POLITECNICO DI TORINO

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea Magistrale

Gioco robotico e laboratori didattici per bambini in età scolare

Relatore

prof. Fabrizio Lamberti

Candidato

Alessio Giuseppe Calì

Sessione di laurea di Dicembre 2018

Ai miei genitori, parenti,

amici ed educatori tutti.

Non sarei qui senza ognuno

di voi. Grazie.

Ringraziamenti

Questo lavoro è stato parzialmente supportato dall'iniziativa *VR@POLITO* del Politecnico di Torino e da *Xké? Il laboratorio della Curiosità*.

Desidero ringraziare il prof. Fabrizio Lamberti per la sua supervisione, che ha reso possibile questa collaborazione; Caterina Ginzburg, Lara De Bortoli, Giuseppe Scicchitano e Mauro Panfalone di Xké per il loro continuo supporto, e per averci aiutato superare molti ostacoli organizzativi; il prof. Claudio Germak e la tesista Lorenza Abbate, insieme ai quali questa tesi è stata realizzata.

Sommario

Lo scopo di questa tesi è indagare i benefici sull'apprendimento derivanti dall'applicazione di tecnologie robotiche all'interno di un contesto educativo non scolastico, informale. Essa nasce come collaborazione tra il Politecnico di Torino e *Xké? Il laboratorio della Curiosità*, un centro per la didattica delle scienze rivolto alle scuole elementari e medie, al fine di promuovere il sapere scientifico. Per realizzare gli obiettivi preposti, verrà prima effettuata un'analisi delle attività già svolte in precedenza su tale argomento, e sui prodotti commerciali disponibili per l'intrattenimento robotico. Successivamente verranno introdotte le attività svolte da parte del Laboratorio, e illustrate alcune proposte progettuali per il potenziamento di uno dei percorsi didattici attualmente in essere. Verrà quindi analizzata nel dettaglio la fase di realizzazione della proposta scelta, un'attività che coinvolge un robot personalizzato basato su Raspberry Pi, un sistema di proiezione ed un'applicazione di controllo per insegnare ai bambini le caratteristiche dei pianeti del Sistema solare. Terminata la descrizione delle tecnologie adoperate e del protocollo di comunicazione adottato per mettere in comunicazione tra loro questi elementi, verrà presentato un questionario mirato a valutare la facilità d'uso del sistema realizzato, nonché la reazione degli studenti alla nuova attività come percepita dai tutor del Laboratorio, unitamente ad alcune osservazioni personali basate sull'osservazione delle classi coinvolte. In appendice a questa tesi, sarà presente un manuale d'uso e manutenzione, pensato per poter riparare o ricostruire il robot e ripristinarne le funzionalità.

Indice

Capitolo 1

Introduzione

Quella del robot è una figura cara alla fantascienza: il solo termine, infatti, suscita spesso immagini di automi umanoidi che sembrano usciti dai romanzi di Isaac Asimov, in grado di compiere azioni indipendenti senza un controllo esterno, e di dialogare tramite il linguaggio umano. Nella realtà, le cose non stanno esattamente così. Lo sviluppo tecnologico degli ultimi anni ha portato ad allargare in maniera considerevole il numero di dispositivi genericamente catalogati come "robot": basti pensare alle enormi differenze tra un aspirapolvere automatico, un robot da cucina o un drone volante.

Tale esplosione di prodotti che giungono sul mercato ha non solo attratto un pubblico di nuovi acquirenti, posti per la prima volta di fronte a dispositivi che fino a un decennio fa sarebbero potuti benissimo apparire tra le pagine di riviste avveniristiche, ma ha anche suscitato un crescente interesse nel mondo della ricerca, finora vincolato da complessi apparati sperimentali. L'utilizzo di componenti commerciali, già rigorosamente sperimentati, lascia spazio allo sviluppo di attività derivate: numerosi sono gli studi sull'interazione uomo-macchina basate su robot di questo tipo, o sull'implementazione di tecniche di navigazione (ad esempio per guidare in maniera automatica droni che richiedono un controllo manuale). Non solo, un effetto secondario di questo nuovo mercato è stato lo sviluppo della componentistica fai-da-te: prodotti come Arduino o Raspberry Pi hanno messo nelle mani degli hobbisti gli strumenti per realizzare ogni sorta di dispositivo intelligente ad un costo contenuto. Un'interessante contaminazione avvenuta nel mercato di consumo, tra il mondo della robotica e quello delle costruzioni giocattolo, è rappresentata da LEGO Mindstorm, un kit di sviluppo robotico della nota produttrice di mattoncini (Figura 1.1). Prodotti

come questo sono sintomatici del crescente interesse verso il settore della robotica da parte di un pubblico non specializzato.

Figura 1.1: LEGO Mindstorm. Immagine tratta da https://www.lego.com/

In questo contesto, un campo applicativo emerso è quello dell'uso di robot per fini ludici: in un *robotic game*, o gioco robotico, uno o più giocatori umani interagiscono con uno o più automi, che possono ricoprire il ruolo di compagno, avversario o strumento per interagire con sistemi più ampi (Figura 1.2). Gli scopi possono essere molteplici: dal puro e semplice intrattenimento, all'insegnamento scolastico, all'allenamento fisico, ecc.. Non sono pochi gli esempi di veri e propri giocattoli robotici disponibili in commercio, alcuni totalmente autonomi, altri meno.

Quando il gioco robotico viene utilizzato come mezzo propedeutico all'insegnamento si parla di robotica educativa. Il termine, potenzialmente fuorviante, non si riferisce all'insegnamento della robotica o alla programmazione di un robot, entrambi campi molto complessi e settoriali, ma ad argomenti solitamente affini alle cosiddette STEM (Science, Technology, Engineering, Mathematics), attraverso l'utilizzo del robot stesso. Tali attività sono tipicamente rivolte a bambini in età scolare, i quali vengono introdotti a una specifica piattaforma robotica e, attraverso il suo montaggio e la sua personalizzazione, apprendono i fondamenti del pensiero logico: imparando a "spiegare" al robot come svolgere uno specifico compito all'interno di un contesto più ampio essi vengono stimolati ad applicare nuove conoscenze, tipicamente logico-matematiche, in maniera creativa.

Un'attività di robotica educativa, a sua volta, può essere declinata in termini di *robotic edutainment*. Come suggerisce quest'ultima parola, contrazione dei termini inglesi *education* ed *entertainment* ("educazione" e "divertimento"), si tratta di un ponte tra le attività educative e quelle ricreative. L'apprendimento viene posto come un gioco in cui i bambini sono motivati ad interagire col robot per arrivare ad un qualche obiettivo. Tale interazione risulta ancora più efficace nei

Figura 1.2: Bambini alle prese coi robot Dash e Dot. Immagine tratta da https://www.robotiko.it/

confronti di un'utenza infantile, la quale tende a empatizzare maggiormente col robot, recependolo spesso come un'entità intelligente anche al di fuori dei reali intenti previsti dal progettista; in sostanza, la spiccata fantasia di cui sono dotati i bambini permette loro di fruire in maniera unica del mezzo robotico.

Attraverso il gioco robotico i bambini approcciano l'apprendimento in maniera nuova rispetto ai canoni classici dell'apprendimento scolastico. Piuttosto che assorbire nozioni tramite studio ed esercizio, possono farlo cercando di comprendere, e interagendo con un oggetto che stimola la loro curiosità e creatività. Indagare le potenzialità di questo mezzo e sviluppare un prodotto educativo per l'uso continuo nel tempo sarà l'obiettivo di questa tesi. Attività didattiche legate al *robotic gaming* sono tipicamente svolte in un ambiente educativo formale, ossia quello scolastico, in cui i tutor che guidano gli studenti sono gli stessi insegnanti con cui si relazionano ogni giorno. Al contrario, la novità introdotta in questo lavoro sarà quella di inserirsi in un contesto informale, in un laboratorio didattico gestito privatamente, alle cui attività partecipano classi di studenti provenienti da scuole primarie e secondarie di primo grado accompagnati da insegnanti.

Il lavoro nasce come collaborazione tra il Politecnico di Torino e *Xké? Il laboratorio della Curiosità*, un centro per la didattica delle scienze che si occupa di organizzare attività educative per promuovere l'apprendimento della conoscenza scientifica tra i bambini delle scuole elementari e medie (Figura 1.3). Tramite una collaborazione del Dipartimento di Automatica e Informatica e del Dipartimento di Architettura e Design, nell'ambito di un progetto di tesi complementare a questa, supervisionato dal prof. Claudio Germak e svolto dalla tesista Lorenza Abbate, è stata progettata e realizzata un'esperienza incentrata sullo studio del Sistema solare, avente un robot come compagno per l'apprendimento. Nel resto del documento, in una prima introduzione

all'argomento verranno presentati alcuni studi esistenti sul *robotic gaming*, accompagnati da alcuni esempi di robot esistenti in commercio appartenenti a varie categorie. Successivamente verranno presentati il Laboratorio e le sue attività, e verranno spiegate le idee di base del progetto ed i suoi requisiti. Si passerà quindi a una dettagliata analisi del sistema realizzato, il quale comprende un robot personalizzato, un'applicazione multimediale in esecuzione su di un server centrale e un'applicazione mobile di controllo, che verranno descritte in tutte le loro parti, con particolare attenzione ai meccanismi di comunicazione. Per concludere, sarà presentato un questionario proposto ai tutor che hanno supervisionato le classi che hanno partecipato ad una prima fase sperimentale di prova della nuova esperienza ed esposte alcune considerazioni personali basate sull'osservazione degli studenti coinvolti. In appendice, sarà presente un manuale di supporto, pensato per la manutenzione ed il ripristino del robot, nel quale verranno spiegati i passaggi per la sua costruzione e le procedure da seguire per poter valutare e monitorare le sue funzioni.

(a) (b)

Figura 1.3: Foto della stanza dove è stata svolta l'attività di questa tesi.

Capitolo 2

Stato dell'arte

Il crescente interesse nei confronti del mezzo robotico ha portato negli ultimi anni non solo a numerose indagini accademiche sui suoi campi d'applicazione, ma anche alla nascita di prodotti d'intrattenimento per il consumatore, esponendo un potenziale mercato che è ancora da esplorare a fondo. Il presente capitolo si pone l'obiettivo di presentare una panoramica di queste applicazioni, partendo dagli ambiti di ricerca e dai risultati già ottenuti per valutarne l'efficacia rispetto ad alternative tradizionali; verranno quindi mostrati alcuni dei robot commerciali più interessanti in campo educativo. Tali robot verranno presi ad esempio nel capitolo successivo per esporre le proposte di progetto, e verrà successivamente motivata la scelta di realizzare un robot personalizzato.

2.1 Lavori precedenti

In questa Sezione verranno brevemente presentati due lavori nel campo del *robotic gaming*, dalle caratteristiche diverse ma egualmente importanti dal punto di vista dei risultati.

Nel primo di questi due, *Physically Interactive Robogames* [1], vengono definite una serie di linee guida generali per la realizzazione di giochi robotici, e messe in pratica per realizzare un'esperienza interattiva dal titolo *Jedi Trainer* che coinvolge un drone volante ed un giocatore umano.

Nel secondo lavoro, *Cellulo: Versatile Handheld Robots for Education* [2], viene esplorato il campo della robotica applicato alla didattica educativa, attraverso la realizzazione di un piccolo

robot maneggevole da impiegare come controller interattivo su di plance di gioco opportunamente preparate.

2.1.1 PIRG e *Jedi Trainer*

Nel 2013, Martinoia, Calandriello & Bonarini [1] del Politecnico di Milano hanno presentato uno studio dal titolo *Physically Interactive Robogames: Definition and design guidelines*, nel quale vengono proposte delle linee guida per la definizione di giochi robotici. Tale studio, svolto nel 2011, parte dalla seguente definizione formale (in lingua inglese):

> *"A Physically Interactive RoboGame consists of a number of autonomous agents (including software, hardware, and physical agents) of which at least one is an autonomous robot, and one is a person. These agents interact with each other in a possibly variable and unknown environment, by following some game rules, so that the human players can have fun."*

Lo studio continua illustrando alcune linee guida sulla realizzazione di un PIRG, che mirano a mantenere qualsiasi implementazione semplice, affidabile, efficace e realizzabile. Seguendo questi principi gli autori hanno realizzato ed esposto un esempio di PIRG ispirato alle vicende della famosa saga fantascientifica *Star Wars*, dal titolo *Jedi Trainer*. In tale gioco, al giocatore viene fatta indossare una veste monocromatica e fornito un lungo tubo (che simula una spada) di colore acceso ed in contrasto con la veste. Un drone, equipaggiato con una telecamera, ha il ruolo di sfidante, ed il suo obiettivo è "colpire" il giocatore (simulando uno sparo tramite una forte vibrazione), spostandosi e mirando al petto. Il giocatore ha il compito di proteggersi usando la spada come uno scudo per deviare i colpi, proteggendo le parti esposte del proprio corpo. Ogni colpo correttamente parato assegna un punto al giocatore.

Il gioco realizzato è stato successivamente esposto al Robotica2011 [1], dove è stato fatto provare ad alcuni partecipanti ai quali è stato chiesto di compilare un questionario post-partita. Dalle risposte è emerso un alto livello di coinvolgimento, prova di un interesse concreto da parte di un campione di potenziali acquirenti, e un generale senso di progressione e apprendimento dato dalla natura competitiva del gioco.

2.1.2 Cellulo

Presso il laboratorio CHILI dell'École Polytechnique Fédérale de Lausanne, in Svizzera, Ozgur et al. [2] hanno sviluppato un'interessante piattaforma che prende il nome di Cellulo (Figura 2.1). Una delle caratteristiche di Cellulo è quella di non apparire, a primo sguardo, come un robot. In verità ha l'aspetto di uno stampo esagonale, e si presta alla manipolazione diretta da parte dell'utente. L'obiettivo di Cellulo è infatti, stando alle intenzioni degli autori, essere visto più come uno strumento da maneggiare che come un giocattolo da osservare, al pari di una penna o una gomma (molto tecnologiche).

Figura 2.1: Illustrazione della piattaforma Cellulo. Immagine tratta da https://chili.epfl.ch/

A un'analisi più dettagliata tuttavia Cellulo presenta un concentrato di tecnologia in soli 75 mm. Dotato di un modulo Bluetooth per la comunicazione con altri dispositivi, può allo stesso tempo essere trascinato e muoversi liberamente grazie a tre motori omnidirezionali; la superficie superiore è sensibile al tocco in sei punti predefiniti, che forniscono un feedback aptico quando premuti.

Una delle caratteristiche più interessanti di Cellulo tuttavia è il modo in cui è in grado di localizzarsi. Dotato di una microcamera e un illuminatore a infrarossi posti al suo interno può osservare la superficie sulla quale si trova. Sfruttando la tecnologia sviluppata da Hostettler et al. [3] per il riconoscimento di uno speciale pattern di punti micrometrici, Cellulo è in grado di riconoscere la propria posizione e il proprio orientamento all'interno di una superficie stampata sul quale tale pattern è stato applicato, con accuratezze inferiori agli 0.3 mm e 2°, rispettivamente. Essendo il pattern in questione veramente minuto (con punti di diametro nell'ordine del decimo di

millimetro) è possibile applicarlo su qualsiasi elemento grafico senza andarne a inficiare la qualità in maniera percepibile.

Su queste basi gli autori hanno costruito due esperienze volte a verificare le potenzialità della piattaforma. Nella prima esperienza, denominata *Treasure Hunt* (vedi [2]) sono state approfondite le caratteristiche di Cellulo come strumento passivo. Gruppi da cinque a sei bambini sono stati dotati di un robot a testa, ed è stato affidato loro il compito di eseguire alcune "missioni" su una plancia a tema *Isola del tesoro*. Lo studio ha evidenziato come la piattaforma risulti intuitiva e di rapido apprendimento, e come da un'esperienza ben progettata possano emergere spontaneamente elementi di collaborazione tra i partecipanti.

Nella seconda esperienza, *Windfield* ([2] e successivamente [4]), gli autori hanno studiato l'efficacia di Cellulo come mezzo di insegnamento scolastico in un campo tipicamente poco affrontato, quello della fisica delle correnti d'aria. Ai bambini sono state proposte due sfide: la prima consisteva nell'individuare i punti di alta e bassa pressione sulla plancia di gioco, usando Cellulo come un pallone aerostatico simulato (il robot era stato programmato per seguire la corrente teoricamente generata dai punti di pressione); la seconda nel far piazzare a loro dei punti di pressione (tramite un'interfaccia su tablet) in modo da far seguire ad un singolo Cellulo un determinato percorso. I risultati di questo studio hanno evidenziato la curiosità dei bambini verso gli strumenti robotici e la generale efficacia nel trasmettere le nozioni richieste almeno ad alto livello, anche se alcune debolezze sono state rilevate nell'assimilazione di concetti più complessi.

Il caso Cellulo è emblematico dell'enorme potenziale che una strumentazione robotica, opportunamente progettata e corredata da attività didattiche adeguate, può fornire all'educazione anche al di fuori delle materie STEM.

2.2 Prodotti commerciali

In questa Sezione verranno presentati brevemente alcuni robot disponibili in commercio per il mercato generalista, riassumendone le caratteristiche salienti e le limitazioni di ognuno. Verranno inoltre riassunti, laddove presenti, alcuni dei lavori in ambito accademico che li hanno visti coinvolti.

2.2.1 Sphero

Come si evince dal nome, Sphero (Figura 2.2) è un piccolo robot sferico di circa 75 mm di diametro, dotato di modulo Bluetooth, un'unità di misurazione inerziale, ed encoder sui motori per il controllo della velocità. Sphero è in grado di rotolare e di spostarsi a una velocità fino a 2 m/s ed è controllato tramite applicazioni mobile compatibili.

Figura 2.2: Il robot Sphero. Immagine tratta da https://store.sphero.com/

Per gli sviluppatori sono presenti strumenti per realizzare applicazioni per iOS o Android, interfacciarsi con l'ambiente di sviluppo Unity e per pre-programmare Sphero con una serie di comportamenti predefiniti. Nonostante l'ampio supporto è tuttavia limitato dai meccanismi di localizzazione (puramente dead-reckon) e dalle interazioni possibili con l'utenza (può solamente muoversi). Nonostante ciò, grazie proprio alla sua essenzialità, Sphero è stato oggetto di un certo numero di studi sull'interazione uomo-macchina.

Ad esempio, Faria, Costigliola, Alves-Oliveira & Piava [5] hanno condotto un esperimento sociale volto a determinare la capacità di percepire intelligenza e recepire intenti nei confronti di un robot privo di caratteristiche comunicative, osservando il comportamento di un campione di soggetti di fronte a uno Sphero guidato da un operatore nascosto. Lo scopo dell'esperimento era attirare la loro attenzione e guidarli verso un obiettivo senza che fossero al corrente della situazione. Da un questionario post-esperimento, è emerso come nonostante l'assenza di tratti somatici, il robot sia stato percepito come intelligente, e sia stato in grado di comunicare correttamente le proprie intenzioni. Una possibile obiezione tuttavia potrebbe essere quella di imputare alla manipolazione dell'operatore l'intelligenza percepita; a tal proposito potrebbe essere interessante ripetere un esperimento simile facendo a meno di quest'ultimo, allestendo un opportuno sistema di tracking e guida per Sphero.

2.2.2 Ozobot Evo

Evo (Figura 2.3) è un piccolo robot della categoria *line-follower*, la cui peculiarità risiede nel modo in cui viene programmato. Alla sua base si trova una microcamera in grado di rilevare 8 colori distinti (la terna RGB, la terna CMY, bianco e nero). Attraverso di essa il robot può operare in due modalità distinte:

- *color codes*, il robot esegue operazioni codificate su una traccia da seguire;
- *OzoBlocky*, il robot apprende ed esegue un programma precedentemente caricato.

Figura 2.3: Il robot Evo di Ozobot. Immagine tratta da https://ozobot.com/

La prima modalità permette all'utente di "disegnare" i propri programmi; Evo infatti segue un arbitrario tracciato nero su sfondo bianco, ma rileva ed interpreta sequenze di colori come istruzioni da eseguire. Ad esempio, la sequenza blu-rosso-blu fa voltare e tornare indietro Evo, ma sono possibili anche istruzioni più complesse come la gestione di timer o contatori. Tale modalità è pensata per introdurre i bambini ai concetti della programmazione in maniera interattiva e intuitiva, senza ricorrere ad alcun ambiente di sviluppo.

La seconda modalità permette di eseguire su Evo un piccolo programma sviluppato tramite un ambiente di programmazione visivo chiamato *OzoBlocky*: pensato per essere appreso a livelli progressivi di difficoltà, arriva a introdurre concetti avanzati quali cicli iterativi e funzioni, con addirittura la possibilità, nella categoria più avanzata, di lavorare direttamente in Javascript.

OzoBlocky espone tutte le capacità di Evo, usando i colori rilevati sul percorso come input e lo speaker e i LED integrati come output per produrre effetti anche molto complessi. Come curiosità tecnica, mancando Evo di dispositivi di comunicazione, i programmi scritti con OzoBlocky vengono caricati in maniera "visiva": l'ambiente chiede all'utente di posizionare Evo sullo schermo, dove viene visualizzata una sequenza di colori in rapida successione che vengono interpretati come i byte del programma stesso.

Nonostante l'indubbia ingegnosità di Evo, le sue caratteristiche lo rendono un prodotto che pecca di flessibilità, in quanto vincolato al suo peculiare sistema di riconoscimento delle istruzioni.

2.2.3 Anki Cozmo

Cozmo (Figura 2.4) si differenzia dalla maggior parte dei robot commerciali per la sua incredibile espressività e apparente intelligenza, avvicinandosi molto all'immagine popolare di un robot senziente. Si tratta di un oggettino dalle fattezze di un muletto in miniatura, capace di muoversi in completa autonomia e dotato di una "faccia" costituita da un piccolo schermo LCD. Cozmo può esplorare l'ambiente circostante, riconosce ostacoli ed evita le cadute. Esso si esprime attraverso il suo schermo ed emettendo suoni dagli speaker, dando l'illusione di essere vivo, ed è in grado di riconoscere e interagire con cani e gatti. A livello di giocabilità il robot è accompagnato da alcuni cubetti che è in grado di distinguere, toccare e sollevare per realizzare piccoli giochi basati sui riflessi.

Figura 2.4: Il robot Cozmo di Anki. Immagine tratta da https://www.anki.com/

Trattandosi di un prodotto relativamente recente (rilasciato nell'autunno del 2017) mancano ancora studi sull'utilizzabilità di Cozmo nell'ambito del *robotic gaming*, ma la presenza di un SDK molto avanzato basato su Python promette un grande potenziale per studi futuri.

2.2.4 Jumping Sumo

Appartenente alla categoria dei droni, Jumping Sumo (Figura 2.5) è un robot di terra a due ruote dotato di connettività WiFi, una telecamera grandangolare e un'unità di misurazione inerziale, in grado di spostarsi fino a 2 m/s e di compiere salti fino ad 80 cm.

Figura 2.5: Il drone Jumping Sumo di Parrot. Immagine tratta da https://www.parrot.com/

Rispetto a soluzioni simili (come Sphero, con il quale condivide la mobilità come sua funzione primaria) la presenza della telecamera rappresenta un grosso passo in avanti per le possibili applicazioni. Ad esempio, uno studio condotto da Piumatti, Prattico, Paravati & Lamberti [6]

ha portato alla realizzazione di un sistema di guida autonoma per il Jumping Sumo mediante il riconoscimento di punti di riferimento visivi. La particolare conformazione del robot inoltre si presta all'estensione mediante periferiche esogene, cosa non consentita da Sphero (dal momento che il suo corpo contribuisce per intero allo spostamento).

Capitolo 3

Progettazione

Il lavoro esposto in questa tesi mira a valutare i potenziali effetti positivi che l'introduzione di elementi robotici all'interno di un contesto educativo può avere sull'assimilazione di nozioni di natura scolastica da parte di un gruppo di studenti. Per rendere possibile tale obiettivo, è stato dato seguito ad una collaborazione già stabilita in anni recenti tra il Politecnico di Torino e *Xké? Il laboratorio della Curiosità*, un centro didattico che propone laboratori interattivi a carattere scientifico con l'intento di stimolare la curiosità e potenziare il processo di apprendimento negli studenti delle scuole primarie e secondarie di primo grado.

L'oggetto di tale collaborazione è il potenziamento e la riprogettazione di una delle esperienze proposte con l'inserimento di un robot nel contesto di quest'ultima. In questo capitolo verranno presentate le attività già svolte dal Laboratorio, con particolare attenzione a quella scelta per il potenziamento. Verranno quindi analizzati gli elementi più interessanti da sviluppare, e illustrato il processo progettuale che ha portato alla definizione di alcune proposte implementative. Infine, verranno analizzate nel dettaglio la proposta accettata e le motivazioni dietro tale scelta. I dettagli riguardanti l'implementazione saranno invece oggetto del capitolo successivo.

3.1 Riguardo il Laboratorio

Xké? Il laboratorio della Curiosità è un centro nato nel 2011 come progetto della Fondazione per la Scuola della Compagnia di San Paolo. Esso ha come obiettivo la divulgazione del sapere scientifico verso studenti di età compresa tra i 6 ed i 13 anni. I laboratori promossi da Xké mettono al centro l'interazione tra i bambini e il mondo, proponendo esperienze *hands-on* che mirano a

suscitare stupore e stimolare la curiosità degli studenti. Un ruolo chiave è quello svolto dai tutor, ragazzi provenienti da corsi universitari prevalentemente scientifici e che fungono da mediatori nei confronti delle classi accompagnate. Nel corso degli anni Xké ha diversificato e suddiviso la propria offerta didattica in base alle fasce d'età (scuola primaria classi prima e seconda, scuola primaria classi terza quarta e quinta, e scuola secondaria di primo grado) e al momento prevede ben ventisette percorsi, che spaziano dalla scienza astronomica, allo studio delle fonti energetiche, alla robotica e molto altro. Oltre a offrire queste attività alle scuole, Xké collabora con associazioni e cooperative che operano in contesti di disagio e fragilità educativa, accogliendo a titolo gratuito gruppi provenienti da realtà difficili.

3.1.1 1, 2, 3... Stelle!

In questo percorso didattico gli studenti vengono avvicinati al sistema Sistema solare: il Sole, i pianeti che vi orbitano attorno, la fascia di asteroidi e gli altri corpi celesti.

In una prima fase, la scolaresca viene fatta accomodare da un tutor in una camera oscurata, dove gli studenti possono sdraiarsi per terra su dei cuscini. Sul soffitto viene proiettato un video introduttivo che spiega sotto forma di corto animato quali sono le caratteristiche salienti del Sistema solare. Gli studenti devono prestare la massima attenzione in quanto tali nozioni saranno necessarie nella fase successiva.

Terminata la visione del video, gli studenti vengono portati nella stanza adiacente e il tutor spiega loro l'attività che dovranno svolgere; si tratta di un semplice gioco di associazioni, durante il quale gli studenti devono ordinare delle fotografie dei pianeti del Sistema solare e abbinare ad ognuno delle etichette recanti delle caratteristiche e/o curiosità. Ad esempio al pianeta Marte è possibile associare la curiosità "Mi chiamo come il dio romano della guerra". Il tutto si riassume nei seguenti, semplici passi:

1. la classe viene divisa in due gruppi, di egual dimensione;

2. ad entrambi i gruppi, in maniera indipendente e separata, viene fornito un set di fotografie e di etichette; i set forniti ad entrambi i gruppi sono assolutamente identici;

3. gli studenti procedono con l'abbinare ogni curiosità al rispettivo pianeta, sotto la supervisione dei tutor, fino alla completa ricostruzione degli abbinamenti corretti.

3.2 Analisi dell'esperienza

Su suggerimento del Laboratorio ci si è concentrati sullo sviluppo di soluzioni per il potenziamento del percorso *1, 2, 3... Stelle!*. Osservando l'esperienza attuale, risulta evidente come il coinvolgimento degli studenti sia affidato esclusivamente ai tutor e ci si aspetta che nasca come spontanea conseguenza della loro cooperazione al fine di completare le associazioni. Inoltre si tratta di un'esperienza molto semplice, che potrebbe essere resa più accattivante dall'inserimento di un robot, il quale stimolerebbe l'interesse degli studenti.

Dagli incontri tenuti con i coordinatori del Laboratorio è emerso l'interesse a voler realizzare un nuovo percorso in continuità con quello attuale, in modo tale che la soluzione sviluppata non rappresenti una rottura col passato ma una sua naturale evoluzione. A seguito di una prima fase di brainstorming guidata dal Dipartimento di Design sono state avanzate due proposte progettuali, una su piccola scala basata su prodotti commerciali, un'altra su larga scala basata sulla realizzazione di un robot personalizzato.

3.2.1 Proposta in piccola scala con Ozobot

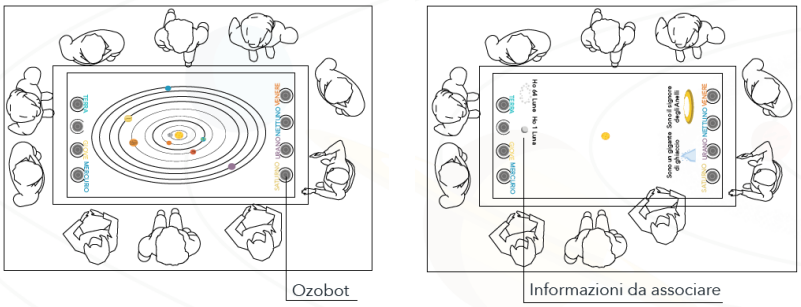
La prima proposta prevede l'utilizzo di un tavolo interattivo multimediale sul quale viene mostrata una videata del Sistema solare. Ai due lati minori vengono posizionati otto Ozobot, ognuno rappresentante un pianeta (Figura 3.1).

Durante una prima fase dell'attività, sullo schermo vengono mostrati degli "adesivi", uno per ogni caratteristica dei pianeti, in ordine sparso. Gli studenti devono trascinare ognuno di essi verso il pianeta al quale ritengono appartenga, "caricandolo" con le informazioni necessarie. Allo stesso tempo, una parte degli studenti deve disegnare con mano su schermo le orbite dei pianeti stessi, guidati da una traccia visibile.

Terminata sia l'associazione delle curiosità sia il tracciamento delle orbite, gli studenti possono posizionare gli Ozobot sulle orbite corrispondenti. Un programma precaricato sugli Ozobot permette loro di riconoscere le orbite dal loro colore: quando il colore è quello corretto, il robot comincia a seguire il percorso in maniera indipendente; altrimenti, esegue un movimento su sé stesso e lampeggia per comunicare l'errore.

Questa proposta consente agli studenti di collaborare tra di loro, ed ha il vantaggio di materializzare ognuno dei pianeti sotto forma di robot. Inoltre il robot scelto è un componente di

mercato e di conseguenza già verificato, cosa che ne garantisce la robustezza e l'affidabilità. Tuttavia, non si è ritenuto fattibile scalare le capacità di rilevamento dell'input tenendo in mente come riferimento gruppi di venti studenti, e sono emerse preoccupazioni circa il danneggiamento del tavolo a seguito dell'uso continuo.



(a) Setup iniziale (b) Associazione delle curiosità

Figura 3.1: Esemplificazione della proposta in piccola scala. Illustrazioni di Lorenza Abbate.

3.2.2 Proposta su larga scala con robot personalizzato

La seconda proposta prevede la predisposizione a terra di un tappeto o altro tipo di stampa sul quale sono riportate le orbite dei pianeti. Su di essa viene proiettata, in sovraimpressione, una vista del Sistema solare. Agli studenti viene affidato un robot di forma sferica, capace di muoversi seguendo le orbite sul tracciato. Per ognuna delle orbite viene evidenziato un punto di partenza, che fungerà da riferimento per il robot.

Ogni sessione si focalizza su di un singolo pianeta, scelto dal tutor durante la preparazione dell'attività. Tale scelta viene comunicata agli studenti, che dovranno individuare l'orbita corretta. Uno di loro dovrà collocare il robot nel rispettivo punto di partenza, che si illuminerà in caso di successo.

Successivamente al posizionamento del robot inizierà l'attività in cui agli studenti vengono poste delle domande. Gli studenti vengono divisi in due gruppi. A ciascun gruppo viene affidato un mazzo di 24 carte, ognuna recante una caratteristica degli otto pianeti (tre curiosità per pianeta). A turno, dovranno avvicinare una carta al robot, il quale fornirà un responso positivo o negativo a seconda che la carta appartenga effettivamente al pianeta oggetto della sessione. In caso di informazione corretta, sulla vista proiettata verrà mostrata un'animazione a tema.

Una volta collocate tutte e tre le informazioni corrette, ha inizio l'ultima fase dell'attività. Gli studenti vengono fatti allontanare dalla scena, e il tutor invia un comando al robot che dà inizio al suo movimento lungo l'orbita. L'immagine del pianeta rappresentato ne segue lo spostamento, in modo da rimanervi sovraimposta. Allo stesso tempo, gli altri pianeti, puramente virtuali, iniziano anch'essi il moto di rivoluzione attorno al Sole. Il tutor ha il controllo sull'esperienza, potendo mettere in pausa lo spostamento del robot e degli altri pianeti. Il moto del robot e dei pianeti può essere interrotto in qualunque momento dall'Educatore, ponendo fine all'esperienza. Tempo permettendo, è possibile iniziare una seconda sessione e scegliere un pianeta diverso.

Diversamente dalla proposta in piccola scala, questa variante può concentrarsi su un solo pianeta alla volta, limitando l'interazione del gruppo a un numero ristretto di studenti. Inoltre, a causa delle caratteristiche peculiari del robot descritto, non è possibile affidarsi a componenti già esistenti senza introdurre pesanti modifiche, per cui si rende necessaria la costruzione di un robot personalizzato, con la conseguente penalizzazione in termini di affidabilità. In compenso, i vincoli summenzionati sull'input sono assenti in questa proposta, e ad essere oggetto di usura sono principalmente il robot e le carte, la cui sostituzione è meno costosa rispetto al tavolo multimediale della prima proposta.

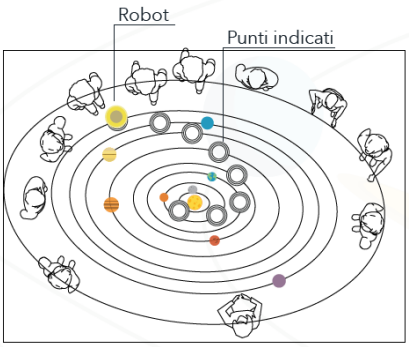


Figura 3.2: Esemplificazione della proposta su larga scala. Illustrazioni di Lorenza Abbate.

3.3 Analisi della proposta scelta

A seguito delle consultazioni tra i due studenti tesisti, i relatori ed il Laboratorio, si è scelto di sviluppare la seconda proposta, in quanto considerata maggiormente fattibile ed in linea di continuità con il percorso attuale. Nel seguito, verranno analizzati con maggiore dettaglio gli elementi utilizzati per l'implementazione, ed enunciati i requisiti funzionali e non cui deve adempiere.

3.3.1 Contesto

Il nuovo sistema prevede l'interazione con due attori principali (Figura 3.3):

- il tutor, che assume il controllo dell'esperienza per mezzo di un dispositivo remoto;
- gli studenti, sempre suddivisi in due squadre, che interagiscono col robot sotto indicazione del tutor.

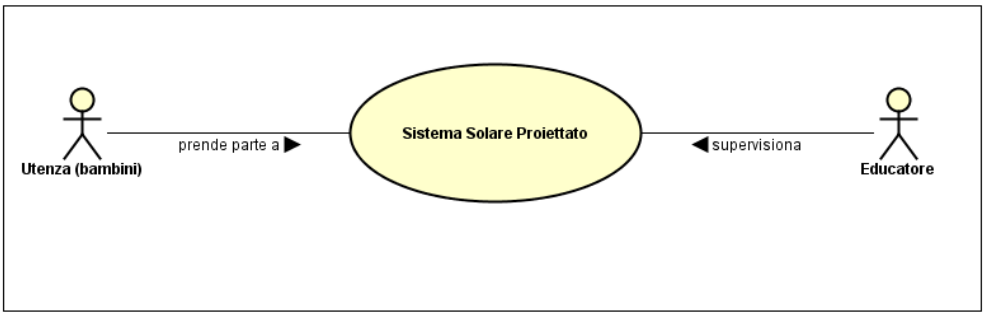


Figura 3.3: Diagramma di contesto della nuova proposta.

Il tracciato a terra è formato da otto piste ellittiche, ognuna rappresentante l'orbita di uno specifico pianeta. Su di esso viene proiettata una rappresentazione grafica del Sistema solare, per mezzo di un proiettore collegato ad un elaboratore centrale.

Il robot, dotato di sensore ottico, è in grado di riconoscere su quale orbita viene inizialmente piazzato (per mezzo di speciali codici) e di seguire il percorso in ambo le direzioni. È inoltre in grado di leggere delle tessere di prossimità preparate per l'esperienza, di rilevare la propria posizione lungo il tracciato, e di comunicare in maniera wireless con l'elaboratore centrale.

Il tutor coordina l'esperienza, dando istruzioni agli studenti e scandendo le diverse fasi per mezzo del dispositivo remoto. Quest'ultimo consente di inizializzare ed eventualmente interrompere l'esperienza, di scegliere l'argomento (ossia il pianeta su cui verterà) e di proseguire, da una fase all'altra.

Infine, agli studenti, oltre a essere data la possibilità di spostare il robot tra un'orbita e l'altra, viene fornito il set delle tessere che vengono riconosciute dal robot come caratteristiche del pianeta che sta impersonando. La Tabella 3.1 elenca le caratteristiche di ogni pianeta, come specificato dal Laboratorio.

Pianeta	*Curiosità*
Mercurio	Sono il pianeta più piccolo del Sistema solare. Sono simile alla Luna perché ci sono tanti crateri sulla mia superficie. Mi chiamo come il messaggero degli dei.
Venere	Sono il pianeta più luminoso dopo la Luna. Qui fa caldissimo, ci sono 465 °C Mi chiamo come la dea romana dell'amore.
Terra	Ci metto 24 ore a fare un giro su me stessa. Sono l'unico pianeta con acqua allo stato liquido e ossigeno. Sono l'unico pianeta del Sistema solare ad ospitare la vita.
Marte	Mi chiamo come il dio romano della guerra. Sono coperto da una polvere rugginosa. Forse una volta c'era su di me dell'acqua allo stato liquido.
Giove	Sono il più grande del Sistema solare. Sulla ma superficie c'è una tempesta enorme che dura da 300 anni. Ho più di 60 lune.
Saturno	Mi chiamano il Signore degli anelli. Sono il secondo pianeta più massiccio. Mi chiamo come il dio romano del tempo.
Urano	Qui fa freddissimo e piove sempre. Ci metto 84 anni a girare intorno al Sole. I miei venti possono raggiungere i 250 m/s.
Nettuno	Qui fa freddissimo, ci sono −220 °C. Sul mio satellite, Tritone, ci sono i vulcani di ghiaccio (criovulcani). Mi chiamo come il dio romano del mare.

Tabella 3.1: Tabella delle curiosità sui pianeti.

3.4 Requisiti funzionali e non funzionali di alto livello

All'interno della proposta scelta, è possibile evidenziare la presenza di tre macro-entità in interazione tra loro: due di esse si interfacciano direttamente con l'utenza (il robot con gli studenti, l'applicazione con il tutor), e costituiscono il sistema di controllo. La terza, ossia il sistema di proiezione, è la mente che collega le altre due, mantenendo un riferimento per lo stato dell'attività e fungendo da ponte di comunicazione. La Figura 3.4 ne mostra uno schema concettuale.

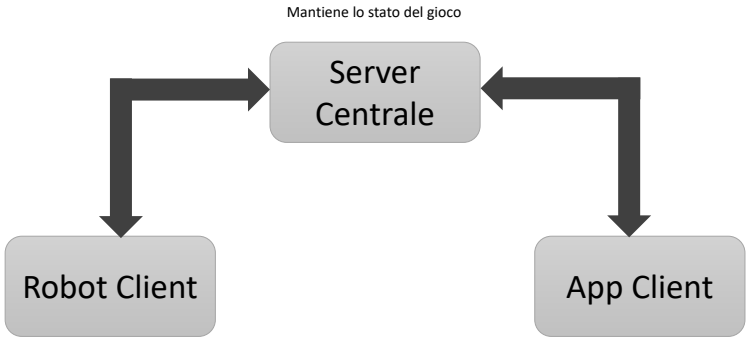


Figura 3.4: Schema ad alto livello del sistema.

3.4.1 Robot

A causa dei diversi compiti assegnatigli, il robot deve essere sufficientemente versatile da sottostare ad una serie di requisiti che riguardano l'interazione, la localizzazione e l'estetica, come ad esempio la capacità di leggere le tessere che rappresentano le curiosità o di comunicare con dispositivi wireless. Tali requisiti sono enumerati per intero nelle Tabelle 3.2 e 3.3. Al momento della definizione del progetto, nessuno dei prodotti in commercio presentava tutte le caratteristiche elencate, oppure le presentava in parte, ma possedeva delle limitazioni tali per cui si sono ritenuti poco pratici l'adattamento o l'estensione.

In virtù di ciò si è preferito optare per un robot auto-costruito, al costo ovviamente di complicarne la realizzazione e ridurne l'affidabilità. Dovendo a questo punto scegliere una piattaforma su cui sviluppare, ci si è orientati verso l'utilizzo di un computer single-board quale il Raspberry Pi. Rispetto a soluzioni alternative, quali ad esempio adoperare un micro-controllore come Arduino, il Raspberry Pi presenta numerosi vantaggi:

- è in grado di gestire un sistema operativo a tutti gli effetti, permettendo una maggiore modularità dei sotto-sistemi da realizzare, che possono essere implementati come singoli programmi in comunicazione tra loro;
- grazie alla maggiore potenza di calcolo, è in grado di effettuare gran parte dell'elaborazione direttamente sulla scheda stessa, senza dover delegare nulla ad un calcolatore centrale;
- è possibile installare o, in assenza di pacchetti già pronti, direttamente compilare un gran numero di librerie pensate per l'ambiente Linux;
- esiste un ventaglio molto ampio di moduli Python pensati appositamente per comandarne i bus ed i GPIO (General Purpose Input / Output, i pin programmabili del Raspberry Pi).

Il prezzo da pagare, in questo caso, è la bassa reattività rispetto a quella che si avrebbe con un micro-controllore, più adatto a effettuare operazioni di controllo in catena chiusa come quelle necessarie a far seguire il tracciato al robot. Tuttavia, mantenendosi su velocità moderate (dai 20 ai 60 cm/s in base al raggio dell'orbita) si riesce ad ottenere un controllo stabile ed a mantenere il robot ben allineato sul percorso.

3.4.2 Sistema di proiezione

Con questo termine si fa riferimento al sistema preposto a gestire gli elementi grafici visualizzati sul terreno, sovraimposti ai tracciati delle orbite ed al rivestimento del robot. Esso è formato da un proiettore rivolto verso il terreno, collegato a un computer centrale sul quale viene eseguito un programma grafico realizzato tramite il motore di gioco Unity [7]. Oltre alla logica sottostante, è cruciale che tale sistema sia in grado di adattarsi alle diverse condizioni ambientali in cui verrà montato, regolando la posizione virtuale dell'immagine proiettata ed il livello di ingrandimento di quest'ultima. Deve inoltre fornire i contenuti grafici necessari a far comprendere i contenuti dell'esperienza all'utenza fruitrice. Le Tabelle 3.5 e 3.6 riassumono la lista dei requisiti.

3.4.3 Controllo per il tutor

Con questo termine ci si riferisce genericamente ad un dispositivo, fornito al tutor, che gli consenta di svolgere le mansioni di cui in Tabella 3.4. Tale controllo verrà implementato tramite un'applicazione mobile per sistemi Android, piuttosto che un'applicazione per iOS o un controllo integrato in qualche dispositivo *embedded* (come un telecomando), garantendo i seguenti vantaggi:

- rispetto ad iOS, Android fornisce un SDK gratuito senza bisogno di particolari licenze;
- un'applicazione mobile non richiede la costruzione di hardware personalizzato, ma può fare uso di un tablet o uno smartphone commerciali;
- dal punto di vista della familiarità dell'utenza, le applicazioni mobili sono ormai diffusissime, rendendo minimo il tempo di apprendimento richiesto da parte del tutor.

Codice	*Nome*	*Descrizione*
R1	Movimento	Deve essere in grado di muoversi autonomamente, a velocità prestabilita, seguendo il tracciato.
R2	Rilevazione	Deve essere in grado di distinguere, in punti prestabiliti, un'orbita dall'altra.
R3	Posizione	Deve essere in grado di rilevare con precisione nell'ordine di 1 cm la propria posizione lungo un'orbita.
R4	Comunicazione	Deve poter ricevere ed inviare dati attraverso un sistema senza fili (ad esempio WiFi 802.11 o Bluetooth 802.15).
R5	Tessere	Deve poter rilevare tessere di prossimità alla distanza minima di 1 cm da un punto arbitrariamente scelto della sua superficie non a contatto col terreno.

Tabella 3.2: Requisiti funzionali per il robot.

Codice	*Nome*	*Descrizione*
RN1	Aspetto	Deve avere la forma di una calotta sferica, di raggio 9 cm. L'altezza dev'essere non inferiore a 12 cm.
RN2	Materiale	Deve essere di colore bianco e opaco in modo da riflettere l'immagine proveniente dal sistema di proiezione.

Tabella 3.3: Requisiti non funzionali per il robot.

Codice	*Nome*	*Descrizione*
A1	Dati	Deve poter ricevere ed inviare dati attraverso la medesima rete a cui sono connessi il robot e il dispositivo di controllo.
A2	Pianeta	Deve permettere di inizializzare l'esperienza scegliendo il pianeta soggetto della sessione.
A3	Avanzamento	Deve permettere di far avanzare l'esperienza, dando inizio alle fasi delle curiosità e di rivoluzione dei pianeti.
A4	Interruzione	Deve permettere in qualunque momento di interrompere l'esperienza e ricominciare.
A5	Monitoraggio	Deve mostrare lo stato corrente dell'esperienza, quale ad esempio quali delle tessere relative alle curiosità sono state già riconosciute.

Tabella 3.4: Requisiti funzionali per il dispositivo di controllo.

Codice	*Nome*	*Descrizione*
U1	Dati	Deve poter ricevere ed inviare dati attraverso la medesima rete a cui sono connessi il robot e il dispositivo di controllo.
U2	Tracciamento	Deve poter rilevare la posizione del robot attraverso le diverse fasi. Nello specifico, deve poter riconoscere quando il robot è stato posizionato sul codice di una specifica orbita, e durante la fase di movimento deve poter allineare la posizione dell'elemento grafico corrispondente al pianeta prescelto con quella del robot.
U3	Regolazione	Il livello di zoom, nonché la posizione della telecamera virtuale rispetto all'area inquadrata devono essere regolabili a piacere in modo da adattarsi alla configurazione finale.
U4	Animazione	Per ogni tessera rilevata, deve mostrare un effetto grafico, che può essere un sistema particellare applicato lungo l'orbita del pianeta corrispondente o un'immagine animata.
U5	Aspetto	Deve rappresentare una vista del Sistema solare limitata ai suoi pianeti, al Sole e alla fascia di asteroidi. Il pianeta scelto deve apparire con una scala differente rispetto agli altri, in modo da adattarsi alle dimensioni del robot.

Tabella 3.5: Requisiti funzionali per il sistema di proiezione.

Codice	*Nome*	*Descrizione*
UN1	Area	Il proiettore deve essere in grado di ricoprire un'area rettangolare di almeno 350 cm di larghezza per 300 di altezza.
UN2	Luminosità	La proiezione deve risultare visibile da un pubblico di bambini fra gli 8 e i 12 anni all'interno di una stanza non illuminata e con finestre oscurate.
UN3	Orbite	Le orbite devono avere una forma ellittica, la cui eccentricità e semiasse maggiore devono poter essere regolate a piacimento.

Tabella 3.6: Requisiti non funzionali per il sistema di proiezione.

Capitolo 4

Implementazione

Il presente capitolo si occuperà di illustrare i dettagli implementativi dietro la realizzazione dei sistemi necessari allo svolgimento dell'esperienza appena descritta. Dopo un'introduzione all'architettura generale, dove verranno spiegate le tecniche scelte per mettere in comunicazione tra loro le diverse parti, si procederà con lo scendere nel dettaglio per ognuna di esse mostrandone i diversi componenti.

4.1 Visione d'insieme

La Figura 4.1 mostra la naturale suddivisione del sistema in tre macro componenti in comunicazione tra loro: una client app, che come detto permette al tutor di coordinare l'esperienza; il client a bordo del robot; un server centrale sul quale è in esecuzione l'applicazione Unity. Essendo quest'ultimo il nodo più stabile, ossia un PC normalmente inaccessibile ai non addetti alla manutenzione, è stato scelto come punto di riferimento per gli altri due. Ogni operazione cui vanno incontro sia il robot, sia l'applicazione, passa attraverso il server, che funge da intermediario e da ponte. Le seguenti Sezioni introdurranno l'architettura dietro il sistema di comunicazione, che verrà successivamente approfondita nei dettagli implementativi di ognuna delle tre parti.

4.1.1 Protocollo di comunicazione e problematiche di sicurezza

Tutte le informazioni trasmesse tramite la rete (locale) vengono inviate attraverso un flusso di caratteri, codificati in UTF-8, e con istruzioni successive separate da *line-feed*. Le istruzioni di

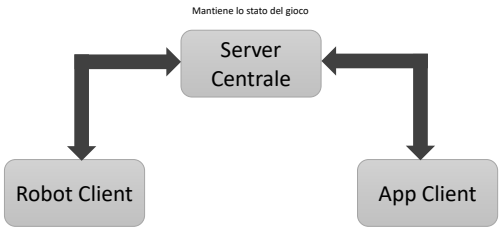


Figura 4.1: Schema generale dell'interazione fra i tre sistemi dell'attività.

controllo sono espresse nel seguente formato:

```
NomeComando[ Argomento1[ Argomento2...]]
```

Fa eccezione la stringa che rappresenta lo stato dell'esperienza (descritta nella Sezione successiva). Data la semplicità del protocollo, si è evitato di implementare un sistema più raffinato basato su JSON o XML in quanto non necessario.

Il testo viene trasmesso in chiaro, non protetto da tecniche di cifratura. Sebbene, potenzialmente, ciò esponga a diversi tipi di manomissione (la più banale potrebbe essere un falso server), si è deciso di non agire in tal senso.

- L'intero apparato gira su una rete locale isolata dalla restante del Laboratorio; gli unici punti di accesso sono le porte LAN dell'access point su cui è costruita la rete (che andrebbero protette fisicamente) e il punto d'accesso wireless, protetto dal protocollo WPA2-PSK. Una debole forma di autenticazione è quindi insita nella conoscenza della password WiFi.

- Trattandosi di un software educativo per conto di un centro didattico, non vi sarebbe guadagno economico nel manometterlo e pertanto la probabilità di un attacco è alquanto bassa.

- Se anche qualcuno riuscisse ad intromettersi nella rete locale e a manomettere l'esperienza, il danno sarebbe minimo. Il robot non può essere comandato direttamente, può soltanto seguire una linea sul pavimento, e in ogni caso non si muove a velocità tali da costituire un pericolo. Il peggior scenario possibile sarebbe l'impossibilità ad usare il sistema (che sarebbe comunque parte di un'attività sperimentale e non una fonte di guadagno vitale per il Laboratorio) fino all'identificazione del problema.

Alla luce di queste considerazioni, la minaccia (intesa come probabilità di un attacco moltiplicata per il danno che essa causerebbe) è veramente minima tanto da non giustificare l'implementazione di sistemi di autenticazione o cifratura.

4.1.2 Identificazione del server nella rete locale

La pubblicizzazione del server all'interno della rete locale avviene mediante broadcast UDP. Ad intervalli regolari, questi invia periodicamente, sulla porta 6600, il seguente messaggio identificativo:

```
123STAR_SERVER_BROAD
```

L'indirizzo broadcast della rete viene ottenuto dalla configurazione della scheda di rete primaria, combinandone l'indirizzo IPv4 con la relativa maschera di rete mediante la ben nota relazione:

```
BroadcastIpv4 = LocalIpv4 OR (NOT(NetMask));
```

4.1.3 Stato di avanzamento dell'esperienza

Trattandosi di un sistema distribuito ci si pone il problema di decidere dove mantenere, e se replicare le informazioni relative allo stato dell'esperienza. Infatti, in caso di fallimento di uno dei nodi (ad esempio, disconnessione accidentale del dispositivo di controllo dalla rete wireless) è importante poter riprendere da dove si è interrotto.

Data l'intrinseca stabilità del server, si è deciso di usarlo come singolo garante dello stato dell'esperienza. In caso di fallimento della rete o nuova inizializzazione, è suo compito comunicarne i dettagli ai nodi client, in modo da poter riprendere correttamente la partita. Nello specifico è necessario tenere traccia delle seguenti informazioni:

- stato di connessione dei client (parziale / completo);
- fase dell'esperienza;
- pianeta scelto;
- numero di tessere riconosciute durante la fase delle curiosità.

Tali informazioni vengono combinate tra loro in una singola stringa nel seguente formato, ed inviate all'inizio della connessione tra un client ed il server:

```
[HOLDING ]<NomeFase>[ <NomePianeta>[ <ListaTessere>]]
```

Gli elementi della stringa hanno il significato riportato di seguito.

Segnalatore `HOLDING`

Se presente, indica che una delle parti non è ancora connessa e non è quindi possibile riprendere l'attività.

Fase corrente

Deve essere uno tra:

`SETUPPING` Fase preparatoria, il tutor deve ancora scegliere il pianeta.

`SCANNING` Fase di ricerca del pianeta scelto. Viene ripetuta due volte, dopo la fase iniziale per far identificare il pianeta agli studenti (lista tessere vuota) e dopo la parte delle curiosità con le tessere per rimettere il robot in posizione prima dell'inizio del moto (lista tessere piena).

`QUIZ` Fase dell'attività con le tessere.

`MOVE` Fase di moto dei pianeti. In realtà quest'ultimo non può mai essere inviato nella stringa di stato in quanto, per ragioni legate al tracciamento del robot, in caso di disconnessioni durante il moto dei pianeti lo stato viene riportato a `SCANNING` per consentire il riposizionamento da zero.

Pianeta Il nome in lingua inglese, con iniziale maiuscola e corpo minuscolo del pianeta scelto (ossia le costanti `Mercury`, `Venus`, `Earth` ...).

Tessere L'elenco delle tessere accettate durante la fase delle curiosità, formata da una sequenza di stringhe alfanumeriche (e dal carattere `_`) separate dal punto e virgola (`;`), ad esempio:

```
LITTLE;ROCK;HARD_TO_SEE
```

4.1.4 Gestione dei timeout

Le potenziali perdite improvvise di connessione vengono gestite mediante messaggi di mantenimento espliciti. Qualora uno dei nodi non riceva messaggi da una controparte per più di tre

secondi, esso è tenuto a inviare il messaggio `KEEPALIVE`. Quando un nodo riceve questo comando è tenuto a rispondere tramite il messaggio `IMALIVE`. Se si verificano due timeout successivi senza alcun messaggio in mezzo il nodo deve assumere che la controparte non sia più connessa e interrompere la comunicazione. In questo modo, è possibile rilevare eventi inattesi riguardo lo stato di connessione senza fraintendere lunghi periodi di inattività (tipici di applicazioni interattive come questa).

4.2 Implementazione del robot

La tipologia scelta per la realizzazione del robot è quella del *line-follower*, ossia un robot in grado di muoversi lungo una linea ad elevato contrasto rispetto al fondo circostante, arricchito con dispositivi aggiuntivi per consentire le funzionalità necessarie alle diverse fasi dell'esperienza. La Figura 4.2 fornisce uno schema generale che coniuga risorse hardware e software in modo da evidenziare le loro interazioni. Nelle seguenti Sezioni verrà espansa la descrizione del robot dal punto di vista hardware e poi software. Per un approfondimento dettagliato sull'assemblaggio dei componenti, si veda l'Appendice A.

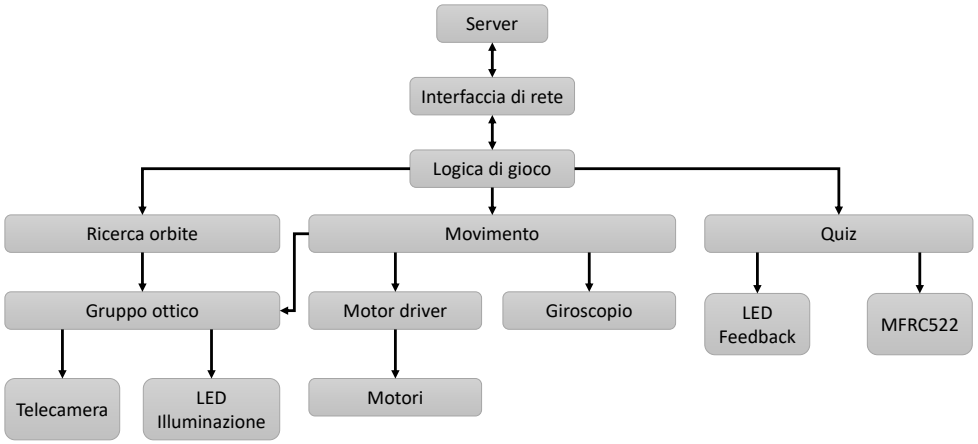


Figura 4.2: Funzionamento generale del robot.

4.2.1 Caratteristiche hardware

Come già accennato, la piattaforma di sviluppo su cui è basato il robot è un Raspberry Pi, modello 3B+, sul quale sono state collegate le seguenti periferiche:

- un Raspberry Pi Camera Module V2;
- un giroscopio Adafruit L3GD20H;
- una coppia di motori DC DFRobot FIT0450;
- un driver motore Adafruit TB6612;
- un ricetrasmettitore MFRC522;
- un set di cinque LED diffusi:
  - una coppia bianca di supporto alla telecamera;
  - una coppia rosso / verde per fornire feedback al rilevamento delle tessere;
  - uno verde isolato per segnalare esternamente lo stato di accensione.

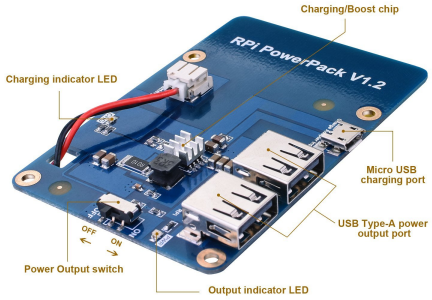
L'alimentazione viene fornita da un circuito stampato con batteria a litio e due uscite USB, una diretta al Raspberry e l'altra ai motori, visibile in Figura 4.3a. Come supporto, è stato scelto uno chassis circolare di diametro 10 cm, adattato con alcuni supporti stampati in 3D, del quale un'immagine commerciale è visibile in Figura 4.3b.

4.2.2 Caratteristiche software

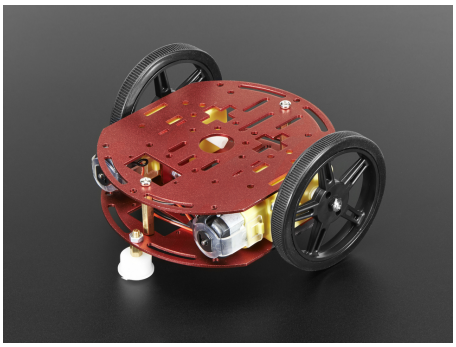
La figura 4.4 illustra uno schema concettuale di come è organizzato il software di gestione del robot. Un modulo centrale (il `MasterModule`) si occupa delle operazioni di inizializzazione e di contattare il server. Dopo aver ricevuto informazioni circa lo stato corrente dell'attività inizializza una `GameMachine` che esegue le operazioni necessarie alla fase corrente, per poi entrare in un ciclo di lettura dal server in attesa di ulteriori istruzioni.

4.2.2.1 Gestione del parallelismo

Durante lo sviluppo di questo progetto si è deciso di fare uso di tecniche sperimentali per la gestione del parallelismo, che mirano a rendere meno complessa, più leggibile e meno prona ad



(a) Modulo batteria. Immagine tratta da https://www.raspberryitalia.it/.



(b) Chassis del robot. Immagine tratta da https://www.adafruit.com/.

Figura 4.3

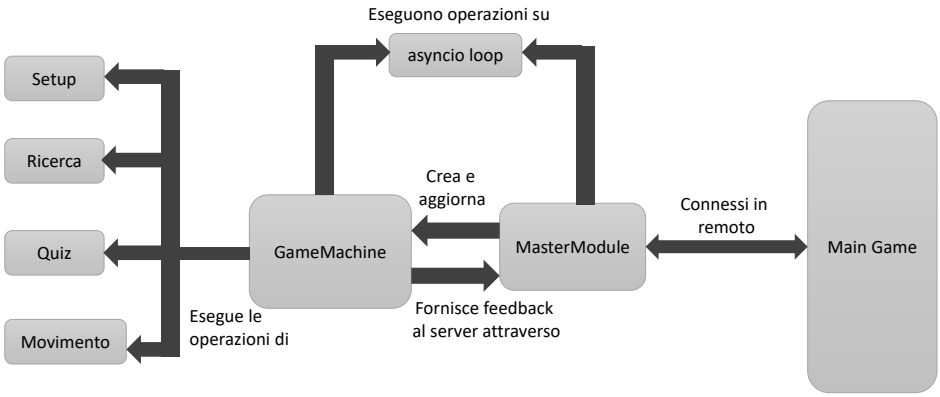


Figura 4.4: Schema generale del software di gestione del robot.

errori la stesura di codice concorrente, senza necessariamente ricorrere agli strumenti classici del multithreading. Tali tecniche sono talvolta raccolte sotto il termine generico di *programmazione asincrona*, e pur variando per caratteristiche implementative e funzionalità a seconda del linguaggio fanno spesso uso di due parole chiave comuni: `async` e `await`.

Nel caso del robot, si è deciso di fare uso del modulo `asyncio` [8] di Python. Riassumendone brevemente le caratteristiche, `asyncio` è un modulo per la schedulazione di operazioni parallele in maniera cooperativa su un singolo thread mediante coroutine (operazioni potenzialmente bloccanti

da eseguire in parallelo, identificate dalla parola chiave `async`). Le coroutine vengono incapsulate in `Task` eseguiti su uno scheduler chiamato `loop`. In ogni istante una sola operazione è realmente in esecuzione, eliminando alla radice problemi intrinseci della concorrenza tramite multithreading.

In `asyncio` la cessione del diritto ad eseguire viene fatta spontaneamente dalle singole coroutine, attraverso il comando `await`. Una coroutine che invoca `await` su un'operazione bloccante dichiara di non aver più bisogno di tenere il controllo della CPU e lo rilascia al `loop` sul quale sta eseguendo, in attesa che l'operazione bloccante venga completata. Di conseguenza, sono sempre ben noti i punti di sincronizzazione fra più operazioni distinte.

Oltre ai vantaggi già citati, l'uso del paradigma `async/await` rende più chiara e ordinata la suddivisione del codice in sotto-operazioni parallele, più simile a un programma non concorrente, evitando fastidiosi meccanismi di sincronizzazione che si sarebbero altrimenti resi necessari per coordinare la comunicazione col server coi diversi sottosistemi posti a gestire le fasi dell'esperienza.

L'implementazione del robot presenta un'eccezione rispetto alle altri due componenti, nei quali la programmazione asincrona è lo strumento dominante per il parallelismo. A corredo di `aysncio` e potendo fare leva sui potenti meccanismi di comunicazione inter-processo di Linux, alcuni moduli sono stati implementati come programmi a sé stanti che operano attraverso lo standard input / output; per la loro gestione è stato utilizzato il modulo `pexpect` [9], che facilita operazioni come l'attesa di pattern di output noti in maniera conforme ad `asyncio`.

4.2.2.2 Suddivisione dei moduli

La Figura 4.5 è una vista gerarchica dei diversi moduli software implementati per il robot. Gli apici evidenziano programmi indipendenti gestiti tramite il modulo `pexpect`, secondo la leggenda di cui in figura. In assenza di tali, sono da intendersi come classi Python gestite dal medesimo script che avvia il `MasterModule`.

`MasterModule`
: Funge da interfaccia verso il server, inizializza la `GameMachine`.

`GameMachine`
: Gestisce lo stato dell'attività, esegue operazioni specifiche per ogni fase.

`PlanetMarkerScanner`

Programma indipendente in C++, ricerca il marcatore caratteristico del pianeta passato come argomento. Restituisce a ogni ciclo di elaborazione la stringa `PLANET_OK` se ha rilevato il marcatore, `PLANET_KO` altrimenti. Vedere la Sezione 4.2.2.8

`TagCardReader`

Programma indipendente in Python, inizializza il ricevitore MFRC522 e comanda i LED tessera giusta / tessera sbagliata quando viene avvicinata una tessera valida per le curiosità. Restituisce la stringa univoca della tessera quando essa appartiene al pianeta passato come argomento. Vedere la Sezione 4.2.2.9.

`RobotController`

Modulo che gestisce la catena di controllo del robot. Vedere la Sezione 4.2.2.6.

`CameraLineFollower`

Programma indipendente in C++, rileva la presenza del tracciato e restituisce l'errore di posizione a ogni frame, oltre a segnalare la presenza di possibili bande di allineamento.

`SensorFilter`

Filtro di Kalman aggiornato da una coroutine che effettua periodicamente il polling delle informazioni del giroscopio. Consente di effettuare la correzione delle letture quando viene rilevata una banda di riallineamento sul percorso.

`MotorControl`

Legge l'output del `CameraLineFollower` ed effettua le operazioni di controllo PID. Scrive il risultato sul `MotorWriter`.

`MotorWriter`

Gestisce i GPIO collegati al motor driver; modifica velocità e direzione di rotazione dei motori.

4.2.2.3 Inizializzazione

La fase preparatoria corrisponde all'implementazione delle fasi descritte nelle Sezioni 4.1.2 e 4.1.3. Il robot resta in ascolto per comunicazioni broadcast UDP sulla porta 6600, in attesa di riconoscere la stringa identificativa del server. Una volta ricevuta ottiene dal pacchetto UDP l'indirizzo del

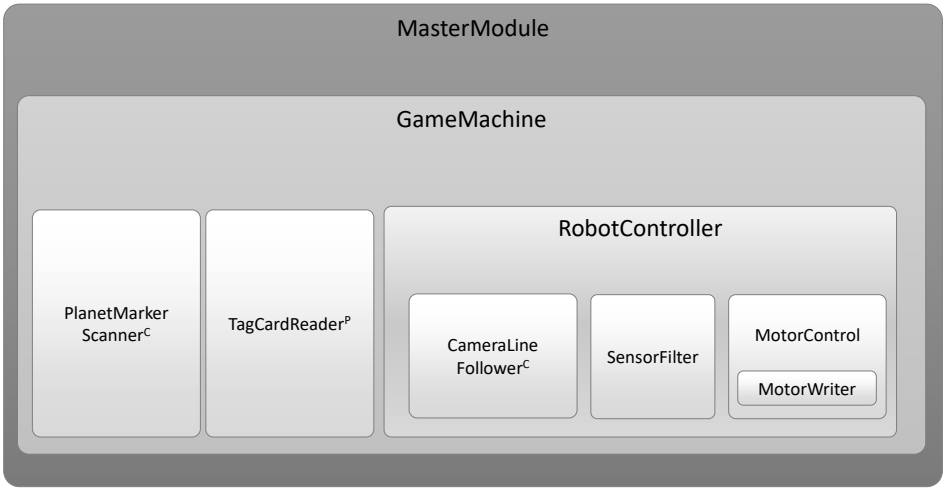


Figura 4.5: Suddivisione in moduli del software in esecuzione sul robot.

server, quindi procede aprendo una connessione TCP sempre sulla porta 6600. Dopo aver ricevuto lo stato dell'attività come descritto nella Sezione 4.1.3 il `MasterModule` procede avviando due coroutine:

- una corrispondente a un ciclo di lettura dal server;
- l'altra specifica per la fase corrente dell'attività, definita all'interno della classe `GameMachine` che incapsula lo stato dell'esperienza.

4.2.2.4 Lista dei comandi accettati dal robot

Ricordando che ogni messaggio dal server deve seguire la sintassi `<NomeComando>[ Argomento1[ Argomento2...]]`, la seguente è la lista dei comandi riconosciuti:

`KEEPALIVE` Messaggio di mantenimento della connessione. Si risponde con `IMALIVE`. Si veda la Sezione 4.1.4.

`IMALIVE` Messaggio di mantenimento della connessione. Nessuna risposta necessaria (il reset del timeout è implicito nella ricezione di un qualsiasi messaggio).

`HOLD` Una delle parti si è disconnessa. Interrompe l'operazione corrispondente all'attuale fase; se la fase corrente è `MOVE`, ripristina lo stato di `SCANNING`.

`RESUME` Tutte le parti hanno ristabilito la connessione. Ricomincia l'operazione corrispondente alla fase attuale.

`ABORT` Richiesta di terminazione della partita. Interrompe la comunicazione e ricomincia dall'inizio.

`PROTOCOL_EXCEPTION`
Si è verificato un errore logico nella trasmissione. Interrompe la partita e ricomincia.

`SCAN_PLANET <PlanetName>`
Passa allo stato di ricerca del pianeta. Se si parte dalla fase di inizializzazione, `<PlanetName>` deve essere un nome di pianeta valido (vedi 4.1.3). Se si sta partendo dalla conclusione della fase delle curiosità, `<PlanetName>` viene ignorato.

`QUIZ_START`
Valido solo nella fase di ricerca del pianeta. Inizia la fase delle curiosità.

`MOVE_START`
Valido solo nella fase di ricerca del pianeta dopo aver rilevato tutte le tessere delle curiosità. Inizia la fase di moto dei pianeti.

`ROBOT_POSITION`
Valido solo nella fase di moto dei pianeti. Risponde inviando l'attuale orientamento del robot, la sua velocità angolare, e il valore angolare usato nell'ultima banda di riallineamento (vedi 4.2.2.7).

`ROBOT_PAUSE`
Valido solo nella fase di moto dei pianeti. Comanda l'arresto dei motori per mettere temporaneamente in pausa lo spostamento dei pianeti.

`ROBOT_RESUME`
Valido solo nella fase di moto dei pianeti. Comanda di riattivare i motori per riprendere lo spostamento dei pianeti.

4.2.2.5 Transizioni di fase e operazioni da eseguire in ognuna

La classe `GameMachine` implementa i metodi corrispondenti ai comandi `SCAN_PLANET`, `QUIZ_START`, `MOVE_START`, `HOLD` e `RESUME`. Prima di eseguire la coroutine relativa al nuovo stato, è necessario

interrompere in maniera ordinata quella corrente come descritto dal frammento di codice riportato in Figura 4.6. L'istruzione alla linea 5 serve a permettere al `Task` cancellato di completare correttamente ed eventualmente gestire eccezioni pendenti.

```
1  async def cancel_update_task(self):
2      if self.update_task is not None:
3          try:
4              self.update_task.cancel()
5              await self.update_task
6          except asyncio.CancelledError:
7              pass
8          finally:
9              self.update_task = None
```

Figura 4.6: [Python] Metodo `cancel_update_task` della classe `GameMachine`.

Le seguenti sono le descrizioni delle coroutine di ogni fase, con l'eccezione della fase di setup (la quale completa immediatamente non richiedendo ulteriori passaggi).

`SCANNING` Il modulo per il rilevamento dei marcatori viene avviato e la coroutine entra in un ciclo di lettura da quest'ultimo. Ogni responso (`PLANET_OK / KO`) viene inoltrato al server.

`QUIZ` Il modulo per il rilevamento delle tessere viene avviato e la coroutine entra in un ciclo di lettura da quest'ultimo. Le stringhe identificative delle tessere correttamente rilevate vengono inoltrate al server tramite il comando `TAG <IdentificativoTessera>`. Una volta rilevate tutte e tre le tessere la coroutine ha termine.

`MOVE` Viene istanziato un nuovo `RobotController`, il quale genera due coroutine:

- un ciclo che avvia il modulo `CameraLineFollower`, ne legge ripetutamente l'output (offset rispetto al centro della linea, se la linea è stata effettivamente rilevata, se è stata rilevata una banda di riallineamento) e lo fornisce al modulo `MotorControl`;
- un ciclo di aggiornamento sul modulo `SensorFilter`.

Durante questa fase il `MasterModule` è in grado di ottenere gli aggiornamenti posizionali dai membri di `RobotController`.

4.2.2.6 Catena di controllo

Determinazione dell'errore Robot di questo tipo vengono solitamente implementati attraverso coppie di foto-emettitori e rilevatori infrarossi, che verificano la posizione della linea rispetto al robot, emettendo luce verso il terreno e verificando quale delle coppie riceve una risposta (nulla in presenza di una superficie scura assorbente). In questo caso, potendo godere della maggiore potenza di calcolo offerta dal Raspberry Pi si è preferito sfruttare invece una telecamera nel visibile insieme a un semplice algoritmo di elaborazione a sogliatura. L'implementazione fa uso della libreria OpenCV [10], lo standard de facto per l'elaborazione di immagini per la visione artificiale. La Figura 4.7 ne è un estratto semplificato, in C++ (la versione completa contiene ulteriori codici di controllo per segnalare il caso in cui non si sia rilevata alcuna linea). In sostanza, può essere riassunto nei seguenti passi:

1. Ottieni l'immagine corrente in scala di grigi;
2. applica un filtro sfocatura all'immagine;
3. effettua la sogliatura binaria in base a un parametro prefissato;
4. applica l'operazione di chiusura all'immagine sogliata;
5. determina il contorno maggiore dell'immagine binaria risultante;
6. calcola il baricentro del contorno maggiore;
7. restituisci la differenza fra l'orizzontale del baricentro del contorno maggiore e il centro dell'immagine.

Attuazione sui motori La coppia di motori che governano le ruote del robot è pilotata mediante controllo PWM, il cui segnale è inviato ad un driver dedicato. L'alimentazione viene fornita dalla medesima batteria che alimenta il Raspberry, sfruttando l'uscita USB secondaria.

Il modulo di gestione dei motori è un controllore PID che riceve in ingresso l'errore segnalato dal modulo ottico appena descritto. L'errore è segnalato positivo quando la linea si trova alla destra del robot, pertanto l'output verrà elaborato come:

$$\begin{aligned} v_l &= v_b + f(\varepsilon) \\ v_r &= v_b - f(\varepsilon) \end{aligned} \tag{4.1}$$

```
while (running) {
    cap.grab();
    cap.retrieve(frame);

    // Apply blur
    cv::Mat processed = preprocess(subframe);

    // Convert image to grayscale
    cv::cvtColor(processed, processed, cv::COLOR_BGR2GRAY);

    // Apply thresholding. line_thr is the thresholding value
    cv::threshold(processed, processed, line_thr, 255, CV_THRESH_BINARY_INV);

    // Close the thresholded image to merge small gaps
    cv::Mat kernel = cv::getStructuringElement(cv::MORPH_RECT, cv::Size(5,
        5));
    cv::morphologyEx(processed, processed, cv::MORPH_CLOSE, kernel);

    // Find contours
    std::vector<std::vector<cv::Point>> contours;
    cv::findContours(processed.clone(), contours, 1, cv::CHAIN_APPROX_NONE);

    if (contours.size() > 0) {
        // Get biggest contour
        std::vector<cv::Point> maxContour = getMaxContour(contours);

        // Find blob center
        cv::Moments M = cv::moments(maxContour);
        int cx = (int)(M.m10 / M.m00);
        int cy = (int)(M.m01 / M.m00);

        // Output error
        std::cout << (int)(cx - processed.size().width) << std::endl;
    }
    else {
        std::cout << (int)(0) << std::endl;
    }

    usleep(33 * 1000);
}
```

Figura 4.7: [C++] Codice per rilevare l'errore di posizione del robot.

Dove v_b è la velocità base del robot (in scala $[-100, 100]$) e f il risultato dell'elaborazione PID. Nello specifico, dati D il valore corrente della derivata temporale di ε e I il suo integrale dall'inizio dell'elaborazione:

$$f(\varepsilon) = k_p\varepsilon + k_d D + k_i I \tag{4.2}$$

I valori di v_b, k_p, k_d e k_i sono stati determinati sperimentalmente per ognuno dei tracciati. Le velocità così calcolate vengono quindi applicate al driver dopo aver passato alcuni aggiustamenti riassunti nella Figura 4.8.

```
# Clamp to -100 ; +100
self.speed_right = float(min(100.0, max(-100.0, self.speed_right *
    MotorWriter.MOTOR_GAIN_RIGHT)))
self.speed_left = float(min(100.0, max(-100.0, self.speed_left *
    MotorWriter.MOTOR_GAIN_LEFT)))

# Rescale to 0.2 ; 1.0
self.motor_right.value = abs(self.speed_right) / 100.0 * 0.8 + 0.2
self.motor_left.value = abs(self.speed_left) / 100.0 * 0.8 + 0.2
```

Figura 4.8: [Python] Adattamento dei valori di velocità.

Entrambi i valori vengono limitati all'intervallo $[-100, 100]$ dopo aver applicato un guadagno correttivo per compensare in parte le differenti risposte dei motori. Il segnale PWM viene generato dalla classe `PWMOutput` del modulo Python `gpiozero`, che richiede il ciclo di lavoro espresso nell'intervallo $[0, 1]$. Quest'ultimo viene ottenuto a partire dai valori precedentemente calcolati, e scalato in modo tale che l'intervallo $[0, 100]$ corrisponda a $[0.2, 1.0]$. Ciò serve a superare la soglia d'attrito dei motori e delle ruote, stimata a un valore di velocità minimo corrispondente a 0.2.

4.2.2.7 Localizzazione

Per tenere traccia del posizionamento del robot lungo l'orbita si è pensato di utilizzare un semplice giroscopio a basso costo. Con riferimento alla Figura 4.9, se sono vere le seguenti condizioni:

- il robot inizia il proprio movimento a partire dal cerchio evidenziato ($\vartheta = \alpha = 0$), orientato verso la parte alta della figura;
- il giroscopio è adagiato sul robot parallelamente al terreno, così da farne coincidere l'asse z con la normale al pavimento;

- il robot segue fedelmente il tracciato senza oscillare o fare movimenti bruschi;

allora è possibile affermare che la direzione frontale del robot $\vec{v}$ è parallela e concorde alla tangente al tracciato $\frac{d\vec{p}}{d\vartheta}$. Il problema di risolvere quindi, istante per istante, l'angolo ϑ e la sua derivata $\phi = \frac{d\vartheta}{dt}$ si scompone in due sottoproblemi.

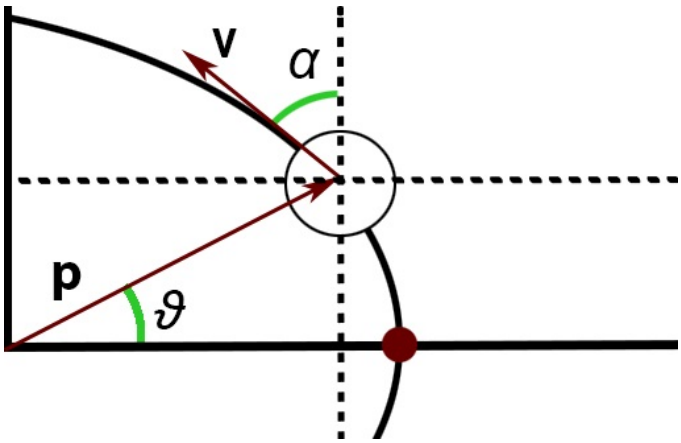


Figura 4.9: Relazione fra posizione del robot lungo l'orbita e suo orientamento.

Stimare l'orientamento del robot La lettura delle misurazioni del giroscopio fornisce una stima della velocità angolare, rispetto all'asse verticale locale, del robot stesso. Per poterne ricavare correttamente l'angolo nel tempo è necessario effettuare un'operazione di integrazione quanto più continua possibile nel tempo. Tale operazione è tuttavia ostacolata da due fattori:

- è sempre presente un bias di lettura, variabile lentamente nel tempo, che se non corretto causa una deriva progressiva dell'integrazione;
- le misure sono affette da rumore gaussiano bianco.

Procedendo in maniera naïve, integrando le letture di velocità, si avranno misure sempre meno accurate e affette da una deriva marcata. Per ovviare a ciò si è pensato di ricorrere a un filtro di Kalman, dopo una breve operazione di calibrazione, basato sulla cinematica del primo ordine.

Per prima cosa vengono presi 100 campioni con un periodo di campionamento di 10 ms. Da tali campioni si ricavano la velocità angolare media $\overline{\omega}$ e la sua varianza σ_ω^2

Successivamente si inizializza il filtro con vettore degli stati $[\alpha, \omega]$ e vettore delle letture $[\omega_{gyro}]$. Il bias di lettura è dato da $[\overline{\omega}]$ mentre la matrice delle covarianze è semplicemente $[\sigma_\omega^2]$. Ad ogni update, si aggiorna il filtro con la nuova lettura $[\omega_{gyro}(t)]$ e con la matrice di transizione:

$$\begin{bmatrix} 1.0 & \delta t \\ 0.0 & 1.0 \end{bmatrix} \tag{4.3}$$

Dove δt è il tempo trascorso dall'ultima iterazione e si assume che il robot stia ruotando a velocità approssimativamente costante. Il filtro produce dei nuovi valori per il vettore degli stati $[\alpha, \omega]$ che minimizza gli errori e corregge il drifting grazie alla calibrazione effettuata.

Si noti che la costanza della velocità di rotazione è solamente un'approssimazione, ed è vera solo quando il robot si muove a velocità lineare costante e l'orbita è circolare (in tal caso, la velocità di rotazione è banalmente $\|\vec{v}\| / r$). Quando l'orbita è ellittica la velocità di fase (e, conseguentemente quella di rotazione del robot su sé stesso) dipende in maniera non lineare dalla velocità di spostamento del robot; pertanto si renderebbe necessaria l'applicazione di un filtro di Kalman esteso, cosa che si è preferito evitare per non complicare eccessivamente il modello a fronte di un'ellitticità delle orbite interessate molto bassa (in molti casi inferiore a 0.1), rilevante solo nel caso di Mercurio ($e = 0.2$) che è il pianeta dall'orbita più piccola e quindi meno influenzato a livello visivo dalla difformità del modello con la realtà.

Conversione in posizione lungo l'orbita Siano A e B i semiassi maggiore e minore dell'orbita, rispettivamente. Sempre con riferimento alla Figura 4.9, si sa che per il vettore di posizione $\vec{p}$ vale:

$$\vec{p} = \begin{bmatrix} A\cos(\vartheta) \\ B\sin(\vartheta) \end{bmatrix} \tag{4.4}$$

Da cui è possibile ricavare il vettore tangente $\vec{v}$:

$$\vec{v} = \frac{d\vec{p}}{d\vartheta} = \begin{bmatrix} -A\sin(\vartheta) \\ B\cos(\vartheta) \end{bmatrix} \tag{4.5}$$

Da semplici considerazioni geometriche, è possibile ricavare il versore $\hat{v}$ a partire dal valore di α...

$$\hat{v} = \begin{bmatrix} -\sin(\alpha) \\ \cos(\alpha) \end{bmatrix} \tag{4.6}$$

Normalizzando $\vec{v}$ ed uguagliando i due versori si ottiene...

$$\hat{v} = \frac{\vec{v}}{\|\vec{v}\|} = \frac{1}{\sqrt{A^2\ \sin^2(\vartheta) + B^2\ \cos^2(\vartheta)}} \begin{bmatrix} -A\sin(\vartheta) \\ B\cos(\vartheta) \end{bmatrix} = \begin{bmatrix} -\sin(\alpha) \\ \cos(\alpha) \end{bmatrix} \tag{4.7}$$

Infine, uguagliando le componenti e dividendo membro a membro si ottiene la relazione finale:

$$\tan(\vartheta) = \frac{B}{A}\tan(\alpha) \Rightarrow \vartheta = \tan^{-1}\left(\frac{B}{A}\tan(\alpha)\right) \tag{4.8}$$

È utile dal punto di vista computazionale ricavare ϑ dalla funzione `atan2` in modo da ottenere un risultato nell'intervallo $[-\pi, \pi]$.

```
theta = atan2(B * sin(alpha), A * cos(alpha))
```

Partendo dall'equazione precedente, è possibile anche ricavare la relazione tra la velocità angolare ω del robot e la variazione ϕ dell'angolo di fase.

$$\frac{d\tan(\vartheta)}{dt} = \frac{1}{\cos^2(\vartheta)}\ \frac{d\vartheta}{dt} = \frac{B}{A}\ \frac{d\tan(\alpha)}{dt} = \frac{B}{A}\ \frac{1}{\cos^2(\alpha)}\ \frac{d\alpha}{dt} \tag{4.9}$$

$$\phi = \frac{B}{A}\ \frac{\cos^2(\vartheta)}{\cos^2(\alpha)}\ \omega \tag{4.10}$$

I valori così ottenuti verranno usati successivamente, da parte del sistema di proiezione, per collocare correttamente l'immagine del pianeta in modo che risulti sovraimposto al robot.

Ulteriori raffinature Nonostante gli accorgimenti presi la tecnica di localizzazione sopra descritta rimane pur sempre una variante della navigazione stimata, soggetta quindi a progressive derive (per quanto contenute). Per ridurre quest'ultime, si è pensato di marcare il tracciato di ogni orbita con delle bande colorate, opportunamente spaziate, in numero proporzionale all'asse maggiore dell'orbita stessa.

Ogni volta che il sistema ottico rileva una delle bande esso notifica il sistema atto a filtrare i dati del giroscopio il quale, basandosi sulla lettura corrente e sulle posizioni delle bande presenti nell'orbita corrente (note a priori), si riallinea alla più vicina. Ad esempio, se vi sono quattro bande, la lettura verrà riallineata al quarto di angolo giro più vicino.

Essendo le orbite ellittiche, le bande vengono posizionate con riferimento all'angolo di fase. Pertanto si rende necessario applicare l'equazione 4.8 per distanziarle in modo tale che le bande siano posizionate a intervalli regolari dell'orientamento del robot.

Modifiche al rilevatore del tracciato Per poter rilevare correttamente le bande colorate è necessario premettere un'analisi nel dominio HSV. L'immagine viene sogliata alla ricerca di segmenti del colore desiderato, quindi vengono applicati gli operatori di apertura e chiusura per ripulire il risultato. Dall'immagine binaria risultante, vengono estratti il numero di contorni (dopo aver filtrato eventuali "macchie" troppo piccole) e una maschera che li evidenzia. Tale maschera viene sommata a quella del tracciato in modo da riempire un possibile buco causato dalla banda stessa. Inoltre, se il numero di contorni è positivo, si segnala un riallineamento. Tali modifiche sono descritte nella Figura 4.10. Per scegliere lo specifico colore delle bande sono state effettuate alcune prove sperimentali tra i colori delle terne additiva e sottrattiva, in seguito alle quali è stato scelto il color magenta in quanto il più facile da rilevare anche in condizioni di illuminazione scarsa.

Modifiche al filtro Il primo passo è dedurre la posizione della banda più prossima. La lista delle posizioni delle bande viene passata come argomento alla seguente funzione:

```
1  def find_best_match(markers_list: List[float], val: float) -> float:
2      match = 0.0
3      dst = float("inf")
4
5      # Detect how many cycles of 360deg have passed (where [0, 360[ is cycle 0)
6      cycle_count = abs(val // 360 + (1.0 if val < 0.0 else 0.0))
7
8      # The markers list is defined between [0, 360]
9      # It must be translated to the current cycle
10     translated_markers_list = [(m + cycle_count * 360) * (-1 if val < 0.0
           else 1) for m in markers_list]
11
12     for marker in translated_markers_list:
13         marker_dst = abs(marker - val)
14         if marker_dst < dst:
15             dst = marker_dst
16             match = marker
```

```
void detectColor(cv::Mat& frame, cv::Scalar colorMin, cv::Scalar colorMax,
    std::vector<std::vector<cv::Point>>& contours, cv::Mat& mask) {
        std::vector<std::vector<cv::Point>> detectedContours;
        cv::Mat processed = frame.clone();
        cv::cvtColor(processed, processed, cv::COLOR_BGR2HSV);

        cv::GaussianBlur(processed, processed, cv::Size(3, 3), 0);
        cv::inRange(processed, colorMin, colorMax, mask);

        cv::Mat kernel = cv::getStructuringElement(cv::MORPH_RECT,
            cv::Size(3, 3));
        cv::morphologyEx(mask, mask, cv::MORPH_OPEN, kernel);
        cv::morphologyEx(mask, mask, cv::MORPH_CLOSE, kernel);
        cv::findContours(mask.clone(), detectedContours, 1,
            cv::CHAIN_APPROX_NONE);

        // Filter noise by selecting only relevant contours
        contours.clear();
        std::copy_if(detectedContours.begin(), detectedContours.end(),
            std::back_inserter(contours),
                [](const std::vector<cv::Point>& p) {
                       return cv::contourArea(p) > 200;
                }
        );
}
...

std::vector<std::vector<cv::Point>> colorContours;
cv::Mat mask = cv::Mat::zeros(processed.rows, processed.cols, CV_8U);

detectColor(processed, cv::Scalar(lowH, lowS, lowV), cv::Scalar(highH, highS,
    highV), colorContours, mask);
int colorCount = static_cast<int>(colorContours.size());

// Set to 0 colore bands
cv::bitwise_not(mask, mask);

// Set to 0 all color values. This will include them in the thresholding
cv::cvtColor(processed, processed, cv::COLOR_BGR2GRAY);
cv::bitwise_and(processed, mask, processed);
```

Figura 4.10: [C++] Codice per la determinazione delle bande di allineamento.

```
17
18      return match
```

dove il simbolo `//` corrisponde alla divisione intera. Dall'angolo d'orientamento trovato si può ricavare l'errore

$$error = realAlpha - currentAlpha \tag{4.11}$$

Per evitare bruschi salti dovuti al riallineamento, questa quantità viene divisa in un numero arbitrario di parti (ad esempio, dieci) e aggiunta gradualmente alle misurazioni successive fino a compensarlo completamente.

4.2.2.8 Identificazione delle orbite

Per riconoscere su quale orbita è stato posizionato il robot si è fatto uso di un tipo di codici spesso usato in applicazioni di realtà aumentata, ossia quelli realizzati ed utilizzati dalla libreria ArUco ([11], [12] e [13]). Si tratta di codici bidimensionali di forma quadrata, più piccoli di un codice QR, caratterizzati da un ID univoco all'interno di un dizionario di dimensione fissa (tra i 50 e i 1000 ID univoci in base al dizionario).

In questo caso, dovendo distinguere solamente otto elementi distinti si è optato per un dizionario da 50 elementi, ossia della minore dimensione disponibile. In questo modo i marcatori risultanti presentano una densità minore di informazioni (ossia, vi è una minore variabilità del pattern bianco/nero) ed è possibile stamparli, a parità di potere risolutivo della telecamera, di dimensione minore. In questo modo l'impatto estetico viene ridotto ed è più facile riconoscere il marcatore all'interno del ridotto campo visivo della telecamera. A seguito di alcune prove è stato prodotto un set di otto codici di lato 1.5 cm posizionati appena sopra l'inizio di ogni orbita, in modo da venire inquadrati dal robot quando esso viene posizionato secondo le indicazioni di cui in Sezione 4.2.2.7. La Figura 4.11 è un esempio preso da tale set.

Figura 4.11: Esempio di codice ArUco.

Per facilitare il corretto posizionamento del robot sono state evidenziate, mediante un apposito simbolo esclusivamente proiettato (e pertanto non presente sul tracciato stampato), l'area che

il robot deve occupare ed il verso in cui deve essere rivolto, specificato da una linea verticale da allineare col robot stesso. Tale simbolo è visibile nella Figura 4.12 in due configurazioni: nella prima, a sinistra, il simbolo è "inattivo", ed il robot non è stato piazzato su di esso; nella seconda, a destra, il simbolo comincia ad irradiare luce per indicare che il robot è stato piazzato correttamente ed il marcatore rilevato corrisponde a quello dell'orbita del pianeta scelto per la sessione dell'esperienza.

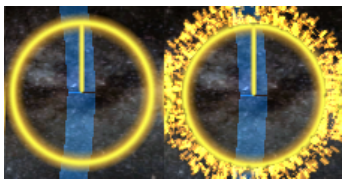


Figura 4.12: Il simbolo che evidenzia la posizione in cui collocare il robot, "inattivo" (sinistra) ed "attivo" (destra).

4.2.2.9 Rilevamento delle tessere

Il tipo di tessere scelte per le curiosità è basato sul circuito MFRC522, un ricetrasmettitore ampiamente diffuso in questo tipo di progetti. Per lo sfruttamento del modulo hardware è stata utilizzata una libreria *open source* [14], con modifiche minori circa la gestione degli errori. In una fase preparatoria su ognuna delle tessere è stata scritta una breve stringa di caratteri identificativa di una delle caratteristiche dei vari pianeti. Durante l'attività con le tessere un programma dedicato itera il seguente semplice ciclo:

1. leggi i dati dalla prossima tessera riconosciuta;
2. se la tessera è già stata riconosciuta, continua;
3. se la tessera non appartiene al pianeta, fai lampeggiare un led rosso (feedback negativo);
4. se la tessera appartiene al pianeta, fai lampeggiare un led verde, scrivi in uscita la stringa identificativa ed aggiungi la tessera alla lista delle tessere riconosciute;
5. se sono state riconosciute tre tessere, esci;
6. altrimenti attendi sei secondi e continua.

Quest'ultimo passaggio è necessario per consentire al sistema di proiezione di riprodurre l'animazione corrispondente e ad impedire che gli studenti facciano calca sul robot tentando una tessera dopo l'altra in rapida successione.

4.3 Implementazione del sistema di proiezione

Per la visualizzazione grafica è stata realizzata una ricostruzione simbolica del Sistema solare, visibile in Figura 4.14, con le orbite ben evidenziate. Vengono mostrate le diverse configurazioni dell'attività, in funzione della fase e dello stato di connessione. Il pianeta selezionato per l'esperienza, dovendo adattarsi alle dimensioni del robot, viene volutamente scalato (nel caso d'esempio, si tratta della Terra).

Da un punto di vista software, le classi più importanti sono evidenziate dal diagramma 4.13, alle quali si aggiungono gli script per gestire un'interfaccia grafica ausiliaria (non mostrata in 4.14 e usata solo a scopo di diagnostica; per maggiori informazioni si veda la Sezione A.4) e alcuni metodi di utilità generici. Di queste, la classe `Broadcaster` è un semplice script che implementa la funzionalità discussa in 4.1.2, e pertanto non verrà ulteriormente approfondita.

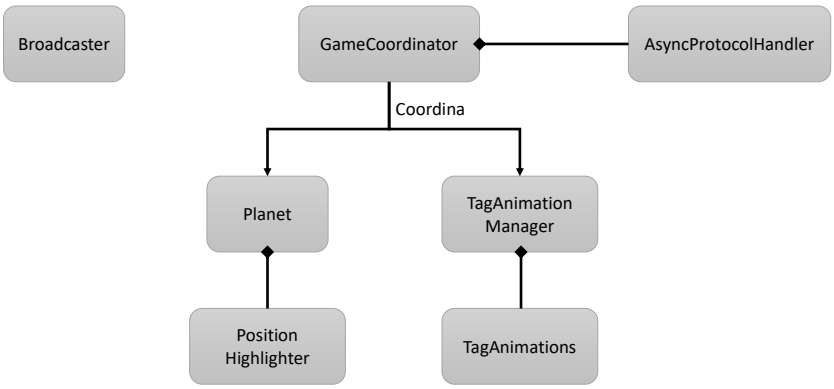


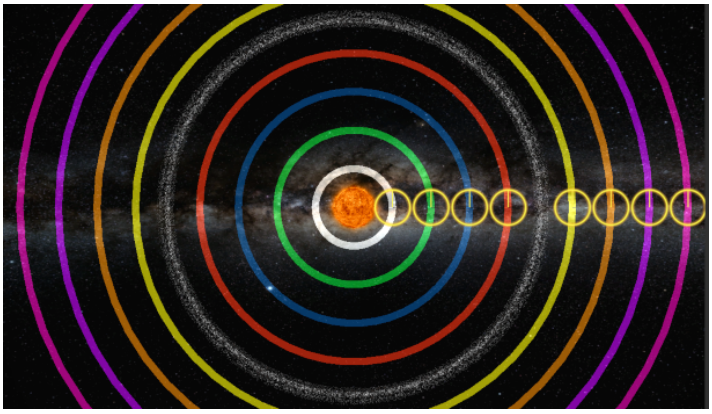
Figura 4.13: Diagramma delle classi principali del server.

4.3.1 La classe `AsyncProtocolHandler`

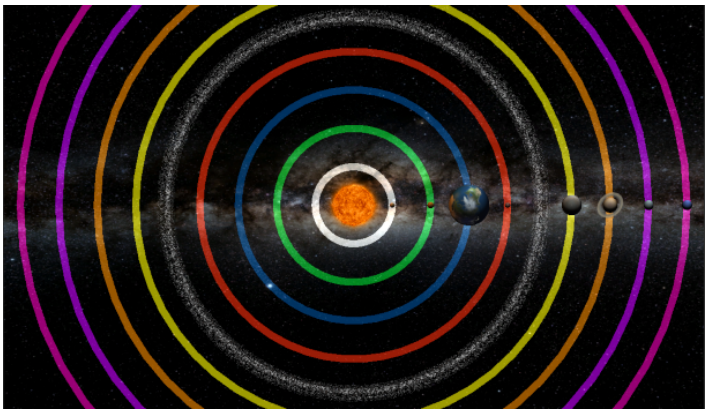
La classe che gestisce la comunicazione con i client. Sfrutta il paradigma `async/await`, nella formulazione del linguaggio C#, per implementare un server TCP ad eventi. Eventuali sottoscrittori agli eventi di questa classe vengono notificati in caso di:

- connessione del client;

(a) Sistema parzialmente connesso. L'interfaccia mostra quali client sono connessi.



(b) Sistema in fase di posizionamento di ricerca del pianeta.



(c) Sistema in fase di attività delle curiosità o moto (Terra evidenziata).

Figura 4.14: Vista del Sistema solare in varie configurazioni.

- ricezione di un messaggio dal client;
- invio di un messaggio verso il client;
- disconnessione del client.

Il metodo `Connect` permette di avviare il server tramite un `Task` asincrono su una porta scelta in fase di costruzione. Una volta avvenuta la connessione, il ciclo di lettura deve essere avviato in maniera esplicita mediante il metodo `StartReadingFromClient`. Da quel momento è possibile inoltre inviare messaggi al client tramite il metodo `WriteToClient(string)`, richiedere la chiusura mediante il metodo `Halt`, e mantenere attiva la connessione mediante il metodo `KeepAlive`. Quest'ultimo dev'essere chiamato periodicamente da uno script esterno per garantire la resistenza ai timeout come descritto nella Sezione 4.1.4; tale procedura purtroppo si rende necessaria a causa di un dettaglio implementativo del metodo `ReadLineAsync` della classe `StreamWriter` del framework .NET, il quale maschera eventuali timeout del socket sottostante.

Quando si riceve un messaggio dal client connesso, esso viene prima di tutto analizzato per verificarne la sintassi. Qualora il comando sia valido, esso viene diviso nelle sue parti (nome del comando ed eventuali argomenti) e inviato ai sottoscrittori. In caso contrario, il server restituisce `PROTOCOL_EXCEPTION` al mittente.

4.3.2 La classe `Planet`

Questo è lo script che gestisce le rappresentazioni dei pianeti. Richiede i seguenti parametri:

- raggio maggiore dell'orbita;
- eccentricità;
- periodo di rotazione;
- periodo di rivoluzione.

Per semplicità di rappresentazione, il Sole è stato posto al centro dell'orbita piuttosto che in uno dei due fuochi. Il tracciato dell'orbita è costruito proceduralmente, a partire dalla sua rappresentazione parametrica:

$$\begin{bmatrix} A\cos\vartheta \\ B\sin\vartheta \end{bmatrix} \tag{4.12}$$

La medesima equazione governa la posizione del pianeta, comandato in fase. Ognuno dei pianeti ruota su sé stesso sulla base del periodo di rotazione fornito. La rivoluzione attorno al Sole non avviene fino alla fase di moto dei pianeti. Per il pianeta scelto per l'esperienza, la fase è dettata dalle informazioni fornite dal robot secondo la metodologia descritta in 4.2.2.7. Tali informazioni vengono passate attraverso il metodo `UpdateRobotTracking(float, float, float)`, che richiede l'orientamento del robot, la sua velocità angolare e la posizione (angolare) dell'ultima banda di allineamento rilevata. La velocità viene utilizzata per estrapolare i valori mancanti tra un aggiornamento e il successivo; poiché tale valore tende ad avere occasionali picchi (nonostante la presenza del filtro di Kalman), viene applicato un filtro mediano a tre campioni per rendere più fluido lo spostamento. Per quanto concerne i pianeti restanti, invece, la rapidità di spostamento è dettata dal periodo di rivoluzione impostato.

Ogni pianeta è collegato a un `PositionHighlighter`, ossia un marchio circolare come quello illustrato dalla Figura 4.12; esponendo i metodi `SetHighlightMarkerVisbile(bool)`, `StartHighlight` e `StopHighlight` è possibile gestirne la visualizzazione.

4.3.3 La classe `TagAnimationManager`

Questa classe permette di gestire le animazioni legate ad ognuna delle caratteristiche del pianeta. Tali animazioni vengono implementate mediante un oggetto il cui nome segue il formato:

```
<NomePianeta>Animator<NomeTag>
```

ad esempio `EarthAnimatorH2O`. Ad esso si aggiunge uno script di tipo `TagAnimation`. Tale script espone i metodi `Play` e `Stop`; quando vengono eseguiti, cercano ed attivano o disattivano i seguenti componenti, se presenti.

- Un `ParticleSystem` [15]. Solitamente viene usato per effetti sovrapposti all'orbita del pianeta; ad esempio una delle curiosità del pianeta Terra simula la caduta di gocce d'acqua radialmente lungo l'orbita.
- Un `Animator` [16]. Viene usato per definire animazioni generiche, come trasformazioni geometriche, immagini animate e simili. L'avvio avviene mediante un trigger predefinito (`Play`).
- Gli script `AudioFade` e `SpriteFade`. Applicano un effetto di *fadein-fadeout* rispettivamente a una componente audio o un'immagine.

L'utilizzo è piuttosto elementare, tramite il metodo `StartTagAnimation(PlanetName, Curiosity)` che richiede il nome del pianeta e l'identificativo della tessera. Esso esegue le seguenti operazioni:

1. costruisci il nome identificativo dell'animatore;
2. ricerca, tra gli oggetti segnati come `TagAnimation`, uno con quel nome;
3. se presente:
   (a) mostra uno sfondo bianco semitrasparente (per rendere più evidenti gli effetti);
   (b) interrompi eventuali effetti già attivi;
   (c) esegui il nuovo effetto.
4. altrimenti genera un errore.

4.3.4 La classe `GameCoordinator`

Questa è la classe che coordina l'attività delle altre, eseguendo le operazioni di interfacciamento con la rete e gestione del protocollo. All'avvio dell'attività inizializza due istanze di `AsyncProtocolHandler`, una sulla porta TCP 6600 (il server per il Robot) e una sulla porta 6603 (il server per l'Applicazione di controllo). Inizializza inoltre le strutture che tracciano lo stato dell'attività, del quale è il garante, e sottoscrive gli eventi dei due server.

4.3.4.1 `ClientConnected`

Quando uno dei due client si connette, la stringa di stato (vedi 4.1.3) viene inviata come primo messaggio. Se la controparte (ossia il robot durante la connessione dell'applicazione, o viceversa) è connessa, allora viene notificata attraverso il comando `RESUME`, e la schermata di attesa (visibile in 4.14) viene nascosta; altrimenti la stessa schermata viene semplicemente aggiornata per notificare il nuovo stato di connessione. Dopodiché vengono ripristinate le condizioni grafiche per lo stato corrente (vengono mostrati o nascosti i marcatori ed i pianeti) e riprende il ciclo di gestione dei messaggi.

4.3.4.2 `MessageReceived` - Messaggi accettati dal robot

`KEEPALIVE`, `IMALIVE`

Messaggi di mantenimento della connessione, vedere la Sezione 4.1.4.

`PLANET_OK`, `PLANET_KO`

Permesso solo durante la fase di `SCANNING`. Mostra / nasconde il marcatore per il pianeta corrente, inoltra il valore all'applicazione.

`TAG <CodiceTag>`

Permesso solo durante la fase di `QUIZ`, se la lista delle tessere è ancora incompleta e `CodiceTag` è un codice identificativo per una delle tessere del pianeta scelto. Aggiunge il codice alla lista delle tessere riconosciute, esegue la relativa animazione tramite `TagAnimationManager` ed inoltra il valore all'applicazione.

`ROBOT_POSITION <Orientamento> <Velocità> <UltimoAllineamento>`

Richiama il metodo `Planet#UpdateRobotTracking()` sul pianeta scelto, aggiornandone la posizione usando le informazioni fornite dal robot. Invia nuovamente il comando `ROBOT_POSITION` al robot in modo da proseguire il ciclo di aggiornamento.

4.3.4.3 `MessageReceived` - Messaggi accettati dall'applicazione

`KEEPALIVE`, `IMALIVE`

Messaggi di mantenimento della connessione, vedere la Sezione 4.1.4.

`SCAN_PLANET <Pianeta>`

Permesso solo durante la fase di `SETUPPING`, oppure durante la fase di `QUIZ` se la lista delle tessere è completa (in tal caso, l'argomento viene ignorato). Inizializza la fase di `SCANNING` col pianeta scelto, laddove `<Pianeta>` dev'essere un nome valido di pianeta. Nasconde la visualizzazione dei pianeti e mostra i marcatori delle posizioni iniziali. Inoltra il messaggio al robot.

`QUIZ_START`

Permesso solo durante la fase di `SCANNING`, se la lista delle tessere è incompleta. Pulisce la lista delle tessere e rende visibili i pianeti, nascondendo i marcatori. Notifica il pianeta scelto di cominciare a tracciare il robot, il quale assume quindi le dimensioni necessarie. Inoltra il comando al robot.

`MOVE_START`

Permesso solo durante la fase di `SCANNING` se la lista delle tessere è completa. Nasconde i marcatori e mostra nuovamente i pianeti, abilitandone lo stato di moto. Inoltra il comando al robot, insieme al comando `ROBOT_POSITION` per iniziare il ciclo di aggiornamento della posizione.

`ROBOT_PAUSE`, `ROBOT_RESUME`

Permesso solo durante la fase di moto dei pianeti. Viene inoltrato direttamente al robot per pausarne / riprenderne il movimento.

`ABORT` Interrompe la comunicazione con entrambi i client, ripristina lo stato della partita.

4.3.4.4 `ClientDisconnected`

Quando uno dei due client si disconnette, lo stato viene messo in pausa, nascondendo i marcatori dei pianeti e resettando il loro stato. Se uno dei due client è ancora connesso, viene inviato il comando `HOLD`. La schermata di attesa della Figura 4.14 ritorna visibile, in maniera coerente con lo stato attuale di connessione.

4.4 Implementazione del sistema di controllo

L'amministrazione ed il tracciamento dello stato dell'attività avvengono mediante un'applicazione per dispositivi mobili, sviluppata per sistemi Android. Su di essa vengono visualizzate, di schermata in schermata, tutte le informazioni relative alla fase corrente, oltre a dei pannelli di attesa durante i momenti di connessione.

Dall'applicazione il tutor può scegliere quale pianeta sarà oggetto della sessione, e decidere quando passare da una fase all'altra. Tutte le diverse schermate di cui è composta sono visualizzate in maniera riassuntiva nella Figura 4.15.

4.4.1 Struttura generale

Per ragioni stilistiche l'applicazione è stata sviluppata su una singola `Activity`, delegando invece la dinamicità dell'interfaccia all'utilizzo di `Fragment` annidati, uno specifico per ogni fase dell'attività. In tal modo si riesce a dare un senso di continuità all'interfaccia nel suo complesso, che non viene spezzata dalle transizioni tipiche del passaggio da un'`Activity` all'altra. D'altro canto

(a) Schermata iniziale. (b) Connessione. (c) Attesa del robot. (d) Scelta del pianeta.

(e) Ricerca del pianeta. (f) Quiz. (g) Preparazione moto. (h) Moto dei pianeti.

Figura 4.15: Schermate dell'applicazione di controllo.

l'esperienza ha uno svolgimento lineare e non avrebbe senso sfruttare la pila delle `Activity` di Android per consentire il ritorno all'indietro.

La gerarchia con cui sono distribuiti i `Fragment` è mostrata in Figura 4.16. Il layout base dell'`Activity` mostra uno sfondo comune, visibile nelle immagini della Figura 4.15, sul quale viene inserito uno dei seguenti `Fragment`.

`MainFragment`

: Schermata introduttiva (Figura 4.15a).

`ConnectingFragment`

: Schermata di connessione al server e attesa del robot (Figure 4.15b e 4.15c).

`ChoosePlanetFragment`

: Fa scegliere il pianeta per la sessione da una galleria orizzontale (Figura 4.15d).

`GameFragment`

Contenitore per gestire le fasi dell'esperienza. Mostra un pulsante per interrompere la partita e ricominciare, l'immagine ed il nome del pianeta, ed uno dei seguenti sotto-`Fragment`:

`ScanFragment`

Monitora il corretto posizionamento del robot sull'orbita. Mostra un pulsante per procedere solo quando riceve conferma dal server (Figure 4.15e e 4.15g).

`QuizFragment`

Monitora quali sono le curiosità che gli studenti hanno correttamente indovinato, relativamente al pianeta corrente. Permette di procedere solo quando tutte e tre sono state accettate (Figura 4.15f).

`RobotControlFragment`

Permette di fermare e rimettere in moto il robot, e di interrompere l'esperienza (Figura 4.15h).

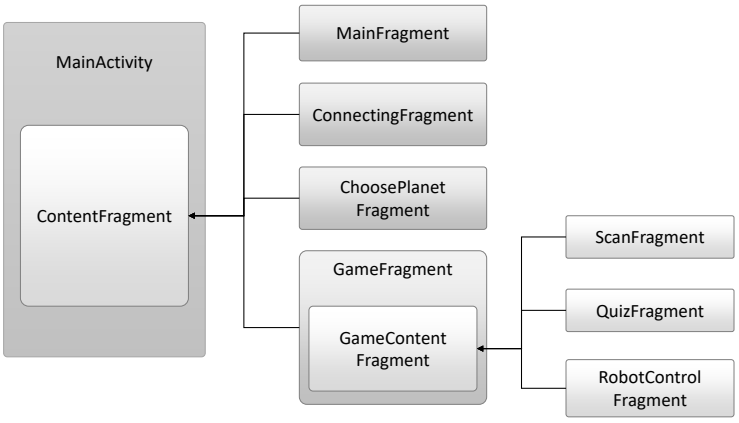


Figura 4.16: Suddivisione dei `Fragment` dell'applicazione di controllo.

4.4.2 Comunicazione di rete

In maniera affine a quanto svolto per il robot ed il server, anche la parte di rete dell'applicazione sfrutta il paradigma della programmazione asincrona mediante le coroutine di Kotlin, le quali fanno parte di una libreria ancora sperimentale ma perfettamente funzionale. Nello specifico, tutte le operazioni (bloccanti) che richiedono l'interazione col server vengono descritte per mezzo di funzioni `suspend` (la parola chiave di Kotlin che identifica le coroutine) che vengono eseguite su un thread preso dalla `CommonPool` per mezzo della funzione `async`. Quando il risultato ottenuto dalle operazioni di rete ha bisogno di essere elaborato e mostrato nell'interfaccia grafica, un'apposita procedura di elaborazione viene invocata direttamente sul thread principale dell'applicazione, in maniera semplice ed elegante mediante la funzione di supporto `launch(UI)`.

Per semplificare la gestione e la cancellazione di tali operazioni ognuna di esse viene imparentata a un `Job`, la classe che identifica operazioni asincrone. Quando bisogna interrompere le operazioni in corso (ad esempio, per disconnettersi dal server a seguito di un errore o della chiusura dell'applicazione) è sufficiente cancellare il `Job` padre per invocare la cancellazione di tutte le operazioni figlie.

4.4.3 Descrizione delle operazioni principali

Inizializzazione All'apertura all'utente viene mostrata la schermata iniziale (4.15a) e un singolo pulsante per iniziare. Alla pressione di tale pulsante viene avviata una coroutine che esegue le seguenti operazioni:

- In maniera identica al robot, resta in attesa sulla porta UDP 6600 per la stringa identificativa del server; dal datagramma ricevuto estrae l'indirizzo IP del server.
- Stabilisce una connessione TCP sulla porta 6603 del server, sulla quale avverranno le successive comunicazioni.
- Ottiene lo stato corrente dal server (vedi 4.1.3), inizializza la struttura dati corrispondente ed aggiorna la UI.
- Avvia (sempre nel contesto della coroutine) un ciclo di lettura dal server per le operazioni successive.

Nel caso in cui si verifichi un'eccezione, essa viene catturata dalla coroutine, la quale come ultima operazione verifica se la causa sia la chiusura dell'applicazione (mediante il metodo `isFinishing()`). Se non è questo il caso, essa invoca nuovamente, sul thread principale dell'applicazione, la procedura di inizializzazione.

Scelta del pianeta La schermata 4.15d viene mostrata una galleria a scorrimento orizzontale con l'elenco dei pianeti del Sistema solare. L'utente può scorrerla e scegliere quali di essi sarà oggetto della sessione corrente. Una volta effettuata la scelta, l'applicazione invia al server il comando `SCAN_PLANET <PlanetName>` e passa alla fase successiva, ossia il posizionamento del robot.

Posizionamento L'applicazione attende che il robot venga collocato correttamente sul marcatore corrispondente all'orbita del pianeta scelto durante la fase precedente. Il corretto posizionamento o meno viene segnalato periodicamente dal robot, e inoltrato all'applicazione dal server. Durante questa fase, i messaggi `PLANET_OK` e `PLANET_KO` servono, rispettivamente, ad attivare o disattivare il pulsante che permette di procedere.

Quiz All'utente viene mostrata la lista delle tre curiosità (con la loro descrizione estesa) corrispondenti al pianeta scelto (4.15d). L'applicazione resta in attesa del comando `TAG <TagId>` da parte del server; ogni qualvolta viene ricevuto, mette una spunta sulla relativa curiosità. Quando tutte e tre sono state trovate viene mostrato un pulsante per procedere: si ritorna alla fase di posizionamento e successivamente sarà possibile passare a quella di gestione del moto.

Moto dei pianeti In questa fase vengono mostrati solamente due pulsanti, "Pausa il moto" e "Termina moto" (4.15h). Il primo invia al server il comando `ROBOT_PAUSE` e si alterna con il pulsante complementare "Riprendi il moto", che invece invia il comando `ROBOT_RESUME`, permettendo di fermare e far ripartire il robot. Il secondo invece funziona analogamente alla croce mostrata nel `GameFragment` ed invia al server il messaggio `ABORT`, interrompendo l'esperienza.

Altri comandi accettati e inviati dall'applicazione

`KEEPALIVE`, `IMALIVE`

Come per le altre parti del sistema, gestiscono i timeout.

`HOLD`, `RESUME`

Analogamente, gestiscono la disconnessione e la riconnessione del robot. La schermata 4.15c viene mostrata fra la ricezione del primo e quella del secondo.

`ABORT` Viene ricevuto di riflesso dal server quando si invia `ABORT` mediante la croce in cima al `GameFragment` o il pulsante "Termina moto". Riavvia l'`Activity` principale dell'applicazione e termina quella corrente.

Capitolo 5

Risultati

Terminata la realizzazione del prototipo, il sistema è stato scalato ed installato presso Xké in una sala adibita. Il proiettore richiesto (un Optoma modello WU515ST) è stato installato a soffitto; per ragioni di approvvigionamento, non è stato possibile ottenere la copertura inizialmente prevista, pertanto l'area dedicata all'attività è stata scalata in modo da rientrare entro 3.2 m. Sul pavimento è stato collocato un adesivo professionale, ricoperto da una patina in PVC. Il colore scelto per lo sfondo è stato campionato da quello del pavimento circostante, mentre i tracciati per il robot sono stati realizzati seguendo le ellitticità reali delle orbite dei pianeti. La Figura 5.1 mostra il pattern dell'adesivo.

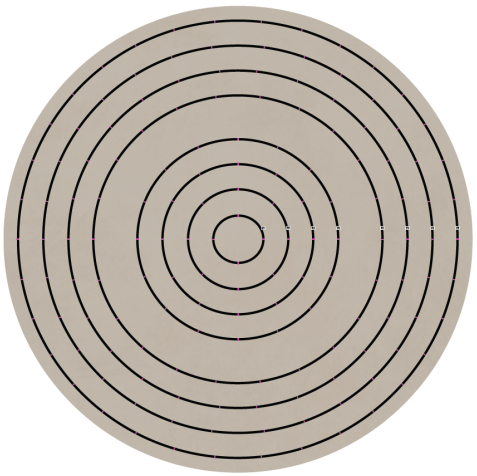


Figura 5.1: Immagine del tracciato stampato.

Il software di proiezione e l'applicazione di controllo sono stati installati su di un PC ed un tablet di proprietà del Laboratorio, sebbene alcuni aggiustamenti si siano resi necessari per supportare la vecchia versione di Android presente su quest'ultimo. Come da progettazione, il sistema è stato collegato ad un access point WiFi isolato rispetto al resto della rete del Laboratorio. Nella Figura 5.2 è possibile vedere il sistema collegato e funzionante all'interno della stanza di Xké.

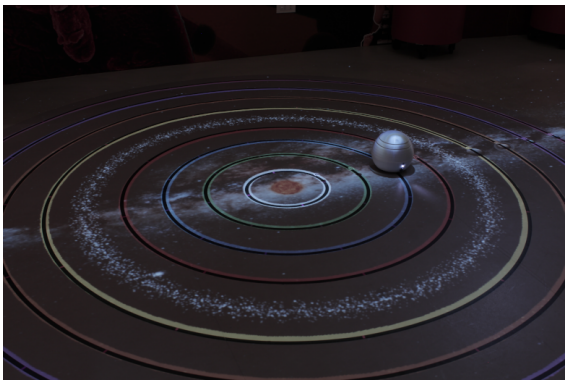


Figura 5.2: Foto del sistema installato nel Laboratorio.

A seguito delle prove fatte sul campo sono emerse delle difficoltà in merito al numero di segmenti disposti per la ricalibrazione del giroscopio. L'idea di mantenere costante la distanza lineare tra di essi si è rivelata in pratica fragile o addirittura dannosa. Essendo infatti le misure basate sull'orientamento del robot, e non sulla sua posizione spaziale, se due segmenti consecutivi hanno orientamenti del robot corrispondenti simili il software di riallineamento rischia di confondere le due posizioni, causando un disallineamento della proiezione. Si è dunque deciso di ridurre il numero di segmenti soprattutto nelle orbite maggiori, limitandone il numero a non più di 13 per orbita (con uno scarto d'angolo di circa 28°).

5.1 Metodologia di sperimentazione

Dopo un iniziale periodo di training informale con i tutor del Laboratorio sono state invitate a prender parte alla nuova esperienza cinque classi delle scuole elementari tra gli 8 e i 10 anni. Durante il corso di queste sessioni gli studenti sono stati osservati nei loro comportamenti per valutare reazioni tipiche e dinamiche sociali emergenti nel nuovo percorso. Al termine di questo periodo di sperimentazione, a tutti i tutor partecipanti è stato proposto un questionario basato

sulla System Usability Scale [17] (Tabella 5.1) per valutare il grado di usabilità del sistema progettato. Esso è stato inoltre affiancato da un questionario personalizzato basato su alcuni lavori esistenti ([18], [19] e [20]) volto a valutare il grado di coinvolgimento, divertimento e spirito collaborativo suscitato dalla nuova esperienza (Tabelle 5.3, 5.4 e 5.5). Per ottenere un confronto con l'esperienza precedente, queste tre parti sono state proposte anche ai tutor che hanno condotto le attività nella versione originaria del percorso didattico *1, 2, 3... Stelle!*. Infine sono state aggiunte due ulteriori Sezioni (Tabelle 5.6 e 5.2), non basate su letteratura, che espandono la ricerca circa l'usabilità del software e l'efficacia del mezzo comunicativo, e due domande generiche a risposta aperta sugli aspetti positivi e negativi di entrambe le versioni (vecchia e nuova, Tabella 5.7). Tutte le domande sono state valutate in una scala da 1 (per niente d'accordo) a 5 (completamente d'accordo), e possono avere accezione "positiva" (domande indicate con una "P" nelle Tabelle) o "negativa" (domande indicate con una "N"). Fanno eccezione le domande contrassegnate da un asterisco* (domande aperte). I risultati derivanti dall'elaborazione dei questionari non saranno presenti in questo documento, ma verranno esposti con dettaglio nella tesi ad essa complementare svolta dalla tesista Lorenza Abbate insieme alla quale questo lavoro è stato realizzato. Nella seguente Sezione verranno invece esposte alcune osservazioni riguardanti alcuni comportamenti frequenti riscontrati negli studenti delle classi coinvolte.

5.2 Comportamenti osservati

Trattandosi di un'attività assolutamente sperimentale, le sessioni svolte durante le prime due settimane sono servite anche per permettere ai tutor di valutare, sulla base delle reazioni degli studenti, se e come declinare diversamente l'esperienza offerta dal sistema rispetto a quanto previsto in fase di progettazione. Un dettaglio che è emerso a tal proposito è il generale disordine, e il malumore tra gli studenti generato dalla competizione per l'interazione con il robot (ad esempio, al momento di posizionarlo lungo l'orbita corretta o di passare la carta scelta su di esso), che per natura progettuale costituisce un collo di bottiglia ostacolo alla completa partecipazione della classe.

Al fine di prevenire queste situazioni di conflitto i tutor hanno preferito, dopo le prime volte, porsi come intermediari col sistema, effettuando essi stessi le interazioni col robot dopo aver interpellato i gruppi circa le loro scelte, spostando il focus sulla scelta corretta delle carte; sebbene ciò abbia ridotto significativamente il livello di interazione tra la classe ed il robot, è servito ad eliminare i contrasti.

1	Penso che mi piacerebbe utilizzare spesso questo sistema.	P
2	Ho trovato il sistema complesso senza che ce ne fosse bisogno.	N
3	Ho trovato il sistema molto semplice da utilizzare.	P
4	Penso che avrei bisogno del supporto di una persona già in grado di utilizzare questo sistema.	N
5	Ho trovato le varie parti del sistema ben connesse.	P
6	Ho trovato che le varie parti del sistema fossero incoerenti tra di loro.	N
7	Penso che la maggior parte dei tutor potrebbe imparare ad utilizzare il sistema facilmente.	P
8	Ho trovato il sistema molto macchinoso da utilizzare.	N
9	Mi sono sentito sicuro nell'utilizzare il sistema.	P
10	Ho avuto bisogno di apprendere molte nozioni prima di riuscire ad utilizzare al meglio il sistema.	N

Tabella 5.1: Usabilità del software (System Usability Scale).

1	Il sistema è intuitivo nell'apprendimento per la conduzione.	P
2	Il sistema è semplice da avviare.	P
3	Il sistema è semplice da condurre.	P
4	Il sistema consente il rispetto dei tempi dedicati all'azione.	P
5	Il sistema non mi ha richiesto un particolare sforzo cognitivo.	P
6	Il sistema non mi ha richiesto un particolare sforzo fisico.	P
7	Il sistema arricchisce le mie competenze come tutor.	P
8	Il sistema mi diverte nella conduzione.	P
9	Quale parte del sistema presenta criticità?	*
10	Quale parte del sistema trovi maggiormente efficace?	*

Tabella 5.2: Efficienza del sistema.

1	Il feedback dato agli studenti durante l'attività è stato efficace.	P
2	Gli studenti erano proattivi nei confronti dell'esperienza.	P
3	Gli aspetti visivi dell'attività hanno catturato l'attenzione degli studenti.	P
4	La qualità degli elementi visivi ha interferito con lo svolgimento dell'attività.	N
5	Gli studenti partecipavano attivamente ed erano interessati all'argomento.	P
6	Le istruzioni date dai tutor agli studenti erano facili da seguire.	P
7	Gli studenti erano confusi sugli obiettivi da raggiungere.	N
8	L'attività è semplice da spiegare agli studenti.	P
9	Le fasi dell'azione sono facilmente memorizzabili dagli studenti.	P
10	Quali parti potrebbero essere maggiormente coinvolgenti, o aumentate nel tempo?	*

Tabella 5.3: Coinvolgimento ed attenzione degli studenti.

Gli studenti erano...		
1	Soddisfatti.	P
2	Divertiti.	P
3	Occupati dall'attività.	P
4	Contenti.	P
5	Di cattivo umore.	N
6	Distratti.	N
7	In grado di svolgere le attività.	P
8	In difficoltà.	N
9	Annoiati.	N
10	In grado di svolgere rapidamente i task assegnati.	P
11	Sotto pressione.	N
12	Sorpresi.	P
13	Concentrati.	P
14	Incuriositi.	P
15	Avrebbero voluto un'azione più lunga.	P
16	Avrebbero voluto una ripetizione dell'azione.	P
17	Quali suggerimenti daresti per migliorare il divertimento degli studenti?	*

Tabella 5.4: Divertimento degli studenti.

1	Le dinamiche di gruppo si risolvevano nella collaborazione tra gli studenti.	P
2	Gli studenti sono stati tutti partecipi dell'esperienza.	P
3	Si sono verificati fenomeni di conflitto o contrasto all'interno della squadra.	N
4	Gli studenti erano soddisfatti nel collaborare tra di loro.	P
5	Gli studenti hanno rispettato i tempi nell'esecuzione delle azioni.	P
6	L'organizzazione dello spazio è idonea alla partecipazione in gruppo.	P
7	Gli studenti hanno sviluppato un senso equilibrato di competizione.	P
8	Quali accorgimenti andrebbero presi per accrescere la collaborazione tra gli studenti?	*

Tabella 5.5: Collaborazione in gruppo tra studenti.

1	L'organizzazione dello spazio è idonea alla istruzione e svolgimento dell'azione.	P
2	Il robot è maneggevole.	P
3	Il robot si muove con velocità percepibile.	P
4	Il robot è percepito come pianeta.	P
5	Le proiezioni sono attraenti per luminosità.	P
6	Le proiezioni sono attraenti per informazione visiva.	P
7	L'azione finale (robot in movimento con proiezioni) è di sorpresa.	P
8	Quali accorgimenti andrebbero presi per migliorare la comunicazione?	*

Tabella 5.6: Efficacia della comunicazione.

1	Elenca tre aspetti positivi dell'esperienza.	*
2	Elenca tre aspetti negativi dell'esperienza.	*

Tabella 5.7: Domande generiche.

Nel seguito vengono elencati alcuni dei comportamenti osservati nel corso di queste sessioni nei confronti degli studenti.

- Gli studenti sono fortemente attratti dalla componente visiva del sistema di proiezione. In tutte le occasioni sono stati osservati tentativi di interagire in maniera fisica con l'immagine proiettata: ad esempio giocare a "catturare" i pianeti o a puntarli con le dita, o l'interazione con la mano per indicare un'orbita di interesse.
- L'attività con le tessere genera un forte senso di competizione tra le due squadre, le quali sono state osservate esultare per i propri successi o per gli insuccessi degli "avversari".
- Dopo un iniziale stupore nei confronti del movimento robot, la classe perde rapidamente interesse per esso, non mostrando particolare curiosità nei suoi confronti.

Il primo punto può esser visto come conseguenza della familiarità degli studenti, nativi digitali, con interfacce di tipo touch. Riguardo il livello di competizione raggiunto sembrerebbe superiore a quello aspettato, sebbene non sia facile, a questo livello, valutarne gli effetti positivi.

Circa l'ultima osservazione, potrebbe essere conseguenza di una serie di fattori; innanzitutto, il robot progettato ha un basso livello di autonomia. Ciò è non solo derivante dalla tipologia di tracking adottato, ma anche da una specifica richiesta da parte del Laboratorio, di non voler costruire un'esperienza passiva per gli studenti in cui il robot svolge la maggior parte del lavoro. In secondo luogo, occorre considerare la già citata mediazione decisa dai tutor rispetto all'interazione col robot; la combinazione di questi fattori può quindi aver portato a non far percepire il robot come un'entità intelligente agli studenti.

Capitolo 6

Conclusioni e sviluppi futuri

In questo lavoro è stato esplorato il campo della robotica applicato ai processi educativi e all'intrattenimento. Attraverso una collaborazione tra il Politecnico di Torino e *Xké? Il laboratorio della Curiosità* è stata seguita la realizzazione di un laboratorio didattico con elementi robotici, a partire dalla fase di progettazione, e via via attraverso la sua implementazione ed installazione in loco. L'obiettivo è stato portare alla concretizzazione questo nuovo laboratorio e attraverso una fase sperimentale verificare il miglioramento portato dalla riprogettazione e dall'introduzione dell'elemento robotico nel contesto didattico.

Dall'analisi delle reazioni delle classe invitate a prender parte a questo primo periodo di sperimentazione è emerso che la nuova formulazione è in grado di catturare l'attenzione degli studenti attraverso la presenza di elementi multimediali ed un feedback efficiente dato dal robot e dalla proiezione. La nuova esperienza, nel suo complesso, intrattiene gli studenti stimolandone l'apprendimento attraverso la competizione. Riguardo il ruolo del robot, l'effetto stupore dato dalla sua presenza è risultato inferiore alle aspettative, relegandone il ruolo percepito a quello di strumento di interazione col sistema e garante della correttezza delle tessere.

Un fattore fortemente limitante per l'esperienza proposta è dato dal sistema scelto per la localizzazione del robot. Affidarsi unicamente ad un giroscopio, per quanto di relativamente semplice implementazione, non consente una navigazione assoluta del robot, ma solo una fragile localizzazione lungo percorsi ellittici prestabiliti. Inoltre, questo metodo di localizzazione è estremamente sensibile a piccole oscillazioni del robot lungo il percorso, man mano che la distanza rispetto al centro aumenta; infatti se il robot non è perfettamente tangente al percorso l'errore d'orientamento risultante si traduce in un errore di posizione (lungo il percorso) che aumenta linearmente

con la distanza dal centro. Se tale errore d'orientamento (di pochi gradi) si traduce in una perdita di posizione trascurabile per orbite la cui distanza media è inferiore al metro, diventa invece significativo per orbite di ordine maggiore.

Trattandosi di un sistema di localizzazione relativo è stato necessario affidarsi a sistemi ottici per accompagnare la localizzazione del robot, sia per correggerne l'andamento in modo che segua il tracciato sia per determinare l'orbita corretta. Alla luce dell'esperienza condotta, per lavori futuri si potrebbero considerare altri meccanismi, possibilmente assoluti, per determinare la posizione del robot.

Da riconsiderare è anche la scelta di affidarsi a robot costruiti ad hoc. Nonostante sia innegabile il valore aggiunto alla flessibilità, la manutenzione diventa estremamente complessa in quanto in caso di guasto non è possibile ordinare un robot sostitutivo per intero, ma è necessario riparare quelli realizzati. Inoltre la reperibilità in futuro di pezzi sostitutivi potrebbe non essere garantita.

Appendice A

Montaggio e manuale d'uso

Dal momento che il sistema realizzato in questa tesi verrà utilizzato in maniera continuativa per attività didattiche, si rende necessario fornire ai fini di manutenzione un manuale completo che illustri le fasi di montaggio del robot, per poterne sostituire eventuali parti guaste, ed un manuale di ripristino e calibrazione, per permettere di sostituire memorie guaste o intervenire con facilità sui parametri che potrebbero non essere più validi a seguito di determinate sostituzioni.

Nel seguito si assumerà che questo manuale sia rivolto a un personale tecnico specializzato, familiare con l'ambiente Unix e la piattaforma Raspberry Pi. Tale preparazione non è necessaria per effettuare semplici operazioni di sostituzione delle parti, ma è fondamentale per poter intervenire direttamente sui parametri di configurazione del robot.

A.1 Elenco dei materiali

Un prospetto completo di tutti i materiali per la costruzione del robot è visionabile nella Tabella A.1. In essa è incluso il quantitativo della maggior parte delle parti di consumo, tra cui i distanziali per la separazione degli elementi e il numero e tipo di viti e bulloni necessari. Vengono inoltre suggeriti dei kit di assortimento disponibili, al momento della stesura di questa tesi, per l'acquisto su Amazon (la cui validità in futuro, ovviamente, non è garantita). Per i prodotti acquistati su Adafruit (un noto rivenditore di materiale elettronico per il fai-da-te) sono forniti non solo il link alla relativa pagina ma anche il codice prodotto, per riferimenti futuri.

Categoria	*Nome*	*Note*	*Qtà.*
Scheda	Raspberry Pi 3 B+		1
Camera	Raspberry Pi Camera Module V2		1
Alimentazione	Quimat Battery Pack	Cod. Prodotto: QKY68-UK. https://www.amazon.it/gp/product/B06XFPZKZW/	1
Memoria	Scheda micro SD (almeno 8 GiB)		1
Kit	Adafruit Round Robot chassis	Cod. Prodotto: 3216. https://www.adafruit.com/product/3216	1
Integrati	Adafruit L3GD20H Breakout	Cod. Prodotto: 1032. https://www.adafruit.com/product/1032	1
	Adafruit TB6612 Breakout	Cod. Prodotto: 2448. https://www.adafruit.com/product/2448	1
	Scheda RFID MFRC522		1
Motori	DFRobot FIT0450		2
PCB	Millefori 5 cm × 7 cm	Kit: APB1-1-AP-IT https://www.amazon.it/gp/product/B076S3XTSJ/	2
	Millefori 3 cm × 7 cm	Vedi kit Anpro APB1-1-AP-IT	1
Connettori	Header pin maschio, 7 totali (separabili)	Vedi kit Anpro APB1-1-AP-IT	1
	Header pin femmina, 8 totali (separabili)	Vedi kit Anpro APB1-1-AP-IT	1
	Morsettiera, due cavi.	Vedi kit Anpro APB1-1-AP-IT	2

Resistenze	220 Ω		1
	330 Ω		1
Distanziali	M/F, M2.5, 6 mm	Kit: EBOOT-BRASS SPACER-01 https://www.amazon.it/gp/product/B06XXV8RTR/	11
	F/F, M2.5, 20 mm	Vedi kit EBOOT-BRASS SPACER-01	2
	M/F, M2.5, 20 mm	Vedi kit EBOOT-BRASS SPACER-01	12
	M/F, M2.5, 15 mm	Vedi kit EBOOT-BRASS SPACER-01	4
	F/F, M2.5, 10 mm	Vedi kit EBOOT-BRASS SPACER-01	4
	F/F, M2.5, 15 mm	Vedi kit EBOOT-BRASS SPACER-01	3
Di consumo	Viti M2.5	Vedi kit EBOOT-BRASS SPACER-01	24
	Bulloni M2.5	Vedi kit EBOOT-BRASS SPACER-01	10
	Viti M2	Kit: https://www.amazon.it/gp/product/B01BWMZRCU/	11
	Bulloni M2	Vedi kit viti M2	15
	Pulsante normalmente aperto	Si consiglia un pulsante lungo con filettatura e dado per ilfissaggio, come quello visibile in FiguraA.5a.	1
	Cavi jumper	Vari assortiti, per lo più M/F. Vedere la Figura A.14 per un'idea delle connessioni necessarie.	–
	Cavo USB maschio	Da modificare per alimentare i motori. Vedere la Figura A.2.2.4.	1
	Cavo da USB a micro USB con interruttore.	Per collegare batteria e Raspberry.	1

	Extender micro USB maschio-femmina.	Per esporre il connettore di ricarica.	1
LED	Rosso, diffuso. Bianco, diffuso. Verde, diffuso*.		1 2 2
* Uno dei due LED verdi, usato come indicatore d'alimentazione, è stato successivamente sostituito da un LED bianco, non diffuso, per ragioni di design.			

Tabella A.1: Lista dei pezzi per la costruzione del robot.

A.2 Montaggio del robot

In questa Sezione verrà illustrato ogni passo per la costruzione del robot, a partire dai materiali base. Ogni istruzione sarà corredata da un'immagine per chiarire meglio il passaggio. Tramite queste istruzioni dovrebbe essere possibile smontare con facilità e sostituire eventuali pezzi guasti, o assemblare un ulteriore robot in caso di necessità.

A.2.1 Stampa dei supporti 3D

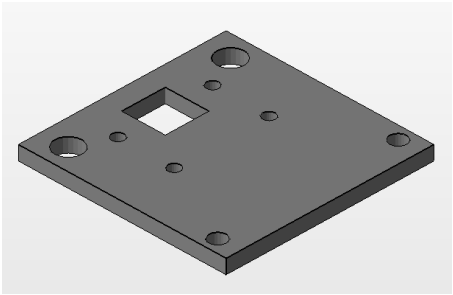
I file per la stampa 3D sono contenuti all'interno del materiale supplementare collegato al documento di tesi. Stamparne uno di ognuno, ad eccezione dei supporti per i gear box dei quali ne servono due. Vedi Figura A.1.

A.2.2 Realizzazione dei circuiti stampati

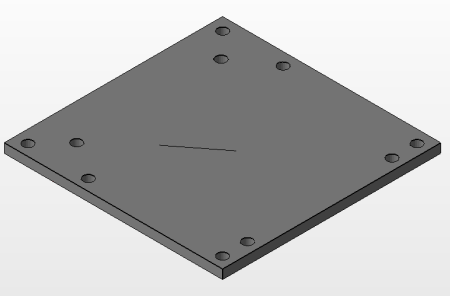
A.2.2.1 Circuito per il motor driver

Per la realizzazione di questo circuito sono necessari i seguenti materiali (Figura A.2a):

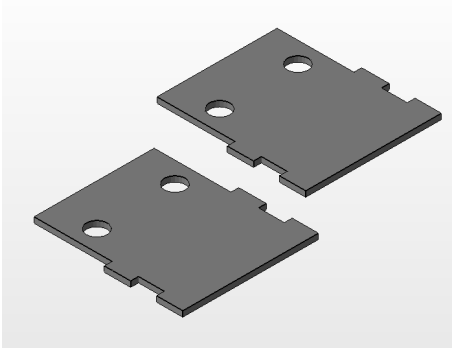
- il motor driver Adafruit TB6612 (con i pin saldati);
- una basetta millefori da 7 cm × 3 cm;



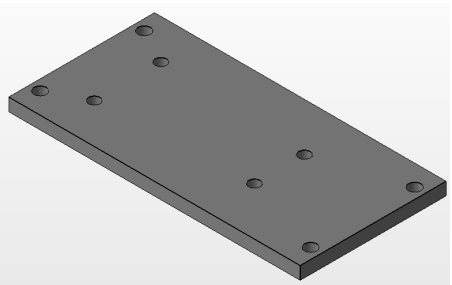
(a) Telecamera. File CameraSupport.stl.



(b) Batteria. File BatterySupport.stl.

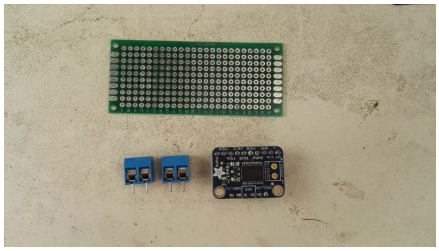


(c) Motori. File GearboxSupport.stl. Stamparne due (uno per motore)

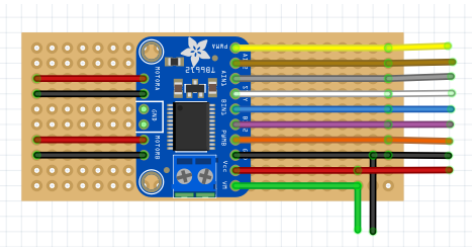


(d) Motor driver. File MotorDriverSupport.stl.

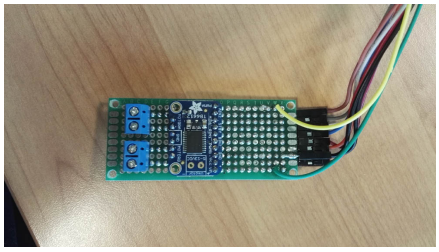
Figura A.1: Elenco dei supporti 3D per il robot.



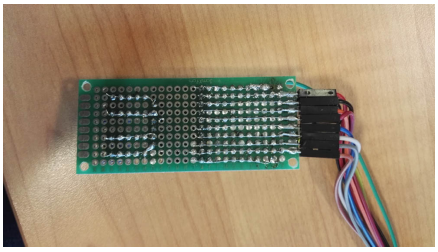
(a) Materiali necessari.



(b) Schema di collegamento del motor driver.



(c) Motor driver completato (fronte).



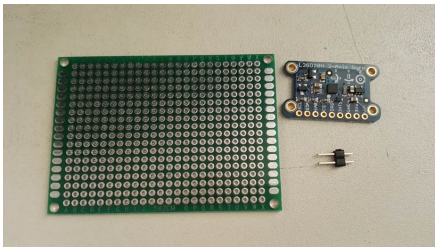
(d) Motor driver completato (retro).

Figura A.2: Passaggi per il montaggio del motor driver.

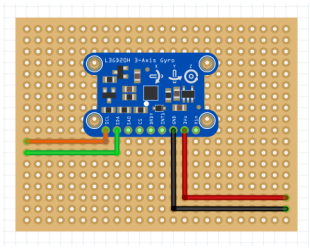
- due morsettiere a due cavi.

Saldare il driver al centro della basetta, e le due morsettiere alle estremità (rivolte verso l'esterno). Effettuare il collegamento sul retro seguendo lo schema in Figura A.2b, quindi saldare dei cavi jumper (del medesimo colore indicato, per semplicità) alle estremità. Per le due terminazioni al bordo non è presente un terminale sulla basetta, per cui è necessario saldare il cavo direttamente su di essa e col filo conduttore. Prestare attenzione al cavo di massa (GND), in nero, per il quale sono presenti due jumper: uno connesso sopra, e uno connesso sotto, ma saldati in modo da essere comunicanti tra loro. Uno andrà ad un Pin GND del Raspberry Pi, mentre l'altro verrà connesso al negativo dell'alimentazione USB secondaria. Il risultato dovrebbe essere simile alle Figure A.2c e A.2d.

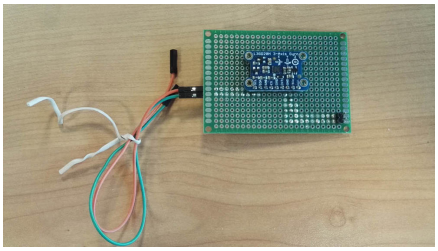
A.2.2.2 Circuito per il giroscopio



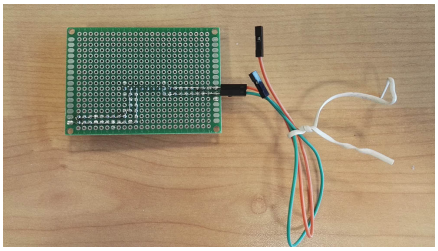
(a) Materiali necessari.



(b) Schema di collegamento del giroscopio.



(c) Giroscopio completato (fronte).



(d) Giroscopio completato (retro).

Figura A.3: Passaggi per il montaggio del giroscopio.

Il circuito del giroscopio è molto semplice, e serve solamente ad esporne i terminali. Sono necessari (Figura A.3a):

- il circuito Adafruit L3G20H (con i pin saldati);
- una basetta millefori da 5 cm × 7 cm;

- una fila di due header pin maschi.

Saldare il giroscopio al centro della base, e i due pin nell'estremità in basso a destra penultima riga, come in Figura A.3b. Collegare i primi due terminali del giroscopio (SCL ed SDA) alle estremità più vicine della base, quindi saldare dei cavi jumper. Infine, collegare i pin GND e VCC agli header pin in basso ed in alto, rispettivamente. Il risultato è mostrato in Figura A.3c e A.3d.

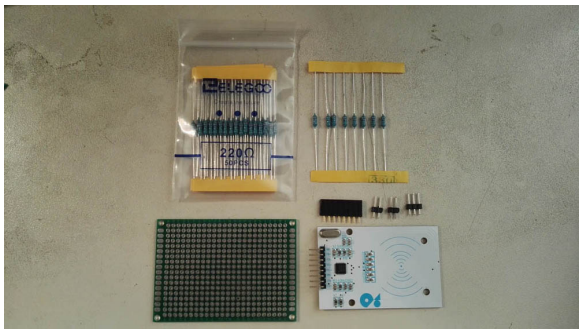
A.2.2.3 Circuito per il lettore RFID e i LED

Per realizzare questo circuito servono (Figura A.4a):

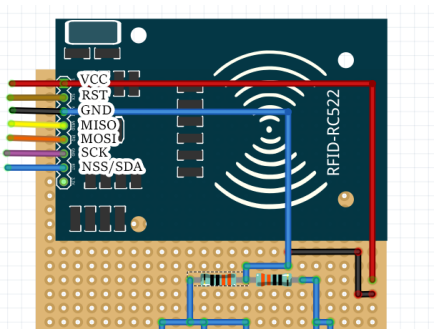
- la scheda MFRC522. Per il montaggio assumeremo l'esatta configurazione visibile in Figura A.4a; eventuali varianti andranno adattate tenendo conto della nota a seguire;
- i seguenti header pin:
  - una fila da 3, maschio;
  - due file da 2, maschio;
  - una fila da 8, femmina.
- una resistenza da 220 Ω;
- una resistenza da 330 Ω;
- una basetta millefori da 5 cm × 7 cm;

Il collegamento del modulo RFID richiede un po' più di attenzione, in base alla specifica implementazione comprata. Il ricetrasmettitore deve essere adagiato parallelamente alla base; nel nostro caso, data la posizione dei suoi terminali, bisogna inserire un header femmina sulla base, dopo averne piegato delicatamente le punte verso il basso usando una pinza, come mostrato in Figura A.4c e A.4d, e successivamente inserire il ricetrasmettitore con la faccia rivolta verso il basso. Assumeremo quindi che la disposizione dei pin sia quella in Figura A.4b. L'importante è seguire il collegamento dei pin VCC e GND, che verranno esposti e condivisi col circuito del giroscopio tramite i terminali in basso a destra; il pin GND inoltre sarà utilizzato come comune per il polo negativo dei cinque LED usati dal robot.

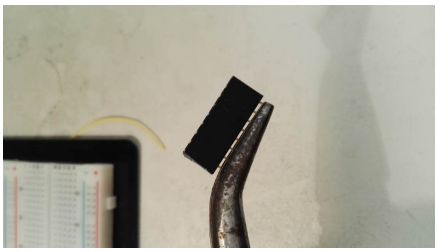
Dopo aver applicato e saldato l'header femmina, proseguire posizionando uno degli header maschi da 2 pin in basso a destra, e le due resistenze come in Figura A.4e: la resistenza da 330 Ω



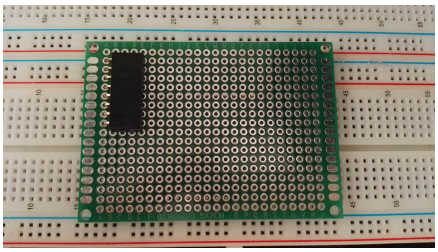
(a) Materiali necessari.



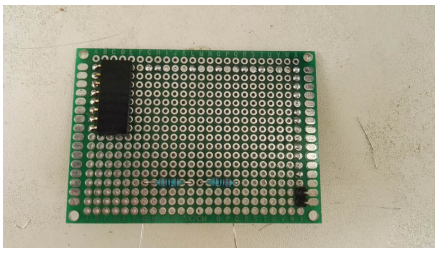
(b) Schema di collegamento del circuito RFID.



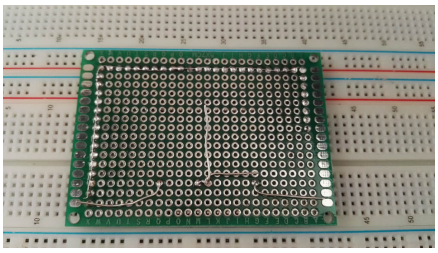
(c) Header pin (piegatura).



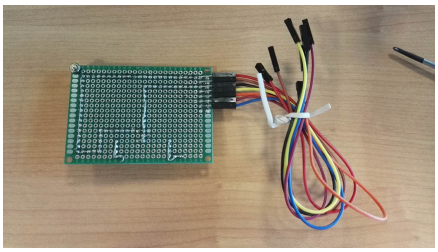
(d) Posizione dell'header.



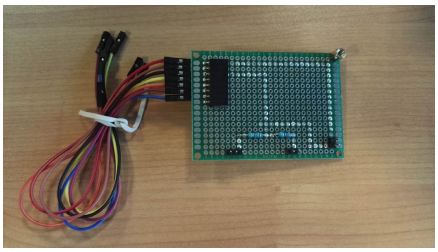
(e) Posizione delle resistenze e pin alimentazione.



(f) Posizione delle resistenze (retro).



(g) RFID completato (fronte).



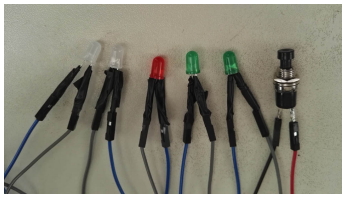
(h) RFID completato (retro).

Figura A.4: Passaggi per il montaggio del circuito RFID.

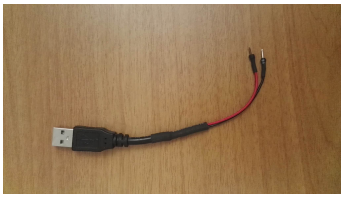
va a destra, mentre quella da 220 Ω va a sinistra. Intrecciare i terminali comuni alle due resistenze, tagliando una delle estremità (Figura A.4f); lasciare per il momento liberi, senza saldarli, i due terminali non a comune. Connettere inoltre il pin superiore dell'header maschio al VCC seguendo lo schema.

Per concludere, posizionare l'header maschio a 3 pin sotto la resistenza da 330 Ω, ed il restante header a 2 pin sotto la resistenza da 220 Ω. Saldare i terminali liberi delle resistenze ai rispettivi header, mettendo a comune i singoli pin di ognuno. Infine collegare il GND al terminale comune delle due resistenze e al pin inferiore dell'header di alimentazione, portando al risultato visibile in Figura A.4g e A.4h.

A.2.2.4 Altri elementi circuitali



(a) LED e pulsante.



(b) Alimentazione USB.

Figura A.5: Elementi circuitali aggiuntivi

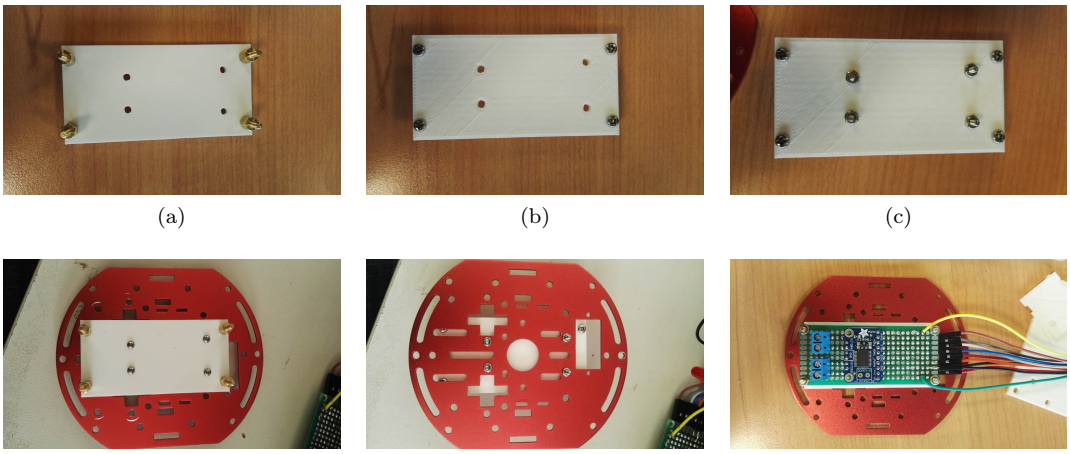
Oltre ai circuiti appena descritti, occorre saldare ai LED e al pulsante di accensione i corrispettivi cavi jumper per effettuare il collegamento, come mostrato in Figura A.5a.

Per collegare l'uscita secondaria della batteria all'alimentazione dei motori presente sul motor driver, è necessario usare un cavo per estrarre la tensione di alimentazione da quest'ultima, come quello in Figura A.5b. È difficile trovarne di già pronti in commercio, ma le istruzioni per realizzarlo sono relativamente semplici. Si veda ad esempio il seguente tutorial:

`https://www.instructables.com/id/Provide-Power-With-an-Old-USB-Cord/`

A.2.3 Assemblaggio del robot

Cominciare fissando quattro distanziali M2.5 maschio / femmina da 6 mm sul supporto per il motor driver (Figure A.6a e A.6b). Fissare quattro viti M2 da 13 mm con altrettanti bulloni facendole sporgere verso il basso (Figura A.6c). I bulloni fungeranno da distanziatori per dopo.

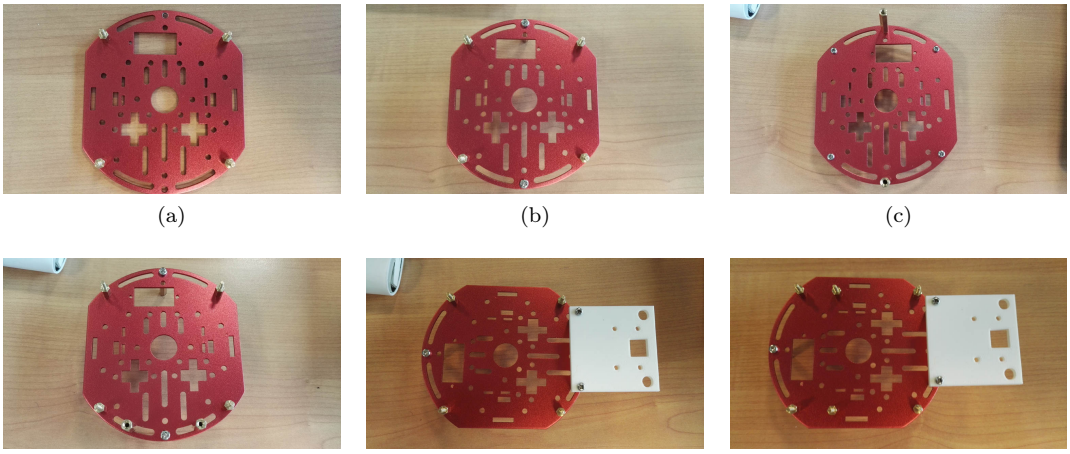


(a) (b) (c)

(d) (e) (f)

Figura A.6: Fissaggio supporto driver.

Fissare il supporto alla base inferiore dello chassis tramite le viti appena aggiunte, come in Figura A.6d e A.6e; è normale che le viti sporgano, serviranno dopo a fissare il robot al rivestimento esterno. Fissare il circuito del motor driver sui distanziali del supporto (Figura A.6f). Fare attenzione a posizionare la parte dei cavi in corrispondenza della finestra rettangolare; potrebbe rendersi necessario allargare i fori della base con una lima.



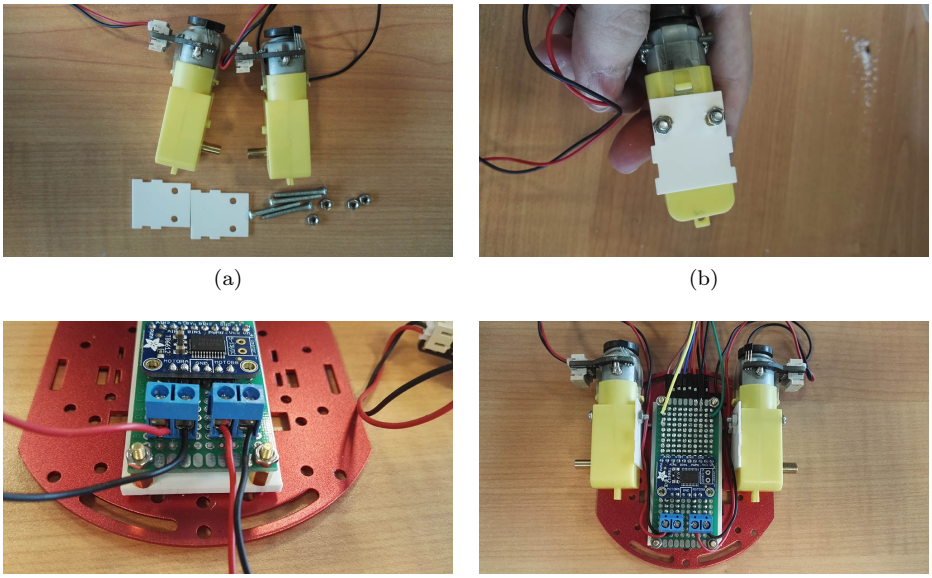
(a) (b) (c)

(d) (e) (f)

Figura A.7: Preparazione della base superiore.

Prendere la base superiore dello chassis, e fissare quattro distanziali M2.5 maschio / femmina da 6 mm nei punti indicati dalla Figura A.7a. Dal kit dello chassis, prendere due distanziali M3 da 23 mm e fissarli verso il basso come indicato in Figura A.7b e A.7c. Aggiungere due distanziali

M2.5 femmina / femmina da 20 mm nei punti indicati dalla Figura A.7d, e regolarli in modo da poter fissare il supporto della telecamera (Figura A.7e). Aggiungere infine altri due distanziali M2.5 maschio / femmina da 6 mm nei punti indicati dalla Figura A.7f.

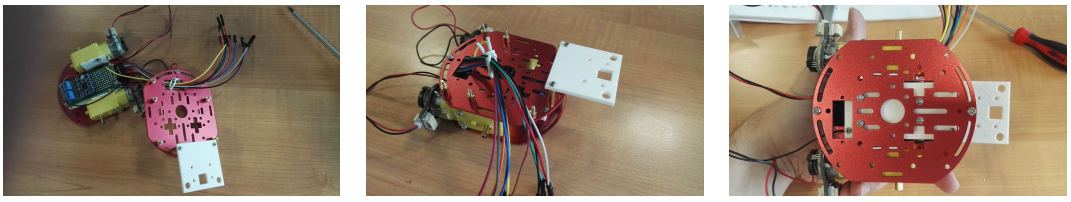


(a) (b)

(c) (d)

Figura A.8: Fissaggio dei motori.

Fissare i motori ai rispettivi supporti usando le viti e i bulloni forniti nel kit dello chassis; il supporto va dal lato interno, opposto a quello dell'asse delle ruote (Figure A.8a e A.8b). Rimuovere il connettore che espone gli encoder, lasciando solamente quello di alimentazione. Fissare i cavi di alimentazione alle morsettiere: per entrambi, il rosso va a sinistra mentre il nero va a destra (Figura A.8c). Adagiare i motori sulla base, facendo in modo di incastrare i supporti sulle corrispondenti barre verticali, come in Figura A.8d.

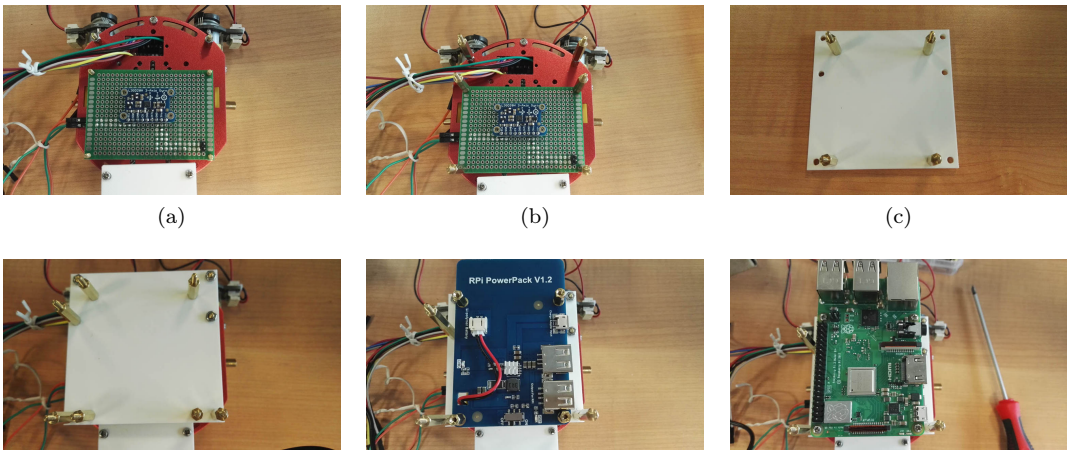


(a) (b) (c)

Figura A.9: Collegamento delle basi.

Far passare i cavi del motor driver dalla finestra rettangolare della base superiore come in Figura A.9a, avendo cura che il supporto camera stia sulla parte alta. Chiudere le due basi,

tenendo fermi i motori e fissando con le viti M3 del kit dello chassis come in Figura A.9b e A.9c. Fare attenzione a lasciare spazio tra i cavi nella parte posteriore per allineare i distanziali.

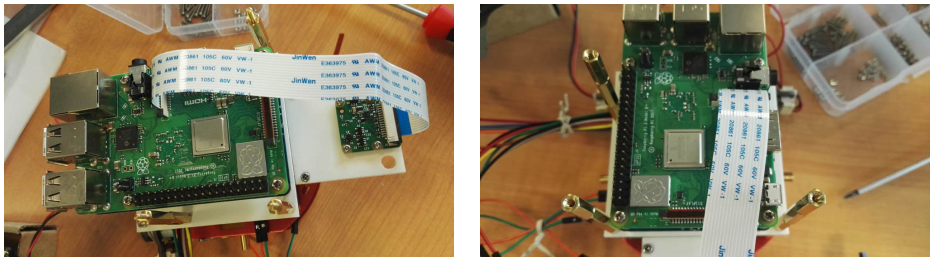


(a) (b) (c)

(d) (e) (f)

Figura A.10: Montaggio batteria e Raspberry.

Inserire il circuito del giroscopio sui distanziali della base superiore. Avvitare due bulloni nei distanziali rimasti per fare spessore (Figura A.10a). Aggiungere sei distanziali M2.5 maschio / femmina da 20 mm (Figura A.10b). Prendere il supporto batteria e fissare, nei punti indicati dalla Figura A.10c, quattro distanziali M2.5 maschio / femmina da 15 mm. Impilare il supporto sopra il circuito del giroscopio, quindi aggiungere i bulloni e altri distanziali M2.5 maschio / femmina da 20 mm come in Figura A.10d. Aggiungere la batteria, fissandola con 4 distanziali M2.5 femmina / femmina da 10 mm. Aggiungere altri 3 distanziali M2.5 maschio / femmina da 20 mm a quelli esposti (Figura A.10e). Aggiungere il Raspberry Pi, fissandolo con delle viti (Figura A.10f). Non dimenticare di aggiungere i dissipatori (non in figura).

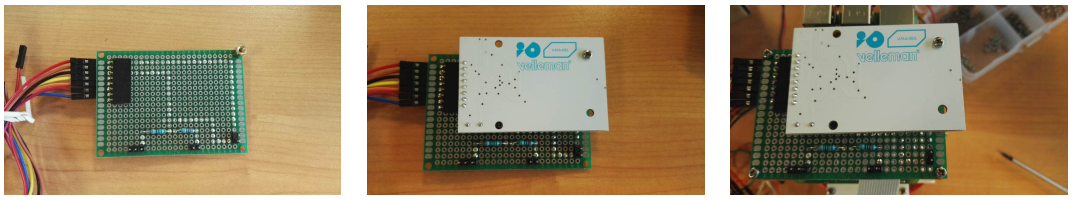


(a) (b)

Figura A.11: Fissaggio telecamera.

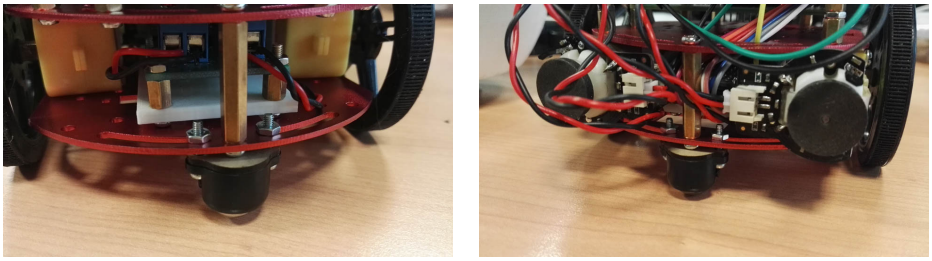
Collegare la telecamera al Raspberry, fissandola con quattro viti e bulloni M2 (Figura A.11a).

Aggiungere 3 distanziali M2.5 femmina / femmina da 15 mm (Figura A.11b).



(a) (b) (c)

Figura A.12: Fissaggio MFRC522.



(a) (b)

Figura A.13: Fissaggio delle caster ball

Aggiungere al circuito RFID un distanziale M2.5 maschio / femmina da 6 mm nell'angolo in alto a destra (Figura A.12a). Inserire il ricetrasmettitore MFRC522 e fissarlo con una vite al distanziale appena inserito come da Figura A.12b.

Prima di montare il circuito appena assemblato procedere al cablaggio, includendo i LED, il pulsante di spegnimento e la batteria usando il cavo precedentemente preparato. I LED bianchi di illuminazione per la telecamera vanno incastrati nei fori presenti sul supporto di quest'ultima. Lo schema generale di collegamento è visibile a pagina intera alla Figura A.14.

Terminato il cablaggio, chiudere la "torre" realizzata ponendo in cima il circuito RFID e fissando con delle viti M2.5, come visibile nella Figura A.12c. Collegare, se non già fatto, i terminali per l'alimentazione a 3.3 V e la massa presenti sul circuito RFID ai corrispondenti del circuito del giroscopio. Infine, fissare le ruote agli assi dei motori e le due caster ball nella parte anteriore e posteriore, come mostrato in Figura A.13a e A.13b. Fare attenzione alle viti usate per fissare le ruote, in quanto il motore scelto richiede delle viti più grandi (M2.5) rispetto a quelle del kit originale. Per fare passare la vite attraverso le ruote è necessario allargare leggermente il foro corrispondente (ad esempio tramite una lima o un cacciavite sottile).

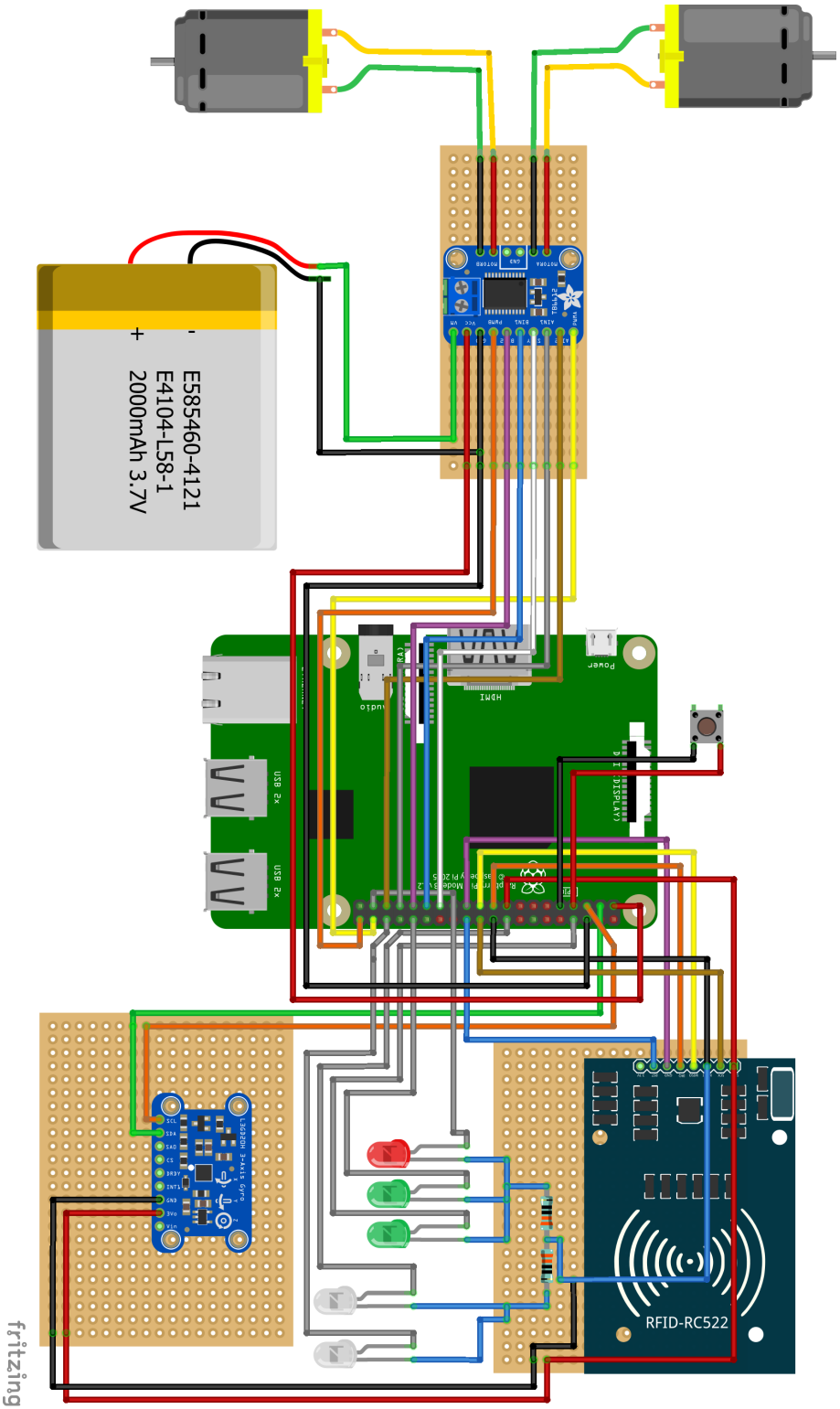
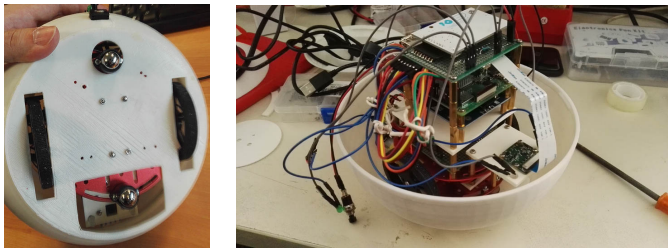
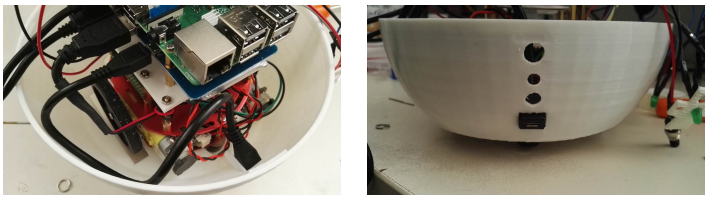


Figura A.14: Schema di collegamento.



(a) (b)

Figura A.15: Fissaggio al rivestimento esterno.



(a) (b)

Figura A.16: Fissaggio del cavo di alimentazione.

A.2.4 Montaggio del rivestimento esterno

Per questa parte occorre aver preparato, tramite stampa 3D, il rivestimento esterno del robot, compreso nel materiale fornito. Dopo aver effettuato la stampa, sovrapporre il robot alla parte inferiore, facendo passare le viti alla base attraverso i fori predisposti come in Figura A.15a, quindi fissare con quattro bulloni M2.

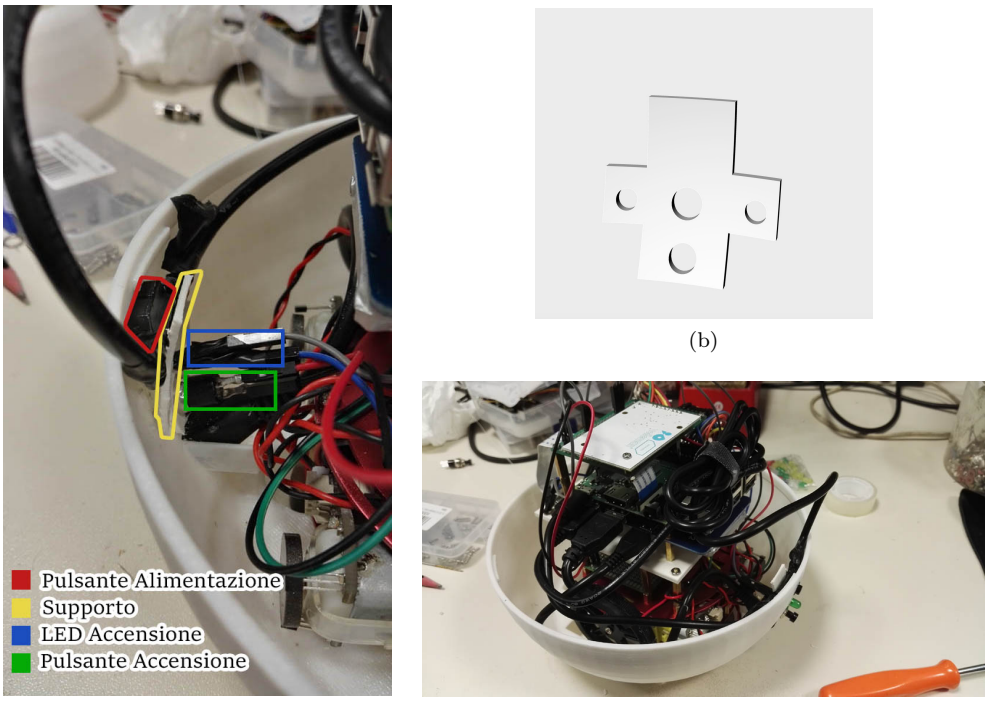
Collegare l'estremità maschio dell'extender micro USB all'alimentazione del modulo batteria, quindi incastrare l'estremità femmina nell'apposito alloggio inferiore mostrato in Figura A.16a e A.16b.

Prima di collegare il pulsante di alimentazione, occorre rimuovere il rivestimento esterno, come in Figura A.17a. Si consiglia di proteggere i collegamenti, ora esposti, mediante nastro isolante (Figura A.17b e A.17c). Procedere quindi a collegare l'uscita USB primaria del modulo batteria con l'alimentazione del Raspberry.

Installare gli elementi del pannello posteriore. A tal fine, sfruttare il supporto a opposizione della Figura A.18b. Collegare il pulsante di alimentazione, il led di alimentazione, ed il pulsante di accensione e spegnimento come mostrato in Figura A.18a. Si consiglia di incastrare il pulsante di alimentazione per ultimo, dopo aver fermato il supporto tramite il dado presente sul pulsante.

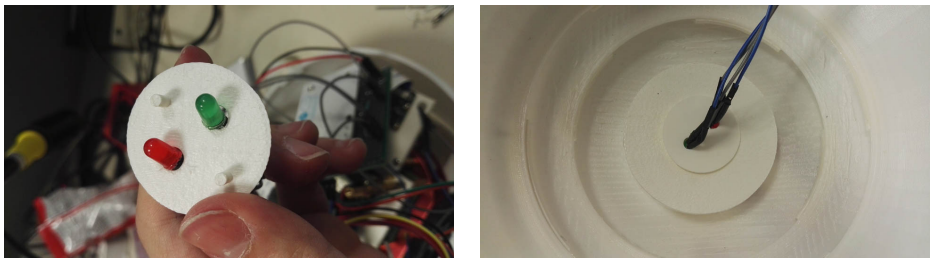
(a) (b) (c)

Figura A.17: Pulsante di alimentazione.



(a) (b) (c)

Figura A.18: Fissaggio pannello posteriore



(a) (b)

Figura A.19: Fissaggio LED delle tessere.

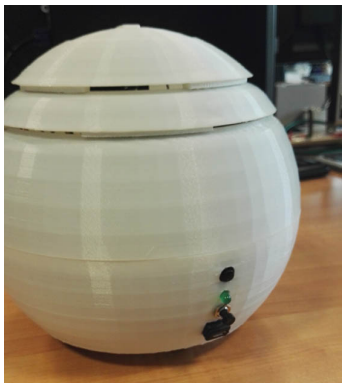


Figura A.20: Risultato finale.

Incastrare la coppia di LED dedicati al feedback per le tessere sul supporto visibile in Figura A.19a, quindi incastrare il supporto alla parte superiore del rivestimento come in Figura A.19b. Chiudere il robot completato, che si presenta ora come in Figura A.20.

A.3 Ripristino, calibrazione e funzioni di prova

In questa Sezione verrà spiegato come ripristinare e riconfigurare il software del robot in caso di danneggiamento dell'unità di memoria. Verranno inoltre presentate alcune funzionalità di prova che potrebbero risultare utili a tal fine. A corredo, verrà inoltre aggiunta una piccola Sezione dedicata alle funzionalità di prova disponibili sul software in esecuzione sul server centrale.

A.3.1 Ripristino dell'immagine di sistema

Per riportare il robot alla configurazione originale, è sufficiente sovrascrivere il contenuto della memoria SD con l'immagine di sistema fornita. Un software popolare per eseguire tale operazione in ambiente Windows è Etcher, reperibile all'indirizzo `https://etcher.io`. Dopo aver selezionato l'immagine da scrivere, scegliere la memoria su cui verrà trasferita ed eseguire il comando di scrittura. Nota: tutti i dati presenti in precedenza verranno sovrascritti.

Nella configurazione originale:

- l'uscita HDMI è *disabilitata*;
- il server ssh è *abilitato*;

- la password per l'utente pi è *quella di default* (raspberry);
- non sono presenti configurazioni wireless.

La prima cosa da fare è configurare il sistema per accedere alla rete wireless su cui dovrà operare. Ciò può essere fatto montando la scheda SD appena scritta in ambiente Unix, o accedendo via ssh in rete locale tramite la porta Ethernet dopo aver inserito la scheda nel Raspberry ed averlo acceso. Dopodiché procedere seguendo le istruzioni disponibili a questo indirizzo, inserendo i parametri della rete:

https://www.raspberrypi.org/documentation/configuration/wireless/wireless-cli.md

Dopo aver configurato le impostazioni wireless (ed eventualmente aver riavviato il Raspberry) il robot è già operativo e funzionale. Tuttavia, per ragioni di sicurezza, si raccomanda di cambiare la password di default e disabilitare il client ssh (dopo essersi assicurati che il sistema funzioni correttamente). Per quest'ultima operazione, consultare la guida al seguente indirizzo:

https://www.raspberrypi.org/documentation/remote-access/ssh/

A.3.2 Uso quotidiano

Dopo aver impostato la configurazione di rete, il robot funziona in maniera *plug-and-play*: è sufficiente accenderlo ed esso resterà in attesa dell'annuncio del server per potersi connettere ad esso.

Sul pannello principale del robot sono presenti i seguenti elementi, dall'alto in basso:

- l'interruttore di alimentazione, che interrompe il collegamento con la batteria; va spento dopo aver spento il Raspberry per non sprecare energia;
- il LED di accensione, segnala lo stato di attività del Raspberry; quando è spento è possibile spegnere il collegamento con la batteria;
- il pulsante di spegnimento del Raspberry; tenendolo premuto per tre secondi il Raspberry si spegne in maniera pulita mediante la procedura di *shutdown*;
- il connettore di ricarica micro USB.

Il robot si accende in automatico dopo aver acceso l'interruttore d'alimentazione.

A.3.3 Nota importante prima di effettuare operazioni di verifica

All'avvio del robot il software viene avviato automaticamente da uno script di inizializzazione. Prima di modificare i parametri di controllo ed effettuare delle prove, occorre interrompere il client invocando:

```
sudo /etc/init.d/main-provider.sh stop
```

Dopo aver terminato, è possibile riavviare il client invocando:

```
sudo /etc/init.d/main-provider.sh start
```

A.3.4 File di configurazione

Il software in esecuzione sul robot è collocato nella cartella `/home/pi/line-follower`. In particolare nella cartella `modules` sono presenti tutti gli script attivi ed i programmi nativi. Di particolare rilevanza è il file `settings.py` (visibile alla Figura A.21).

```
simulate = False
mock_orbits = False
mock_tags = False
camera_args = ["-t", "25", "-tck", "0", "-cthr", "150", "170", "0", "179",
    "255", "255"]
orbit_scanner_args = ["-thr", "23", "23", "10", "7", "-sleep", "33"]

motor_gain_left = 1.02
motor_gain_right = 1.0

control = {
    Planets.MERCURY: {
        'speed': 40,
        'kp': 0.225,
        'kd': 0.06,
        'ki': 0.06,
        'segments': [360 / 4 * n for n in range(5)] # Extremes (0 and 360)
            must be included both
    },
...
```

Figura A.21: Estratto del file `settings.py`.

Questo script contiene alcune impostazioni globali per il software, tra le quali quelle di controllo per ogni orbita, che può essere necessario adattare in caso di sostituzione dei motori.

`simulate` Se impostato a `True`, la velocità dei motori è impostata a zero. Utile per verificare funzionalità che non riguardano il movimento del robot.

`mock_orbits`
Se impostato a `True`, sostituisce lo scanner dei marcatori delle orbite con uno finto che restituisce sempre `PLANET_OK`. Utile se non si ha bisogno di verificare lo scanner dei marcatori.

`mock_tags` Se impostato a `True`, simula il passaggio delle tessere, fornendo subito le risposte corrette.

`camera_cmd`
La lista di argomenti aggiuntivi da passare al software di line-following (vedi 4.2.2.6). Le opzioni verranno spiegate in (A.3.6.1).

`orbit_scanner_args`
La lista di argomenti aggiuntivi da passare al programma di scansione delle orbite (vedi 4.2.2.8). Le opzioni verranno spiegate in A.3.6.2.

`motor_gain_left` / `right`
Guadagni moltiplicativi costanti per compensare (approssimativamente) piccole discrepanze tra i due motori.

`control` Un dizionario che comprendere delle impostazioni specifiche per pianeta, nello specifico:

- `speed` è la velocità base del robot, in scala da 0 a 100;
- `kp`, `kd`, `ki` sono i parametri PID del controllo in retroazione;
- `segments` è l'array delle posizioni dei segmenti (in gradi sessadecimale) di riallineamento per ogni orbita.

A.3.5 Verifica dei motori e del giroscopio

Lo script `robot_control.py` contiene un'utile funzione di prova che permette di provare autonomamente la funzionalità di line following del robot. Dalla cartella `modules`, invocare:

```
python3.6 -c "import␣planets;␣import␣robot_control␣as␣rc;␣
    rc.run(planet=planets.MERCURY,␣simulate=False,␣sample_read_interval=1.0)"
```

I parametri possono essere variati a piacere. Dopo l'avvio, il robot effettua la calibrazione del giroscopio, quindi inizia il programma di line following usando le impostazioni per il pianeta passato come argomento. All'intervallo di tempo definito come parametro, manda in output i dati del giroscopio (orientamento e velocità di rotazione). Se il parametro `simulate` è impostato a `True`, i motori non vengono azionati, permettendo di provare solamente il giroscopio.

A.3.6 Verifica della telecamera

Prima di procedere, occorre attivare manualmente i LED di illuminazione (in quanto tale funzionalità non è presente nei programmi che gestiscono la telecamera). Invocare lo script d'aiuto:

```
python3.6 ledon.py &
```

Per spegnere i LED basta terminare il processo appena iniziato.

I due programmi che sfruttano la telecamera si trovano nella cartella `modules/cpp`. È presente un file di configurazione CMake per poter ricompilare i sorgenti nel caso in cui si rendesse necessario. Il resto del programma si aspetta di trovare i file eseguibili nella cartella `cpp/build`, per cui assicurarsi di produrre l'output in quella directory.

A.3.6.1 Camera Module

Il programma `camera-module` serve a rilevare la linea nera sul pavimento, le bande di riallineamento e a restituire l'offset rispetto al centro. L'output è formato da un codice identificativo numerico (non leggibile, per ragioni storiche di implementazione), seguito dall'offset. Esso accetta i seguenti argomenti:

`-gui` Utile solamente se il robot è connesso a uno schermo. Mostra l'immagine catturata, insieme a informazioni di debug (contorni rilevati in verde, baricentro del contorno maggiore con un cursore blu), direttamente sull'interfaccia grafica.

`-stream` Come `-gui`, ma invia l'immagine ad un client remoto tramite un server TCP. La funzionalità è molto basilare e l'invio dei dati non ottimizzato, per cui si sconsiglia di usarlo mentre si prova lo spostamento del robot (introduce un ritardo potenzialmente importante). Il server viene lanciato all'avvio e il programma resta in attesa di una connessione prima di procedere. Per l'uso del client, vedere a seguire.

`-nodraw` Disabilita i contorni e il cursore sull'immagine di debug (se attivata). Utile se si eseguono prove di thresholding sul client (vedere a seguire).

`-cthr <lh> <ls> <lv> <hh> <hs> <hv>`
Soglia di colore per le bande di riallineamento. I primi tre parametri sono il valore minimo di HSV accettato, mentre gli ultimi tre sono i valori massimi. Aggiustarli se si hanno problemi a rilevare correttamente le bande.

`-t <threshold>`
Soglia binaria per distinguere la linea dal fondo. Aggiustare questo parametro se si hanno problemi di rilevamento; ad esempio, se il robot si ferma sulla linea (valore troppo basso) o esce fuori dal percorso (valore troppo alto).

`-tck <thickness>`
Soglia di tolleranza per il massimo errore accettato. Se l'offset è in valore assoluto inferiore al parametro, viene riportato come zero. Si tratta di un'opzione deprecata, normalmente impostata a 0.

Come accennato è possibile visualizzare dal vivo, in remoto, l'immagine rilevata dalla telecamera. A tal fine è presente nel materiale consegnato un client che si connette al server attivo sul Raspberry, e che permette di verificare al volo diversi valori di sogliatura per le bande. Si trova all'interno della cartella `stream-receiver`. Per la compilazione è necessario che siano installate le librerie Boost, ad esempio tramite VCPKG[1], ma è presente anche una build testata su Windows 10.

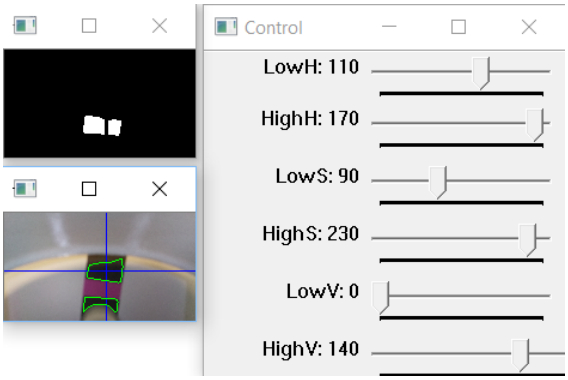


Figura A.22: Schermata del ricevitore video di prova.

[1] Vedi https://github.com/Microsoft/vcpkg

Il client va lanciato mediante il comando:

```
stream-receiver <server-ip> <server-port>[ -cthr <lh> <ls> <lv> <hh> <hs>
    <hv>]
```

La porta di default del server è la 2020. L'opzione `-cthr` è assolutamente analoga a quella presente sul robot e permette di modificare i parametri di partenza per la sogliatura a valle delle bande.

A.3.6.2 Orbit Scanner

Il programma `orbit-scanner` usa la libreria ArUco per rilevare il marcatore di un pianeta passato come parametro. La metodologia è stata già descritta in 4.2.2.8; in questa Sezione mi addentrerò meglio nei parametri che è possibile passare per configurarlo. Nota: il primo argomento è sempre il nome inglese, con l'iniziale maiuscola e il corpo minuscolo del pianeta da cercare (ad esempio `Mercury`, `Earth` etc.).

`-stream` Assolutamente identico a quanto visto in A.3.6.1; permette di trasmettere a un client TCP l'immagine rilevata e decodificata.

`-test <file>`
: Invece di usare la telecamera, effettua una prova su di un'immagine di prova passata come argomento.

`-thr <min> <max> <step>`
: Parametri di sogliatura per l'algoritmo di ArUco, per i quali si rimanda alla documentazione[2].

`-sleep <millis>`
: L'intervallo tra un'analisi del frame e la successiva, in millisecondi.

`-c <contrast>`
: Costante moltiplicativa tra zero e uno per l'immagine. Opzione deprecata e non usata nella configurazione finale.

[2] https://docs.opencv.org/3.2.0/d1/dcd/structcv_1_1aruco_1_1DetectorParameters.html

Il protocollo usato con l'opzione `-stream` è compatibile con lo stesso programma `stream--receiver` descritto sopra, anche se ovviamente le opzioni di sogliatura per le bande non hanno utilità. Ad ogni modo se l'opzione è attivata l'immagine viene inviata con già attive le tracce di debug della libreria ArUco, che evidenziano quali marcatori sono stati rilevati.

A.3.7 Verifica e riscrittura delle tessere RFID

Figura A.23: Software di scrittura e verifica delle tessere.

Lo script `PlanetReader.py` implementa la funzionalità di lettura delle tessere. Può essere eseguito direttamente per provarlo, tramite:

```
python3.6 PlanetReader.py <nome-pianeta>
```

dove `<nome-pianeta>` è, come negli altri casi, il nome inglese, con iniziale maiuscola, del pianeta da verificare. Il suo funzionamento è stato già discusso in 4.2.2.9. Più utile, per la manutenzione, è invece lo script `PlanetFlasher.py`. Se eseguito, mostra una semplice interfaccia testuale (visibile in Figura A.23) che permette di provare singole tessere o scrivere sulle tessere gli identificativi delle diverse curiosità per pianeta, singolarmente o tutti insieme in sequenza. Può essere usato per sostituire una tessera guasta o verificare una tessera che non si riesce a riconoscere.

A.3.8 Monitoraggio generale

Il client di gestione del robot viene eseguito tramite lo script `main.py`; normalmente tale script viene avviato indirettamente all'avvio da un init script (`/etc/init.d/main-provider.sh`), ma può essere anche avviato manualmente. In tal caso, viene mostrata un'interfaccia grafica (visibile alla Figura A.24) che tiene traccia dello stato dell'esperienza, molto simile concettualmente a quella disponibile sul sistema di proiezione (vedi A.4). Può essere usata per visualizzare in tempo

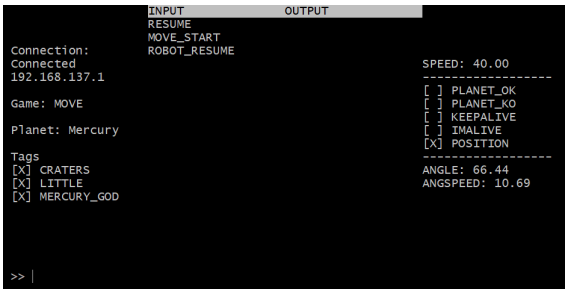


Figura A.24: Schermata di monitoraggio del robot.

reale dati rilevanti quale quelli provenienti dai sensori, tracciare i messaggi di rete o eseguire piccoli comandi, quali:

`q` Esce dall'applicazione

`s <attr> <val>`
Se <attr> è un attributo definito in `settings.py`, ne imposta il valore a <val>. Il tipo viene inferito da quello attuale, e il nuovo valore viene ottenuto applicando il costruttore del tipo inferito alla stringa <val>.

`p <attr>` Se <attr> è un attributo definito in `settings.py`, ne stampa il valore a schermo.

Oltre all'interfaccia di monitoraggio un'altra feature molto utile è il file `log` generato dall'applicazione, sul quale vengono scritte eventuali informazioni aggiuntive e tracciate le eccezioni. Il file viene sovrascritto ad ogni avvio, ma è possibile visualizzarlo in corso d'opera tramite l'utility `lnav`.

A.4 Uso del software di proiezione

Prima di avviare il software di proiezione, è importante assicurarsi che le connessioni in ingresso al server non vengano bloccate dal firewall in esecuzione sul PC. A tal fine, assicurarsi di consentire l'accesso alle porte 6600 e 6603.

Sebbene normalmente non vi sia bisogno di operare direttamente sul sistema di proiezione, sono disponibili alcuni comandi che, se usati, permettono una serie di azioni ausiliarie utili in fase di sviluppo o settaggio. La lista completa dei comandi è presente in Tabella A.2.

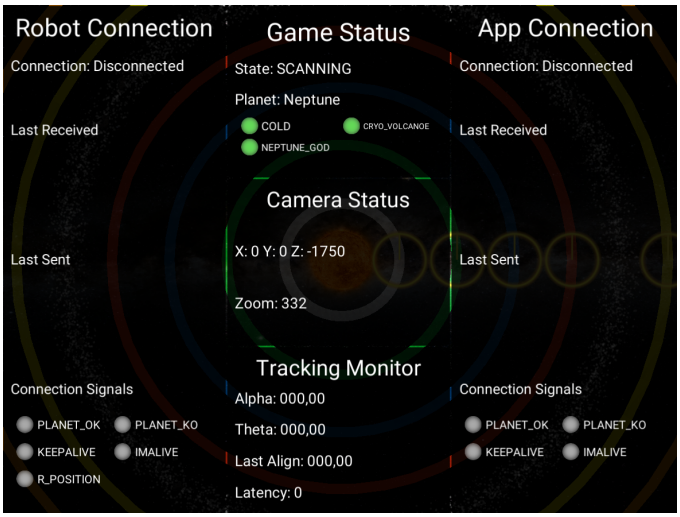


Figura A.25: Schermate di debug.

Tasti	Descrizione
`W`, `A`, `S`, `D`, `Spazio`	Mostrano / nascondono alcuni pannelli con delle informazioni di diagnostica. Nella Figura A.25 vengono mostrati tutti i pannelli attivi contemporaneamente.
Frecce direzionali	Spostano la telecamera sul piano orizzontale, per aggiustarne la posizione rispetto al tracciato.
`+`, `-`	Aumentano / diminuiscono il livello di zoom, per adattarlo al tracciato.
`(Alt+) 1...9`	Aumentano / diminuiscono l'asse maggiore delle orbite e il raggio della cintura di asteroidi. Ogni tasto corrisponde a un pianeta (in ordine crescente di distanza reale dal Sole), col tasto 5 assegnato alla cintura di asteroidi.
`Ctrl+(Alt+) 1...9`	Aumentano / diminuiscono l'offset verticale delle orbite dei pianeti (numero 5 escluso).
`Z`	Ripristina le dimensioni originali delle orbite dei pianeti e della cintura di asteroidi.

Tabella A.2: Elenco dei comandi di debug e calibrazione per il sistema di proiezione.

Il livello di zoom, la posizione della telecamera e le dimensioni delle orbite permangono alla chiusura del programma, per cui non è necessario impostarli nuovamente durante le sessioni successive.

Bibliografia

[1] Diego Martinoia, Daniele Calandriello, and Andrea Bonarini. Physically interactive robogames: Definition and design guidelines. *Robotics and Autonomous Systems*, 61(8):739 – 748, 2013.

[2] Ayberk Ozgur, Séverin Lemaignan, Wafa Johal, Maria Beltran, Manon Briod, Léa Pereyre, Francesco Mondada, and Pierre Dillenbourg. Cellulo: Versatile handheld robots for education. *HRI '17: ACM/IEEE International Conference on Human-Robot Interaction Proceedings*, pages 119–127, 2017. Best Human-Robot Interaction Design paper award.

[3] L. Hostettler, A. Özgür, S. Lemaignan, P. Dillenbourg, and F. Mondada. Real-time high-accuracy 2d localization with structured patterns. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4536–4543, May 2016.

[4] Ayberk Ozgur, Wafa Johal, Francesco Mondada, and Pierre Dillenbourg. Windfield: Learning wind meteorology with handheld haptic robots. *HRI '17: ACM/IEEE International Conference on Human-Robot Interaction Proceedings*, pages 156–165, 2017.

[5] M. Faria, A. Costigliola, P. Alves-Oliveira, and A. Paiva. Follow me: Communicating intentions with a spherical robot. In *2016 25th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pages 664–669, Aug 2016.

[6] G. Piumatti, F. G. Prattico, G. Paravati, and F. Lamberti. Enabling autonomous navigation in a commercial off-the-shelf toy robot for robotic gaming. In *2018 IEEE International Conference on Consumer Electronics (ICCE)*, pages 1–6, Jan 2018.

[7] Unity 3d website. https://unity3d.com/.

[8] asyncio - Asynchronous I/O, event loop, coroutines and tasks. https://docs.python.org/3/library/asyncio.html.

[9] Pexpect version 4.6 - Pexpect 4.6 documentation. https://pexpect.readthedocs.io/en/stable/.

[10] Opencv library. http://opencv.org/.

[11] Francisco J. Romero-Ramirez, Rafael Muñoz-Salinas, and Rafael Medina-Carnicer. Speeded up detection of squared fiducial markers. *Image and Vision Computing*, 76:38 – 47, 2018.

[12] S. Garrido-Jurado, R. Mu noz Salinas, F.J. Madrid-Cuevas, and M.J. Marín-Jiménez. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280 – 2292, 2014.

[13] S. Garrido-Jurado, R. Mu noz Salinas, F.J. Madrid-Cuevas, and R. Medina-Carnicer. Generation of fiducial marker dictionaries using mixed integer linear programming. *Pattern Recognition*, 51:481 – 491, 2016.

[14] Mfrc522-python. https://github.com/pimylifeup/MFRC522-python.

[15] Unity - Scripting API: ParticleSystem. https://docs.unity3d.com/ScriptReference/ParticleSystem.html.

[16] Unity - scripting api: Animator. https://docs.unity3d.com/ScriptReference/Animator.html.

[17] John Brooke. Sus: A quick and dirty usability scale, 1996.

[18] W.A. IJsselsteijn, Y.A.W. de Kort, and K. Poels. *The Game Experience Questionnaire*. Technische Universiteit Eindhoven, 2013.

[19] Bob G. Witmer and Michael J. Singer. Measuring presence in virtual environments: A presence questionnaire, 1998.

[20] Flinters university teaching for learning questionnaire. http://www.flinders.edu.au/Teaching_and_Learning_Files/Documents/questions_for_self_appraisal.pdf.