ELABORATO ESAME DI MATURITÀ

Modello di un Parcheggio Automatizzato

5B AUTOMAZIONE 2020/2021
ANTONIO BERNARDINI
IIS Enzo Ferrari

Sommario

Sistema di Acquisizione ed Elaborazione Dati

Un settore importante dell'Elettronica è quello che si occupa dell'acquisizione e dell'elaborazione dei segnali analogici dipendenti da grandezze fisiche di varia natura, al fine di effettuare il controllo della grandezza in esame o anche semplicemente la visualizzazione, la memorizzazione o la trasmissione dei valori che essa assume.

Acquisizione

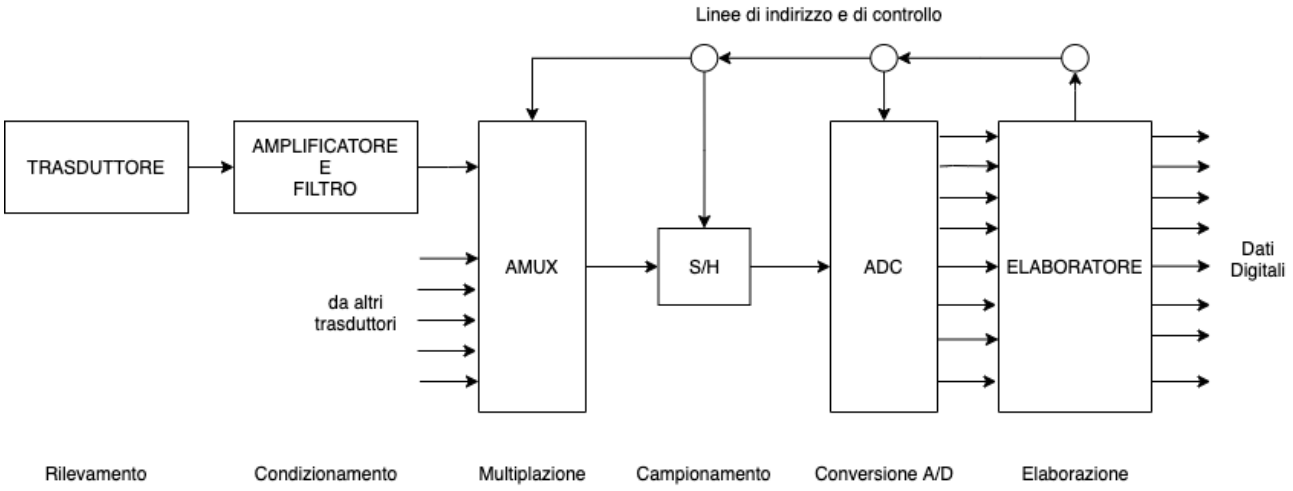


Figura 1

Il primo elemento da considerare è il trasduttore, la cui funzione tipica è di fornire in uscita una grandezza elettrica di valore proporzionale all'entità o alla variazione della grandezza fisica in esame. Per esempio una termocoppia fornisce una tensione proporzionale alla temperatura. I segnali generati dai trasduttori di solito devono essere condizionati in modo che il trasferimento dell'informazione possa avvenire con le caratteristiche di precisione, linearità, immunità al rumore, isolamento elettrico richieste per una data applicazione. Il blocco circuitale di condizionamento viene generalmente realizzato mediante amplificatori, talvolta anche molto sofisticati, e filtri.
La crescente diffusione dei microprocessori e microcontrollori e in genere dei sistemi programmabili fa preferire soluzioni di tipo digitale, in quanto queste offrono prestazioni decisamente superiori per quanto riguarda l'immunità al rumore e consentono di effettuare elaborazioni anche molto sofisticate e di modificare il tipo e i parametri dell'elaborazione stessa semplicemente intervenendo sul programma.
Pertanto, i segnali analogici, opportunamente condizionati, vengono trattati da convertitori analogico-digitale (ADC: analog to digital converter). Essi forniscono in uscita stringhe di bit che rappresentano numeri proporzionali ai valori del segnale analogico di ingresso.
Parametri fondamentali per i convertitori A/D sono, oltre al numero di bit di uscita, l'escursione massima del segnale di ingresso e il tempo di conversione, ovvero il tempo richiesto affinché a un segnale stabile di ingresso corrisponda un valore numerico stabile di uscita.

In figura 1, il convertitore A/D è preceduto da due blocchi, il multiplatore analogico (AMUX: analog multiplexer) e il circuito di campionamento e mantenimento (S/H: sample and hold) che svolgono compiti ben precisi e l'opportunità del loro impiego deve essere valutata in funzione dell'applicazione specifica.
Il "multiplexer" seleziona, a seconda del codice presente sulle linee digitali di indirizzo, uno solo dei segnali analogici di ingresso, trasferendolo in uscita; in tal modo è possibile trattare più segnali indipendenti impiegando lo stesso convertitore.
Il circuito "sample and hold" risponde alle esigenze di campionare, in un tempo relativamente breve, il segnale analogico da convertire e di mantenerlo stabile per tutta la durata della conversione.
Il blocco indicato come "elaboratore" rappresenta dispositivi di varia complessità: potrebbe essere un processore che memorizza i dati o più semplicemente potrebbe essere un sistema di visualizzazione. In ogni caso, dovranno essere previste le opportune interfacce e le linee di controllo per l'acquisizione dei dati digitali.

Distribuzione

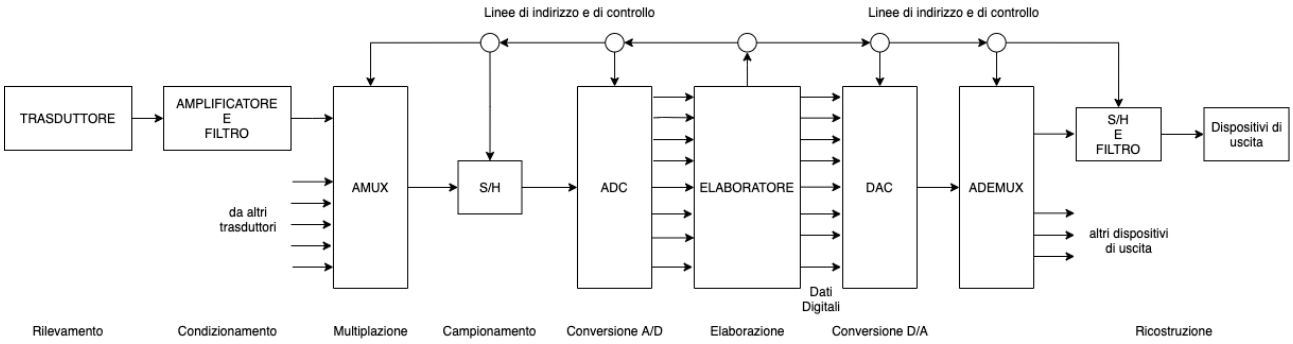
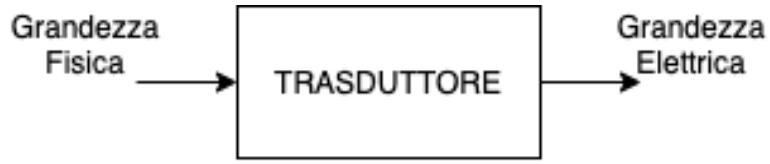


Figura 2

Dopo che i dati analogici sono stati convertiti in forma digitale e sono stati memorizzati o elaborati, spesso il risultato delle elaborazioni deve nuovamente interagire con il "mondo esterno".
I dati di uscita del sistema di elaborazione possono essere utilizzati, in forma digitale o analogica, per azionare motori, accendere lampade o riscaldatori, far suonare allarmi, visualizzare informazioni, ecc…, sia localmente sia a distanza. Spesso il risultato delle elaborazioni viene utilizzato per influenzare la grandezza rilevata all'inizio della catena di acquisizione, realizzando in tal modo un sistema di controllo ad anello chiuso. Il trasferimento dei dati di uscita, sia digitali sia analogici, viene comunemente indicato con il termine distribuzione.
In figura 2, i dati forniti dall'elaboratore su otto linee parallele vengono convertiti in forma analogica dal convertitore digitale-analogico (DAC: digital to analog converter). Il segnale di uscita del DAC viene mandato all'ingresso di un demultiplatore analogico (ADEMUX: analog demultiplexer) che lo trasferisce all'uscita selezionata dal codice presente sulle linee di indirizzo.
Naturalmente la sincronizzazione del convertitore e del demultiplatore dovrà essere molto precisa in modo che la distribuzione dei segnali avvenga correttamente. Infine

potranno essere presenti circuiti S/H e filtri che consentano un'adeguata ricostruzione dei segnali analogici dopo la conversione e la distribuzione.

Trasduttori



In un sistema di acquisizione dati, i trasduttori sono dispositivi che forniscono in uscita una grandezza elettrica funzione della grandezza fisica da rilevare.
La varietà di trasduttori è molto ampia, essi possono essere classificati considerando la grandezza fisica che sono in grado di rilevare o la grandezza elettrica che forniscono in uscita oppure il principio fisico su cui si basa il loro funzionamento.
Una suddivisione comune è quella che distingue i trasduttori attivi da quelli passivi. I primi, che comprendono per esempio i trasduttori piezoelettrici e le termocoppie, per effetto della grandezza fisica applicata in ingresso generano autonomamente un segnale di tensione o di corrente. I trasduttori passivi invece, per essere utilizzati richiedono una sorgente di alimentazione esterna, chiamata eccitazione: la grandezza fisica di ingresso produce la variazione di un parametro elettrico, per esempio resistenza, capacità, induttanza la cui entità controlla il segnale di uscita. Fra i dispositivi di questo tipo, noti anche come trasduttori regolatori, si possono citare gli estensimetri e i termistori.

Condizionamento del segnale

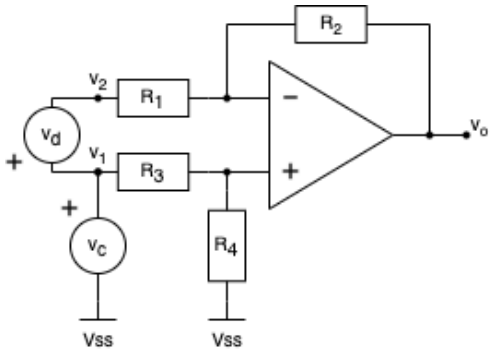
I segnali forniti dai trasduttori sono in genere di ampiezza modesta mentre i convertitori sono in grado di convertire tensioni dell'ordine dei volt (con valori di fondo scala $V_{FS} = \pm 5{,}5\ V \div 10\ V$); conviene pertanto provvedere a un'adeguata amplificazione del segnale in modo che la sua escursione sia compatibile con i valori di tensione ammessi dal convertitore e si avvicini al valore della tensione di fondo scala. Per esempio il segnale di un trasduttore con escursione $0 \div 100\ mV$, che debba essere convertito da un ADC con tensione di fondo scala $V_{FS} = 10\ V$, dovrà essere amplificato di un fattore 100. In questo modo si ottiene una migliore precisione e una maggiore sensibilità del sistema di acquisizione complessivo. Normalmente l'amplificatore deve avere elevata resistenza di ingresso in modo da non caricare il circuito del trasduttore.

In generale i circuiti di condizionamento hanno diversi scopi:

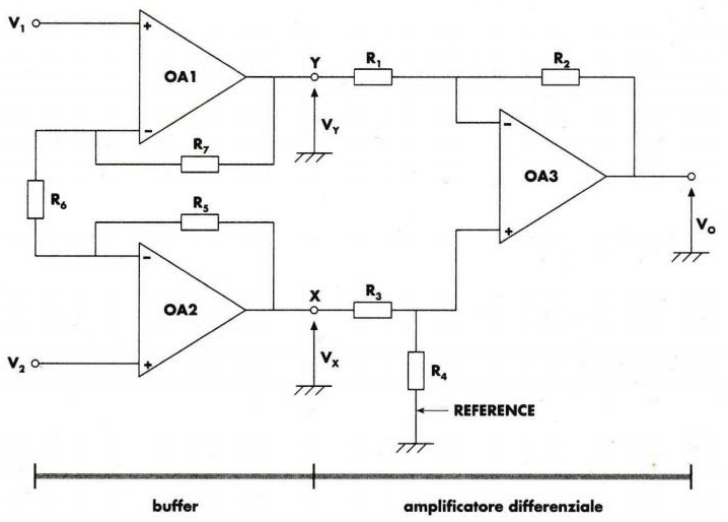
1) Amplificare il segnale di uscita dal trasduttore per adeguarlo ai valori compatibili con l'ADC;

2) Traslare il livello del segnale in uscita dal trasduttore per compensare eventuali livelli indesiderati di tensione continua;
3) Filtrare i disturbi a frequenza maggiore di quella del segnale in uscita dal trasduttore;
4) Limitare la banda di frequenza del segnale in uscita dal trasduttore.

Amplificatori da strumentazione



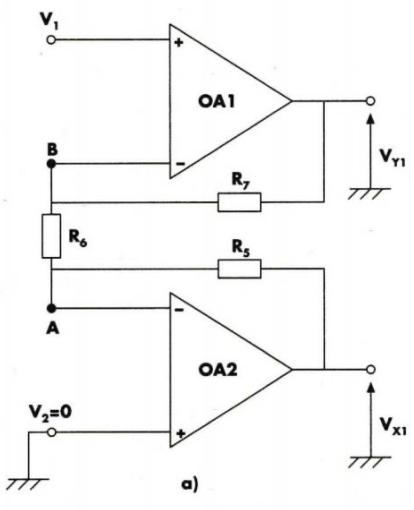
Idealmente il guadagno di un amplificatore differenziale dipende solo dal rapporto R_2/R_1, che deve essere uguale a R_4/R_3 e il $CMRR$ (A_d/A_c) è infinito. In realtà la tensione di modo comune, rappresentata dal generatore v_c, influisce sul segnale di uscita. Ciò è dovuto soprattutto al fatto che una non perfetta uguaglianza fra R_2/R_1 e R_4/R_3 fa sì che l'effetto della tensione v_c non sia più bilanciato e che quindi il $CMRR$ globale del circuito sia inferiore al $CMRR$ intrinseco dell'operazionale. Il $CMRR$ si degrada con lo sbilanciamento dei rapporti R_2/R_1 e R_4/R_3 e peggiora ancora per effetto delle derive termiche, oltre a dipendere dalla frequenza.
Occorre inoltre sottolineare l'inconveniente della limitata resistenza vista dagli ingressi, che vale $R_3 + R_4$ per il segnale v_1 e R_1 per il segnale v_2. Tale disparità produce un ulteriore sbilanciamento della configurazione.
Questi problemi vengono risolti in gran parte utilizzando amplificatori differenziali integrati, chiamati amplificatori da strumentazione, la cui struttura è rappresentata in figura:



Si nota la presenza dell'amplificatore differenziale preceduto da un buffer che serve a disaccoppiare l'amplificatore differenziale dai segnali in ingresso. In questo modo ogni segnale è applicato all'ingresso non invertente di un amplificatore operazionale che ha una resistenza d'ingresso di valore molto alto.

Se ipotizziamo rispettata la condizione $\frac{R_2}{R_1} = \frac{R_4}{R_3}$, con riferimento alle tensioni nei punti X e Y la tensione di uscita dell'amplificatore differenziale è data dalla seguente formula:

$$V_o = \frac{R_2}{R_1} \cdot (V_X - V_Y)$$

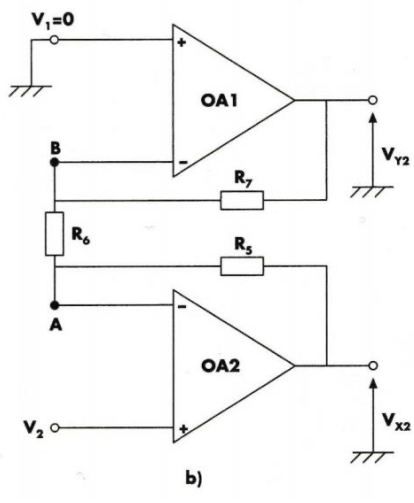


Per calcolare le tensioni V_X e V_Y possiamo utilizzare il principio di sovrapposizione degli effetti applicando uno alla volta i segnali di ingresso V_1 e V_2.
Quando $V_1 \neq 0$ e $V_2 = 0$, considerando gli amplificatori operazionali ideali, la tensione V_{Y1} può essere calcolata applicando il principio di massa virtuale al secondo operazionale, così da avere una tensione $V_A = V_2 = 0$. Quindi, considerando il primo operazionale in configurazione non invertente, si ottiene:

$$V_{Y1} = \left(1 + \frac{R_7}{R_6}\right) \cdot V_1$$

La tensione V_{X1}, invece, applicando il principio di massa virtuale al primo operazionale, così da avere una tensione $V_B = V_1 = V_A$. Quindi, considerando il secondo operazionale in configurazione invertente, si ottiene:

$$V_{X1} = -\frac{R_5}{R_6} \cdot V_1$$



Gli effetti della tensione V_2, possono essere calcolati con considerazioni analoghe a quelle precedenti risultando:

$$V_{X2} = \left(1 + \frac{R_5}{R_6}\right) \cdot V_2$$

$$V_{Y2} = -\frac{R_7}{R_6} \cdot V_2$$

Per calcolare la tensione V_X occorre sommare i contributi delle tensioni V_{X1}e V_{X2}:

$$V_X = V_{X1} + V_{X2} = -\frac{R_5}{R_6} \cdot V_1 + \left(1 + \frac{R_5}{R_6}\right) \cdot V_2$$

e allo stesso modo per calcolare la tensione V_Y occorre sommare i contributi delle tensioni V_{Y1} e V_{Y2}:

$$V_Y = V_{Y1} + V_{Y2} = \left(1 + \frac{R_7}{R_6}\right) \cdot V_1 - \frac{R_7}{R_6} \cdot V_2$$

Quindi sostituendo nella formula $V_o = \frac{R_2}{R_1} \cdot (V_X - V_Y)$ i valori delle tensioni V_X e V_Y si ottiene:

$$V_o = \frac{R_2}{R_1} \cdot (V_X - V_Y) \Rightarrow$$

$$\Rightarrow V_o = \frac{R_2}{R_1} \cdot \left(-\frac{R_5}{R_6} \cdot V_1 + \left(1 + \frac{R_5}{R_6}\right) \cdot V_2 - \left(1 + \frac{R_7}{R_6}\right) \cdot V_1 + \frac{R_7}{R_6} \cdot V_2\right) \Rightarrow$$

$$\Rightarrow V_o = \frac{R_2}{R_1} \cdot \left(-V_1 \cdot \left(1 + \frac{R_5}{R_6} + \frac{R_7}{R_6}\right) + V_2 \cdot \left(1 + \frac{R_5}{R_6} + \frac{R_7}{R_6}\right)\right) \Rightarrow$$

$$\Rightarrow V_o = \frac{R_2}{R_1} \cdot \left(1 + \frac{R_5}{R_6} + \frac{R_7}{R_6}\right) \cdot (V_2 - V_1)$$

Infine, usando un solo valore resistivo R per tutto il circuito e un potenziometro R_6 è possibile regolare il valore dell'amplificazione variando, a scelta, il valore ohmico del potenziometro. Da questa considerazione ne segue che il guadagno dell'amplificatore da strumentazione è il seguente:

$$A = \frac{V_o}{(V_2 - V_1)} = \frac{R_2}{R_1} \cdot \left(1 + \frac{R_5}{R_6} + \frac{R_7}{R_6}\right) = \frac{R}{R} \cdot \left(1 + \frac{R}{R_6} + \frac{R}{R_6}\right) = \left(1 + \frac{2 \cdot R}{R_6}\right)$$

Filtri

Nei sistemi di acquisizione dati i filtri sono principalmente di tipo passa-basso. Infatti, mentre i segnali forniti dai trasduttori sono di solito lentamente variabili, con larghezza di banda spesso inferiore a $10\,Hz$, i disturbi che a essi si sovrappongono sono generalmente di frequenza maggiore. Inoltre, operando su segnali che dovranno essere campionati e convertiti in forma digitale, è quasi sempre necessario limitarne la banda

di frequenza ad un valore inferiore alla metà della frequenza di campionamento al fine di evitare un'errata acquisizione del segnale.
La scelta del filtro dovrà essere effettuata tenendo conto della risposta e della pendenza desiderate, del tipo di segnale (continuo, sinusoidale, …) e della sua frequenza, dell'errore (dovuto all'attenuazione e allo sfasamento delle varie armoniche) che il filtro introduce nel segnale. Anche la frequenza di taglio dovrà essere scelta in modo da ottimizzare le prestazioni; in un filtro passa-basso dovrà essere più bassa possibile ma tale da consentire il trasferimento del segnale con un errore accettabile.

Conversione A/D e D/A

Come già descritto, per il trattamento dei segnali vengono preferite quasi sempre soluzioni di tipo digitale. È quindi necessario, in fase di acquisizione, impiegare dispositivi che convertano i segnali analogici in forma digitale o numerica. D'altra parte, i risultati delle elaborazioni, costituiti da informazioni digitali, devono in genere essere convertiti in forma analogica per interagire con il mondo esterno.

Quantizzazione

Il processo di digitalizzazione dei segnali analogici introduce il concetto di quantizzazione. Infatti mentre un segnale analogico può assumere infiniti valori in un campo continuo, la sua rappresentazione digitale può assumere soltanto un numero finito di valori discreti.
Gli infiniti valori del segnale analogico devono pertanto essere quantizzati ovvero raggruppati in un certo numero di fasce delimitate da livelli fissi detti livelli di quantizzazione; a ciascuna fascia di valori analogici corrisponderà un valore digitale.
La distanza fra due livelli di quantizzazione contigui costituisce il passo di quantizzazione Q,a cui corrisponde il valore del bit meno significativo (LSB: least significant bit).
Un dato digitale a n bit può esprimere 2^n valori; il valore digitale 2^n viene pertanto associato al valore di fondo scala (FS: full scale range) della grandezza analogica. Conseguentemente il valore analogico corrispondente al bit meno significativo sarà $Q = FS/2^n$.
Per esempio, un convertitore A/D con tre bit di uscita potrà quantizzare il segnale di ingresso con $2^3 = 8$ valori, essendo solo otto le possibili combinazioni di tre bit. Se lo stesso convertitore ha un fondo scala $FS = 8\,V$, il passo di quantizzazione, pari al valore dell'LSB, è $1\,V$.

In figura 3, sono illustrati un segnale a rampa v_a, variabile da $0\,V$ a $7{,}5\,V$, con i corrispondenti valori digitali e una forma d'onda a gradinata v_a' che si otterrebbe riconvertendo i valori digitali. Come si vede, per tutti i valori di v_a compresi fra $2{,}5\,V$

e 3,5 V, il valore binario corrispondente è 011 che, riconvertito, fornirebbe $v_a' = 3\ V$. Così per tutti i valori compresi fra 0 V e 0,5 V, il valore digitale corrispondente è 000. Dunque l'errore ε che si commette nella quantizzazione è sempre minore o uguale a $\pm 0{,}5\ V$, cioè:

$$\varepsilon \leq {}^{1}/_{2} \cdot LSB$$

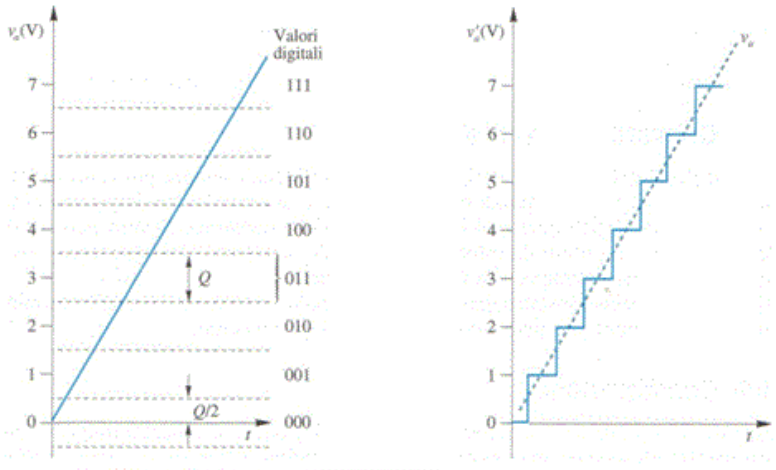


Figura 3

Se invece si disponessero i livelli di quantizzazione a $0\ V, 1\ V, \ldots, 7\ V$, associando al valore digitale 000 la fascia di valori analogici da 0 V a 1 V, a 001 i valori da 1 V a 2 V, a 111 i valori da 6 V a 7 V, l'errore di quantizzazione risulterebbe maggiore ossia si avrebbe:

$$\varepsilon \leq 1 \cdot LSB$$

In un convertitore analogico-digitale i valori digitali di uscita non riproducono fedelmente il segnale di ingresso ma ne danno una rappresentazione approssimata, tanto più precisa quanto minore è il passo di quantizzazione Q,cioè quanto più numerosi sono i livelli di quantizzazione. Questi ultimi, d'altra parte, sono legati al numero di bit utilizzati per la rappresentazione digitale e quindi sono necessariamente in numero limitato. Sono comuni convertitori A/D con uscite a $8, 10, 12, 16$ bit che consentono, rispettivamente, $2^8 = 256, 2^{10} = 1024, 2^{12} = 4096, 2^{16} = 65536$ livelli di quantizzazione.
Il numero di bit di un convertitore A/D viene generalmente chiamato risoluzione poiché implicitamente indica qual è la minima variazione del segnale di ingresso che può essere rilevata in uscita (pari a $FS/2^n$ per un convertitore a n bit).

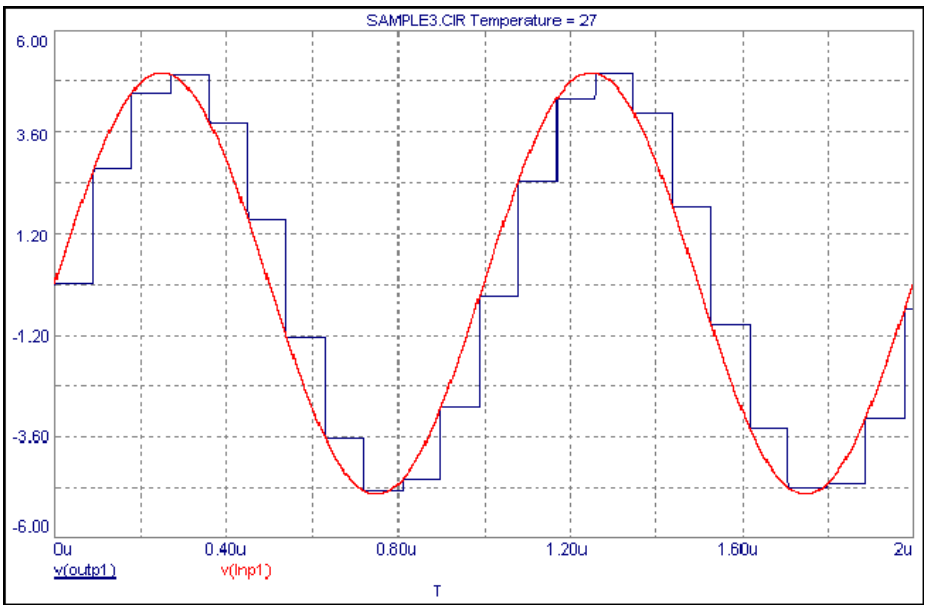
Campionamento

Un concetto implicito nella conversione A/D è quello del campionamento del segnale in vari instanti successivi. La conversione consiste nel prelevamento di un campione del segnale a un dato istante e nella determinazione del corrispondente valore digitale,

che resterà fisso finché non verrà prelevato un altro campione per una nuova conversione.
La frequenza con cui il segnale viene prelevato è detta frequenza di campionamento; essa ha un'importanza fondamentale in riferimento al contenuto informativo del segnale campionato e alle possibilità di ricostruire fedelmente il segnale analogico originario.
Il teorema del campionamento, noto anche come teorema di Shannon-Nyquist, stabilisce che la frequenza di campionamento deve essere maggiore o uguale al doppio di quella della componente di frequenza più elevata del segnale in esame.
Dunque un segnale analogico la cui componente armonica più elevata abbia frequenza f_M, potrà essere determinato univocamente a partire dai valori campionati se la frequenza di campionamento f_c è:

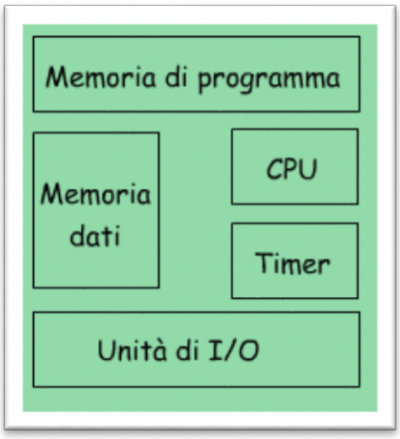
$$f_c \geq 2 \cdot f_M$$

Per ricostruire fedelmente il segnale occorrerà trattare il segnale campionato con un filtro passa-basso la cui risposta sia piatta fino alla frequenza f_M e si annulli per una frequenza minore o uguale a $f_c - f_M$.
Nonostante la frequenza di campionamento minima sia $f_c = 2 \cdot f_M$, in pratica si preferisce campionare a frequenza maggiore per rendere possibile il filtraggio in fase di ricostruzione.

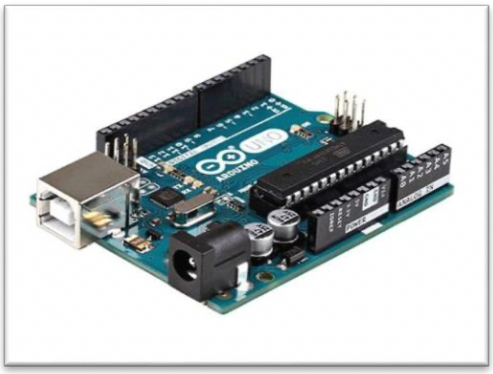


Trattando segnali non sinusoidali contenenti un numero molto elevato di armoniche, il teorema del campionamento può essere rispettato solo se il segnale viene preventivamente filtrato, in modo che la più elevata tra tutte le armoniche che lo compongono abbia frequenza $f_M \leq f_c/2$. Diversamente, la presenza nel segnale da campionare di armoniche superiori a $f_c/2$ determina l'insorgere di componenti armoniche estranee di ampiezza anche elevata (aliasing).

Arduino UNO



I microcontrollori (μC) sono circuiti integrati digitali che nella forma più ridotta contengono in un unico chip un microprocessore (μP), una memoria di programma di tipo non volatile (ROM), una memoria dati volatile (RAM), un'unità di ingresso-uscita (I/O) ed almeno un timer. La memoria di programma può essere di tipo PROM e in tal caso il μC può essere programmato solo una volta oppure di tipo EEPROM nel caso possa essere programmato più volte. I μC più complessi sono dotati al loro interno di svariate periferiche come comparatori, convertitori analogico-digitali, generatori di segnali PWM (Pulse Width Modulation), porte di comunicazione seriale, eccetera.



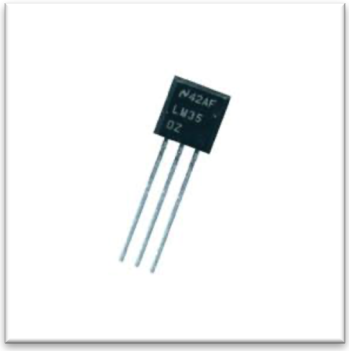
Un esempio di microcontrollore è quello che si trova nella scheda Arduino UNO. Questa scheda è paragonabile a un piccolo computer in grado di interagire con l'ambiente circostante. A differenza dei computer tradizionali che sono in grado di restare in attesa di comandi impartiti dall'utente, una scheda Arduino che è stata programmata e alla quale sono stati collegati dei componenti, è in grado di interagire con il mondo esterno attraverso grandezze fisiche diverse: luce, temperatura, pressione.

La potenza di elaborazione di una scheda Arduino è assai inferiore rispetto a un normale PC, ad esempio il processore presente nella scheda Arduino UNO (ATMega328) possiede 32KB di memoria FLASH utili alla memorizzazione del programma, chiamato sketch, 2KB di RAM e 1KB di memoria EEPROM, quest'ultima accessibile direttamente dagli sketch per salvare informazioni.

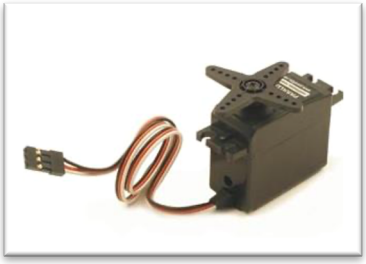
Uno dei campi di applicazione della scheda Arduino è il *Physical Computing* che ha come scopo quello di realizzare sistemi in grado di interagire con il mondo esterno attraverso l'uso di hardware e software. Il *Physical Computing* prevede l'utilizzo di sensori e attuatori in relazione al funzionamento di un sistema per eseguire le seguenti fasi:

- Acquisizione dati dal mondo reale mediante sensori;
- Esecuzione di valutazioni su di essi;
- Azione conseguente, mediante gli attuatori.

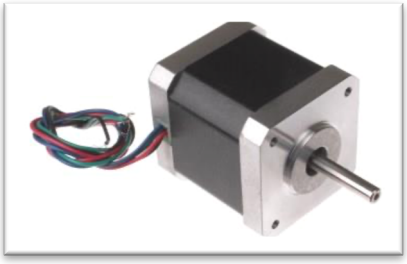
I sensori sono dispositivi che consentono di rilevare e acquisire grandezze fisiche diverse, come ad esempio di temperatura, luminosità, prossimità, velocità, posizione, forza, infrarossi. Un sensore molto noto presente nei dispositivi mobili come tablet e smartphone, è l'accelerometro attraverso il quale viene misurata l'accelerazione del dispositivo che lo ospita e viene usato ad esempio per valutare l'inclinazione.

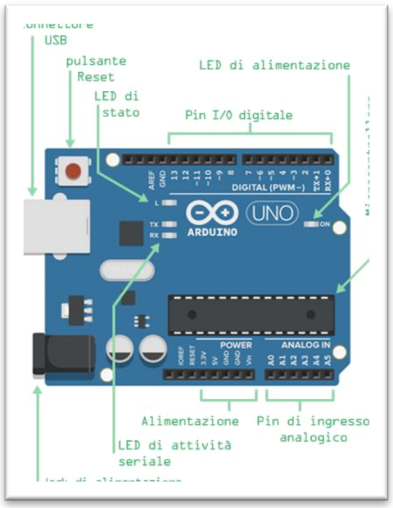


Il mondo dei sensori è estremamente vasto e in continua evoluzione, se pensiamo a una qualsiasi grandezza fisica, esiste molto probabilmente il relativo sensore in grado di misurarla e acquisirla. Possiamo suddividere i sensori in due categorie: analogici e digitali. Il sensore analogico restituisce un valore continuo nel tempo, al contrario di quello digitale che fornisce un valore numerico discreto. Mentre un sensore digitale riceve in ingresso una grandezza fisica, che è sempre per definizione analogica, e restituisce un risultato digitale, un sensore analogico restituisce ancora una grandezza analogica. Tuttavia possiamo connettere direttamente alla scheda Arduino anche sensori analogici in quanto provvederà la circuiteria interna alla scheda a effettuare la conversione analogico/digitale mediante campionamento.



Un esempio di attuatore può essere rappresentato da un altoparlante acustico che converte un segnale elettrico in onde sonore, per questo motivo viene anche definito come un trasduttore elettroacustico. Gli attuatori consentono di agire sull'ambiente modificandone eventualmente lo stato, per esempio, accendere un LED oppure far muovere un motore. Ad esempio in un sistema di sicurezza un LED acceso può segnalare all'utente una intrusione, oppure mediante un motore può essere chiusa una porta di sicurezza, oppure ancora in un sistema di riscaldamento il LED può segnalare la temperatura elevata e mediante un motore può essere azionata una ventola di raffreddamento. Alcuni tipici attuatori che consentono di effettuare delle azioni sono i motori passo-passo (usati nei robot, nelle stampanti 3D, ...) e i servomotori (usati nei macchinari industriali, ...).





La scheda Arduino UNO oltre ad essere costituita dal μC ATMega328 (all'interno di questo chip risiede un firmware chiamato Bootloader, in grado di caricare le istruzioni dello sketch nella memoria della scheda e può essere paragonato al BIOS del PC), dalle memorie FLASH (32KB), RAM (2KB) ed EEPROM (1KB), lavora ad una frequenza di 16MHz e possiede 14 pin configurabili come ingressi/uscite digitali, 6 pin di ingressi analogici, 6 pin di uscite analogiche simulate con la tecnica PWM (Pulse Width Modulation) e una porta USB. Inoltre, la scheda Arduino può essere alimentata ad una tensione di 3.3V o 5V.

Le funzioni che si utilizzano per la gestione degli ingressi e delle uscite digitali sono:

- **pinMode(pin, mode);** utilizzato nella funzione setup(), serve per configurare un determinato pin e stabilire se deve essere un ingresso o un'uscita.
- **digitalRead(pin);** l'istruzione permette di leggere lo stato di un pin di input e restituisce un valore `HIGH` se al pin è applicata una tensione o un valore `LOW` se non è applicato nessun segnale. Il pin può essere specificato come una variabile o una costante.
- **digitalWrite(pin, valore);** attiva o disattiva un pin digitale, quindi l'istruzione pone il pin di uscita a livello logico `HIGH` o `LOW`. Il pin può essere specificato come una variabile o una costante.

Le funzioni che invece si utilizzano per la gestione degli ingressi e delle uscite analogiche sono:

- **analogRead(pin);** legge il valore di tensione applicato a un pin di input analogico con una risoluzione pari a 10 bit. Questa funzione restituisce un numero intero compreso tra 0 e 1023.
- **analogWrite(pin, valore);** cambia la percentuale della modulazione di larghezza di impulso PWM su uno dei pin contrassegnati dal trattino (~). Il microcontrollore presente sulla scheda Arduino non è in grado di attribuire valori intermedi (tra HIGH e LOW), tuttavia grazie all'elevata velocità di esecuzione del codice possiamo simulare una grandezza analogica come una digitale. Questo avviene facendo passare dallo stato HIGH allo stato LOW e viceversa, il segnale applicato a un piedino di Arduino. Il rapporto tra il tempo passato in stato HIGH e tempo passato in stato LOW indica il valore medio del segnale trasmesso. Questa tecnica prende il nome di modulazione di larghezza di impulso. La funzione dovrà assumere come primo parametro uno dei pin PWM $(\sim 3, \sim 5, \sim 6, \sim 9, \sim 10, \sim 11)$, e come secondo parametro un numero compreso tra 0 e 255. 0 indicherà un pin sempre in stato LOW mentre 255 un

pin sempre in stato HIGH. Inoltre la funzione analogWrite() genera un duty cycle, che se si conosce, si può facilmente risalire al valore da inserire come secondo parametro.

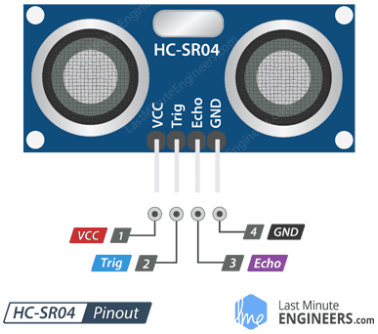
Esempio: duty cycle al 30% $\Rightarrow val = \frac{255 \cdot 30}{100} = 76{,}5$

Esistono altre funzioni molto utili:

- **delay(ms);** mette in pausa un programma per la quantità di tempo specificato in millisecondi. Il valore 1000 è pari a 1 secondo.
- **delayMicroseconds(us);** mette in pausa un programma per la quantità di tempo specificato in microsecondi.
- **millis();** restituisce il numero di millisecondi da quando la scheda Arduino ha iniziato l'esecuzione del programma corrente. Il tipo di dato è un unsigned long.
- **Serial.begin(9600);** configura la porta seriale di comunicazione RS232 e imposta la velocità di comunicazione; di default viene settata a 9600 daud rate.
- **Serial.println();** invia su seriale il valore dell'argomento con il carattere di invio a capo alla fine (\n). L'argomento può essere una variabile oppure una costante (numerica o stringa, va racchiusa tra doppi apici).

Sensore di Prossimità HC-SR04

Il Sensore HC-SR04 è uno dei più comuni sensori ad ultrasuoni utilizzati per la misurazione delle distanze. Esso riesce a rilevare le distanze che vanno dai 2 cm fino ai $4m$, anche se a grandi distanze le interferenze dell'aria possono facilmente impedire la corretta misurazione.

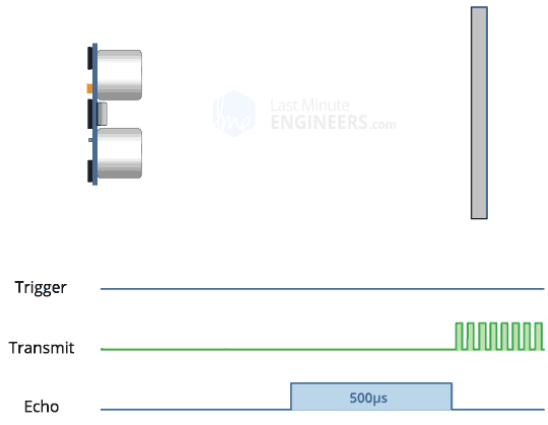


Il principio di funzionamento è molto semplice: uno dei due cilindri emette delle onde sonore ad ultrasuoni che rimbalzando dall'oggetto che si trova davanti al sensore, ritornano indietro e vengono rilevate dall'altro cilindro. Dato che la velocità del suono è una grandezza fisica nota è possibile, conoscendo il tempo impiegato dall'onda sonora per arrivare dal primo cilindro al secondo, risalire alla distanza.

È importante tenere presente che nonostante il sensore sia molto preciso si basa comunque sulle onde sonore e quindi in alcuni casi può essere ingannato dato che alcuni oggetti, per esempio i peluche, assorbono le onde sonore impedendo di conseguenza la corretta misurazione.

Il sensore ad ultrasuoni HC-SR04 presenta 4 pin:

1) V_{cc} viene collegato alla tensione di alimentazione pari a 5 V;
2) $Trig$ è il pin "Trigger" che deve essere attivato per inviare il segnale ad ultrasuoni;
3) $Echo$ è il pin che produce un impulso che si interrompe quando viene ricevuto il segnale riflesso dall'oggetto;
4) GND viene collegato alla messa a terra.



Il sensore ad ultrasuoni HC-SR04 lavora nel campo delle frequenze ad ultrasuoni ed emette onde ad una frequenza di 40 kHz che sono soggette a fenomeni di riflesse (in fisica la riflessione è il fenomeno per cui un'onda cambia di direzione a causa di un impatto con un materiale riflettente). Questa caratteristica permette al sensore HC- SR04 di rilevare la distanza dall'oggetto colpito dal segnale sonoro. Il sensore lavora nel seguente modo: un impulso a 5 V di almeno 10 μs di durata viene applicato al $pin\ Trigger$, generando così un treno di 8 impulsi ultrasonici con frequenza di 40 kHz, che si allontanano dal sensore viaggiando nell'aria circostante. Il segnale sul $pin\ Echo$ diventa alto e così inizia il conteggio del tempo di ritorno in attesa dell'onda riflessa.

Si possono verificare due casi:

1) Se l'impulso non viene riflesso il segnale sul $pin\ Echo$ torna basso dopo 38 ms e questo vuol dire che il sensore non ha rilevato alcun ostacolo.
2) Se invece il treno di onde ultrasoniche viene riflesso da un oggetto, il segnale sul $pin\ Echo$ diventa basso quando il cilindro del sensore viene colpito dal segnale riflesso.

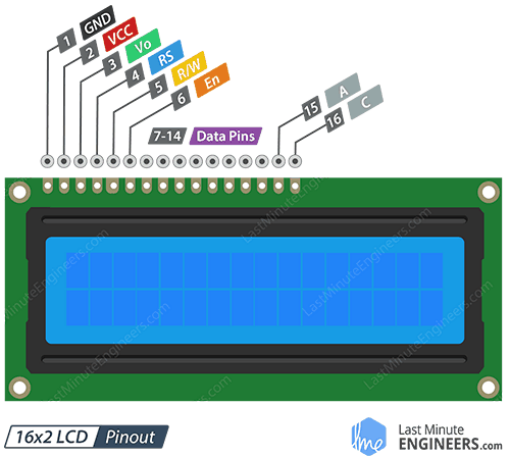
Il tempo ottenuto servirà per calcolare la distanza dall'oggetto. Prima di effettuare il calcolo della distanza bisogna fare alcune considerazioni: la velocità del suono nell'aria alla temperatura di 20 °C è pari a $343{,}8\frac{m}{s}$, mentre, l'onda sonora percorre due volte la distanza, in particolare, quando viene emessa verso l'oggetto e dopo la riflessione verso il sensore, quindi bisognerà dividere per due la distanza. Inoltre dato che il tempo è espresso in μs, la velocità dovrà essere convertita in $\frac{cm}{\mu s}$, come segue:

$$v = 343{,}8\frac{m}{s} = \frac{34380}{10^6 \cdot 10^{-6}} \cdot \frac{cm}{s} = \frac{34380}{10^6} \cdot \frac{cm}{\mu s} = 0{,}03438\ \frac{cm}{\mu s}$$

Quindi si può calcolare la distanza, espressa in cm, tenendo conto delle considerazioni precedenti, con la seguente formula:

$$d = \frac{v \cdot t}{2}$$

Display LCD



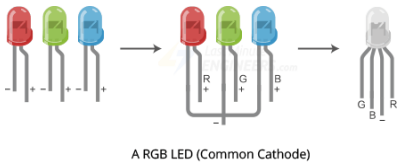
Gli LCD (*Liquid Crystal Display*) sono display con tecnologia basata sulle proprietà ottiche dei cristalli liquidi. Il liquido intrappolato fra due superfici vetrose possiede molecole di forma allungata, paragonabili a piccoli cilindri. Tale liquido ha la caratteristica di disporre le molecole in posizione orizzontale o verticale a seconda di come viene polarizzato il liquido: in questo modo trasmette o meno la luce sottostante.

I display LCD consentono di visualizzare caratteri alfanumerici e ne esistono di diverse dimensioni che vanno da pochi centimetri a diversi metri. Per il funzionamento richiedono una tensione elettrica di 5 V in corrente continua. Nel kit Arduino è presente il display LCD HD44780 alfanumerico da 2 righe e 16 colonne con 16 piedini.
Di seguito è riportata la piedinatura del display HD44780:

Pin	Significato	Spiegazione
1	VSS	Collegamento a massa
2	VDD	Alimentazione 5V
3	V0	Controlla il contrasto dei caratteri in base alla tensione in ingresso (da 0V a 5V): al variare della tensione varia il contrasto. In genere viene connesso un potenziometro per poterne tarare il contrasto manualmente
4	RS	Segnale che controlla dove posizionare i caratteri sullo schermo
5	Read/Write	Seleziona la modalità di funzionamento: lettura/scrittura, a massa per scrittura e 5V per lettura
6	Enable	Indica la ricezione di un comando
Da 7 a 10	D0-D3	Linee dei dati che inviano i caratteri sul display (non utilizzati)
Da 11 a 14	D4-D7	Linee dati collegate ai pin della scheda Arduino
15	A	Tensione di 4,2V per alimentare la retroilluminazione del display
16	K	Massa per consentire la retroilluminazione

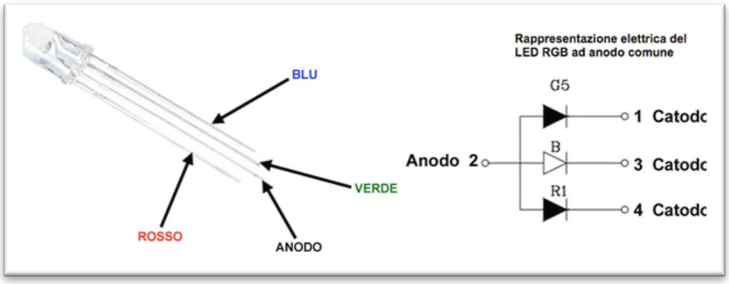
La libreria `LiquidCrystal.h` contiene le istruzioni principali che consentono di gestire un display LCD.

Led RGB



I Led Multicolore, anche chiamati Led RGB, sono componenti in grado di produrre una luce formata da tre differenti lunghezze d'onda, quella del Rosso, del Verde e del Blu.

A differenza dei Led tradizionali possiedono un anodo e tre catodi (RGB ad anodo comune), oppure tre anodi e un catodo (RGB a catodo comune). Essendo in grado di generare forme d'onda basate sulle tre tinte fondamentali (rosso, verde e blu), questi tipi di Led consentono di ottenere praticamente tutti i colori visibili semplicemente combinando opportunamente queste tre tinte fondamentali. Per ottenere la miscela di colore desiderata basta applicare una tensione diversa ai suoi pin in ingresso.

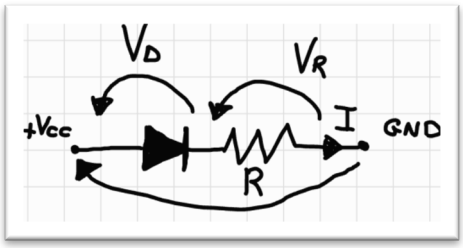


Per quanto riguarda il progetto ho utilizzato un Led RGB ad anodo comune. Possiamo notare che il piedino 2 rappresenta l'anodo ed è facilmente individuabile in quanto è il *pin* più lungo.

Ora per dimensionare le tre resistenze che devono essere collegate al Led RGB procediamo come segue: prima di tutto colleghiamo il Led RGB ad anodo comune con il piedino 2 a massa, quindi, sapendo che le uscite digitali della scheda Arduino UNO forniscono 5 V e che la tensione necessaria per accendere ogni singolo colore, cioè ogni singolo diodo Led, è la seguente, dal datasheet:

Corrente	Colore	Tensione
20 *mA*	Rosso	$2{,}0\ V - 2{,}5\ V$
20 *mA*	Verde	$3{,}5\ V - 4{,}0\ V$
20 *mA*	Blu	$3{,}5\ V - 4{,}0\ V$

Prendendo in considerazione un diodo Led con in serie una resistenza R, come in figura:



Applicando il II principio di Kirchhoff possiamo ricavare il valore della tensione V_R ai capi della resistenza R:

$$V_{cc} = V_D + V_R \quad \Rightarrow \quad V_R = V_{cc} - V_D$$

Sapendo che $V_R = R \cdot I$ si può ricavare il valore della resistenza R:

$$R = \frac{V_{cc} - V_D}{I}$$

Per determinare la resistenza R_{rosso}, ovvero la resistenza in serie al diodo Led adibito per il colore rosso, dal datasheet considerando $V_{cc} = 5\,V, V_D = 2\,V$ e $I = 20\,mA$, si ottiene:

$$R_{rosso} = \frac{V_{cc} - V_D}{I} = \frac{5 - 2}{20 \cdot 10^{-3}} = 150\,\Omega$$

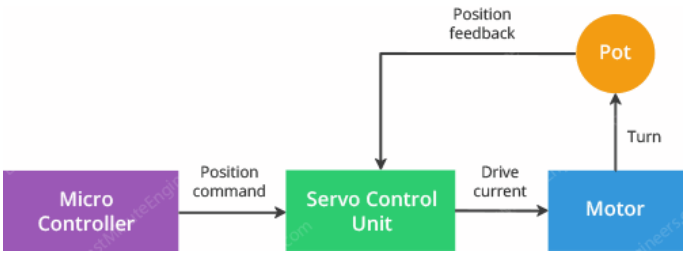
In modo analogo si ottengono le resistenze R_{verde} ed R_{blu} che risultano uguali perché la tensione ai capi del diodo Led in questi due casi è la stessa rispetto a quella sulla resistenza R_{rosso}, quindi, considerando $V_{cc} = 5\,V, V_D = 3{,}5\,V$ e $I = 20\,mA$ si ottiene:

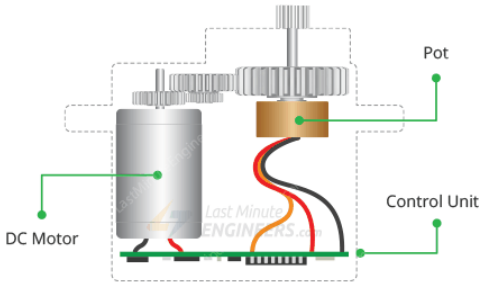
$$R_{verde} = R_{blu} = \frac{V_{cc} - V_D}{I} = \frac{5 - 3{,}5}{20 \cdot 10^{-3}} = 75\,\Omega$$

Quindi teoricamente servono due resistenze da $75\,\Omega$ e una da $150\,\Omega$. Tuttavia utilizzeremo le resistenze, presenti nella dotazione disponibile nello Starter Kit di Arduino Mega, che si avvicinano di più ai valori calcolati, ovvero $220\,\Omega$ (valore commerciale). In questo modo il colore rosso sarà più acceso e brillante mentre il colore verde e blu saranno più spenti e meno brillanti.

Servomotore

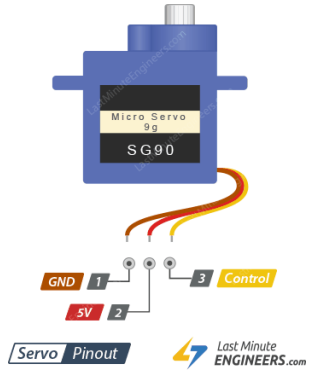
Un servomotore è un motore elettrico da cui fuoriesce un perno in grado di ruotare di un angolo compreso tra 0° e 180°, mantenendosi stabile nella posizione raggiunta. Per ottenere la rotazione del perno viene utilizzato un motore a corrente continua e un meccanismo di demoltiplica che consente di aumentare la coppia in fase di rotazione. La rotazione del motore è effettuata tramite un circuito di controllo interno in grado di rilevare l'angolo di rotazione raggiunto dal perno tramite un potenziometro resistivo e bloccare il motore sul punto desiderato.





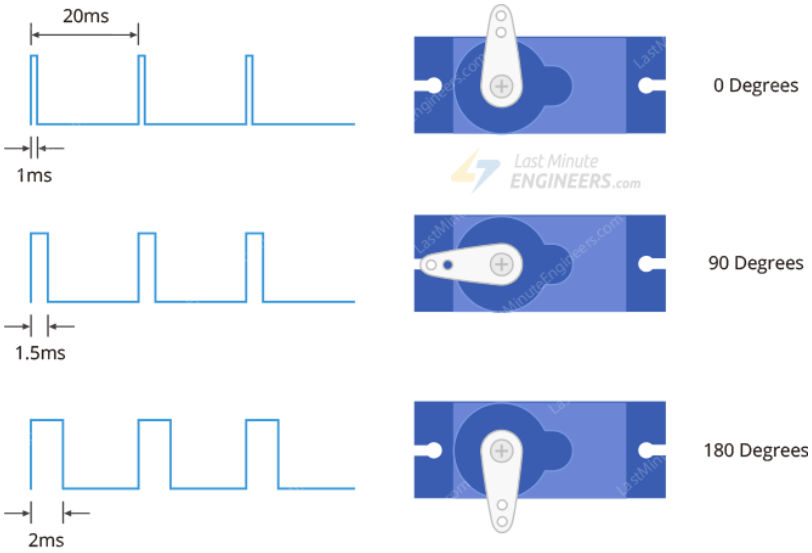
Il motore e il potenziometro sono collegati al circuito di controllo e l'insieme di questi tre elementi definisce un sistema di feedback ad anello chiuso. Il circuito di controllo e il motore vengono alimentati da una tensione continua stabilizzata, in genere di valore compreso tra $4{,}8\ V$ e $6\ V$.

Per far ruotare il motore bisogna inviare un segnale digitale al circuito di controllo. In questo modo esso si attiverà e, attraverso una serie di ingranaggi, varierà la posizione dell'albero del potenziometro. Quando il potenziometro raggiunge la posizione desiderata, il circuito di controllo spegne il motore.
I servomotori vengono progettati in genere per effettuare una rotazione parziale piuttosto che un moto rotatorio continuo. Superare i limiti meccanici di un servomotore provocherebbe lo sfregamento o la vibrazione degli ingranaggi. Se questi effetti proseguono per più di qualche secondo, gli ingranaggi del motore e lo stesso potrebbero danneggiarsi in modo irreparabile.



Il servomotore presenta tre fili, due dei quali sono riservati all'alimentazione in corrente continua. Il positivo è di colore rosso, il negativo di colore nero, il terzo filo, normalmente di colore bianco, è riservato per il controllo del posizionamento. Il colore di questi fili può però variare a seconda della casa costruttrice. Tramite il filo del controllo è necessario applicare un segnale impulsivo o PWM le cui caratteristiche sono quasi univoche per qualsiasi servomotore disponibile in commercio. In un sistema di questo tipo, il servo risponde alla durata di un

segnale definito all'interno di un treno di impulsi a frequenza fissa. In particolare, il circuito di controllo risponde a un segnale digitale i cui impulsi hanno una durata variabile da circa 1 ms a circa 2 ms. Questi impulsi vengono trasmessi alla velocità di 50 kHz. La durata esatta di un impulso, espressa in frazioni di millisecondo, stabilisce la posizione del servo. Alcuni servo consentono di variare la frequenza del segnale PWM, altri invece non funzionano correttamente oppure "tremano" nel caso in cui gli impulsi vengono inviati a frequenze diverse.
Per garantire il corretto funzionamento di un servo, bisogna verificare sempre che ci siano circa 20 ms di pausa tra l'inizio di un impulso e quello successivo. È quindi deducibile che alla durata di 1 ms il servo viene comandato per ruotare completamente in una direzione, per esempio in senso orario, mentre a 2 ms il servo ruota completamente nella direzione opposta. Di conseguenza, un impulso di 1,5 ms comanda il servo in modo da posizionarlo nella sua posizione centrale, o di riposo.



La durata degli impulsi a volte può variare con marche diverse e possono essere, per esempio, 0,5 ms per 0° e 2,5 ms per 180°.
L'alimentazione fornita al motore all'interno del servo è anche proporzionale alla differenza tra la posizione attuale dell'albero e la posizione che deve raggiungere. Se il servo deve effettuare un movimento breve per raggiungere la nuova posizione, il motore viene guidato con una velocità di rotazione bassa. In questo modo si garantisce che il motore non superi la posizione desiderata. Al contrario, se il servo deve effettuare un movimento più accentuato per raggiungere la nuova posizione, il motore viene pilotato alla massima velocità consentita, in modo da arrivare appena possibile e ovviamente rallenta quando il servo si avvicina alla posizione finale. Questo processo in apparenza abbastanza complesso avviene in un breve intervallo di tempo.

Parcheggio Automatizzato con Arduino Mega

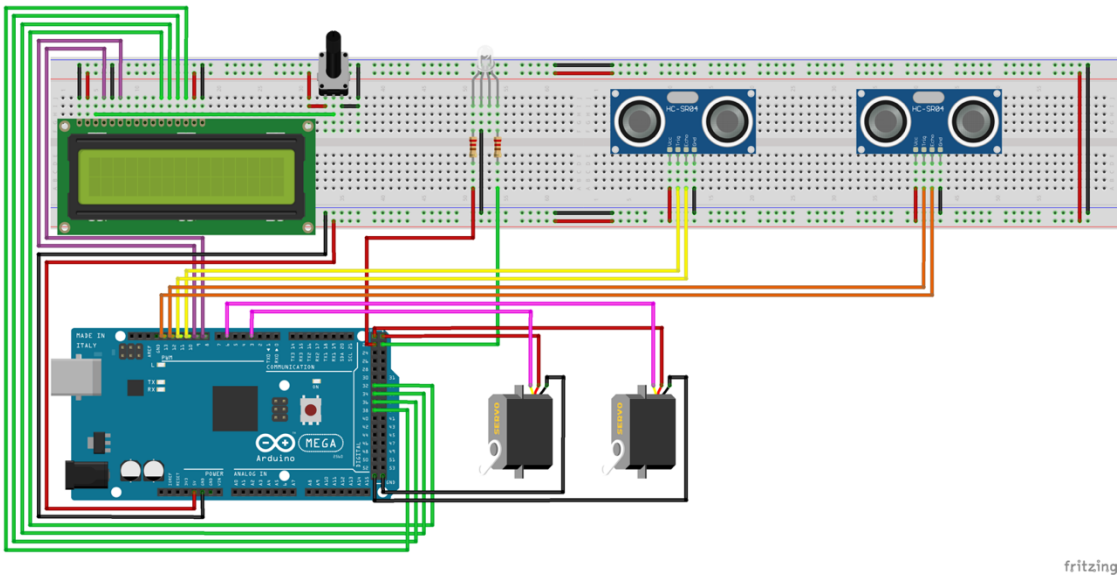
Descrizione Generale

Per la realizzazione del parcheggio automatizzato è stata utilizzata la scheda Arduino Mega, poiché la scheda Arduino UNO non ha un numero sufficiente di pin per ospitare tutti i componenti descritti in precedenza, al fine di simulare il funzionamento di un parcheggio. Per tale scopo sono stati utilizzati due sensori ad ultrasuoni HC-SR04 per rilevare la distanza, un display LCD (16x2) per visualizzarla, un led RGB che a seconda del colore, verde o rosso, indica se il parcheggio presenta posti liberi o meno (per quanto riguarda il colore blu, che non viene utilizzato in questo progetto, basterà semplicemente non collegare il pin del Led RGB, adibito per tale colore, ad uno dei pin digitali di Arduino Mega) e due servomotori per simulare le sbarre automatiche.

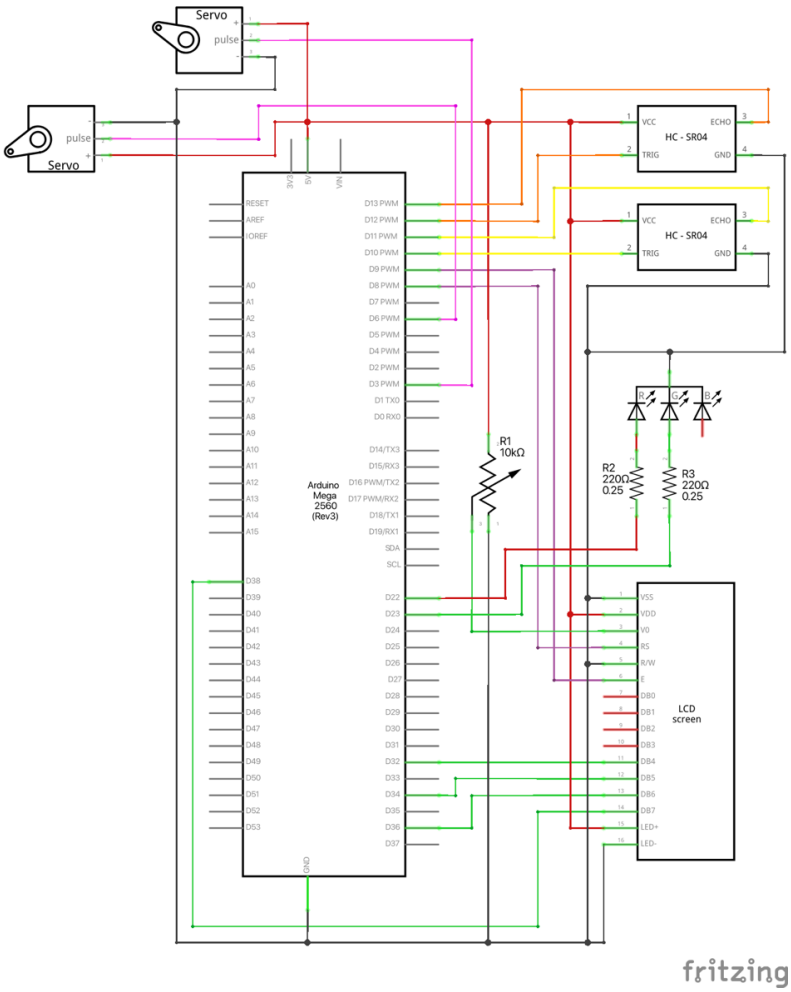
Materiali Utilizzati

- Scheda Arduino Mega;
- Display LCD (16x2);
- Potenziometro con resistenza interna pari a 10 $k\Omega$;
- 2 sensori di prossimità HC-SR04;
- Led RGB;
- 2 servomotori;
- 2 resistenze da 220 Ω;
- Fritzing;
- Vari cavi per i collegamenti.

Schema di Montaggio



Schema Elettrico



Sketch

```
#include <LiquidCrystal.h>
#include <Servo.h>

// Corrispondenza pin LCD con i pin digitali di Arduino Mega
#define REGISTER_SELECT 8
#define ENABLE 9
#define DATA_PIN_7 38
#define DATA_PIN_6 36
#define DATA_PIN_5 34
#define DATA_PIN_4 32

// Variabili per il LED Multicolore RGB
#define RED 22
#define GREEN 23

// Variabili per i sensori ad ultrasuoni HC-SR04
#define TRIGGER_PIN_INPUT 10
#define ECHO_PIN_INPUT 11
#define TRIGGER_PIN_OUTPUT 12
#define ECHO_PIN_OUTPUT 13
```

```
// Variabili per i servo motori
#define SERVO_IN_PIN 3
#define SERVO_OUT_PIN 6
int pos_in_min;
int pos_in_max;
int pos_out_min;
int pos_out_max;
int current_in_pos;
int current_out_pos;

Servo servo_in;
Servo servo_out;

int seats = 5; // Posti totali del parcheggio
int range_input = 4;
int range_output = 5;
bool old_input = false;
bool old_output = false;

LiquidCrystal lcd(REGISTER_SELECT, ENABLE, DATA_PIN_4, DATA_PIN_5,
DATA_PIN_6, DATA_PIN_7);

void setup() {
  pinMode(RED, OUTPUT);
  pinMode(GREEN, OUTPUT);

  pinMode(TRIGGER_PIN_INPUT, OUTPUT);
  pinMode(ECHO_PIN_INPUT, INPUT);
  pinMode(TRIGGER_PIN_OUTPUT, OUTPUT);
  pinMode(ECHO_PIN_OUTPUT, INPUT);

  digitalWrite(TRIGGER_PIN_INPUT, LOW);
  digitalWrite(TRIGGER_PIN_OUTPUT, LOW);

  lcd.begin(16, 2); // Impostazioni display LCD (colonne, righe)
  servo_in.attach(SERVO_IN_PIN);
  servo_out.attach(SERVO_OUT_PIN);

  pos_in_min = servo_in.read() + 90;
  pos_in_max = servo_in.read();
  current_in_pos = pos_in_max;

  pos_out_min = servo_out.read() + 90;
  pos_out_max = servo_out.read();
  current_out_pos = pos_out_max;

  Serial.begin(9600);
}

void loop() {
  digitalWrite(TRIGGER_PIN_INPUT, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIGGER_PIN_INPUT, LOW);

  unsigned long input_time = pulseIn(ECHO_PIN_INPUT, HIGH);
  float input_distance = 0.03438 * input_time / 2;

  digitalWrite(TRIGGER_PIN_OUTPUT, HIGH);
  delayMicroseconds(10);
```

```
  digitalWrite(TRIGGER_PIN_OUTPUT, LOW);

  unsigned long output_time = pulseIn(ECHO_PIN_OUTPUT, HIGH);
  float output_distance = 0.03438 * output_time / 2;

  bool input = input_distance < range_input;
  bool output = output_distance < range_output;

  if (input == true && old_input == false) {
    if (seats > 0) {
      seats--;
      while (current_in_pos < pos_in_min) {
        current_in_pos++;
        servo_in.write(current_in_pos);
        delay(10);
      }
    }
  } else if (input == false && old_input == true) {
    while (current_in_pos > pos_in_max) {
      current_in_pos--;
      servo_in.write(current_in_pos);
      delay(10);
    }
  } else if (output == true && old_output == false) {
    if (seats < 5) {
      seats++;
      while (current_out_pos < pos_out_min) {
        current_out_pos++;
        servo_out.write(current_out_pos);
        delay(10);
      }
    }
  } else if (output == false && old_output == true) {
    while (current_out_pos > pos_out_max) {
      current_out_pos--;
      servo_out.write(current_out_pos);
      delay(10);
    }
  }
  old_input = input;
  old_output = output;

  if (seats > 0) {
    digitalWrite(GREEN, HIGH);
    digitalWrite(RED, LOW);
  } else {
    digitalWrite(GREEN, LOW);
    digitalWrite(RED, HIGH);
  }

  Serial.println(seats);
  lcd.clear(); // Pulisce lo schermo
  lcd.setCursor(0, 0); // Va in posizione: colonna 1, riga 1
  lcd.print("Posti");
  lcd.setCursor(0, 1); // Va in posizione: colonna 1, riga 2
  lcd.print("Liberi: " + String(seats));

  delay(500);
}
```

Spiegazione Sketch

Analizzando lo Sketch si può notare come esso sia diviso in tre parti fondamentali:

1) **Definizione di costanti (macro) e variabili globali:** vengono definite delle macro globali (`REGISTER_SELECT`, `ENABLE`, …) per far corrispondere i pin del display LCD con i pin digitali della scheda Arduino Mega. Successivamente vengono definite due macro per comandare il led RGB (`RED`, `GREEN`), quattro macro per comandare il sensore di prossimità HC-SR04 (`TRIGGER_PIN_INPUT`, `ECHO_PIN_INPUT`, …) e infine due macro per comandare i due servomotori (`SERVO_IN_PIN`, `SERVO_OUT_PIN`).
   Successivamente vengono definite delle variabili utili per controllare l'angolo del perno presente nei due servomotori (`pos_in_min`, `pos_in_max`, …), una variabile per memorizzare i posti del parcheggio (`seats`), due variabili che definiscono il range al di sotto del quale i due sensori di prossimità HC-SR04 rilevano un'automobile (`range_input`, `range_output`) e due variabili per controllare lo stato dei sensori di prossimità HC-SR04 (`old_input`, `old_output`).
   Per utilizzare il display LCD bisogna importare una libreria dedicata chiamata "`LiquidCrystal.h`". Successivamente bisogna inizializzare il display LCD, creando un oggetto, in questo caso chiamato "`lcd`" e passargli come parametri tutti i pin che si vogliono utilizzare:

   ```
   LiquidCrystal lcd(REGISTER_SELECT, …);
   ```

   Per utilizzare i due servomotori bisogna importare una libreria dedicata chiamata "`Servo.h`". Successivamente vengono creati due oggetti (`servo_in`, `servo_out`), uno per ogni servomotore, in questo modo:

   ```
   Servo servo_in;
   Servo servo_out;
   ```

2) **La funzione `setup()`:** utilizzando l'istruzione `pinMode()` vengono impostati in modalità `OUTPUT` i pin del Led RGB e i pin Trigger di entrambi i sensori di prossimità HC-SR04, mentre i pin Echo vengono impostati in modalità `INPUT`. Inizialmente i pin Trigger dei due sensori di prossimità non devono generare alcuna onda sonora di conseguenza utilizzando l'istruzione `digitalWrite()` vengono impostati i pin allo stato `LOW`. Utilizzando l'istruzione `lcd.begin(16, 2)` viene impostata la dimensione del display LCD, infatti i valori 16 e 2 rappresentano rispettivamente il numero di colonne e il numero di righe. Successivamente vengono inizializzate le variabili per controllare l'angolo del perno presente nei due servomotori, in particolare utilizzando il metodo `read()` viene letto l'angolo attuale dei due servomotori

e, visto che quest'ultimi simulano delle sbarre automatiche, bisogna inizializzare l'angolo del perno con l'aggiunta di 90°, proprio perché le sbarre automatiche normalmente passano da un angolo di 0° ad un angolo di 90° e viceversa.
Infine utilizzando l'istruzione `Serial.begin(9600)` viene configurata la porta seriale di comunicazione RS232 e inoltre viene impostata la velocità di comunicazione, ovvero `9600 baud rate`.

3) **La funzione `loop()`:** utilizzando l'istruzione `digitalWrite()` viene impostato il pin Trigger in ingresso allo stato `HIGH`, così facendo viene generata un'onda sonora per la durata di 10 μs (`delayMicroseconds()`). Terminati i 10 μs utilizzando l'istruzione `digitalWrite()` viene impostato il pin Trigger in ingresso allo stato `LOW`. In seguito viene definita una variabile per memorizzare il tempo che ci mette l'onda sonora a propagarsi verso un oggetto, nel caso del progetto verso un'automobile, a riflettersi e a tornare indietro nel sensore di prossimità HC-SR04, dove viene rilevata dal pin Echo. Quanto appena descritto viene effettuato utilizzando l'istruzione `pulseIn()`. Poi viene definita una variabile per memorizzare la distanza che si calcola come descritto nella teoria del sensore di prossimità. La routine appena descritta per il sensore di prossimità HC-SR04 posto in entrata viene ripetuta tale a quale per il sensore di prossimità HC-SR04 posto in uscita, chiaramente utilizzando le opportune variabili. Successivamente vengono definite e inizializzate due variabili booleane, una per l'ingresso e una per l'uscita, che sono utili per controllare se la distanza rilevata dai sensori di prossimità si trova al di sotto di un determinato range (`range_input`, `range_output`).
Utilizzando un costrutto `if-else` si riescono a controllare gli stati dei due sensori di prossimità, in ingresso e in uscita, che consentiranno lo spostamento dell'angolo del perno dei servomotori a seconda delle casistiche riscontrate. In particolare quando la variabile `input` ha valore booleano `true`, cioè il sensore di prossimità in ingresso rileva un'automobile, e la variabile `old_input` ha valore booleano `false`, quindi c'è un cambiamento di stato, se il numero di posti nel parcheggio è maggiore di zero, questi vengono decrementati di uno, allora con un ciclo `while` viene incrementata la posizione attuale del perno del servomotore in ingresso facendola passare da 0° a 90°, cioè la sbarra automatica viene alzata. Viceversa se la variabile `input` ha valore booleano `false`, cioè il sensore di prossimità in ingresso non rileva un'automobile, e la variabile `old_input` ha valore booleano `true`, quindi c'è un cambiamento di stato, allora con un ciclo `while` viene decrementata la posizione attuale del perno del servomotore in ingresso facendola passare da 90° a 0°, cioè la sbarra automatica viene abbassata. In maniera del tutto analoga si costruiscono altri due costrutti `if-else` per il sensore di prossimità e per il servomotore presenti in uscita.
Dato che la funzione `loop()` viene ripetuta all'infinito bisogna aggiornare il vecchio stato (`old_input`, `old_output`) del sensore di prossimità con il

nuovo stato (`input`, `output`), ogni volta che la funzione `loop()` viene ripetuta.
Successivamente con un costrutto `if-else` si controllano i posti disponibili nel parcheggio: se ci sono posti disponibili allora il Led RGB si accenderà emanando una luce verde, viceversa se non ci sono posti disponibili allora il Led RGB si accenderà emanando una luce rossa. Per controllare lo stato dei pin del Led RGB viene utilizzata l'istruzione `digitalWrite()`.
Nella funzione `setup()` era stata precedentemente configurata la porta seriale, ora utilizzando l'istruzione `Serial.println(seats)`, vengono stampati il numero di posti presenti nel parcheggio, all'interno del monitor seriale e questo tornerà utile in seguito.
Infine utilizzando l'istruzione `lcd.clear()` viene pulito lo schermo del display e utilizzando l'istruzione `lcd.setCursor(0, 0)` viene impostato il cursore per poter scrivere nella colonna 1 e nella riga 1, cioè nell'angolo in alto a sinistra del display, quindi con l'istruzione `lcd.print("Posti")` viene stampata sul display la stringa tra le virgolette. Ora utilizzando nuovamente l'istruzione `lcd.setCursor(0, 1)` viene spostato il cursore nella colonna 1 e nella riga 2, cioè in basso a sinistra. Per stampare il numero di posti presenti nel parcheggio, bisogna convertire la variabile `seats`, che è stata definita come variabile di tipo intero, in una variabile di tipo stringa e per fare ciò si utilizza la funzione `String(seats)` che dovrà essere concatenata ("+") con la stringa "`Liberi: `" come segue:

```
lcd.print("Liberi: " + String(seats));
```

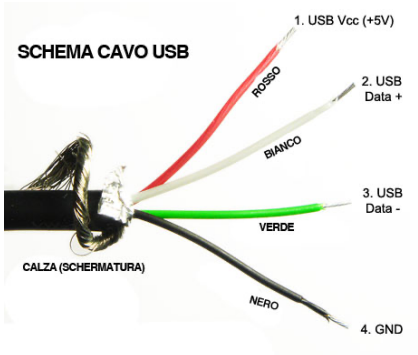
Parcheggio Automatizzato con Arduino Mega e Raspberry Pi

Descrizione Generale

Raspberry Pi è un `single-board computer` (ovvero un caso specifico di `System on Chip`, o `SoC`), con `CPU ARM`. Per la realizzazione di questo progetto viene utilizzato un Raspberry Pi 3 Model B che si presenta come una piccola scheda elettronica con integrati diversi moduli per poter utilizzare il dispositivo come semplice computer o come scheda di sviluppo per la prototipazione di moduli intelligenti per la domotica, per la robotica o più in generale per qualsiasi sistema automatico. Inoltre Raspberry supporta diversi sistemi operativi, principalmente basati su kernel Unix (in particolare Linux/Debian) e su architetture RISC. Le caratteristiche tecniche di Raspberry Pi 3 Model B sono le seguenti:

- Processore quad-core Broadcom BCM2837 a 64 bit, con frequenza di 1,2 GHz;
- 1 GB di memoria RAM;
- Un modulo di connettività Wireless e Bluetooth Low Energy BCM43438 (BLE);
- Un'interfaccia $GPIO$ a 40 pin;
- Quattro porte USB 2.0;
- Un'uscita stereo a 4 poli con porta video composita;
- Una porta HDMI con uscita video-audio;
- Una porta CSI per connettere la Raspberry Pi Camera;
- Una porta DSI per connettere il display touchscreen di Raspberry Pi.

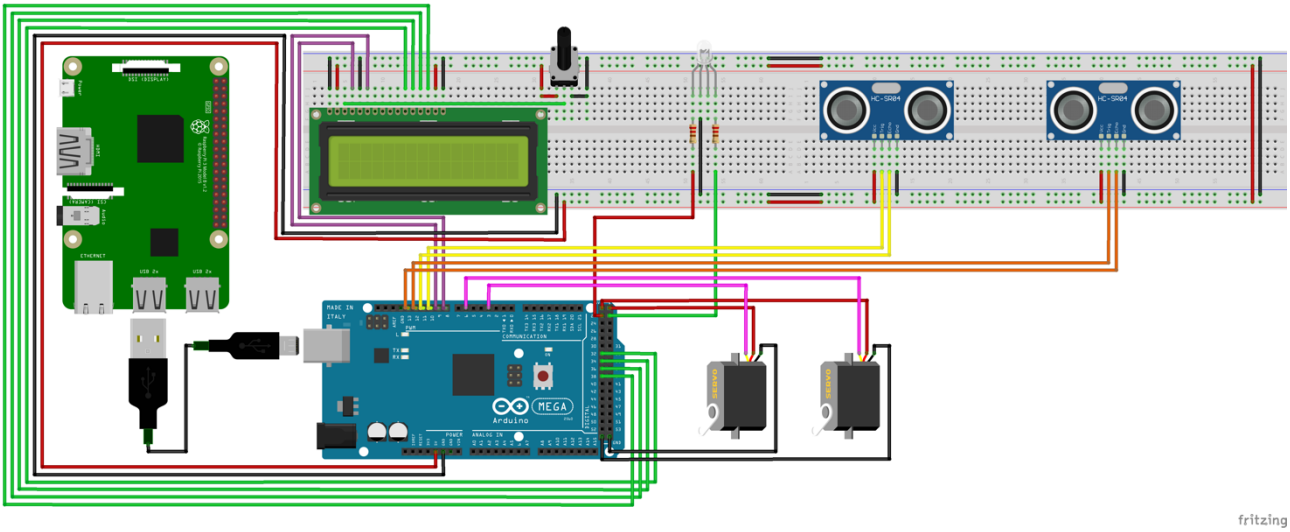
Come visto in precedenza all'interno dello sketch di Arduino Mega, utilizzando l'istruzione `Serial.begin(9600)` viene configurata la porta seriale di comunicazione RS232 e inoltre viene impostata la velocità di comunicazione, ovvero `9600 baud rate`. Poi, utilizzando l'istruzione `Serial.println(seats)`, vengono stampati il numero di posti presenti nel parcheggio all'interno del monitor seriale.



Per consentire lo scambio di dati tra Arduino Mega e Raspberry Pi è stata utilizzata la connessione via cavo USB tra le due schede che però presenta un inconveniente: Raspberry Pi 3 alimenta Arduino Mega con una tensione di 5 V che non è sufficiente per la gestione dei servomotori in contemporanea con l'utilizzo del sensori di prossimità, del display LCD e così via. Pertanto collegando Arduino Mega alla rete domestica attraverso l'attacco "Jack Japan" di Arduino Mega, che

consente un'alimentazione dai 7 V ai 12 V, e tagliando il cavo rosso di alimentazione presente all'interno del cavo USB è possibile trasmettere i dati, cioè il numero di posti disponibili nel parcheggio, da Arduino Mega a Raspberry Pi senza che quest'ultimo alimenti il microcontrollore.
L'aggiunta della scheda Raspberry Pi 3 Model B all'interno del progetto del parcheggio automatizzato ha come scopo quello di effettuare un monitoraggio dei posti disponibili in modalità remota, utilizzando uno smartphone, un computer o un tablet. Infatti grazie al linguaggio di programmazione Python e al servizio di messaggistica Telegram è stato realizzato un bot in grado di leggere, memorizzare e gestire i dati stampati sul monitor seriale di Arduino Mega.

Schema di Montaggio



Scripts di Python

bot.py

```
from parking import Parking
from telebot import types
import telebot
import json

parking = Parking()
db = json.load(open("db.json"))
bot = telebot.TeleBot(db["token"], parse_mode="HTML")
print("Bot in esecuzione.")

@bot.message_handler(commands=["start"])
def send_start(message):
    markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
    state_button = types.KeyboardButton(db["name_state_button"])
    help_button = types.KeyboardButton(db["name_help_button"])
```

```
    markup.row(state_button)
    markup.row(help_button)
    bot.send_message(message.chat.id, text=db["start_text"],
reply_markup=markup)

@bot.message_handler(commands=["state"])
@bot.message_handler(func=lambda message: message.text ==
db["name_state_button"])
def send_state(message):
    current_seats = parking.getSeats()
    bot.send_message(message.chat.id, text=f"Informazioni
Parcheggio\n\n{parking.getMessage(current_seats)}")

@bot.message_handler(commands=["help"])
@bot.message_handler(func=lambda message: message.text ==
db["name_help_button"])
def send_help(message):
    bot.send_message(message.chat.id, text=db["help_text"])

bot.polling()
```

parking.py

```
import serial, time

class Parking:
    def __init__(self):
        self.total_seats = 5

        try:
            self.arduino = serial.Serial(port="/dev/ttyACM0",
baudrate=9600, timeout=0)
        except serial.serialutil.SerialException:
            print("Non è stata rilevata alcuna scheda Arduino... Riprova
più tardi!")
            exit()

    def getSeats(self):
        try:
            return
int(self.arduino.read(self.arduino.inWaiting()).decode("utf-
8").split()[-1])
        except Exception as e:
            print(f"Si è verificato un errore... Riprova più
tardi!\n{e}")
            exit()

    def getMessage(self, current_seats):
        try:
            hour = time.strftime("%H:%M:%S")
            date = time.strftime("%d/%m/%Y")
            if current_seats > 0:
                return f"Posti Disponibili: {current_seats}\nPosti
Totali: {self.total_seats}\n\nUltimo Aggiornamento:\n{date} {hour}"
```

```
            else:
                return f"Il parcheggio è pieno!\n\nUltimo
Aggiornamento:\n{date} {hour}"
        except Exception as e:
            return f"Si è verificato un errore... Riprova più
tardi!\n{e}"
```

db.json

```
{
    "token": "1750146887:AAEtsFrt0GZrCwEfn3LGNTwOeyB67GDP68c",
    "start_text": "Benvenuto nel Progetto di Maturità 2020/2021 di
Antonio Bernardini!",
    "help_text": "<b>Come si usa @ParcheggioArduinoBot?</b>\n\n1)
Cliccando sul bottone <b>\"Info Parcheggio\"</b>, il bot invierà
delle informazioni relative al parcheggio;\n\nInfo
Sviluppatore:\n<b>Github:</b>
https://github.com/AntonioBerna\n<b>Telegram:</b> @CleverCode",
    "name_state_button": "Info Parcheggio",
    "name_help_button": "Aiuto"
}
```