



Università degli Studi di Ferrara

Corso di Laurea in Informatica

Analisi e programmazione del sistema Mars Rover Picar-B in ambito IoT

Relatore

Prof. Carlo Giannelli

Laureando

Marco Beltrame

Correlatore

Dott. Simon Dahdal

Indice

Indice	2
Prefazione	4
Introduzione	5
Capitolo 1. Internet of Things	6
1.1 Definizione	6
1.2 Storia	8
1.3 Panoramica e Architettura IoT	11
1.4 Aree principali dell'IoT	21
Capitolo 2. Strumenti e Tecnologie utilizzate	26
2.1 Protocollo HTTP	26
2.2 Servizi RESTful	33
2.3 Raspberry Pi 3	38
2.4 Mars Rover PiCar-B	41
2.5 Software utilizzati	50
2.5.1 Requests	51
2.5.2 Tkinter	51
2.5.3 Flask	52
2.5.4 Altre librerie	53
2.5.5 Software	53
Capitolo 3. Programmazione del sistema Mars Rover PiCar-B	55
3.1 Analisi e sperimentazioni nel concentrato	55
3.2 Programmazione del Mars Rover PiCar-B nel distribuito	64
Conclusioni e lavori futuri	74
Riferimenti Bibliografici	75
Riferimenti Figure	77

Prefazione

Si ringraziano il Prof. Giannelli Carlo e i Dottori Dahdal Simon e Collura Salvatore per avermi dato l'opportunità di utilizzare il bellissimo Mars Rover Picar-B.

Nonostante le difficoltà e i rallentamenti che si sono verificate nella mia vita nell'ultimo periodo universitario, hanno saputo essere sempre disponibili e aiutarmi nel modo migliore.

Un sentito ringraziamento va anche alla mia famiglia: a mia mamma Serenella, mia sorella Moira, mio papà Fabrizio, che hanno contribuito attivamente ed economicamente al perseguitamento del mio percorso di Laurea.

Introduzione

Le reti di dispositivi interessano ormai gli aspetti più disparati della società dell'era digitale. Grazie alla miniaturizzazione della tecnologia, ai costi decrescenti e alle capacità elaborative sempre più potenti, le tecnologie dell'informazione e della comunicazione (ICT, Information and Communication Technology) sono diventate, per eccellenza, le più pervasive. I profondi cambiamenti dell'attuale contesto socioeconomico, dovuti al proliferare delle ICT e delle reti digitali, hanno di fatto modificato le basi dell'organizzazione delle società avanzate (finanza, trasporti, sanità, educazione, cultura, turismo, relazioni e gruppi sociali) offrendo considerevoli prospettive di crescita anche per le società in via di sviluppo.

Oggi, infatti, si sta collegando alla rete Internet sempre di più non soltanto persone, ma anche oggetti e dispositivi, anche i più “impensabili”, ad esempio: borracce intelligenti connesse ad uno smartphone, che, ad intervalli regolari ricordano di bere più acqua a chi le usa; oppure, la “*Furbo Dog Camera*”, una fotocamera interattiva che aiuta ad addestrare un cane con un'app connessa che consente al padrone del cane di vedere, parlare e persino dargli bocconcini quando il padrone non è a casa [BOR].

Questi nuovi dispositivi e oggetti hanno fatto cambiare le architetture e le tecnologie, e si ha sempre più il bisogno di competenze differenti per supportare l'evoluzione della rete.

Ciò ha permesso all'*Internet of Things* di essere uno dei trend tecnologici più importanti del decennio [MPI]. Nello specifico, il presente elaborato riguarda lo studio e l'implementazione di un'applicazione per il monitoraggio e il controllo remoto della Mars Rover PiCar-B, un auto robotica intelligente ed open source, dall'aspetto del tutto simile ad un rover spaziale.

Si vuole ora fornire una panoramica generale sulla strutturazione della Tesi:

Nel Capitolo 1 sarà introdotto il concetto di IoT, come si è arrivati a tale fenomeno, le sue componenti e come esse interagiscono tra di loro. Infine saranno presentati alcuni casi d'uso in cui è applicato l'IoT.

Nel Capitolo 2 saranno invece presentate le numerose tecnologie hardware e software utilizzate, tra cui Python 3 e varie librerie open source per la parte di sviluppo e testing, e il Raspberry Pi come “punto d'incontro” tra il software e la macchina hardware”.

Nel Capitolo 3 si disquisirà sulle sperimentazioni effettuate sul sistema IoT in esame, ai fini di realizzare un'applicazione di controllo remoto della Mars Rover PiCar-B.

Capitolo 1. Internet of Things

La crescente espansione delle tecnologie informatiche, l'uso assai diffuso dell'elettronica di consumo nelle attività svolte nel tempo libero, i fenomeni di globalizzazione che aumentano la necessità di mobilità da parte dei lavoratori, e la *pervasività della rete Internet* hanno richiesto lo sviluppo di dispositivi di dimensioni sempre più ridotte e più ricchi in termini di funzionalità e capacità di elaborazione. [TRC01]

I modelli che hanno delineato l'evoluzione delle tecnologie ICT, basati sulla terna *reti-dati-applicazioni*, sono tutt'ora rappresentativi: le reti permettono le comunicazioni, i dati fluiscono attraverso le reti per divenire informazioni, le applicazioni o servizi fanno uso dell'informazione.

Prendendo come riferimento questa semplice schematizzazione è possibile sistematizzare una trattazione dei principali e più recenti fenomeni e tendenze tecnologiche che, nell'ultimo decennio, hanno caratterizzato le reti di dispositivi che governano la società dell'era digitale: mobile computing, social networking, big data, software apps, cloud, e, in particolare l'*Internet of Things*.

1.1 Definizione

In letteratura esistono molteplici definizioni che spiegano il fenomeno dell'Internet of Things, traducibile in italiano come: "*Internet delle cose*".

Una delle più complete è quella fornita dall'Institute of Electrical and Electronics Engineers (IEEE), che vede l'IoT come una rete di oggetti *identificabili univocamente e connessi ad Internet*, che possiedono capacità di *sensorizzazione* e di *attuazione*, quindi, di monitoraggio e controllo, e che possono essere *potenzialmente programmabili*.

Ciò significa che, da remoto vi è la possibilità di modificare il software che risiede all'interno degli oggetti, o che interagisce direttamente con essi. Il livello di programmabilità di un dispositivo può variare molto: andando dalla possibilità di settare parametri fondamentali, a

quella di modificare effettivamente, in maniera ampia il comportamento di tale oggetto, ad esempio effettuandone remotamente l'update del firmware.

Grazie all'IoT è possibile *inviare comandi* e *collezionare informazioni* eterogenee, legate allo *stato* degli oggetti, che può variare nel tempo.

Se un dispositivo riesce ad eseguire queste ultime due operazioni da ovunque, ogni qualvolta se ne abbia la necessità e da qualunque altro oggetto remoto (sia esso un servizio, un'applicazione web, un'applicazione mobile, etc.), allora gli si può attribuire la caratteristica *dell'ubiquità*.

Inoltre, gli oggetti devono poter comunicare tra di loro, interconnettendosi non soltanto alla stessa sottorete (non essendo dunque vincolati ad una limitata area geografica), bensì potendosi *collegare anche ad Internet*, realizzando, di fatto, una Rete mondiale di Cose.

I dispositivi possono altresì avere un'*intelligenza embedded* (non è però questo un requisito obbligatorio), ossia essere dotati di un software più o meno avanzato, che consenta loro di prendere delle decisioni anche localmente.

A titolo di esempio, se si pensi ad una caldaia avente un sensore di temperatura che invia periodicamente i dati sulla temperatura remotamente: così facendo, si avrà un monitoraggio continuo sull'andamento della caldaia. Tuttavia, nel caso in cui la temperatura dell'acqua superi i 90°, nulla impedisce che la caldaia possa avere al suo interno un software che, in modo automatico, senza il bisogno di inviare informazioni remotamente, *decida localmente* di spegnere la caldaia poiché, al di sopra dei 90°, si potrebbe presentare una situazione di potenziale pericolo.

Ancora, le cose devono anche essere in grado di comunicare in maniera interoperabile.

L'*interoperabilità* si basa tipicamente su alcuni *standard*. Quindi, la comunicazione degli oggetti non è abilitata da soluzioni pensate da zero per ciascuno oggetto o per ciascuna soluzione IoT, ma è supportata e basata sull'uso di protocolli ben definiti. In tal modo, grazie all'adozione di tali standard è certamente più agevole non solo realizzare soluzioni, bensì mantenerle e farle evolvere in un secondo momento [IEE],[UFE].

Tutti gli oggetti che sono in possesso delle proprietà e dei requisiti menzionati fino ad ora, possono essere più propriamente qualificati come “*smart object*” (oggetti intelligenti), i quali acquisiscono una propria identità digitale, un ruolo attivo, che gli consente di comunicare con gli altri device collegati alla rete e di poter fornire servizi agli utenti.

In questi termini, non è l'utente che decide di dialogare con l'oggetto, ma è l'ambiente che può diventare intelligente ed autonomo grazie al fatto che tutti questi oggetti possono parlarsi tra di loro.

Da notare è che le entità connesse non devono essere inanimate, ma possono anche essere animali (cani, gatti etc.), oppure persone dotate di transponder o impianti con le giuste caratteristiche [UUD].

1.2 Storia

Già nel 1926 si iniziavano a teorizzare le prime idee embrionali di IoT. In quello stesso anno, Nikola Tesla, in un'intervista, affermò quanto segue:

“Quando il wireless sarà applicato perfettamente, l'intera Terra sarà convertita in un enorme cervello” [NTS].

Anche lo stesso Internet, emerso dal “progetto” ARPANET alla fine degli anni ‘70 fu pensato per collegare oggetti, corrispondenti a quell'epoca a grandi armadi in grado di trasmettere pochi bit.

In quegli anni vi erano le prime connessioni satellitari tra nodi in America e nodi in Europa, e, i nodi connessi erano una qualche decina.

Questo dimostra come fosse presente, fin dall'inizio, l'idea di distribuzione, del voler collegare oggetti differenti con tecnologie differenti [MPI],[WIK02].

Fino ad allora, l'ICT era incentrata soprattutto sull'elettronica. L'avvento dell'informatica di massa, verificatosi intorno agli inizi degli anni 80 che vide la diffusione dei primi home computer sul mercato (ad esempio Commodore VIC 20, Commodore 64, Sinclair ZX Spectrum o Sinclair QL) permise ai tanti “affamati” di informatica e futuro di esprimere la loro creatività attraverso l'uso e la produzione di software. A quel punto, in un certo senso, le strade degli elettronici e quelle degli informatici iniziarono a dividersi [CAM] .

A poco a poco, il termine *“Rete di Calcolatori”* cominciava ad essere datato, visto il crescente numero di dispositivi non tradizionali collegati a Internet.

Per cercare di interpretare l' onnipresenza di tutti quegli oggetti in rete, inizialmente si utilizzarono espressioni del tipo: “*Pervasive Computing*”, “*Ubiquitous Computing*”, “*Mobilità del codice e dei dati*”. Poco tempo dopo, nel settembre del 1985 Peter T. Lewis coniò il termine “*Internet of Things*”, utilizzandolo in una conferenza a Washington DC [UUD].

L'IoT fu effettivamente applicato per la prima volta nel 1999, quando Kevin Ashton, mentre lavorava per lanciare un nuovo marchio di cosmetici colorati presso l'azienda Procter & Gamble, si accorse che il colore di rossetto più popolare sembrava non essere mai disponibile e reperibile in nessuno dei negozi.

Si domandò pertanto dove si trovassero quei rossetti e cosa gli fosse capitato, ma nessuno fu in grado di dargli una risposta. Analizzando a fondo la questione, scoprì che l'errore era a livello umano: in tutti i negozi, l'inventario era gestito attraverso dei lettori di codice a barre da parte di un operatore umano. Come spesso accade quando grandi quantità di dati sono gestite da un individuo, tali dati risultavano errati.

Ashton volle trovare un modo più completo per tracciare i prodotti e, dal momento che in U.K. si stavano svolgendo i primi esperimenti su un innovativo chip a radiofrequenze, egli pensò all'idea di installare quello stesso chip sulla confezione di quel rossetto irreperibile.

Ashton manifestò un ampio interesse all'argomento e co-fondò l'Auto-ID center presso il MIT, studiando e perfezionando la tecnologia dei tag RFID (identificazione wireless di dispositivi a radiofrequenza) e le sue applicazioni nell'ambito dello *smart packaging*.

Questa stessa idea la applicò poi al contesto della *supply chain*, ossia alla capacità di tracciare gli oggetti che sono movimentati, cioè spostati di volta in volta, partendo dal produttore, passando poi per il magazzino, per le consegne, fino ad arrivare all'utente finale.

Da questa vicenda, Ashton affermò:

“*Se avessimo computer che sapessero tutto quello che c'è da sapere sulle cose, utilizzando i dati raccolti senza il nostro aiuto, saremmo in grado di tracciare e contare tutto e ridurre notevolmente sprechi, perdite e costi. Sapremo quando le cose devono essere sostituite, riparate o richiamate e se sono fresche o hanno superato il loro meglio.*

Dobbiamo potenziare i computer con i propri mezzi di raccolta delle informazioni, in modo che possano vedere, ascoltare e annusare il mondo da soli, in tutta la sua gloria casuale. La tecnologia RFID e dei sensori consente ai computer di osservare, identificare e comprendere il mondo, senza i limiti dei dati immessi dall'uomo ” [UFE],[ASH01]

In aumento era anche il numero di servizi connessi (offerti da Provider quali Google.com, Yahoo.com, Facebook.com, Bing.com etc.).

Al contempo, la barriera di ingresso rispetto alla creazione di nuove soluzioni e servizi si abbassò, portando nel corso degli anni alla creazione di nuove startup e nuovi servizi.

Inoltre, con il passare degli anni la rivoluzione del mondo dei dispositivi mobile cambiò il modo con il quale ci si interfacciava ai servizi. Considerando Google, questi 15 anni fa aveva una modalità di accesso principalmente da Desktop, mentre ora avviene principalmente da mobile.

In più, gli smartphone iniziavano a creare attorno a loro un piccolo ecosistema, abbastanza frammentato, quello dei primi oggetti connessi, ad esempio occhiali intelligenti, smartwatch, dispositivi per il tracciamento delle attività [MPI].

Infine, nell'ultimo decennio, la rivoluzionaria tecnologia delle *schede per microcontroller* sta riunendo le strade degli elettronici e degli informatici.

Un microcontrollore è un piccolo computer in un singolo chip: in un unico circuito integrato sono così presenti il microprocessore, la memoria le periferiche di input-output. Apposite schede ospitano il microcontroller e lo connettono al mondo esterno attraverso varie tipologie di porte.

Le porte presenti su tali schede consentono di collegare tutta una serie di *sensori* per misurare caratteristiche fisiche dell'ambiente circostante, ad esempio temperatura o luminosità.

Il microcontroller può dunque essere programmato per eseguire specifiche operazioni al modificarsi di queste proprietà fisiche.

Il salto di qualità che si realizza con queste schede è la possibilità di tradurre queste situazioni in modifiche fisiche vere e proprie, che coinvolgono il mondo reale.

Ad esempio, è possibile far scattare dei relè per accendere delle luci o per azionare motori o qualsiasi altro dispositivo elettronico collegato.

Tali meccanismi prendono il nome di *attuatori*. Tutto ciò rientra in quello che viene definito come *physical computing*. Quando poi questi dispositivi riescono a interagire e a comunicare tra loro si inizia a parlare, per l'appunto, di IoT.

Ecco che si arriva dunque alla nascita di questa rete, quella dell'Internet delle Cose, in cui i vari oggetti intelligenti, sia fisici che virtuali, acquisiscono una sorta di consapevolezza del mondo reale comunicando informazioni tra di essi [CAM].

Il mondo elettronico traccia una mappa di quello reale, dando un'identità elettronica alle cose e ai luoghi dell'ambiente fisico.

1.3 Panoramica e Architettura IoT

I dispositivi dell'Internet of Things sono progettati e realizzati per assolvere ad alcune importanti funzioni.

In primo luogo, devono poter realizzare due azioni “simili” ma distinte, ovvero:

Monitoraggio: ottenere informazioni dai dispositivi e fornirle remotamente, verificando lo stato corrente del dispositivo da remoto.

Controllo: inviare comandi da remoto ai dispositivi fisici, affinché li applichino per modificare il loro comportamento, riconfigurando di fatto lo stato dei device.

Occorrerà anche la possibilità di trasferire le informazioni sullo stato e i messaggi di controllo dai dispositivi a delle locazioni remote, che possono essere data center privati on-premise o cloud pubblici.

Arrivate in un luogo remoto (ad esempio nel data center di un'azienda), sarà necessario *aggregare, storizzare e persistere in un database* tali informazioni.

Inoltre, se le informazioni hanno utilità solamente se ottenute in real time, allora sarà opportuno che esse siano anche *visualizzabili* da remoto.

Dovrà essere offerta la possibilità di *Analizzare* queste informazioni, eventualmente in un secondo momento e in modo asincrono, anche per comprendere meglio lo stato dei dispositivi, allo scopo di poter effettuare ad esempio delle analisi comparative tra lo stato di device simili, ma che sono utilizzati in modo diverso.

L'obiettivo finale è quello, comunque, di *prendere delle decisioni*, sulla base delle informazioni acquisite dai dispositivi.

Le decisioni potranno essere *Human-assisted*: l'operatore umano, sulla base di queste informazioni, prende delle decisioni che eventualmente possono scaturire e portare a dei messaggi di controllo sui dispositivi, oppure relative alle strategie di business che si vogliono perseguire.

Grazie all'IoT, le decisioni possono essere adottate anche in *modo automatico* [UFE].

Andando più nel dettaglio, è doveroso precisare alcuni importanti concetti e introdurne altri, propedeutici alla comprensione del funzionamento di un'Architettura IoT, vista la grande eterogeneità di dispositivi in essa presenti.

Un **Sensore** è un dispositivo meccanico, elettronico o chimico, installato in apparecchiature o meccanismi che *rileva* i valori di una grandezza o fenomeno fisico, producendo un segnale in uscita e trasmettendolo ad un sistema di misurazione o di controllo.

Dunque, i sensori fanno sì che l'oggetto possa relazionarsi con l'ambiente in cui si trova, per rilevare dati utili al suo funzionamento e adattarsi di conseguenza.

Si pensi, ad esempio, a un impianto di illuminazione pubblica che, sfruttando i dati di sensori stradali, si accende e si spegne al passaggio di un'automobile.

Esiste una grandissima varietà di sensori, ad esempio: termometri, rivelatori di luminosità, barometri, bussole, giroscopi, accelerometri, infrarossi, etc. [UUD],[UFE],[TRC02].

Invece, un **Attuatore** è un device in grado di *modificare* lo stato degli oggetti, trasformando una decisione automatica di comando, elaborata da una scheda elettronica di controllo, in un'azione fisica sul processo oggetto della regolazione.

Permette quindi agli utenti di monitorare a distanza il funzionamento di un dispositivo IoT.

Esempi di attuatori possono essere: dispositivi di riscaldamento o raffreddamento, altoparlanti, luci, display, motori, oppure sistemi SCADA (Supervisory Control And Data Acquisition) con accesso remoto. Questi ultimi, abilitati già dagli anni '70, consentono di controllare il funzionamento di un macchinario o di un impianto industriale anche trovandosi a centinaia di chilometri di distanza; tuttavia, risultano essere soluzioni software proprietarie, con scarsa integrazione con servizi di terze parti dal momento che non si basano su tecnologie e protocolli standard; il loro utilizzo è in genere giustificato in sistemi mission critical e in impianti costosi [UFE], [TRC03].

Si introduce ora il concetto di **Cloud Computing**, strumento quest'ultimo che consente agli utenti che lo utilizzano (clienti finali) di poter usufruire di risorse (applicazioni e dati) situate in data center remoti e distribuiti, all'interno della propria macchina locale.

Il Cloud Computing permette l'accesso o l'attivazione on-demand (al momento del bisogno) di alcuni servizi, quali: uso di tecnologie e risorse informatiche (es. server), spazio per archiviazione dati, software applicativi e capacità di calcolo computazionale.

Questi servizi sono erogati da un Cloud Provider a diversi livelli di astrazione: ciò significa che la gestione di alcuni di essi può essere delegata parzialmente o interamente all'utente che li richiede, a seconda della soluzione che ha deciso di adottare (IaaS, PaaS, SaaS).

Il Cloud computing è oggi indispensabile poiché presenta numerosi vantaggi, come:

- *eliminazione dei costi* di acquisto di hardware, software, creazione e gestione dell'infrastruttura. Basterà pagare un canone d'affitto;
- *disponibilità*: i grandi provider garantiscono livelli di disponibilità molto elevati rispetto alle soluzioni tradizionali, diminuendo il rischio di crash;
- *strumenti di integrazione*: poter accedere ai servizi non solo tramite UI, ma anche tramite servizi remoti basati su SOAP, WSDL, *ReST* etc., per facilitare l'integrazione di componenti (anche legacy).
- *accesso da qualunque dispositivo* (PC, tablet, cellulare), mediante uno User Agent.
- *sicurezza*: i cloud service providers hanno regole, politiche, regolamenti e controlli che rafforzano la sicurezza dei clienti finali, proteggendo dati, applicazioni e infrastruttura da potenziali minacce.
- *affidabilità*: garantisce la continuità operativa offrendo ad esempio il backup e la ridondanza dei dati, duplicandoli in più siti ridondanti sulla rete del provider di servizi cloud.
- *scalabilità*: i servizi di cloud computing includono la capacità di scalare in modo elastico e flessibile. Potendo quindi aumentare o diminuire la quantità di risorse IT, ovvero più o meno potenza di calcolo, storage, larghezza di banda secondo i requisiti

Tuttavia nasconde anche svantaggi. La possibile *mancanza di connessione ad Internet* può interrompere l'accesso ai servizi; inoltre, il *controllo* sui propri dati e applicazioni non è diretto, ma è gestito da un Data center di terze parti; *privacy* e *sicurezza*: i dati non sono più entro il perimetro aziendale, ma sono in mano a terze parti, con il rischio ad esempio di eventuali furti di segreti industriali piuttosto che sanitari.

Ad ogni modo, il Cloud Computing è molto utilizzato, in quanto i dati sono raccolti dai diversi sensori o inviati agli attuatori e questi, tramite Internet vengono salvati all'interno del Cloud, rendendo accessibili queste informazioni da ovunque nel mondo.

Permette anche di ottenere soluzioni scalabili, ad esempio, consentendo l'espansione del numero di sensori e/o attuatori presenti in una casa [TRC04], [UFE].

L'ultimo elemento da considerare è il **Gateway**, un'entità hardware e software avente l'obiettivo di fungere da intermediario sia per comunicazione device to device, sia tra

sensori/attuatori e le piattaforme Cloud, attraverso la rete Internet. La presenza di un gateway, che evita una comunicazione diretta tra dispositivo e cloud, è fondamentale giacché alcuni oggetti potrebbero non supportare il protocollo IP (privi di connettività ad internet), oppure non avere sufficienti capacità hardware o software per scambiarsi dati in rete. Un IoT gateway è uno strumento estremamente intelligente, che può svolgere molte funzioni, tra cui:

- *Protocol translation*: trasformare le informazioni e il protocollo utilizzato dagli oggetti in un formato fruibile via Internet. Poiché spesso, i dispositivi fisici comunicano con protocolli proprietari o comunque industriali, realizzati per ambienti chiusi e sicuri e per ambienti industriali. Sono protocolli nati negli anni '70, '80 estremamente utili e performanti in reti locali, che però non sono stati pensati per elevate latenze che ci sono su Internet, e neanche eventualmente per bande limitate o per ambienti in cui si perdono molti pacchetti.

Dunque, il Gateway deve prelevare dati e comunicare con i dispositivi con un protocollo specifico industriale, eventualmente anche proprietario e non standard; dall'altra parte deve comunicare col Cloud, inviando e ricevendo comandi con i protocolli tipici utilizzati nella rete Internet.

- Servizi di Sicurezza, *Geolocalizzazione* e *Billing*¹;
- *Pre-processamento dei dati* già all'interno del gateway: bufferizzazione, efficientamento, aggregazione e filtraggio sui dati [UFE], [IOT01].

Si può ora parlare di Architettura IoT, per capire come si “orchestrano” tra loro gli elementi precedentemente menzionati, asserendo fin da subito che non esiste un’Architettura consolidata per l’IoT. Per facilità di trattazione, si è scelto di analizzare l’Architettura IoT a 3 layer (se ne poteva scegliere arbitrariamente una a 5 o 7 strati).

1. **Livello di Percezione (Things)**: E’ lo strato fisico, che include un *insieme eterogeneo di device*, tra cui sensori che raccolgono varie quantità di dati, dispositivi perimetrali, attuatori che interagiscono l’ambiente circostante.
2. **Livello di Rete (Gateways)**: E’ il layer responsabile della connessione ad altre cose intelligenti, dispositivi di rete e server. Le sue caratteristiche vengono utilizzate anche per la trasmissione e l’elaborazione dei dati dei sensori, sfruttando le succitate proprietà degli *IoT gateway*, situati in prossimità degli oggetti.

¹Il billing consiste nel tracciare la quantità di informazioni prodotte dai device fisici allo scopo di richiedere un pagamento all’utente in relazione alle risorse effettivamente utilizzate.

I gateway si interfacciano poi con le componenti del Cloud, sfruttando alcuni *meccanismi di comunicazione*, implementati da opportuni *protocolli di comunicazione*.

3. **Livello Applicazione (Cloud/Servers)**: E' il livello con cui l'utente interagisce, responsabile della fornitura di servizi specifici dell'applicazione all'utente: piattaforme di IoT, database, servizi di analytics, tutto ciò che interessa la storicizzazione delle informazioni a lungo periodo, la loro gestione ed analisi ed eventualmente il dashboarding dei dati [IOT03],[IOT04].

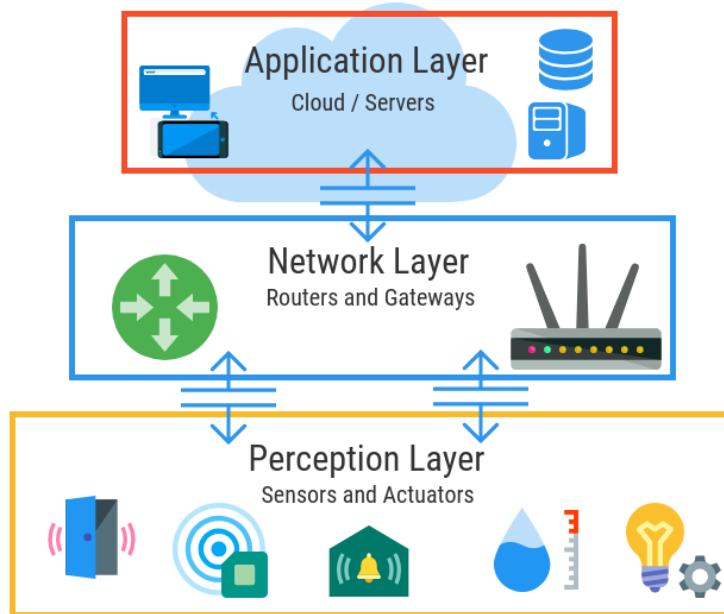


Figura 1.3a. Gli strati di un'architettura IoT a 3 livelli.

Ciò che si riesce a fare oggi grazie ai servizi di Cloud computing, era impensabile 5 o 6 anni fa. Tuttavia, ci sono scenari nel mondo dell'IoT dove la dipendenza dal Cloud può creare problemi legati alla latenza di comunicazione tra oggetto smart e cloud.

Si è iniziato dunque a pensare a nuove soluzioni (Figura 1.3b).

L'Architettura IoT che si sta sviluppando, altamente distribuita e avente il cloud come elemento di riferimento, poco a poco sta infatti portando intelligenza, capacità computazionale e di storage verso la periferia, più vicino alla sorgente, quindi verso i dispositivi.

E' possibile realizzare ciò dotando i Gateway di una capacità di effettuare analisi sui dati molto più avanzate, diminuendo di fatto la latenza. Questo approccio prende il nome di *Edge Computing*.

Di recente, con l'obiettivo di migliorare il comportamento degli oggetti ottenendo servizi intelligenti sempre più affidabili e in tempo reale, il Cloud Computing, si è evoluto nel *Fog Computing*, che prevede una stratificazione del numero di dispositivi, sempre più vicini al Cloud, sempre più lontani dal dispositivo e sempre più potenti, in modo da distribuire la computazione a diversi livelli del percorso tra oggetto e Cloud. Ciò risponde ad esempio alle esigenze interne di una fabbrica, oppure di un'industria [MPI].

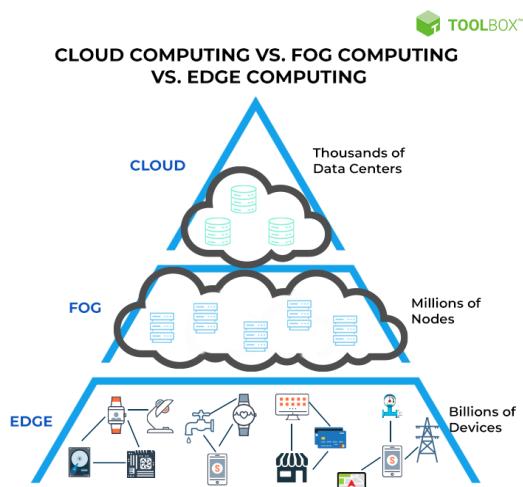


Figura 1.3b. Concettualizzazione dell'evoluzione del Cloud Computing.

Il **mercato dell'IoT** conta ormai il miliardo di dispositivi connessi, in tanti settori, tra i quali: i veicoli connessi o i veicoli autonomi; il mercato delle Smart Home, le Smart City; l'IoT Industriale (come portare intelligenza e connessione nell'ambito industriale); la sanità (possibilità di monitorare apparati all'interno di ospedali o anche da remoto, ad esempio il monitoraggio a casa).

Un'analisi condotta da McKinsey pronostica che nel 2025 il mercato IoT sarà stimabile in un valore pari a 6.200 miliardi di dollari [UUD].

Un altro dato interessante riguarda i Data center: questi, negli ultimi 15 anni hanno avuto un'espansione enorme, dal 2008 con 16 miliardi di dollari al 2016 con ben 171 miliardi di dollari [UFE].

L'IoT è senz'altro essenziale per il business perchè offre alle aziende uno sguardo in tempo reale su come funzionano realmente i loro sistemi, fornendo informazioni dettagliate su tutto, dalle prestazioni delle macchine alla catena di approvvigionamento e alle operazioni logistiche; consente inoltre alle aziende di automatizzare i processi e ridurre i costi di manodopera. Inoltre, riduce gli sprechi e migliora l'erogazione dei servizi, rendendo meno costosa la produzione e la consegna delle merci, oltre a offrire trasparenza nelle transazioni dei clienti [IOT02].

Alcuni dei possibili principali *portatori di interessi* legati al business dell'IoT sono aziende di telecomunicazioni, di soluzioni per il networking; aziende hi-tech (ossia tutti i produttori di chip, dispositivi, sensori, batterie). Altri stakeholder sono: i fornitori di servizi di Cloud Storage, sviluppatori software, aziende operanti nell'ambito dell'automazione, della manutenzione predittiva per l'Industria 4.0, grandi player, come Google, Apple, Microsoft, IBM, Amazon, etc. [UUID].

L'IoT ha portato, direttamente o indirettamente ad una frammentazione del mondo dell'informatica: l'ingegnere informatico non può più pensare di creare in autonomia un intero sistema IoT, dal device al sistema intelligente. Vi è dunque la *possibilità di affrontare progetti con competenze differenti*, lavorando in team per poter creare uno o più sistemi IoT che dialogino in modo innovativo.

Nasce la figura professionale del *Data Analyst*, necessaria per la parte di processing dei dati, e si arricchisce anche l'ambito del Machine Learning [MPI].

Una problematica essenziale dell'ecosistema IoT è quella del *dialogo tra i dispositivi intelligenti*.

Le tipiche **tecnologie di comunicazioni wireless** all'interno del mondo IoT sono: Wifi, ZigBee, Bluetooth, le *reti cellulari* 3G, 4G e, più recentemente, il 5G. Una tradizionale tecnologia *wired* è invece rappresentata da Ethernet.

I dispositivi IoT possono connettersi localmente tramite “reti non IP”, che consumano meno energia, e si possono collegare ad Internet tramite un gateway intelligente.

I canali di comunicazione non IP come Bluetooth, RFID e NFC sono abbastanza diffusi e giovano di un minor consumo di energia, ma hanno di contro una banda molto limitata. (fino a pochi metri). Pertanto, le loro applicazioni sono limitate a piccole reti personali (PAN) utilizzate ad esempio dai dispositivi indossabili connessi agli smartphone.

Una delle più popolari soluzioni per aumentare la portata delle reti locali è 6LoWPAN (IPv6 su reti personali wireless a bassa potenza), che incorpora IPv6 con reti di area personale a bassa potenza. Molte volte, i dati provenienti da un singolo sensore non sono utili per monitorare grandi aree e attività complesse. Nodi di sensori diversi devono interagire tra loro in modalità wireless. Per risolvere i problemi di portata delle tecnologie non IP, si possono creare reti di sensori wireless (WSN) composte da decine o migliaia di “nodi di sensori” collegati tramite tecnologie wireless, possedente una qualche topologia (stella, mesh, etc).

Le WSN raccolgono dati sull'ambiente e li comunicano ai gateway che a loro volta trasmettono le informazioni al cloud tramite Internet [UFE],[IOT03].

Quelli appena citati sono tutti protocolli di comunicazione per trasportare bit di informazioni e per creare dei link di comunicazione. Al di sopra di essi, è necessario adottare dei protocolli per lo scambio di dati o, più in generale, di *messaggi* (in grado di portare *informazioni strutturate con una certa sintassi e semantica*).

I protocolli per lo scambio di dati si differenziano in relazione all'overhead che introducono, all'affidabilità con cui trasportano le informazioni e al *modello di comunicazione* che adottano per scambiare le informazioni.

La scelta del protocollo per lo scambio dei dati è importante, ed è realizzata in modo indipendente dalla tecnologia di comunicazione utilizzata per realizzare link di comunicazione. Risulta altresì importante valutare quale delle entità del sistema IoT debba iniziare la comunicazione. In genere sono previste due tipi di interazioni: *push* e *pull*.

In modalità **Push**, i dispositivi che producono le informazioni (o i gateway che interagiscono con tali dispositivi) avviano la comunicazione verso il server e decidono di inviarli le informazioni in modo autonomo e periodico.

Invece, in modalità **Pull** è il Server (o meglio, la *Piattaforma IoT*) che domanda al dispositivo (che ha prodotto l'informazione) o al gateway di fornirgli le informazioni di cui necessita.

I due principali modelli di comunicazione possono essere *Request/Response* oppure *Publish/Subscribe*.

Il modello di comunicazione **Request-Response** consiste di due entità: *Client* e *Server*.

Un'applicazione client richiede una particolare risorsa inviando una richiesta ad un'applicazione server. Il server, al ricevimento della richiesta decide come rispondere, recupera la risorsa ed invia una risposta al client.

Nel modello in esame la comunicazione è *diretta*. Inoltre, client e server sono *accoppiati nel*

tempo: se il client vuole comunicare qualcosa, ma il server in quel preciso momento non è attivo, la comunicazione non può avvenire, e il client deve riprovare ad effettuare la comunicazione in un secondo momento. Tale modello prevede che sia sempre il client ad iniziare la comunicazione.

Tipici protocolli basati su questo paradigma sono *CoAP* (Constrained Application Protocol) ed *HTTP*; quest'ultimo sarà approfondito nel capitolo successivo.

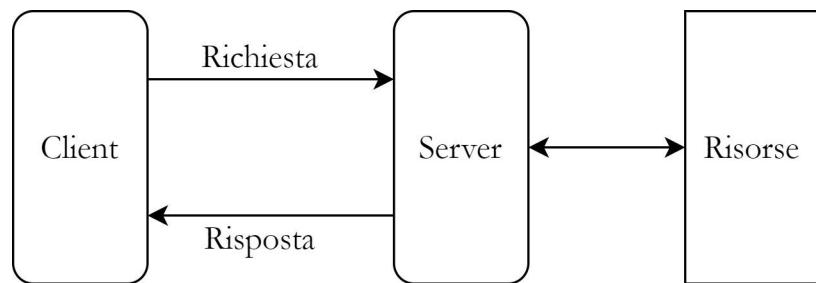


Figura 1.3c. Il funzionamento dell'architettura Client/Server

Il modello **Publish-Subscribe** coinvolge 3 soggetti: *Publishers*, *Subscribers* e *Brokers*.

I **Publishers** rappresentano i mittenti del messaggio; essi producono dei dati che sono relativi a vari *Topics*. I **Subscribers** sono i riceventi del messaggio, coloro che consumano i dati che hanno richiesto. Lo stesso messaggio inviato da un publisher è ricevuto da tutti i nodi Subscriber che sono interessati a tale messaggio. Questi messaggi sono associati ad un determinato argomento (*topic*) e i Subscribers dichiarano a quali topic sono interessati.

Nel mezzo vi sono i **Brokers**, che sono i gestori dei dati; questi rappresentano il sistema di messaggistica, avente il compito di tener traccia di quali subscriber sono attivi e di quali topic ciascun subscriber è interessato. I Publishers e i Subscribers sono collegati tra di loro grazie ai Brokers.

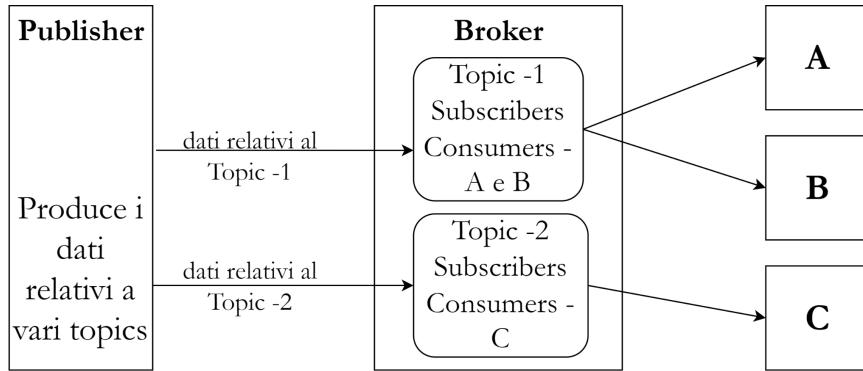


Fig 1.3d. Esempio di funzionamento del modello Pub/Sub. Un Publisher produce dati relativi a 2 topics: “Topic -1” e “Topic -2”. Il Broker forma dei gruppi in base ai Topic, successivamente, identifica dei consumatori (Subscribers: A, B e C) che si vogliono iscrivere ad un particolare Topic ed invia loro i dati.

Il Publisher invia il messaggio indicando anche il topic e il Broker identifica quali subscriber sono interessati al messaggio che deve gestire, e invia il messaggio a tutti i Subscriber che sono interessati a quel topic. Vi è una connessione uno a molti. Eventualmente, un messaggio potrebbe anche non essere consegnato ad alcun subscriber.

I Publishers non sono a conoscenza dei Subscribers, non c’è un contatto diretto tra Publisher e Subscribers. Con questo modello l’interazione è solitamente di tipo push.

I principali protocolli che implementano il pattern Publish/Subscribe sono: *MQTT* (Message Queue Telemetry Transport), *AMQP* (Advanced Message Queuing Protocol), *DDS* (Data Distribution Service).

I molteplici protocolli per lo scambio di dati e messaggi hanno capacità e performance diverse dall’uno all’altro e la selezione del più adatto da utilizzare dipende fortemente dai requisiti applicativi che si hanno e anche dai vincoli del sistema che si vuole andare a realizzare.

Ad esempio, CoAP può essere usato nel caso in cui dispositivi che devono comunicare abbiano risorse estremamente limitate, mentre DDS è utilizzato per applicazioni che necessitano performance molto elevate.

In realtà, nella maggior parte dei casi d’uso, lo stile più utilizzato per il modello Richiesta/Risposta è *REST*, invece MQTT e AMQP sono i protocolli di maggior successo per il Pub/Sub [UFE], [IOT01].

1.4 Aree principali dell'IoT

L'Internet delle Cose possiede un ampio spettro di applicazioni: dalle applicazioni industriali (processi produttivi), alla logistica e all'infomobilità, fino all'efficienza energetica, all'assistenza remota e alla tutela ambientale. A seguire ne saranno descritti alcuni.

Smart Wearables (Figura 1.4a): sono tutti quegli oggetti digitali e collegati con altri dispositivi (formando delle reti BAN), comunque connessi ad Internet, per fornire informazioni e manovrare lo stato degli utenti. Dunque, qualunque oggetto "*indossabile*" dotato di un indirizzo IP e di un trasmettitore può essere considerato un wearable tech, ad esempio: *Smartwatch*, fitness trackers, smart t-shirts, telecamere indossabili, occhiali smart, pannolini intelligenti, patch elettronici per la pelle etc.

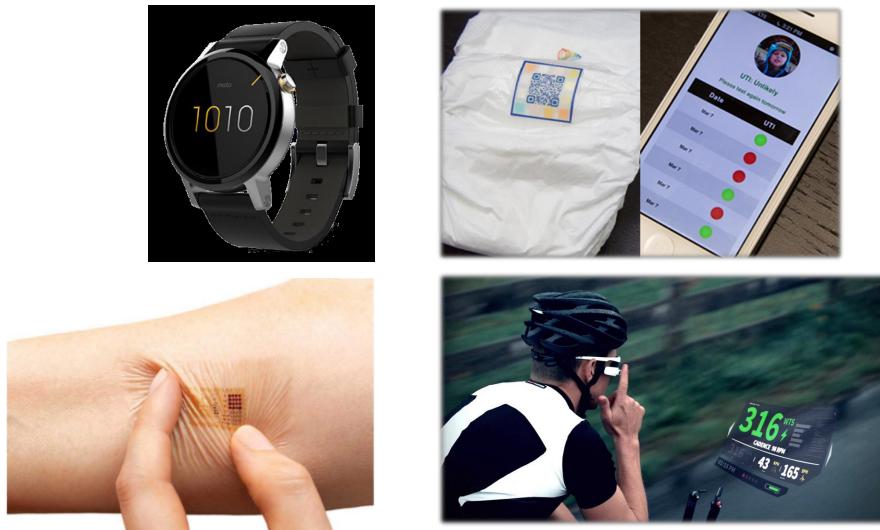


Figura 1.4a. Alcuni Smart Wearables e le loro applicazioni nella vita quotidiana.

Smart Home e Smart Cities (Figura 1.4b): indicano la possibilità di poter monitorare lo stato di un edificio o di una casa, ad esempio i suoi consumi: un edificio smart potrebbe ridurre i costi energetici usando sensori che rilevano quanti occupanti ci sono in una stanza. Comprende anche tutto ciò che ha a che fare con la *Domotica*: le Smart TV; l'accensione e lo spegnimento di luci intelligenti; digitalizzare l'interazione con forni, frigoriferi, caldaie, lavatrici, termostati, rilevatori di fumo, finestre, sistemi di allarme etc. [IOT2]

In questo ambito vi sono molte sfaccettature rispetto al tema della sicurezza, al controllo degli accessi e alla gestione delle politiche di interazione con questi dispositivi.

Quella delle *Smart Cities* è una realtà caratterizzata da moltissimi sottosistemi, che include la gestione della mobilità, dei veicoli, dei lampioni e parcheggi intelligenti, della raccolta dei rifiuti, dell'energia [MPI].



Figura 1.4b. A sinistra: l'ubiquità di dispositivi che compongono una Smart Home. A destra: lo schema delle attività presenti in una tipica Smart City. ([Immagine](#))



Figura 1.4c. Smart Agriculture e Droni

Smart Agriculture e Droni (Figura 1.4c): possibilità di sensorizzare e digitalizzare mezzi agricoli come i trattori, che sono sempre di più dei mezzi estremamente digitalizzati e molto avanzati tecnologicamente; oppure di accendere/spiegnere gli irrigatori da remoto. I sensori possono raccogliere dati che permettono di monitorare: le precipitazioni, la temperatura e l'umidità del terreno e dell'aria in real time, il contenuto del suolo, nonché altri fattori che aiuterebbero ad automatizzare le tecniche di coltivazione.

E' inoltre possibile sfruttare il sistema aeromobile a pilotaggio remoto offerto dai *droni* per poter: effettuare riprese aeree; esplorare aree pericolose; trasportare oggetti; raccogliere dati ambientali; sorvegliare e monitorare aree protette [UFE].

Industrial Internet of Things (IIoT, Figura 1.4d). È un sottoinsieme dell'IoT che si focalizza su diversi segmenti di mercato industriali.

Uno fra questi è l' *industria manifatturiera*, con la possibilità di effettuare *predictive*

maintenance (manutenzione predittiva degli oggetti) e di utilizzare la *robotica*, e l'*automazione*.

Trova applicazione anche nell'ambito degli *Smart Grid* (contatori intelligenti), o sugli impianti per la produzione e gestione di energia elettrica (*Smart Energy*).

Grazie all'IoT, le aziende di trasporti di merci e passeggeri hanno la possibilità di monitorare in modo migliore gli spostamenti e la gestione dei propri veicoli (*Fleet management*).

Riguarda inoltre aspetti di *logistica*, prevedendo la possibilità di ottimizzare la *supply chain*, partendo dalla produzione, il magazzino e la gestione delle informazioni anche considerando altri fornitori e altri clienti.

Nel mondo industriale, quindi, l'IoT è utilizzato per permettere a dispositivi intelligenti, unità di processamento delle informazioni industriali e alle reti di interagire con l'ambiente esterno e con l'ambiente locale, con l'obiettivo di *generare informazioni* e di *strutturarle* in modo che sia facile *gestirle* e *analizzarle* per poter *creare nuova conoscenza*. In tal modo, è possibile *migliorare l'efficienza operazionale e la produttività* all'interno delle aziende e delle linee di produzione, poiché, grazie a queste informazioni è possibile identificare un miglior settaggio dei dispositivi industriali. È anche possibile offrire alle aziende nuovi servizi a valore aggiunto, ad esempio fornendo non solo i macchinari per produrre gli oggetti, bensì anche altri servizi, come la manutenzione e la gestione remota degli oggetti che sono stati venduti.

Tra i possibili usi dell'IoT nel mondo industriale, potendo ottenere maggiore informazione (dunque maggiore conoscenza), è possibile *diminuire i casi di rottura dei dispositivi*, ottimizzarne la manutenzione, al fine di migliorare la produzione ed evitare, ad esempio, che a causa della rottura inaspettata di un macchinario sia necessario bloccare la produzione per un certo lasso di tempo.

Da un punto di vista del business di un'azienda, l'IoT, può essere uno strumento di *decision making*, ad esempio per andare a modificare la strategia aziendale in relazione ai prodotti che vende, monitorare ad esempio se e quanto i propri asset vengono utilizzati dagli utenti, quali funzionalità tra quelle offerte sono più utilizzate; è dunque possibile andare a migliorare i propri prodotti in modo da tenere in considerazione le vere esigenze degli utenti.

Al momento, il settore di applicazione più ricettivo è la *Connected Industry*, ovvero la possibilità di connettere tra di loro, specialmente in ambito manifatturiero, macchinari diversi eventualmente anche prodotti e venduti da aziende diverse.

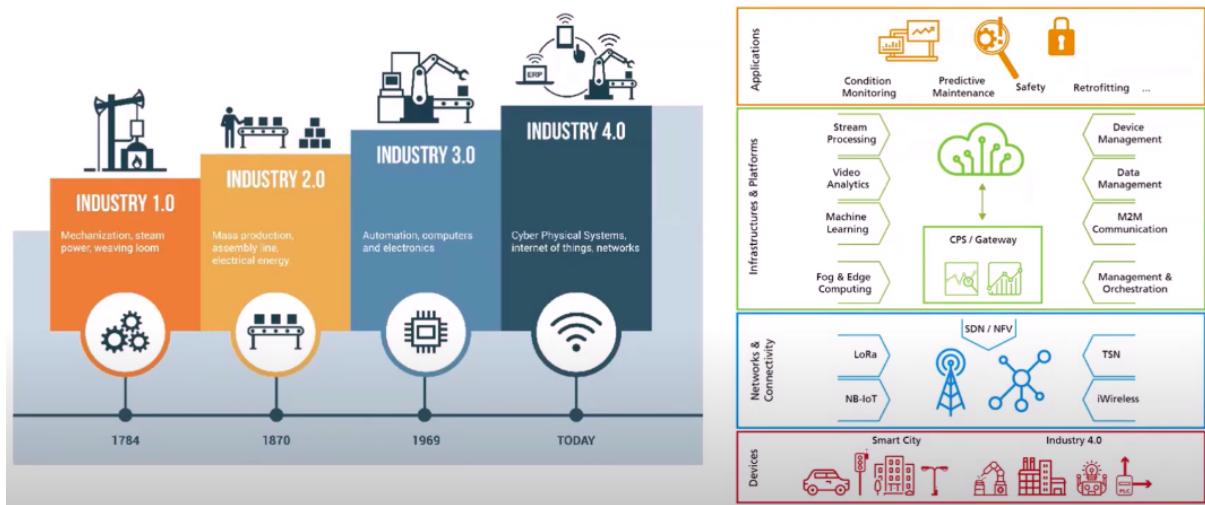


Figura 1.4d. Una possibile schematizzazione del mondo dell'Industrial Internet of Things.

Automotive: si parla di “IoV”, ossia “*Internet of Vehicles*” per indicare il mondo delle Smart Car (Figura 1.4..), che a bordo possono contare dai 30 agli oltre 100 computer collegati a sensori per garantire sicurezza, assistenza alla guida, affidabilità, comfort etc., compiendo attuazioni per il monitoraggio del traffico, oppure per il calcolo del percorso ottimale, etc.

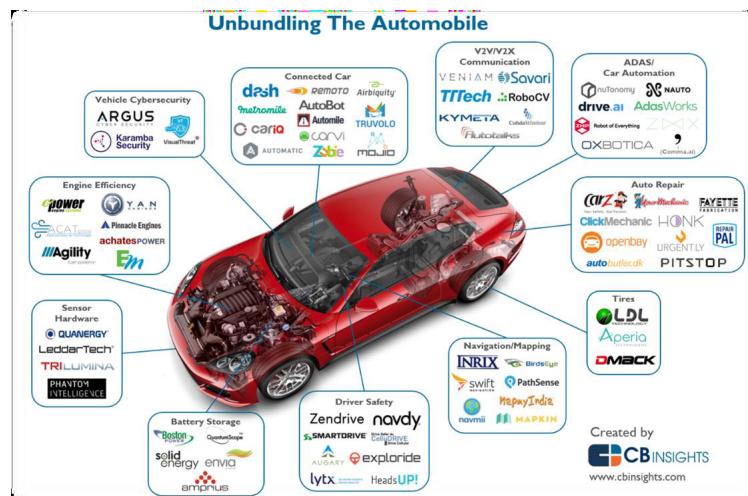


Figura 1.4d. Il set di funzionalità offerte da una Smart car.

Mars rover - Perseverance: appartiene all'ambito dell'*IoT interstellare*; probabilmente si rivelerà una delle applicazioni più importanti per questa tecnologia andando avanti nel tempo. Ci sono molti luoghi nel sistema solare ancora irraggiungibili. I robot che sono inviati sull'universo e i sensori che trasportano, aiuteranno a spianare la strada agli astronauti e agli esploratori per le generazioni a venire.

Perseverance, già prima di atterrare sul Pianeta Rosso per iniziare a lavorare ha fatto uso di sensori, infatti, il guscio che proteggeva il rover durante la sua discesa era anche pieno zeppo di dispositivi sensoristici chiamati *Mars Entry, Descent, and Landing Instrumentation 2 (MEDLI2)*, e offriva agli scienziati le prime misurazioni in tempo reale su pressioni e temperature durante l'ingresso nell'atmosfera.

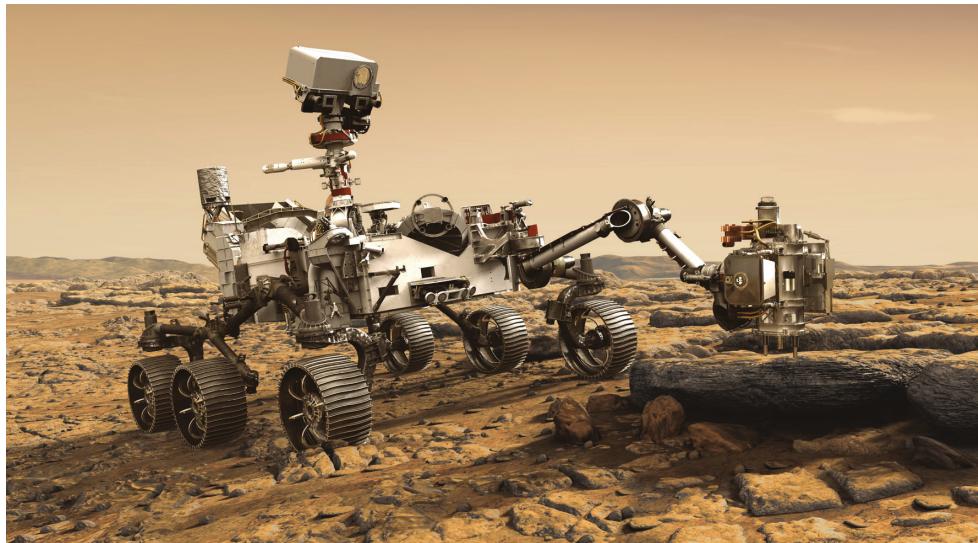


Figura 1.4e. Perseverance.

Un rover dall'aspetto simile, ma di dimensioni più contenute è l'oggetto intelligente che è stato sperimentato nel lavoro di tesi, e sarà presentato nel capitolo successivo.

[UD],[UFE],[MPI][IOT02], [PSV01].

Capitolo 2. Strumenti e Tecnologie utilizzate

In questo secondo capitolo il lettore sarà accompagnato da una carrellata di tutti gli “strumenti” sia hardware che software che hanno cooperato alla realizzazione del lavoro di tesi.

2.1 Protocollo HTTP

Facendo riferimento al modello OSI dell’ISO, l’HTTP (**Hypertext Transfer Protocol**), definito nelle **RFC 1945** e **2616**, è un protocollo a livello di applicazione, basato sul modello Request/Response (Figura 2.), in cui client e server sono in esecuzione su sistemi periferici diversi. Queste due entità comunicano mediante lo **scambio di messaggi HTTP**.

Tipicamente, nel contesto del web, il **lato client** di HTTP è implementato da un **browser web**, mentre il **lato server** da un **web server** sempre attivo e con un indirizzo IP fisso, che risponde potenzialmente alle richieste provenienti da milioni di diversi browser, e che può ospitare diversi **oggetti** (pagine HTML, immagini, video, audio, script javascript etc.).

Invece, in ambito di Internet of Things il ruolo di client è rivestito da un sensore (o da un gateway) che non è interessato solo a richiedere un servizio o un’informazione, ma può utilizzare la richiesta HTTP per inviare dei dati al server; per tale ragione, l’interazione tipica nel contesto dell’IoT è di tipo push .

Il server può rispondere per fornire informazioni, oppure semplicemente per informare il client che le ha ricevute correttamente.

Essendo il client che deve iniziare la comunicazione, se il server vuole un’informazione, quest’ultimo dovrà attendere che il client instauri una connessione. Dal momento che il gateway (o il dispositivo) può richiedere al server se vi sono o meno comandi che deve eseguire, l’interazione in questo caso è di tipo pull.

Le risorse ospitate nei server sono indirizzabili mediante un *URL*² (Uniform Resource Locator). Gli URL sono rappresentati secondo il seguente schema:

<protocollo>://<host>[:<porta>] [<percorso>] [?<query>]

² Più precisamente, gli URL Insieme agli URN (Uniform Resource Name) costituiscono gli URI (Uniform Resource Identifier), ossia una sintassi usata sul Web per identificare le risorse sulla rete Internet.

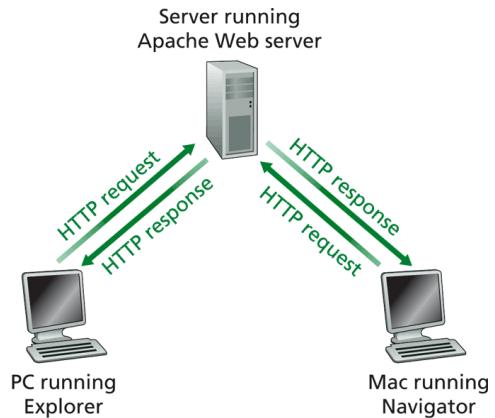


Figura 2.1a. Comportamento richiesta-risposta di HTTP

ad esempio: <http://www.unife.it:80/informatica/logo.jpeg>. Tuttavia sono soggetti ad essere “fragili”, poiché, se per qualche ragione fossero modificati perderebbero il loro “valore”. Se si aggiornasse l’URL di esempio senza spostare la risorsa “logo.jpg” dalla locazione in cui risiede attualmente, all’interno del directory “immagini”, ad esempio: <http://www.unife.it:80/informatica/immagini/logo.jpeg>, si rischierebbe di perdere tale risorsa ad esso associata, “rompendo” l’URL.

Con riferimento al campo del Web, HTTP definisce in che modo i client web richiedono le pagine ai web server e come questi ultimi le trasferiscono ai client utilizzando **TCP come protocollo di trasporto**, il quale offre un servizio di *trasferimento dati affidabile* (assicurando che ogni messaggio di richiesta HTTP emesso da un client arrivi intatto al server e viceversa). La gestione di eventuali dati smarriti, il recupero delle perdite o i riordini dei dati all’interno della rete sono dunque operazioni che HTTP delega a TCP e ai protocolli di livello inferiore dello stack ISO/OSI.

È importante sottolineare che il server invia i file richiesti ai client senza memorizzare alcuna informazione di stato a proposito del client. Per questa ragione, *HTTP è classificato come protocollo senza memoria di stato (stateless protocol)*.

Pertanto, in caso di ulteriori richieste dello stesso oggetto da parte dello stesso client, anche nel giro di pochi secondi, il server procederà nuovamente all’invio, non avendo mantenuto alcuna traccia di quello precedentemente effettuato.

Ogni coppia richiesta-risposta può essere inviata su una connessione TCP separata (HTTP con connessioni non persistenti) oppure sulla stessa connessione TCP (HTTP con connessioni persistenti).

Nel **caso di connessioni non persistenti**, il passaggio di una pagina web dal server al client avviene nel seguente modo:

1. Il Client HTTP inizializza una connessione TCP sulla porta 80 (porta di default per HTTP). Associate alla connessione TCP ci saranno una *socket* per il client ed una per il server. Dato che il client sta facendo un'esplicita richiesta al server per ottenere un'informazione, si dice che che HTTP è una tecnologia *pull*.
2. Il Client, tramite la propria socket, invia al server un *messaggio di richiesta HTTP*.
3. Il Processo Server riceve il messaggio di richiesta attraverso la propria socket associata alla connessione, recupera l'oggetto dalla memoria (centrale o di massa), lo incapsula in un *messaggio di risposta HTTP* che viene inviato al Client mediante la socket.
4. Quando il Server comunica a TCP di chiudere la connessione, quest'ultimo non terminerà la connessione fino a che non si sarà accertato che il Client abbia ricevuto integro il messaggio di risposta.
5. Quando il Client riceve il messaggio di risposta, la connessione TCP termina.

Una pagina web contiene al suo interno diversi oggetti web e la versione 1.1 di HTTP è in grado di caricare una sola risorsa alla volta, pertanto, quando il client avrà estratto dal messaggio di risposta il file HTML e lo avrà esaminato, quasi certamente troverà dei riferimenti ad altri oggetti (script javascript, immagini, audio, link, fogli di stile CSS, etc.). Ciò implica che per ogni oggetto che incontrano, i client debbano effettuare una richiesta,

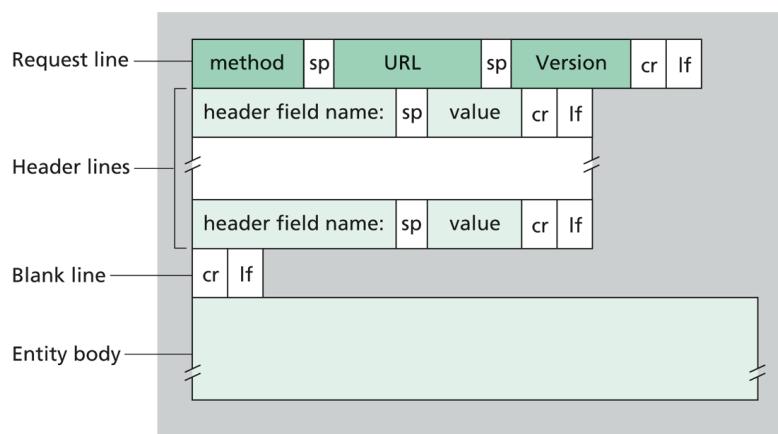


Figura 2.1c. Rappresentazione generale dei messaggi di richiesta HTTP

dovendo ripetere i primi quattro passi. Ad esempio, se tale richiesta è effettuata a: <https://www.unife.it/it> e al suo interno sono presenti n oggetti, in totale dovranno essere gestite $n+1$ connessioni TCP.

Il protocollo HTTP/2 giova invece del *multiplexing*, una funzionalità che permette di incapsulare le diverse richieste in una unica, portando ad una diminuzione della latenza.

(Immagini)

Un tipico **messaggio di richiesta HTTP**, schematizzato in Figura 2.1c, è il seguente:

```
GET /direttorio/pagina.html HTTP/1.1
Host: www.unife.edu
Connection: close
User-agent: Mozilla/5.0
Accept-language: fr
```

In generale, i messaggi di richiesta possono essere costituiti da un numero indefinito di righe, anche una sola.

Tale messaggio è scritto in testo ASCII, in modo che l'utente sia in grado di leggerlo. Consiste di cinque righe, ciascuna seguita da un carattere di ritorno a capo (carriage return) e un carattere di nuova linea (line feed).

La prima riga è detta riga di richiesta (**request line**) e quelle successive righe di intestazione (**header lines**).

L'ultima riga è seguita da una coppia di caratteri di ritorno a capo e nuova linea aggiuntivi.

La *riga di richiesta* presenta tre campi: metodo, URL e versione di HTTP.

Il campo **metodo** può assumere diversi valori, tra cui GET, POST, HEAD, PUT e DELETE.

La maggioranza dei messaggi di richiesta HTTP usa il **metodo GET**, adottato quando il browser richiede un oggetto identificato dal campo **URL**.

La **versione** è auto esplicativa: nell'esempio, il browser, che implementa la versione HTTP/1.1, sta richiedendo l'oggetto: “/direttorio/pagina.html”.

Si considerino ora le righe di intestazione dell'esempio.

La riga *Host: www.unife.edu* specifica l'host su cui risiede l'oggetto.

Si potrebbe pensare che questa riga di intestazione non sia necessaria, dato che è già in corso una connessione TCP con l'host. Ma, l'informazione fornita dalla linea di intestazione dell'host viene richiesta dalle cache dei proxy.

Con la linea di intestazione *Connection: close*, il browser sta comunicando al server che non

si deve occupare di connessioni persistenti, ma vuole che questi chiuda la connessione dopo aver inviato l’oggetto richiesto.

User-agent: specifica il tipo di browser che sta effettuando la richiesta al server, in questo caso Mozilla/5.0, un browser Firefox. Questa riga è utile poiché il server può inviare versioni differenti dello stesso oggetto a browser di tipi diversi.

Accept language:fr indica che il client preferisce ricevere una versione in francese dell’oggetto se disponibile; altrimenti, il server dovrebbe inviare la versione di default.

Accept-language: è la riga che rappresenta solo una delle molte intestazioni di negoziazione dei contenuti disponibili in HTTP.

Osservando il formato di un messaggio di richiesta HTTP, si nota che dopo le linee di intestazione si trova un “*entity body*” (**corpo**), vuoto nel caso del metodo GET, ma utilizzato dal metodo POST.

In genere, un client HTTP usa il *metodo POST* quando l’utente riempie un form: per esempio, quando un utente fornisce le voci da trovare a un motore di ricerca. Nel caso di messaggio POST, l’utente sta ancora richiedendo una pagina web al server, ma i contenuti specifici della pagina dipendono da ciò che l’utente ha immesso nei campi del form.

Quindi, se il valore del campo metodo è POST, allora il corpo contiene ciò che l’utente ha immesso nei campi del form.

Tuttavia, le richieste generate con il form non usano necessariamente il metodo POST. Anzi, i form HTML utilizzano spesso il metodo GET e includono i dati immessi (nei campi del form) nell’URL richiesto. Per esempio, se un form impiega il metodo GET e presenta due campi, e se i dati immessi sono “dipartimento” e “informatica”, allora l’URL avrà una struttura simile a: www.unife.it/search?dipartimento&informatica.

Il *metodo HEAD* è simile a GET: quando un server riceve una richiesta con il metodo HEAD, risponde con un messaggio HTTP, ma *tralascia gli oggetti richiesti*.

E’ utilizzato in genere dagli sviluppatori di applicazioni web per verificare la correttezza del codice prodotto.

Il *metodo PUT*, è frequentemente usato assieme agli strumenti di pubblicazione sul Web, e consente agli utenti di inviare un oggetto a un determinato percorso su uno specifico web server; viene anche utilizzato dalle applicazioni che richiedono di inviare oggetti ai web server. Il *metodo DELETE* permette invece la cancellazione di un oggetto su un server³.

³ Per motivi di sicurezza i metodi PUT e DELETE sono spesso disabilitati nei web server e si preferisce surrogarli con il metodo POST in cui si specifica nei vari campi che cosa aggiungere o cancellare dal server; in tal modo l’applicazione è in grado di fare dei controlli aggiuntivi.

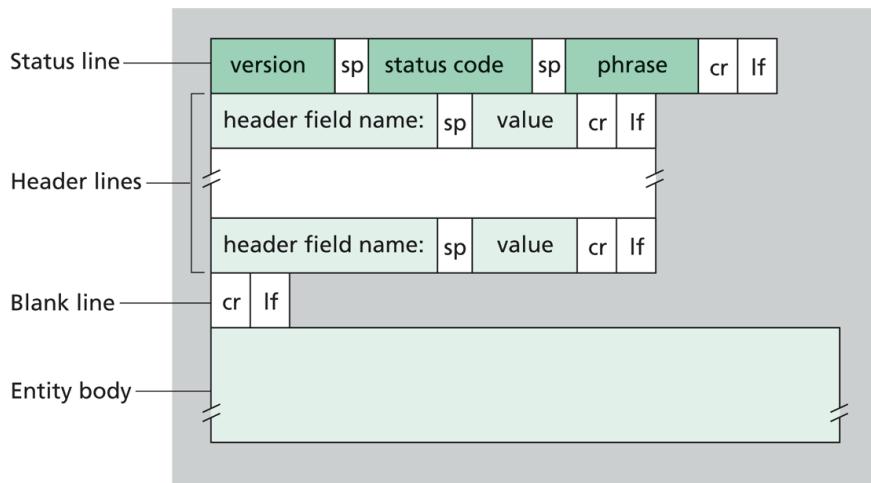


Figura 2.1d. Formato generale dei messaggi di risposta di HTTP.

Un tipico **messaggio di risposta HTTP**, che potrebbe rappresentare la risposta al messaggio di richiesta dell'esempio precedente è il seguente:

```

HTTP/1.1 200 OK
Connection: close
Date: Thu, 18 Aug 2022 15:44:04 GMT
Server: Apache/2.2.3 (CentOS)
Last-Modified: Thu, 15 Sep 2022 17:25:13 GMT
Content-Length: 6821
Content-Type: text/html
<Dati>

```

Le sezioni che si notano analizzandolo in dettaglio sono: la *riga di stato iniziale*, sei *righe di intestazione* e un *corpo*.

Quest'ultimo è il fulcro del messaggio dal momento che contiene l'oggetto richiesto (rappresentato da: <Dati>)

La **riga di stato** possiede tre campi: la *versione del protocollo*, un *codice di stato* e un corrispettivo *messaggio di stato*.

Nell'esempio proposto, la riga di stato indica che il server ha trovato e sta inviando l'oggetto richiesto utilizzando HTTP/1.1. Considerando le **righe di intestazione**:

Connection: close per comunicare al client che ha intenzione di chiudere la connessione TCP dopo l'invio del messaggio.

La *riga Date*: indica l'ora e la data di creazione e invio, da parte del server, della risposta HTTP. Si noti che non si tratta dell'istante in cui l'oggetto è stato creato o modificato per l'ultima volta, ma del momento in cui il server recupera l'oggetto dal proprio file system, lo inserisce nel messaggio di risposta e invia il messaggio.

La *riga Server*: indica che il messaggio è stato generato da un web server Apache; essa è analoga alla riga User-agent nel messaggio di richiesta HTTP.

La *riga Last-Modified*: indica l'istante e la data il cui l'oggetto è stato creato o modificato per l'ultima volta. Tale riga è importante per la gestione dell'oggetto nelle cache, sia nel client locale sia in alcuni server in rete (proxy server o proxy).

La riga di intestazione *Content-Length*: contiene il numero di byte dell'oggetto inviato.

La *riga Content-Type*: indica che l'oggetto nel corpo è testo HTML. Il tipo dell'oggetto viene ufficialmente identificato tramite l'intestazione Content-Type: e non tramite l'estensione del file.

E' doveroso spendere qualche parola aggiuntiva sui **codici di stato** e sulle loro *espressioni*. Il codice di stato e l'espressione associata indicano il *risultato della richiesta*.

Tra i più comuni codici di stato e relative espressioni si trova:

- *200 OK*: la richiesta ha avuto successo e in risposta si invia l'informazione;
- *301 Moved Permanently*: l'oggetto richiesto è stato trasferito in modo permanente; il nuovo URL è specificato nell'intestazione *Location*: del messaggio di risposta. Il client recupererà automaticamente il nuovo URL;
- *400 Bad Request*: si tratta di un codice di errore generico che indica che la richiesta non è stata compresa dal server;
- *404 Not Found*: la risorsa richiesta non esiste sul server;
- *505 HTTP Version Not Supported*: il server non dispone della versione di protocollo HTTP richiesta.

Le specifiche HTTP definiscono moltissime altre righe di intestazione utilizzabili nei messaggi di richiesta e di risposta HTTP [UFE], [KRS],[PIV].

2.2 Servizi RESTful

Fino a qualche anno fa esistevano sistemi informativi e middleware indipendenti l'uno dall'altro ed era nata l'esigenza di farli comunicare tra di loro; pertanto, lo sviluppo iniziale dei servizi web e dell'ingegneria del software orientato ai servizi si basava sull'uso di *standard*, che permettessero di evitare la proliferazione di sistemi proprietari non interoperabili [UFE],[IAN].

Tali standard avevano come obiettivo la definizione delle interfacce dei servizi (delle funzioni che era possibile invocare); definivano inoltre come poteva avvenire la comunicazione tra i componenti del sistema e come tale comunicazione potesse essere utilizzata per l'invocazione dei servizi. L'ottica generale era quella della SOA (Service Oriented Architecture).

Da queste necessità era nato lo standard dei *Web Services RPC-based*: una specifica precisa per la comunicazione tra le parti di applicazioni distribuite.

I protocolli costituenti i Web Services RPC based sono: XML (extensible Markup Language), che definisce in che modo le informazioni vengono formattate (XML); HTTP, per il trasporto delle informazioni; SOAP (Simple Object Access Protocol): definisce come i messaggi vengono incapsulati; WSDL (Web Services Description Language), che regola come le interfacce vengono definite.

Il possesso di standard è di aiuto, poiché essi rappresentano un linguaggio comune, concordato e condiviso per permettere l'interazione tra sistemi software legacy e proprietari, prodotti e mantenuti da organizzazioni distinte.

Tuttavia, definire uno standard tende a normare in modo forse esagerato ogni singolo dettaglio e a portare il quantitativo di informazioni all'interno del protocollo ad un livello tale per cui l'overhead di comunicazione e di produzione dei servizi a volte è eccessivo.

Infatti, la definizione degli standard associati ai WS RPC-Based ha portato ad una comunicazione inefficiente dal momento che può essere complesso ed oneroso sviluppare dei servizi usando lo standard WSDL e può essere altresì costoso in termine di overhead di comunicazione formattare i dati in XML utilizzando SOAP come protocollo di incapsulamento [UFE].

Per superare questi problemi è stato sviluppato un approccio più “leggero” all'architettura dei servizi web, basato su **REST** (REpresentational State Transfer).

E' doveroso sottolineare che REST non è uno standard, né un protocollo, bensì è uno **stile architetturale**, introdotto per la prima volta nel 2002 da Roy Fielding nella sua tesi di Dottorato, che si basa sul trasferimento delle *rappresentazioni di risorse* da un Server a un Client [PIV],[IAN].

Per *stile architetturale* si intende un insieme di linee guida per la realizzazione di una "architettura di sistema" in grado di produrre dei servizi web per mettere in comunicazione e consentire lo scambio di dati tra Client e Server.

Quando si realizza un sistema Server in grado di comunicare con i Client attraverso l'utilizzo di REST, si parla di un *sistema RESTful* [PIV].

In un'*architettura RESTful* tutto è rappresentato come una **risorsa**.

Quest'ultima, in termini molto generici può rappresentare un qualsiasi servizio o una qualunque entità remota, ad esempio: una tabella di un database; oppure, nel contesto dell'IoT, un macchinario o un singolo dispositivo come un sensore; oppure un documento, come un capitolo di questa tesi.

Ad esempio, tale capitolo possiederà due rappresentazioni: una in Microsoft Word, utilizzata per l'editing ed una rappresentazione PDF, utilizzata per visualizzare il capitolo nel Web.

Tuttavia, la risorsa logica sottostante, fatta di testi e immagini, è la stessa in tutte e due le rappresentazioni [IAN].

In generale dunque, le *rappresentazioni delle risorse* possono essere in un qualsiasi formato che abbia una certa struttura, ad esempio JSON (Javascript Object Notation) o XML [UFE].

Solitamente, per il trasferimento delle informazioni REST *si basa su HTTP*, ereditandone i limiti. REST risulta tuttavia indipendente dal protocollo sottostante utilizzato; è dunque possibile realizzare un servizio REST senza l'uso del protocollo HTTP.

Gli URI identificano univocamente la risorsa remota attraverso l'uso di *endpoint strutturati*, composti dal nome di un dominio seguito da una *struttura gerarchica* (diversa da quella che solitamente rappresenta un file system o i file all'interno di directory), che permette di rappresentare l'organizzazione delle informazioni in modo astratto.

Le risorse sono simili agli oggetti e sono associate a quattro operazioni polimorfiche (o verbi) fondamentali (CRUD), come illustrato in Figura 2.2a.

1. *Create*: generare la risorsa;

2. *Read*: restituire una rappresentazione della risorsa;
3. *Update*: modificare il valore di una risorsa;
4. *Delete*: rendere inaccessibile una risorsa.

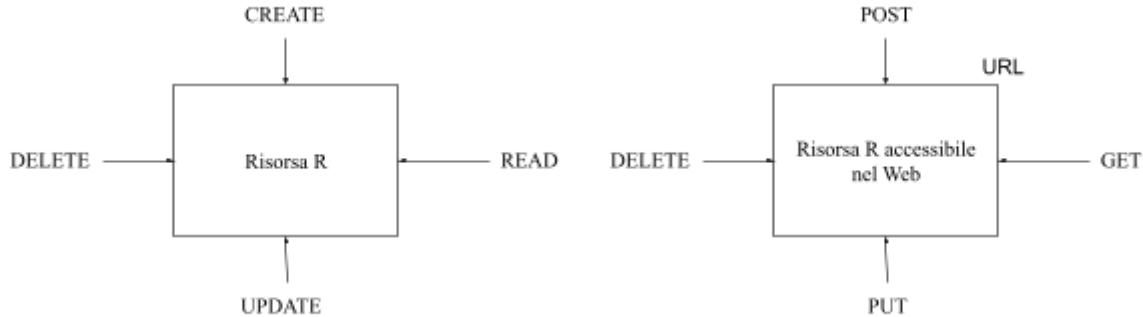


Figura 2.2a. Azioni di una risorsa

Figura 2.2b. Risorse del Web

Come forse già intuito dal precedente paragrafo, il web è un esempio di sistema che ha un’architettura RESTful, in cui le pagine sono risorse, e l’identificatore unico di una pagina web è il suo URL.

REST utilizza i metodi dell’HTTP al fine di renderlo autoesplicativo. Infatti, le quattro generiche azioni su una risorsa sono in relazione uno ad uno con i seguenti metodi di HTTP:

1. *POST*: per creare una risorsa, associando i dati che la definiscono;
2. *GET*: per leggere il valore di una risorsa e restituirlo al richiedente nella rappresentazione specificata (come XHTML, che può essere visualizzata in un browser);
3. *PUT*: per aggiornare il valore di una risorsa;
4. *DELETE*: per cancellare una risorsa [IAN].

(Figura 2.2b).

La gestione dei metodi HTTP è *implementation-specific*: significa che non è sempre definito al 100% come alcuni di questi debbano essere gestiti. La differenza tra PUT e POST, dal punto di vista architettonico è ben precisa. In realtà, in gran parte dei servizi REST questa distinzione non è così marcata, giacché a volte è possibile trovare servizi REST che utilizzano il verbo POST non solo per aggiornare un’informazione, ma anche per crearne una nuova, e viceversa.

Nel caso di PUT e POST, è necessario inserire nella richiesta un *payload*, contenente le informazioni che si vogliono aggiornare e quelle che si vogliono inserire [UFE].

Tutti i servizi, in qualche misura, operano sui dati. Considerando ad esempio un servizio che restituisce le temperature massime e minime di una località, i cui dati meteorologici sono contenuti in un database, sfruttando l'approccio RESTful è possibile accedere ai dati meteorologici di ciascuna località memorizzati nel database utilizzando gli URL in questo modo:

<http://meteo-info-esempio.net/temperature/ferrara>

<http://meteo-info-esempio.net/temperature/rovigo>

chiamando quindi l'operazione di GET, che restituisce una lista di temperature massime e minime. Per richiedere le temperature in una specifica data, si può utilizzare una query (*query string*) con l'identificatore URL:

<http://meteo-info-esempio.net/temperature/rovigo?date=20220825>

Questo approccio può essere usato per eliminare eventuali ambiguità nella richiesta, dato che ci possono essere più località nel mondo con lo stesso nome, ad esempio:

[http://meteo-info-esempio.net/temperature/rovigo?date=20220825®ione=Veneto&stato=""IT"](http://meteo-info-esempio.net/temperature/rovigo?date=20220825®ione=Veneto&stato=).

Una differenza importante tra i servizi RESTful e i servizi SOAP è che i primi non sono servizi che si basano esclusivamente su XML. Pertanto, quando una risorsa viene richiesta, creata o modificata, è possibile specificare la sua rappresentazione.

Questo aspetto è importante per i servizi RESTful, in quanto è possibile utilizzare rappresentazioni come JSON, che possono essere elaborate più efficientemente delle notazioni basate su XML, riducendo gli overhead in una chiamata di servizio.

Quindi, la precedente richiesta di temperature massime e minime per “Rovigo” potrebbe restituire le seguenti informazioni:

```
{  
  "città": "Rovigo",  
  "regione": "Veneto",  
  "stato": "IT",  
  "data": "25/08/2022",  
  "unità": "Celsius",  
  "temp max": "39",  
  "temp min": "23"  
}
```

La risposta a una richiesta GET in un servizio RESTful può includere gli identificatori URL. Pertanto, se la risposta a una richiesta è un insieme di risorse, allora potrebbe essere incluso l'URL di ciascuno di questi servizi. Il servizio richiedente potrebbe elaborare le richieste in un suo particolare modo. Quindi, una richiesta di informazioni meteorologiche con il nome di una località che non è unico potrebbe restituire gli URL di tutte le località che soddisfano la richiesta.

Per esempio:

<http://meteo-info-esempio.net/temperature/ferrara-nord>

<http://meteo-info-esempio.net/temperature/ferrara-centro>

<http://meteo-info-esempio.net/temperature/ferrara-sud>

Un principio fondamentale della progettazione dei servizi RESTful, che deriva dalla scelta del protocollo HTTP, è che dovrebbero essere stateless: in una sessione interattiva, la risorsa stessa non dovrebbe includere alcuna informazione sullo stato, come l'ora dell'ultima richiesta; piuttosto, tutte le informazioni necessarie sullo stato dovrebbero essere restituite al richiedente. Qualora fossero necessarie in successive richieste, tali informazioni di stato dovrebbero essere restituite al server dal richiedente; ciò porta a rendere più semplice e ad aumentare la scalabilità del sistema e ad un minor overhead soprattutto per chi offre servizi di tipo REST. Di contro, lato Client aumentano un po' la complessità e l'overhead (ad esempio, dal punto di vista di autenticazione e autorizzazione) dal momento che è necessario ogni volta ripartire dall'inizio con la comunicazione [IAN], [UFE].

REST offre il *caching delle informazioni*, offrendo cioè la possibilità di bufferizzare le informazioni non tanto sul client che ha effettuato una richiesta precedente, ma anche su proxy, che possono fungere da nodi intermediari tra chi effettua la richiesta e chi fornisce la risposta in modo tale da poter intercettare richieste successive, anche di client diversi con la stessa richiesta; questo porta a scaricare il carico di lavoro sul server, aiutando certamente ad aumentare la scalabilità del sistema [UFE].

I servizi RESTful si sono diffusi molto di più negli ultimi anni a causa della vasta diffusione dei dispositivi mobili, dotati di capacità di elaborazione limitate, quindi i minori overhead dei servizi RESTful consentono migliori prestazioni dei sistemi.

Essi sono anche più facili da utilizzare con i siti web esistenti - *implementare un'API⁴ RESTful* per un sito web, di solito, è un'operazione abbastanza semplice [IAN] .

Tuttavia, l'approccio RESTful presenta alcuni *problem*i:

1. Quando un servizio ha un'interfaccia complessa e non è una risorsa semplice, può essere difficile progettare un insieme di servizi RESTful per rappresentare la sua interfaccia.
2. Non ci sono standard per la descrizione delle interfacce RESTful, quindi gli utenti dei servizi devono affidarsi alla documentazione informale per capire l'interfaccia.
3. In alcuni casi, un fornitore di servizi web potrebbe voler elaborare operazioni preservando lo stato (*stateful*), ma REST non permette di farlo. Questo accade, ad esempio, nel caso di bonifici bancari. Una soluzione a questo problema di esempio può prevedere l'utilizzo di SOAP [SOP].

Spesso è possibile fornire interfacce SOAP e RESTful allo stesso servizio o alla stessa risorsa. Tale approccio duale adesso è comune ai servizi cloud forniti da provider come Microsoft, Google e Amazon. I client dei servizi possono quindi scegliere il metodo di accesso ai servizi che meglio si adatta alle loro applicazioni [IAN].

2.3 Raspberry Pi 3

Il Raspberry Pi, nei suoi diversi modelli, è un single-board computer di dimensioni paragonabili ad una carta di credito, che monta un processore ARM e che possiede tutte le funzioni di base di un personal computer. È stato progettato nel 2011 nel Regno Unito dalla “Raspberry Pi Foundation” con l'intento di promuovere lo studio dell'informatica nelle scuole. Oggi è utilizzato principalmente in ambiti di smart home, intelligenza artificiale e

⁴ Un' API (Application Programming Interface) è un insieme di strumenti, definizioni e protocolli per l'integrazione di software e servizi applicativi . È il materiale che consente di interagire con un computer o un sistema per recuperare informazioni o eseguire una funzione. L'API facilita la comunicazione con il sistema che può così comprendere e soddisfare la richiesta. Un'API RESTful, in quanto tale deve essere conforme ai vincoli dello stile architetturale REST [IAN].

Internet of Things. Insieme ad altri dispositivi come Arduino, Tessel, Parallelia etc., il Raspberry Pi appartiene al mondo dell'*Open Hardware*, ossia a quell'hardware costruito a partire da componenti e materiali facilmente reperibili, con processi standard e architetture aperte, basandosi su contenuti senza restrizioni e strumenti di progettazione open-source per massimizzare la capacità degli individui di fare e utilizzare l'hardware [SLD].

Il modello di scheda utilizzato nel progetto di tesi è: "Raspberry Pi 3 Model B Rev 1.2"



Figura 2.3b. Un sistema Raspberry Pi 3 B.

Le principali componenti (Figura 2.3b) sono:

- Processore Broadcom BCM2837 64bit Quad Core 1.2GHz 1GB RAM
- BCM43438 wireless LAN e Bluetooth Low Energy (BLE) on board
- 100 Base Ethernet
- GPIO esteso a 40-pin
- 2 blocchi USB da 2 porte ciascuno
- Uscita stereo a 4 poli e porta video composita
- HDMI
- Porta CSI camera per collegare una Raspberry Pi camera
- DSI porta connettere un display touchscreen per il Raspberry Pi
- Porta Micro SD port per il caricamento del sistema operativo e per la memorizzazione dei dati.

- Upgraded switched Micro USB power source up to 2.5A [RSP]

Particolarmente rilevante risulta essere il *General-Purpose Input Output* (GPIO, Figura 2.3c), uno slot di 40 pin disposti su due file all'interno della scheda, che può essere utilizzato per collegare vari dispositivi elettronici periferici e sensori per poterli controllare o monitorare tramite segnali di livello di ingresso/uscita.

Si può utilizzare il GPIO ad esempio per controllare la velocità di un motore CC o per leggere la distanza misurata da un sensore a ultrasuoni.

A differenza delle altre porte che si trovano sul Raspberry Pi, che sono state programmate per funzionare in un certo modo, l'interfaccia GPIO offre agli sviluppatori la libertà di operare manualmente.

I pin presenti sul sistema di comunicazione GPIO si possono dividere in gruppi sulla base della loro funzionalità:

- 8 Pin di massa o ground (gnd):
- 4 pin di alimentazione (i pin 1 e 17 forniscono 3,3V, mentre i pin 2 e 4 restituiscono una tensione pari a 5V).
- UART (Universal Asynchronous Receiver/Transmitter), rispettivamente il pin 8 (TXD) per la trasmissione e il pin 10 (RXD) per la ricezione; questi due pin sono utilizzati come interfaccia seriale, per collegare ad esempio un dispositivo Arduino o eventuali schede simili.
- I2C (Inter Integrated Circuit): tutti quei pin che servono per la comunicazione sincrona con architettura master-slave tra più circuiti integrati, come sensori di temperatura e umidità. Rientrano in questo insieme i pin SDA e SCL. In particolare, I2C è un bus seriale sincrono e bidirezionale, sviluppato da Philips che Richiede per l'appunto solo due fili per trasferire le informazioni tra più dispositivi e sensori collegati al bus.

La loro comunicazione avviene tramite SDA (data pin) e SCL (clock speed pin). Ogni dispositivo slave ha un indirizzo univoco, consentendo una comunicazione rapida con molti dispositivi. Il pin ID_EEPROM è anche un protocollo I2C ed è utilizzato per comunicare con i cappelli motore.

- SPI (Serial Peripheral Interface): Viene utilizzata per controllare la velocità dei dati in entrata e in uscita tra il Raspberry e il dispositivo connesso. I pin costituenti questa interfaccia consentono ad esempio di collegare schermi LCD, touch screen o convertitori analogico-digitali.

- W1-GPIO (One-Wire Interface): interfaccia che permette di connettere più dispositivi in modo bidirezionale attraverso un solo filo.
- PWM (Pulse-Width Modulation): i pin 12, 32, 33 e 35 necessari per regolare ad esempio la luminosità di un led, oppure la velocità di un motore.
- 17 pin tra quelli appena elencati ed altri completamente generici che per funzionare utilizzano la differenza di potenziale. Possono operare sia in entrata che in uscita [PIN], [ADP].

Si può ottenere una maggior comprensione dei pin che compongono il GPIO, lanciando dal terminale il comando `pinout`, all'interno del sistema Raspberry Pi.

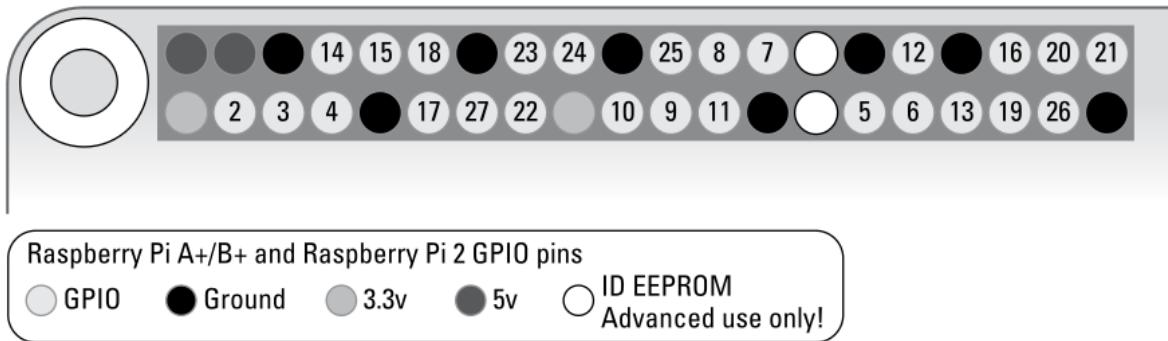


Figura 2.3c. I Pin del modulo GPIO di un Raspberry Pi con le relative funzioni

2.4 Mars Rover PiCar-B

Il **Mars Rover PiCar-B** (Figura 2.4a) è un' auto robotica intelligente ed open source, dall'aspetto del tutto simile al rover spaziale incontrato alla fine del capitolo 1, che è stato pensato per essere sperimentato nell'ambito dell'IoT da parte di appassionati e studenti di robotica. È stato sviluppato ed è mantenuto da *Adeept*, un team di assistenza tecnica di software e hardware, dedicato all'applicazione di Internet e della più recente tecnologia industriale nell'area open source, con lo scopo di fornire il miglior supporto hardware e servizio software per produttori generali e appassionati di elettronica in tutto il mondo.

Il prodotto in questione è facile da montare e presenta una curva di apprendimento relativa al suo utilizzo non particolarmente complessa.

Offre molte funzionalità tra cui: evitamento automatico degli ostacoli, riconoscimento del colore, rilevamento di oggetti in movimento, indicatore di illuminazione e tracciamento di linea con l’opportuno modulo di tracciamento [ADP].

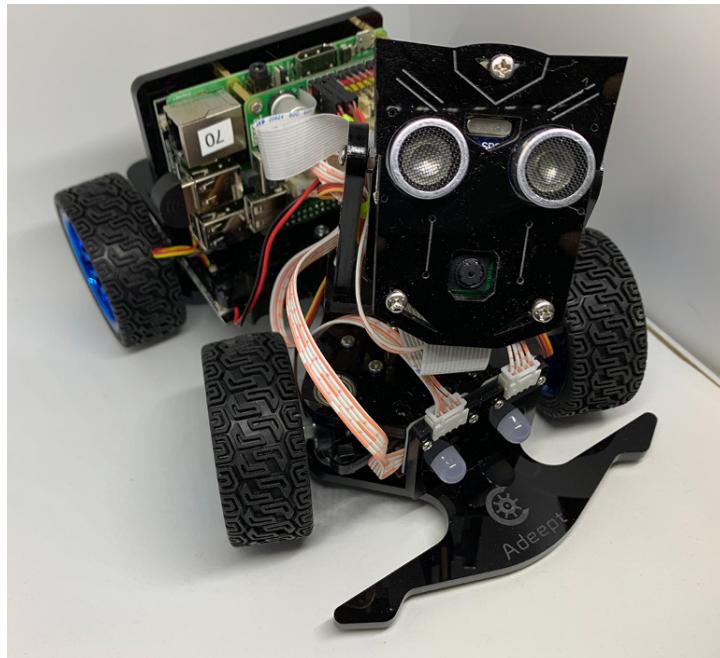


Figura 2.4a. Un sistema Mars Rover PiCar-B già assemblato.

In particolare, possiede la peculiarità di poter essere *telecomandato via web*, per mezzo di dispositivi di varia natura, quali smartphone e tablet, computer etc.

Il PiCar-B necessita di due batterie 18650 da 3,7 V LiPo 5000 mAh. In alternativa, è possibile alimentarlo spegnendo l'interruttore di alimentazione (o rimuovendo le batterie) e fornendo alimentazione per il Raspberry Pi dalla presa USB, che quindi alimenta sia il robot che il Raspberry Pi.

Il “cuore” del prodotto è rappresentato dalla scheda “**Adeept Motor HAT V2.0**” (Scheda di controllo motore, Figura 2.4-b), progettata per interfacciare il Raspberry Pi con i sensori e i motori del Mars Rover PiCar-B.

Il resto della scheda collega i pin del GPIO dal Raspberry Pi ai vari sensori e attuatori.

La scheda del controller del motore ha anche un alimentatore da 5.0 V che fornisce energia al Raspberry Pi e ai motori dalle batterie LiPo [ADP], [PAO].

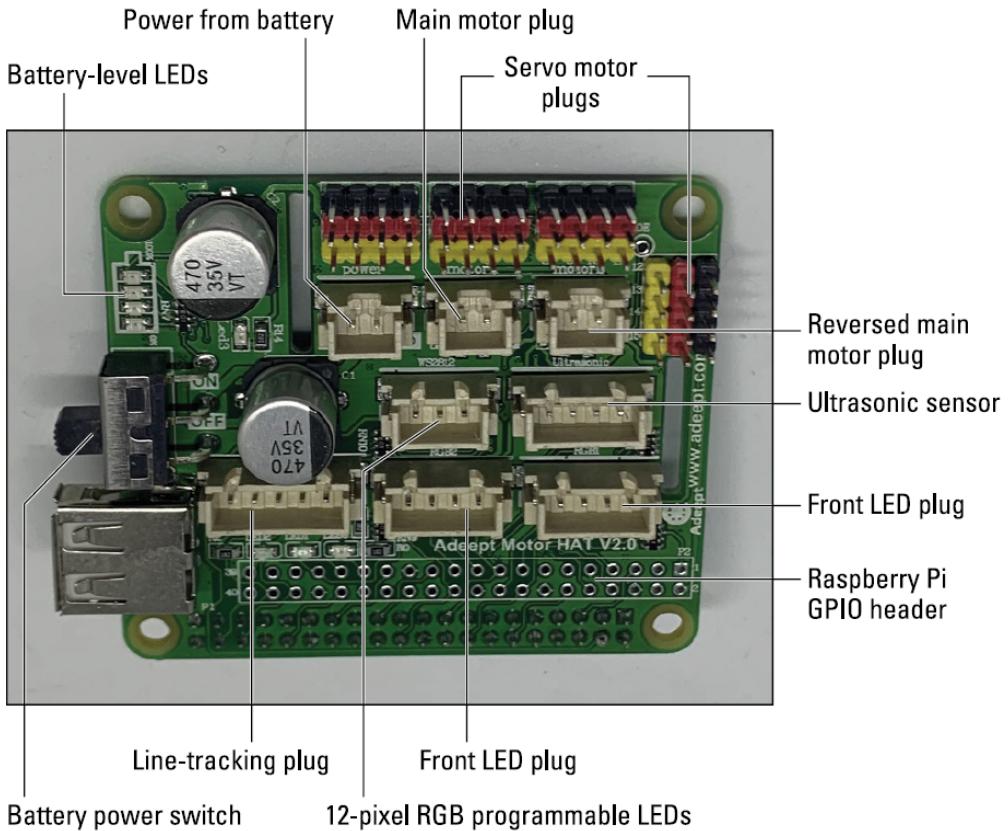


Figura 2.4b. Il circuito stampato “Adept Motor HAT V2.0” del Mars Rover PiCar-B.

Osservando la Figura 2.4b da sinistra verso destra si nota la presenza di diverse interfacce:

- “*Battery power switch*”, ossia l’interruttore per accendere o spegnere il Motor HAT (da considerare qualora fosse alimentato con le 2 batterie agli ioni di litio ad alta energia);
- “*Battery-level LEDs*” indica il livello di carica delle 2 batterie;
- “*Power from battery*” necessaria per l’alimentazione esterna;
- “*Line-tracking plug*” è l’interfaccia pin del modulo di tracciamento;
- “*12-pixel RGB programmable LEDs*”, rappresenta l’ingresso pin del modulo WS2812;
- “*Front LED plug*” sono invece le due interfacce che consentono la connessione di 2 led RGB posti frontalmente al robot;
- “*Ultrasonic sensor*” è l’ingresso per il collegamento di un sensore ultrasonico;

- “*Servo motor plugs*” è un’interfaccia che ospita i servi motori necessari; questi ultimi sono controllati da un chip fondamentale presente sulla scheda: il PCA9685, un dispositivo I2C che controlla fino a 16 servomotori alla volta (di cui 3 utilizzati in questo robot, dotandolo di molto spazio per l’espansione);
- “*Main motor plug*” e “*Reversed main motor plug*” sono gli ingressi necessari per l’inserimento di un motore DC. Sono necessari due slot per il motore per risolvere eventuali problemi legati al cablaggio del motore in una direzione piuttosto che in quella inversa. In altre parole, per evitare ad esempio che il comando “avanti” faccia indietreggiare il robot, è sufficiente spostare il motore sull’altro collegamento del Motor HAT [ADP], [PAO].

È ora indispensabile esaminare i componenti del Mars Rover PiCar-B, senza soffermarsi sulla struttura meccanica soggiacente al robot, piuttosto su ogni apparecchio “attivo” come sensori, motori e attuatori. Si disserterà del codice Python utilizzato per comunicare con tali componenti nel terzo capitolo dell’elaborato.

Uno di questi è il *servo*. In generale, un **servomotore** (Figura 2.4c) è la combinazione di tre elementi: un motore DC (a corrente continua), un semplice circuito di controllo e un set di ingranaggi.

Talvolta può includere anche un potenziometro (ossia un resistore con un valore di resistenza variabile), in grado di fornire un feedback posizionale. Il servomotore controlla la rotazione del motore DC attraverso il circuito di controllo, che ne regola l’angolo, dando la possibilità di mantenere stabile la posizione raggiunta.

Come preannunciato, sono stati installati 3 servi aventi un angolo di inclinazione variabile tra 0° e 180°; il primo per poter controllare la rotazione della testa del robot dall’alto verso il basso (e viceversa), il secondo da sinistra verso destra (e viceversa), mentre il terzo per poter controllare la rotazione delle ruote da sinistra verso destra (e viceversa).

È necessario connettere i microservi desiderati alle porte PWM disponibili nel Motor HAT, seguendo scrupolosamente le convenzioni sui colori dei fili (giallo, rosso e marrone).

Siffatti servi sono controllati dal chip PCA9685, situato sul bus di comunicazione I2C del Raspberry Pi [ADP], [PAO].

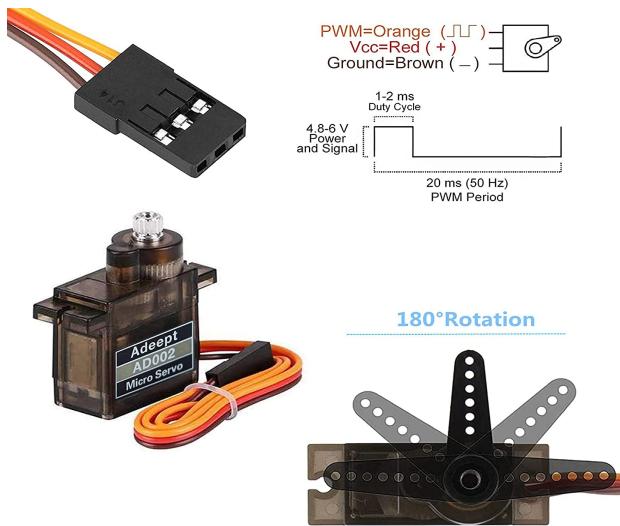


Figura 2.4c. Un microservo e la raffigurazione della cavetteria ad esso associata con accanto un frame che illustra la rotazione dello stesso fino a 180°, resa possibile dalla PWM.

Ogni servo è controllato mediante l'invio di segnali ad impulsi da parte del microcontrollore indicando al servomeccanismo del servo la direzione in cui muoversi.

Per attuare tale movimento, il servo a 180° può utilizzare la **PWM** (Pulse-Width modulation).

La *Modulazione della larghezza di impulso* (PWM) è una tecnica basata sulla variazione della quantità di tempo in cui un segnale è “alto” (generalmente a 5V) rispetto a quando è “basso” (0V).

Il rapporto tra il *periodo* di tempo in cui il segnale è alto (indicato con τ) e il tempo totale considerato (indicato con T) è chiamato *Duty Cycle* (indicato con D).

$$D = \frac{\tau}{T}$$

Oltre che per regolare l'angolo di sterzata di un microservo, la stessa metodologia può essere applicata per controllare l'attenuazione di un LED RGB. Infatti, se si altera il tempo in cui il LED è acceso rispetto a quando è spento, è possibile controllare la luminosità registrata dall'occhio umano. Osservando ad esempio la Figura 2.4-c, con un ciclo di lavoro del 100 percento, il LED è acceso il 100 percento delle volte e con un duty cycle dello 0 percento è sempre spento. Al variare del tempo di attivazione del segnale cambia la luminosità apparente del LED. Il LED si accende e si spegne a una *frequenza* predefinita di 100 volte al secondo (Hz), ma è possibile modificare la frequenza [PWM01]. [PWM02].

Il range di valori di duty cycle PWM dei servi utilizzati si attesta tra 100 e 560, corrispondente a un intervallo di rotazione compreso tra 0° e 180° [ADP], [PAO].

La seguente formula permette di calcolare l'ampiezza dell'angolo di inclinazione corrispondente all'attuale ciclo di lavoro:

$$\frac{(<\text{ciclo di lavoro}> - 100)}{2.55}$$

Se si volesse ad esempio ottenere una sterzata di 80°, sarebbe necessario applicare un ciclo di lavoro pari a 304.

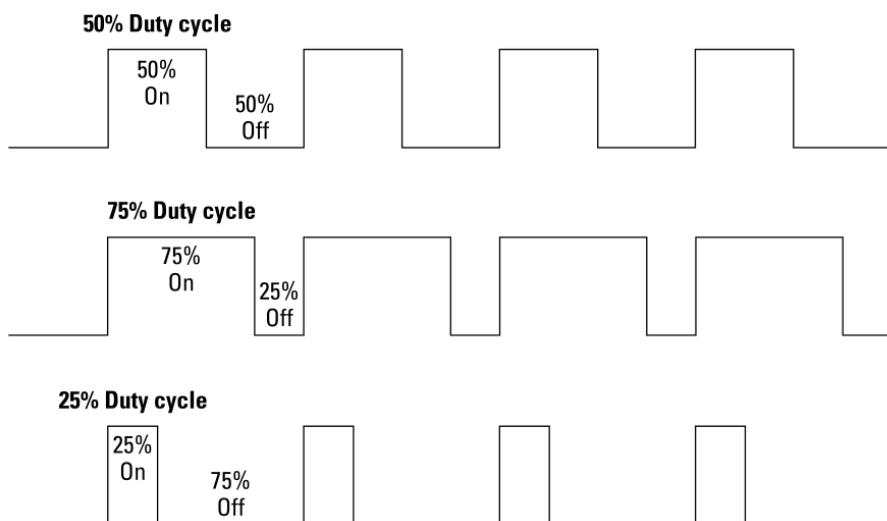


Figura 2.4d. Illustrazione di tre diversi cicli di lavoro in percentuale di un segnale PWM.

Un altro blocco collocato sul Mars Rover PiCar-B è il **Motore DC** (figura 2.4-e), un device capace di convertire l'energia elettrica (DC) in energia meccanica, e che rappresenta il meccanismo di locomozione del robot [ADP], [PAO].

Nuovamente, la sua velocità può essere controllata sfruttando la PWM.

Il motore utilizza sei pin GPIO del Raspberry Pi per controllare la velocità e la direzione.



Figura 2.4e. Un motore a corrente continua.

Nella parte anteriore del robot sono posizionati *due LED RGB* singoli (Figura 2.4-f), il cui utilizzo permette per l'appunto di produrre luce rossa, verde e blu. Per generare altri colori, è sufficiente mescolare i tre colori con diverse intensità. Ad esempio, per creare luce blu pura, è necessario impostare il LED rosso sull'intensità più alta e i LED verde e blu sull'intensità più bassa. La luce bianca, è ottenibile assegnando la massima intensità a tutti e tre i LED.

Come già affermato, per regolare l'intensità di ciascun LED, si utilizza la PWM [ADP], [PAO].



Figura 2.4f. Un singolo LED RGB.

All'interno del Mars Rover PiCar-B sono presenti quattro moduli hardware WS2812: due disposti sul retro e tre nella parte inferiore del robot. (Figura 2.4g)

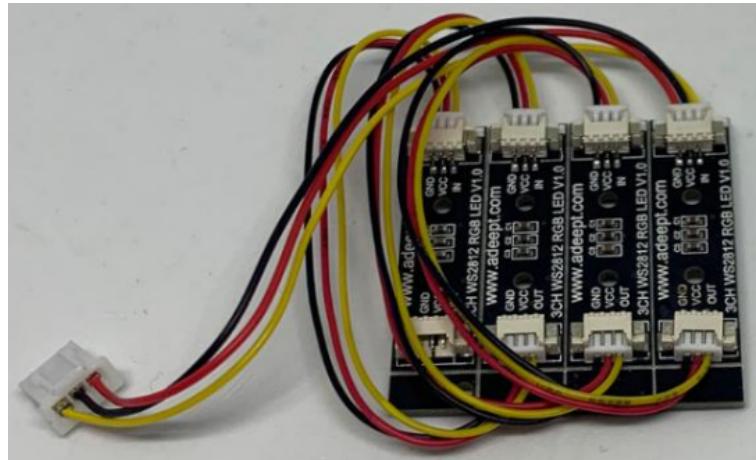


Figura 2.4g. I 12 LED RGB programmabili residenti nel Mars Rover PiCar-B.

Ogni modulo a sua volta contiene **3 LED RGB programmabili** e un chip di controllo della corrente integrato. Il pin di riferimento del GPIO per tutti i led è il 12.

I LED RGB, chiamati *Pixel LED*, sono attraversati da un unico bus seriale e sono collegati in serie: se uno si spegne, si spengono anche tutti gli altri.

Questo perché il Raspberry Pi controlla i LED inviando un singolo flusso seriale (sincronizzato con precisione) di dati e impulsi che attraversa i LED.

Per via del complesso sistema operativo Raspberry Pi che è Linux-based, multitasking e che implementa il meccanismo della *preemption* (che implica che qualsiasi attività, anche quelle dell'utente possa essere interrotta), tali impulsi non possono essere generati in modo sufficientemente accurato utilizzando i segnali del GPIO.

A causa di questo fatto, il flusso seriale che attraversa i LED a Pixel può essere ritardato e danneggiato. Questo danneggiamento si verifica sulla stringa di Pixel quando alcuni singoli LED non sono impostati sul colore corretto. Fortunatamente, il modulo DMA (accesso diretto alla memoria) può trasferire byte di memoria tra parti del processore senza utilizzare la CPU. Utilizzando il DMA per inviare una specifica sequenza di byte al modulo PWM, il segnale dati Pixel può essere generato senza essere interrotto dal sistema operativo del Raspberry Pi. In altre parole, l'uso del DMA consente di inviare i dati ai LED senza la partecipazione del sistema operativo del Pi, quindi nessuna corruzione [ADP], [PAO].

Di estrema importanza risulta essere il **Sensore ultrasonico** (Figura 2.4-h): un modulo che fornisce un rilevamento della distanza senza contatto da un valore minimo di 2 cm ad un

massimo di 400 cm, con una precisione di 3 mm. Al suo interno sono presenti: un trasmettitore ad ultrasuoni, un ricevitore ed un circuito di controllo [ADP], [PAO], [ULT].



Figura 2.4h. Il sensore di distanza ad ultrasuoni HC-SR04 situato nel robot in esame.

Detto sensore lavora usando 4 pin:

1. Gnd (massa)
2. Trig: “Trigger” che si attiva al momento dell’ invio del segnale ad ultrasuoni.
3. Echo: pin che produce un impulso che si interrompe quando viene ricevuto il segnale ad ultrasuoni.
4. Vcc: pin collegato alla tensione di alimentazione di 5V.

Un segnale di alto livello (5V) di almeno 10 μ s (microsecondi) è applicato al pin Trig.

Il modulo trasmette 8 onde quadre ultrasoniche a 40 kHz, che iniziano a viaggiare nell’aria circostante allontanandosi dal sensore, e rileva automaticamente se c’è un ritorno del segnale. Il segnale sull’Echo intanto diventa alto, e inizia a registrare il tempo di ritorno attendendo l’onda ultrasonica riflessa.

La durata del livello alto è il tempo che intercorre dall’emissione dell’onda ultrasonica fino al suo ritorno e, se dopo 38 ms (millisecondi) l’impulso non viene riflesso, il segnale sul pin Echo tornerà basso, per indicare l’assenza di ostacoli.

Altrimenti se vi è un oggetto che riflette all’indietro le onde ultrasoniche, il segnale sul pin Echo diventa basso e terminerà la registrazione della sua durata.

Il tempo così ottenuto servirà per calcolare la distanza tra il Mars Rover e l’oggetto che ha davanti. Considerando che l’onda ultrasonica ha percorso per due volte quello spazio e che la velocità di propagazione delle onde nell’aria è di circa 340 m/s, allora la distanza risulterà:

$$s = \frac{340}{2} * t$$

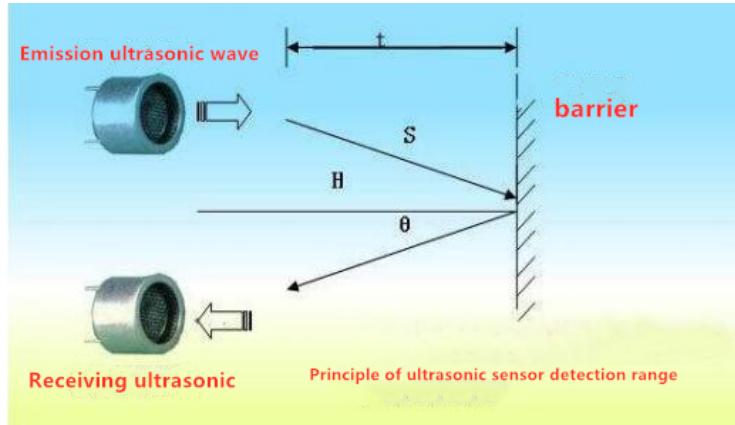


Figura 2.4-g Funzionamento di rilevazione della distanza. Da notare è che se l'ostacolo si trova di fronte al sensore o in un ipotetico settore circolare di 30° d'ampiezza (15° da ambo i lati rispetto alla direzione frontale) si ottengono misure più accurate.

2.5 Software utilizzati

Per il Raspberry Pi sono disponibili molti linguaggi di programmazione. In generale, qualsiasi linguaggio che può essere compilato per l'architettura ARM (come il linguaggio C) può essere utilizzato al suo interno. Nello specifico, si è fatto uso di *Python*, un linguaggio di alto livello orientato agli oggetti, interpretato, multiplattforma, a tipizzazione dinamica, che utilizza diversi paradigmi di programmazione e che consente la creazione di programmi suddivisi in moduli funzionalmente indipendenti. Con Python sono disponibili in modo predefinito numerose strutture dati di alto livello, che fanno risparmiare al programmatore il tempo per la loro implementazione [PYT].

Oltre ai tanti linguaggi di programmazione, esistono anche numerosi sistemi operativi che il Raspberry Pi è in grado di supportare, ad esempio: Windows 10 IoT Core, Ubuntu MATE, OSMC, LibreELEC, PiNet, RISC OS etc. Quello più proposto è *Raspbian*, lo stesso Sistema Operativo utilizzato nel lavoro di tesi (“Raspbian GNU/Linux 11 - bullseye”). Quest’ultimo è ospitato in una micro SD preferibilmente da 16 GB. Invece, gli sviluppi software sono stati svolti all’interno di un Personal Computer al cui interno gira il SO: “Ubuntu 22.04.1 LTS”,

mentre per la parte di testing del software, è stato d'ausilio un ulteriore computer ospitante Windows 11.

Per la parte di programmazione si è adoperato l'IDE Visual Studio Code e, di notevole utilità è stata la estensione “Remote - SSH v0.84.0”, che ha permesso la navigazione remota e sicura all'interno del file system del Raspberry Pi.

All'interno del Raspberry Pi come ambiente di sviluppo integrato si è utilizzato: “Thonny”.

Saranno ora presentate le varie librerie utilizzate nel corso del lavoro.

2.5.1 Requests



Requests ha permesso lato client di poter inviare richieste HTTP/1.1 in modo semplice e pratico. È particolarmente apprezzata e utilizzata dal momento che risulta utile nei casi d'uso più comuni, consentendo ad esempio di scrivere programmi in grado di connettersi ad Internet per inviare e ricevere dati verso siti web e REST API [REQ].

2.5.2 Tkinter



Tkinter ha avuto applicazione nel lato client del lavoro di tesi e rappresenta l'insieme dei moduli che si basano sulle librerie Tcl/Tk che il linguaggio Python usa per realizzare e gestire le GUI. Tcl (Tool Command Language) è un linguaggio di programmazione utilizzato su piattaforme diverse, per una grande varietà di applicazioni. Tk è l'insieme degli strumenti software (toolkit) per le interfacce grafiche di Tcl. Python interagisce con questi software attraverso tkinter, il modulo software che avvolge le librerie Tcl/Tk rendendole disponibili al programmatore. Tkinter è di fatto lo standard che definisce un insieme di elementi grafici che sono indipendenti dalla piattaforma su cui vengono utilizzati. Questo meccanismo garantisce la portabilità delle applicazioni python tra piattaforme diverse [TKI].

2.5.3 Flask



Flask è un web framework open source che è stato impiegato nel lato server del progetto di tesi, caratterizzato da flessibilità, leggerezza e semplicità d'uso. È più propriamente considerato un micro-framework per via del suo approccio allo sviluppo minimalista non opinionato, che lascia agli sviluppatori la possibilità di scegliere quando e soprattutto come implementare ogni aspetto delle proprie applicazioni web a partire dalla struttura del

progetto, scegliendo quali funzionalità è davvero necessario implementare, quanti e quali file/moduli creare, le convenzioni di sviluppo a cui fare riferimento etc. Possiede un core semplice e performante che mette a disposizione funzionalità fondamentali quali server di sviluppo e debugger, routing, supporto unit testing integrato, protezione contro cross-site-scripting (XSS) e l'impiego di Jinja 2 come template engine. A differenza dei web framework full stack, Flask non include ad esempio un sistema di autenticazione o validazione dei form.

Questo genere di funzionalità vengono incluse in una web app o sito scritto con Flask tramite l'utilizzo di estensioni dedicate, che una volta integrate nel progetto potranno essere usate facilmente come se facessero parte del framework stesso [FLK].



2.5.4 Altre librerie

adeept_picar-b: è la libreria principale contenente tutti i file Python necessari al controllo del Mars Rover PiCar-B. Tale libreria deve essere clonata all'interno del sistema operativo del Raspberry Pi. In particolare, “server” è il direttorio che si è consultato maggiormente durante l'attività di Tesi [LIB01].

Adafruit_PCA9685: è la libreria necessaria per la comunicazione con il PCA9685. Ad oggi deprecata e sostituita con “Adafruit_CircuitPython_PCA9685” [LIB02].

RPi.GPIO: è una specifica libreria che consente di configurare in modo semplice i pin GPIO sia in lettura sia in scrittura all'interno di uno script Python [LIB03].

In alternativa esiste anche gpiozero.

rpi_ws281x: libreria per controllare singolarmente i LED WS2812 la quale, facendo uso di un driver complesso, utilizza l'interfaccia DMA (accesso diretto alla memoria) sul Raspberry Pi per generare gli impulsi, risolvendo il problema descritto nel paragrafo precedente [LIB04].

2.5.5 Software

Alcuni programmi che si sono sfruttati sono:

VNC Viewer: consente di accedere da remoto a qualsiasi computer Windows, Linux o Mac. Questo software ha permesso di remotizzare il controllo del Mars Rover PiCar-B, eliminando l'onere di collegare periferiche di input/output quali schermo, tastiera e mouse utilizzando, di fatto, quelle del pc portatile.

Insomnia: è un client API con interfaccia grafica, multipiattaforma ed open source per servizi come: GraphQL, gRPC e REST.

È difatti risultato molto d'aiuto per la fase di test della API Rest.

Lo scopo di Insomnia è principalmente quello di verificare se un determinato metodo è stato implementato correttamente, se risponde nel giusto modo alle richieste di un client, evitando, almeno per la prima fase di sviluppo del software, di andarsi a scrivere un client poi contatti l'applicazione.

Oltre ad Insomnia, esistono altri strumenti, come Postman che consentono di effettuare diversi tipi di richieste a un qualsiasi server Rest.

Capitolo 3. Programmazione del sistema Mars Rover PiCar-B

Questo capitolo si propone di assumere un approccio più pratico rispetto ai precedenti.

Infatti, saranno implementati i concetti e i meccanismi introdotti per poter concretizzare il lavoro di tesi, avente come fine ultimo l'analisi e la programmazione del sistema Mars Rover PiCar-B.

Nella prima parte sarà analizzato il codice Python che consente il controllo delle singole componenti del robot all'interno del file system locale del Raspberry Pi.

La seconda parte prevede invece l'implementazione di una semplice ReST API che consente di controllare il robot da remoto, al fine di fargli attuare delle azioni specifiche.

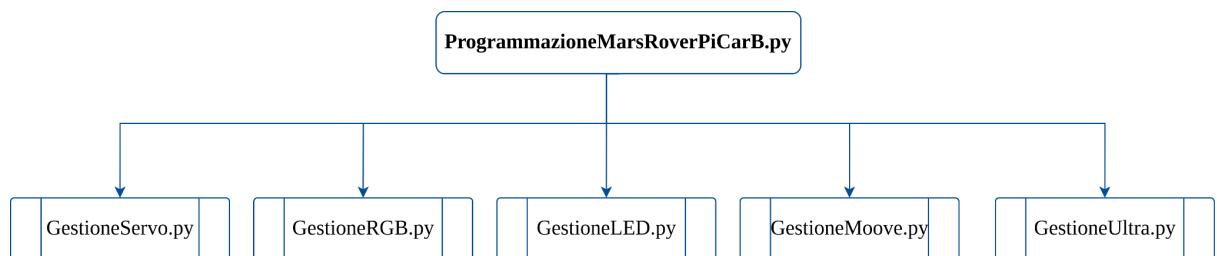
Saranno riportate solamente le parti più significative del codice Python.

3.1 Analisi e sperimentazioni nel concentrato

Si è scelto di progettare il software seguendo un approccio top-down, creando tanti moduli Python quante sono le parti del robot da esaminare. All'interno di ogni modulo sono state raggruppate più funzioni, ognuna afferente ad uno specifico comportamento della parte robotica di interesse. A loro volta, tali procedure utilizzano delle funzioni *built-in* della libreria “`adeept_picar-b`”.

Con questo sistema, l'intento è quello di facilitare il lavoro di manutenzione di questo software, offrendo la possibilità di intervenire con modifiche o correzioni su un solo sottoprogramma, avendo allo stesso tempo cognizione di quello che fa l'intero programma.

Il seguente diagramma a blocchi riassume l'organizzazione del programma in moduli.



Ogni modulo è opportunamente documentato così come le funzioni che contiene, seguendo la convenzione delle *docstring*.

Decidendo di seguire l'ordine di presentazione delle componenti del Mars Rover PiCar-B

adottato nel Capitolo 2, si inizia dando uno sguardo al codice python di gestione di un *servomotore*, presente nel file “GestioneServo.py”

```
"""
Modulo: GestioneServo.py
Autore: Marco Beltrame
...
Gestione delle principali operazioni ...
"""

import ...

pwm = Adafruit_PCA9685.PCA9685()
pwm.set_pwm_freq(50)

def giraRuoteDestra(velocita=20, inclinazione=300):
    for i in range(0,100, velocita):
        pwm.set_pwm(3,0,(inclinazione+i))
        sleep(0.05)
    pwm.set_pwm(3,0,0)

def abbassaTesta(velocita, inclinazione=100):
    for i in range(0, 100, velocita):
        pwm.set_pwm(2, 0, (inclinazione-i))
        sleep(0.05)
    pwm.set_pwm(2, 0, 0)

def clean_all():
    pwm.set_pwm_freq(50)
    pwm.set_all_pwm(0, 0)
...  
...
```

Dopo aver importato le librerie indispensabili al funzionamento delle funzioni presenti nel modulo, si va ad istanziare l’oggetto `pwm` che permetterà la comunicazione con il chip PCA9685, necessario per gestire la PWM. Successivamente, in accordo con il modello di servo utilizzato, si imposta la frequenza per il segnale PWM, in questo caso pari a 50 Hz.

In generale, l’istruzione `pwm.set_pwm(<porta>, <acceso>, <spento>)` serve per controllare la rotazione del servo in una determinata posizione:

`<porta>` rappresenta il numero di porta, il canale a cui è collegato il servo nella scheda driver Robot HAT;

`<acceso>` è il valore di deviazione per il controllo della rotazione dello sterzo, in questo caso settato sempre a 0;

`<spento>` è il valore del ciclo di lavoro che si desidera impostare.

Più in dettaglio, `set_pwm()` setta il singolo canale PWM, calcolando degli specifici valori di 8-bit e dei registri specifici del bus I2C sulla base dei parametri passatagli, e scrive tali valori in quei registri. Nello specifico, è stato scelto di collegare i servomotori utilizzati ai canali 1 (per il movimento della testa lungo l'asse orizzontale), 2 (per il movimento della testa lungo l'asse verticale), 3 (per il movimento da sinistra a destra e viceversa delle ruote).

Da sola, l'istruzione appena menzionata non consente di controllare la *velocità di rotazione* dello sterzo, motivo per cui è stata inserita all'interno di un ciclo for. Il parametro formale `velocita`, fornisce la granularità necessaria per raggiungere il limite superiore del ciclo (in questo caso impostato sempre a 100). La velocità di movimento di un servo è pertanto inversamente proporzionale al valore assunto dallo step del ciclo for. Infine, per ottenere l'effetto desiderato (far oscillare lo sterzo lentamente avanti e indietro tra due posizioni) è opportuno temporizzare un'iterazione dalla successiva con un valore inferiore al secondo, (ad esempio di 5ms). La funzione `clean_all()`, il cui codice viene eseguito qualora fosse ricevuto un segnale SIGINT, consente di spegnere il segnale PWM di tutti i 16 canali del PCA9685 dopo aver impostato la frequenza a 50Hz. Durante l'esecuzione di un movimento il flusso di controllo del programma risulterà bloccato. Per ovviare a questo inconveniente esistono soluzioni più eleganti ma complesse che fanno uso del multi-threading.

Il codice python per pilotare un motore DC (scritto in ‘GestioneMove.py’), che permette l'avanzamento in avanti oppure indietro dell'auto robotica è il seguente:

```
import RPi.GPIO as GPIO
...
Motor_A_EN    = 7
Motor_B_EN    = 11
Motor_A_Pin1  = 8
Motor_A_Pin2  = 10
Motor_B_Pin1  = 13
Motor_B_Pin2  = 12
Dir_avanti   = 0
Dir_indietro  = 1
stato_rotazione = 1 #Rotazione motore (se non ruota allora è fermo)
avanti       = 1      #Motore avanti
indietro     = 0      #Motore indietro
START_MOTORE = 1
STOP_MOTORE = 0

def setup():#Inizializzazione del motore
    global pwm_A, pwm_B
    GPIO.setwarnings(False)
    GPIO.setmode(GPIO.BOARD)
```

```

GPIO.setup(Motor_A_EN, GPIO.OUT)
GPIO.setup(Motor_B_EN, GPIO.OUT)
GPIO.setup(Motor_A_Pin1, GPIO.OUT)
GPIO.setup(Motor_A_Pin2, GPIO.OUT)
GPIO.setup(Motor_B_Pin1, GPIO.OUT)
GPIO.setup(Motor_B_Pin2, GPIO.OUT)
try:
    pwm_A = GPIO.PWM(Motor_A_EN, 1000)
    pwm_B = GPIO.PWM(Motor_B_EN, 1000)
except:
    pass

def motorStop():
    GPIO.output(Motor_A_Pin1, GPIO.LOW)
    GPIO.output(Motor_A_Pin2, GPIO.LOW)
    GPIO.output(Motor_B_Pin1, GPIO.LOW)
    GPIO.output(Motor_B_Pin2, GPIO.LOW)
    GPIO.output(Motor_A_EN, GPIO.LOW)
    GPIO.output(Motor_B_EN, GPIO.LOW)

def motore_sinistra(stato_rotazione, direzione, velocita):
    global pwm_A
    if stato_rotazione == 0: #Se non c'è rotazione,
        motorStop() # allora il motore non produce energia meccanica
    else:
        if direzione == Dir_avanti:
            GPIO.output(Motor_A_Pin1, GPIO.HIGH)
            GPIO.output(Motor_A_Pin2, GPIO.LOW)
            pwm_A.start(100)
            pwm_A.ChangeDutyCycle(velocita)
        elif direzione == Dir_indietro:
            GPIO.output(Motor_A_Pin1, GPIO.LOW)
            GPIO.output(Motor_A_Pin2, GPIO.HIGH)
            pwm_A.start(0)
            pwm_A.ChangeDutyCycle(velocita)
    return direzione

def liberaRisorse():
    motorStop()
    GPIO.cleanup()

def avanti(velSinistra=100, velDestra=100):
    setup()
    motore_sinistra(START_MOTORE, avanti, velSinistra)
    time.sleep(1.0)
    motore_sinistra(STOP_MOTORE, avanti, velSinistra)
    liberaRisorse()

def retromarciaTemporizzata(secondi=5):
    timeout = time.time() + secondi
    while True:

```

```
sleep(0.05)
retromarcia(85,85)
if time.time() > timeout:
    break
...
...
```

In prima analisi, le funzioni offerte dalla libreria di Adeept sono state rivisitate e sono state aggiunte funzionalità personalizzate, come la possibilità di andare in retromarcia (**retromarciaTemporizzato()**) per un determinato numero di secondi (di default 5).

Per il motivo discusso nel capitolo precedente, si possono dover gestire due motori, quindi si hanno due comandi per loro il funzionamento (due per l'avvio e due per l'arresto).

Le prime righe del codice definiscono i pin di interesse e la loro numerazione nella scheda GPIO (che può differire a seconda del tipo di Raspberry).

La funzione **motorStop()** permette al robot di fermare la sua rotazione, andando ad impostare lo stato dei pin ad un livello logico basso con le istruzioni: **GPIO.output(Motor_*_Pin*, GPIO.LOW)**. La procedura **setup()** si occupa di inizializzare i pin.

Il comando **GPIO.setmode(GPIO.BOARD)** usa un sistema di numerazione dei pin che si riferisce ai numeri dei pin presenti sull'intestazione P1 della scheda del Raspberry Pi.

Il vantaggio di *BOARD* è che l'hardware funzionerà sempre, indipendentemente dalla revisione della scheda Raspberry che si ha; non sarà necessario ricablarlo o modificare il codice. Le istruzioni: **GPIO.setmode(Motor_*_*, GPIO.OUT)** impostano i pin specificati come uscite; le due istruzioni **GPIO.PWM(Motor_*_EN, 1000)** generano un segnale PWM sui pin specificati ad una frequenza di 1000Hz, necessario per controllare la velocità del motore. La funzione **motore_sinistra()** si occupa di controllare il motore connesso alla porta di sinistra del Motor HAT; **direzione** è il parametro che indica il verso di rotazione del motore e può assumere i valori 0 o 1 che fanno andare l'auto avanti o indietro. Il parametro **velocita** ammette valori da 0 (lento, fermo) a 100 (massima velocità) ed è gestito dal comando **ChangeDutyCycle(velocita)**, che modifica il ciclo di servizio del segnale PWM. La procedura **avanti()**, essenzialmente fa uso delle istruzioni precedenti per avviare in avanti il motore di sinistra, attendere 1 secondo e spegnerlo.

Il seguente script si occupa invece dei “fari” del Mars Rover (il cui codice è reperibile nel file ‘GestioneRGB.py’).

```
import ...

left_R = 15, left_G = 16, left_B = 18
right_R = 19, right_G = 21, right_B = 22
on = GPIO.LOW, off = GPIO.HIGH

def both_on():
    GPIO.output(left_R, on)
    GPIO.output(left_G, on)
    GPIO.output(left_B, on)
    GPIO.output(right_R, on)
    GPIO.output(right_G, on)
    GPIO.output(right_B, on)

def both_off():
    GPIO.output(left_R, off)
    GPIO.output(left_G, off)
    GPIO.output(left_B, off)
    GPIO.output(right_R, off)
    GPIO.output(right_G, off)
    GPIO.output(right_B, off)

def setup():
    GPIO.setwarnings(False)
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(left_R, GPIO.OUT)
    GPIO.setup(left_G, GPIO.OUT)
    GPIO.setup(left_B, GPIO.OUT)
    GPIO.setup(right_R, GPIO.OUT)
    GPIO.setup(right_G, GPIO.OUT)
    GPIO.setup(right_B, GPIO.OUT)
    both_off()

def side_on(side_X):
    GPIO.output(side_X, on)

def side_off(side_X):
    GPIO.output(side_X, off)

def green():
    side_on(right_G)
    side_on(left_G)

def yellow():
    red()
    green()

def pink():
    red()
    blue()
```

```
def blu(tempoDiAccensione=2):
    setup()
    blue()
    sleep(tempoDiAccensione)
    both_off()
    ...
```

Anche qui si è sfruttata la libreria RPi.GPIO. Le funzioni `both_on()` e `both_off()` accendono e spengono rispettivamente tutti i pin GPIO che pilotano i led di sinistra e destra. Il funzionamento di `setup()` è analogo a quello visto per il motore DC.

Le procedure `side_on(side_X)` e `side_off(side_X)` rispettivamente accendono e spengono il led desiderato; la ‘X’, presente nel nome del loro argomento si riferisce al led di sinistra o di destra.

Ad esempio, la funzione `green()` accende il led RGB di destra e lo rende di colore verde usando il giusto pin, e fa lo stesso con il pin di sinistra. È risaputo che nel sistema RGB ogni colore è definito da tre componenti: il rosso, il verde e il blu. Ognuna di queste componenti può assumere un valore numerico compreso tra 0 e 255. Un valore basso indica che la componente incide poco sul colore, mentre un valore alto significa che il colore usa molto quella componente. Sfruttando i 3 colori di base è possibile ottenerne altri. Questa politica è implementata ad esempio nelle funzioni `yellow()` e `pink()`.

Nuovamente, è sempre possibile customizzare una funzione per ottenere il comportamento desiderato. Ad esempio, la funzione `blu()` accende un led di colore blu per il tempo desiderato. Combinando le funzioni `side_on()`, `sleep()` e `both_off()` all'interno di cicli `for` è possibile ottenere un effetto di colori lampeggianti ad intermittenza, chiamato “police”, gestito dell'omonima funzione all'interno della libreria `adeep_picar-b`.

Per programmare i *Pixel LED* invece si è usata la libreria di terze parti `rpi_ws281x`. Sono di seguito riportati alcuni dei comandi più rappresentativi, sperimentati nel modulo ‘`GestioneLED.py`’:

```
1. led = LED()
2. led.colorWipe(ROSSO,VERDE, BLU)
3. sleep(4)
4. led.colorWipe(0,0,0)
5. led.strip.setPixelColor(i, Color(ROSSO, VERDE, BLU))
6. led.strip.show()
7. led.colorWipe(0,0,0)
8. led.rainbow(wait_ms=20, iterazioni=1)
9. led.theaterChaseRainbow(wait_ms=20)
```

Dopo aver dichiarato un oggetto della classe LED presente nella libreria, con la seconda linea è possibile andare ad accendere tutti i led di un modulo WS2812 specificando le intensità dei 3 colori di base. La terza riga consente di persistere l'effetto appena ottenuto per 4 secondi. Le righe 4 e 7 si occupano di spegnere tutti i led. Le righe 5 e 6 permettono di pilotare un singolo led: nella funzione `setPixelColor()` il parametro `i`, i cui valori assunti sono compresi tra 0 e 2, rappresenta quale tra i 3 pin del modulo andare ad azionare.

La procedura `rainbow()` disegna un “arcobaleno” che sfuma su tutti i pixel contemporaneamente. Sebbene su ciascun Pixel siano presenti solo LED RGB, il driver crea molti colori diversi variando la luminosità di ciascun LED individualmente. `wait_ms` imposta un ritardo (in millisecondi) tra ogni cambio di colore. Il parametro `iterazioni` imposta quante volte ripetere un ciclo di colore completo (ogni ciclo di colore ha 256 diversi colori). `theaterChaseRainbow()` genera al contempo un arcobaleno e un effetto police.

Infine, per la gestione del sensore ultrasonico si è usato il seguente codice:

```
import ...

Tr = 11 # Numero del pin della parte finale di input del sensore ultrasonico
Ec = 8 # Numero del pin della parte finale di output del sensore ultrasonico

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(Tr, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(Ec, GPIO.IN)

def calcolaDistanza(*args):
    GPIO.output(Tr, GPIO.HIGH)
    time.sleep(0.000015) # almeno 10 ms
    GPIO.output(Tr, GPIO.LOW)
    while not GPIO.input(Ec): #
        pass
    t1 = time.time()
    while GPIO.input(Ec):
        pass
    t2 = time.time()
    return round((t2-t1)*340/2,2)

def calcolaDistanzaMedia(tempoDiCalcolo=15):
    vettoreDistanze = [] # contiene tutte le distanze
    timeout = time.time() + tempoDiCalcolo
    while True:
        sleep(1)
        distanza = calcolaDistanza()
```

```

    print ("Distanza = {:.3f}cm ".format( distanza*100))
    vettoreDistanze.append(distanza)
    if time.time() > timeout:
        break
    return sum(vettoreDistanze)/len(vettoreDistanze)

```

Ancora una volta è si è utilizzata la libreria RPi.GPIO.

Osservando il codice, vengono dichiarati ed inizializzati i pin Trigger e Echo.

Questa volta, dentro alla funzione `setup()` è adottato il sistema di numerazione *BCM*: quest'ultimo lavora ad un livello inferiore rispetto a *BOARD* ed elenca i pin in base ai numeri del canale sul SOC Broadcom. Il pin Tr è poi impostato in uscita ad un livello logico iniziale nullo, mentre il pin Ec è settato come pin in ingresso.

La funzione `calcolaDistanza()` contiene la logica applicativa per il calcolo della distanza tra l'ostacolo e il modulo ultrasonico, rispettando le condizioni introdotte nel paragrafo 2.4.

Prendendo come base di partenza tale funzione, è possibile sfruttare il sensore contenuto nell'oggetto Mars Rover per creare altri tipi di report, come ad esempio il triviale calcolo della media tra varie distanze calcolate in un determinato lasso di tempo, realizzato da `calcolaDistanzaMedia()`.

Tutte le funzioni sono state testate all'interno del modulo ‘`ProgrammazioneMarsRoverPiCarB.py`’:

```

from gestioneLED import *
from gestioneServo import *
from gestioneMoove import *
from gestioneRGB import *
from gestioneSensoreUltrasonico import *
...

if __name__ == "__main__":
    try:
        alzaTesta(20)
        sleep(2)
        abbassaTesta(20,50)
        GPIO.cleanup() # Serve per eliminari conflitti tra BCM e BOARD
        rosso()
        sleep(1.5)
        avanti()
        GPIO.cleanup() # Serve per eliminari conflitti tra BCM e BOARD
        sleep(0.5)
        verde(3)
        retromarcia()

```

```

sleep(1.5)
ciano()
sleep(0.5)
abbassaTesta(30,450)
sleep(1)
giallo()
avantiTemporizzato(2)
sleep(0.5)
abbassaTesta(10)
sleep(0.5)
stampaDistanza(calcolaDistanza())
GPIO.cleanup() # Serve per eliminare conflitti tra BCM e BOARD
giraRuoteDestra()
rosa()
# GESTIONE DEI LED PIXEL
led.colorWipe(0, 255, 255) # Cambia il colore di tutti e 3 i LED
sleep(4)
led.strip.setPixelColor(1, Color(165, 42, 42)) # Accensione singolo
led.strip.show()
sleep(4)
led.colorWipe(0,0,0) # Spegnimento LED
clean_all()
except KeyboardInterrupt:
    clean_all()

```

3.2 Programmazione del Mars Rover PiCar-B nel distribuito

Inizia ora la seconda parte del lavoro di tesi, che consiste nel *riutilizzare* le stesse funzioni analizzate nel paragrafo precedente e che operano nell'ambiente locale, anche nel *distribuito*, attraverso l'implementazione di un'applicazione full stack (realizzando sia la parte client, sia la parte server) conforme ai vincoli dello stile architetturale REST in python.

L'idea che soggiace all'applicazione è riportata di seguito (Figura 3.2a).

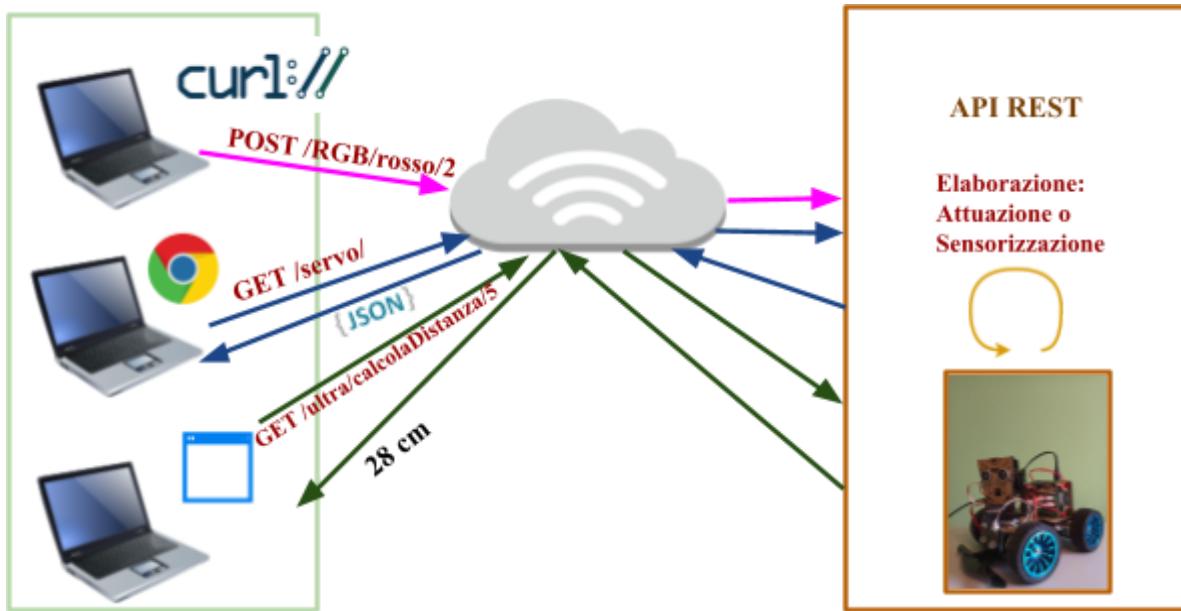


Figura 3.2a. Architettura del sistema remoto realizzato.

Il funzionamento è semplice: in un qualunque tipo di rete (anche nella rete Internet), vari User Agent che si trovano in un qualsivoglia sistema periferico remoto, effettuano delle chiamate REST al Server che sta eseguendo all'interno del sistema Mars Rover PiCar-B, il quale, una volta elaborata l'attuazione o la sensorizzazione richiesta, ritorna un messaggio di risposta o l'informazione domandata ai Client.

La precedenza è stata data alla realizzazione del server in Flask. Ai fini dell'applicazione, si è deciso di utilizzare i soli verbi GET e POST per reperire e aggiornare lo stato delle risorse. Come formato di rappresentazione delle informazioni invece è stato scelto JSON. Di seguito sono riportate tutte le operazioni che la ReST API è in grado di supportare.

Metodo HTTP	URI	Operazione
GET	/RGB	Il client vuole l'elenco dei colori RGB disponibili
GET	/moove	Il client vuole ottenere la lista dei movimenti disponibili
GET	/servo	Il client desidera reperire l'elenco di tutti i movimenti del servomotore
GET	/ultra	Il client richiede la lista di tutte le operazioni offerte dal sensore ultrasonico
GET	/ultra/<operazione>/<durata>	Il client vuole avviare il sensore ad ultrasuoni per calcolare un risultato che poi visualizzerà
POST	/RGB/<colore>/<durata>	Il client vuole accendere un LED RGB frontale di un determinato colore e per un determinato numero di secondi.

POST	/RGB/spegni	Il client richiede lo spegnimento di tutti i LED RGB
POST	/moove/<movimento>/<velocita>	Il client chiede l'attuazione di un movimento ad una certa velocità
POST	/moovetemporizzata/<movimento>/<tempo>	Il client fa richiesta di attuare un movimento per un certo numero di secondi
POST	/servo/<movimento>/<velocita>/<inclinazione>	Il client demanda l'azionamento di un servo ad una certa velocità
POST	/led/<R>/<G>/	Il client richiede l'accensione di un LED in pixel
POST	/led/<R>/<G>//<led_i>	Il client vuole accendere uno solo dei 3 LED in pixel
POST	/led/spegni	Il client vuole spegnere tutti i LED in pixel

Tabella 3.2. Le principali operazioni fornite dal server RESTful. Si noti che le stringhe presenti tra parentesi angolari sono libere di assumere un qualsiasi valore.

Flask lavora con i cosiddetti *decoratori* (delle righe che iniziano con il carattere '@'), grazie ai quali permette di mappare una richiesta HTTP (in particolare, sia il metodo che l'URL) ad una determinata *funzione* scritta dal programmatore.

Tale funzione, ovviamente definisce che cosa andrà ad effettuare il codice alla ricezione di una GET, una POST, ..., e in generale di un qualsiasi altro metodo.

Quello che segue è il codice necessario per recuperare un qualsiasi elenco, ad esempio la lista dei movimenti dei servomotori presenti nel robot.

```
...
listaServo = ["alzaTesta", "abbassaTesta", "giraRuoteDestra"]
...
@app.get("/servo")
def get_movimenti_servo():
    return jsonify(listaServo), 200
...
```

Per interagire con delle risorse (per recuperarle, modificarle, inserirle, etc.) si ha bisogno di una sorgente di dati. Per semplicità e senza particolari esigenze, si è deciso di memorizzare i dati a livello di applicazione, “simulando” il comportamento di un database tramite l’uso delle liste e dei dizionari in python.

Il matching tra l'URL che viene richiesto da un client remoto e come gestirlo è realizzato in modo semplice e anche visivamente immediato grazie alla riga: `@app.get("/servo")`, con la quale si sta indicando a Flask che GET è il metodo a cui passare tutte le richieste che arrivano

in questo server, all'URL "/servo" e che tali richieste sono gestite dalla funzione sottostante. In questo caso, la funzione `get_movimenti_servo()` restituisce al client l'elenco richiesto in formato JSON, con stato HTTP 200.

E' possibile usare la stessa idea per tutti gli URL desiderati e per tutti i verbi HTTP che si vogliono implementare.

Con la stessa procedura sono state scritte le funzioni associate alle prime quattro operazioni (che usano il metodo GET) della tabella 3.2.

Per avviare remotamente la sensorizzazione ultrasonica e ottenerne il risultato, si è previsto il seguente codice:

```
...
sensorizzazioni = { "calcolaDistanza": calcolaDistanza,
                     "calcolaDistanzaMedia": calcolaDistanzaMedia }
...
@app.get("/ultra/<operazione>/<durata>")
def get_risultato_ultra(operazione, durata):
    durata = int(durata)
    if operazione in sensorizzazioni:
        try:
            return jsonify(stampa(sensorizzazioni[operazione](durata))), 200
        except ValueError:
            GPIO.cleanup() # Per eliminare conflitti tra BCM e BOARD
            print("Cambio settaggi per il GPIO")
            return jsonify(stampa(sensorizzazioni[operazione](durata))), 200
    else:
        return {"messaggio": "Operazione non implementata."}, 404
...
```

In questo caso il client vuole ottenere il risultato di una certa operazione che può offrire il sensore ultrasonico con una determinata durata di tempo. Tale operazione e tale durata però possono variare: un utente potrebbe voler fare una GET sull'URI "/ultra/calcolaDistanza/10" ma anche su "/ultra/calcolaDistanzaMedia/12345" etc.

Pertanto, il modo per indicare al framework di comprendere qual è la giusta operazione e la specifica durata e passarla poi alla funzione che effettuerà le logiche desiderate per servire la richiesta del client, è quello di inserire nel decoratore i parametri di interesse tra parentesi angolari: `<operazione>`, `<durata>`.

Successivamente, alla funzione scritta nella riga successiva si passano come argomenti i nomi che sono stati conferiti ai parametri.

In particolare, la funzione `get_risultato_ultra()` utilizza un *dizionario* per accedere in modo simbolico alla funzione corretta presente nel modulo "GestioneUltra.py" e farla scatenare. Il

risultato ottenuto, formattato correttamente dalla procedura `stampa()`, è poi serializzato in JSON e restituito al client in caso di successo. Se invece l'operazione non è presente nel dizionario, allora l'utente riceverà una risposta contenente un messaggio con codice di stato “Not Found”.

Con questo approccio, parte dell'URL serve per identificare quale funzione dovrà gestire l'API, l'altra parte dell'URL diventa invece un input della funzione stessa.

Lo stesso Design Pattern in realtà è presente in molti altri framework di altri linguaggi di programmazione, ad esempio in Laravel con PHP.

Per poter eseguire remotamente attuazioni sul robot, l'API prevede l'uso del metodo POST.

Dal momento che la business logic è molto simile per le altre azioni, si riporta, arbitrariamente, l'insieme delle istruzioni per l'accensione di un LED RGB:

```
...
coloriRGB = {
    'rosso': rosso,
    'verde':verde,
    'blu': blu,
    'giallo':giallo,
    'ciano':ciano,
    'rosa':rosa
}
...
@app.post("/RGB/<colore>/<durata>")
def accendi_RGB(colore, durata):
    durata = int(durata)
    if colore in coloriRGB:
        try:
            coloriRGB[colore](durata)
        except ValueError:
            GPIO.cleanup()    # Per eliminare conflitti tra BCM e BOARD
            print("Cambio settaggi per il GPIO")
            coloriRGB[colore](durata)
        return {"messaggio": "Acceso il led RGB " + colore},200
    else:
        return {"messaggio": "Errore sulla scelta del colore"}, 404
...
```

Ad esclusione del tipo di decoratore coinvolto (in questo caso ‘post’), la semantica del codice risulta del tutto simile al precedente.

A causa di alcuni problemi con la libreria rpi_ws281x si è invece utilizzato un modo differente per programmare i LED in pixel. Considerando ad esempio il listato di istruzioni per gestire un singolo LED:

```
@app.post("/led/<R>/<G>/<B>/<led_i>")
def accendi_led_i(R, G, B, led_i):
    subprocess.call(['sh', 'avviaGestioneLED.sh', R, G, B, led_i])
    return {"risposta": "Acceso il LED "+led_i}, 200
```

Come si vede, viene invocata la procedura `subprocess()`, che si occupa di mandare in esecuzione il programma che le viene passato, con un numero indefinito di eventuali argomenti passati a linea di comando. Il codice del processo che viene attivato è all'interno del file “avviaGestioneLED.sh”, riportato di seguito.

```
#!/bin/bash

for i in "$@"
do
    echo $i
    cont=$((cont+1))
done
echo "In tutto sono stati raccolti: $cont parametri"
case $cont in
    1)
        echo `cat pwd.dat` | sudo -S python3 gestioneLED_SpegniTutti.py "$@"
        ;;

    3)
        echo `cat pwd.dat` | sudo -S python3 gestioneLED_AccendiTutti.py "$@"
        ;;

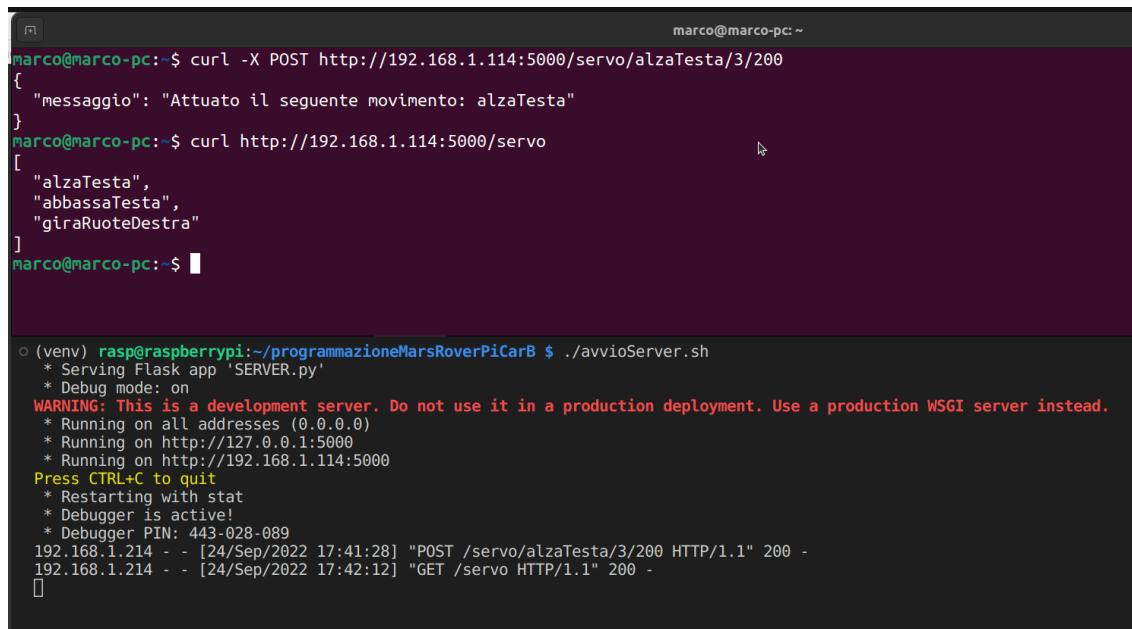
    4)
        echo `cat pwd.dat` | sudo -S python3 gestioneLED_Accendi_i_Esimo.py "$@"
        ;;

    *)
        echo -n "Passare il giusto numero di parametri"
        ;;
esac
```

A seconda dei parametri passati allo script, quest'ultimo a sua volta manda in esecuzione con privilegi di amministratore uno specifico modulo python avente un determinato scopo; inoltre, lo script passa al modulo anche la lista degli argomenti della linea di comandi.

In questo esempio, lo script python messo in esecuzione è: “gestioneLED_Accendi_i_Esimo.py”, dentro al quale vi sono gli specifici comandi per controllare il LED.

Per il collaudo della REST API è risultato comodo il comando `curl`, uno strumento per il trasferimento di dati da o verso un server e che supporta vari protocolli, tra cui anche HTML. Un esempio di utilizzo di questo comando è:



```
marco@marco-pc:~$ curl -X POST http://192.168.1.114:5000/servo/alzaTesta/3/200
{
  "messaggio": "Attuato il seguente movimento: alzaTesta"
}
[{"alzaTesta", "abbassaTesta", "giraRuoteDestra"}]
marco@marco-pc:~$ █

○ (venv) rasp@raspberrypi:~/programmazioneMarsRoverPiCarB $ ./avvioServer.sh
* Serving Flask app 'SERVER.py'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://192.168.1.114:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 443-028-089
192.168.1.214 - - [24/Sep/2022 17:41:28] "POST /servo/alzaTesta/3/200 HTTP/1.1" 200 -
192.168.1.214 - - [24/Sep/2022 17:42:12] "GET /servo HTTP/1.1" 200 -
█
```

Figura 3.2b. L’uso del comando `curl` da un computer remoto, connesso tramite SSH al Raspberry Pi in cui è in ascolto il server.

Il file “avvioServer.sh” contiene i comandi necessari all’avvio del server in Flask. Di default, con Flask viene aperta la porta 5000 in localhost.

Anche Insomnia ha avuto un ruolo centrale durante la fase di testing.

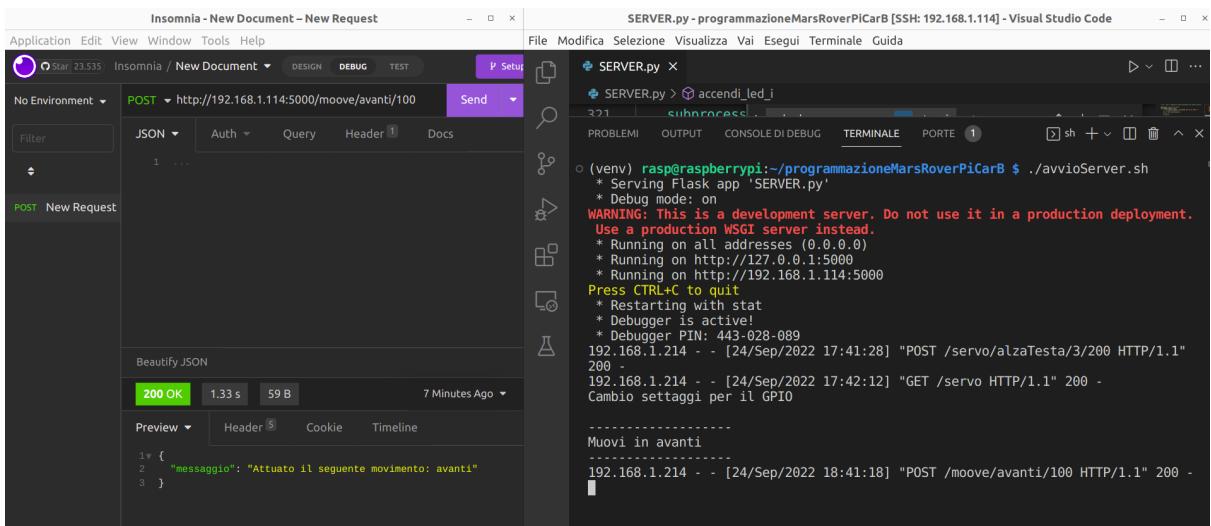


Figura 3.2c. Test del Server RESTful mediante il software Insomnia.

Facendo riferimento alla Figura 3.2b, andrà specificato l'URL dell'API che si andrà a contattare, il “Content-type: application/json”, dando così indicazione alla Rest API che si vogliono ricevere le informazioni in JSON e anche nel caso di invio di informazioni, queste saranno in formato JSON.

Dai menù a tendina, è possibile modificare il metodo e il formato della richiesta (o dell'informazione che si vuole inviare alla Rest API).

Una volta compilati a dovere tutti i campi, è possibile cliccare su “Send” per inviare la richiesta; nella sezione “Preview” si avrà un riscontro di quanto effettuato (in questo caso lo stato HTTP 200 OK , a seguito di una richiesta POST), sia il messaggio di risposta del server.

Il connubio tra requests e tkinter ha dato origine al lato Client dell'applicazione: è stato possibile infatti generare un “pannello di controllo” dotato di GUI, che consente di pilotare il Mars Rovers PiCar-B a distanza. Sarà ora esaminata la figura 3.2d.

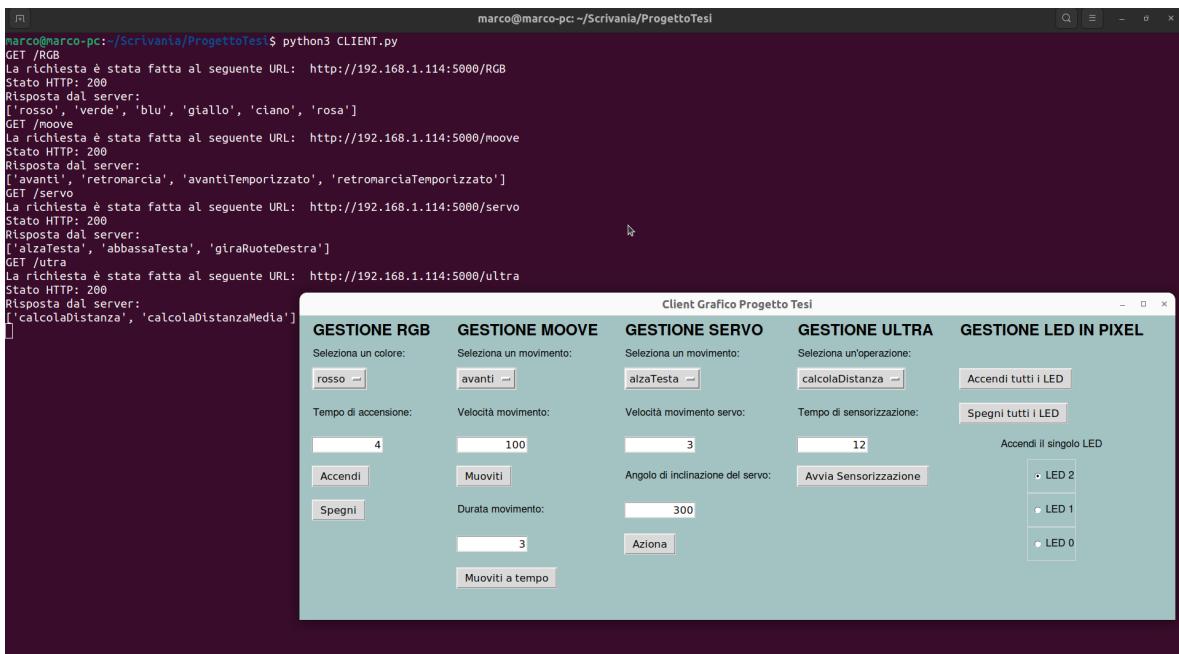


Figura 3.2d Client con interfaccia grafica per il controllo del Mars Rover nel distribuito.

Il programma è conforme alle scelte progettuali di modularizzazione introdotte all'inizio del capitolo, infatti il pannello è suddiviso in colonne, ognuna gestente un diverso gadget del robot. Quello che si può evincere dall'immagine è che, contestualmente all'avvio dello script, vengono fatte una serie di richieste di tipo GET su alcune *collezioni* presenti lato server; tali richieste sono necessarie per popolare i drop down menu⁵ presenti. Ad esempio, il recupero della lista dei colori RGB disponibili è effettuato dalle seguenti linee di codice:

```
import requests
...
api_url = "http://192.168.1.114:5000"
my_headers = {"Content-type": "application/json"}
...
print("GET /RGB")
richiestaListaRGB = requests.get(f"{api_url}/RGB", headers=my_headers)
print("La richiesta è stata fatta al seguente URL: ", richiestaListaRGB.url)
print(f"Stato HTTP: {richiestaListaRGB.status_code}")
print(f"Risposta dal server:\n{richiestaListaRGB.json()}")
richiestaListaRGB = richiestaListaRGB.json()
...
```

L'URL univoco sul quale fare la richiesta è costruito grazie all'ausilio delle *f-strings* di Python. Sono anche specificate le intestazioni di tale richiesta; in questo caso si desidera

⁵ Nel gergo di tkinter, i menù a tendina sono dei Widget di tipo *OptionMenu*.

comunicare tramite la rappresentazione - "Content-type"- di tipo JSON.

Tramite gli attributi dell'oggetto `richiestaListaRGB`, che viene popolato dalla libreria, è possibile ottenere tutti i dettagli di cui si ha bisogno circa la specifica richiesta effettuata, ad esempio lo status code HTTP, oppure la risposta che il server ha fornito, in formato JSON.

L'effetto delle prossime istruzioni è quello di popolare effettivamente il dropdown menù dei colori RGB. Con la riga 2 si sceglie l'elemento di default della lista recuperata da visualizzare in cima al menù. La riga 3 crea e si inizializza l'elemento grafico sulla base della lista. La riga 4 dà indicazioni sul posizionamento del menu all'interno della GUI.

```
1. variabileColoreRGB = StringVar(master)
2. variabileColoreRGB.set(richiestaListaRGB[0])
3. selezioneRGB = OptionMenu(master, variabileColoreRGB, *richiestaListaRGB)
4. selezioneRGB.grid(row=2, column=0, sticky=W, padx=20, pady=10)
```

Per fare in modo che alla pressione di un pulsante sia scatenata un qualsiasi tipo di attuazione o sensorizzazione, ad esempio per azionare il motore DC, è necessario il seguente codice:

```
button = Button(master, text="Muoviti",
command=lambda:print(requests.post(f"{api_url}/moove/{variabileMovimento.get()}/{variabileVelocitaMovimento.get()}").json())
button.grid(row=5, column=1, sticky=W, padx=20, pady=10)
```

Con lo stesso stile è stato scritto il codice di comando per gli altri bottoni presenti nella GUI del Client.

Si vuol far notare che il codice e le funzionalità del programma sono stati pensati per essere aperti a modifiche ed estensioni future: basterà infatti aggiungere la logica desiderata lato server, mentre, nel client side dell'applicazione basterà inserire un'ulteriore colonna con gli appropriati *Widget* (Button, Checkbutton, Entry, Frame, Label, LabelFrame, Menubutton, PanedWindow, Radiobutton, Scale, Scrollbar, etc) sulle base delle proprie necessità.

Lo stesso software è stato reso *portable* (può essere eseguito in diverse piattaforme con differenti sistemi operativi) grazie al gestore di pacchetti di python *pip3*, e in particolare grazie al comando:

```
pyinstaller <nome programma>.py
```

[PYT].

Conclusioni e lavori futuri

La seguente esperienza di tesi mi ha dato modo di poter sperimentare per la prima volta un altro modello di programmazione, diverso dalla programmazione imperativa, funzionale o guidata da eventi, ossia la programmazione dei microcontrollori hardware di un sistema embedded. Ho sentito di aver potuto applicare tutte le nozioni apprese nei tre anni universitari, di ogni disciplina, nessuna esclusa.

Un potenziale sviluppo futuro della applicazione corrente è quello della realizzazione di ulteriori moduli e feature avanzate usando altri sensori e la pi camera. Ad esempio: per sperimentare la *computer vision* con l'utilizzo della potente libreria OpenCV, per *riconoscere e tracciare oggetti* di una forma o colore specifici; per implementare la caratteristica del *line tracking*, che si basa sulla riflessione ad infrarossi e che consente all'auto di muoversi lungo il percorso che si è impostato.

Si poteva inoltre sfruttare la fotocamera del Raspberry Pi per trasferire immagini e video in tempo reale, su un computer remoto, e visualizzando tali media lato client attraverso un carousel offerto dal Widget *ImageTk* della libreria tkinter.

Installando un microfono USB che può essere collegato al Raspberry Pi e la libreria *SpeechRecognition*, è peraltro possibile implementare un *voice control system* per eseguire azioni e controllare la macchina.

Le possibilità sono infinite, e questo è dovuto alla vasta eterogeneità di tecnologie e oggetti come sensori, attuatori e librerie software, sviluppati appositamente per il controllo di moltissimi device che cooperano per costituire il mondo dell'Internet of things.

Contento di quanto questa esperienza di Tesi mi abbia lasciato, desidero ringraziare il lettore.

Riferimenti Bibliografici

- [ADP] Adeept, “*Mars Rover PiCar-B Smart Robot Car Kit for RPi*”, 2021. Reperibile presso: <https://www.adeept.com/learn/tutorial-316.html>.
- [ASH01] K. Maney, “*Meet Kevin Ashton, Father of the Internet of Things*”, 2015. Reperibile presso: <https://www.newsweek.com/2015/03/06/meet-kevin-ashton-father-internet-things-308763.html>
- [BOR] nonowerk.com, “*10 Most Unusual Smart Devices - That Still Can Be Useful*” Reperibile presso: <https://www.nanowerk.com/smart/10-most-unusual-smart-devices-that-still-can-be-useful.php>
- [CAM] C.A. Mazzone, “*C e C++ Le chiavi della programmazione*”, 2016.
- [FLK] Flask, “*User’s Guide*”, 2022. Reperibile presso: <https://flask.palletsprojects.com/en/1.0.x/>
- [IEE] IEEE, “*IoT Towards Definition Internet of Things*”, 2015. Reperibile presso: https://iot.ieee.org/images/files/pdf/IEEE_IoT_Towards_Definition_Internet_of_Things_Revision1_2_7MAY15.pdf
- [IOT01] ENGINEERING TUTORIAL, “*Internet of Things | A Basic Introduction | IoT*”, 2020. Reperibile presso: https://www.youtube.com/watch?v=O6qDwLmC_jU&list=PLVsrfTSIZ_40kl6HvTnR9xF3NVREX2afP
- [IOT02] Alexander S. Gillis, “*What is the internet of things (IoT)?*”, 2022. Reperibile presso: <https://www.techtarget.com/iotagenda/definition/Internet-of-Things-IoT>
- [IOT03] R. Khanna, “*Internet of Things: Architectures, Protocols, and Applications*” 2017. Reperibile presso: <https://www.hindawi.com/journals/jece/2017/9324035/>
- [IOT04] Ian J.H. Reynolds, “*IOT Architecture*”, 2020. Reperibile presso: <https://www.zibtek.com/blog/iot-architecture/>
- [NTS] Twenty-First Century Books, “*WHEN WOMAN IS BOSS*”. Reperibile presso: <http://www.tfcbooks.com/tesla/1926-01-30.htm>
- [LIB01] adeept, “*Adeept 4WD Smart Car Kit for Raspberry Pi PiCar-B*”. Reperibile presso: https://github.com/adeept/adeept_picar-b
- [LIB02] C. Nelson, T. DiCola, “*Adafruit Python PCA9685*”, 2016. Reperibile presso: https://github.com/adafruit/Adafruit_Python_PCA9685

[LIB03] B. Croston, “*RPi.GPIO 0.7.1*”, 2022. Reperibile presso: <https://pypi.org/project/RPi.GPIO/>

[LIB04] J. Garff, “*rpi_ws281x*”, 2014. Reperibile presso: https://github.com/jgarff/rpi_ws281x

[MRP01]

[KRS] James F. Kurose, Keith W. Ross, “*Reti di calcolatori e internet, Un approccio top-down, Settima edizione, Edizione italiana a cura di: Antonio Capone e Sabrino Gaito*”, 2017.

[UFE] C. Giannelli, M. Tortonesi, “*Industrial Internet of Things*”, 2021.

[UUD] S. Ivan, “*IOT - PANORAMICA INTRODUTTIVA ALL'INTERNET OF THINGS*”, 2019. Reperibile presso:

<https://users.dimil.uniud.it/~ivan.scagnetto/IoT.pdf>,

[PIN] P. Howard, “*Raspberry Pi Pinout*”, 2022. Reperibile presso: <https://pinout.xyz>

[PIV] G. Piva, “*Tecnologie Web*”, 2020.

[PYT] A. Lorenzi, E. Cavalli, V. Moriggia, “*Linguaggio Python*”, 2019.

[PAO] J. C. Shovic, A. Simpson “*Python All-In-One For Dummies 2st Edition*”, 2019.

[PSV01] E. Newton, “*What IoT sensors are onboard NASA's Perseverance rover?*”, 2021 Reperibile presso: <https://techfruit.com/2021/07/16/what-iot-sensors-are-onboard-nasas-perseverance-rover/>

[PWM01] Wikipedia, “*Duty cycle*”

Reperibile presso: https://it.wikipedia.org/wiki/Duty_cycle

[PWM02] Wikipedia, “*Modulazione di larghezza d'impulso*” Reperibile presso: https://it.wikipedia.org/wiki/Modulazione_di_larghezza_d%27impulso

[REQ] Requests, “*Requests: HTTP for Humans*”, 2022. Reperibile presso: <https://requests.readthedocs.io/en/latest/>

[RHA] Redhat, “*Cos'è un'API REST?*” 2020.

Reperibile presso: <https://www.redhat.com/it/topics/api/what-is-a-rest-api>

[RSP] Raspberry Pi, “*Raspberry Pi 3 Model B*”, 2022.

Reperibile presso: <https://www.raspberrypi.com/products/raspberry-pi-3-model-b/>

[SLD] S. Monteleone, “*IOT: SCENARI E SOLUZIONI*”, 2015. Reperibile presso: http://utenti.dieei.unict.it/users/smonteleone/corsi/aa2015-16/la/slides/01_IoT.pdf

[SOP] E. Modafferi, “*REST vs SOAP, “come funzionano i Web Services?”*”

Reperibile presso: <https://dinotraining.it/rest-soap-vantaggi-e-svantaggi/>

[IAN] I. Sommerville, “*Ingegneria del software, Decima Edizione*”, 2017.

[MPI] M. Picone, “*Seminario su Internet of Things (Prof. Picone)*”, 2021.

Reperibile presso: <https://www.youtube.com/watch?v=dFuQFlnfNbM>

[TKI] tkinter, “*tkinter — Python interface to Tcl/Tk*”, 2022. Reperibile presso: <https://docs.python.org/3/library/tkinter.html>

[TRC01] Treccani, “*dispositivo*”.

Reperibile presso: <https://www.treccani.it/enciclopedia/dispositivo>

[TRC02] Treccani, “*sensore*”.

Reperibile presso: <https://www.treccani.it/enciclopedia/sensore>

[TRC03] Treccani, “*attuatore*”, 2008. Reperibile presso: https://www.treccani.it/enciclopedia/attuatore_%28Enciclopedia-della-Scienza-e-della-Tecnica%29/

[TRC04] Treccani, “*cloud computing*”, 2012. Reperibile presso: https://www.treccani.it/enciclopedia/cloud-computing_%28Lessico-del-XXI-Secolo%29/

[ULT] G. Monti, “*TUTORIAL - SENSORE AD ULTRASUONI HC-SR04*”. Reperibile presso: <https://www.weturtle.org/dettaglio-tutorial/11/tutorial-sensore-ad-ultrasuoni-hcsr04.html>

[WIK01] Wikipedia, “*Internet delle cose*”, 2022.

Reperibile presso: https://it.wikipedia.org/wiki/Internet_delle_cose

[WIK02] Wikipedia, “*Arpanet logical map, march 1977.png*”. Reperibile presso:

https://en.wikipedia.org/wiki/File:Arpanet_logical_map,_march_1977.png

Riferimenti Figure

Figura 1.3a. <https://mpython.readthedocs.io/en/master/tutorials/advance/iot/index.html>

Figura 1.3b. <https://www.spiceworks.com/tech/cloud/articles/edge-vs-fog-computing/>

Figura 1.4b.

https://media.monclick.it/CMS/it/20181220/cosaeunasmartcity_793a953c-46ff-4cb7-8421-e66cafe40f1f.jpg

Figura 1.4e. https://mars.nasa.gov/files/mars2020/Mars2020_Fact_Sheet.pdf