UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA

Corso di Laurea in Ingegneria Informatica e dell'Automazione

Studio e simulazione di sistemi CPS per veicoli (semi) autonomi esplorativi di ambienti estremi

Study and simulation of CPS system for (semi) autonomous vehicles exploring extreme environments

Relatore
Prof. David Scaradozzi

Candidato
Marco Conte

Correlatrice
Flavia Gioiello

Anno Accademico 2023/2024

Non possiamo risolvere i nostri problemi con lo stesso livello di pensiero che avevamo quando li abbiamo creati.

Albert Einstein

Abstract

La tesi si concentra sullo studio della gestione della comunicazione tra sistemi a basso e alto livello tramite tecnologie IoT, andando più nel dettaglio nell'interazione tra il dispositivo M5StickCPlus, il software MATLAB e i protocolli di comunicazione ROS2 e MQTT. Sono stati esaminati tre casi di studio: l'accensione del LED di M5StickCPlus controllata da MATLAB, la trasmissione dell'input dei pulsanti del microcontrollore verso MATLAB e il controllo dell'angolo di posizione di un servomotore collegato al dispositivo. I risultati hanno dimostrato che MQTT e ROS2 si sono rivelati efficienti per questo prototipo di sistema CPS, dotato di un volume di dati ristretto e una comunicazione rapida in tempo reale.

Indice

Introduzione

L'*Internet of Things* (IoT), tradotto in italiano come l'*internet delle cose*, si riferisce a una rete di dispositivi fisici, veicoli, elettrodomestici e altri oggetti fisici dotati di sensori, software e connettività di rete, che consentono loro di raccogliere e condividere dati. Le potenziali applicazioni dell'IoT sono varie e hanno avuto un grande impatto in un'ampia gamma di settori, tra cui manifatturiero, trasporti, sanitario e agricolo; motivo per cui il suo sviluppo è in costante crescita negli ultimi anni.

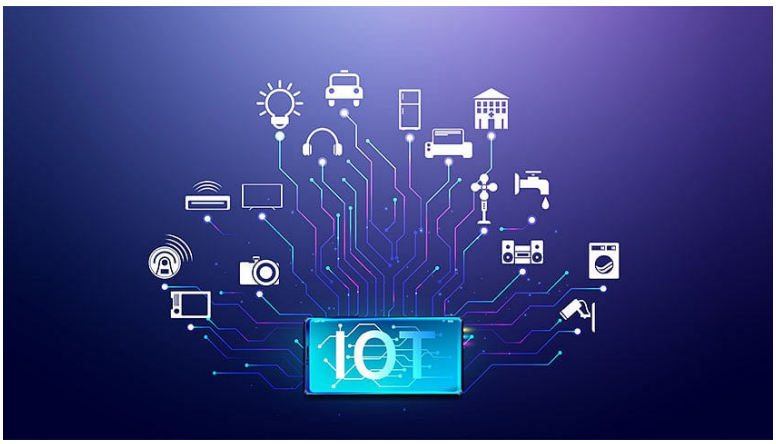

Figura 1. Internet of Things (Fonte: simplilearn.com)

In un contesto aziendale, i dispositivi IoT vengono utilizzati per monitorare una vasta gamma di parametri come temperatura, umidità, qualità dell'aria, consumo energetico e prestazioni di una macchina. Questi dati possono essere analizzati in tempo reale per identificare modelli, tendenze e anomalie che possono aiutare le aziende a ottimizzare le loro operazioni e migliorare il loro risultato economico. Ad esempio, i sensori IoT possono essere utilizzati per monitorare le prestazioni delle apparecchiature e rilevare o risolvere potenziali problemi prima che causino tempi di inoperatività, riducendo i costi di manutenzione e migliorando il tempo di attività [1].

Citando Treccani, l'IoT è la *rete di oggetti dotati di tecnologie di identificazione, collegati fra di loro, in grado di comunicare sia reciprocamente sia verso punti nodali del sistema, ma soprattutto in grado di costituire un enorme network di cose dove ognuna di esse è rintracciabile per nome e in riferimento alla posizione. L'espressione «Internet delle cose» è stata coniata nel 1999 da Kevin Ashton*. Ciò ha portato alla diffusione dei *Cyber-Physical System* (CPS), sistemi che comprendono una rete di dispositivi informatici, i quali interagiscono con sensori e attuatori per manipolare l'ambiente naturale.

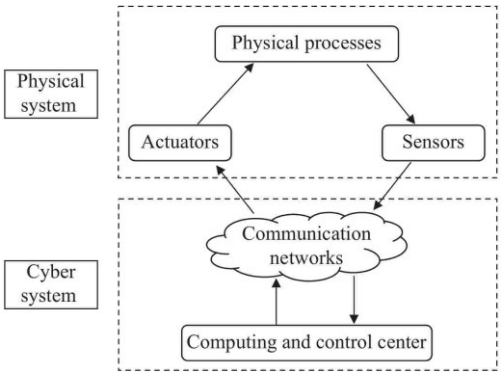

Figura 2. Cyber Physical System (Fonte: ieee-jas.net)

I sistemi cyberfisici sono sistemi automatizzati che consentono la connessione delle operazioni della realtà fisica con infrastrutture di elaborazione e comunicazione. Diversamente dai sistemi embedded, che sono progettati come dispositivi stand-alone, il CPS si concentra sulla coordinazione e comunicazione di diversi dispositivi [3]. Inoltre, con l'IoT e i CPS è possibile creare i *digital twin,* cioè copie virtuali di modelli reali che permettono di effettuare dei test per migliorare le funzionalità e ridurre al minimo tutti i rischi associati alla sperimentazione, ad esempio in ambienti estremi come quello marino.

L’architettura dell’IoT può essere vista come una struttura contenente tre livelli principali: percezione, trasporto ed applicazione [2] (Figura 3).

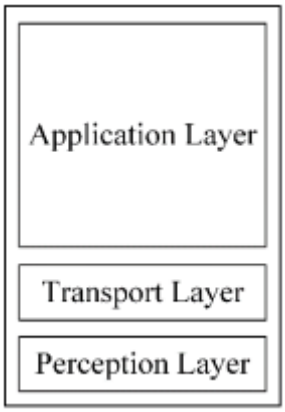

Figura 3. Livelli dell'architettura IoT (Fonte: [2])

- **Livello di percezione:** è un insieme di oggetti che fungono da intermediario tra l'ambiente e il digitale. Il suo scopo è quello di catturare i dati dall'ambiente utilizzando vari sensori quali temperatura e umidità, gas, luci, GPS e altri a seconda delle esigenze dell'applicazione.
- **Livello di trasporto:** questo livello mira a collegare gli oggetti e condividere informazioni tra gli stessi connessi in modo sicuro.
- **Livello di applicazione:** questo livello è responsabile della fornitura di applicazioni specifiche e di servizi agli utenti, occupandosi della gestione e dell’interpretazione dei dati.

Questa tesi si occupa di spiegare il funzionamento di una delle attività principali dell’IoT: la comunicazione M2M (macchina a macchina) tra più dispositivi. In particolare, tramite lo studio di tre esempi, si vuole approfondire la comunicazione dal livello di percezione al livello di applicazione. Il primo corrisponde all’hardware del sistema (il cosiddetto *basso livello*), composto dal microprocessore M5StickCPlus con i suoi attuatori integrati, cioè il LED e i

pulsanti, e il servomotore; il secondo è ciò che riguarda l'*alto livello*, rappresentato da MATLAB. La comunicazione è possibile tramite l'utilizzo del protocollo MQTT (*Message Queuing Telemetry Transport*) e del middleware ROS2 (*Robot Operating System 2*) che gestiscono lo scambio di messaggi e rappresentano il livello di trasporto. Il collegamento dei dispositivi costituisce un prototipo di CPS.

Nel primo capitolo, si esamina lo stato dell'arte dell'IoT e dei Cyber Physical System; nel secondo si analizzano i materiali hardware e software utilizzati nei tre esempi con la spiegazione di ognuno di essi; il terzo capitolo si focalizza principalmente sui risultati del progetto e infine nell'ultima sezione si hanno le conclusioni e gli sviluppi futuri.

1. Stato dell’arte

I sistemi cyberfisici nascono per rispondere alla crescente esigenza di un'integrazione sempre più stretta tra componenti fisici e sistemi informatici. La necessità principale risiedeva nella capacità di gestire in tempo reale sistemi complessi, che combinano dispositivi fisici, quali sensori, attuatori e macchinari, con software e reti digitali. Gli approcci tradizionali, che trattavano separatamente queste due dimensioni, non risultavano più adeguati a garantire i livelli di efficienza, affidabilità e sicurezza richiesti nei nuovi contesti, come le Smart Cities, i veicoli autonomi e le fabbriche intelligenti. I CPS sono stati preferiti per la loro abilità nel connettere il mondo fisico e digitale, consentendo interazioni continue, monitoraggio in tempo reale e adattamenti autonomi. Tuttavia, permangono criticità legate alla sicurezza, all'affidabilità e alla complessità progettuale di tali sistemi.

1.1. Ruolo dei CPS

Il sistema cyberfisico (CPS) svolge una funzione poiché si sta diffondendo una nuova generazione di sistemi multidisciplinari con capacità computazionali e fisiche integrate che permettono di realizzare in tempo reale una collaborazione sicura, affidabile e dinamica con i sistemi embedded. La tecnologia si basa sulla disciplina più vecchia - ma ancora molto giovane - dei sistemi embedded; la missione principale dei CPS non è l'elaborazione ma l’integrazione delle dinamiche dei processi fisici con quelle del software e del networking, fornendo astrazioni e tecniche di modellazione, progettazione e analisi per l'insieme integrato. Quindi, i CPS sono in grado di risolvere problemi legati alla gestione, all’ottimizzazione e alla sincronizzazione dei componenti dei sistemi complessi. Integrano il mondo fisico con il calcolo avanzato combinando i dati provenienti dai sensori per poter prendere decisioni o svolgere determinate

azioni, ad esempio per coordinare e controllare gli elettrodomestici in una Smart Home oppure un insieme di pesci robotici che monitorano l'ambiente marino. Computer e reti embedded monitorano e controllano i processi fisici con cicli di feedback in cui gli stessi processi influenzano le elaborazioni e viceversa [3]. Le dinamiche tra computer, reti e sistemi fisici interagiscono in modi che richiedono un nuovo design di tecnologia [4]. Il potenziale economico e sociale di tali sistemi è di gran lunga maggiore di quanto realizzato finora e si stanno facendo grandi investimenti in tutto il mondo per sviluppare questa nuova tecnologia [3]. I CPS consistono generalmente in tre elementi principali: sensori, attuatori e controllore. Questi tre elementi contribuiscono a ridurre i costi di esercizio e ottimizzare il periodo di elaborazione dei dati. Vengono applicati negli strumenti medici digitali, nella tecnologia di controllo, anche aerospaziale, nel sistema sociale e in quello autonomo e nella sorveglianza. Il principale scopo dei CPS è di effettuare un controllo del feedback nel sistema informatico tramite le cosiddette *tre C:* calcolo, comunicazione e controllo. I processi di calcolo e quelli fisici monitorano e controllano un'entità fisica in tempo reale. Le caratteristiche principali dei CPS sono:

- Sistema fisico
- Sistema cyber e informatico
- Prodotto di integrazione di sistemi eterogenei
- Requisiti di sicurezza, capacità in tempo reale e prevedibilità

Il sistema fisico corrisponde agli attuatori o ai sistemi elettromeccanici. Il sistema cyber si riferisce ai sensori e al controllo logico, mentre quello informatico riguarda la rete, l'archiviazione e la memoria. Il terzo punto si occupa dell'integrazione e dell'interazione tra sistema fisico e sistema informatico. Infine, il CPS deve essere scalabile, tollerante alle minacce e in grado di superare l'incertezza del sistema. Quindi, questi sistemi si fondano su più oggetti

connessi tra di loro che, attraverso sensori, attuatori e connessioni in rete, acquisiscono dati di varia natura, si scambiano informazioni a vicenda stabilendo una comunicazione dall'alto al basso livello e viceversa a seconda del sistema. I sistemi cyberfisici, insieme all'IoT, fanno parte delle tecnologie di quella che è stata denominata *Industria 4.0*.

1.2. Industry 4.0

Questo termine è stato presentato alla Fiera di Hannover nel 2011 e ha ottenuto un gran successo anche dopo la pandemia COVID-19 che ha accelerato il processo digitale delle aziende, modificando anche il loro approccio al mercato. Nel settore dei CPS, la Germania ha un ruolo di primo piano e può contare su quasi 20 anni di esperienza. All'inizio del XX secolo, le industrie manifatturiere hanno iniziato a generare una massiccia produzione a causa della crescente domanda del mercato globale. L'obiettivo dell'Industria 4.0 è far emergere un nuovo tipo di azienda, noto come *Smart Factory,* dotato di [5]:

- **Reti intelligenti**: le Smart Factory sono composte da sistemi e attrezzature automatizzati e costantemente interconnessi con l'aiuto della tecnologia informatica, servizi di comunicazione, attuatori e sensori intelligenti e tecnologie delle telecomunicazioni.
- **Mobilità**: tramite i dispositivi mobili è possibile fornire un accesso indipendente dal punto di vista temporale e spaziale ai processi e servizi dei sistemi automatizzati. Questo crea una nuova dimensione nella diagnostica, manutenzione e funzionamento di questi sistemi.
- **Flessibilità**: è possibile selezionare l'offerta migliore da un ampio pool di fornitori di componenti, moduli e servizi. La diagnosi può essere effettuata in parte dall'utente. L'informazione può essere recuperata su richiesta e viene utilizzata in modo intelligente

al fine di ottenere una diagnosi autonoma. I pezzi di ricambio possono essere ordinati automaticamente presso i produttori più economici, contrastando così il problema della carenza di competenze.

- **Integrazione dei clienti**: con l'industria 4.0 è possibile personalizzare prodotti a specifiche e individuali esigenze e bisogni dei clienti di tutte le fasce d'età. Si tratta di sistemi automatizzati che assisteranno le persone in tutte le situazioni e nelle diverse fasi della vita.
- **Nuovi modelli innovativi di business**: i prodotti diventano modulari e configurabili in modo che possano essere adattati alle esigenze specifiche.

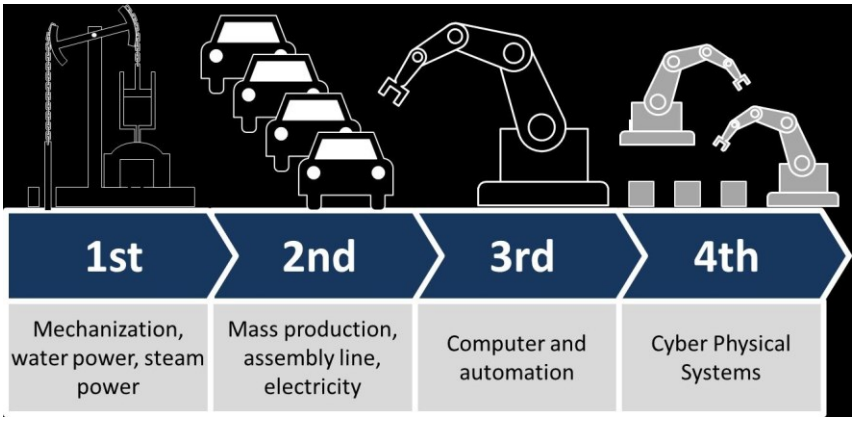

Figura 1.1. Rivoluzioni Industriali (Fonte: terasoft.com)

La Figura 1.1 mostra l'evoluzione del processo industriale dalla prima rivoluzione industriale alla quarta in modo da raggiungere più efficienza e produttività attraverso un controllo intelligente di sistemi embedded connessi in rete. L'industria 4.0 si basa sulle invenzioni della Terza Rivoluzione Industriale, o rivoluzione digitale, che si è sviluppata dagli anni '50 e fino ai primi anni 2000 e ha portato computer, altri tipi di elettronica, Internet e molto altro. Con l'ultima rivoluzione si sono diffusi quattro tipi fondamentali di tecnologie prorompenti che possono essere applicate lungo tutta la catena del valore: **connettività, dati e potenza di**

calcolo (tecnologia cloud, Internet, blockchain, sensori); **analytics e intelligenza** (analisi avanzata, apprendimento automatico, intelligenza artificiale); **interazione uomo-macchina** (realtà virtuale (VR) e realtà aumentata (AR), robotica e automazione, veicoli guidati autonomi) e **ingegneria avanzata** (produzione additiva, energia rinnovabile, nanoparticelle) [6]. Si indica con il termine Industria 4.0 la propensione dell'odierna automazione industriale a inserire alcune nuove tecnologie produttive per migliorare le condizioni di lavoro, creare nuovi modelli di business, aumentare la produttività degli impianti e migliorare la qualità dei prodotti [7].

1.3. Applicazioni

L'IoT e i CPS è un paradigma che non conosce, potenzialmente, confini applicativi. Si parte dall'autovettura intelligente (Smart Car) che dialoga con l'infrastruttura stradale per prevenire incidenti, agli elettrodomestici di casa che si coordinano per ottimizzare l'impegno di potenza (Smart Home). Esistono poi diverse applicazioni IoT in impianti di produzione che scambiano dati con i manufatti per la gestione del loro ciclo di vita. Infine, si passa dai dispositivi medicali, che si localizzano nel presidio di un pronto soccorso, agli sci che inviano informazioni sullo stato della neve. Si illustrano di seguito i principali ambiti di applicazione dell'IoT e dei CPS, con focus sulle proprietà degli *smart objects* e sulle tecnologie abilitanti [8].

- Con *Smart Car*, nel campo della mobilità intelligente con l'obiettivo di garantire la massima efficienza di spostamento, flessibilità, sicurezza e convenienza, si intende la connessione delle auto per comunicare informazioni in tempo reale al consumatore, connessione tra veicoli o tra questi ultimi e l'infrastruttura circostante per la prevenzione e la rivelazione degli incidenti, nonché l'offerta di nuovi modelli assicurativi e/o di informazioni geo-referenziate sulla viabilità.

L’obiettivo delle auto smart e delle relative tecnologie è quello di agevolare la guida del veicolo e aumentare la sicurezza stradale, anche in termini di guida autonoma, che include tutte le tecnologie hardware e software che permettono di rilevare potenziali rischi derivanti sia da elementi esterni al veicolo che interni. È possibile anche ricevere informazioni circa la manutenzione e il monitoraggio dei componenti e fornire dati in tempo reale, come le condizioni del traffico [9].

Figura 1.2. Smart Car (Fonte: fastweb.it)

- La *Smart Home* riguarda il mondo della domotica, la robotica della casa, cioè la scienza delle tecnologie che cercano di migliorare la vita all’interno delle proprie abitazioni. Il termine si riferisce alla possibilità di gestire in maniera automatica o da remoto impianti e dispositivi, al fine di risparmiare energia, semplificare la vita domestica e garantire la sicurezza delle persone all’interno. Si possono gestire l’illuminazione, l’impianto di riscaldamento, di climatizzazione e di sorveglianza, gli elettrodomestici e anche ottenere

informazioni sui consumi energetici, la salubrità della casa e avvisi in caso di cadute di anziani in casa o assistenza a disabili [10].

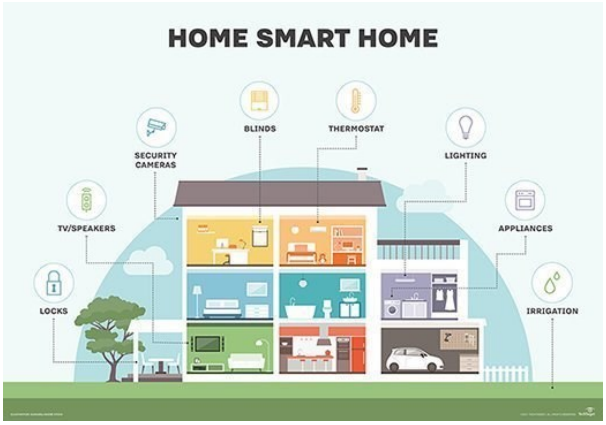

Figura 1.3. Smart Home (Fonte: bestarthome.com)

- L'espressione *Smart City* racchiude in sé una concezione della realtà urbana che travalica i confini tecnologici e che – in una visione ampia che spazia dalla mobilità all'efficienza energetica, dalla valorizzazione dei dati alla partecipazione attiva dei cittadini – si pone come obiettivo l'innalzamento degli standard di sostenibilità, vivibilità e dinamismo economico delle città del futuro. Tra i benefici chiave, infatti, rispetto a una città tradizionale si rilevano: riduzione dei consumi energetici, ottimizzazione della raccolta dei rifiuti, sviluppo del trasporto pubblico e digitalizzazione dei servizi per i cittadini che

Figura 1.4. Smart City (Fonte: bnova.it)

aiutano a far fronte alla crescita della popolazione urbana e a ridurre l'impatto delle città sull'ambiente e sull'inquinamento. Ci sono diverse Smart City nel mondo, tra cui Zurigo, Barcellona, Helsinki, che hanno adottato modelli per la valorizzazione dei dati**,** per aumentare le competenze e per coinvolgere i cittadini nei processi decisionali. Inoltre, esistono anche progetti di Smart City in Italia**,** tra questi Milano, ma anche Torino**,** Trento o Firenze, che hanno come obiettivo quello di migliorare la qualità di vita dei cittadini all'interno dei contesti urbani [11].

- Anche il settore dell'agricoltura e quello energetico sono interessati da questo fenomeno, generando la *Smart Agricolture* e la *Smart Energy*. Con la prima si intende la raccolta automatica, l'integrazione e l'analisi di dati provenienti dal campo, da sensori e da qualsiasi altra fonte terza. Gli scopi sono quelli di fornire il supporto all'agricoltore nel processo decisionale relativo alla propria attività e al rapporto con altri soggetti della filiera e aumentare la profittabilità e la sostenibilità economica, ambientale e sociale dell'agricoltura [12]. Con il secondo termine si indica l'integrazione e il coordinamento di tutte quelle

Figura 1.5. Smart Agricolture

applicazioni, soluzioni e progettualità per la gestione dell'energia che consentono un miglioramento dell'efficienza energetica di case, edifici e impianti, cercando anche di ottimizzare il livello di comfort per gli occupanti degli spazi smart. Si utilizzano gli *smart meter*, cioè contatori connessi per la misura dei consumi (elettricità, gas, acqua e calore), oppure la *smart grid*, o rete intelligente, usata per la gestione dell'energia e per ottimizzare la distribuzione, gestendo produzione distribuita e mobilità elettrica [13].

Nell'ambito sanitario, i CPS sono utili a memorizzare le informazioni della cartella clinica. In caso di urgenza, le informazioni possono essere immediatamente accessibili da diversi centri medici. Nel centro medico intelligente i CPS vengono applicati per le sale operatorie e per il controllo del flusso di fluidi con dispositivi controllati dalla rete. Nell'ambito dei trasporti, i CPS possono servire alla gestione del traffico aereo e degli aeromobili fornendo servizi in tempo reale quali i comandi di volo e sistemi critici per la sicurezza [14].

Figura 1.6 Sala operatoria con CPS (Fonte: [14])

Secondo la Federazione internazionale di robotica, sono circa 20 milioni i robot sulla Terra, mentre quelli industriali già operativi sono oltre 3 milioni, con 14083 presenti in Italia. Nel 2021 le nuove installazioni sono aumentate del 65%: il nostro Paese è il secondo mercato d'Europa e il sesto nel mondo [15]. Infatti, analizzando i numeri è possibile notare un incremento dell'utilizzo in questi ultimi anni. Nel 2021 è cresciuta la diffusione del numero di auto connesse in Italia, per un totale di 18,4 milioni di Smart Car [9]. Nel 2022 il mercato delle soluzioni Internet of Things per la Smart Home in Italia ha confermato un tasso di crescita del +18% rispetto al 2021, toccando quota 770 milioni di euro. A spingere il mercato sono stati soprattutto gli incentivi, quali "Superbonus" ed "Ecobonus", che hanno trainato le vendite dei dispositivi smart legati al risparmio energetico, come caldaie, valvole termostatiche, climatizzatori e termostati [10].

1.4. Debolezze dell'IoT e dei CPS

L'aspetto negativo dell'IoT è la minaccia alla sicurezza e alla privacy. Nel 2022 la maggior parte delle aziende operanti in Europa nel mercato Smart Home (e non solo) ha dovuto dimostrare di essersi conformata al Regolamento europeo GDPR (General Data Protection Regulation). Per quanto riguarda la sicurezza informatica, invece, il 15 settembre 2022, la Commissione Europea ha rilasciato il *Cyber Resilience Act* (CRA), un regolamento che introduce nuove direttive per produttori e venditori di prodotti digitali, al fine di garantire la sicurezza informatica del consumatore [10]. Quest'ultimo si basa su cinque principi fondamentali: i produttori dovranno migliorare la sicurezza dei dispositivi già in fase di progettazione e sviluppo, e non solo, rilasciando vari aggiornamenti di sicurezza; le aziende dovranno garantire l'assenza di vulnerabilità su tutti i prodotti venduti e, in caso contrario,

dovranno notificare l'ENISA (l'agenzia europea per la sicurezza informatica) tempestivamente nonché una maggiore trasparenza delle proprietà di sicurezza informatica che caratterizzano i propri prodotti. Sarà infine necessaria la presenza di un'autorità di vigilanza del mercato alla quale tutti i produttori sono tenuti a fornire informazioni sulla conformità con il CRA. Per quanto riguarda la sensibilità degli utenti, questa varia significativamente a seconda del tipo di applicazione considerata. Ad esempio, ben il 45% dei consumatori italiani si dichiara preoccupato per l'utilizzo dei dati raccolti dagli oggetti smart in casa, mentre il 28% lo è in relazione ai dati di un'auto connessa [16].

1.5. Innovazioni e tecnologie

In occasione della RomeCup 2023, sono stati presentati dei nuovi progetti che fondono il mondo dell'IoT, dell'intelligenza artificiale e della robotica nei diversi ambiti. Sono stati esibiti il tool di linguistica computazionale per analizzare testi in modo automatico ed estrarre importanti informazioni circa la loro forma e il loro contenuto, ***Rid-it***, sviluppato dal Cnr di Pisa, ed il sistema robotico indossabile ***AKO***, strumento riabilitativo o assistivo per il movimento del ginocchio per poter camminare, salire o scendere le scale, alzarsi o sedersi, inventato dall' Istituto di robotica della Scuola Superiore Sant'Anna di Pisa in collaborazione con l'azienda spin off IUVO. Inoltre, nel campo della robotica biomedica, sono stati esposti anche ***Atlas***, l'esoscheletro indossabile per la riabilitazione dei bambini tra i 4 e i 10 anni con paralisi cerebrale infantile e altre malattie neuromuscolari, prodotto dall'Irccs San Raffaele di Roma; il progetto europeo ***Vrailexia*** dell'Università degli Studi della Tuscia che offre supporto a studenti dislessici e il progetto ***Hover Easy*** di OZ Officine Zero che motorizza le sedie a rotelle per garantire una maggiore mobilità e indipendenza ai disabili. Sono stati presentati

anche robot di servizio, cioè non a scopo industriale, tra cui ***TIAGo***, prodotto da PAL Robotics S.L., utile a svolgere funzioni complesse come quelle di percezione, navigazione, manipolazione e interazione con l'uomo, e ***MARRtino***, piattaforma robotica a basso costo e di semplice utilizzo basata su ROS, sviluppata dall'Università Sapienza di Roma e adatta a implementare task come la navigazione, l'interazione uomo-robot e l'analisi delle immagini. Al RomeCup ha partecipato anche l'Università Politecnica delle Marche insieme a LabMACS, ANcybernetics e DII con ***Guizzo***, un pesce robotico programmabile dotato di una coda motorizzata, utilizzato come kit robotico educativo per le scuole.

Infine, si è esposto il veicolo robotico ***Shark,*** sviluppato dai ricercatori di Cnr-Istituto di ingegneria del mare, basato su elettronica Raspberry e dotato di motori brushless. È stato utilizzato nel 2015 nel progetto di ricerca *Unmanned Vehicles for Autonomous Sensing and Sampling*, nella campagna scientifica nel fiordo Kongsfjord presso l'arcipelago delle Isole Svalbard, per acquisire dati atmosferici, marini e glaciali in luoghi pericolosi [15].

Nel campo dei pesci motorizzati in ambienti estremi sono stati sviluppati anche altri veicoli esplorativi. In generale, esistono due tipi di veicoli senza equipaggio: AUV (Autonomous Underwater Vehicle) o ROV (Remotely Operated Vehicle). Il primo esplora le profondità oceaniche senza cavi collegati, quindi viene lanciato nell'oceano e poi raccolto in una posizione preselezionata; invece il secondo è collegato a una nave tramite cavi che trasmettono segnali di comando e controllo tra l'operatore e il ROV, consentendo la navigazione a distanza del veicolo. I ROV sono spesso utilizzati quando le immersioni da parte degli esseri umani sono poco pratiche o pericolose, come quando si lavora in acque profonde o si indagano su pericoli sommersi. I ROV e gli AUV trasportano attrezzature come videocamere, luci, bracci robotici per afferrare le cose e studiare l'oceano in sicurezza andando dove gli esseri umani non possono. Entrambi possono essere utilizzati per missioni di ricognizione subacquea come il

rilevamento e la mappatura di relitti sommersi, rocce e ostacoli che possono rappresentare un pericolo per la navigazione di imbarcazioni commerciali e da diporto. Questi veicoli non sono concepiti per sostituire le indagini idrografiche dei subacquei, ma potrebbero fungere da sostituti se i subacquei non sono disponibili o la loro sicurezza è in discussione [17]. Ad esempio, il robot U-CAT è un veicolo autonomo, di piccole dimensioni e a basso costo per la penetrazione di relitti. Fa parte del progetto ARROWS (archaeological robot systems for the world's seas), il cui scopo è quello di sviluppare strumenti che assistano gli archeologi subacquei durante i loro studi [18]. Utilizza un sistema di attuazione a quattro pinne che conferisce al veicolo un'elevata manovrabilità per operare in ambienti complessi con pareti, corde, reti e altri ostacoli. L'uso di pinne consente anche un movimento più silenzioso rispetto alle eliche tradizionali e inoltre rimuovono meno sedimenti dal fondo e dalle pareti, mantenendo così una maggiore visibilità [19]. Lo studio [20] ha proposto l'allestimento di uno speciale ROV destinato a documentare il sito archeologico prima e dopo le attività di restauro condotte da subacquei attraverso l'uso di una serie di strumenti innovativi progettati per pulire le strutture e rimuovere gli organismi viventi (alghe, spugne, molluschi, ecc.) e per eseguire la manutenzione programmata necessaria per prevenire la colonizzazione biologica dei manufatti sommersi. Un altro esempio di robot submarini esplorativi è AQUA, dotato di sei pagaie, le quali si comportano da superfici di controllo durante il nuoto e come gambe mentre si cammina. Ciò gli permette di determinare lo stato attuale dell'ambiente misurando la corrente e riconoscere la natura dei diversi terreni. Inoltre, l'università di Okayama ha progettato un AUV biologicamente ispirato ad una manta per l'esplorazione dell'ecologia subacquea [18]. Infine, si cita il robot biomimetico marino BRAVe, [18], targato UNIVPM, che presenta come innovazione il sistema di propulsione ibrido che rende il sistema nel suo insieme efficiente: è composto da un propulsore biomimetico a forma di pinna caudale che permette la navigazione in vaste aree

risparmiando batteria e massimizzando l'autonomia e da eliche orizzontali a vite che consentono al robot di fermarsi o accelerare. Nel primo propulsore è stata installata una trasmissione meccanica innovativa che è in grado di trasformare la rotazione a velocità costante di un motore elettrico nell'oscillazione armonica della pinna [21]. È dotato anche di un propulsore verticale che controlla la profondità (Figura 1.7).

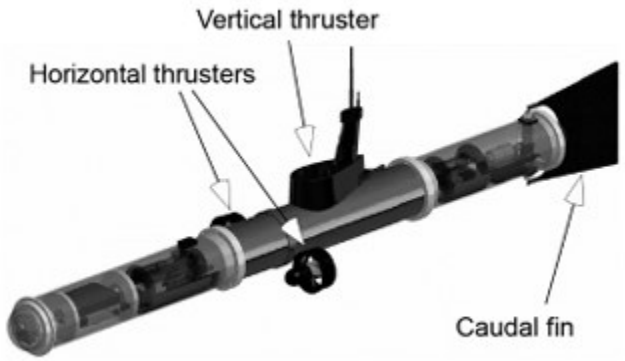

Figura 1.7. Propulsori di BRAVe (Fonte: [18])

2. Materiali e metodi

In questo capitolo si pone lo sguardo ai materiali utilizzati per la realizzazione del progetto, in particolare si esaminerà la loro tipologia e la loro funzione. La seconda parte è dedicata ad un focus sui tre esempi studiati

2.1. Materiali

Nella prima sezione del paragrafo, si elencano i materiali che riguardano l'hardware e il software usato nel sistema.

2.1.1. M5StickCPlus

M5StickC è un dispositivo compatto e portatile di M5Stack, una piattaforma di sviluppo hardware caratterizzata da modularità e facilità nell'utilizzo. M5Stack, quindi, è un sistema che può essere composto da più unità che possono essere sovrapposte in modo da formare una pila, da qui il termine *Stack.* Generalmente, si collega un microcontrollore con una serie di componenti, tra cui schermi integrati - anche del tipo a colori OLED, che permettono la visualizzazione diretta di informazioni o grafici per l'utente - sensori, altoparlanti, pulsanti [22]. La funzione principale di M5Stack è quella di semplificare lo sviluppo di progetti IoT ed embedded relativi, al monitoraggio ambientale o all'automazione industriale, e più in generale, al mondo della domotica o della robotica.

Figura 2.1. M5StickCPlus (Fonte: [24])

In questo progetto, è stata utilizzata la versione più aggiornata di M5StickC, M5StickCPlus (Figura 2.1), dotata di uno schermo più ampio e di una struttura hardware più stabile. La scheda di sviluppo è basata sul microcontrollore ESP32, creato e sviluppato dall'azienda cinese *Espressif Systems* [23]*;* in particolare è il ESP32-PICO-D4 dual-core a 240MHz con 4 MB di memoria flash, dotato di Wi-Fi, Bluetooth e una batteria integrata al litio ricaricabile da 120 mAh che consente un utilizzo autonomo per brevi periodi [24]. Le caratteristiche dell'M5 sono elencate nella Figura 2.2.

ESP32	240MHz dual core, 600 DMIPS, 520KB SRAM, Wi-Fi
Flash Memory	4MB
Power Input	5V @ 500mA
Port	TypeC x 1, GROVE(I2C+I/O+UART) x 1
LCD screen	1.14 inch, 135*240 Colorful TFT LCD, ST7789v2
Button	Custom button x 2
LED	RED LED
MEMS	MPU6886
Buzzer	built-in buzzer
IR	Infrared transmission
MIC	SPM1423
RTC	BM8563
PMU	AXP192
Battery	120 mAh @ 3.7V
Antenna	2.4G 3D Antenna
PIN port	G0, G25/G36, G26, G32, G33
Operating Temperature	0°C to 60°C
Net weight	15g
Gross weight	21g
Product Size	48.2*25.5*13.7mm
Package Size	65*25*15mm
Case Material	Plastic (PC)

Figura 2.2. Caratteristiche M5StickCPlus (Fonte: [24])

Come si rileva dalla tabella, l'M5 è caratterizzato da sensori e attuatori integrati. Tra i sensori sono presenti l'IMU (Inertial Measurement Unit) che consiste in un accelerometro e un giroscopio, utili a rilevare e a misurare movimento, inclinazione e orientamento del dispositivo; un microfono per il riconoscimento vocale; l'RTC (Real-Time Clock), l'orologio in tempo reale e un sensore ad infrarossi per il controllo remoto di dispositivi. Gli attuatori principali comprendono un buzzer che genera suoni, un LED rosso in cima al dispositivo e due pulsanti che possono essere programmati dall'utente, uno centrale (A) sotto lo schermo e l'altro (B) sul lato destro [24].

L'M5 è provvisto nella sua parte inferiore di una porta USB-C per collegarlo al computer e caricare il codice o trasferire dati. Invece, nella parte superiore sono presenti dei canali GPIO

(General Purpose Input Output) che consentono il collegamento con altri dispositivi, tra cui *l'8Servos HAT v1.1* (Figura 2.3).

Figura 2.3. 8Servos HAT v1.1 (Fonte: [25])

2.1.2. 8Servos HAT v1.1

Questo componente è un modulo driver che utilizza il microcontrollore STM32F030F4 per guidare i servi con segnale PWM (Pulse Width Modulation). È dotato di un circuito di gestione dell'alimentazione per controllare il servo, di una batteria ricaricabile al litio 16340 (con capacità di 700mAh) per garantire un'alimentazione esterna e indipendente e anche di una protezione per evitare danni in caso di inserimento errato della batteria. La serie HAT è compatibile con M5Stack tramite la comunicazione I2C. (Figura 2.4) [25]

Figura 2.4. Collegamento M5StickCPlus-8Servos HAT v1.1 (Fonte: m5stack.com)

L’Inter Integrated Circuit (I2C) è un bus di comunicazione seriale, introdotto da Philips nel 1982, che permette la trasmissione di dati tra diversi dispositivi. Infatti, l’M5StickCPlus si comporta da master e trasmette le informazioni all’8Servos HAT v1.1 che agisce da slave. Vengono utilizzate due linee: la linea di dati seriale (SDA) e la linea di clock seriale (SLC), che consentono rispettivamente lo scambio di dati e la rilevazione della frequenza, sincronizzando master e slave. Dalla documentazione ufficiale di M5Stack si legge che bisogna collegare i pin GPIO 21 e 22 dell’M5StickCPlus rispettivamente ai pin SDA e SCL del driver e inoltre anche i pin dell’alimentazione di 5V e GND di entrambi i dispositivi. Inoltre, l’8Servos utilizza un indirizzo I2C predefinito pari a 0x36. Le caratteristiche del modulo driver sono riassunte nella Figura 2.5. [25]

Specification	Parameters
Lithium Battery	Specification: 16340, Capacity: 700mAh
Servo drive channel	8-channel
Maximum drive load capacity	8-channel maximum load capacity: DC4.2V@1.3A
Driver no-load standby current	DC4.2V@2.2uA
Fixing hole	M3
Communication Protocol	I2C:0x36
Net weight	28.3g
Gross weight	39.7g
Product Size	52*38*19mm
Package Size	75*46*29mm

Figura 2.5. Caratteristiche 8Servos HAT v1.1 (Fonte: [25])

È possibile collegare otto servomotori al modulo driver e comandarli individualmente, assegnando l’angolo di posizione.

2.1.3. Servo SG90

Si è adottato il servo SG90 (Figura 2.6) perché compatibile con la serie HAT e la famiglia M5StickC. È un servomotore da 9g di piccole dimensioni, molto semplice da usare, dotato di un ingranaggio di plastica che può compiere una rotazione massima di 180°, e che lavora ad una frequenza di 50Hz. La larghezza di impulso richiesta varia da 1 a 2ms (la posizione -90° corrisponde a 1ms, 0° a 1.5ms e 90° a 2ms). Andando oltre questo intervallo, si rischia di danneggiare il servo. È dotato di tre fili che devono essere collegati ai pin del canale scelto del driver 8Servos: il filo rosso è per la tensione di alimentazione Vcc, il filo giallo per il segnale di controllo e quello marrone per la massa. [26]

Figura 2.6. Servo SG90 (Fonte: [26])

Ora, si approfondisce il software del progetto.

2.1.4. Arduino IDE

In relazione all'impiego, M5StickCPlus ha il vantaggio di poter essere programmato tramite Arduino IDE, MicroPython o UIFlow. In questo progetto, si è utilizzato Arduino IDE (Integrated Development Environment), un ambiente di sviluppo integrato che permette di

caricare codici e comunicare con l'M5StickCPlus. Contiene una barra di strumenti e un editor di testo per scrivere i programmi, chiamati *sketch*, in un linguaggio derivato dal C, definendo almeno le due funzioni obbligatorie: *void setup()*, che viene eseguita una volta sola all'inizio del programma al fine di configurare l'ambiente iniziale, e *void loop()* che viene invocata ad ogni ciclo finché non viene rimossa l'alimentazione. Nel software è incluso un elenco di librerie che forniscono funzionalità da utilizzare negli *sketch*, tuttavia, è possibile scaricarne delle altre tramite *Library Manager* [27].È importante selezionare gli strumenti che si utilizzano prima di caricare lo sketch; in particolare, bisogna scegliere la scheda M5StickCPlus, dopo aver scaricato la libreria omonima e le schede di M5Stack tramite il *Board Manager*, e assicurarsi che sia collegata alla porta USB. La Figura 2.7 rappresenta la schermata principale di Arduino IDE con le funzioni setup e loop, i tasti per verificare, caricare, fare il debug del codice e per scegliere o scaricare schede e librerie [28].

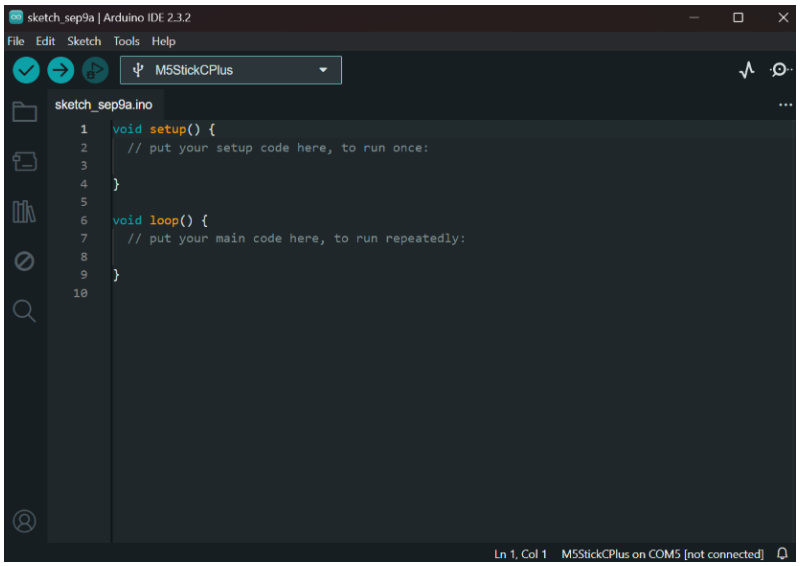

Figura 2.7. Interfaccia Arduino IDE

Nel progetto sviluppato in questa tesi, gli argomenti finora affrontati rappresentano il *basso livello*, quindi la gestione diretta dei componenti fisici e l'interazione con periferiche hardware e con i pin di input od output (GPIO).

Per quanto riguarda la comunicazione tra basso e alto livello, sono stati utilizzati ROS (Robot Operating System) e MQTT (Message Queue Telemetry Transport).

2.1.5. ROS

Figura 2.8. Logo di ROS (Fonte: it.wikipedia.org)

Il ROS non è un sistema operativo, come si direbbe dal nome, ma è un software o framework gratuito e open-source che definisce componenti, interfacce e strumenti per agevolare la costruzione, la programmazione e la comunicazione di robot avanzati. ROS si è sviluppato e ha ottenuto sempre più successo, secondo i creatori Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler e Andrew Ng, grazie a cinque punti cardini [29]:

- **Configurazione peer-to-peer**: un sistema costruito con ROS è basato su un'architettura a grafo in cui i nodi rappresentano i processi, i quali possono scambiarsi messaggi indistintamente poiché si trovano tutti allo stesso livello; pertanto, non può essere definita una comunicazione del tipo Client-Server. Ciò permette di evitare un

sovraccarico di messaggi, che altrimenti si potrebbe verificare nel caso in cui fosse collegato un numero elevato di componenti periferici ad un server centrale.

- **Multilingue**: ROS supporta quattro linguaggi diversi: C++, Python, Octave e LISP. Per supportare lo sviluppo cross-linguistico, ROS si avvale di un semplice linguaggio neutro di definizione delle interfacce (IDL) per descrivere i messaggi inviati tra i moduli. L'IDL utilizza brevi file di testo per permettere la composizione dei messaggi e per descriverne i campi di ciascuno.

- **Basato su strumenti**: per gestire la complessità del ROS si è scelto un design a microkernel, in cui un gran numero di piccoli strumenti sono utilizzati per costruire e far funzionare il ROS, con il quale si ottiene più stabilità a discapito dell'efficienza, evitando di adottare un'architettura monolitica.

- **Leggero**: si preferisce sviluppare driver e algoritmi in librerie che non hanno dipendenze con ROS in modo da creare solo dei piccoli eseguibili che espongono le funzionalità della libreria al software, permettendo di poter riutilizzare il codice.

- **Libero e open-source**: ROS è distribuito secondo la licenza BSD, che permette lo sviluppo di progetti commerciali e non, con il suo codice sorgente disponibile a tutti.

Inoltre, ROS si differenzia dagli altri framework legati al mondo della robotica per la sua facilità e velocità di sviluppo. Attualmente, opera principalmente su sistemi Unix, in particolare Ubuntu, ma si sta cercando di renderlo disponibile anche per altri tra cui Mac OS e Windows.

In realtà, nel progetto si è usato ROS2, versione Humble Hawksbill, che è stato sviluppato come conseguenza della grande espansione della sua versione originale. Tale diffusione ha determinato la necessità di superare i difetti di ROS in termini di efficienza, sicurezza, affidabilità ed elaborazione in tempo reale con compatibilità multipiattaforma, ad esempio Linux, Windows, Mac, Real-Time OS (RTOS). È stato modificato il protocollo di rete passando

al DDS (Data Distribution Service) che garantisce la comunicazione peer-to-peer tra i nodi, senza la necessità di un Ros Master che permette il collegamento dei nodi in ROS. DDS è uno standard industriale pubblicato dall'Object Management Group (OMG) che definisce un middleware per la distribuzione affidabile e in tempo reale dei dati secondo il paradigma di pubblicazione/sottoscrizione, consentendo così a ROS2 di soddisfare i requisiti di sicurezza, protezione, resilienza e tolleranza ai guasti dei sistemi distribuiti [29].

Si può notare, guardando la Figura 2.9, che ROS2 contiene il *livello di astrazione DDS* perché non è necessario che gli utenti siano a conoscenza delle API DDS e ciò gli consente di avere configurazioni di alto livello ottimizzando l'utilizzo del protocollo di rete.

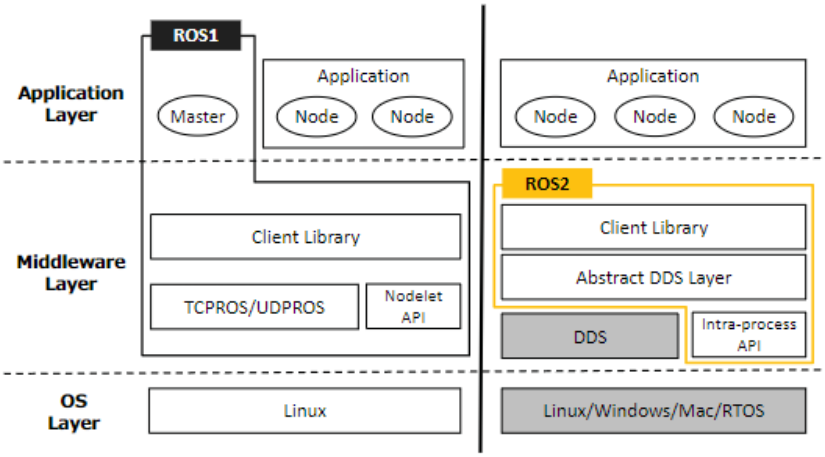

Figura 2.9. Differenze ROS e ROS2 (Fonte: intelligenzaartificialeitalia.net)

I concetti fondamentali di ROS2 sono nodi, messaggi, topic e servizi: l'elaborazione avviene nei nodi che comunicano tra di loro scambiandosi messaggi in modo asincrono, che vengono pubblicati su dei topic a cui i nodi possono iscriversi per leggere il contenuto. Il software è organizzato in *package* che possono contenere nodi, librerie o file di configurazione e che funzionano come dei moduli che permettono di riutilizzare il codice. Infatti, per poter

utilizzare ROS2 si devono scaricare e installare i suoi pacchetti dalla documentazione ufficiale della distribuzione scelta, come sarà trattato in seguito.

I nodi sono processi che eseguono qualsiasi attività di calcolo all'interno del sistema; spesso un sistema basato su ROS contiene molti nodi, a causa della sua struttura modulare. Quindi, ogni nodo è visto come modulo software e si occupa di una parte diversa del complesso in modo da distribuire il carico computazionale con il vantaggio di ottenere un'alta tolleranza ai guasti. Come si è detto prima, quando molti nodi sono in esecuzione, è conveniente rappresentare le comunicazioni peer-to-peer tra processi con una struttura a grafo, con i processi rappresentati come nodi del grafo e con i collegamenti peer-to-peer come archi.

I messaggi sono strutture dati tipizzate che supportano i tipi primitivi standard, come ad esempio integer, boolean, string, o array dei tipi primitivi. Hanno una struttura a lista costituita da campi e costanti che si riferiscono rispettivamente ai dati inviati all'interno del messaggio e ai valori numerici che consentono di decifrare il significato dei campi.

I topic rappresentano il canale di comunicazione attraverso cui i nodi si scambiano messaggi, secondo il modello *publish/subscribe*: un nodo può pubblicare i messaggi su un topic specifico, inviando dati e informazioni in modo asincrono; quindi, senza preoccuparsi di chi leggerà il messaggio, oppure può sottoscriversi ad un topic per leggere e interpretare i messaggi ricevuti. È possibile che ci siano più nodi, *publisher* o *subscriber,* legati ad un topic oppure che un nodo pubblichi o si iscriva a diversi topic. Ovviamente anche i topic sono tipizzati e possono contenere un solo tipo di messaggio, sia in lettura che in scrittura [30].

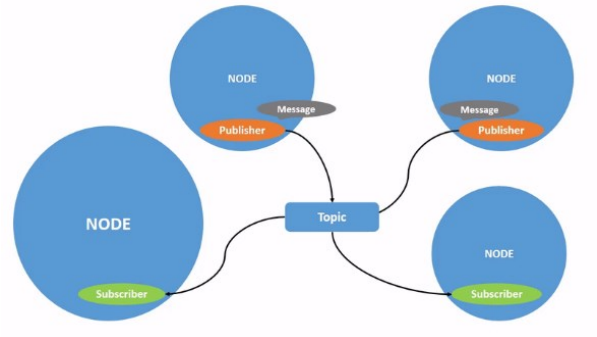

Figura 2.10. Scambio di messaggi tra nodi (A)

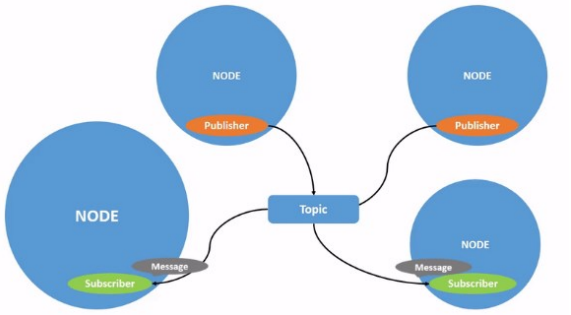

Figura 2.10. Scambio di messaggi tra nodi (B)

(Fonte: https://docs.ros.org/en/foxy/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Topics/Understanding-ROS2-Topics.html)

La figura 2.10 mostra, in modo semplificato, il percorso dei messaggi: vengono pubblicati sul topic dai nodi publisher (A) e arrivano ai nodi subscriber a partire dallo stesso topic (B).

I servizi di ROS rappresentano il canale di comunicazione sincrono secondo il modello *request/reply*: un nodo invia la richiesta ad un altro nodo che la elabora e trasmette una risposta; tale procedura è analoga alla Remote Procedure Call.

2.1.6. MQTT

Figura 2.11. Logo di MQTT (Fonte: inDomus.it)

Message Queuing Telemetry Transport (MQTT) è un protocollo sempre più utilizzato per la comunicazione all'interno dell'IoT. È stato sviluppato da IBM nel 1999 come metodo di comunicazione leggero da macchina a macchina. È un protocollo di messaggistica che utilizza

il modello di trasmissione di pubblicazione-sottoscrizione. Un'architettura MQTT ha due componenti principali:

- *Client MQTT*: è qualsiasi dispositivo (ad esempio un computer o un telefono cellulare) che si connette al broker. Come per il ROS, un client che invia messaggi è un publisher e un client che riceve messaggi è un subscriber.
- *Broker MQTT*: è il centro di controllo, instaura le connessioni con i vari client che si collegano alla rete, archivia, distribuisce e coordina tutti i messaggi.

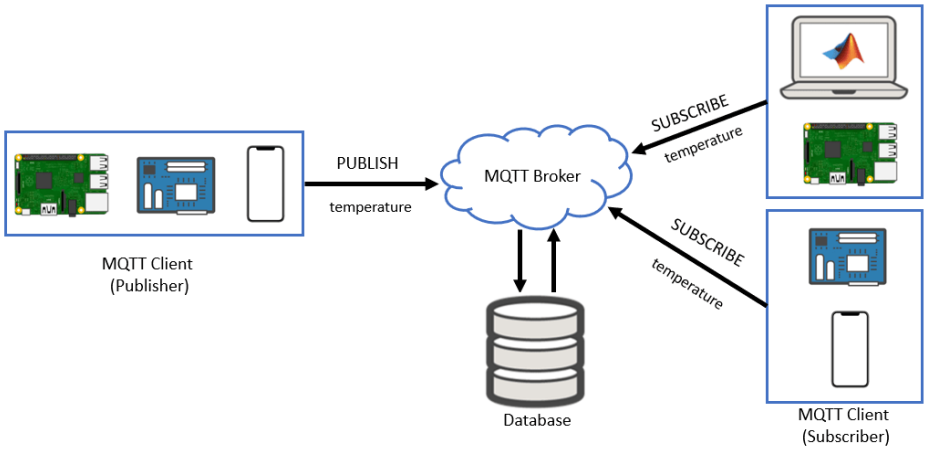

Figura 2.12. Architettura MQTT (Fonte: mathworks.com)

Anche MQTT è un sistema che funziona in base ai topic: un client pubblica un messaggio in un topic specificato dal client stesso, l'archiviazione dei messaggi viene effettuata dal broker che crea un'istanza per ogni topic e la redistribuzione si attua ai client che si sono iscritti al topic nel quale è stato pubblicato il messaggio. Un topic è una stringa case-sensitive, con almeno un carattere che può essere costituita da uno o più livelli di topic generando così una struttura ad albero. Ogni livello è separato da una barra (/) [31].

Il fatto che ci siano più client contemporaneamente contribuisce alla comunicazione asincrona, in cui le sottoscrizioni o le pubblicazioni avvengono parallelamente. Inoltre, i client stessi non richiedono aggiornamenti, determinando così una riduzione delle risorse utilizzate, il che rende questo modello ottimale per l'impiego in un ambiente con larghezza di banda ridotta.

MQTT utilizza diversi modelli per lo scambio dei messaggi e sono chiamati *Quality of Service* (QoS):

- QoS0 (Figura 2.13): è la modalità di trasferimento più rapida e viene utilizzata per inviare i dati in tempo quasi reale. Tuttavia, il messaggio viene consegnato al massimo una volta o potrebbe non essere consegnato affatto, infatti potrebbe essere perso se il client è disconnesso o se il server fallisce. Se il client non è connesso nel momento in cui il server riceve la pubblicazione, questa potrebbe essere eliminata, a seconda dell'implementazione del server. La consegna nella rete non viene riconosciuta e il messaggio non viene memorizzato.

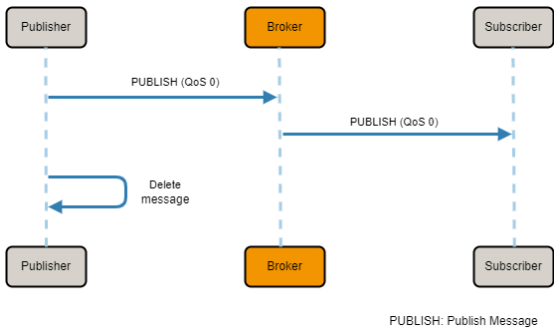

Figura 2.13. QoS0 (Fonte: mathworks.com)

- QoS1 (Figura 2.14): il messaggio viene sempre recapitato almeno una volta. Se si verifica un errore prima che un riconoscimento venga ricevuto dal destinatario, un

messaggio può essere consegnato più volte. Il messaggio viene memorizzato localmente al mittente fino a quando questi non ha la conferma dell'avvenuta ricezione da parte del destinatario, nel caso in cui debba essere inviato nuovamente.

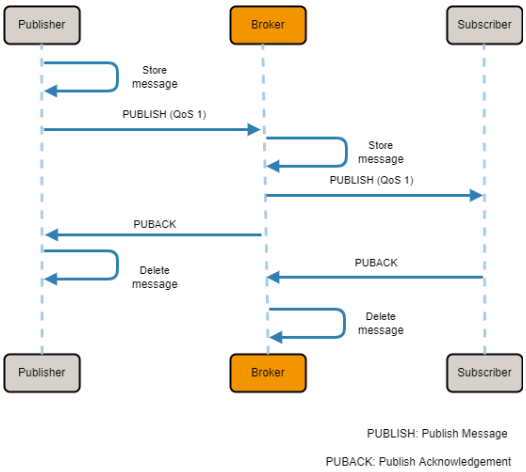

Figura 2.14. QoS1 (Fonte: mathworks.com)

- QoS2 (Figura 2.15): fornisce la consegna dei messaggi più affidabile, ma è anche la modalità di trasferimento più lenta. Il messaggio viene sempre consegnato esattamente una volta e deve essere memorizzato anche localmente al mittente fino a quando lo stesso non riceve la conferma che il messaggio è stato ricevuto dal destinatario. Il messaggio viene memorizzato nel caso in cui debba essere inviato nuovamente. È utile in caso di invio di comandi, in quanto si può avere la conferma che il solo comando specificato venga eseguito una sola volta. [31]

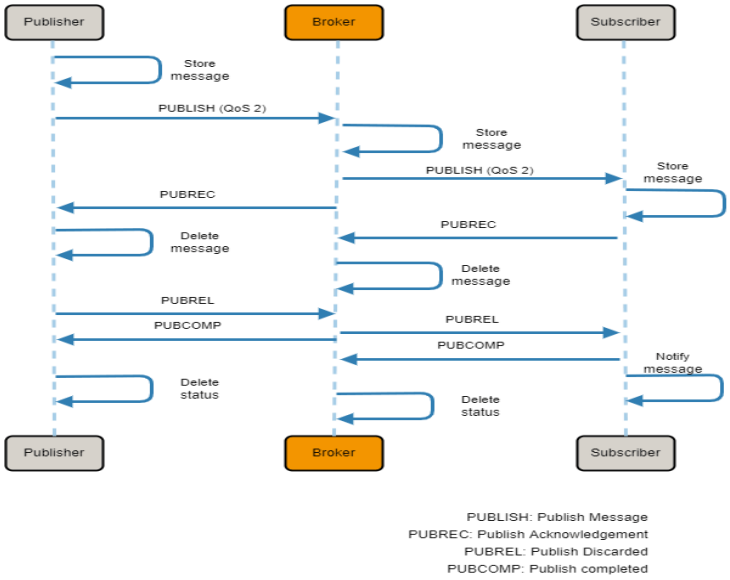

Figura 2.15. QoS2 (Fonte: mathworks.com)

In questo progetto gli MQTT Client sono M5StickCPlus e MATLAB, mentre come MQTT Broker si è utilizzato Eclipse Mosquitto. Quest'ultimo è leggero ed adatto per l'uso su tutti i dispositivi, dai computer a scheda singola a basso consumo ai server completi. È un broker di messaggi open source e multipiattaforma (licenza EPL/EDL) che implementa le versioni 5.0, 3.1.1 e 3.1 del protocollo MQTT. Si è utilizzata la versione 3.1.1. Il progetto Mosquitto fornisce anche una libreria C per l'implementazione dei client MQTT e i client MQTT a riga di comando *mosquitto_pub* e *mosquitto_sub*. [32]

Figura 2.16. Logo di Mosquitto (Fonte: hub.docker.com)

ROS e MQTT sono stati installati su una macchina virtuale con la distribuzione Ubuntu 2.1.8 di Linux tramite VirtualBox, un software di virtualizzazione completo open source per uso generale per hardware x86_64, destinato all'uso con laptop, desktop, server ed embedded. Creato dalla Oracle Corporation, consente di simulare un ambiente hardware completo, fornendo risorse di calcolo come CPU, memoria RAM e dischi virtuali che sono emulati a livello software. Ubuntu è un sistema operativo basato su Debian GNU General Public License, quindi chiunque può accedere al codice, modificarlo e distribuirlo. Sviluppato negli anni 90, è diventato una delle distribuzioni di Linux più usate per la stabilità e la sicurezza che offre. Si è utilizzato Ubuntu rispetto alle altre versioni perché presenta diverse caratteristiche vantaggiose, tra cui:

- Semplifica l'interfaccia utente in modo da essere utile anche per dispositivi con schermi più piccoli; implementa l'interfaccia grafica GNOME offrendo un ambiente desktop moderno e facile da navigare.
- Semplifica l'installazione rendendo automatiche molte decisioni, basandosi sulle scelte della maggior parte degli utenti; inoltre, si ha il supporto di una community attiva di utenti e sviluppatori che forniscono risorse, consigli e guide per facilitare l'utilizzo del sistema operativo.
- Cambia gli strumenti di gestione del software utilizzando APT (Advanced Package Tool) per installare, rimuovere o aggiornare i pacchetti; è un ambiente flessibile per cui ogni utente può modificare a proprio piacimento temi, icone o scegliere estensioni.

- Blocca l'account utente root tramite il comando *sudo*. [33]

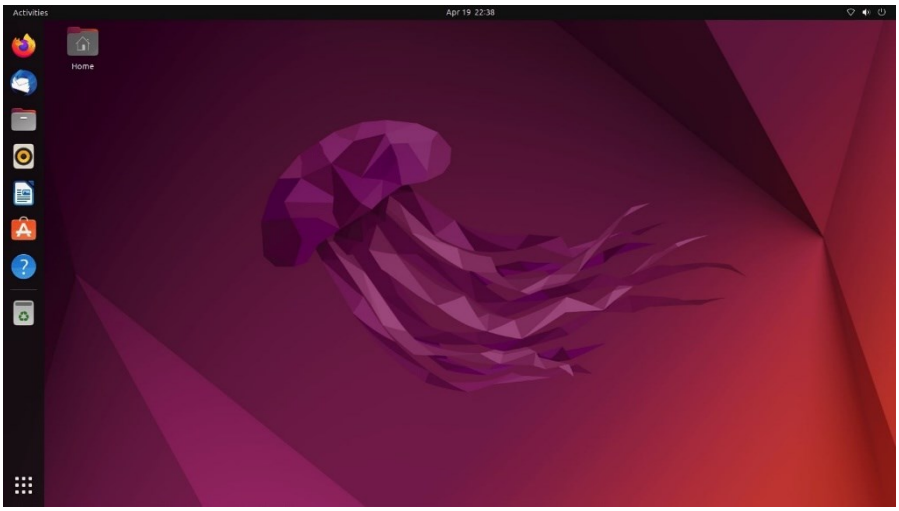

Figura 2.17. Interfaccia di Ubuntu (Fonte: html.it)

2.1.7. MATLAB

Per la gestione dell'*alto livello*, si è utilizzato MATLAB r2023b perché consente di occuparsi del sistema simulato, inviandogli direttamente i comandi, senza avere la necessità di interagire con l'hardware. MATLAB (MATrix LABoratory) è una piattaforma di programmazione e calcolo numerico utilizzata da milioni di ingegneri e scienziati per l'analisi e la visualizzazione di dati, lo sviluppo di algoritmi e applicazioni, la creazione di modelli e simulazioni [34]. È basato su un linguaggio di programmazione ad alto livello che consente di esprimere il sistema in termini di matrici e array che entrambi non richiedono dimensionamento, dotato di istruzioni di controllo del flusso, funzioni, strutture dati, input/output e funzionalità tipiche della programmazione orientata a oggetti. Combina insieme calcolo, visualizzazione e programmazione in un ambiente di facile utilizzo che permette la creazione sia di piccoli programmi più semplici che di programmi applicativi più grandi e complessi. MATLAB dispone di toolbox sviluppati professionalmente, rigorosamente testati e interamente

documentati, che consentono di apprendere e applicare tecnologie specializzate. I toolbox sono raccolte complete di funzioni MATLAB che permettono di lavorare in diversi settori. [35]

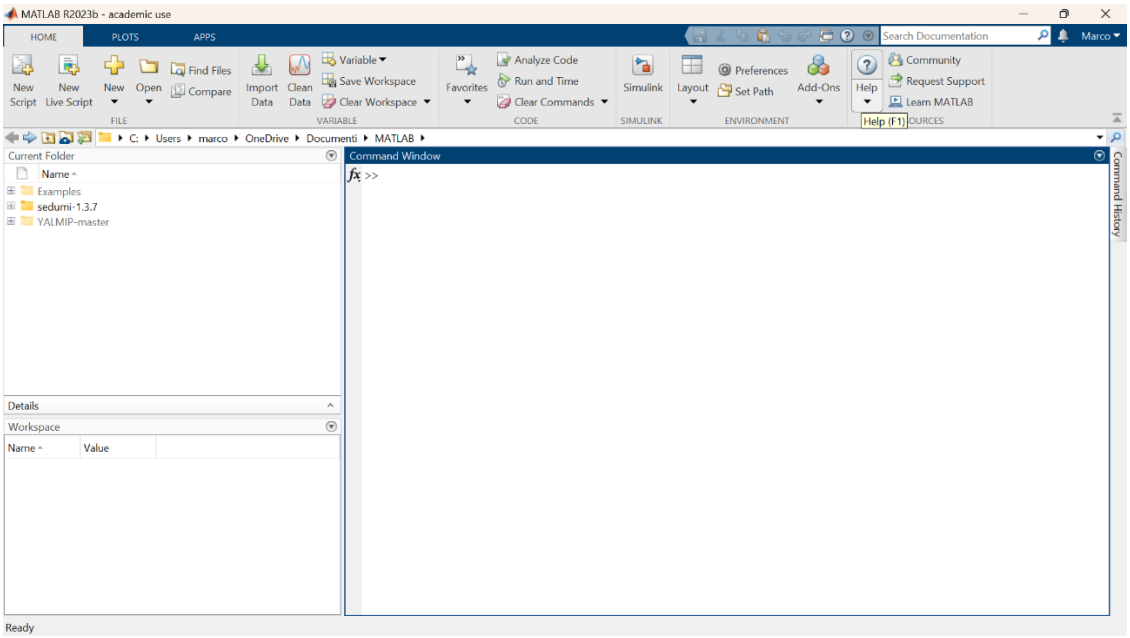

Figura 2.18. Interfaccia di MATLAB

2.2. Metodi

In questa tesi si sono analizzati tre esempi principali per testare la connessione e la comunicazione tra basso e alto livello:

- controllo del LED di M5StickCPlus;
- controllo dei pulsanti di M5StickCPlus;
- controllo del movimento di un servo connesso all'M5StickCPlus.

I nodi utilizzati per il LED, per il pulsante e per il servo e i topic associati sono:

- per il LED, si è generato il nodo */led_control* che si lega al topic */accensione*;

- per il pulsante, si è sviluppato il nodo */but_node* che si connette al topic /*m5stickc/button*;
- per il servo, si è realizzato il nodo */servo_node* che pubblica i messaggi sul topic */servo_angle.*

2.2.1. LED

Con il primo esempio si vuole studiare la comunicazione unidirezionale dall'alto livello, quindi utilizzando MATLAB, al basso livello, rappresentato da M5StickCPlus insieme ad Arduino IDE. L'obiettivo che si vuole raggiungere è che il microcontrollore accenda il suo LED integrato dopo aver ricevuto il messaggio di accensione inviato da MATLAB. È un modo molto semplice di esaminare la corretta comunicazione tra due sistemi distinti in diversi livelli di astrazione perché permette di monitorare visivamente il flusso delle informazioni. Inoltre, è un esempio di controllo remoto da parte di un sistema esterno come MATLAB e di interazione in tempo reale con l'M5StickCPlus.

Andando più nel dettaglio, MATLAB invia un comando in tempo reale attraverso un nodo ROS del tipo *publisher* che contiene il messaggio di accensione o spegnimento del LED; M5StickCPlus si comporta da nodo MQTT del tipo *subscriber* in modo da ricevere il comando e interpretarlo di fatto con l'azione diretta sul LED. In sintesi, MATLAB comunica attraverso ROS2; ROS2 e MQTT si scambiano messaggi tra di loro attraverso il ponte bash nella macchina virtuale e il microcontrollore si connette a MQTT. Quindi, in MATLAB si crea il nodo ROS2 */led_control* tramite la funzione `ros2node` e si definisce il topic */accensione*. Per semplicità e velocità di scrittura le due strutture possono essere assegnate a delle variabili. All'interno del nodo, si genera il publisher *ledPub* tramite la funzione `ros2publisher` che pubblica il

messaggio nel topic, infatti tra i parametri della funzione si indicano il nodo, il topic e il tipo di messaggio. In questo caso, si utilizza il messaggio di tipo *String* poiché il publisher può inviare "accendi" o "spegni" per controllare il LED. Il messaggio è una struttura dati creata con la funzione `ros2message`, indicando come parametro il mittente, e il suo testo deve essere inserito all'interno del campo *data* del messaggio. Per pubblicare l'informazione su ROS2 si utilizza la funzione `send` che prende in input il publisher e il messaggio.

Si stabilisce la comunicazione tra ROS2 e MQTT pubblicando il messaggio al broker MQTT utilizzando il comando `mosquitto_pub`, incluso nella libreria di Mosquitto, che permette di inviare messaggi ad un topic specificato nel broker. Si esplicitano i vari termini interessati allo scambio del messaggio: l'indirizzo del broker MQTT, la porta del broker, il messaggio ed il topic in cui è stato pubblicato. L'ascolto dei messaggi dal topic specifico di ROS2, `$ROS2_TOPIC_SUB`, avviene tramite il comando `ros2 topic echo`. Di seguito, si imposta il profilo QoS per la qualità del servizio per i messaggi `sensor_data`, uno dei profili standard definiti da ROS2, secondo cui è più importante ricevere i messaggi in modo tempestivo piuttosto che assicurarsi che arrivino tutti. È configurato per garantire bassa latenza e un'alta affidabilità in modo da inviare continuamente i dati più recenti appena catturati, anche al costo di perderne qualcuno, con il vantaggio di avere il carico sulla rete ridotto evitando così l'accumulo di messaggi obsoleti. Lo script è configurato affinché i messaggi vengano letti riga per riga tramite il ciclo while e vengano elaborati solo quelli che hanno il suffisso `data:`. La sintassi `${line#data: }` fa in modo che venga estratto solo il testo del messaggio per poterlo stampare e visualizzare. Dopodiché, viene pubblicato il testo su MQTT che trasmette la risposta come conferma dell'accensione del LED, cioè, inviando il messaggio "acceso", se viene soddisfatta la condizione di controllo sulla sintassi del messaggio (deve coincidere con il dato "accendi" definito in MATLAB). Ovviamente, il processo dipende dall'informazione

inviata da MATLAB perché si può compilare anche con `messageToSend = 'spegni'` e quindi generare la risposta “spento”. Lo script Arduino gestiste l’hardware pertanto, come conseguenza dell’arrivo del messaggio su MQTT, fa in modo che l’M5StickC accenda o spenga il LED. Si implementano la `void setup()` e la `void loop()` dell’IDE. La prima funzione rappresenta l’inizializzazione della comunicazione, per cui si configurano l’M5 ed il Wi-Fi; la funzione `setup_wifi()` configura la connessione tramite il comando `WiFi.begin(ssid, password)` che legge in input il nome dell’hotspot e la password associata. Vengono stampati una serie di messaggi in un ciclo while che indicano i tentativi di collegamento al Wi-Fi, fino a che non si ottiene la conferma dell’effettiva connessione. Il LED viene definito in modo che sia spento inizialmente, tramite `digitalWrite(ledPin, LOW)`. Con `client.setServer(mqtt_server, mqtt_port)` e `client.setCallback(callback)` si indica l’indirizzo IP e la porta del broker MQTT al client MQTT per inizializzarlo e si imposta una funzione di *callback* che verrà chiamata ogni volta che il client riceve un messaggio su un topic a cui è sottoscritto. Questa funzione gestisce i messaggi MQTT al loro arrivo. Viene chiamata ogni volta che il client riceve un messaggio al topic a cui è sottoscritto, cioè ogni volta che si avvia lo script di MATLAB. La funzione richiede in input dei parametri particolari della libreria MQTT, tra cui il topic dove è stata ricevuta l’informazione, il puntatore ai dati del messaggio, il *payload*, che rappresenta il contenuto trasmesso, e la sua lunghezza. Viene poi creata una stringa riferita al messaggio in modo tale da ottenerne una rappresentazione testuale trasformando ogni byte del payload in un carattere. Viene stampato il messaggio come conferma che la conversione sia avvenuta in modo esatto e dopodiché si controlla che il messaggio sia quello desiderato. Si accerta che il topic della *callback* corrisponda al topic creato da MATLAB */accensione* in modo da passare al controllo fisico del LED di M5StickCPlus: se il messaggio ricevuto è uguale alla stringa “accendi” allora

si procede con l'accensione del LED, altrimenti se il messaggio è "spegni" il LED resta o viene spento. Esiste la funzione `void reconnect()` che permette il legame con il broker MQTT: è un ciclo continuo che stampa messaggi che indicano che il programma sta tentando di connettersi a MQTT finché non si ottiene la conferma quando si è identificato il client ID "M5StickCPlus", in caso contrario si genera un errore di connessione insieme al codice dello stato del cliente. Se si ottiene un esito positivo (viene stampato il messaggio "connected" sul *Serial Monitor* di Arduino), avviene la sottoscrizione al topic denominato */accensione* in modo che il client così possa ricevere i messaggi pubblicati e possa inviare risposte. La `void loop` permette di eseguire in modo continuo le azioni e per questo è relativa alla connessione con MQTT, scopo principale dello script. Controlla se il client MQTT è collegato virtualmente al broker MQTT: nel caso in cui la condizione non sia soddisfatta sarà utilizzata la funzione definita precedentemente `void reconnect()`. Il corpo del loop è rappresentato da `client.loop()` che gestisce la comunicazione MQTT: infatti, mantiene la connessione attiva, ascolta e riceve i nuovi messaggi ricevuti controllando anche i possibili vari topic a cui il client è sottoscritto e garantendo che riceva tutte le informazioni necessarie; si può anche occupare dell'invio di messaggi di risposta, se sono previsti. Inoltre, è una funzione del tipo non bloccante che non ferma il programma per cui si possono eseguire altre istruzioni mentre è aperta la comunicazione con MQTT.

2.2.2. Pulsanti

Il secondo esempio rappresenta la procedura opposta rispetto a quella del LED: viene trattata la comunicazione dal basso livello, quindi dal dispositivo embedded M5StickCPlus con Arduino IDE, al software di controllo ad alto livello MATLAB. Lo scopo di questo esempio è

verificare che l'azione fisica da parte di un utente, che consiste nel premere uno dei due pulsanti integrati di M5StickCPlus, generi un messaggio su MATLAB che indichi quale tasto è stato selezionato. Pertanto, si vuole dimostrare che il sistema è capace di raccogliere input dalla scheda di sviluppo, i quali vengono pubblicati su MQTT che dialoga con ROS2 in modo che i dati vengano trasmessi a MATLAB. In generale, l'esempio rappresenta l'interazione tra un dispositivo hardware e un sistema ad alto livello esterno che genera un feedback visivo e permette di elaborare dati. Si tratta di una comunicazione bidirezionale perché M5StickCPlus invia il messaggio a MATLAB e quest'ultimo risponde con un altro messaggio. In questo caso M5StickCPlus si comporta da nodo MQTT del tipo *publisher* perché reagisce alla pressione di un suo tasto pubblicando un messaggio che indica che il tasto A o il tasto B è stato schiacciato. MATLAB ha la funzione di nodo del tipo *subscriber* perché riceve il messaggio e risponde semplicemente stampando una frase per verificare la correttezza del procedimento. In sintesi, M5StickCPlus pubblica su MQTT, MQTT e ROS2 comunicano tra di loro tramite lo script bash, eseguibile su Ubuntu, e M5StickCPlus legge il messaggio su ROS2. Viene creato il nodo */but_node* e, poiché la comunicazione inizia dall'hardware, all'interno della funzione `void loop()` di Arduino IDE viene creato e pubblicato il messaggio tramite il `client.loop()`. Esistono due comandi che verificano la condizione che indica se è stato premuto il tasto A o il tasto B e di conseguenza, se l'if è soddisfatto, generano il messaggio tramite il comando `client.publish("/m5stickc/button", "Button A pressed")`, dove sono indicati rispettivamente il nome del topic su cui pubblicare ed il contenuto del messaggio inviato, che nel secondo caso è `"Button B pressed"`. Ovviamente, tramite il `void setup()`, si devono prima configurare M5, Wi-Fi e la connessione con MQTT tramite un ciclo di messaggi che indicano i tentativi di connessione finché, come nel caso precedente, non si riconosce il client ID "M5StickCPlus". Per sicurezza, è meglio inserire nel loop il comando

`client.connect("M5StickCPlus")` all'interno di un while per assicurarsi di mantenere la connessione. Si stabilisce la comunicazione tra MQTT e ROS2 tramite lo script bash che permette di ascoltare i messaggi di MQTT e pubblicarli su ROS2 in modo che MATLAB possa riceverli. Questa volta si utilizza il comando `mosquitto_sub`, il client MQTT incluso nella liberia di Mosquitto, per ascoltare il messaggio, per cui si indicano l'indirizzo IP e la porta utilizzati dal broker ed il topic a cui bisogna sottoscriversi. Dopodiché, tramite `ros2 topic pub --once`, si invia il messaggio a ROS2 indicando il topic in cui pubblicare il messaggio, coincidente con il topic di MQTT, e il suo contenuto. A differenza del caso del LED, in MATLAB all'interno del nodo viene creato il subscriber *sub,* tramite il comando `ros2subscriber`, per leggere il messaggio mandato da MQTT. In questo caso, la funzione di callback è inclusa tra gli argomenti del comando in MATLAB insieme al nodo e al topic interessato, poiché è questo software che deve agire di conseguenza ai messaggi. La callback viene chiamata ogni volta che viene premuto uno dei due tasti di M5StickCPlus e fa in modo che, dopo aver controllato il contenuto del messaggio, stampi il messaggio "Tasto centrale premuto" per il pulsante A oppure "Tasto a destra premuto" per il B.

2.2.3. Servo

Il terzo esempio si focalizza nella comunicazione dall'alto livello al basso livello, come nel caso del LED, ma si pone l'obiettivo sul controllo di un attuatore fisico, cioè il servomotore, collegato al dispositivo hardware M5StickCPlus. MATLAB è in grado di inviare un messaggio che contenga l'angolo di posizione del servo in modo che venga convertito in un segnale fisico e quest'ultimo ruoti fino alla posizione trasmessa dal software. Si vuole studiare l'integrazione tra hardware e software ed il controllo remoto di un dispositivo fisico. È un caso che può essere

applicato in diversi contesti attuali del mondo della robotica o dell'automazione industriale. In sintesi, MATLAB comunica con ROS2; ROS2 e MQTT si connettono tra di loro attraverso il ponte bash nella macchina virtuale e il microcontrollore, connesso al servomotore tramite il modulo driver, legge i messaggi su MQTT. Quindi, MATLAB crea il nodo ROS2 */servo_node* e al suo interno definisce il publisher *pub_angle* tramite la funzione `ros2publisher` in cui si indicano il nodo, il topic associato e il tipo del messaggio scelto che, anche se riguarda la posizione, è del tipo *String* perché più facile da trattare. Il publisher crea il messaggio, come testo si inserisce la posizione desiderata e lo pubblica all'interno del topic di ROS2 specificato. ROS2 deve comunicare con MQTT, per cui si ascoltano i messaggi inviati su ROS2 tramite lo stesso comando utilizzato nell'esempio del LED, `ros2 topic echo`, servendosi della stesso tipo di qualità di servizio. Si estrae il contenuto del messaggio con la sintassi già vista per il LED, effettuando però un ulteriore controllo in quanto il servo accetta solo angoli che vanno da 0° a 180°; quindi si verifica che il messaggio sia composto solo da cifre e che sia un numero compreso nell'intervallo adoperato dal servomotore. Se queste condizioni non sono soddisfatte il sistema genera un errore; invece, se tutti i controlli vengono superati, il messaggio viene pubblicato su MQTT con l'utilizzo di `mosquitto_pub`. Arduino IDE si occupa della gestione hardware, fa muovere il servo come conseguenza del messaggio ricevuto; infatti, la `void loop()` contiene la funzione di *callback* come nell'esempio del LED: viene creata una stringa riferita al messaggio trasformando ogni byte del payload in un carattere, e in più il testo viene convertito in un numero intero, tramite `angleStr.toInt()`, necessario per definire il movimento del servo e la rispettiva angolazione. Infine, si fa muovere il servo con il comando `drive.setServoAngle(1, angle)`, in cui si indicano come argomenti il canale del modulo driver e l'angolo desiderato, dopo un ulteriore controllo sull'intervallo degli angoli.

MQTT è un metodo di comunicazione efficiente ed è stato anche dimostrato da uno studio condotto da Matthias Pohl, Janick Kubela, Sascha Bosse e Klaus Turowski [36], nel quale sono stati presi in considerazione diversi protocolli IoT da confrontare, tra cui MQTT, Advanced Message Queuing Protocol (AMQP) ed Extensible Messaging and Presence Protocol (XMPP). AMQP è noto per la sua interoperabilità, anch'esso si basa su un modello di pubblicazione-sottoscrizione con l'estensione dei messaggi che possono essere accodati al broker; pertanto, essi possono avere priorità in base a delle condizioni dichiarate precedentemente. XMPP, invece, non è dotato di un broker centrale, ma esistono più server; inoltre, la comunicazione può essere implementata anche con il modello richiesta/risposta utilizzando HTTP come protocollo di trasporto al posto di TCP. Secondo Ammar Rayes e Samer Salam, due ingegneri informatici esperti, autori del libro *Internet of Things from Hype to Reality,* il concetto di Richiesta/Risposta è adatto per la comunicazione interattiva; al contrario, Publish/Subscribe è un modello di comunicazione unidirezionale che può stabilirsi tra un publisher e più subscriber ed è adatto per applicazioni che dipendono da accoppiamento lasco tra i dispositivi, cioè che non sono rigidamente fissati l'uno all'altro. L'IoT utilizza l'infrastruttura di Internet per consentire la connettività; tuttavia, i protocolli di comunicazione utilizzati da Internet non sono adatti ai requisiti dell'IoT perché i dispositivi associati sono spesso limitati per quanto riguarda la loro potenza computazionale ed energia disponibile. Per esempio, HTTP fornisce dati in un formato leggibile dall'utente, che determina un sovraccarico del protocollo non utile per la comunicazione M2M tipica di MQTT. Pertanto, sono necessari dei protocolli leggeri ed efficienti a livello di applicazione per gestire la comunicazione tra dispositivi IoT; di conseguenza, i protocolli esistenti sono stati adeguati e ne sono stati progettati di nuovi per soddisfare i requisiti IoT. Le caratteristiche rilevanti dei protocolli IoT sono tempo di latenza, throughput, l'utilizzo della larghezza di banda, l'affidabilità e il consumo energetico. Il tempo

di trasferimento o di latenza è definito come il tempo necessario per inviare un messaggio nel formato del protocollo particolare dal broker al cliente e per decrittografarlo. Il *round trip* di un messaggio inviato è il viaggio di andata e ritorno dal broker al cliente e viceversa, definito senza includere la decodifica e la crittografia del messaggio stesso. Pertanto, il numero medio di round trip eseguito in un secondo consente un confronto del throughput dei protocolli IoT, che corrisponde alla quantità dei dati trasmessi in un'unità di tempo. Per misurare la larghezza di banda di un protocollo IoT, si confronta l'efficienza del protocollo, la quale è definita come il rapporto tra i pacchetti di rete inviati per trasferire un messaggio e le informazioni contenute in esso. Il numero identificato di byte aggiuntivi trasmessi attraverso la rete permette di stabilire la larghezza di banda utilizzata da un protocollo. La relazione tra i messaggi inviati dal broker e quelli ricevuti dal client è un indicatore dell'affidabilità del protocollo [36]. Poiché gli ambienti IoT del mondo reale sono interessati da vari fattori, come ad esempio la perdita di connessione di rete, si considerano come parametri il payload del messaggio, il *Packet-Loss-Rate* (PLR), percentuale di pacchetti di dati che non riescono a raggiungere la loro destinazione in una rete di comunicazione, e la latenza della rete. Grazie agli studi, è stato identificato che il PLR di rete ha un'influenza sulla latenza e sull'affidabilità di un protocollo IoT [37]. Inoltre, è stato rilevato che la latenza e il PLR di una rete influenzano il comportamento di un protocollo IoT [38] ed è stato riconosciuto che il payload del messaggio ha un effetto sulle caratteristiche menzionate precedentemente. Per l'utilizzo della larghezza di banda del file, MQTT mostra la migliore efficienza tra i tre protocolli considerati. Le differenze possono essere spiegate riguardo al modo in cui un messaggio viene pubblicato dallo specifico protocollo. MQTT invia un solo pacchetto per pubblicare un messaggio che contiene le proprietà del messaggio come l'argomento, il topic dove dovrebbe essere pubblicato e il payload in un formato non strutturato. Al contrario, AMQP invia tre pacchetti separati per pubblicare un messaggio: *Basic.Publish*,

con le informazioni sulla pubblicazione; *Content-Header*, con i dettagli del messaggio, e *Content-Body,* contenente il payload. Anche XMPP utilizza un pacchetto per pubblicare un messaggio, ma il sovraccarico è dovuto al fatto che formatta il messaggio utilizzando XML.

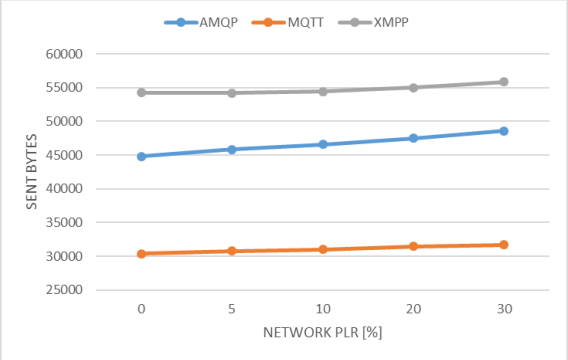

Figura 2.19. Influenza del PLR di rete sui byte inviati (Fonte: [36])

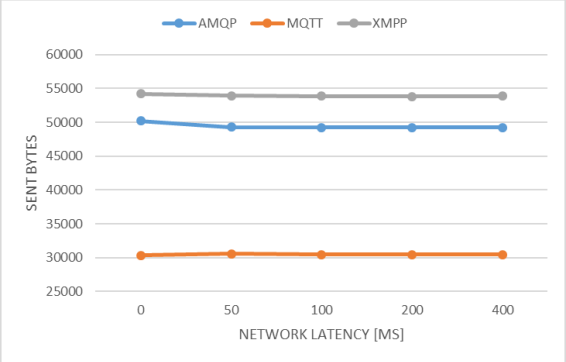

Figura 2.20. Influenza della latenza di rete sui byte inviati (Fonte: [36])

La Figura 2.19 e la Figura 2.20 mostrano rispettivamente l'influenza del PLR e della latenza di rete sui byte inviati in ognuno dei tre protocolli. Si può notare che effettivamente MQTT è il migliore in termini di utilizzo di larghezza di banda poiché, nonostante l'aumento del tasso di perdita dei pacchetti, usa in modo costante meno byte, quindi una banda minore,. Inoltre, MQTT è anche il più stabile e non è condizionato dai ritardi di rete.

In generale, il tempo di trasferimento di tutti i protocolli aumenta insieme al payload del messaggio. Dalla Figura 2.21, si può notare che la latenza di MQTT e di AMQP cresce più lentamente di quella di XMPP perché, come si è detto in precedenza, quest'ultimo utilizza XML per strutturare il messaggio; invece, AMQP e MQTT inviano testi semplici che richiedono meno tempo per essere crittografati e decrittografati, producendo un minore sovraccarico del messaggio. In secondo luogo, i broker di AMQP e MQTT si limitano ad inoltrare ai client i messaggi che ricevono senza inviare una conferma o eseguire una trasformazione del formato. Questo spiega il fatto che, se i pulsanti dell'M5StickCPlus vengono premuti troppo rapidamente e la rete ha un tempo di trasferimento più lungo a causa del carico o di problemi di connessione, alcuni messaggi potrebbero subire un tempo di latenza o addirittura essere scartati.

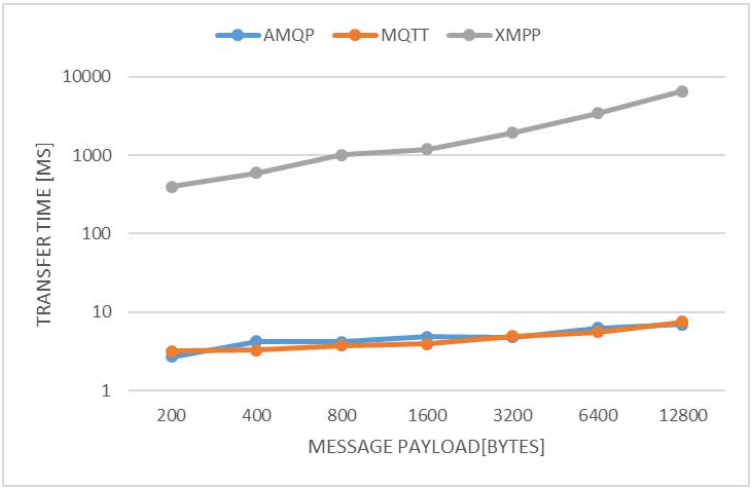

Figura 2.21. Influenza del payload del messaggio sul tempo di latenza (Fonte: [36])

La Figura 2.22 mostra il tempo di trasferimento medio dei tre protocolli in relazione al PLR di rete: mentre il tempo di latenza di XMPP è costante, quello di AMQP e MQTT aumenta sincronicamente. Ciò è collegato al fatto che la latenza di XMPP è già elevata per cui il processo di ritrasmissione in caso di perdita dei pacchetti non lo influenza. Al contrario, AMQP e MQTT

hanno un tempo di trasferimento basso, quindi sono molto influenzati dai processi di ritrasmissione.

Infine, per analizzare il throughput dei tre protocolli IoT si guardi la Figura 2.23.

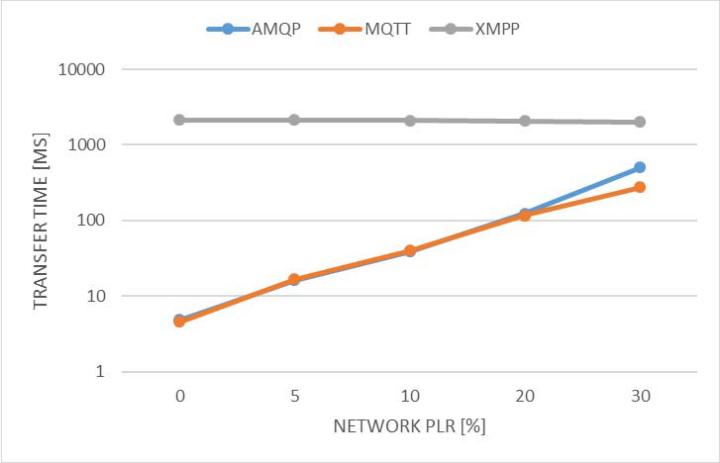

Figura 2.22. Influenza del PLR di rete sul tempo di trasferimento (Fonte: [36])

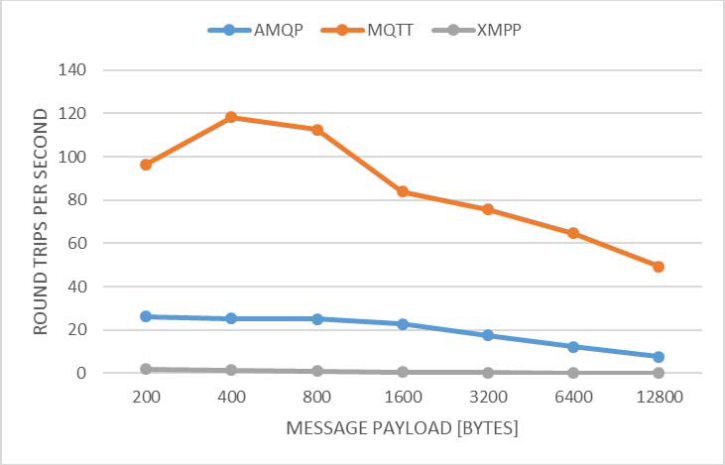

Figura 2.23. Influenza del payload sul throughput (Fonte: [36])

Si può notare che MQTT offre le migliori prestazioni, seguito da AMQP e XMPP. L'elevata differenza è dovuta al sovraccarico del messaggio dei protocolli, come si è detto prima. Tuttavia, aumentando il carico del messaggio, i tre throughput tendono a valori simili a causa dell'elevata influenza del payload del messaggio sul totale dei byte inviati. La Figura 2.24 indica

l'influenza del PLR di rete sulla velocità di throughput dei protocolli esaminati. L'esistenza di una rete PLR ha un forte impatto su AMQP e MQTT. I messaggi inviati dai protocolli sono ridotti al 35% per AMQP e al 22% per MQTT se il PLR di rete è impostato al 5%. Al contrario, il numero di messaggi inviati da XMPP rimangono quasi al 100% fino al PLR fissato al 30%.

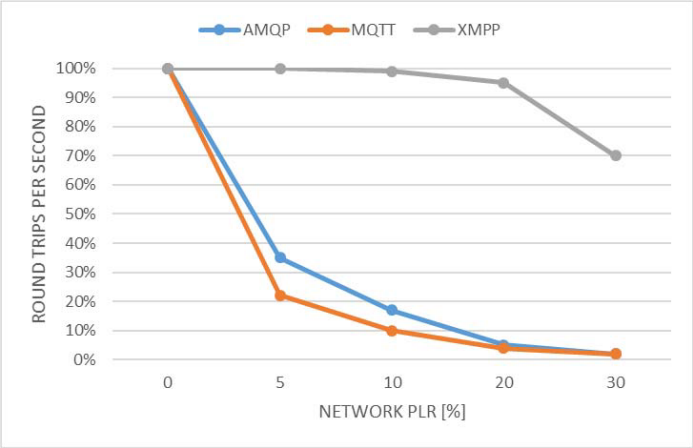

Figura 2.24. Influenza del PLR di rete sul throughput (Fonte: [36])

Per questo progetto, si è utilizzato MQTT poiché, in un contesto in cui il payload non occupa molto spazio e si lavora a bassa larghezza di banda, si è rilevato molto efficiente. In generale, però, lo studio [36] ha dimostrato che MQTT pecca nella sicurezza e nella flessibilità. La prima è importante per applicazioni aziendali che gestiscono dati riservati per ridurre il rischio di manipolazione e intercettazione dei dati. Rispetto al supporto MQTT, AMQP e XMPP hanno dei meccanismi di sicurezza più elevati includendo SASL e SSL/TLS, framework per l'autenticazione e la sicurezza dei dati nei protocolli Internet. Per quanto riguarda la flessibilità, AMQP è progettato come protocollo via cavo, quindi la trasmissione dei dati può essere adattata a requisiti specifici. Inoltre, AMQP e XMPP supportano entrambi i due modelli di richiesta/risposta e di pubblicazione/sottoscrizione, mentre MQTT non consente alcuna strutturazione dei suoi messaggi e supporta solo il secondo modello. In più, XMPP offre una

scalabilità maggiore rispetto ad AMQP e MQTT a causa dell'utilizzo di broker decentralizzati. Al contrario, MQTT offre affidabilità a livello di applicazione supportando il concetto di QoS e supporta le funzionalità di conoscere l'ultima richiesta e di conservare i messaggi. MQTT è il protocollo con il minor consumo energetico, poiché per le sue caratteristiche, il suo utilizzo di risorse di rete è minimo.

A proposito di ROS2, si è già detto che il middleware offre più flessibilità rispetto a ROS poiché contiene i Quality of Service che permettono di controllare la comunicazione a seconda dell'utilizzo; inoltre è anche più affidabile, grazie alla presenza del protocollo DDS [39]. In ogni caso è stato dimostrato da uno studio condotto da Yuya Maruyama, Shinpei Kato e Takuya Azumi [40] che il DDS ha problemi con dati di grandi dimensioni e, nello specifico, FastRTPS, il protocollo utilizzato nel progetto, non supporta tali dati perché è realizzato come un'implementazione leggera per sistemi embedded. ROS2 ha un sovraccarico significativo a seconda della dimensione dei dati, che genera un tempo di latenza sempre più grande, come mostrato nella Figura 2.25. Il sovraccarico è attribuito a due fattori, ovvero ai dati di conversione per DDS e a quelli relativi alla loro elaborazione; in sintesi è necessaria una duplice conversione dei messaggi, cioè da ROS2 a DDS e viceversa: tra le due conversioni, ROS2 chiama le API DDS e passa i messaggi a DDS.

Invece il throughput è limitato dal network e non dal protocollo DDS.

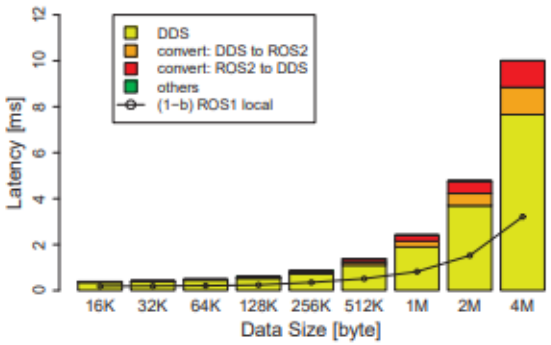

Figura 2.25. Influenza del payload sul tempo di latenza di ROS2 (Fonte: [39])

3. Implementazione e risultati

In questo capitolo, si studia la configurazione fisica e come si sono eseguiti gli script. Nella seconda parte, si vedranno i risultati, in termini di funzionamento e problemi incontrati.

3.1. Implementazione

Per tutti e tre gli esempi, bisogna accendere l'M5StickCPlus premendo il pulsante di accensione posto sul lato sinistro e controllare che sia connesso alla rete Wi-Fi corretta. Il LED e i pulsanti utilizzati sono parte integrante della scheda di sviluppo, per cui non è necessario collegarli fisicamente, ma vengono controllati tramite i pin GPIO oppure attraverso le funzioni della libreria Arduino. L'8Servos HAT v1.1 è connesso fisicamente ad M5StickCPlus: bisogna posizionare il modulo driver nella parte posteriore del dispositivo in modo da inserire tutti i pin disponibili per poter permettere la loro comunicazione come in Figura 2.4 (pag. 8). Il servo SG90 è connesso alla scheda di espansione facendo sì che i cavi di segnale, di alimentazione e di massa del servomotore siano collegati ai pin corrispondenti dell'8Servos (Figura 3.1).

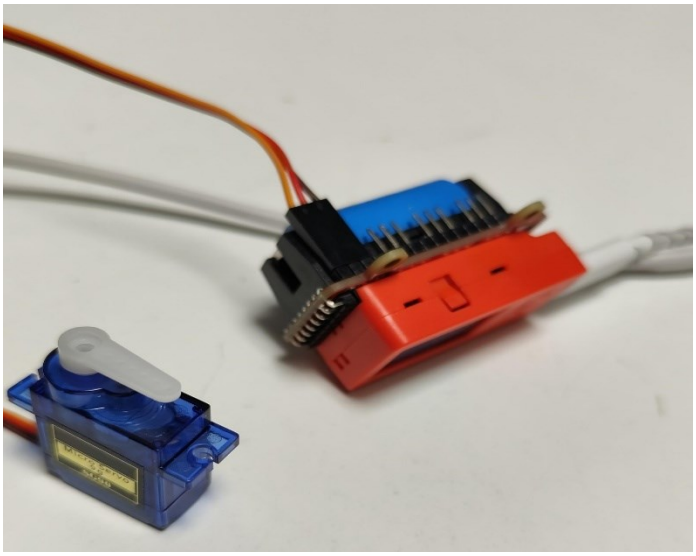

Figura 3.1. Collegamento fisico del servomotore

Dopo aver collegato M5StickCPlus al PC, viene compilato lo script di Arduino, con estensione *.ino*, e, se non sono presenti errori di sintassi, si procede con l'*upload,* cioè si carica il codice dentro la memoria di M5StickCPlus attraverso semplicemente un pulsante dell'interfaccia dell'IDE. Dopo il caricamento, tramite il *Serial Monitor* all'interno degli strumenti, si verificano i messaggi di debug e informazioni sulla connessione per controllare l'avvenuto collegamento. Per evitare errori di comunicazione o attese indefinite, è utile eseguire il codice MATLAB, con estensione *.m*, prima del ponte esclusivamente al fine di introdurre nel *Workspace* le variabili definite, quindi il dispositivo utilizzato, il nodo, il topic e il messaggio scambiato. In questo modo, ROS2 ne riconosce l'esistenza e si inizializza la connessione; è possibile verificare questa condizione tramite i comandi da terminale Linux `ros2 node list`, il quale dovrebbe restituire il nodo creato per ogni esempio, oltre a quelli di default, e `ros2 topic list`, il quale dovrebbe far comparire il topic associato creato. Ora occorre che ROS2 pubblichi il messaggio ricevuto da MATLAB su MQTT attraverso lo script ponte su bash: si crea il file bridge per scrivere il codice tramite il comando `touch LEDbridge.sh` e si aggiungono aggiungendo i permessi di esecuzione mediante `chmod +x LEDbridge.sh`. Si esegue il ponte con `./LEDBridge.sh` per aprire la comunicazione tra ROS2 e MQTT e si vedranno stampate le frasi del codice del tipo "Avvio ascolto su ROS 2 e pubblicazione su MQTT..." che attestano l'avvenuta connessione. A questo punto, si esegue lo script di MATLAB per un numero desiderato di volte e si potranno vedere da terminale i messaggi scambiati che vengono stampati. Nel caso del LED, ad esempio, si visualizzeranno messaggi del tipo "Ricevuto messaggio ROS 2: accendi" o "Pubblicazione su MQTT: acceso", e il LED di M5StickCPlus che si accende o si spegne a seconda della stringa indicata su MATLAB (Figura 3.2).

Figura 3.2. LED acceso di M5StickCPlus

3.2. Risultati

Con il primo esempio studiato si è dimostrata la corretta gestione della comunicazione dall'alto al basso livello, in quanto il sistema ha interpretato correttamente i comandi inviati tra MATLAB e M5StickCPlus accendendo o spegnendo il LED in base al messaggio scambiato con ROS2 e MQTT. Tramite il secondo esempio, si è studiata la comunicazione opposta dal basso all'alto livello del sistema poiché quest'ultimo ha trasmesso i dati relativi al pulsante premuto di M5 in MATLAB generando a sua volta un messaggio di risposta appropriato e dimostrando che il sistema può comunicare in modo bidirezionale con la capacità di inviare dei feedback. L'esempio del servo estende la comunicazione vista con il LED poiché si è voluto testare anche l'integrazione dell'attuatore fisico con il sistema di controllo remoto dimostrando anche la capacità del sistema di controllare un servomotore connesso a M5StickCPlus tramite un angolo di posizione inviato da MATLAB. Per quanto riguarda il LED, si è constatato che si accende o si spegne a seconda del messaggio inviato da MATLAB confermando che la comunicazione, tramite ROS2 e MQTT, arriva a M5StickCPlus. Nello script non si è inserito un ciclo loop per la trasmissione del segnale per illuminare il LED, poiché si è notato che ciò genera un tempo di latenza con la conseguente perdita di alcuni messaggi. Infatti, se vengono pubblicati messaggi in modo continuo, il LED dell'M5 non riesce a stare al passo con l'invio del messaggio. Tra il comando di MATLAB e l'accensione del LED passano circa 100 millisecondi, infatti la comunicazione è rapida con tempo di latenza molto basso. Ciò dimostra l'efficienta del protocollo MQTT, confermando lo studio [36], per la gestione di payload di piccola dimensione. Si ottiene la stessa cosa con i pulsanti dell'M5: la comunicazione avviene, ma se vengono premuti in modo continuativo, non vengono visualizzati tutti i messaggi corrispondenti su MATLAB. Si può dire che la comunicazione è avvenuta effettivamente in

real-time poiché la maggior parte dei pacchetti è stata letta, tralasciandone una percentuale molto bassa, inferiore all'1%, e inoltre la scrittura dei messaggi sul software è pressoché immediata con una media di 200 ms di tempo di latenza. La perdita dei pacchetti può essere influenzata anche dalla congestione della rete, mentre il tempo di latenza può aumentare in mancanza di connessione Wi-Fi, con conseguente perdita dei messaggi. Inoltre, se cade la connessione Internet, il sistema dovrà essere riavviato poiché non continuerà lo scambio di messaggi al momento del ripristino della rete. Similmente per il servo, il tempo di latenza tra l'invio della posizione e l'effettivo movimento è di circa 120 ms; il servomotore ha dimostrato una buona prontezza con una precisione dell'angolo di spostamento. In tutti e tre gli esempi, si è ottenuta un'accuratezza della trasmissione vicina al 100% poiché nei casi di ricezione riuscita l'input è stato interpretato bene dal sistema generando l'output esatto in linea con i codici scritti in MATLAB o in Arduino IDE.

In questo progetto, si può concludere che sia MQTT che ROS2 si sono dimostrati abbastanza efficienti con tempo di latenza trascurabile poiché i messaggi inviati nei tre esempi sono di dimensioni ridotte in quanto costituiti da semplici comandi che vengono gestiti con affidabilità e velocità in real-time. Tuttavia, si sono verificati problemi di instabilità con ROS2 che hanno portato all'interruzione della comunicazione, per cui potrebbe essere ideale provare versioni più recenti del middleware con una Quality of Service migliore e porre attenzione nei casi critici della vita reale.

Nell'esecuzione degli esempi, si è constatato che è importante inserire dei ritardi di alcuni secondi (*delay*) nei codici in modo da permettere una stabilità di comunicazione, non sovraccaricare troppo la rete con comandi inviati frequentemente e osservare in modo chiaro la risposta da parte del sistema, d'altro canto aumenta la latenza del sistema. Nel caso del LED i delay sono utili per poter capire quale comando è stato eseguito in quel momento; nell'esempio

dei pulsanti sono importanti per evitare errori nella risposta di MATLAB in cui i messaggi potrebbero non coincidere; nel caso del servomotore, i delay permettono al componente di compiere il movimento completo indicato dall'angolo, in modo da escludere la possibilità che si sovrappongano più richieste di MATLAB generando errori di posizione. Per un approfondimento dei codici utilizzati, si rimanda all'Appendice B.

In più, per quanto riguarda il servo, è importante verificare il suo corretto collegamento e funzionamento tramite UIFlow (di cui è disponibile anche la versione online), un IDE di programmazione grafica di facile utilizzo poiché consente la creazione di programmi semplicemente selezionando e trascinando blocchi di codici senza avere la necessità di scriverne altri molto complessi. D'altra parte, UIFlow supporta MicroPython che permette di personalizzare lo script. Inoltre, è compatibile con M5Stack e offre diversi moduli per semplificare il controllo dei componenti, tra cui il modulo driver nella sezione HAT. Bisogna associare il dispositivo utilizzato a UIFlow, inserendo l'API Key indicata sullo schermo di M5StickCPlus dopo averlo connesso alla rete Wi-Fi; si aggiunge l'8Servos e si crea un programma molto semplice con il codice blocco in cui si indica l'angolo di rotazione desiderato per il canale a cui è collegato il servo da testare (Figura 3.3).

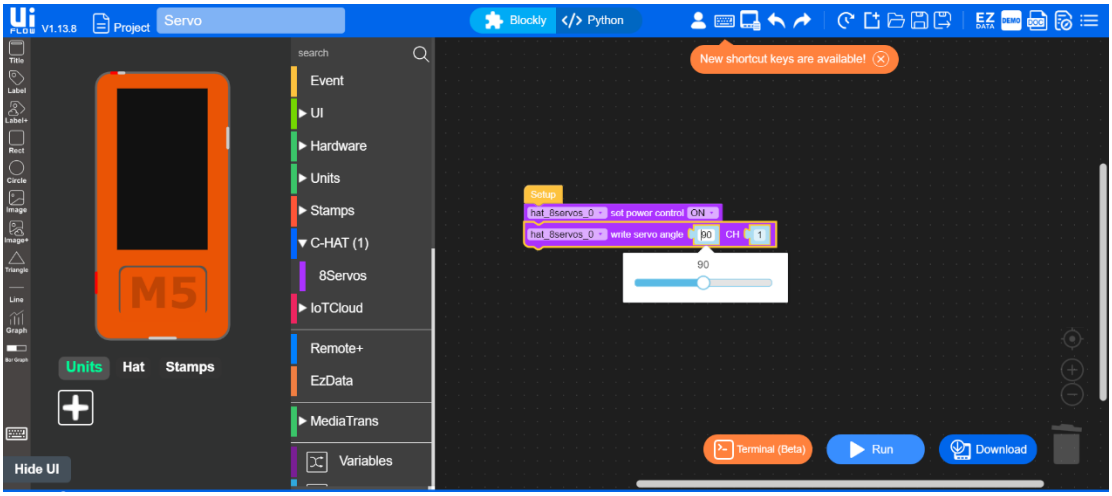

Figura 3.3. Programma di UIFlow per testare il servo SG90

Questa metodo di verifica, in fase di programmazione in Arduino IDE, ha permesso di escludere il problema di funzionamento del servo (assenza di movimento) in caso di mancata comunicazione con MATLAB. Un altro problema incontrato è stato la scelta della libreria utilizzata nell'esempio del servo poiché alcune tra quelle presenti consideravano il valore PWM del segnale, ma l'obiettivo era quello di controllare il servo nello stile di UIFlow. Infine, bisogna fare attenzione che la connessione I2C è possibile solo tra 8Servos HAT v1.1 e M5StickCPlus e non tra il modulo driver e il servomotore SG90.

Gli studi precedenti, come anche gli esempi trattati, evidenziano l'importanza della configurazione del Wi-Fi in quanto problemi di rete possono compromettere le performance di MQTT e di ROS2 generando errori nello scambio dei messaggi in termini di perdita, di attesa infinita nel terminale e di mancata comunicazione.

Al fine di evitare dei problemi nell'installazione di MQTT e ROS2 è importante seguire bene i procedimenti indicati dai siti ufficiali. In particolare, nel progetto effettuato, dopo l'istallazione sulla macchina virtuale, inizialmente è stata riscontrata una mancata comunicazione a causa del firewall. Quest'ultimo può essere sia un componente software che hardware che monitora gli accessi alle risorse di un sistema e controlla il flusso di dati tra dispositivi [38]. Difatti, come è accaduto nel corso del progetto, il firewall ha bloccato i dati trasmessi dal computer fisico alla macchina virtuale e viceversa, non favorendo il dialogo tra ROS2 e MQTT e di conseguenza tra MATLAB e M5StickCPlus. Allora, per evitare che ciò accada, bisogna procedere con la creazione di nuove "Regole connessioni in entrata" e "Regole connessioni in uscita", sia per il computer fisico dalle impostazioni del firewall che per la macchina virtuale da terminale, indicando le porte utilizzate dal protocollo di rete DDS e i due indirizzi IP in modo tale da garantire la trasmissione delle informazioni. Infine, per una

maggiore stabilità, è conveniente rendere statici gli indirizzi IP del dispositivo fisico e del simulatore, in modo che non occorra modificare periodicamente i codici di Arduino IDE, MATLAB e del bridge, evitando anche di aggiungere nuove regole di connessione per il firewall.

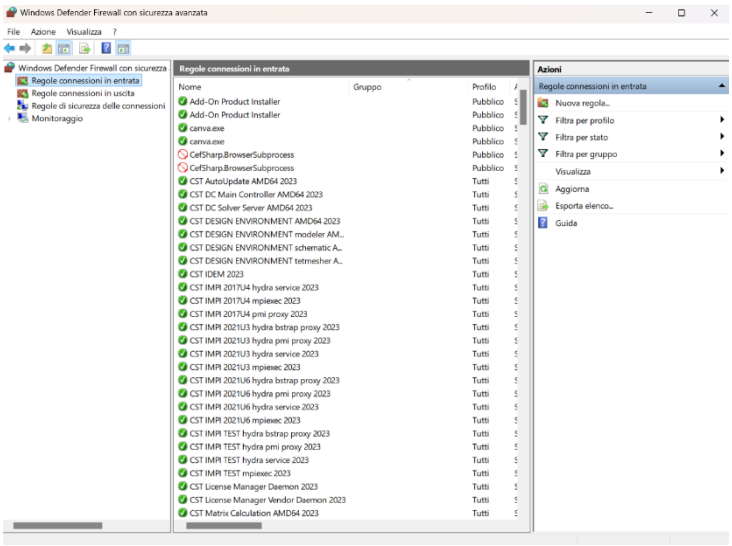

Figura 3.4. Interfaccia Windows Defender Firewall

Si invita la lettura dell'Appendice A per le librerie utilizzate, la configurazione del Wi-Fi e la guida all'installazione di MQTT e ROS2.

4. Conclusioni e sviluppi futuri

I tre esempi studiati dimostrano che è possibile far comunicare il basso livello con l'alto livello tramite degli strumenti che sono risultati efficienti. È un modo per affrontare con semplicità il problema della comunicazione e che può essere esteso a dei progetti più ampi e più realistici; ad esempio il movimento del servomotore potrebbe essere un prototipo di controllo della coda di attuazione di un pesce robotico.

Il futuro dell'IoT è promettente, con molti nuovi sviluppi e innovazioni all'orizzonte, con i fornitori di dispositivi che offrono prezzi interessanti, mentre il costo della produzione dei dispositivi IoT diminuisce. Si prevede che il numero di dispositivi IoT continuerà a crescere rapidamente e si stima che ci saranno decine di miliardi di dispositivi IoT in uso nei prossimi anni. Inoltre, l'*edge computing* sta diventando sempre più importante per l'IoT, in quanto consente di elaborare e analizzare i dati più vicino alla fonte, piuttosto che in un data center centralizzato. Questo può migliorare i tempi di risposta, ridurre la latenza e la quantità di dati che devono essere trasferiti sulle reti IoT. Inoltre, per migliorare la sicurezza e la privacy nell'IoT la tecnologia *blockchain* è in fase di esplorazione e potrà essere utilizzata per creare reti sicure e decentralizzate per i dispositivi IoT, che possono ridurre al minimo le vulnerabilità della sicurezza dei dati [42].

Secondo la Ricerca dell'Osservatorio il 2022 è stato nel complesso un anno positivo per l'Internet of Things, sia in Italia sia a livello internazionale: il mercato IoT in Italia ha proseguito la sua corsa, crescendo di ben 1 miliardo e raggiungendo così gli 8,3 miliardi di euro (+13% rispetto al 2021), per un totale di oltre 124 milioni di connessioni IoT attive (2,1 per abitante, a fronte di 1,8 nel 2022).

Oltre al mercato è cresciuta sia la consapevolezza da parte delle organizzazioni pubbliche, private e dei consumatori che la competitività nel settore. Allo stesso tempo i frequenti rincari del costo dell'energia hanno spinto aziende e consumatori a porre maggiore attenzione verso i propri consumi, sfruttando – in parte – proprio le tecnologie smart. Questo fatto, insieme ai numerosi investimenti previsti dal PNRR, ha aumentato le aspettative per il futuro dell'IoT, complici anche l'evoluzione di grandi ecosistemi e consorzi internazionali [8]. Con il continuo aumento del numero di dispositivi IoT, le aziende devono essere pronte ad adattarsi alle nuove tecnologie e ad abbracciare nuovi casi d'uso e applicazioni: coloro che saranno in grado di farlo potranno raccogliere i benefici di questa tecnologia trasformativa [42].

APPENDICE A: Inizializzazione e installazione dei software

Per utilizzare i materiali descritti nel secondo capitolo, è necessario innanzitutto inizializzarli e configurarli. Bisogna collegare l'M5StickCPlus al computer con il cavo USB type-C per permettere di configurare la scheda di sviluppo tramite *M5Burner*, un software di masterizzazione firmware che integra l'esportazione, la pubblicazione, la condivisione e altre funzioni del firmware. Si seleziona dall'elenco la scheda di sviluppo adottata, si procede con il download di *UIFlow2.0 StickC Plus* (Figura A.1) per poi eseguire il *burn* permettendo il ripristino di M5StickCPlus e la configurazione del dispositivo e della rete. Si inserisce il nome SSID e la password della connessione che si vuole adottare (è preferibile usare un hotspot personale), scegliendo il numero di transizioni al secondo che avvengono sulla linea, indicato come *baud rate*, pari a 115200, e la porta seriale dove è collegata la scheda (A.2).

Figura A.1. Interfaccia M5Burner

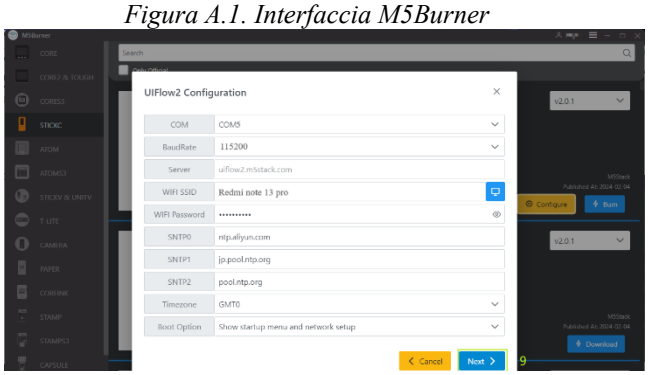

Figura A.2. Configurazione M5StickCPlus

M5StickCPlus e la rete Wi-Fi devono essere configurati anche su Arduino IDE per poter stabilire la connessione. Se non sono già disponibili, bisogna installare in Arduino IDE, tramite la documentazione ufficiale, l'ultima versione delle librerie *M5StickCPlus.h* e *WiFi.h* dal *Library Manager*. Inoltre, l'URL del *Board Manager* viene utilizzato per indicizzare le informazioni della scheda di sviluppo di una piattaforma specifica; per cui bisogna copiare l'URL di gestione della scheda M5Stack nell'*Additional Board Manager URLs:* `https://static-cdn.m5stack.com/resource/arduino/package_m5stack_index.json`.

Dopodiché, si installa la board M5Stack dal *Board Manager* e si seleziona dal menù *Tools* la scheda M5StickCPlus (Figura A.3).

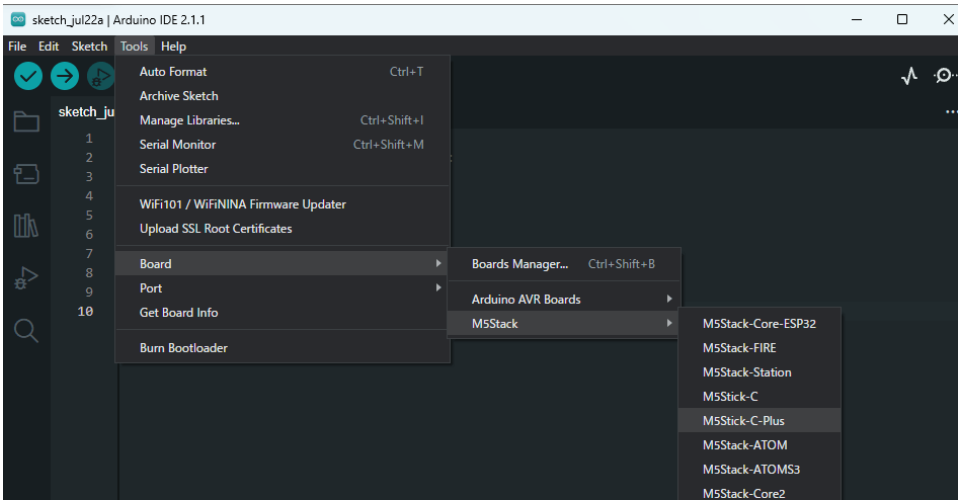

Figura A.3. Selezione scheda M5StickCPlus in Arduino IDE

Allo stesso modo, bisogna individuare la porta seriale utilizzata, che di solito viene subito riconosciuta automaticamente. *M5StickCPlus.h* consente di avere a disposizione le funzioni tipiche dell'M5 per poterlo inizializzare, specificando il suo *baud rate*:

```
M5.begin();
```

```
Serial.begin(115200);
```

Per la configurazione del Wi-Fi, è necessaria l'omonima libreria; si introducono le variabili `const char* ssid` e `const char* password`, scegliendo gli stessi valori delle credenziali utilizzate nel burner, per costruire la funzione *setup_wifi()* che garantisce la connessione e che verrà chiamata nella funzione principale *setup()* di Arduino IDE.

Viene definita, in Arduino, la variabile `const int ledPin = 10`, corrispondente al valore del pin riferito al LED superiore, indicato dalla documentazione ufficiale di M5, *M5Docs*. Invece, *M5StickCPlus.h* fornisce le funzioni *`M5.BtnA.wasPressed()`* e *`M5.BtnB.wasPressed()`* per controllare i pulsanti A e B programmandone gli effetti al momento della loro attivazione.

In Arduino, i due componenti per il controllo del servo vengono gestiti tramite *Hat_8Servos.h*, una libreria che non è standard di Arduino, ma è sviluppata da M5Stack ed è contenuta in una repository di GitHub tra gli esempi della documentazione ufficiale del 8Servos HAT v1.1:

```
{
  "name": "M5Hat-8Servos",
  "description": "Library for M5Stack HAT 8SERVO",
  "keywords": "M5Stack HAT 8SERVO",
  "authors": {
    "name": "M5Stack",
    "url": "https://www.m5stack.com/"
  },
  "repository": {
    "type": "git",
    "url": "https://github.com/m5stack/M5Hat-8Servos"
  },
  "version": "0.0.2",
  "frameworks": "arduino",
  "platforms": "espressif32"
                              }.
```

Codice della libreria Hat_8Servos.h

La libreria permette di trattare il modulo driver come un oggetto; infatti, viene istanziato l'oggetto `Hat_8Servos drive,` che permette di controllare il movimento del servo con lo stesso principio di UIFlow, cioè, attivare il servo ed indicare l'angolo desiderato per il canale scelto. È necessario anche verificare l'esistenza di comunicazione tra M5StickPlus e l'8Servos. Come si è detto in precedenza, la comunicazione tra i due dispositivi è di tipo I2C e quindi deve essere configurata. Dalla documentazione ufficiale della scheda di espansione, si possono leggere i pin SDA e SLC, i quali rispettivamente hanno il valore di 0 e di 26, e l'indirizzo I2C del driver, pari a 0x36. Questi dati devono essere inseriti all'interno della funzione di inizializzazione dell'8Servos HAT v1.1, `drive.begin(&Wire, 0, 26,0x36),` per verificare che è stato inizializzato correttamente.

Per gestire la comunicazione, essendo ROS e MQTT compatibili con i sistemi UNIX, occorre installare una macchina virtuale dal sito ufficiale di VirtualBox oppure di VMWare ed inserire l'ISO della versione di Ubuntu reperibile dal sito ufficiale della distribuzione Linux. Una volta installati i vari pacchetti APT, si procede con l'installazione di ROS e MQTT seguendo la guida nei loro siti ufficiali. Per il ROS si consulti la documentazione della versione Humble Hawksbill. Da terminale, bisogna verificare che l'ambiente utilizza la codifica UTF-8 per garantire la corretta visualizzazione ed elaborazione dei testi nelle varie lingue contenenti caratteri speciali:

```
locale  #check for UTF-8
sudo apt update && sudo apt install locales
sudo locale-gen en_US en_US.UTF-8
sudo update-locale LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8
export LANG=en_US.UTF-8
locale  #verify settings.
```

In seguito, si aggiungono i pacchetti di ROS2, assicurandosi che sia abilitato l'Ubuntu Universe repository, una raccolta di pacchetti di applicazioni e strumenti sviluppata dalla comunità:

```
sudo apt install software-properties-common
sudo add-apt-repository universe.
```

Si installa la chiave GPG di ROS2 che permette di verificare che i pacchetti software di ROS2 siano sicuri e senza problemi di autenticità:

```
sudo apt update && sudo apt install curl -y
sudo curl -sSL
https://raw.githubusercontent.com/ros/rosdistro
/master/ros.key -o /usr/share/keyrings/ros-archive-
keyring.gpg.
```

Si include la repository di ROS 2 alla lista delle sorgenti di APT del tuo sistema per permettere l'installazione e l'aggiornamento dei pacchetti ROS2 tramite il gestore di pacchetti di Ubuntu:

```
echo "deb [arch=$(dpkg --print-architecture) signed-by=
/usr/share/keyrings/ros-archive-keyring.gpg]
http://packages. ros.org/ros2/ubuntu $(. /etc/os-release
&& echo $UBUNTU_CODENAME) main" | sudo tee
/etc/apt/sources.list.d/ros2.list > /dev/null.
```

A questo punto occorre installare gli strumenti ROS e di sviluppo:

```
sudo apt update && sudo apt install -y
   python3-flake8-docstrings
   python3-pip
   python3-pytest-cov
   ros-dev-tools

sudo apt install -y
   python3-flake8-blind-except
   python3-flake8-builtins
   python3-flake8-class-newline
   python3-flake8-comprehensions
   python3-flake8-deprecated
   python3-flake8-import-order
   python3-flake8-quotes
   python3-pytest-repeat
   python3-pytest-rerunfailures.
```

È necessario creare una cartella di lavoro in cui si copiano tutte le repository:

```
mkdir -p ~/ros2_humble/src
cd ~/ros2_humble
vcs import --input
https://raw.githubusercontent.com/ros2/ros2/humble
```

```
/ros2.repos src.
```

Infine, bisogna assicurarsi che tutti i pacchetti siano aggiornati tramite `sudo apt upgrade`.

Per MQTT, si scarica Mosquitto abilitando il servizio:

```
sudo apt install mosquitto mosquitto-clients

sudo systemctl start mosquitto
sudo systemctl enable mosquito.
```

In Arduino, per la comunicazione con MQTT, si utilizza la libreria *PubSubClient.h* che istanzia un oggetto *client* per riferirsi al MQTT Client tramite alcune funzioni. Si inizializza il client inserendo l'indirizzo IP, indicato prima tra le variabili definite, `const char* mqtt_server = "192.168.61.91",` e la porta 1883 che è quella tipica utilizzata dalla comunicazione MQTT. Per visualizzare l'indirizzo IP legato alla macchina virtuale dove gira MQTT, si può utilizzare il comando `ip a`, se installato. In generale, è importante che il computer fisico, dove si ha MATLAB e Arduino IDE, e la macchina virtuale, che contiene ROS2 e MQTT, siano connessi tra di loro; quindi, tra le impostazioni di VirtualBox per la macchina virtuale, bisogna scegliere il tipo di connessione di rete a ponte e collegarli virtualmente tramite gli indirizzi IP. Per individuare l'indirizzo IP del computer fisico, si apre il Prompt dei Comandi, nel caso di Windows, e si digita `ipconfig`. Ora si procede con il ping degli indirizzi IP: nella macchina virtuale si esegue il `ping` con l'indirizzo IP del computer fisico e viceversa. La comunicazione sarà attiva se non compariranno messaggi di errore.

L'inizializzazione di ROS2, invece, avviene all'interno di MATLAB. Ci si connette al dispositivo ROS, quindi alla macchina virtuale, tramite la funzione `ros2device` in cui si indica l'indirizzo IP dell'ambiente virtuale, il nome utente e la password associati. Inoltre, si

impostano le variabili di ambiente di ROS2 in modo che siano le stesse tra fisico e virtuale. Quindi, in MATLAB si impostano il numero di dominio e il valore di implementazione:

```
setenv('ROS_DOMAIN_ID', '0');
setenv('RMW_IMPLEMENTATION', 'rmw_fastrtps_cpp');
```

si implementano le stesse variabili anche sul bash di Linux:

```
export ROS_DOMAIN_ID=0
export RMW_IMPLEMENTATION=rmw_fastrtps_cpp.
```

È importante che le variabili d'ambiente si configurino ad ogni avvio della macchina virtuale, altrimenti potrebbero esserci problemi di comunicazione. Si possono semplificare i passaggi, inserendo i comandi per impostare le variabili all'interno di un file che viene caricato all'avvio tramite il comando `source`. Da terminale Linux deve essere avviato anche ROS2 tramite:

```
source /opt/ros/humble/setup.bash.
```

Poiché ROS2 comunica con MATLAB e MQTT con Arduino IDE, allora è indispensabile uno script ponte di tipo bash che permette di gestire lo scambio di messaggi tra ROS e MQTT. Ciò permette di configurare MQTT indicando sempre l'indirizzo IP e la porta di connessione; inoltre, devono essere definiti anche i topic utilizzati sia per pubblicare i messaggi che per sottoscriversi, a seconda dell'esempio trattato. Deve essere indicato anche il tipo di messaggio utilizzato che deve coincidere con quello stabilito in MATLAB, altrimenti genera un errore nella comunicazione. Per il controllo del LED superiore di M5StickCPlus, si definiscono:

```
# Configurazione MQTT
MQTT_HOST="192.168.61.91"
MQTT_PORT="1883"
MQTT_TOPIC_PUB="/accensione"
MQTT_TOPIC_SUB="/accensione"

# Configurazione ROS 2
ROS2_TOPIC_SUB="/accensione"
ROS2_MSG_TYPE="std_msgs/String".
```

Per l'esempio dei pulsanti della scheda di sviluppo, si determinano:

```
# Configurazione MQTT
MQTT_HOST="192.168.61.91"
MQTT_PORT="1883"
MQTT_TOPIC_SUB="/m5stickc/button"

# Configurazione ROS 2
ROS2_TOPIC_PUB="/m5stickc/button"
ROS2_MSG_TYPE="std_msgs/String".
```

Per il controllo del movimento del servo, si utilizzano:

```
# Configurazione MQTT
MQTT_HOST="192.168.61.91"
MQTT_PORT="1883"
MQTT_TOPIC_PUB="/servo_angle"
MQTT_TOPIC_SUB="/servo_angle"

# Configurazione ROS 2
ROS2_TOPIC_SUB="/servo_angle"
ROS2_MSG_TYPE="std_msgs/String".
```

APPENDICE B: Codici utilizzati

MATLAB

```
% Connetti al dispositivo ROS 2
ros2device('192.168.61.91', 'marco', 'Ubuntu');

% Imposta le variabili d'ambiente ROS 2
setenv('ROS_DOMAIN_ID', '0');  % Assicurati che questo sia lo stesso
numero di dominio DDS impostato in ROS 2
setenv('RMW_IMPLEMENTATION',     'rmw_fastrtps_cpp');     %     Usa
'rmw_fastrtps_cpp' o il valore appropriato

% Crea un nodo ROS 2
ros2Node = ros2node('/led_control');

% Nome del topic ROS 2
rosTopicName = '/accensione';

% Crea un publisher
ledPub = ros2publisher(ros2Node, rosTopicName, 'std_msgs/String');

% Messaggio da inviare
messageToSend = 'spegni';

% Crea un messaggio ROS
msg = ros2message(ledPub);
msg.data = messageToSend;

% Pubblica il messaggio
send(ledPub, msg);
```

BASH

```
#!/bin/bash

# Sorgente delle variabili d'ambiente ROS 2
source /opt/ros/humble/setup.bash

# Configurazione MQTT
MQTT_HOST="192.168.26.91"
MQTT_PORT="1883"
MQTT_TOPIC_PUB="/accensione"
MQTT_TOPIC_SUB="/accensione"

# Configurazione ROS 2
ROS2_TOPIC_SUB="/accensione"
ROS2_MSG_TYPE="std_msgs/String"

# Funzione per pubblicare su MQTT
publish_to_mqtt() {
```

```
    local msg=$1
    echo "Pubblicazione su MQTT: $msg"
    mosquitto_pub -h $MQTT_HOST -p $MQTT_PORT -t $MQTT_TOPIC_PUB -m
"$msg"
}

# Ascolta i messaggi ROS 2 e pubblicali su MQTT
echo "Avvio ascolto su ROS 2 e pubblicazione su MQTT..."
ros2 topic echo $ROS2_TOPIC_SUB --qos-profile sensor_data | while
read -r line; do
    if [[ $line == data:* ]]; then
        msg=${line#data: }
        echo "Ricevuto messaggio ROS 2: $msg"

        # Pubblica il messaggio su MQTT
        publish_to_mqtt "$msg"

        # Se il messaggio ROS 2 è "accendi", rispondi con "acceso"
su MQTT
        if [ "$msg" = "accendi" ]; then
            publish_to_mqtt "acceso"
        fi
    fi
done
```

```
ARDUINO IDE

#include <WiFi.h>
#include <M5StickCPlus.h>
#include <PubSubClient.h>

// WiFi credentials
const char* ssid = "Redmi note 13 pro";
const char* password = "hotspot0202";

// MQTT broker configuration
const char* mqtt_server = "192.168.61.91";
const int mqtt_port = 1883;

// WiFi and MQTT client
WiFiClient espClient;
PubSubClient client(espClient);

// Pin GPIO per il controllo del LED superiore
const int ledPin = 10; // Sostituisci con il pin corretto per il LED
superiore

void setup() {
  M5.begin();
  Serial.begin(115200);

  // Configure WiFi
  setup_wifi();
```

```
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, HIGH); // Assicurati che il LED sia spento
all'avvio

  // Configure MQTT client
  client.setServer(mqtt_server, mqtt_port);
  client.setCallback(callback);

  // Attempt to connect to MQTT broker
  reconnect();
}

// WiFi Connection
void setup_wifi() {

  delay(10);
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println();
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}

void reconnect() {
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    if (client.connect("M5StickCPlus")) {
      Serial.println("connected");
      client.subscribe("/accensione"); // Subscribe to topic for LED
control
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      delay(5000);
    }
  }
}

void loop() {
  if (!client.connected()) {
    reconnect();
  }
```

```
  client.loop();
}

// Callback function for MQTT messages
void callback(char* topic, byte* payload, unsigned int length) {
  String message;
  for (unsigned int i = 0; i < length; i++) {
    message += (char)payload[i];
  }
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] ");
  Serial.println(message);

  // Check if the message is to control the LED
  if (String(topic) == "/accensione") {
    if (message.equals("accendi")) {
      digitalWrite(ledPin, LOW);
    } else if (message.equals("spegni")) {
      digitalWrite(ledPin, HIGH);
    }
  }
}
```

Tabella B.1. Codici per l'esempio del LED

```
MATLAB

% Connetti al dispositivo ROS 2
device = ros2device('192.168.61.91', 'marco', 'Ubuntu');

% Imposta le variabili d'ambiente ROS 2
setenv('ROS_DOMAIN_ID', '0');  % Usa lo stesso numero di dominio DDS
impostato in ROS 2
setenv('RMW_IMPLEMENTATION', 'rmw_fastrtps_cpp');

% Crea un nodo ROS2
ros2Node = ros2node('/but_node');

% Sottoscrivi al topic ROS2 che riceve i messaggi MQTT
sub      =      ros2subscriber(ros2Node,      '/m5stickc/button',
@messageCallback);

% Funzione callback per gestire i messaggi ricevuti
function messageCallback(message)
    disp('Received message:');
    disp(message);
    % Verifica il contenuto del messaggio e stampa il messaggio
appropriato su MATLAB
    if strcmp(message.data, 'Button A pressed')
        disp('Tasto centrale premuto');
```

```
    end
    if strcmp(message.data, 'Button B pressed')
        disp('Tasto a destra premuto');
    end
end
```

```
BASH

#!/bin/bash

# Sorgente delle variabili d'ambiente ROS 2
source /opt/ros/humble/setup.bash

# Configurazione MQTT
MQTT_HOST="192.168.61.91"
MQTT_PORT="1883"
MQTT_TOPIC_SUB="/m5stickc/button"

# Configurazione ROS 2
ROS2_TOPIC_PUB="/m5stickc/button"
ROS2_MSG_TYPE="std_msgs/String"

# Funzione per pubblicare su ROS 2
publish_to_ros2() {
    local msg=$1
    echo "Pubblicazione su ROS 2: $msg"
    ros2 topic pub --once $ROS2_TOPIC_PUB $ROS2_MSG_TYPE "{data:
'$msg'}"
}

# Ascolta i messaggi MQTT e pubblicali su ROS 2
echo "Avvio ascolto su MQTT e pubblicazione su ROS 2..."
mosquitto_sub -h $MQTT_HOST -p $MQTT_PORT -t $MQTT_TOPIC_SUB | while
read -r line; do
    echo "Ricevuto messaggio MQTT: $line"
    publish_to_ros2 "$line"
done
```

```
ARDUINO IDE

#include <M5StickCPlus.h>
#include <WiFi.h>
#include <PubSubClient.h>

const char* ssid = "Redmi note 13 pro";
const char* password = "hotspot0202";
const char* mqtt_server = "192.168.61.91";

WiFiClient espClient;
PubSubClient client(espClient);

void setup() {
    M5.begin();
    M5.Lcd.print("Connecting");
```

```
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        M5.Lcd.print(".");
    }

    client.setServer(mqtt_server, 1883);
    while (!client.connected()) {
        M5.Lcd.print("Connecting to MQTT...");
        if (client.connect("M5StickCPlus")) {
            M5.Lcd.print("Connected to MQTT");
        } else {
            M5.Lcd.print("Failed   to   connect,   retrying   in   5
seconds");
            delay(5000);
        }
    }

    M5.Lcd.print("Connected");
}

void loop() {
    M5.update();
    if (!client.connected()) {
        while (!client.connected()) {
            M5.Lcd.print("Reconnecting to MQTT...");
            if (client.connect("M5StickCPlus")) {
                M5.Lcd.print("Reconnected to MQTT");
            } else {
                M5.Lcd.print("Failed  to  reconnect,  retrying  in  5
seconds");
                delay(5000);
            }
        }
    }
    client.loop();
    if (M5.BtnA.wasPressed()) {
        client.publish("/m5stickc/button", "Button A pressed");
        M5.Lcd.print("\nButton A pressed");
    } else if (M5.BtnB.wasPressed()) {
        client.publish("/m5stickc/button", "Button B pressed");
        M5.Lcd.print("\nButton B pressed");
    }
}
```

Tabella B.2. Codici per l'esempio dei pulsanti

```
MATLAB

% Connettersi al dispositivo ROS 2
```

```
ros2device('192.168.61.91', 'marco', 'Ubuntu');

% Imposta le variabili d'ambiente ROS 2
setenv('ROS_DOMAIN_ID', '0');  % Usa lo stesso numero di dominio DDS
impostato in ROS 2
setenv('RMW_IMPLEMENTATION', 'rmw_fastrtps_cpp');

% Creare un nodo ROS2
node = ros2node("/servo_node");

% Creare un publisher per inviare l'angolo del servo
pub_angle = ros2publisher(node, "/servo_angle", "std_msgs/String");

% Creare un messaggio per l'angolo del servo
msg_angle = ros2message(pub_angle);


% Impostare l'angolo desiderato
desiredAngle = '180';
msg_angle.data = desiredAngle;

% Inviare il messaggio per impostare l'angolo del servo
send(pub_angle, msg_angle);
```

```
BASH

#!/bin/bash

# Sorgente delle variabili d'ambiente ROS 2
source /opt/ros/humble/setup.bash

# Configurazione MQTT
MQTT_HOST="192.168.61.91"
MQTT_PORT="1883"
MQTT_TOPIC_PUB="/servo_angle"
MQTT_TOPIC_SUB="/servo_angle"

# Configurazione ROS 2
ROS2_TOPIC_SUB="/servo_angle"
ROS2_MSG_TYPE="std_msgs/String"

# Verifica se il topic esiste e qual è il suo tipo
if ! ros2 topic info $ROS2_TOPIC_SUB; then
    echo "Topic $ROS2_TOPIC_SUB non esiste o il tipo non può essere
determinato"
    exit 1
fi

# Funzione per pubblicare su MQTT
publish_to_mqtt() {
    local msg=$1
    echo "Pubblicazione su MQTT: $msg"
```

```
    mosquitto_pub -h $MQTT_HOST -p $MQTT_PORT -t $MQTT_TOPIC_PUB -m
"$msg"
}

# Ascolta i messaggi ROS 2 e pubblicali su MQTT
echo "Avvio ascolto su ROS 2 e pubblicazione su MQTT..."
ros2 topic echo $ROS2_TOPIC_SUB --qos-profile sensor_data | while
read -r line
do
    if [[ $line == *"data:"* ]]; then
        msg=${line#*data: }
        echo "Ricevuto messaggio ROS 2: $msg"

        # Controlla che l'angolo sia valido (tra 0 e 180 gradi)
        if [[ "$msg" =~ ^[0-9]+$ ]] && [ "$msg" -ge 0 ] && [ "$msg"
-le 180 ]; then
            publish_to_mqtt "$msg"
        else
            echo "Angolo non valido ricevuto: $msg"
        fi
    fi
done
```

ARDUINO IDE

```
#include <WiFi.h>
#include <M5StickCPlus.h>
#include <PubSubClient.h>
#include "Hat_8Servos.h"

Hat_8Servos drive;

// WiFi credentials
const char* ssid = "Redmi note 13 pro";
const char* password = "hotspot0202";

// MQTT broker configuration
const char* mqtt_server = "192.168.61.91";
const int mqtt_port = 1883;

// WiFi and MQTT client
WiFiClient espClient;
PubSubClient client(espClient);

void setup() {
  M5.begin();
  Serial.begin(115200);

  // Configure WiFi
  setup_wifi();

  if (drive.begin(&Wire, 0, 26, 0x36)) {
    M5.Lcd.print("8Servos HAT inizializzato correttamente");
```

```
  }
  drive.enableServoPower(1);  // Abilita l'alimentazione del servo
sul canale 1
  vTaskDelay(1000);

  // Configure MQTT client
  client.setServer(mqtt_server, mqtt_port);
  client.setCallback(callback);

  // Attempt to connect to MQTT broker
  reconnect();
}

// WiFi Connection
void setup_wifi() {
  delay(10);
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println();
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}

void reconnect() {
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    if (client.connect("M5StickCPlus")) {
      Serial.println("connected");
      client.subscribe("/servo_angle"); // Subscribe to topic for
servo control
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      delay(5000);
    }
  }
}

void loop() {
  if (!client.connected()) {
    reconnect();
  }
```

```
  client.loop();
}

// Callback function for MQTT messages
void callback(char* topic, byte* payload, unsigned int length) {
  if (String(topic) == "/servo_angle") {
    // Converti il payload in una stringa
    String angleStr = "";
    for (unsigned int i = 0; i < length; i++) {
      angleStr += (char)payload[i];
    }

    // Converti la stringa in un intero
    int angle = angleStr.toInt();
    Serial.print("Angle received: ");
    Serial.println(angle);

    if (angle >= 0 && angle <= 180) {
      drive.setServoAngle(1, angle);  // Imposta l'angolo del servo
su 0 gradi
      vTaskDelay(2000);
    }
  }
}
```

Tabella B.3. Codici per esempio del servo

Bibliografia/Sitografia

[1] What is the IoT?: ibm.com

[2] An Empirical Study on System Level Aspects of Internet of Things (IoT): ieeexplore.ieee.org

[3] Cyber-Physical System: https://ptolemy.berkeley.edu/projects/cps/

[4] A Survey of Cyber-Physical Systems: ieeexplore.ieee.org

[5] Cyber Physical Systems in the Context of Industry 4.0: ieeexplore.ieee.org

[6] What are Industry 4.0, the Fourth Industrial Revolution, and 4IR?: mckinsey.com

[7] Industria 4.0: it.wikipedia.org

[8] Internet of Things (IoT): significato, esempi e applicazioni: Osservatorio IoT del Politecnico di Milano, blog.osservatori.net

[9] Smart Car e Smart Mobility, cosa sono e come funzionano: Osservatorio IoT del Politecnico di Milano, blog.osservatori.net

[10] Smart Home, come funziona l'IoT per una casa intelligente: Osservatorio IoT del Politecnico di Milano, blog.osservatori.net

[11] Smart City, cos'è e come funziona la città intelligente: Osservatorio IoT del Politecnico di Milano, blog.osservatori.net

[12] Agricoltura 4.0: cos'è, evoluzione, vantaggi e tecnologie: Osservatorio IoT del Politecnico di Milano, blog.osservatori.net

[13] Smart Energy e efficienza energetica: startup e soluzioni: Osservatorio IoT del Politecnico di Milano, blog.osservatori.net

[14] Cyber-Physical System (CPS): State of the Art: ieeexplore.ieee.org

[15] Tra robotica e AI, 16 innovazioni Made in Italy da tenere d’occhio: fortuneita.com

[16] Internet of Things: i rischi per la Sicurezza Informatica e la Privacy: Osservatorio IoT del Politecnico di Milano, blog.osservatori.net

[17] What is the difference between an AUV and a ROV?: oceanservice.noaa.gov

[18] Scaradozzi, D., Palmieri, G., Costa, D., Zingaretti, S., Panebianco, L., Ciuccoli, N., … & Callegari, M. (2017). "UNIVPM BRAVe: A Hybrid Propulsion Underwater Research Vehicle". *International Journal of Automation Technology (IJAT), 11(3),* 404-414.

[19] T. Salumäe, R. Raag, J. Rebane, A. Ernits, G. Toming, M. Ratas, and M. Kruusmaa, "Design principle of a biomimetic underwater robot U-CAT," MTS/IEEE OCEANS 2014, St. John's, 2014.

[20] F. Bruno, M. Muzzupappa, A. Lagudi, A. Gallo, F. Spadafora, G. Ritacco, A. Angilica, L. Barbieri, N. Di Lecce, G. Saviozzi, C. Laschi, R. Guida, and G. Di Stefano, "A ROV for supporting the planned maintenance in underwater archaeological sites," MTS/IEEE OCEANS2015, Genoa, 2014

[21] Robotica sottomarina: mdm.univpm.it

[22] M5StickC: m5stack.com

[23] M5StickCPlus: it.wikipedia.org

[24] M5StickCPlus: m5stack.com

[25] 8Servos HAT v1.1: m5stack.com

[26] Servo SG90: m5stack.com

[27] Arduino IDE: it.wikipedia.org

[28] Arduino IDE: m5stack.com

[29] ROS: an open-source Robot Operating System: researchgate.net

[30] ROS documentation: docs.ros.org

[31] Introduction to MQTT: it.mathworks.com

[32] Eclipse Mosquitto: mosquito.org

[33] Christopher Negus, “Ubuntu Linux Toolbox: 1000+ Commands for Power Users”

[34] MATLAB: it.mathworks.com

[35] “MATLAB, Starting Matlab” The MathWorks, Natick, MA, 2012: itb.biologie.hu-berlin.de, Google Scholar

[36] Performance Evaluation of Application Layer Protocols for the Internet-of-Things: ieeexplore.ieee.org

[37] D. Thangavel, X. Ma, A. Valera, H. X. Tan, and C. K. Y. Tan, “Performance evaluation of mqtt and coap via a common middleware,” in *2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, April 2014, pp. 1–6.

[38] Y. Chen and T. Kunz, “Performance evaluation of iot protocols under a constrained wireless access network,” in *2016 International Conference on Selected Topics in Mobile Wireless Networking (MoWNeT)*, April 2016, pp. 1–7.

[39] A ROS2-Based Framework for Industrial Automation Systems: ieeexplore.ieee.org

[40] Y. Maruyama, S. Kato, and T. Azumi, “Exploring the performance of ros2,” in Proceedings of the 13th International Conference on Embedded Software. New York, NY, USA: ACM, 2016.

[41] Che cos’è un firewall: support.microsoft.com

[42] The future of IoT: ibm.com

Ringraziamenti

Questo percorso è giunto al termine e vorrei ringraziare le persone che mi sono state accanto.

Innanzitutto, vorrei ringraziare il mio relatore, il professore David Scaradozzi, per avermi dato questa opportunità e la mia correlatrice, Flavia Gioiello, che mi ha supportato durante il tirocinio e la stesura della tesi.

Ringrazio i miei genitori che mi hanno sostenuto, hanno creduto in me e mi hanno dato la possibilità di compiere questo percorso e non solo. Nonostante la lontananza, questa esperienza ci ha reso più uniti e il rapporto è diventato più armonioso e sono sicuro che andrà sempre meglio. Per questo vorrei ringraziare anche Cristiana, che è stata per me fondamentale.

A Benedetta, la mia sorella preferita, per essermi stata accanto nelle decisioni, nelle lamentele, nei consigli e nelle chiacchierate perché so che posso e potrò contare sempre su di te per il rapporto speciale che abbiamo.

Ad Aurora e Miriam, che dire? Ci pensa Fili fila 3 a spiegarlo. Vi ringrazio perché, nonostante siamo stati in tre città diverse in questi tre anni, mi avete fatto e fate sentire la vostra vicinanza ovunque perché siamo sempre in contatto per dire qualsiasi cosa, per ridere, per cantare, per raccontare un sogno, una figuraccia, per mandare audio di monologhi che sappiamo che a nessuno importa ma che comunque ascolteremo (tranne Miriam). Vi ringrazio per la semplicità del nostro rapporto che lo rende unico in modo da sentirmi sempre a mio agio e che ogni volta che siamo tutti e tre, anche se poche purtroppo, è come se non ci fossimo mai allontanati geograficamente.

A Filippo, perché so che posso contare su di te, dalle uscite sotto casa, ai consigli e alle telefonate psicologiche dove senti tutte le lamentele, le preoccupazioni e lo sfogo per la città, lo studio e qualsiasi cosa. Grazie per il supporto reciproco che è sempre stato forte nonostante la lontananza e che sono sicuro ci sarà sempre.

Ai coinquilini e ai colleghi di università che hanno reso l'esperienza più colorata in un'Ancona un po' grigia e in particolare ad Alessio, per avermi supportato e sopportato in questi tre anni. Ti ringrazio per avermi fatto ridere, per essere stato sempre presente dall'università fino alle chiacchierate e anche lamentele sul balcone e per avermi fatto vedere il mondo con occhi diversi. Grazie per i nostri giovedì culinari e tutte le grandi e piccole cose che hanno reso Ancona piacevole.

Ad Alessia, Veronica, Beatrice, Giulia, Francesca e Federico, vi ringrazio per le serate passate insieme, in disco o anche per le serate tranquille, a parlare e a ridere delle cose che ci sono successe, lasciando da parte l'ansia e lo stress dell'università.

A Eddy, Marti, Anita, Diego, Asia, Alice, Michele e Ilaria per avermi ascoltato nei momenti di sfogo e dato buoni consigli che però a volte io non seguivo. Grazie per le chiacchierate e i bei momenti passati insieme.

Ai miei nonni, zii e cugini perché so che, anche se siamo distanti, mi supportate nelle scelte che prendo.

È difficile esprimere a parole in una pagina il bene che voglio a tutti voi, ma ora ci tengo solo a dirvi: grazie,

Marco Conte