Itip “Luigi Bucci” Faenza

Articolazione Informatica e Telecomunicazioni

Odysseus: Progettazione di un robot che gioca a scacchi

di

Enea Vignoli

Ringraziamenti

Prima di ogni cosa desidero ringraziare tutte le persone, amici, docenti, e appassionati che mi hanno aiutato e sostenuto, sperando di non dimenticare nessuno di coloro che lungo questo percorso ha riposto fiducia in me.

Questo progetto non sarebbe stato possibile senza l'aiuto, l'incoraggiamento e l'ispirazione fornitami dall'intera associazione Maker Station – Fablab della Bassa Romagna, la quale mi ha procurato conoscenze, mezzi e spesso competenze pratiche vitali per molti aspetti concernenti la progettazione e la realizzazione di Odysseus. In particolare vorrei citare Massimo Frassetto, Enzo Cortesi ed Etela Manaresi.

Vorrei inoltre ringraziare tutti i professori che mi hanno guidato e aiutato, per il supporto tecnico vorrei ricordare Raoul Gioia, Laura Facciponte e Samuele Cattabriga.

Considerevole è stato anche l'apporto di amici e familiari, per cui desidero citare Andrea Ghetti e Gresjan Tabaku, oltre ai miei genitori, sempre presenti e disponibili.

Sommario

Questa relazione presenta Odysseus, una scacchiera automatizzata che, grazie a un basilare agente intelligente e ad alcuni semplici componenti elettro-meccanici, è in grado di giocare a scacchi contro un avversario umano.

Quest'ultimo si ritroverà a fronteggiare un algoritmo di ricerca a profondità limitata, il quale analizzerà ciclicamente una notevole mole di mosse ammissibili, schematizzandole tramite una struttura dati chiamata albero di ricerca. Una volta generata questa struttura, verrà applicata in maniera ricorsiva una funzione di valutazione della qualità che una determinata mossa può avere, fornendo quindi un criterio di valutazione necessario e di vitale importanza per la scelta dell'effettiva mossa da attuare.

Terminato il processo di decisione, viene mosso il pezzo interessato grazie al magnete posto sotto di esso e all'elettromagnete che lo trascina, a sua volta situato sotto il piano di gioco. L'elettromagnete si muove sull'asse delle ascisse e delle ordinate su tubi di alluminio, di fatto grazie ad un sistema di cinghie azionato da motori elettrici passo passo, che permettono un accurato controllo degli spostamenti.

Ora il giocatore umano potrà agire di conseguenza, spostando manualmente il proprio pezzo, perché è attivo un sistema di rilevamento dei movimenti delle pedine, che manterrà il computer sempre aggiornato sulla situazione di gioco. Invero, sensori di rilevamento dei campi magnetici, dei pezzi per l'appunto, sono posti sotto ogni cella della scacchiera.

Qualsiasi eventualità che potrebbe generare incongruenze di gioco può essere agevolmente gestita tramite l'interfaccia grafica del computer.

Indice dei contenuti

1 Introduzione

1.1 Gli Scacchi e i Computer

Il gioco degli scacchi è stato tra i primi problemi affrontati nel corso degli studi legati all'Intelligenza Artificiale, con i primi lavori di molti pionieri dell'informatica, tra cui Konrad Zuse nel 1945, Norbert Wiener nel 1948 e Alan Turing nel 1950. Tuttavia, fu l'articolo *Programming a Computer for Playing Chess* di Claude Shannon nel 1950 che fece veramente la differenza, descrivendo una rappresentazione per le posizioni della scacchiera, una funzione di valutazione, una ricerca non completa e alcuni spunti per una ricerca selettiva dell'albero di gioco.

Il primo incontro scacchistico fra computer vide contrapposti il programma ITEP, scritto a metà degli anni '60 all'Istituto di Fisica Teorica e Sperimentale di Mosca, e il programma Kotok-McCarthy del MIT. I calcolatori non erano situati nello stesso luogo, infatti le mosse venivano comunicate grazie al telegrafo; l'incontro finì nel 1967 con la vittoria per 3-1 di ITEP.

Negli anni successivi il livello di questi agenti artificiali è cresciuto costantemente, al punto che nel 1997 il programma Deep Blue, sviluppato da IBM, riuscì a battere il campione del mondo Garry Kasparov, segnando per sempre l'immaginario collettivo.

L'evoluzione dell'hardware e dell'IA negli ultimi decenni ha raggiunto livelli tali da rendere nulle le possibilità di vittoria umane contro un moderno programma per giocare a scacchi. Nonostante il fatto che sia gli umani che i computer cerchino di predire come si evolverà la situazione di gioco, i primi sono molto più selettivi nella scelta di quali mosse possibili analizzare. D'altronde, i computer fanno affidamento sulla forza bruta per esaminare più posizioni possibili, senza essere in grado di effettuare una selezione simile a quella umana. Basti pensare che Garry Kasparov potrebbe esaminare non più di 3-5 posizioni al secondo, in contrapposizione alle 200 milioni al secondo analizzate da Deep Blue, per raggiungere un livello di gioco non troppo dissimile.

Risulta chiaro come questa differenza abissale vada analizzata per fornire delucidazioni in seno al differente approccio che uomo e macchina adottano per sfidarsi.

Attraverso la notevole quantità di documenti e studi pubblicati su questo argomento nel corso degli anni, ho costruito una scacchiera automatizzata, Odysseus, che è in grado di fronteggiare un giocatore umano alle prime armi.

2 Architettura Hardware

2.1 Strumenti Programmabili

Per suddividere il carico di lavoro e quindi anche le problematiche da gestire per realizzare Odysseus, è stato deciso di utilizzare due piattaforme programmabili: Raspberry Pi 3 Model B e Arduino Mega 2560 Rev3.

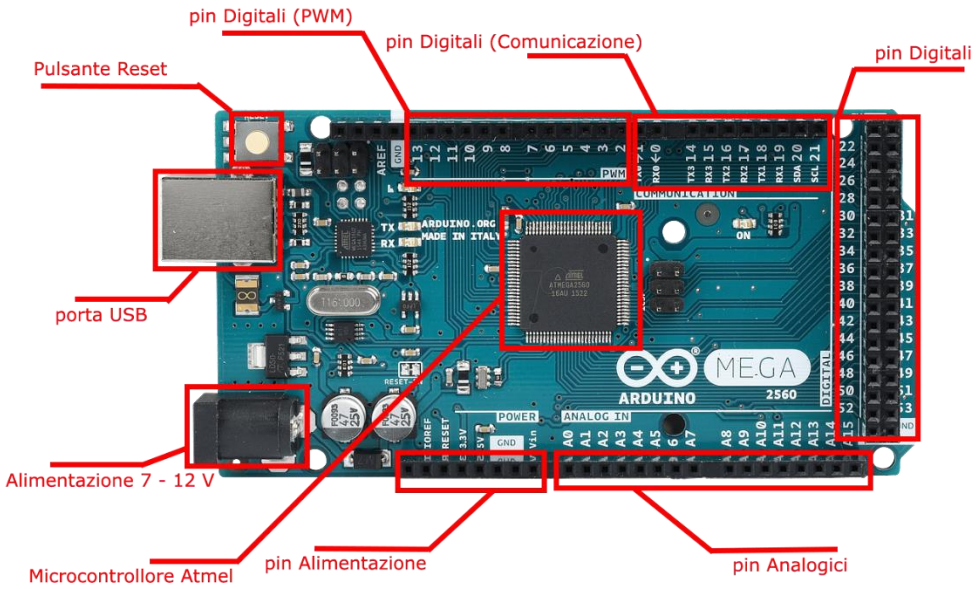
La parte legata all'aspetto di controllo dell'elettronica viene gestita da Arduino, mentre la componente "intelligente" del sistema, cioè l'intelligenza artificiale programmata in Python viene retta da Raspberry Pi. I due dispositivi comunicano in serialmente tramite le corrispondenti porte USB.

2.1.1 Arduino

Arduino è una piattaforma elettronica open-source nata presso l'Interaction Design Institute di Ivrea nel 2005, e negli ultimi tredici anni migliaia di persone in tutto il mondo lo hanno scelto per realizzare i propri progetti. Questi ultimi hanno trovato posto nella casa di hobbisti, negli studi di professionisti e in molte scuole, dove Arduino può diventare una grande risorsa didattica.

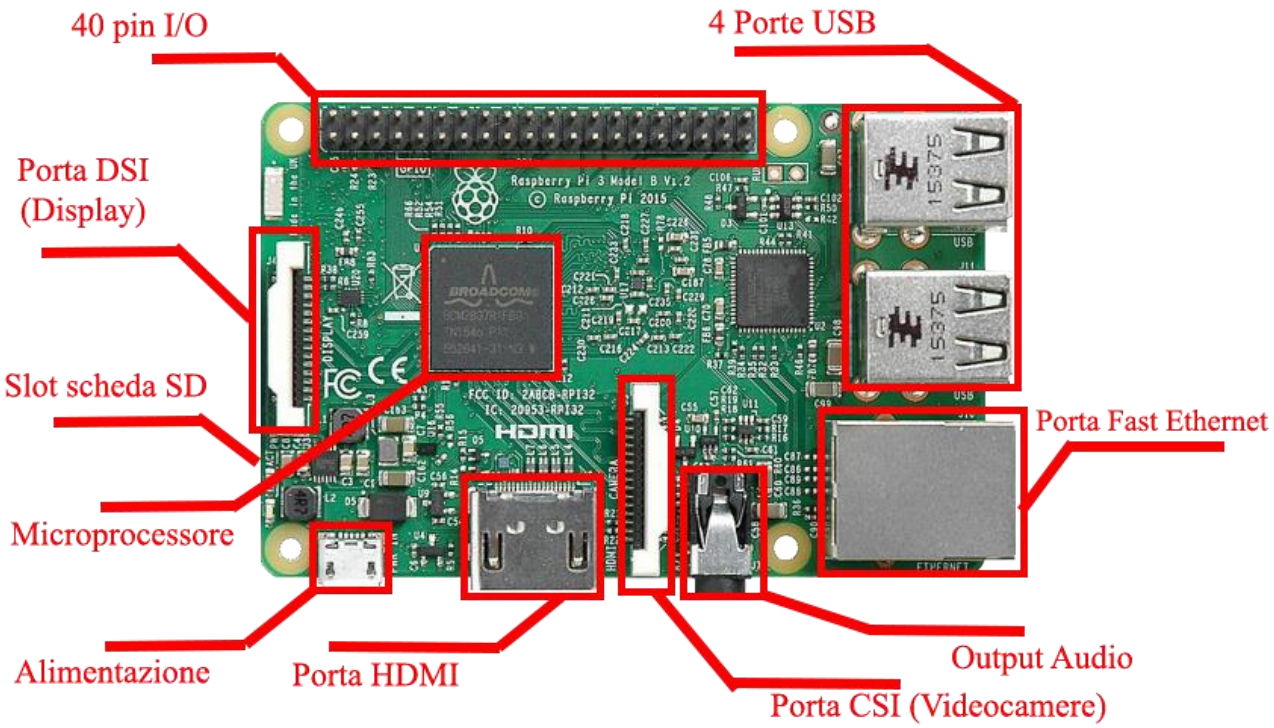
Difatti, Arduino è un elaboratore che legge un input e restituisce un output; può essere programmato usando l'Arduino Programing Language e l'Arduino Software (IDE), disponibili per Windows, Mac e Linux.

In questo progetto è stato utilizzato Arduino Mega 2560 Rev3, che offre maggiori risorse hardware rispetto alle versioni basilari: un microcontrollore più potente e una memoria più ampia.



2.1.2 Raspberry Pi

Raspberry Pi non è altro che un single-board computer, ovvero una scheda dalle dimensioni molto contenute, equipaggiata con un processore ARM a 1 Ghz, 1 GB di RAM ed una serie di ingressi e uscite per il controllo di circuiti elettronici.



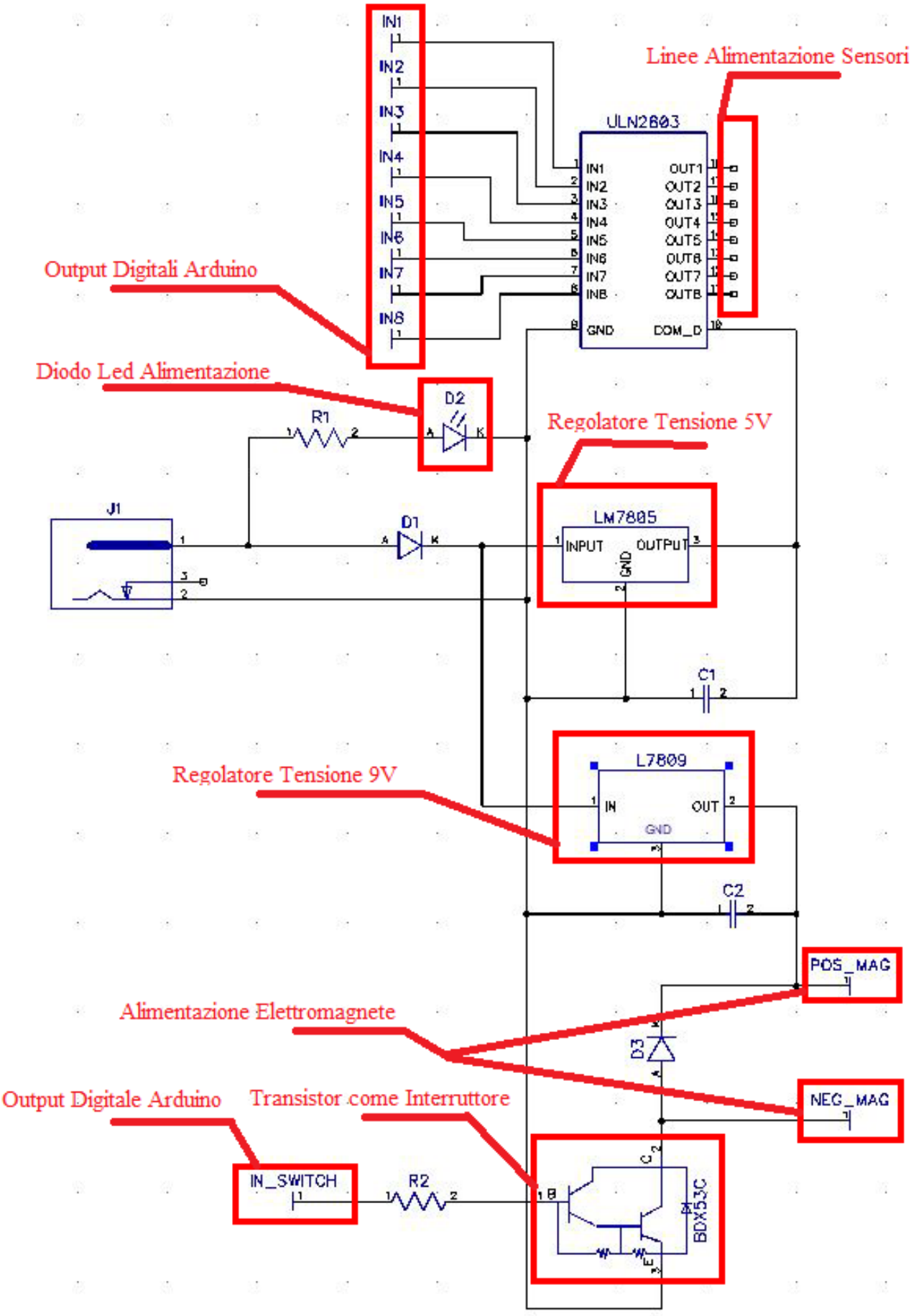
Questa scheda può permettere l'esecuzione di veri e propri sistemi operativi, comportandosi come un normale computer. Per questo progetto è stato scelto Raspbian, versione ad hoc della ben nota distribuzione Linux Debian, in quanto flessibile, veloce e open source.

Per poter visualizzare l'interfaccia grafica dell'applicativo Python di Odysseus, è stato acquistato uno schermo touch Kuman da 3.5 pollici, che supporta una risoluzione di 320 x 480 pixel, connesso al Raspberry direttamente grazie ai pin di Input/Output.

Inoltre, per rendere possibile l'interazione fra IA e attuatore, Arduino e Raspberry sono connessi tramite le rispettive porte USB, comunicandosi quando necessario mutamenti della situazione di gioco.

2.2 Scheda elettronica di controllo

Per poter gestire la componentistica elettronica, è stata creata su piastra forata una scheda che offra le funzionalità necessarie. Essa verrà alimentata da un alimentatore a 12V che può fornire fino a 5A, talvolta necessari per gestire i motori passo passo e l'elettromagnete; inoltre esso alimenterà anche Arduino.


Linee Alimentazione Sensori
Output Digitali Arduino
Diodo Led Alimentazione
Regolatore Tensione 5V
Regolatore Tensione 9V
Alimentazione Elettromagnete
Output Digitale Arduino
Transistor come Interruttore
IN1
IN2
IN3
IN4
IN5
IN6
IN7
IN8
ULN2803
OUT1
OUT2
OUT3
OUT4
OUT5
OUT6
OUT7
OUT8
GND
COM_D
D2
R1
J1
D1
LM7805
INPUT
OUTPUT
C1
L7809
IN
OUT
C2
POS_MAG
D3
NEG_MAG
IN_SWITCH
R2
BDX53C

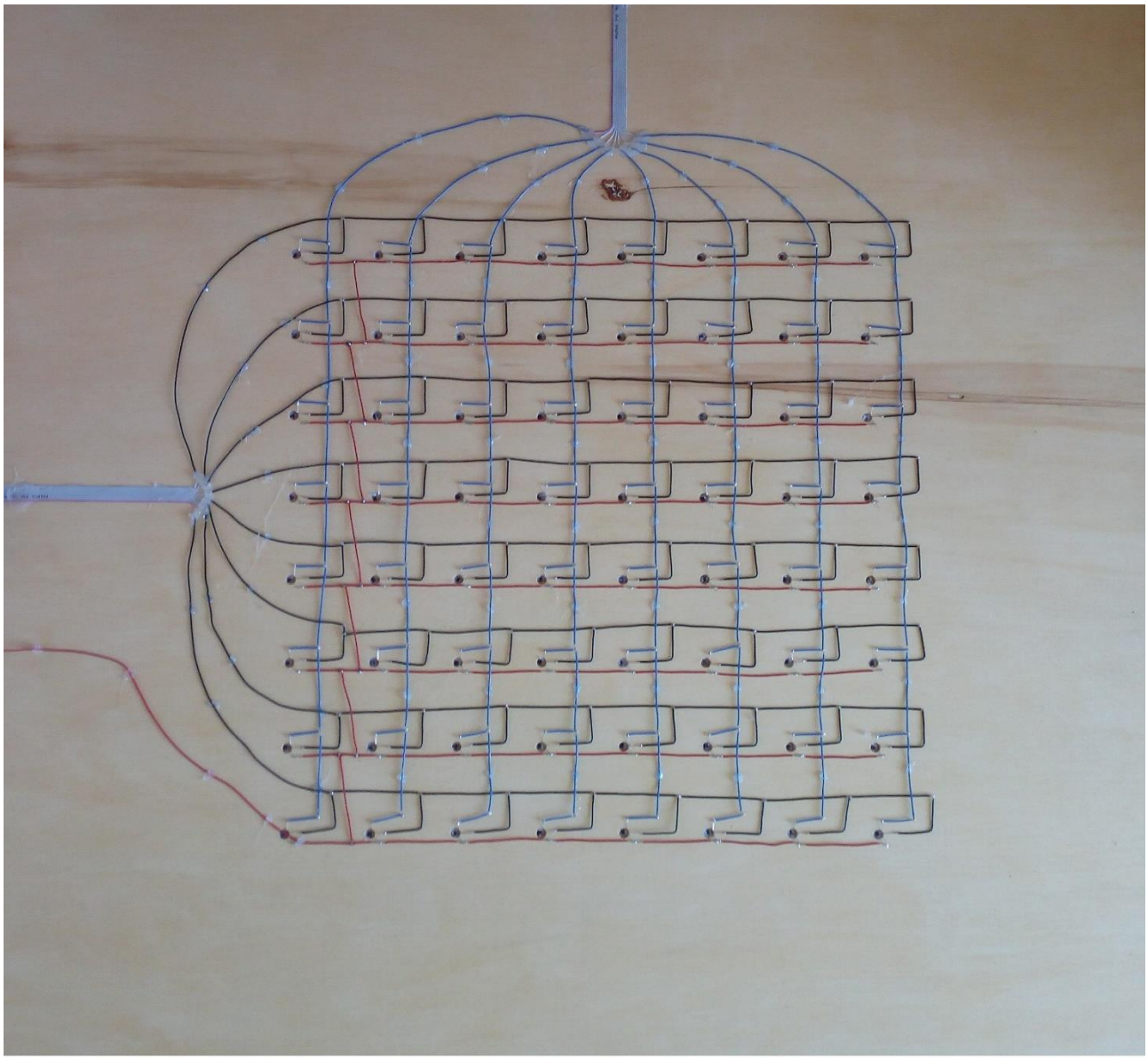
2.3 Rilevamento

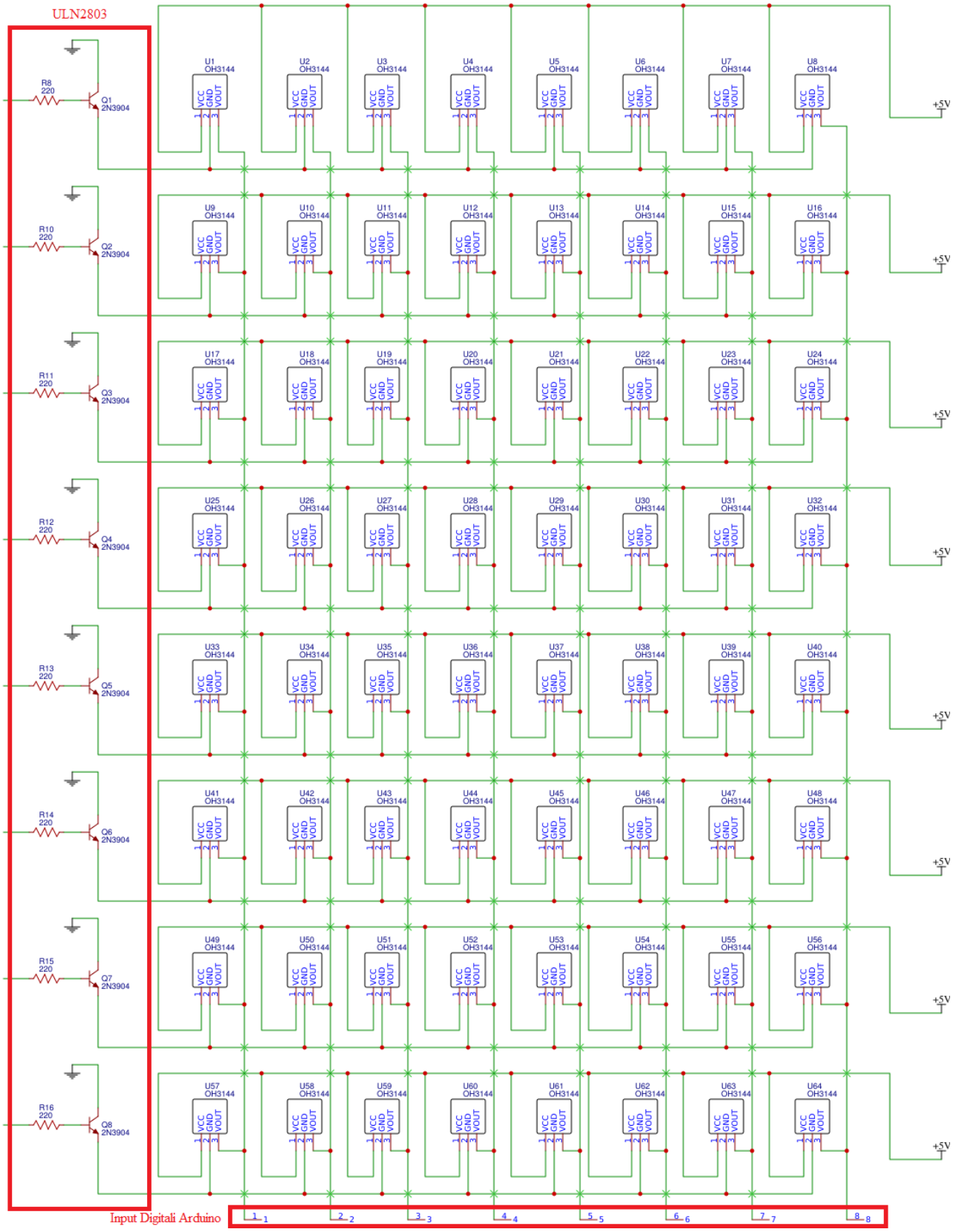
Di vitale importanza è il sistema di rilevamento dei pezzi mossi dal giocatore umano, realizzato tramite sensori di rilevamento dei campi magnetici posti sotto ogni cella della scacchiera. Difatti, sotto ogni scacco è stato inserito, in un'apposita cavità, un piccolo magnete al neodimio la cui funzionalità è doppia: permetterne il rilevamento del campo magnetico, ma allo stesso tempo agganciare e trascinare lo scacco con l'ausilio dell'elettromagnete.

Questo sistema verrà attivato nel momento in cui l'umano dovrà effettuare la propria mossa, controllando ciclicamente ogni cella per rilevare mutamenti. Il tutto è gestito da Arduino, che comunicherà eventuali aggiornamenti al Raspberry, per mantenere l'IA sempre aggiornata.

2.3.1 Matrice di rilevamento

Nello specifico, Arduino Mega non dispone di 64 pin di input digitali per poter leggere ogni sensore sotto alle celle, per cui è stata realizzata una matrice in cui si alimenta una sola riga per volta e si legge per colonna; un'analisi della figura (ecc.) chiarirà meglio la situazione.




ULN2803
R8 220
Q1 2N3904
R10 220
Q2 2N3904
R11 220
Q3 2N3904
R12 220
Q4 2N3904
R13 220
Q5 2N3904
R14 220
Q6 2N3904
R15 220
Q7 2N3904
R16 220
Q8 2N3904
U1 OH3144
U2 OH3144
U3 OH3144
U4 OH3144
U5 OH3144
U6 OH3144
U7 OH3144
U8 OH3144
U9 OH3144
U10 OH3144
U11 OH3144
U12 OH3144
U13 OH3144
U14 OH3144
U15 OH3144
U16 OH3144
U17 OH3144
U18 OH3144
U19 OH3144
U20 OH3144
U21 OH3144
U22 OH3144
U23 OH3144
U24 OH3144
U25 OH3144
U26 OH3144
U27 OH3144
U28 OH3144
U29 OH3144
U30 OH3144
U31 OH3144
U32 OH3144
U33 OH3144
U34 OH3144
U35 OH3144
U36 OH3144
U37 OH3144
U38 OH3144
U39 OH3144
U40 OH3144
U41 OH3144
U42 OH3144
U43 OH3144
U44 OH3144
U45 OH3144
U46 OH3144
U47 OH3144
U48 OH3144
U49 OH3144
U50 OH3144
U51 OH3144
U52 OH3144
U53 OH3144
U54 OH3144
U55 OH3144
U56 OH3144
U57 OH3144
U58 OH3144
U59 OH3144
U60 OH3144
U61 OH3144
U62 OH3144
U63 OH3144
U64 OH3144
VCC GND VOUT
+5V
Input Digitali Arduino
1 2 3 4 5 6 7 8

2.3.2 Componenti elettronici

Per quanto riguarda la parte di rilevamento degli spostamenti, questi sono i componenti più importanti che sono stati utilizzati:

- Interruttore ad effetto Hall A3144
- Array di transistor Darlington ULN2803APG
- Regolatore di tensione 7805

La lettura dei segnali dei sensori ad effetto Hall viene effettuata abilitando la resistenza di pullup interna ad Arduino da 10 kΩ, questo perché si evitano incongruenze legate a segnali flottanti qualora non fossero rilevati campi magnetici. La direttiva utilizzata è *INPUT_PULLUP*, nella fase di inizializzazione dei pin digitali della scheda.

I transistor presenti nell'integrato ULN2803APG sono stati utilizzati a livello pratico come interruttori, per abilitare o disabilitare il collegamento a massa delle linee di sensori della matrice di rilevamento.

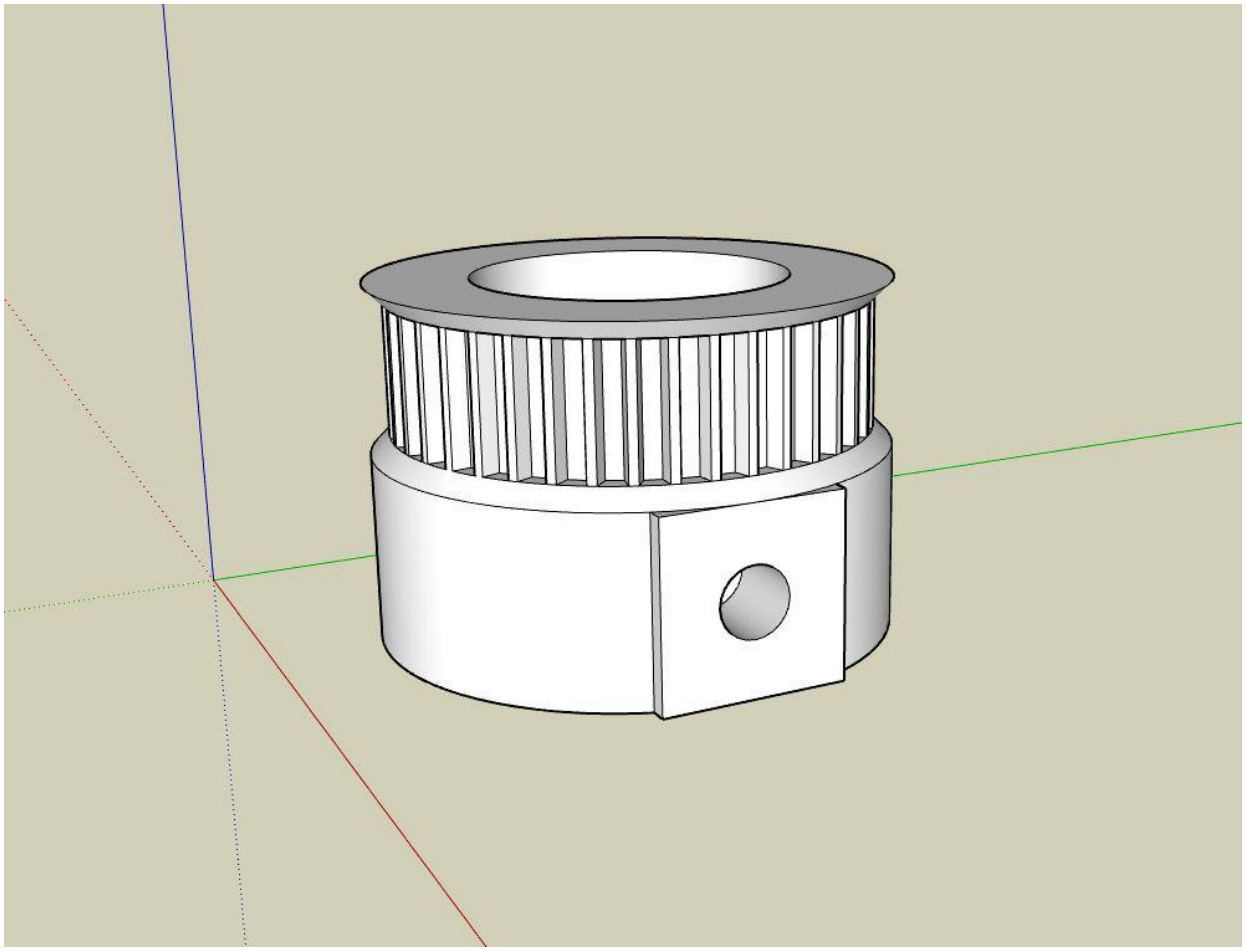
Il regolatore di tensione 7805, che come suggerisce il nome regola la tensione in uscita a 5V, è quello che si occupa di alimentare tutti i sensori ad effetto Hall e gli interruttori di inizio e fine corsa, il cui utilizzo è specificato più avanti. Ciò comporta un assorbimento di corrente importante, che ne causerebbe il surriscaldamento se non fosse per l'aletta di raffreddamento installata appositamente.

2.4 Spostamento

Il sistema di movimentazione dei pezzi si basa sul trascinamento di questi ultimi tramite un magnete posto sotto di essi e un elettromagnete, azionato al momento del bisogno, che si muove sotto al piano di gioco. Sostanzialmente, il movimento sull'asse delle ascisse avviene su due tubi in alluminio che corrono parallelamente, ma a una distanza di circa 85 cm di distanza. I pezzi che vi scorrono sopra sono trainati da una cinghia ciascuno, evitando puntellature grazie anche ad un albero di trasmissione. Chiaramente ad azionare il tutto ci sarà un motore elettronico particolare, che viene azionato a piccoli passi (o step), che corrisponderanno ad un dato spostamento lineare.

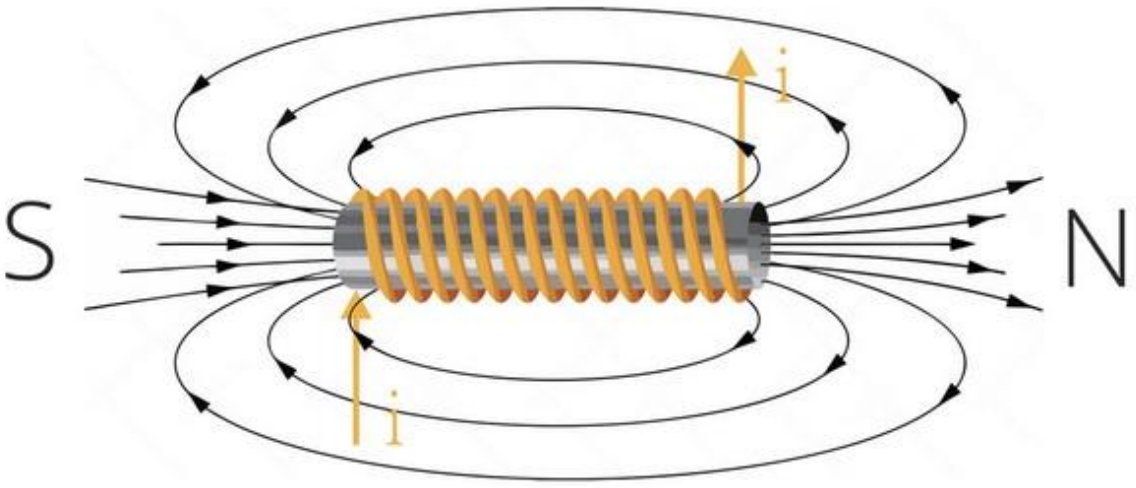
Sull'asse delle ordinate è presente un motore identico, che permetterà di muovere il pezzo dentro al quale è presente l'elettromagnete, questa volta tuttavia senza necessitare di un albero di trasmissione.

Infine, per rendere possibile la calibratura di questo sistema, sono stati aggiunti degli interruttori di inizio corsa e di fine corsa. Con questo metodo si evitano buona parte degli errori derivanti da passi saltati, o eventuali collisioni coi sostegni terminali che danneggerebbero il meccanismo.



2.4.1 Elettromagnete

Un elettromagnete è un magnete che, per essere attivato, sfrutta l'elettricità. Esso è composto da un nucleo di ferro dolce, avvolto da numerose spire metalliche. Le sue spire sono, in questo caso, realizzate in filo di rame in ragione della sua eccellente conduttività elettrica.



Dunque, quando le spire sono attraversate da una corrente, il solenoide genera un campo magnetico caratterizzato dalla presenza di due poli: nord e sud. Ciò è importante perché le linee di campo puntano in direzione opposta al polo nord, dirigendosi, senza mai incrociarsi, verso il polo sud. Pertanto, se azionato in maniera errata, l'elettromagnete potrebbe respingere gli scacchi invece di attrarli.

Per questo progetto è stato utilizzato un elettromagnete alimentato a 9V, che assorbe circa 1,2 A. Chiaramente un'uscita digitale di Arduino non può erogare così tanta corrente, per cui viene impiegato un transistor, il BDX53C, come interruttore. Inoltre, è stato aggiunto un diodo di ricircolo della corrente per evitare danneggiamenti dovuti al carico induttivo presente allo spegnimento dell'elettromagnete.

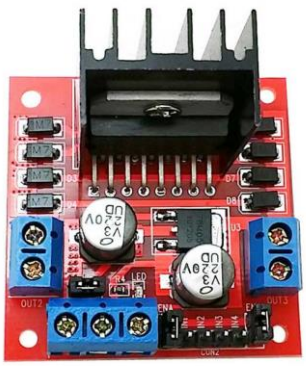
2.4.2 Motore passo passo

Negli ultimi anni, grazie anche alla diffusione di stampanti 3D, è divenuta sempre più comune una famiglia di motori in corrente continua: i motori passo passo. La peculiarità di questi motori è la capacità di suddividere la propria rotazione in un grande numero di passi (step). Per cui sono ideali per le applicazioni che necessitano di precisione nello spostamento e nella velocità di rotazione.

La scelta è ricaduta sui Quimat Nema 17, motori unipolari bifase, che alimentati a 12V assorbono 400 mA per fase. Per compiere un'intera rotazione devono fare 200 passi e hanno una coppia statica di 0,26 Nm.

Per pilotare questi stepper sono stati acquistati due driver appositi: Elegoo A4988 e L298N con doppio ponte H. Essi sono alimentati tramite la scheda di alimentazione precedentemente analizzata e le fasi sono comandate da Arduino.

Gli alloggiamenti di questi due motori sono differenti, perché quello che deve azionare anche l'albero di trasmissione è sostenuto da una staffa pre-acquistata che è fissata al piano di compensato, mentre per l'altro è stata realizzata una gabbia di bloccaggio su misura.



3 Architettura Software

3.1 Arduino come attuatore

Come già anticipato, Arduino agisce come attuatore rispetto alle direttive di Raspberry. Ciò si traduce a livello software in un'architettura a due strati, ideale per rendere il progetto scalabile e per frazionare la gestione delle problematiche.

3.1.1 Rilevamento

Su Arduino la porzione di codice che si occupa di rilevare spostamenti sulla scacchiera è caratterizzata da due cicli annidati, chiaramente quello esterno si occuperà di abilitare e disabilitare l'alimentazione delle linee di sensori, mentre quello interno di leggere colonna per colonna i segnali dei sensori. Inoltre, per ogni lettura, si confronta il nuovo dato con quello precedente e, se c'è un mutamento, si attendono 100 ms e si effettua una conferma di lettura.

3.1.2 Spostamento

Il codice che gestisce lo spostamento è molto semplice, perché si basa sul principio di poter regolare il punto di partenza dei pezzi ad ogni accensione, grazie agli interruttori di inizio corsa. Poi, avendo come dati la circonferenza delle pulegge utilizzate dai motori e il numero di passi necessari per fare un giro, si può calcolare a che spostamento lineare corrisponde un passo del motore.

Circonferenza Puleggia GT2 = 20 mm

Passi per rotazione completa motore = 200

Spostamento per passo = 20 mm : 200 = 0,1 mm per passo

3.2 Raspberry Pi come agente intelligente

Raspberry Pi agisce come componente "intelligente" nell'architettura ideata. Questo significa che tutta la sua potenza di calcolo verrà sfruttata per rendere l'IA che gioca a scacchi il più forte possibile.

3.2.1 Alberi di ricerca

Gli algoritmi di ricerca richiedono una struttura dati per tenere traccia dell'albero di ricerca costruito. Per ogni nodo *n* è stata costruita una struttura contenente i seguenti elementi:

- *n*.Stato: lo stato dello spazio degli stati a cui corrisponde il nodo, cioè una "fotografia" dei pezzi sulla scacchiera e di quelli catturati, oltre a quale giocatore spetta muovere
- *n*.Padre: il nodo dell'albero di ricerca che ha generato il nodo corrente
- *n*.Costo-Di-Cammino: il costo, tradizionalmente denotato con *g(n)*, del cammino che va dallo stato iniziale al nodo, come indicato dai puntatori verso i padri

Dati i componenti di un nodo padre, è facile calcolare i componenti necessari per un nodo figlio. La funzione Nodo-Figlio accetta un nodo genitore e un'azione, cioè una mossa da fare, e restituisce il nodo figlio risultante.

function Nodo-Figlio(*problema*, *padre*, *azione*) **returns** un nodo
    return un nodo con
        Stato = *problema*.Risultato(*padre*.Stato, *azione*),
        Padre = *padre*,
        Azione = *azione*,
        Costo-Di-Cammino = *padre*.Costo-Di-Cammino +
        problema.Costo-Di-Passo(*padre*.Stato, *azione*, Stato)

Finora non è stata fatta particolare attenzione a distinguere tra nodi e stati, ma tale distinzione è importante. Un nodo è una struttura dati utilizzata per rappresentare l'albero di ricerca, mentre uno stato corrisponde ad una particolare configurazione del mondo. Di conseguenza i nodi sono situati su particolari cammini definiti da puntatori Padre, a differenza degli stati. Inoltre, due nodi diversi possono contenere lo stesso stato del mondo, se questo è generato tramite due cammini di ricerca diversi.

Una volta ottenuti i nodi, serve una struttura adatta a memorizzarli. La frontiera deve essere memorizzata in modo da consentire all'algoritmo di ricerca di scegliere facilmente il successivo nodo da espandere in base alla propria strategia. La struttura dati adatta a questo scopo è la coda, che può essere soggetta alle seguenti operazioni:

- Vuota?(*coda*) restituisce true solo se non ci sono più elementi nella coda
- Pop(*coda*) rimuove il primo elemento della coda e lo restituisce
- Inserisci(*elemento, coda*) inserisce un elemento nella coda e restituisce la coda risultante

In Python questo meccanismo è stato realizzato tramite le liste, che implementano già numerose funzioni utili.

3.2.2 IA convenzionali

Nonostante differiscano nelle implementazioni, la maggior parte delle IA scacchistiche (fra cui quelle ai massimi livelli) si basano sugli stessi algoritmi. Esse sono tutte fondate sull'idea dell'applicazione dell'algoritmo Minimax su un albero a profondità limitata fissa, sviluppato inizialmente da John Von Neumann nel 1928 e poi adattato al problema degli scacchi da Claude E. Shannon nel 1950.

3.2.3 Algoritmo Minimax

Dato un albero di gioco, la strategia ottima può essere determinata esaminando il valore Minimax di ogni nodo, cioè Minimax(n). Il valore Minimax di un nodo corrisponde all'utilità per l'IA, di trovarsi nello stato corrispondente. Risulta quindi che per l'avversario umano la posizione preferibile sarebbe quella in cui il valore Minimax è minimo.

Il Minimax è un semplice algoritmo ricorsivo per dare un valore ad una determinata posizione, basato sull'assunto che l'avversario ragioni come noi, pertanto voglia vincere la partita come noi. La ricorsione percorre tutto l'albero fino alle foglie (nodi terminali), quindi i valori Minimax sono "portati su" attraverso l'albero durante la fase di ritorno.

Questa è una delle sue versioni più basilari:

```
function Minimax(posizione) returns un numero
    if posizione è vinta da IA then
        return 1
    elsif posizione è vinta da Umano then
        return -1
    elsif posizione è in pareggio then
        return 0

    migliorPunteggio = -∞

    for each a in Azioni(posizione) do
        sottoPunteggio = -Minimax(a)
        if sottoPunteggio > migliorPunteggio then
            migliorPunteggio = sottoPunteggio
    return migliorPunteggio
```

L'algoritmo Minimax esegue un'esplorazione completa in profondità dell'albero di gioco. Prendendo l'esempio degli scacchi, il fattore di ramificazione medio, cioè le possibili mosse attuabili da una posizione di partenza, è circa di 30. Una partita di scacchi si sviluppa, mediamente, su circa 100 turni, ciò significherebbe generare, memorizzare e analizzare 30^{100} posizioni, che è evidentemente un numero troppo grande persino per i moderni computer. Pertanto, per i giochi reali, il costo temporale e quello spaziale sono inaccettabili, ma quest'algoritmo fornisce la base per un'analisi matematica dei giochi e per sviluppare algoritmi più efficienti, che analizzino solo determinate parti dell'albero di gioco.

3.2.4 Ricerca a profondità limitata

L'approccio più comune per contrastare questo problema è la ricerca a profondità limitata, che consiste nel limitare artificiosamente quanto "lontano" guarderemo avanti nella partita. Così, quando siamo alla fine del sotto-albero che vogliamo esplorare, richiamiamo una funzione di valutazione statica che assegna un punteggio ad una posizione, analizzandola staticamente, cioè fornendo una stima senza analizzarne i futuri sviluppi.

Una semplice funzione di valutazione consiste nel sommare i pezzi da entrambe le parti, ognuno moltiplicato per una costante in base al tipo (Regina = 9, Torre = 5, ecc.). La funzione di valutazione è estremamente importante, ed è uno degli elementi che meglio decreta l'efficienza di un algoritmo scacchistico moderno.

```
function Minimax(posizione, profondità) returns un numero
    if posizione è vinta da IA then
        return 1
    elsif posizione è vinta da Umano then
        return -1
    elsif posizione è in pareggio then
        return 0

    if profondità = 0 then
        return Eval(posizione)

    migliorPunteggio = -∞

    for each a in Azioni(posizione) do
        sottoPunteggio = -Minimax(a, profondità - 1)
        if sottoPunteggio > migliorPunteggio then
            migliorPunteggio = sottoPunteggio
    return migliorPunteggio
```

Si possono notare 3 cambiamenti:

- Minimax() ha un parametro in più, *profondità*, che rappresenta il numero di livelli di profondità in cui vogliamo cercare
- Se siamo a profondità 0, viene richiamata la funzione di valutazione utilità e restituito il risultato
- Se siamo ad una profondità maggiore di 0, viene richiamato ricorsivamente il Minimax, passando come profondità per i sottoalberi l'attuale profondità -1

Adesso la ricerca terminerà in un tempo ragionevole, se la profondità fornita come parametro è a sua volta plausibile, ma si incorre in un problema spinoso: l'effetto orizzonte.

Nella versione precedente vengono tagliati tutti i rami alla stessa distanza dalla radice (effetto orizzonte). Questo si rivela pericoloso perché, per esempio, se in un braccio l'ultima posizione si rivela essere la cattura di un pedone da parte della regina, la funzione Eval() restituirebbe un valore piuttosto buono in quanto abbiamo catturato un pezzo avversario. Tuttavia, l'IA non si renderebbe conto del fatto che il pedone è difeso da un altro pedone, il quale catturerebbe la nostra regina al prossimo turno. Ciò si rivelerebbe un disastro.

Questo problema non può essere risolto semplicemente allontanando l'orizzonte dell'algoritmo, perché non importa quanto in profondità si cerca, ci sarà sempre un orizzonte ad ostacolarci. La soluzione è la ricerca "riposata". L'idea fondamentale è che una volta giunti all'orizzonte prestabilito, invece di richiamare Eval() e restituire il risultato, si entra in una modalità di ricerca peculiare, che espande solamente certi tipi di mosse, per richiamare Eval() una volta raggiunta una posizione "tranquilla".

La costruzione di una tipologia di ricerca di questa categoria è un problema annoso, che ancora oggi continua a far emergere opinioni divergenti in merito, pertanto esula dagli obiettivi di questo progetto.

3.2.5 Potatura α- β

Senza introdurre nessuna funzione euristica per risolvere il problema dell'orizzonte, uno dei modi per ottimizzare l'algoritmo Minimax è quello di introdurre la potatura alfa-beta. Invero, il problema della ricerca Minimax è che il numero degli stati da esaminare cresce esponenzialmente con la profondità dell'albero. Sfortunatamente non si può rendere questa crescita lineare, ma si può dimezzare l'esponente: infatti è possibile calcolare la decisione corretta senza guardare tutti i nodi dell'albero di gioco.

Il concetto di potatura è non troppo dissimile da quello che si ha in agricoltura: si tagliano dei rami che non concorrono attivamente alla produzione o alla crescita della pianta, per agevolare la crescita dei rami più proficui per i nostri scopi.

In altre parole, potando foglie o sottoalberi interi, si mantiene comunque il risultato Minimax della radice indipendente dai valori potati. Considerato un nodo n da qualche parte nell'albero, tale che uno dei due giocatori abbia la possibilità di muoversi in quel nodo, se c'è una scelte migliore m a livello del nodo padre o di un qualunque nodo precedente, allora n non sarà mai raggiunto in tutta la partita. Possiamo quindi potare n non appena avremo raccolto abbastanza informazioni (esaminando alcuni dei suoi discendenti) da raggiungere questa conclusione.

Essendo Minimax una ricerca in profondità, in ogni momento vanno considerati solamente i nodi lungo un singolo cammino dell'albero. La potatura alfa-beta prende il suo nome dai seguenti due parametri che descrivono i limiti sui valori "portati su" in un qualsiasi punto del cammino:

- α = il valore della scelta migliore (quella con valore più alto) per IA che abbiamo trovato finora in un qualsiasi punto di scelta lungo il cammino
- β = il valore della scelta migliore (quella con valore più basso) per Umano che abbiamo trovato finora in un qualsiasi punto di scelta lungo il cammino

La ricerca alfa-beta aggiorna i valori α e β a mano a mano che procede e pota i rami restanti che escono da un nodo (ovvero, fa terminare le chiamate ricorsive) non appena determina che il valore del nodo è peggio di quello di α per l'IA o, rispettivamente, di β per Umano.

3.2.7 Ordinamento delle mosse

L'efficacia della potatura alfa-beta dipende in gran parte dall'ordine in cui sono esaminati gli stati. Infatti, è molto più efficace analizzare i successori di un nodo che sembrano più promettenti; se ciò viene fatto correttamente si riesce veramente a dimezzare, come precedentemente asserito, l'esponente relativo al numero di stati Minimax da esaminare. Questo comporta, senza scendere

troppo in analisi di complessità spaziale e computazionale, la capacità di risolvere un albero profondo circa il doppio nello stesso lasso di tempo.

Per ordinare le mosse da analizzare si procede, in ambito scacchistico, in questo modo: prima si cerca di catturare i pezzi, poi di minacciarli, poi le mosse in avanti, infine quelle all'indietro.

3.2.8 Funzioni di valutazione dell'utilità

Una funzione di valutazione è una parte molto importante di un algoritmo scacchistico. Difatti, la maggior parte dei miglioramenti negli algoritmi scacchistici al giorno d'oggi proviene dal miglioramento delle rispettive funzioni di valutazione.

Quello che fa questa funzione è assegnare un punteggio statistico ad una posizione, senza analizzarne i successori. La maggior parte lavora calcolando valori separati per ogni caratteristica, combinandoli poi insieme per formare il valore finale.

Matematicamente si necessita di una funzione lineare pesata, che può essere espressa come

$$\text{Eval}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s) = \sum_{i=1}^{n} w_i f_i(s)$$

dove ogni w_i è un peso (weight) e ogni f_i è una caratteristica (feature) di una posizione.

Per Odysseus è stata creata in maniera tale che ogni f_i è il numero dei pezzi di un particolare tipo sulla scacchiera, w_i il loro valore.

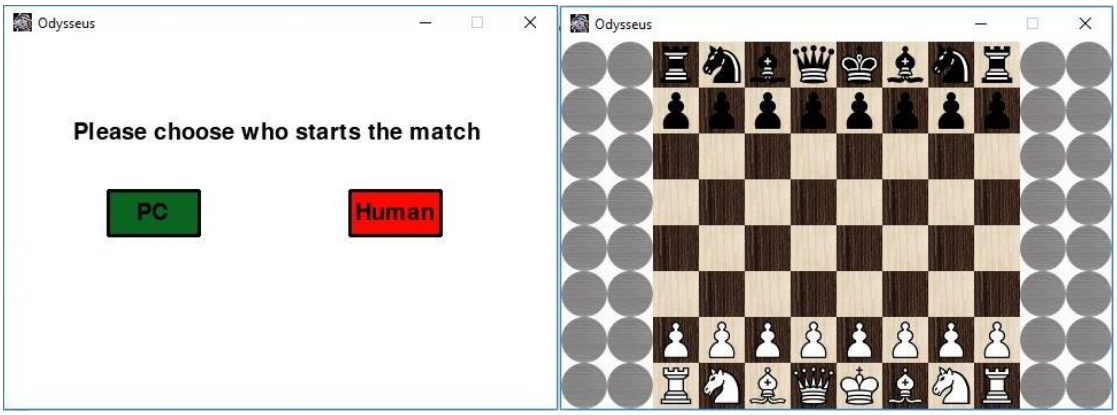
Pezzo	**Valore**
Re	Infinito
Regina	9
Torre	5
Alfiere	3
Cavallo	3
Pedone	1

3.2 Interfaccia grafica

Per realizzare un applicativo facilmente fruibile ho scelto di creare un'interfaccia grafica visibile tramite il piccolo schermo su Raspberry Pi. Ho utilizzato il framework PyGame, che offre varie funzionalità implementabili facilmente.

Prima di accendere la scacchiera automatizzata vanno posizionati tutti i pezzi al centro delle rispettive celle sul piano di gioco.

L'applicativo si avvia subito dopo il caricamento del sistema operativo e richiede inizialmente solo l'inserimento di quale giocatore dovrà iniziare la partita.



Successivamente verrà visualizzata la situazione iniziale della scacchiera e, in base a chi deve iniziare a muovere, sarà il computer ad attendere una mossa umana o il computer caricherà una delle aperture preimpostate, attuando la prima mossa.

Ad ogni mutamento della zona di gioco verrà visualizzato il corrispettivo su schermo, con tanto di messaggi che comunicano l'inizio e la fine di un turno. La cattura di un pezzo da parte del giocatore umano potrebbe, talvolta, generare errori, per cui l'applicativo chiede in quei casi la conferma di corretta analisi della situazione di gioco.

Invece, nel caso in cui un pedone del giocatore umano giungesse a fine scacchiera, avrebbe la possibilità di essere sostituito con uno dei pezzi catturati dall'IA, per cui va comunicata la scelta di quale pezzo è stato scelto.

Qualora vi fosse una situazione particolare, come lo scacco al re, lo scacco matto, o una delle condizioni di vittoria e/o pareggio, verrà visualizzato l'apposito messaggio a schermo.

Per le regole di gioco si è fatto affidamento al regolamento della Federazione Scacchistica Italiana.

4 Conclusioni

4.1 Risultati

Il progetto ha raggiunto e superato le aspettative iniziali, essendo i mezzi e le conoscenze limitate. Alcuni elementi della parte fisica del progetto andrebbero ingegnerizzati in maniera molto più approfondita di come è stato fatto finora. Difatti è mancata la preparazione teorica necessaria a realizzare un sistema di spostamento veramente efficiente. Inoltre, i pezzi stampati in 3D sono stati disegnati in buona parte con un SketchUp 2017 che, come il disegnatore, si è rivelato inadatto a soddisfare l'elevato livello di accuratezza necessario.

Dal punto di vista elettronico sono stati realizzati collegamenti alle volte poco eleganti, oltre alla scelta di usare la piastra forata, che non è sicuramente l'ideale per una versione definitiva.

L'aspetto software, invece, è stato sicuramente curato in maniera più perfezionista, anche se Python stesso e Raspberry Pi non sono scelte ottimali per l'esecuzione di algoritmi che richiederebbero capacità computazionali, velocità e memoria notevoli. Tuttavia, con qualche rinuncia è stato possibile coprire ogni aspetto significativo di un applicativo software di questo tipo.

Per completezza, tutto il materiale elaborato per il progetto è disponibile liberamente su un repository Github raggiungibile a quest'indirizzo: www.github.com/eneavignoli/Odysseus

4.2 Sviluppi futuri

In molti modi si potrebbe migliorare Odysseus, sia a livello hardware che a livello software, disponendo di tempo e risorse maggiori.

Partendo dall'aspetto hardware, si potrebbe studiare una soluzione che permetta di isolare parte delle linee dei campi magnetici che lateralmente portano gli scacchi a respingersi e che, di conseguenza, hanno portato ad un ampliamento del piano di gioco. Ciò permetterebbe di ridimensionare quest'ingombrante meccanismo, riducendo gli attriti anche grazie all'utilizzo di cuscinetti per far scorrere i pezzi sui tubi di alluminio.

Inoltre, il circuito su scheda perforata potrebbe essere professionalizzato con la creazione di un PCB.

Mentre l'aspetto software potrebbe seguire due differenti strade: l'evoluzione sempre su Raspberry Pi utilizzando Python, oppure la totale reingegnerizzazione utilizzando una piattaforma più potente ed un linguaggio veloce a basso livello, come il C. In entrambi i casi si potrebbero aggiungere alcune funzionalità tutt'ora mancanti, come l'en passant e l'arrocco.

L'IA potrebbe essere ottimizzata affinando la funzione Eval() con parametri che tengano conto della posizione in campo dei pezzi e della loro mobilità, non solo della quantità. Per evitare sprechi di memoria dovuti alla ridondanza di stati si potrebbe aggiungere una tabella hash che ne tenga traccia.

5 Fonti

5.1 Bibliografia

1. Stuart Russell, Peter Norvig - Intelligenza Artificiale, Un approccio moderno – Volumi 1 e 2. Pearson Education, Prentice-Hall, 2010. Terza edizione italiana a cura di Francesco Amigoni.
2. Claude E Shannon - Programming a computer for playing chess. The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, 1950.
3. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein – Introduzione agli algoritmi e strutture dati. McGraw-Hill, 2010.
4. John von Neumann - Zur Theorie der Gesellschaftsspiele, 1928.
5. Louis Victor Allis - Searching for Solutions in Games and Artificial Intelligence, 1994.

5.2 Sitografia

1. Sito web ufficiale di Arduino. www.arduino.cc
2. Sito web ufficiale di Raspberry Pi. www.raspberrypi.org
3. Elettronica Open Source. it.emcelettronica.com
4. Istituto Europeo del Rame. www.copperalliance.it
5. Motori Passo Passo. www.motoripassopasso.it
6. Federazione Scacchistica Italiana. www.federscacchi.it
7. Stockfish AI scacchistica open source. stockfishchess.org