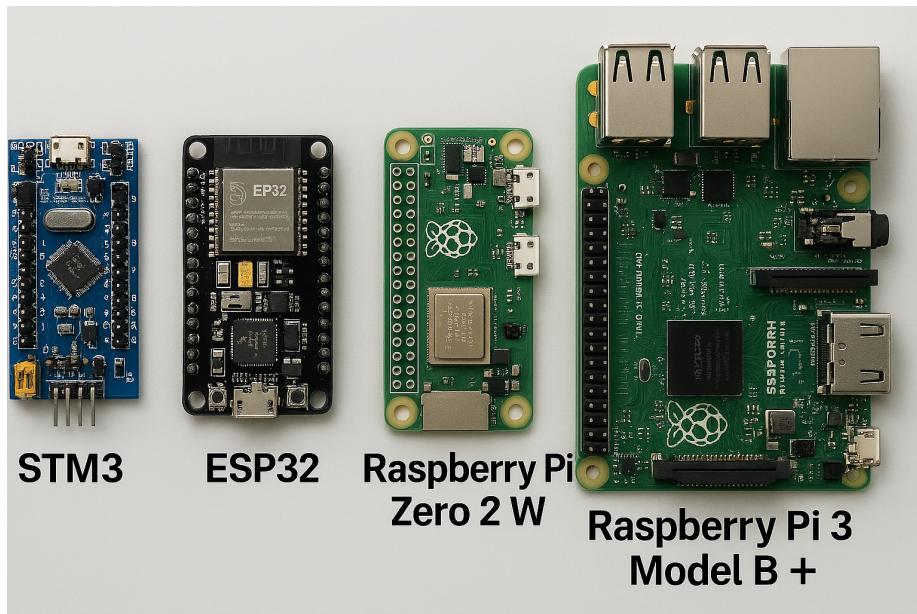


Figura 2.16: Microcontrollori e Microprocessori

È vero che le **elaborazioni più onerose saranno effettuate su un server remoto** — con prospettive future che includono anche l'impiego di modelli di machine learning di grandi dimensioni — tuttavia, date le prestazioni del sistema, se in futuro decidessi di implementare **modelli direttamente in locale**, non dovrei incontrare particolari difficoltà. Inoltre, esistono diversi studi che stanno applicando soluzioni simili ai droni aerei, i quali sono limitati da batterie più piccole. Nel mio caso, essendo il consumo dei motori ridotto, è anche possibile sostituire l'alimentazione con una più performante, riducendo i problemi legati all'assorbimento di corrente del processore, visto che questo chip consuma leggermente di più.

Specifiche	Dettagli
CPU	Broadcom BCM2837B0, quad-core ARM Cortex-A53, 64-bit @ 1.4 GHz
RAM	1 GB LPDDR2 SDRAM
GPU	Broadcom VideoCore IV
Storage	MicroSD (supporta anche boot da USB o rete)
Wireless	Wi-Fi 802.11 b/g/n/ac dual-band (2.4 GHz e 5 GHz), Bluetooth 4.2, BLE
Porte USB	4× USB 2.0
Ethernet	Gigabit Ethernet (via USB 2.0 bridge, max 300 Mbps effettivi)
GPIO	40 pin (pieno supporto compatibile con modelli precedenti)
Video/audio output	HDMI full-size, jack da 3.5 mm (audio stereo + video composito)
Interfacce	CSI (Camera Serial Interface), DSI (Display Serial Interface), UART, SPI, I2C
Alimentazione	5V 2.5A via micro-USB (consumo medio tra 2W e 6W a seconda del carico)
Sistema operativo	Supporta Raspberry Pi OS (ex Raspbian), Ubuntu, Kali Linux, ecc.

Tabella 2.3: Caratteristiche tecniche del Raspberry Pi 3 Model B+

Per questo progetto è stato installato il sistema operativo **Raspbian OS**, una distribuzione Linux ottimizzata per Raspberry Pi. Raspbian supporta **chiamate di sistema dirette** per inviare impulsi ai pin, il controllo dei canali I2C, controllo **qualità alimentazione** (per prevenire malfunzionamenti), i driver di rete per il Wi-Fi all'avvio e la **configurazione SSH** al primo boot. Grazie all'integrazione di **Avahi**, è possibile accedere al dispositivo tramite nome host (es. `raspberrypi.local`) anche senza conoscere l'indirizzo IP.

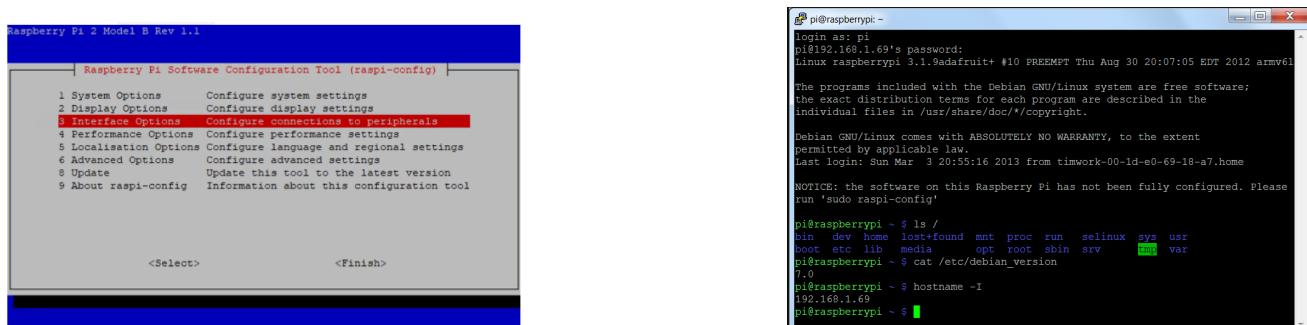


Figura: Configurazione iniziale con `raspi-config` e accesso remoto tramite SSH.

Il sistema mantiene tutte le funzionalità di un **ambiente Linux**, come l'uso di pip e apt per l'installazione di librerie, il supporto a dispositivi audio/video, la compatibilità con Python e relative **librerie** (pygame, OpenCV, GStreamer, asyncio, ecc.), nonché il supporto al **multithreading**. Sono stati condotti test con fino a 7 thread simultanei senza mai sovraccaricare la CPU. Inoltre, Raspbian supporta **comunicazioni tramite protocolli** HTTP, MQTT, TCP e UDP, oltre all'utilizzo di VPN, crittografia e **firewall** per la sicurezza.

2.4.5 GPIO e Pinout del Raspberry Pi

Per quanto riguarda l'hardware, la scheda include **interfacce dedicate**: una per la fotocamera Raspberry (CSI), una per il display (DSI), una porta HDMI e una porta Ethernet. La restante parte della scheda è occupata dai pin **GPIO** (General Purpose Input/Output), disposti in verticale a destra su due file da 20 pin ciascuna.

I pin oltre a fornire input digitali, posseggono anche **funzionalità specifiche**:

- canali PWM (pin 32 , 12 , 33 , 35),
- massa (GND),
- canali I2C (pin 3 , 5),
- altre funzionalità come SPI, UART ...

I pin 1, 2 e 4 sono particolari: forniscono **tensione continua** (3.3V o 5V) e possono anche essere utilizzati per **alimentare il Raspberry stesso**, come è stato fatto nel circuito finale del progetto, tramite il già citato convertitore buck.

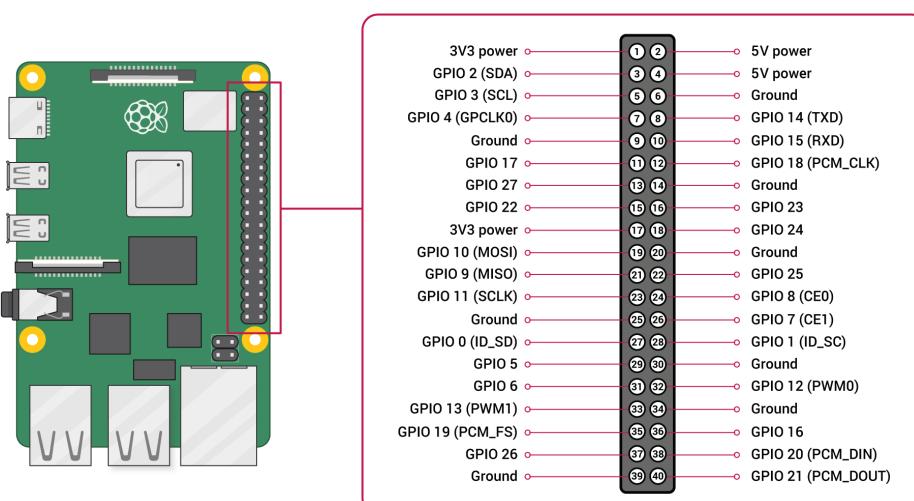


Figura: Schema pinout del Raspberry Pi.

Protocollo I2C

Poiché il protocollo I2C [39] è utilizzato più volte nel progetto (accelerometro e ADS1115), si fornisce una spiegazione generale. L'I2C è un protocollo **seriale sincrono** per la comunicazione tra dispositivi, che condivide un bus a due fili: SDA (dati) e SCL (clock).

Il **master** (in questo caso il Raspberry Pi) avvia la comunicazione, fornendo il ritmo di trasmissione. Ogni dispositivo **slave** risponde solo al proprio indirizzo (7-10 bit), con un **massimo teorico di 128 indirizzi** (alcuni riservati). Dopo il messaggio iniziale, lo slave invia un segnale di **riconoscimento** (ACK).

Il protocollo è efficiente, supporta più slave sullo stesso bus, ma ha una portata limitata e richiede attenzione in caso di **conflitti di indirizzo**. La comunicazione è **bidirezionale**.

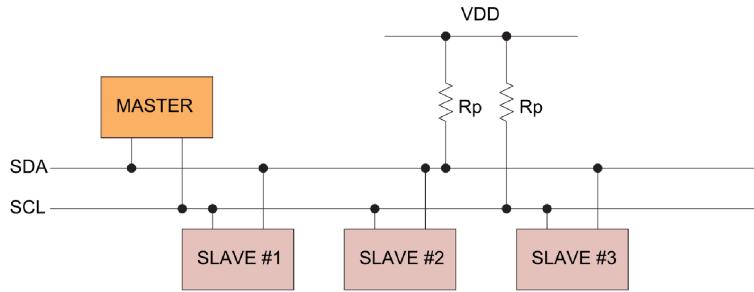


Figura: Schema del funzionamento del protocollo I2C.

2.4.6 Circuito finale

Segue lo schema completo del circuito, che mostra la disposizione finale dei pin.

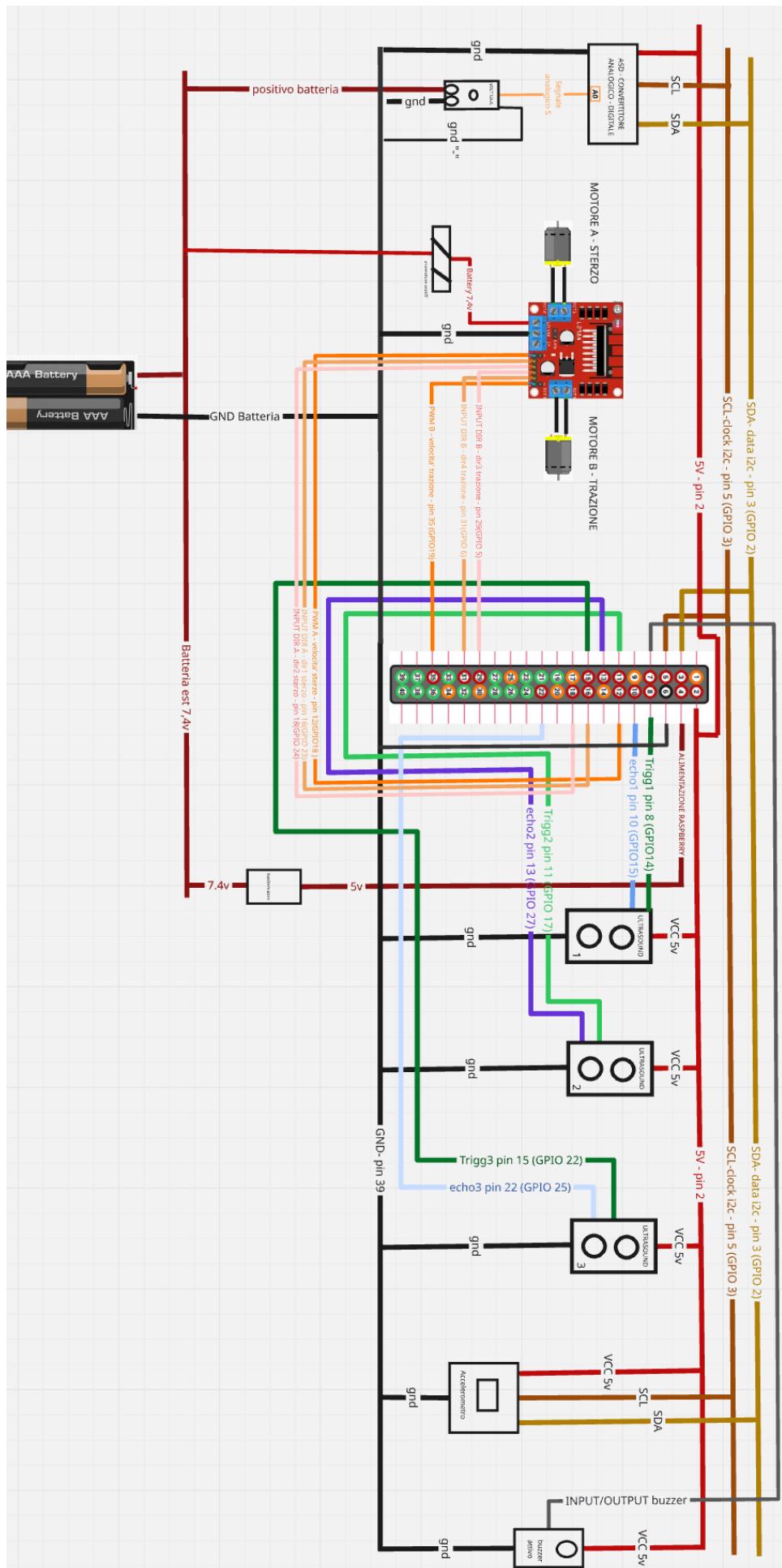


Figura 2.17: Schema del circuito totale implementato

2.5 Stack di Comunicazione

In questa sezione si approfondiscono le tecnologie adottate ai **vari livelli dello stack di comunicazione**, descrivendo le scelte progettuali relative alla trasmissione dei dati, differenziate in base alla tipologia (comandi, telemetria, streaming video/audio) e ai requisiti di **latenza, banda e affidabilità**.

2.5.1 Livello Fisico e Datalink

Il livello fisico e il livello datalink rappresentano la base della comunicazione, ovvero il mezzo e le modalità con cui i dati vengono effettivamente trasmessi. Stabiliscono il tipo di **codifica**, l'**accesso al mezzo**, gestione dei **turni di trasmissione** ecc..

Nel contesto del progetto, che richiede:

- trasmissione di **grandi quantità di dati** (es. streaming audio/video),
- **latenza ridotta** per comandi in tempo reale,
- **connessione senza fili** per mobilità del veicolo.

Sono state valutate diverse tecnologie:

Tecnologie escluse:

- **Ethernet**: escluso per la necessità di mobilità e l'impossibilità di utilizzare cavi fisici.
- **LoRa/LoRaWAN**: tecnologia a lungo raggio e bassissimo consumo, ma con un throughput estremamente limitato (tipicamente < 50 kbps), inadatta per lo streaming.
- **WiMAX**: rete wireless a lunga distanza, scarsamente supportata da dispositivi embedded come il Raspberry e ormai poco diffusa anche per colpa dei costi.
- **Reti cellulari (3G, 4G, 5G)**: nonostante l'ampia copertura e la buona banda, sono state escluse per i seguenti motivi:
 - necessità di SIM e contratto dati;
 - latenza potenzialmente elevata dovuta all'instradamento su più nodi (gateway NAT, router di rete pubblica);
 - costi energetici e di traffico maggiori;
 - complessità di configurazione.
 (bisognerebbe usare un dispositivo cellulare legato direttamente alla macchina)

Tecnologia scelta: Wi-Fi

È stata quindi selezionata la tecnologia Wi-Fi per i seguenti motivi:

- supportata nativamente dal **Raspberry Pi 3 Model B+**, anche su frequenze a 5 GHz;
- configurabile facilmente in una **rete LAN locale chiusa**, con gestione diretta tramite un access point;
- **ampio throughput** (fino a centinaia di Mbps);
- **bassa latenza** all'interno della rete locale.

Configurazione della rete Wi-Fi È stato utilizzato un router **TP-Link di fascia media**, configurato dual-band per trasmettere su entrambe le bande:

- **2.4 GHz**: maggiore portata, migliore **attraversamento di ostacoli** (muri, porte), ma maggiore congestione.
- **5 GHz**: minore portata, ma minore interferenza e maggiore velocità.

Il Raspberry Pi 3B+ supporta entrambe le bande. Durante i test, l'utilizzo della rete 5 GHz ha mostrato una significativa **riduzione del jitter** nello streaming video, migliorando fluidità e qualità visiva.

Sicurezza

Il Wi-Fi è stato cifrato tramite protocollo **WPA2-PSK**, con autenticazione tramite password simmetrica. Questo standard offre un buon compromesso tra **sicurezza e semplicità**, pur non essendo il più avanzato (come WPA3 che però ha problemi di compatibilità con il Raspberry).

Protocollo MAC e accesso al canale

Il Wi-Fi utilizza il protocollo **CSMA/CA** (Carrier Sense Multiple Access with Collision Avoidance), un sistema di accesso che include l'uso dei frame **RTS** (Request to Send) e **CTS** (Clear to Send) per **evitare collisioni**. Tuttavia, all'**aumentare del numero di dispositivi** connessi alla rete, l'**efficienza può degradare**. In questo progetto, la rete è composta solo da:

- il router,
- il Raspberry Pi (MCU),
- il computer client (che riceve i dati).

In questo contesto, la rete è sufficientemente leggera da garantire bassa latenza e alta affidabilità.

Portata e potenza del segnale La potenza massima ammessa per i dispositivi Wi-Fi in Europa, senza necessità di licenza radio, è tipicamente:

$$P_{max} = 100 \text{ mW} = 20 \text{ dBm}$$

La **distanza teorica d che il segnale può coprire** è correlata alla **potenza** tramite la formula semplificata:

$$d = \sqrt{\frac{P_t G_t G_r \lambda^2}{(4\pi)^2 P_r}}$$

- d — distanza tra trasmettitore e ricevitore (in metri)
- P_t — potenza trasmessa (in Watt)
- P_r — potenza ricevuta (in Watt)
- G_t, G_r — guadagni delle antenne trasmittente e ricevente (adimensionali)
- λ — lunghezza d'onda del segnale trasmesso, calcolabile come $\lambda = \frac{c}{f}$, con c velocità della luce e f frequenza del segnale
- 4π — costante geometrica dovuta alla propagazione sferica del segnale elettromagnetico

Il limite normativo serve a evitare interferenze con altri dispositivi. Nella banda 2.4 GHz, la portata può arrivare **fino a 500 m in spazi aperti**, mentre in ambienti chiusi si riduce notevolmente. Il 5 GHz ha **portata inferiore**, ma **maggior banda** utile, a discapito della penetrazione degli ostacoli.

2.5.2 Livello di Rete

A livello di rete, il progetto utilizza il protocollo **IPv4** per identificare in modo univoco i dispositivi all'interno della rete locale. **Non essendoci necessità di passare da router esterni** o accedere a Internet, l'uso dell'indirizzo IP consente una comunicazione più comoda ed efficiente rispetto al solo utilizzo degli indirizzi MAC, sfruttando lo stack TCP/IP già integrato nei sistemi operativi.

Grazie al protocollo **ARP** (Address Resolution Protocol), il sistema è in grado di associare un indirizzo IP a un indirizzo MAC, permettendo così la comunicazione a livello di data link.

Per scoprire automaticamente i dispositivi connessi alla rete locale, si possono utilizzare strumenti di scansione più avanzati come **nmap**, molto utili in fase di debug o configurazione.

Ad esempio, per eseguire una scansione dell'intera rete locale con indirizzi IP del tipo 192.168.1.X, è possibile utilizzare il seguente comando:

```
nmap -sn 192.168.1.0/24
```

Questo comando esegue una scansione *ping scan* (**-sn**) per rilevare quali host sono attivi nella sottorete.

2.5.3 Livello di Trasporto

A livello di trasporto, la scelta ricade principalmente tra i due protocolli fondamentali: **TCP** (Transmission Control Protocol) e **UDP** (User Datagram Protocol). La decisione viene presa in funzione del tipo di dati da trasmettere, valutando il compromesso tra affidabilità e latenza. In generale:

- **TCP** garantisce consegna **ordinata, senza duplicati** e con controllo degli errori, ma introduce **latenza maggiore**;
- **UDP** è un protocollo connectionless, leggero e a **bassa latenza**, ma non garantisce l'affidabilità dei pacchetti.

Nel progetto sono stati adottati **entrambi**, in base allo scenario applicativo.

Comandi di movimento — UDP

Per l'invio dei comandi di movimento al veicolo è stato scelto l'uso diretto del protocollo **UDP**, per minimizzare la latenza. È stato quindi sviluppato un **protocollo personalizzato semplificato**, senza meccanismi di conferma o riordinamento dei pacchetti.

Il Raspberry Pi apre un server UDP in ascolto:

```
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
try:
    sock.bind((IP, PORTA))
except OSError:
    print("Porta già in uso")
    exit(1)
```

Nel caso in cui il processo venga terminato in modo anomalo, la porta potrebbe rimanere occupata; per questo è stata inserita una gestione dell'eccezione.

Il client (PC mittente) può inviare comandi tramite:

```
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.sendto(b'avanti", (IP, PORTA))
```

Il veicolo **interpreta il messaggio ricevuto**, elabora lo **stato attuale del veicolo**, per ottenere movimenti fluidi e coerenti e **richiama la funzione appropriata** (es. `motori.avanti()`) per attivare i motori.

Vedremo meglio il resto del codice nelle sezioni successive.

Dati dei sensori — TCP via MQTT

Per il trasferimento dei dati dei sensori è stato adottato il protocollo applicativo **MQTT** (Message Queuing Telemetry Transport) [40], che utilizza **TCP** come protocollo di trasporto. Pur essendo meno reattivo rispetto a UDP, MQTT rappresenta un buon compromesso tra **affidabilità e overhead**, grazie a un'intestazione estremamente leggera (solo 2 byte nel formato più semplice) e all'architettura di tipo **publish/subscribe**. Inoltre, la connessione persistente con il broker consente di evitare l'instaurazione di un nuovo handshake TCP per ogni messaggio, **riducendo così la latenza complessiva** nella trasmissione dei dati.

Per esigenze di sicurezza, è possibile usare **MQTTS**, ovvero MQTT su TLS/SSL.

Streaming audio/video — UDP con GStreamer

Per la trasmissione in tempo reale di audio e video dal veicolo al computer controllore, è stato impiegato il framework **GStreamer** [41]. Esso permette la creazione di *pipeline* multimediali configurabili tra sorgente e destinatario.

È possibile **configurare il protocollo di trasporto** scegliendo tra i possibili parametri:

- `udpsink` / `udpsrc` per una trasmissione UDP a bassa latenza;
- `tcpclientsrc` / `tcpserversink` per trasmissioni TCP affidabili, qualità maggiore ma con maggiore latenza.

Nel progetto si è optato per la versione **UDP**, in quanto la trasmissione deve essere il più possibile **in tempo reale**, rendendo la latenza un parametro critico per la guida da remoto, più della qualità video.

Streaming Audio Video naive

Durante la fase di test è stata valutata anche la possibilità di realizzare lo streaming audio/video utilizzando direttamente socket **UDP** e librerie Python come **OpenCV** per l'acquisizione e la trasmissione dei frame video.

Sebbene questa soluzione risulti più personalizzabile e leggera in termini di dipendenze esterne, sono emerse diverse criticità, come l'elevato rate di trasmissione di pacchetti **senza un adeguata compressione**, perdite frequenti di frame e **incompatibilità tra diverse librerie**.

Per questi motivi, si è deciso di adottare una soluzione più robusta e ad alte prestazioni come **GStreamer**, che fornisce strumenti ottimizzati per lo streaming multimediale, con un'architettura modulare e protocolli già testati per la trasmissione dati in tempo reale.

2.5.4 Livello di Applicazione

In questa sezione vengono analizzate le principali soluzioni di comunicazione a livello applicativo, alcune delle quali sono già state accennate nel paragrafo precedente, ma qui verranno approfondite in modo più dettagliato.

Protocolli alternativi analizzati:

- **HTTP/REST**: il classico approccio *request-response*, in cui il client invia una richiesta e il server restituisce una risposta. Sebbene sia diffuso per le pagine web, HTTP è sincrono, ha **overhead elevato** (header verbosi) e non è adatto a sistemi real-time o ad aggiornamenti continui come quelli richiesti in questo progetto.
- **CoAP (Constrained Application Protocol)**: pensato per dispositivi embedded, utilizza UDP ed è molto leggero, ma **manca di alcune funzionalità avanzate** di MQTT come la persistenza, il QoS, o la gestione dei topic.
- **WebSocket**: protocollo full-duplex basato su TCP, ottimo per comunicazioni bidirezionali continue, ma più **complesso da implementare** in ambienti embedded rispetto a MQTT.
- **XMPP**: nato per la messaggistica istantanea, è basato su XML ed è troppo pesante per applicazioni IoT.
- **DSS (Data Stream System)**: sistemi orientati allo streaming di dati, generalmente usati in contesti ad alta intensità (e.g., Big Data), **non adatti a microcontrollori**.
- **AMQP (Advanced Message Queuing Protocol)**: protocollo orientato a messaggi, simile a MQTT ma più complesso e pensato per applicazioni enterprise con **esigenze diverse dall'IoT**.

MQTT

MQTT (Message Queuing Telemetry Transport) è un protocollo di messaggistica pubblicato nel 1999 da IBM, progettato originariamente per il monitoraggio remoto di impianti petroliferi. È oggi uno dei protocolli più utilizzati in ambito IoT grazie alla sua architettura **publish/subscribe**, alla leggerezza del payload e alla possibilità di funzionare anche su reti instabili.

Architettura Pub/Sub: MQTT si basa su tre elementi principali:

- **Broker**: un server centrale (es. Mosquitto) che riceve, smista e inoltra i messaggi tra i client.
- **Publisher**: un client che **pubblica un messaggio** su un certo *topic*.
- **Subscriber**: un client che **si iscrive a uno o più topic** per ricevere i messaggi pubblicati.

Esempio di codice in Python (usando paho-mqtt):

```
# Publisher
client.connect("ip_broker", porta_broker)
client.publish("sensori/temperatura", "28.5")

# Subscriber

# Funzione chiamata ogni volta che si riceve un messaggio da un canale a cui si è iscritti
# Qui è possibile smistare i vari messaggi in base ai loro topic
def on_message(client, userdata, msg):
    print(msg.topic, msg.payload.decode())

client.connect("ip_broker", porta_broker)
client.subscribe("sensori/#")
client.on_message = on_message
client.loop_forever()
```

Caratteristiche dei topic: I topic sono stringhe strutturate gerarchicamente (es. sensori/ultrasuoni/davanti), che possono essere gestite anche con caratteri jolly:

- + sostituisce un solo livello: `sensori/+/davanti`
- # sostituisce tutti i livelli successivi: `sensori/#`

Retain e QoS: MQTT consente anche l'invio di messaggi con flag `retain=true` per mantenerli nel broker e consegnarli al prossimo client che si iscrive.

Inoltre supporta diversi livelli di QoS (Quality of Service):

- 0: at most once (senza conferma)
- 1: at least once (con possibile duplicazione)
- 2: exactly once (con handshake)

Mosquitto: il broker scelto

Per questo progetto è stato scelto **Mosquitto**, un broker MQTT open source, leggero, stabile e largamente documentato. Una volta avviato, il processo rimane in background come demone e gestisce le connessioni dei vari client, i topic, la distribuzione dei messaggi e le eventuali code.

La configurazione standard è adatta per la rete locale e richiede minime risorse computazionali. È possibile configurarlo anche con TLS per aumentare la sicurezza del canale MQTT.

GStreamer

Per l'invio dei dati audio e video, si è scelto di adottare **GStreamer**, un framework multimediale open source estremamente modulare e potente, **scritto in linguaggio C**. GStreamer consente la costruzione di pipeline personalizzate per l'acquisizione, l'elaborazione e la **trasmissione in tempo reale** di flussi audio/video. È largamente utilizzato in applicazioni come *OBS Studio*, *Firefox*, *VLC* e in molti progetti nel settore IoT.

Il framework può essere utilizzato tramite linea di comando attraverso l'eseguibile ‘gst-launch-1.0‘, ma è anche accessibile da diversi linguaggi di programmazione (come Python) tramite **specifici binding**. Una pipeline GStreamer è strutturata come una **catena di elementi collegati** con l'operatore “!”, il quale definisce il flusso dei dati da un modulo al successivo.

L'architettura va intesa come un sistema di elaborazione unidirezionale, dove ogni blocco esegue una specifica trasformazione:

[*Camera*] → [*Convertitore*] → [*EncoderH.264*] → [*RTPPacketizer*] → [*UDPSender*]

[*Microfono*] → [*Convertitore*] → [*Resampler*] → [*EncoderOPUS*] → [*RTPPacketizer*] → [*UDPSender*]

Pipeline del mittente (Sender Raspberry)

Segue un esempio di comando per l'invio simultaneo di flusso video e audio:

```
gst-launch-1.0 -v

# === Video ===
v4l2src device=/dev/video0 !
videoconvert !
x264enc tune=zerolatency bitrate=500 speed-preset=ultrafast !
rtpx264pay config-interval=1 pt=96 !
udpsink host=IP_DESTINATARIO port=PORTA_VIDEO

# === Audio ===
alsasrc device=hw:2,0 !
audioconvert ! audioresample !
opusenc !
rtpopuspay pt=97 !
udpsink host=IP_DESTINATARIO port=PORTA_AUDIO
```

- **v4l2src**: sorgente video basata su Video4Linux2; `/dev/video0` rappresenta il primo dispositivo video USB riconosciuto da Linux come un file.
- **videoconvert**: converte il formato del video per garantirne la compatibilità con il codec.
- **x264enc**: codifica il flusso in formato H.264; l'opzione `tune=zerolatency` riduce la latenza, `bitrate` e `speed-preset` controllano qualità e velocità di compressione.
- **rtpH264pay**: incapsula i dati in pacchetti RTP. L'opzione `pt=96` definisce il payload type.
- **alsasrc**: acquisisce l'audio dal dispositivo ALSA; `hw:2,0` è l'indirizzo hardware.
- **opusenc** e **rtpopuspay**: codificano e incapsulano l'audio nel formato OPUS via RTP.
- **udpsink**: invia il flusso tramite protocollo UDP verso l'indirizzo e la porta specificata.

Pipeline del ricevente (Receiver pc)

Il destinatario invece attraverserà in maniera inversa la pipeline, scomponendo il pacchetto.

```
# === Video ===
udpsrc port=PORTA_VIDEO caps="application/x-rtp,media=video,encoding-name=H264,payload=96" !
rtpH264depay !
avdec_h264 !
videoconvert !
fpsdisplaysink sync=false text-overlay=false video-sink=autovideosink

# === Audio ===
udpsrc port=PORTA_AUDIO caps="application/x-rtp,media=audio,encoding-name=OPUS,payload=97" !
rtpopusdepay !
opusdec !
audioconvert ! audioresample !
autoaudiosink sync=false
```

- **udpsrc**: sorgente UDP, in ascolto sulla porta indicata.
- **caps**: specifica le *capabilities* del flusso RTP in ingresso.
- **rtpH264depay** / **rtpopusdepay**: rimuovono l'incapsulamento RTP.
- **avdec_h264** / **opusdec**: decodificano il flusso video/audio.
- **autovideosink** / **autoaudiosink**: gestiscono l'output automatico su schermo o altoparlanti.
- **synk=false** disabilita la sincronizzazione temporale per ridurre la latenza.

È possibile integrare ulteriori moduli per la registrazione locale del flusso.

Modificare questi parametri può portare a un miglioramento delle prestazioni.

2.5.5 Mantenimento e rappresentazione del dato

Come ultimo livello dello stack, si introduce una componente superiore che, pur non appartenendo formalmente al modello TCP/IP, rappresenta la **fase conclusiva del ciclo di vita del dato**: la sua persistenza e successiva analisi.

Scelta del Database

Nel contesto del progetto, è stato necessario valutare diverse soluzioni per l'archiviazione dei dati, in particolare tra:

- **Database relazionali** (es. MySQL, PostgreSQL), caratterizzati da strutture tabellari, linguaggio SQL e maggiore diffusione;
- **Database NoSQL**, in particolare orientati alle *time-series*, come InfluxDB, più complessi da configurare ma estremamente performanti per scenari IoT.

La scelta finale è ricaduta su **InfluxDB**, un database NoSQL open-source ottimizzato per la gestione di serie temporali, ovvero dati indicizzati tramite un **timestamp** ad alta precisione (nanosecondi, microsecondi o millisecondi). Questo lo rende particolarmente adatto a contesti in cui i dati cambiano nel tempo in modo continuo, come sensoristica, logging di eventi, metriche di sistema e telemetria.

InfluxDB organizza i dati in **bucket**, ovvero contenitori logici che sostituiscono i tradizionali database. Ogni **Point** rappresenta un oggetto contenuto nel bucket, ed è rappresentato dalla stringa:

```
<measurement>,<tag_key>=<tag_value> <field_key>=<field_value> <timestamp>
```

Esempio:

```
UltrasoundSensor, direction=dx value=23.5 16789000000000000000
```

- **measurement** è stato usato come nome del sensore,
- **tag** (facoltativi) usati per indicare altre informazioni,
- **field** viene usato per specializzare più valori dallo stesso tipo di sensore,
- **timestamp** il numero di secondi (o sottomultipli) trascorsi dal 1 gennaio 1970 Epoch time.

L'interfaccia web integrata (di default alla porta 8086) consente inoltre la consultazione dei dati tramite dashboard grafiche, molto utili per il monitoraggio e il **debugging in tempo reale**.

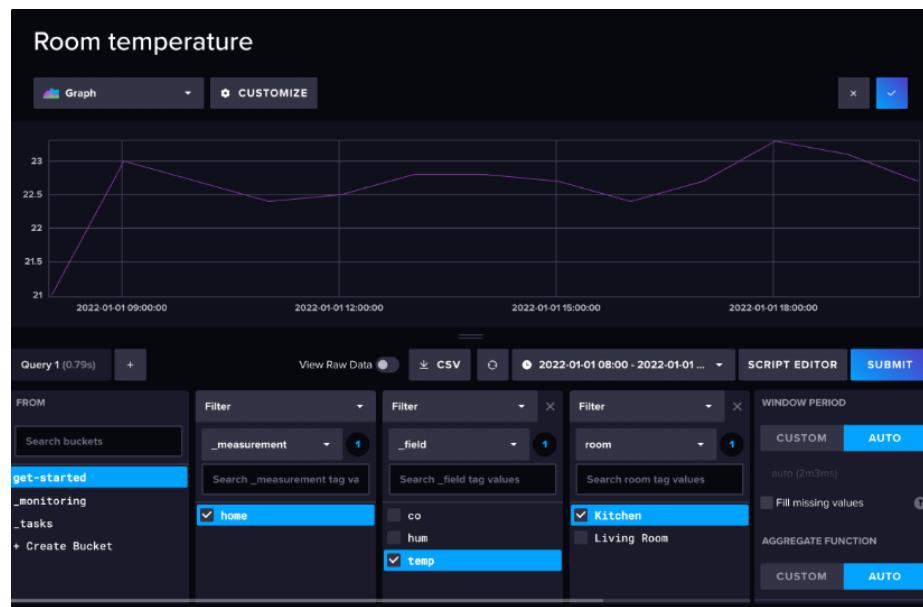


Figura 2.18: Dashboard di esempio in InfluxDB

Scrittura dei dati in Python

Una volta ricevuti i messaggi MQTT, l'invio verso InfluxDB avviene tramite la libreria ufficiale `influxdb-client-python`, utilizzando l'API di scrittura:

```
from influxdb_client import InfluxDBClient, Point

client = InfluxDBClient(url="http://localhost:8086", token=TOKEN)
write_api = client.write_api()

point = Point("ultrasuoni").field("distanza", 0.43).time(timestamp)
write_api.write(bucket="Sensori", record=point)
```

Query e lettura dei dati

L'interrogazione del database avviene tramite il linguaggio **Flux**, evoluzione di InfluxQL. Un esempio di query per **recuperare l'ultimo valore di un campo specifico**:

```
query = f'''
from(bucket: "{BUCKET}")
|> range(start: -1m)
|> filter(fn: (r) => r._measurement == "{measurement}")
|> filter(fn: (r) => r._field == "{field}")
|> last()
'''

tables = client.query_api().query(query)
```

```
for table in tables:
    for record in table.records:
        valore = record.get_value()
```

Le query ritornano una **serie di tabelle**, in base ai measurement o tag usati, ogni tabella è un **insieme di records** (FluxRecords), insieme di valori con le informazioni sul dato estratto.

```
tables = [
    table_1 = {
        records = [
            record_1 → _value = 22.3,
            record_2 → _value = 23.1,
            ...
        ]
    },
    table_2 = {
        records = [
            ...
        ]
    }
]
```

Visualizzazione dei dati

Per la visualizzazione dei dati, la soluzione ideale sarebbe stata l'integrazione con software già consolidati come **Grafana**, che consente di collegarsi direttamente a *InfluxDB* e visualizzare le informazioni tramite dashboard altamente configurabili, anche all'interno di un server web dedicato.

Tuttavia, per motivi legati alle tempistiche progettuali, si è optato per una soluzione più semplice e immediata: è stata realizzata un'interfaccia grafica custom utilizzando la libreria **Pygame**, con lo scopo di visualizzare in tempo reale alcuni sensori ritenuti critici durante le fasi di test.

Nello specifico:

- è stato implementato un grafico a scorrimento per rappresentare l'andamento nel tempo della **tensione di alimentazione**;
- un'icona a forma di **batteria**, accompagnata da una percentuale, indica visivamente lo stato di carica residua per prevenire spegnimenti imprevisti;
- sono stati integrati due pannelli di visualizzazione semplificati: uno per i **sensori a ultrasuoni**, utile per la percezione degli ostacoli, e uno per la **rappresentazione dell'accelerazione rilevata dall'IMU**.

Questa soluzione, pur essendo limitata rispetto a un sistema professionale come Grafana, ha permesso di garantire un monitoraggio continuo ed efficace durante la fase di sperimentazione sul campo.



Figura 2.19: Visualizzazione dati

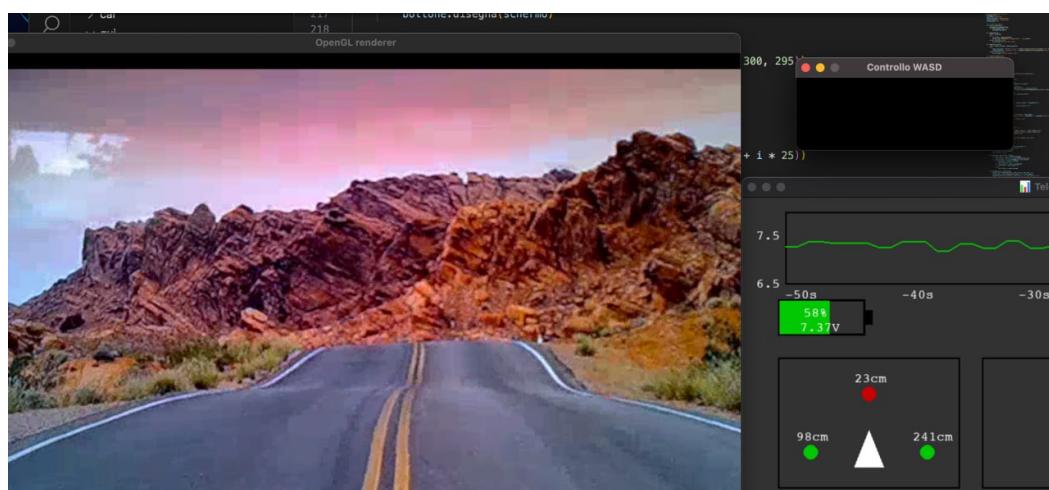


Figura 2.20: Trasmissione video e dati visualizzati

2.6 Codice e Architettura di sistema

In questa ultima sezione verranno analizzate alcune scelte implementative significative, con particolare attenzione a frammenti di codice rappresentativi delle funzionalità principali del sistema.

2.6.1 Architettura di sistema

Il codice è stato giustamente suddiviso tra il raspberry e il pc di controllo, da cui l'utente potrà interfacciarsi direttamente con il veicolo.

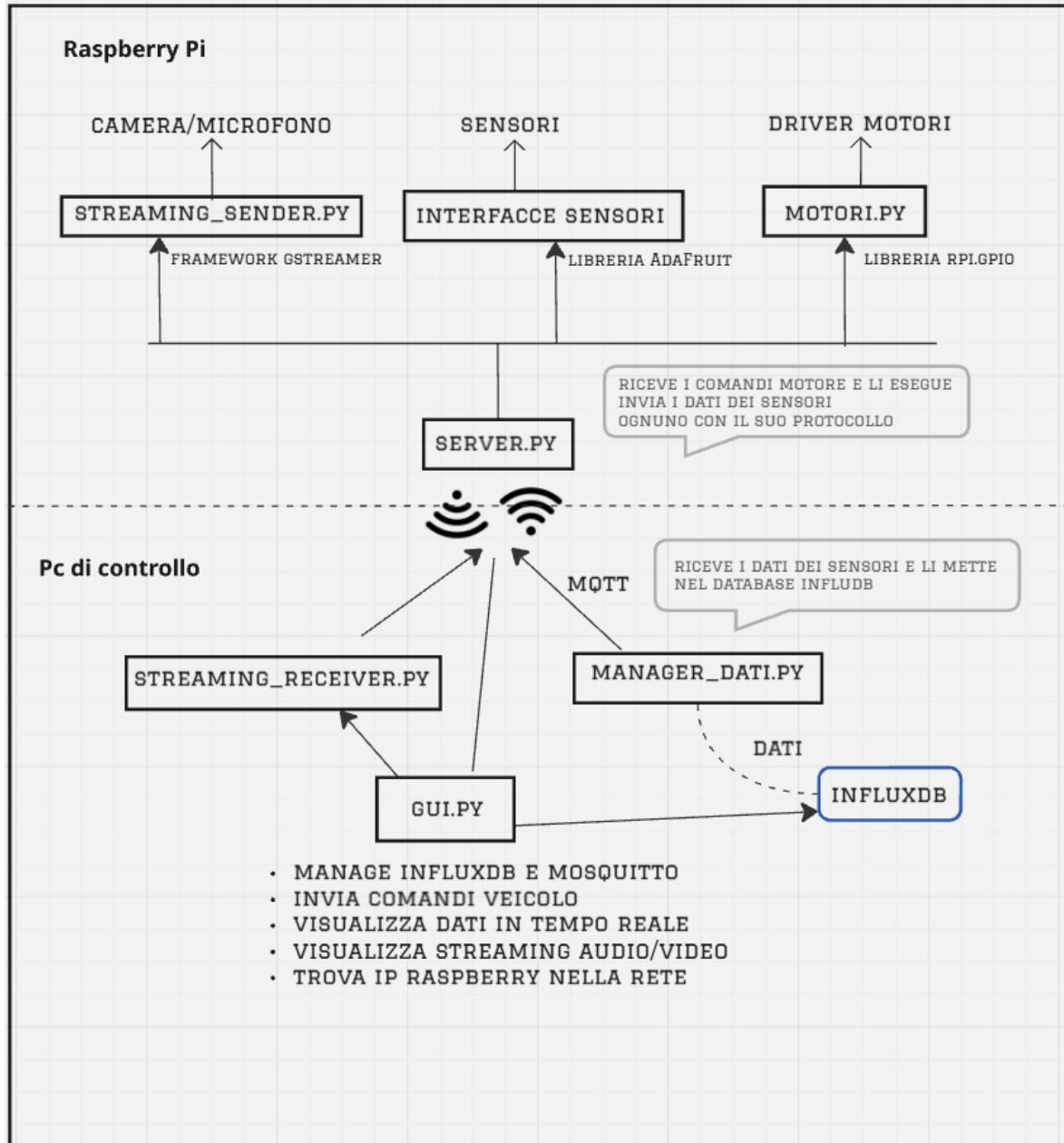


Figura 2.21: Diagramma delle classi

2.6.2 Interfaccia Grafica (GUI)

Per semplificare e centralizzare il controllo delle principali funzionalità del sistema, come l'avvio dei servizi, l'attivazione dei sensori e la gestione dei servizi di backend, è stata sviluppata un'interfaccia grafica basata su **Pygame**, semplice ma efficace per l'ambiente operativo previsto.

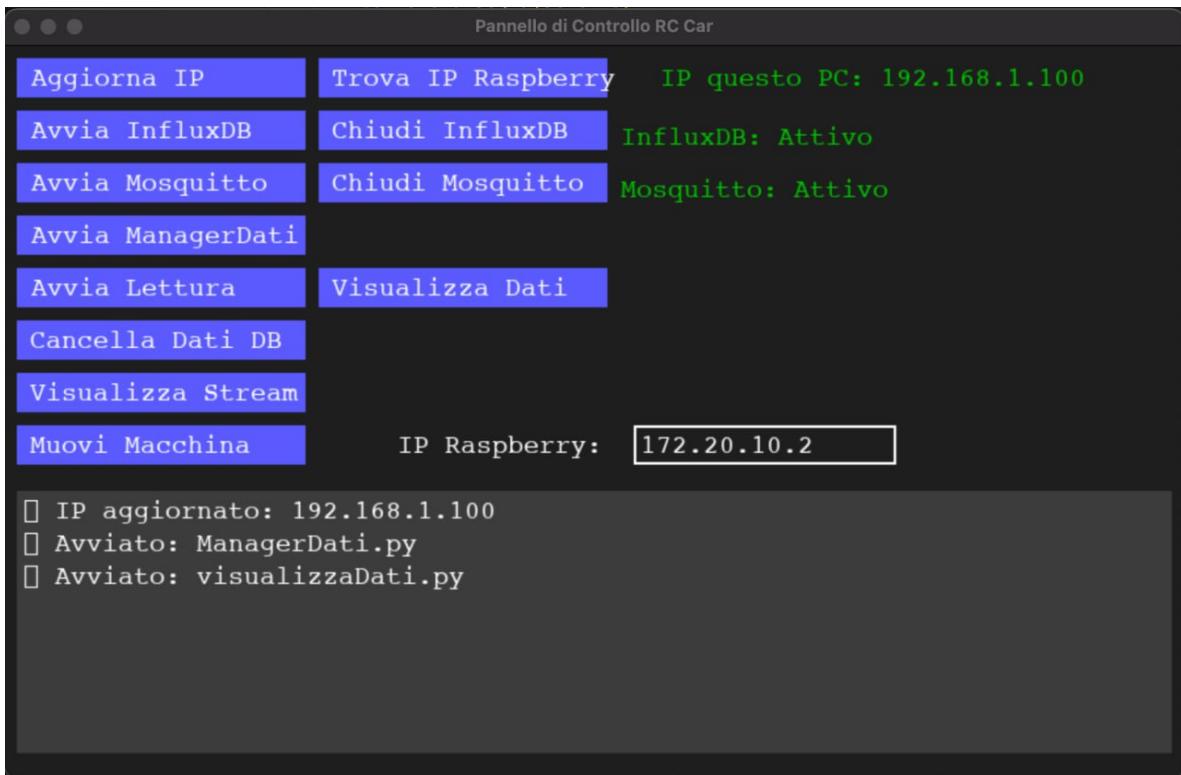


Figura 2.22: Interfaccia grafica principale per il controllo e il monitoraggio del sistema

Il codice associato alle varie funzionalità dei pulsanti è distribuito su più file, organizzati in moduli dedicati. Per ciascun pulsante dell'interfaccia, sono state costruite funzioni ausiliarie che, attraverso l'**esecuzione di processi esterni** (tramite `subprocess`), attivano script specifici o componenti di sistema separati.

Ad esempio, per l'attivazione e la verifica della disponibilità dei servizi **InfluxDB** e **Mosquitto**, è stato realizzato uno script che utilizza comandi `bash` per accertarsi della presenza di processi attivi sulle rispettive porte predefinite (8086 per InfluxDB e 1883 per Mosquitto).

Codice estrazione processi attivi e verifica se influxDB è in stato di running

```

# esegue il comando da terminale "lsof -i :8086" (processi con socket aperta)
result = subprocess.run(['lsof', '-i', ':8086'], stdout=subprocess.PIPE,
stderr=subprocess.PIPE, text=True)

output = result.stdout

for riga in output.splitlines():
    # se c'è almeno una socket in LISTEN vuol dire influxDB è connesso
    if "LISTEN" in riga and ("8086" in riga):
        return True
return False

# Esempio di output:
# COMMAND PID TYPE DEVICE NODEF NAME
# influxd 1159 IPv4 0xad45aefbebf719 TCP localhost:58625->localhost:8086 (ESTABLISHED)
# influxd 1159 IPv6 0xad45aefbecf799 TCP localhost:8086->localhost:58625 (ESTABLISHED)
# influxd 1159 IPv6 0xad45aefbecf799 TCP *:8086 (LISTEN)

## 8086 è in established ,
## ma anche in listen per accettare una nuova connessione

```

Sono inoltre stati sviluppati e adattati alla GUI:

- uno script per determinare l'indirizzo IP locale del computer, mediante la creazione e chiusura di una socket;
- uno script per il rilevamento della presenza del **Raspberry Pi** in rete locale. Questo si basa sulla risoluzione del nome host `marco.local` e sulla verifica della raggiungibilità via ping. In caso di risposta, lo script restituisce l'IP rilevato.

2.6.3 ManagerDati

Uno dei primi moduli da avviare all'interno del sistema è il file `ManagerDati.py`, eseguito in un terminale dedicato con output visibile, così da monitorarne lo stato di funzionamento in tempo reale.

Lo scopo principale di questo modulo è quello di fungere da intermediario tra il **Raspberry Pi** e il **database InfluxDB**. In particolare, il modulo:

- riceve i dati trasmessi dal Raspberry Pi tramite protocollo **MQTT**;
- verifica la corretta ricezione dei messaggi;
- effettua la conversione in formato `Point` e li invia al database.

La funzione principale di questo modulo è **InviaDatoDB**. Essa accetta come parametri (`measurement`), (`field`) e il valore, e genera un oggetto `Point` collegando al TimeStamp attuale che viene poi scritto nel bucket InfluxDB tramite la `write_api`.

Funzione invio misurazioni al Database

```
# Funzione per inviare dati al Database
def inviaDatoDB(measurement, field, value, timestamp=None):

    # Calcola il timestamp in nanosecondi
    if timestamp is None:
        timestamp = int(time.time_ns())

    # in base ai dati forniti crea il Point
    point = Point(measurement).field(field, value).time(timestamp)

    # e lo scrive nel bucket (il setup del database è stato fatto prima)
    try:
        write_api.write(bucket=INFLUXDB_BUCKET, org=INFLUXDB_ORG, record=point)

        # scrive <measurement> <field> <value> <timestamp>
        print("caricato: " + point.to_line_protocol())

    except Exception as e:
        print(f"Errore invio al DB: {e}")
```

La seconda funzione rilevante è il `callback` associato all'evento `on_message` di MQTT. Questa funzione viene **invocata automaticamente** ogni volta che il modulo riceve un **messaggio su un topic** del tipo `sensori/#`. Il messaggio viene quindi smistato in base al topic specifico, interpretato correttamente, ed eventualmente inviato al database tramite la funzione sopra descritta.

```
def on_message(client, userdata, msg):  
  
    # ricevo un messaggio sul canale "sensori/#"  
    try:  
        topic = msg.topic  
        payload = msg.payload.decode()  
  
        # Viene smistato il messaggio in base al canale  
        # e vengono generati i dati corretti da inviare  
        # al database  
  
        if topic == "sensori/voltaggio":  
            valore = float(payload) # si presume un numero  
            inviaDatoDB("volt_batteria_principale",  
                        "volt", valore)  
  
        elif topic == "sensori/ultrasuoni/davanti":  
            valore = float(payload)  
            inviaDatoDB("distanza_ultrasuoni",  
                        "distanza_davanti", valore)  
  
        elif topic == "sensori/ultrasuoni/dx":  
            valore = float(payload)  
            inviaDatoDB("distanza_ultrasuoni",  
                        "distanza_dx", valore)  
  
        elif topic == "sensori/ultrasuoni/sx":  
            valore = float(payload)  
            inviaDatoDB("distanza_ultrasuoni",  
                        "distanza_sx", valore)  
  
        elif topic == "sensori/accelerometro/x":  
            valore = float(payload)  
            inviaDatoDB("acc", "acc_x", valore)  
  
    ...
```

2.6.4 Lettura e visualizzazione dei dati

Entrambi i moduli coinvolti in questa fase utilizzano la medesima logica per il recupero dei dati dal database InfluxDB. La differenza principale risiede nel tipo di output: il primo modulo stampa i valori direttamente nel terminale, mentre il secondo utilizza la libreria Pygame per fornire una visualizzazione dinamica e interattiva dei dati in tempo reale.

I dati vengono estratti mantenendo esattamente la struttura utilizzata in fase di scrittura nel database, ovvero facendo riferimento ai medesimi `measurement` e `field`. Attraverso la funzione di interrogazione precedentemente descritta nella tesi, viene recuperato l'ultimo valore disponibile per ciascun sensore e formattato per la visualizzazione.

```
# (measurement, nome field, valore field) come sono salvati i sensori nel database
SENSORI = [
    ("volt_batteria_principale", "volt", " Batteria principale"),
    ("distanza_ultrasuoni", "distanza_davanti", " Ultrasuoni davanti"),
    ("distanza_ultrasuoni", "distanza_dx", " Ultrasuoni destra"),
    ("distanza_ultrasuoni", "distanza_sx", " Ultrasuoni sinistra"),
    ("acc", "acc_x", " Accelerazione X"),
    ("acc", "acc_y", " Accelerazione Y"),
    ("acc", "acc_z", " Accelerazione Z"),
]

# dato un sensore e il valore di un field, estrae l'ultimo valore
def leggi_ultimo_valore(measurement, field):
    ...
    ...

# stampa l'ultimo valore di ogni sensore
def avvia lettura():
    for sensore in SENSORI:
        ...
        ...
```

Grazie all'efficienza intrinseca di InfluxDB, l'estrazione dei dati è praticamente istantanea, consentendo una gestione *real-time* delle informazioni sensoriali.

Un aspetto particolare riguarda la rappresentazione grafica della tensione della batteria: ogni punto acquisito viene salvato localmente, così da poter costruire uno storico visivo che mostri l'andamento nel tempo. Questo consente di monitorare facilmente eventuali cali di tensione e prevenire situazioni critiche durante le sessioni di test.

2.6.5 Movimento del veicolo – lato PC

L’ultimo modulo di rilievo è quello relativo all’invio dei comandi di movimento del veicolo, realizzato tramite un’applicazione sviluppata con la libreria `pygame`. Questo modulo apre una finestra grafica e gestisce l’interazione da tastiera, rilevando la pressione dei tasti direzionali per controllare la macchina da remoto attraverso l’invio di pacchetti UDP, secondo un protocollo personalizzato sviluppato appositamente.

```

comando = "fermo"

if w and d:
    comando = "avanti+destra"
elif w and a:
    comando = "avanti+sinistra"
elif s and a:
    comando = "indietro+sinistra"
elif s and d:
    comando = "indietro+destra"
elif w:
    comando = "avanti"
elif s:
    comando = "indietro"
elif a:
    comando = "sinistra"
elif d:
    comando = "destra"

# pulsante clacson
if k and not clacson_premuto:
    sock.sendto(b"clacson_on", (IP_RASPBERRY, PORT))
    clacson_premuto = True
elif not k and clacson_premuto:
    sock.sendto(b"clacson_off", (IP_RASPBERRY, PORT))
    clacson_premuto = False

```

Il protocollo supporta anche comandi speciali, come l’attivazione del buzzer acustico (comando a pressione del tasto `k`) e la gestione dello streaming audio-video (tasto `p`), che consente di attivare o disattivare la trasmissione in tempo reale.

È stato inoltre implementato un sistema per la gestione simultanea di più comandi: ad esempio, premendo contemporaneamente i tasti `w` e `d`, il veicolo esegue un’azione combinata di trazione e sterzata,

per fare ciò è servito implementare un sistema intelligente che fermasse i motori al rilascio, altrimenti il motore rimaneva bloccato nella direzione su cui era in moto.

```

if comando != ultimo_comando:
    comandi_correnti = set(comando.split("+"))
    comandi_prec = set(ultimo_comando.split("+"))

    # Se rilascio destra o sinistra, riporta lo sterzo a zero
    if "destra" in comandi_prec - comandi_correnti or \
        "sinistra" in comandi_prec - comandi_correnti:
        sock.sendto(b"portasterzoa0", (IP_RASPBERRY, PORT))

    # Se rilascio avanti o indietro, ferma la trazione
    if "avanti" in comandi_prec - comandi_correnti or \
        "indietro" in comandi_prec - comandi_correnti:
        sock.sendto(b"portatrazione0", (IP_RASPBERRY, PORT))

    # Invia i nuovi comandi attivi
    for c in comandi_correnti:
        sock.sendto(c.encode(), (IP_RASPBERRY, PORT))

    ultimo_comando = comando

for event in pygame.event.get():
    if event.type == pygame.QUIT:
        running = False
    elif event.type == pygame.KEYDOWN and event.key == pygame.K_q:
        running = False

clock.tick(30) # 30 FPS

# Invio comandi finali per fermare tutto
sock.sendto(b"fermo", (IP_RASPBERRY, PORT))
sock.sendto(b"stop", (IP_RASPBERRY, PORT))
pygame.quit()

```

Per quanto riguarda il lato *PC di controllo*, resterebbero due componenti secondari: il modulo `streaming_receiver.py`, di cui si è già discusso nelle sezioni precedenti, e un'ulteriore funzionalità legata a un pulsante dedicato, che sfrutta l'interfaccia di scrittura del client InfluxDB per eseguire la cancellazione completa dei dati presenti nel database. Questa funzione è particolarmente utile nelle fasi di *debug* e di ripristino dell'ambiente di test.

2.7 Analisi del Codice Lato Raspberry

All'interno dell'interfaccia grafica precedentemente analizzata, è presente un pulsante denominato **Cerca Raspberry**, il quale effettua una scansione della rete per individuare il dispositivo. Una volta completato il boot e connesso alla rete Wi-Fi, il Raspberry Pi verrà rilevato e l'indirizzo IP associato sarà mostrato nella pseudo-console dell'interfaccia.

L'unico intervento manuale richiesto sul Raspberry è l'avvio del terminale tramite SSH e l'esecuzione del comando da me configurato:

```
avvi(server <ip_pc>
```

Questo comando permette l'avvio automatizzato del server e l'inizializzazione della connessione con il PC. L'indirizzo IP viene propagato a tutti i moduli software (come MQTT), rendendo possibile l'invio dei dati senza ulteriori configurazioni. Da questo momento, l'intero sistema può essere gestito direttamente dal PC.

2.7.1 Movimento del Veicolo

Il processo principale in esecuzione sul Raspberry è quello dedicato al movimento del veicolo. Questo modulo apre una socket UDP in ascolto, riceve i pacchetti e decide, sulla base del contenuto, quale funzione motore eseguire.

Il controllo dei motori è effettuato agendo direttamente sulla corrente dei pin GPIO del Raspberry Pi, utilizzando la libreria `RPi.GPIO`, che consente anche la gestione dei pin PWM per il controllo della potenza.

La funzione principale per l'avvio dei motori è la seguente:

```
def avviaMotore():
    global pwmA, pwmB
    GPIO.setmode(GPIO.BCM)

    GPIO.setup([IN1, IN2, IN3, IN4, ENA, ENB], GPIO.OUT) # pin digitali

    pwmA = GPIO.PWM(ENA, 1000) # 1000 è la frequenza utilizzata
    pwmB = GPIO.PWM(ENB, 1000)

    pwmA.start(0)
    pwmB.start(0)
```

Per attivare i pin digitali, è sufficiente utilizzare comandi del tipo:

```
GPIO.output(IN1, GPIO.HIGH)
GPIO.output(IN2, GPIO.LOW)
```

Per variare la potenza del motore tramite modulazione PWM, si utilizza invece:

```
pwmA.ChangeDutyCycle(potenza)
```

Codice controllo motori

```
def avanti(potenza=50):
    GPIO.output(IN3, GPIO.HIGH)
    GPIO.output(IN4, GPIO.LOW)
    pwmB.ChangeDutyCycle(potenza)

def indietro(potenza=50):
    GPIO.output(IN3, GPIO.LOW)
    GPIO.output(IN4, GPIO.HIGH)
    pwmB.ChangeDutyCycle(potenza)

def destra(potenza=50):
    GPIO.output(IN1, GPIO.LOW)
    GPIO.output(IN2, GPIO.HIGH)
    pwmA.ChangeDutyCycle(potenza)

def sinistra(potenza=50):
    GPIO.output(IN1, GPIO.HIGH)
    GPIO.output(IN2, GPIO.LOW)
    pwmA.ChangeDutyCycle(potenza)

def spegniMotore():
    fermo()
    pwmA.stop()
    pwmB.stop()
    GPIO.cleanup()

# prova
def sterzo_a_zero():
    pwmA.ChangeDutyCycle(0)
```

```

GPIO.output(IN1, GPIO.LOW)
GPIO.output(IN2, GPIO.LOW)

def trazione_a_zero():
    pwmB.ChangeDutyCycle(0)
    GPIO.output(IN3, GPIO.LOW)
    GPIO.output(IN4, GPIO.LOW)

```

È importante ricordare che per il corretto funzionamento del ponte H, solo uno dei due canali (ad esempio IN1 oppure IN2) deve essere attivato per ciascun motore, al fine di evitare malfunzionamenti del driver.

2.7.2 Codice Sensori

Tutti i sensori sono implementati in moduli distinti e vengono richiamati dal server principale mediante la creazione di *thread* separati, al fine di garantire una gestione parallela e reattiva dei dati raccolti. Ogni thread ha il compito dileggere periodicamente il dato, stamparlo in console per il debugging e inviarlo tramite protocollo MQTT a Mosquitto.

```

client.connect(MQTT_BROKER, MQTT_PORT)
client.publish("sensori/ultrasuoni/dx", 102.2)

```

Accelerometro

Il sensore accelerometrico utilizzato è un ADXL345, gestito tramite il bus I²C e la libreria `adafruit_adxl34x`. L'inizializzazione del bus e del sensore avviene come segue:

```

i2c = busio.I2C(board.SCL, board.SDA)
self.accel = adafruit_adxl34x.ADXL345(i2c)
self.offset = (0.0, 0.0, 0.0)

```

La funzione `calibra()` effettua una calibrazione registrando 100 campioni e calcolando la media per ciascun asse. Successivamente, la funzione `leggi()` restituisce i valori compensati rimuovendo l'offset.

```

def calibra(self, campioni=100, attesa=0.01):
    """Calcola la media su più letture per compensare l'offset"""
    print("[Accelerometro] Avvio calibrazione...")

    somma_x, somma_y, somma_z = 0.0, 0.0, 0.0
    for _ in range(campioni):

```

```

        x, y, z = self.accel.acceleration # legge il valore del sensore
        somma_x += x
        somma_y += y
        somma_z += z
        time.sleep(attesa)

    self.offset = (
        somma_x / campioni,
        somma_y / campioni,
        somma_z / campioni - 9.81 # corregge la gravità sulla Z
    )

    ox, oy, oz = self.offset
    print(f"[Accelerometro] Offset calcolato: "
        f"X={ox:.2f} Y={oy:.2f} Z={oz:.2f}")

def leggi(self):
    try:
        x, y, z = self.accel.acceleration
        ox, oy, oz = self.offset
        return round(x - ox, 2), round(y - oy, 2), round(z - oz, 2)
    except Exception as e:
        print(f"[Accelerometro] ERRORE: {e}")
        return None, None, None

```

Sensore a Ultrasuoni

Il modulo per il sensore a ultrasuoni sfrutta la libreria `time` per misurare l'intervallo di tempo tra l'invio del segnale e la ricezione dell'eco. Il codice è attento a:

- gestire la durata del suono riflesso,
- evitare il blocco del sistema impostando un timeout massimo di ascolto.

```

def misura_distanza_cm(self):
    # Invia impulso di 10 microsecondi
    GPIO.output(self.trigger, GPIO.HIGH)
    time.sleep(0.00001)
    GPIO.output(self.trigger, GPIO.LOW)

    start_time = time.time()
    timeout = start_time + 0.04 # timeout max 40ms (distanza max)

```

```

# finché il sensore non percepisce niente, resta bloccato
# esce appena il sensore inizia a percepire → start_time
while GPIO.input(self.echo) == 0 and time.time() < timeout:
    start_time = time.time()

# finché percepisce il segnale resta bloccato,
# esce appena smette di percepirllo → stop_time
stop_time = time.time()
timeout = stop_time + 0.04
while GPIO.input(self.echo) == 1 and time.time() < timeout:
    stop_time = time.time()

# Durata dell'impulso
durata = stop_time - start_time
distanza = (durata * 34300) / 2 # in cm

return round(distanza, 1)

```

Sensore di Tensione

Il modulo per la lettura della tensione è relativamente semplice. L'interazione avviene con il convertitore analogico-digitale (ADC), che riduce il valore della tensione in ingresso di un fattore 5. La lettura è ottenuta tramite:

```

self.chan = AnalogIn(self.i2c, ads.p0)
tensione = self.chan.voltage * 5

```

Questo permette di riottenere il voltaggio originale misurato dal voltage sensor.

3

Navio2 e ArduRover

A titolo di confronto e per completezza progettuale, è utile analizzare come il progetto sarebbe potuto essere implementato utilizzando una piattaforma hardware e software già consolidata come **Navio2** abbinata al software open-source **ArduRover**, parte del progetto *ArduPilot*.

3.0.1 Cos'è Navio2

Navio2 è una **scheda d'espansione** (HAT) progettata per essere montata direttamente su Raspberry Pi. Fornisce una suite completa di sensori e interfacce pensate per la robotica autonoma, tra cui:

- IMU a 9 assi (accelerometro, giroscopio, magnetometro).
- Barometro per la quota.
- GPS ad alta precisione integrato.
- Convertitori ADC, PWM, UART, I2C, SPI già esposti.
- Supporto per riceventi RC (PPM/S.Bus).

Con Navio2, Raspberry Pi diventa un flight controller a tutti gli effetti, capace di eseguire **ArduPilot**, software di pilotaggio per veicoli autonomi.

3.0.2 ArduRover: il software

ArduRover è il modulo di ArduPilot dedicato ai **veicoli terrestri autonomi**. Offre un'interfaccia ricca e altamente configurabile che consente:

- Gestione autonoma del movimento e delle missioni.

- PID integrato per la stabilizzazione e il controllo.
- Comunicazione con *Ground Control Station* tramite MAVLink.
- Configurazione da remoto tramite interfacce grafiche (es. Mission Planner, QGroundControl).
- Log dettagliati e sistema di telemetria già integrato.

3.0.3 Considerazioni finali

In conclusione, l'utilizzo della piattaforma Navio2 avrebbe **certamente semplificato l'intero processo progettuale**. L'hardware integra già la maggior parte dei sensori necessari, e anche molti altri aggiuntivi come GPS, giroscopio e barometro, eliminando così la necessità di realizzare manualmente i cablaggi e l'interfacciamento via I²C o GPIO, con una conseguente riduzione del rischio di errori hardware. Gli unici sensori da aggiungere sarebbero stati quelli non previsti dal modulo, come ad esempio gli ultrasuoni, ma in ogni caso l'integrazione sarebbe risultata analoga.

Dal punto di vista software, la gestione dei segnali PWM sarebbe stata nettamente più efficiente grazie al supporto diretto offerto da ArduPilot, evitando la complessità della programmazione manuale di basso livello. Inoltre, l'intero scambio di dati telemetrici e i comandi di movimento del veicolo sarebbero stati gestiti tramite il protocollo **MAVLink**, molto più robusto, efficiente e standardizzato rispetto alla soluzione personalizzata basata su pacchetti UDP e MQTT.

Per quanto riguarda lo streaming audio/video, si sarebbe potuto comunque adottare GStreamer in parallelo, dato che non è una funzionalità nativamente prevista da ArduPilot.

Infine, un grande vantaggio sarebbe stato rappresentato dalla disponibilità di strumenti già pronti all'uso, come interfacce grafiche professionali (Mission Planner, QGroundControl), la registrazione automatica dei log, la configurazione remota e il monitoraggio avanzato, riducendo drasticamente il rischio di bug o errori.

Pertanto, sebbene il mio progetto abbia raggiunto gli obiettivi prefissati e rappresenti un ottimo esercizio didattico e sperimentale, un'implementazione basata su Navio2 e ArduRover avrebbe condotto agli stessi risultati con maggiore affidabilità, qualità e velocità di sviluppo.

4

Conclusione

Il progetto descritto in questa tesi rappresenta un primo passo concreto verso la realizzazione di un sistema mobile di sorveglianza urbana, basato su tecnologie open-source e facilmente replicabili. L'intero lavoro, dalla progettazione hardware all'implementazione software, ha richiesto un'attenta analisi dei protocolli di comunicazione, dell'interfacciamento tra sensori e microcontrollore, e della visualizzazione dei dati in tempo reale. Il risultato è un veicolo controllabile da remoto, capace di raccogliere e trasmettere dati sensoriali e multimediali, adattabile a diversi contesti operativi.

Tra gli spunti per lo sviluppo futuro, ritengo particolarmente interessante l'integrazione di un sistema di rilevamento automatico di comportamenti anomali o potenzialmente illeciti, basato su algoritmi di machine learning, in grado di elaborare i dati raccolti senza la necessità di riconoscere o identificare le persone coinvolte, preservando così la privacy. Questo amplierebbe significativamente l'utilità del veicolo nel contesto della sicurezza urbana.

Sotto il profilo tecnico, si potrebbero prevedere miglioramenti nella stabilità e nella portata della connessione Wi-Fi, passando a un modulo radio di livello superiore per operare anche a distanze maggiori. Il codice di movimento potrebbe essere ulteriormente ottimizzato per ottenere manovre più fluide e precise, integrando eventualmente anche un sistema di guida autonoma.

Dal lato software, il modulo di visualizzazione dati sviluppato in questa tesi costituisce una base solida per qualsiasi futura applicazione di analisi avanzata o controllo remoto. Miglioramenti della sicurezza del sistema, in particolare nella gestione delle connessioni SSH e della trasmissione dati, costituirebbero un naturale proseguimento del lavoro.

In definitiva, il sistema realizzato offre un'ottima piattaforma sperimentale, personalizzabile e scalabile, in grado di evolversi verso soluzioni sempre più sofisticate nell'ambito del monitoraggio intelligente e distribuito.

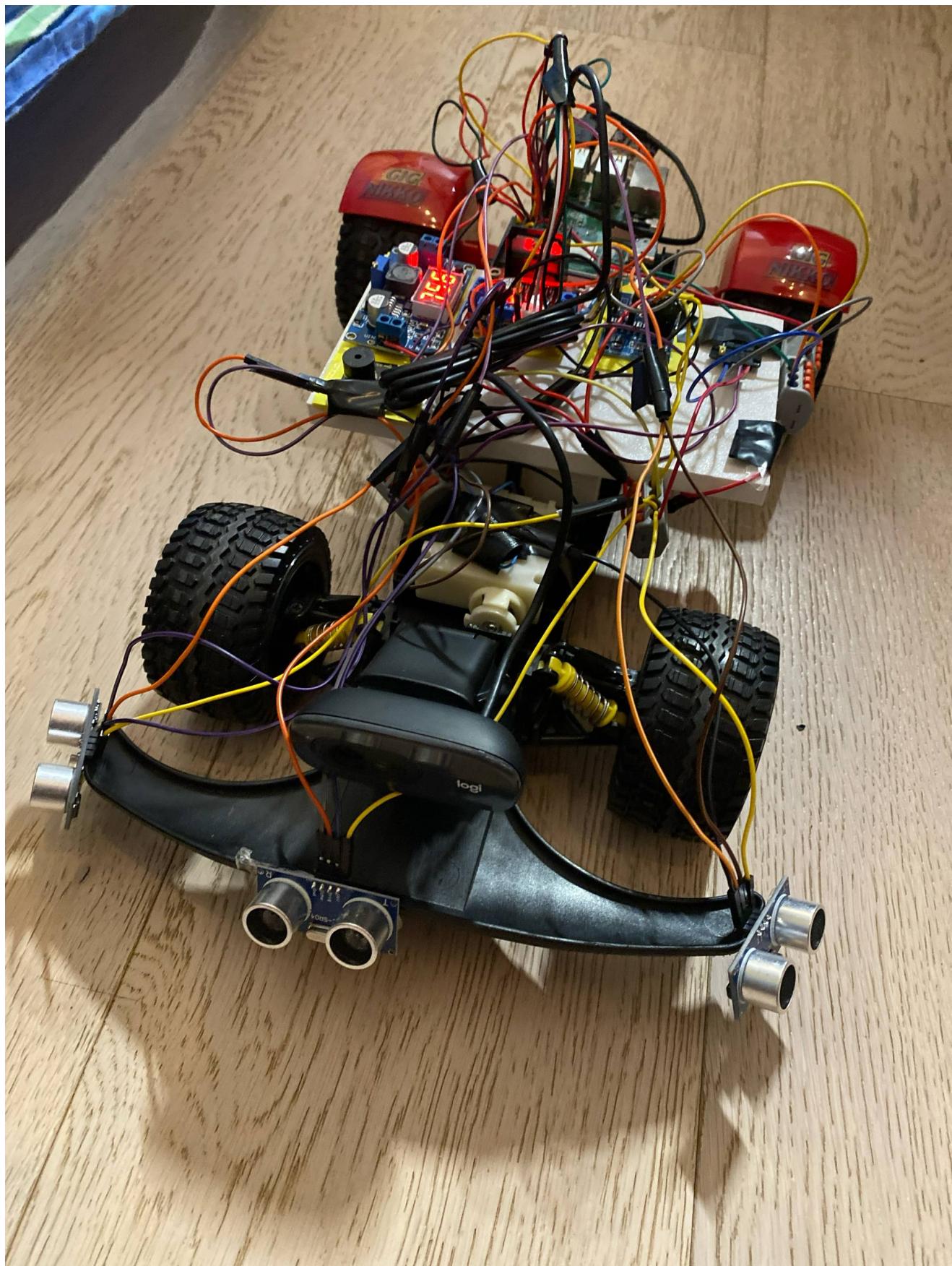


Figura 4.1: Il veicolo completato

Bibliografia

- [1] Fraga-Lamas, P., Fernández-Caramés, T. M., Suárez-Albela, M., Castedo, L., González-López, M. *A Review on Internet of Things for Defense and Public Safety.* arXiv preprint arXiv:2402.03599, 2024.
- [2] Kashif, M., Latif, N. *The Emergence of the Internet of Things in Military Defense: A Comprehensive Review.* International Journal of Innovations in Science & Technology, Vol. 6, 2024.
- [3] Fraga-Lamas, P., Fernández-Caramés, T. M. *Tactical Edge IoT in Defense and National Security.* MDPI Sensors, 24(6), 2024.
- [4] Chiu, C. C., Alvi, S. A., Zhou, X., Durrani, S., Wilson, N., Yebra, M. *A Survey on IoT Ground Sensing Systems for Early Wildfire Detection: Technologies, Challenges and Opportunities.* arXiv preprint arXiv:2312.10919, 2023.
- [5] Wikipedia contributors. *Battle of Tsushima – Wireless telegraphy.* https://en.wikipedia.org/wiki/Battle_of_Tsushima#Wireless_telex
- [6] Wikipedia contributors. *Multiple Integrated Laser Engagement System.* https://en.wikipedia.org/wiki/Multiple_integrated_laser_engagement_system
- [7] Wikipedia contributors. *Closed-circuit television.* https://en.wikipedia.org/wiki/Closed-circuit_television
- [8] Yanczer, P. F. *The Mechanics of Television.* Early Television Foundation, 2003. https://www.earlytelevision.org/pdf/peter_f_yanczer-the_mechanics_of_television.pdf
- [9] Liang, Z. and Das, V., Hu, Y., Zhan, N., Huang, L. *China's Social Credit System as a State Surveillance Infrastructure.* 2018. Disponibile su ResearchGate.
- [10] Kahla, L.Z., Gayflor, N., Mishra, A. *Leveraging Artificial Intelligence for Crime Detection and Prevention.* International Journal of Scientific Research in Engineering and Management, 2024.
- [11] He, M., Yue, X., Zheng, Y., Chen, J., Wu, S., Heng, Z., Zhou, X. Cai, Y. (2023). *State of the art and future trends in obstacle-surmounting unmanned ground vehicle configuration and dynamics.* Robotica, 41(9), 2625–2647. doi:10.1017/S0263574723000577
- [12] Albus, J.S., Huang, H.-M., Messina, E.R., Murphy, K., Juberts, M., Lacaze, A., Balakirsky, S.B., Shneier, M.O., Hong, T.H., Scott, H.A., Proctor, F.M., Shackelford, W.P., Michaloski,

- J.L.,Wavering, A.J.,Kramer, T.,Dagalakis, N.,Rippey, W.G.,Stouffer, K.A.,Legowik, S. (2002). *4D/RCS Version 2.0: A Reference Model Architecture for Unmanned Vehicle Systems*. NIST Interagency/Internal Report (NISTIR)6910, National Institute of Standards and Technology, Gaithersburg, MD. <https://doi.org/10.6028/NIST.IR.6910> :contentReference[oaicite:1]index=1
- [13] Nikko Garage, *Storia della Nikko*, disponibile su <http://nikkogarage.altervista.org/storia-nikko.html>, ultimo accesso 11 luglio 2025.
- [14] Elettronmodel, *Catalogo GIG Nikko 1988*, 1988. Disponibile su: <https://elettronmodel.wordpress.com/wp-content/uploads/2008/02/catalogo-gig-nikko-1988-by-elettronmodel.pdf>, ultimo accesso 11 luglio 2025.
- [15] Mabuchi Motor Co., Ltd., *RC-280SA-2865 Specifications*, 2024. Disponibile online: <https://product.mabuchi-motor.com/result.html?t=1487148886>, ultimo accesso: 11 luglio 2025.
- [16] Mabuchi Motor Co., Ltd., *FA-130RA DC Motor Datasheet*, Pololu, Disponibile online: https://www.pololu.com/file/0J11/fa_130ra.pdf, accesso 11 luglio 2025.
- [17] Wikipedia contributors. *Convertitore Flyback*. Wikipedia, l'enciclopedia libera. Disponibile su: https://it.wikipedia.org/wiki/Convertitore_Flyback. Ultimo accesso: 13 luglio 2025.
- [18] Wikipedia contributors. *H-bridge*. Wikipedia, l'enciclopedia libera. Disponibile su: <https://it.wikipedia.org/wiki/H-bridge>. Ultimo accesso: 13 luglio 2025.
- [19] Wikipedia contributors. *Modulazione di larghezza d'impulso*. Wikipedia, l'enciclopedia libera. Disponibile su: https://it.wikipedia.org/wiki/Modulazione_di_larghezza_d%27impulso. Ultimo accesso: 13 luglio 2025.
- [20] Wikipedia contributors. *Elettronica digitale*. Wikipedia, l'enciclopedia libera. Disponibile su: https://it.wikipedia.org/wiki/Elettronica_digitale. Ultimo accesso: 13 luglio 2025.
- [21] Electropeak, *L298N Motor Driver Module Manual*, disponibile su: <https://www.habit.dev/posts/89/how-to-use-the-l298n-motor-driver-module> [Accesso: luglio 2025].
- [22] Wikipedia, *Accumulatore litio-polimero*, disponibile su https://it.wikipedia.org/wiki/Accumulatore_litio-polimero, ultimo accesso: luglio 2025.
- [23] Wikipedia, *Accumulatore agli ioni di litio*, disponibile su https://it.wikipedia.org/wiki/Accumulatore_agli_ioni_di_litio, ultimo accesso: luglio 2025.
- [24] Wikipedia, *Convertitore buck*, disponibile su https://it.wikipedia.org/wiki/Convertitore_buck, ultimo accesso: luglio 2025.
- [25] Educetecnica.it *Partitore di resistenze*. Disponibile su: <https://www.edutecnica.it/elettrotecnica/wheatstone/wheatstone.htm#:~:text=Il%20partitore%20di%20tensione%20,sempre%20in%20serie%20alle%20stesse..> Ultimo accesso: 19 luglio 2025.

- [26] ADS1115 datasheet. *ADS1115*. Disponibile su: https://www.ti.com/product/ADS1115?utm_source=google&utm_medium=cpc&utm_campaign=asc-dc-null-44700045496400844_prodfolderdynamic-cpc-pf-google-ww_en_int&utm_content=prodfolddynamic&ds_k=DYNAMIC+SEARCH+ADS&DCM=yes&gclsrc=aw.ds&gad_source=1&gad_campaignid=6465450446&gclid=Cj0KCQjwzt_FBhCEARIIsAJGFWVkf9ycZNh8tWmtMuw8cIm2doISxYWqVCL9ziB1nAqYheMupvRWSymYaAsFYEALw_wkB. Ultimo accesso: 19 luglio 2025.
- [27] adxl345 datasheet. *ADXL345*. Disponibile su: https://www.alldatasheet.com/view.jsp?Searchword=Adxl345%20datasheet&gad_source=1&gad_campaignid=160596484&gclid=Cj0KCQjwzt_FBhCEARIIsAJGFWVluoApxj4TbwN-0XDwQKDyIUTHGvnYiFE6025X5YUzSwYj0K4PKE1UaAk3CEALw_wkB. Ultimo accesso: 19 luglio 2025.
- [28] Wikipedia contributors. *Microelectromechanical systems*. Wikipedia, l'enciclopedia libera. Disponibile su: https://it.wikipedia.org/wiki/Sistemi_microelettromeccanici. Ultimo accesso: 19 luglio 2025.
- [29] HC-SR04 datasheet. *HC-SR04*. Disponibile su: https://www.alldatasheet.com/view.jsp?Searchword=HCSR04&gad_source=1&gad_campaignid=1434060638&gclid=Cj0KCQjwzt_FBhCEARIIsAJGFWVnHmNJ27wTiZdVHqj9Cp1odJV0lgdhLbzEkn_xDivSrn1_0JA10jpAaAskgEALw_wkB. Ultimo accesso: 19 luglio 2025.
- [30] Wikipedia contributors. *Buzzer*. Wikipedia, l'enciclopedia libera. Disponibile su: [https://it.wikipedia.org/wiki/Buzzer_\(componente_elettronico\)](https://it.wikipedia.org/wiki/Buzzer_(componente_elettronico)). Ultimo accesso: 19 luglio 2025.
- [31] Wikipedia contributors. *Optoelettronica*. Wikipedia, l'enciclopedia libera. Disponibile su: <https://it.wikipedia.org/wiki/Optoelettronica>. Ultimo accesso: 19 luglio 2025.
- [32] Wikipedia contributors. *Demosaicing*. Wikipedia, l'enciclopedia libera. Disponibile su: <https://en.wikipedia.org/wiki/Demosaicing>. Ultimo accesso: 19 luglio 2025.
- [33] Wikipedia contributors. *H.264/MPEG-4 AVC*. Wikipedia, l'enciclopedia libera. Disponibile su: <https://it.wikipedia.org/wiki/H.264>. Ultimo accesso: 19 luglio 2025.
- [34] Wikipedia contributors. *USB*. Wikipedia, l'enciclopedia libera. Disponibile su: <https://it.wikipedia.org/wiki/USB>. Ultimo accesso: 19 luglio 2025.
- [35] Wikipedia contributors. *UART*. Wikipedia, l'enciclopedia libera. Disponibile su: https://en.wikipedia.org/wiki/Universal_asynchronous_receiver-transmitter. Ultimo accesso: 19 luglio 2025.
- [36] Wikipedia contributors. *GPS*. Wikipedia, l'enciclopedia libera. Disponibile su: https://it.wikipedia.org/wiki/Global_Positioning_System. Ultimo accesso: 19 luglio 2025.
- [37] Wikipedia contributors. *GNSS*. Wikipedia, l'enciclopedia libera. Disponibile su: https://it.wikipedia.org/wiki/Sistema_satellitare_globale_di_navigazione. Ultimo accesso: 19 luglio 2025.

- [38] Wikipedia contributors. *Raspberry Pi*. Wikipedia, l'enciclopedia libera. Disponibile su: https://it.wikipedia.org/wiki/Raspberry_Pi. Ultimo accesso: 19 luglio 2025.
- [39] Wikipedia contributors. *I²C*. Wikipedia, l'enciclopedia libera. Disponibile su: <https://it.wikipedia.org/wiki/I%C2%B2C>. Ultimo accesso: 19 luglio 2025.
- [40] Wikipedia contributors. *MQTT*. Wikipedia, l'enciclopedia libera. Disponibile su: <https://it.wikipedia.org/wiki/MQTT>. Ultimo accesso: 19 luglio 2025.
- [41] Wikipedia contributors. *GStreamer*. Wikipedia, l'enciclopedia libera. Disponibile su: <https://it.wikipedia.org/wiki/GStreamer>. Ultimo accesso: 19 luglio 2025.
- [42] Internet_of_Things_Projects Disponibile su: https://www.google.it/books/edition/Smart_Internet_of_Things_Projects/foJcDgAAQBAJ?hl=en&gbpv=1&printsec=frontcover. Ultimo accesso: 15 settembre 2025