ARDUINO UNO
Programmazione avanzata e Librerie di sistema

di

Simone Majocchi

VISPA Edizioni

Prefazione

Un bel po' di anni fa ero un ragazzino appassionato di elettronica che consumava riviste di elettronica dove scrivevano spesso altri ragazzini appassionati come me.
Uno di questi "ragazzini" che anticipavano i tempi era Simone, l'autore di questo libro.
Allora apprezzavo le sue qualità di divulgatore e sono molto contento che abbia deciso di applicarle ad Arduino al punto di arrivare al suo secondo libro sul tema.
In Italia si sente la mancanza di buoni divulgatori e di materiale introduttivo su Arduino e credo che questo libro possa essere di grande aiuto a chi vuole imparare a creare con questa piattaforma.
Arduino ha il potenziale di trasformare delle buone idee in progetti funzionati e potenzialmente in prodotti vendibili, in un momento difficile per l'economia questo può fare la differenza per molti lettori di questo libro.
Spero che vi godrete questo libro e Arduino, due prodotti di due "proto-makers" italiani.

Massimo Banzi

Sommario

CAPITOLO 1

Un po' di teoria

7

Il bello di Arduino sta nella sua estrema semplicità da un lato e scalabilità dall'altro. I primi passi si fanno con pochissimo sforzo, mentre con un po' di pazienza e una certa dose di costanza si può entrare in un mondo ricco di sfaccettature hardware e software, capaci di aprire la strada a realizzazioni importanti e complesse.

CAPITOLO 2

Preparazione all'ambiente di lavoro

19

Molta della versatilità di Arduino nasce da Internet e dalla facilità con cui è possibile condividere il materiale e, conseguentemente, le librerie dedicate a device hardware o funzioni software. In questo capitolo impariamo predisporre l'ambiente di lavoro e a comprendere i meccanismi di funzionamento delle librerie.

CAPITOLO 3

EEPROM - Lettura e scrittura nella memoria permanente

34

La memoria interna al microcontrollore è di tipo EEPROM e ha una libreria dedicata per la sua gestione. Una risorsa importante per leggere e scrivere fino a 1024 byte di dati, sempre disponibili e permanentemente memorizzati anche in assenza di alimentazione. L'ideale per fare un sequenze di luci a sei canali.

CAPITOLO 4

La libreria Ethernet

54

Arduino può sembrare solo un sistema per far accendere qualche luce e gestire qualche sensore, ma con l'aggiunta di uno shield Ethernet è in grado di entrare a pieno titolo nel mondo di Internet. La libreria dedicata permette infatti di far diventare Arduino un client, un server o un dispositivo che dialoga in UDP.

CAPITOLO 5

La libreria firmata

74

Un PC può essere potente nell'elaborazione e con la grafica, ma Arduino è l'ideale per interfacciarsi con il mondo reale. Per mettere questi due mondi insieme c'è Firmata: una soluzione che permette di dialogare con le risorse hardware di Arduino tramite un programma. Processing diventa con Firmata l'ambiente in cui scrivere programmi come le luci psichedeliche con strip RGB che ci siamo inventati per l'occasione.

CAPITOLO 6

La libreria LiquidCrystal

90

Gli schermi a cristalli liquidi sono ovunque, dai termometri digitali ai distributori automatici di cibi e bevande, ai sofisticati cruscotti delle automobili. Poteva Arduino restare fuori da questo mondo? No, e lo fa grazie a una libreria dedicata che elimina gran parte della complessità del pilotaggio dei display a caratteri che si possono acquistare in vari formati e colori a cifre sempre molto abbordabili.

CAPITOLO 7

La libreria SD

106

Le schede di memoria sono una soluzione molto economica e facilmente trasferibile da un sistema all'altro. Con l'apposita libreria anche Arduino può leggere e scrivere file su questi supporti e l'ausilio di appositi shield. L'ideale per realizzare dei compatti sistemi di data logging ambientali come quello che abbiamo preparato con una NTC e una foto resistenza.

CAPITOLO 8

La libreria Servo

127

Pilotare fino a 12 servo significa poter realizzare sistemi robotizzati anche di una certa complessità e la libreria Servo si prende cura dei vari aspetti legati alla generazione dei segnali necessari a gestire il posizionamento degli assi in modo stabile e preciso. Con due servo, un brandeggio che punta i numeri della tombola su un tabellone diventa un gioco da ragazzi.

CAPITOLO 9

La libreria SPI

145

Fra i vari tipi di interfaccia seriale, la SPI trova applicazione nel controllo di svariati dispositivi digitali. La sua discreta longevità è frutto delle sue caratteristiche che combinano la velocità alla relativa semplicità di implementazione a livello hardware. Con SPI abbiamo interfacciato un ChipCorder con cui potrete cimentarvi alla Morra Cinese (vocale).

CAPITOLO 10

La libreria Sottware Serial

166

Quando non basta una sola porta seriale o si devono utilizzare pin diversi da quelli previsti in hardware, si ricorre alla libreria SoftwareSerial, grazie alla quale quasi ogni piedino digitale può diventare parte di una porta seriale. Il nostro esempio applicativo usa un modulo seriale per la lettura delle tessere RFID con il quale potrete gestire accessi o attivare dispositivi, monitorando gli eventi sulla consolle seriale dell'IDE.

CAPITOLO 11

La libreria Stepper

191

Sono i motori che hanno permesso di realizzare dispositivi elettromeccanici molto interessanti o quasi indispensabili, come le stampanti. Arduino può gestirne fino a tre grazie alla libreria che si occupa di generare i segnali nella giusta sequenza per avere rotazioni orarie e antiorarie, lasciando a voi il piacere della creazione di applicazioni particolari, magari come il nostro mini scanner laser.

CAPITOLO 12

La libreria Wire

207

Ancora un metodo per il colloquio seriale con dispositivi esterni. Questa volta i fili sono solo 2 e grazie alla libreria Wire Arduino dialoga in I2C con dispositivi anche complessi come ad esempio l'accelerometro a tre assi, cuore del nostro ultimo progetto che non mancherà di attirare l'attenzione se qualcuno vorrà prenderlo in mano e lo ruoterà nello spazio.

1

Un po' di teoria

Il bello di Arduino sta nella sua estrema semplicità da un lato e scalabilità dall'altro. I primi passi si fanno con pochissimo sforzo, mentre con un po' di pazienza e una certa dose di costanza si può entrare in un mondo ricco di sfaccettature hardware e software, capaci di aprire la strada a realizzazioni importanti e complesse.

I comandi di base e le librerie

Ogni volta che scriviamo un programma – o sketch – sull'IDE di Arduino UNO, lo facciamo utilizzando i comandi e le funzioni che gli sviluppatori della piattaforma hanno pensato di rendere disponibili per accedere in modo semplice alle funzionalità hardware e per sviluppare algoritmi di una certa complessità.
Rispetto ad altri ambienti di sviluppo e a versioni più avanzate di C, questo ha una serie di vincoli e di restrizioni che derivano fondamentalmente dal fatto che il microcontrollore per il quale stiamo generando il codice oggetto ha una memoria limitata.
I microcontrollori **ATmel** su cui sono state sviluppate le varie soluzioni Arduino hanno un'architettura di tipo **Harvard**, mentre i personal computer hanno un'architettura di tipo **Von Neumann**. Questo determina che in Arduino la memoria di programma sia separata da quella per le variabili, offrendo una maggiore affidabilità di funzionamento da un lato, ma comporti dei limiti operativi che non possono essere facilmente superati.
Nello specifico, le variabili hanno uno spazio in RAM limitato e ogni progetto deve tenere conto di questo.
In un'architettura Von Neumann, programma e variabili condividono una memoria comune con la possibilità di dedicare più spazio a variabili o a programma a seconda delle necessità operative.
Se a questo aggiungiamo che stiamo parlando – nel caso di Arduino UNO con il suo **ATmega328** - di **32Kbyte** di spazio programma e solo **2 Kbyte** di RAM più **1Kbyte** di EEPROM, risulta chiaro che le dimensioni in gioco sono molto piccole (**Fig. 1**).

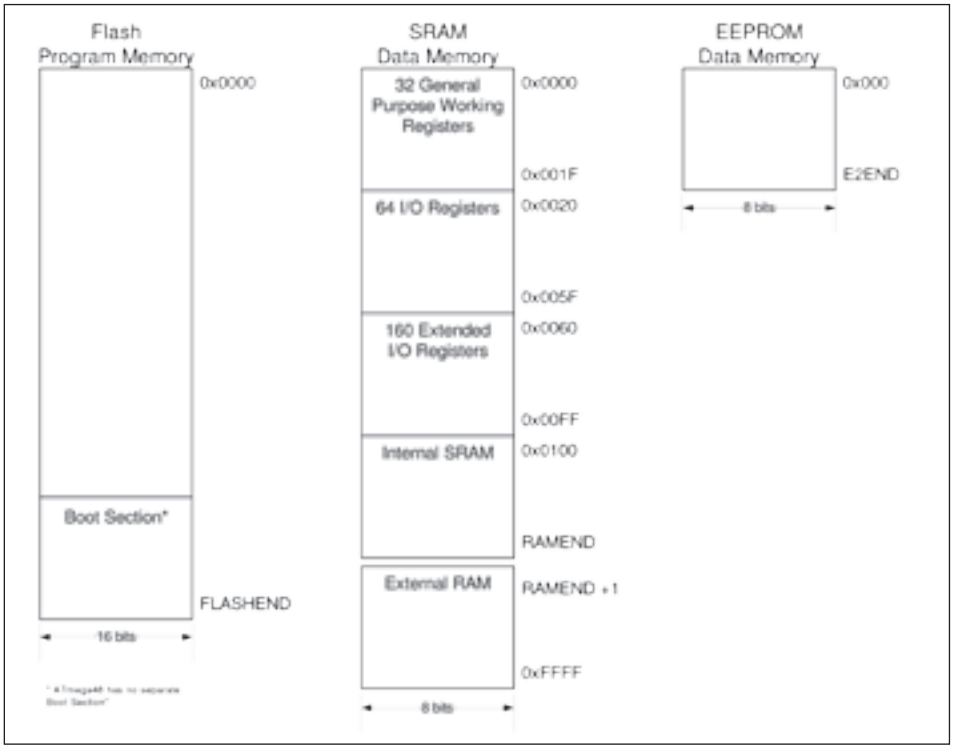


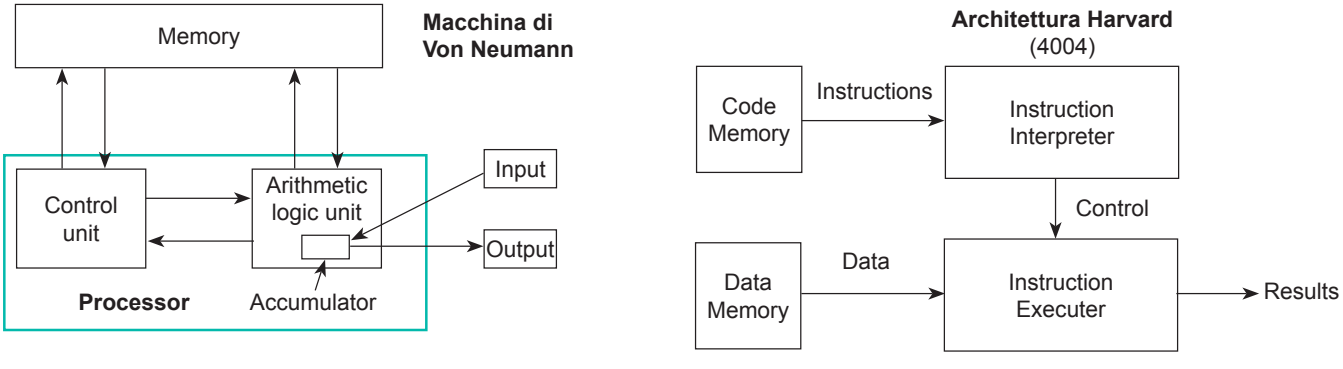
Fig. 1

Da qui il significativo lavoro di ottimizzazione di tutto quello che poteva essere ottimizzato senza che il sistema diventasse più complicato o meno funzionale. Un lavoro di bilanciamento che non a caso ha richiesto diversi anni e un significativo lavoro di valutazione dei feedback forniti dagli utenti e in generale dalla nutrita **community** che è cresciuta attorno a questa piattaforma (**Fig. 2**).

Funzioni e dimensione del codice

Ogni riga degli **sketch** che scriviamo viene gestita da tutto il sistema di analisi e compilazione per diventare del codice eseguibile direttamente

Fig. 2



dal microcontrollore ATmega di Arduino. Non si tratta di un procedimento semplice e per ciascuna delle situazioni possibili il software applica delle soluzioni di trasposizione in altro codice intermedio fino alla compilazione vera e propria in codice oggetto.
Alcune funzioni, come ad esempio l'impostazione di un pin come ingresso o uscita digitale, portano l'IDE a generare del codice che – per compattezza – già include degli elementi in modo permanente e non con chiamate a routine a cui sono passati dei valori. In altri casi la funzione è realizzata come blocco di codice a cui vengono passati parametri e che restituisce – dove necessario – dei valori, senza che il blocco sia ripetuto ad ogni istanza nello sketch. La scelta di usare una, l'altra o una eventuale terza soluzione è stata ponderata per ciascuna funzione proprio nell'ottica di prestazioni e compattezza del codice oggetto.
Un'altra ottimizzazione riguarda l'inserimento nel codice oggetto delle sole funzioni utilizzate e quindi palesemente necessarie. Se non abbiamo inserito nello sketch un AnalogRead il software produrrà del codice oggetto che per certo non conterrà le routine di inizializzazione, gestione e lettura del convertitore A/D del microcontrollore.
Praticamente, non vengono inserite parti di codice ridondanti o non utilizzate per risparmiare sullo spazio della memoria **Flash** (o di programma) a disposizione.
Anche includendo **librerie**, se queste sono state scritte nel modo opportuno, non si ha quindi un aumento della memoria di programma utilizzata, se non per quelle poche istruzioni che sono necessarie a definire le variabili della libreria e a inizializzare l'eventuale hardware a cui la libreria fa riferimento. Per il resto anche le funzioni presenti nelle librerie vengono analizzate e gestite come quelle di base.

Gestire la RAM

La memoria **RAM** è quella dove sono ospitate le variabili e le costanti di programma ed è evidente che **2.048 byte** – quelli del microcontrollore di Arduino UNO - non sono affatto tanti. Ogni definizione di variabile si porta via un pezzetto di RAM e, per questo motivo, non è conveniente dichiararle senza pensare allo spazio che occupano. Nel primo volume "**Primi passi con Arduino UNO**" abbiamo spiegato come ciascun tipo di variabile occupi bit, byte o gruppi di byte e quindi verificare se ci basta un intero o se dobbiamo usare una variabile a virgola mobile - nei programmi di una certa complessità – può permetterci di arrivare al risultato o ci fa incappare in un superamento della RAM disponibile. Ovviamente per i programmi più semplici questo problema non sussiste, a meno che nella loro semplicità non facciano ampio uso **array** e **matrici**.
Per completezza, ripercorriamo il tema delle variabili, ricordando che queste sono dei seguenti tipi:

- void (**1b**)
- boolean (**1b**)

Contributed Libraries

Le attuali Contributed Librearies sono suddivise per categorie e sono:

Comunicazione (networking e protocolli):
Messenger – Per elaborare messaggi basati su testo provenienti dal computer.
NewSoftSerial – Una versione migliorata della libreria SoftwareSerial library originaria, incorporata in Arduino 1.0 e successivi come libreria ufficiale per la comunicazione seriale al posto della precedente.
OneWire – Controlla I dispositive della Dallas Semiconductor che usano il protocollo One Wire.
PS2Keyboard – Legge i caratteri da una tastiera PS/2.
Simple Message System – Invia messaggi fra Arduino e il computer.
SSerial2Mobile – Invia messaggi di testo o email utilizzando un telefono cellulare come modem e attraverso i comandi AT inviati tramite software serial.
Webduino – Una libreria modulare per Web Server da usare con la scheda Ethernet shield di Arduino.
X10 – Per inviare segnali X10 sulla rete elettrica.
XBee – Per comunicare fra XBee con la modalità API.
SerialControl – Controllo remoto di altri Arduino tramite un collegamento seriale.

Sensori:
Capacitive Sensing – Trasforma due o più pin in sensori capacitivi.

- char (**1B**)
- unsigned char (**1B**)
- byte (**1B**)
- int (**2B**)
- unsigned int (**2B**)
- word (**2B**)
- long (**4B**)
- unsigned long (**4B**)
- float (**4B**)
- double (**4B**)
- string - char array (**nB**)
- String - object (**nB**)
- Array (**nB**)

Partendo dall'inizio dell'elenco, abbiamo le variabili che occupano un solo bit, passando poi a quelle da un byte, due byte, quattro byte per arrivare a un numero variabile di byte in base al numero di elementi "n" della stringa o del vettore.
Anche se ogni sketch contiene le **dichiarazioni** delle variabili, dando di fatto la possibilità al compilatore di fare dei conteggi, con l'IDE di Arduino non si ha alcun messaggio di errore se viene superata la capacità della memoria RAM disponibile, mentre se lo sketch ha dimensioni superiori alla memoria Flash disponibile, questo genera un errore.
Superare la RAM disponibile è un problema serio in quanto compromette

Debounce – per leggere input digitali rumorosi, come ad esempio I pulsanti.

Display e LED:
Improved LCD library risolve alcuni bug di inizializzazione della libreria presente nella distribuzione standard di Arduino.
GLCD – Routine grafiche per display basati su chip KS0108 o equivalente.
LedControl – Per controllare matrici di LED o display a sette segmenti con un MAX7221 o un MAX7219.
LedControl – Un'alternativa alla libreria Matrix per pilotare svariati led con i chip Maxim.
LedDisplay – Controllo di un display LED a scorrimento di tipo HCMS-29xx.

Generazione di frequenze e Audio:
Tone – Genera onde quadre in frequenze audio su qualsiasi piedino e in background.

Motori e PWM:
TLC5940 – Gestione del controller a16 canali PWM a 12 bit.

Temporizzazione:
DateTime – Una libreria per tenere traccia di data e ora correnti, via software.
Metro – Aiuta a temporizzare delle azioni con intervalli regolari.
MsTimer2 – Usa l'interrupt del timer 2 per attivare una azione ogni "n" millisecondi.

Utilità:
PString – Una classe "leggera" per stampare sui buffer.
Streaming – Un metodo per semplificare le righe di programma per la stampa.

il funzionamento del programma e porta a effetti che vanno dal **blocco** nell'esecuzione a più subdoli **bug**, dovuti alla **sovrascrittura** delle variabili con valori errati. Senza un sistema per fare il debug, riuscire a capire perché il programma non si comporta come deve diventa molto difficile e quindi evitare di incappare in un superamento della capacità della RAM è fondamentale.
In questo discorso si inseriscono prepotentemente le librerie con le loro dichiarazioni di variabili: ogni libreria infatti ha al suo interno una serie di dichiarazioni che l'utente solitamente non vede, ma che occupano RAM. Per questo motivo l'utilizzo delle librerie diventa un aspetto significativo nell'ottimizzazione delle risorse ed in particolare della RAM.
Il compilatore, come abbiamo detto, incorpora solo il codice delle funzioni delle librerie usate nello sketch, ma non riesce a fare alcun intervento di scrematura sulle variabili e quindi se una libreria necessita di diverse variabili, avrà un impatto significativo sui 2K disponibili.
Da qui l'esigenza di includere ciò che serve, non tanto per un problema di dimensione del codice oggetto – che ci viene puntualmente comunicata al termine della compilazione – quanto per il rischio di includere qualche libreria con molte variabili ed incappare in un **overflow** della RAM (ovvero del superamento del limite superiore nello spazio di indirizzi a cui è mappata la memoria RAM e la conseguente scrittura di dati in posti che potrebbero essere registri di sistema o avere comunque effetti imprevedibili).
A titolo informativo, vale la pena segnalare come questo genere di pro-

blema sia una delle **vulnerabilità** più comuni ricercate dagli hacker per violare i sistemi come gli smartphone e i dispositivi di tipo embedded.

L' approccio funzionale

In Arduino le librerie sono un elemento distintivo per numerosi aspetti. Uno fra questi è la possibilità che le librerie hanno dato agli sviluppatori di mantenere l'ambiente semplice e di facile approccio, senza rinunciare alla scalabilità in termini di funzioni e potenzialità.
Limitando l'IDE a un insieme tutto sommato chiaro e ben circoscritto di funzioni basilari, chiunque si avventuri in un primo "giro delle funzioni" trova tutto il necessario per creare degli **algoritmi** anche di una certa complessità, ma non si perde in una marea di oscure funzioni legate all'hardware e ad applicazioni speciali. C'è il controllo funzionale delle porte in ingresso e in uscita, c'è il PWM, ci sono le porte analogiche in ingresso, mentre non ci sono funzioni per la gestione dei motori o per il dialogo con dispositivi SPI (banalmente, un sensore di temperatura intelligente), eppure non sarebbe stato difficile includerle.
Rispetto a un tipico sistema di sviluppo per microcontrollori, l'IDE di Arduino UNO è più asciutto e circoscritto nella sua versione "base", ma si espande e scala facilmente su livelli di complessità e funzionalità molto elevati grazie alle librerie.
Possiamo quindi affermare che il team ideatore di Arduino ha preferito enucleare e trasformare in librerie una serie di funzionalità tipiche del mondo dei microcontrollori proprio per limitare le difficoltà dei "primi passi" e di questo siamo certi che molti utenti ne sono grati.
Ad esempio, se da un lato è chiaro che la scrittura nella memoria EEPROM ha una sua utilità abbastanza frequente, è anche altrettanto chiaro che richiede già una conoscenza più approfondita del microcontrollore e della sua architettura hardware. Lasciare le funzioni per la sua lettura e scrittura come libreria non fa che rimandare ad un secondo momento – meno traumatizzante dei primi passi – lo studio di come questa memoria si comporta e va gestita.

Le librerie Standard

Partendo da questo processo di **semplificazione**, gli sviluppatori hanno definito l'IDE corredandolo con una serie di librerie comunque mantenute, documentate, esemplificate e gestite in modo solidale con l'IDE stesso, come a voler dire che Arduino UNO di base fa tutto quanto presente sia nelle funzioni base, sia nelle **librerie standard** accessorie.
Ovviamente non ha senso includere tutte le 10 librerie standard in ogni progetto, per non sprecare RAM, ma la loro conoscenza approfondita è alla base di un utilizzo avanzato di Arduino, prima di ampliarne ulteriormente le potenzialità con le librerie di **terze parti**.
Molte delle librerie di terze parti si appoggiano a una o più librerie stan-

dard e per questo il processo di "crescita" nella programmazione e nella prototipazione con Arduino passa dalle librerie standard (**Fig. 3**).
Queste sono:

- **EEPROM** – per leggere e scrivere su memorie EEPROM, che conservano i dati anche senza alimentazione.
- **Ethernet** - per collegarsi ad Internet e a una rete Ethernet attraverso Arduino Ethernet Shield.
- **Firmata** – per comunicare con il computer collegato ad Arduino secondo un protocollo standard.
- **LiquidCrystal** – per gestire i display a cristalli liquidi basati su controller HD44780 o compatibile (LCD).
- **SD** – per leggere e scrivere le schede di memoria SD.
- **Servo** – per controllare i servomotori.
- **SPI** – per dialogare con dispositivi che utilizzano l'interfaccia e il protocollo Serial Peripheral Interface (SPI).
- **SoftwareSerial** – per creare una porta seriale su qualsiasi pin digitale.
- **Stepper** – per gestire I motori passo passo (stepper)
- **Wire** - per gestire la comunicazione con uno o più dispositivi che impiegano la modalità di comunicazione a due fili Two Wire Interface (TWI/I2C).

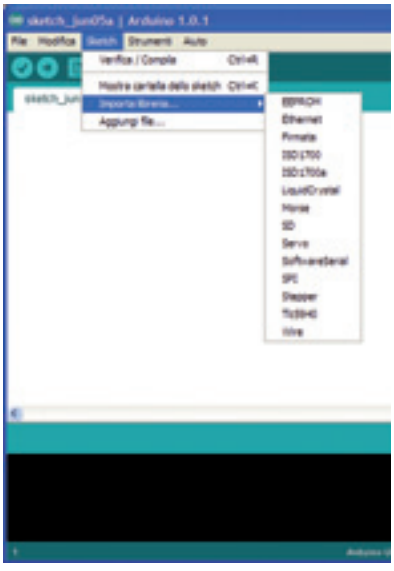


Fig. 3

Dando una scorsa alle varie funzionalità che queste librerie aggiungono, diventa evidente come Arduino UNO non abbia nulla da invidiare ai sistemi a microcontrollore ritenuti più completi o evoluti; nella parte pratica realizzeremo un **progetto** con ciascuna delle librerie, proprio per dimostrarne le potenzialità pratiche e l'impatto che hanno sulle funzionalità della scheda base, dotata o meno di eventuali shield legati alla libreria in uso.

Le librerie di terze parti

Oltre alle **dieci librerie** direttamente supportate dal team di Arduino, esistono molte altre librerie nate per le esigenze più svariate. Per cercare di mettere un po' di ordine in questo mare di codice, si può utilizzare un'organizzazione funzionale delle stesse, suddividendole in librerie legate a uno shield, librerie legate a un componente e librerie di tipo puramente software per funzioni particolari.
Tenere traccia di tutte queste librerie è un compito non da poco, ma proprio per questo motivo il team di Arduino sta sviluppando una nuova modalità con **ricerca** e **download** direttamente dall'IDE. In pratica l'IDE acquisterà una funzionalità online che gli consentirà di individuare e scaricare **on demand** una qualsiasi delle librerie di terze parti inclusa nel gruppo "terze parti".
Perché una libreria entri a far parte di questo gruppo lo sviluppatore accetta di supportare e mantenere nel tempo il suo lavoro, creando librerie che rispondono da un lato ad alcuni requisiti circa la loro documentazione

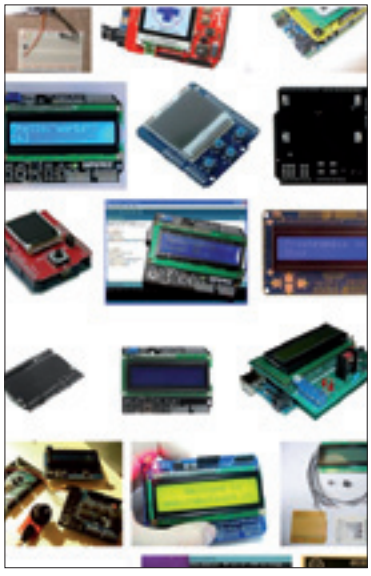


Fig. 4
Un assortimento di shield LCD.

e struttura, ma soprattutto accetta di non abbandonare il codice a sé stesso in caso di problemi o richieste.
Questo sistema dovrebbe essere incorporato in una delle prossime release dell'IDE di Arduino e nel momento in cui scriviamo è in pieno svolgimento il lavoro di promozione dell'iniziativa e costruzione della **repository**.

Contributed Libraries

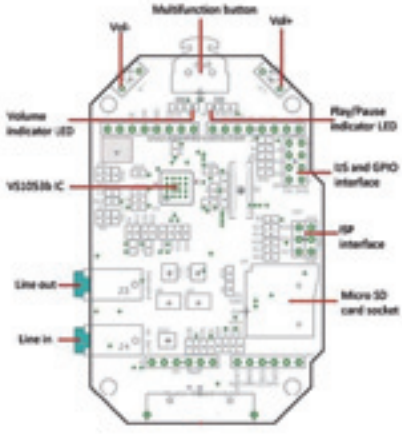
In attesa di questo sviluppo, il sito di Arduino offre comunque una raccolta di librerie di terze parti – definite come **contributed libraries** - per le quali esiste una pagina sul **Wiki** che contiene informazioni circa struttura, comandi e applicazioni; non si tratta di librerie che fanno parte della distribuzione **core** dell'IDE, ma il loro livello di testing, documentazione e supporto le rendono comunque un punto di riferimento per chi scrive software con l'IDE di Arduino.

Librerie legate agli shield

A questa categoria appartengono le librerie che sono state sviluppate per supportare in modo specifico uno **shield** o un particolare **chip** alla base di una serie di shield equivalenti. In alcuni casi, infatti, l'idea di un particolare shield è venuta a più persone e grazie alla natura **open** di Arduino non ci sono stati problemi a immettere sul mercato più shield con la medesima funzionalità e addirittura il medesimo chip. Come risultato, la libreria è diventata **compatibile** con la serie di shield, purché implementati in modo equivalente a livello di hardware.
Sul mercato esistono quasi **cinquecento shield** e molti di questi utilizzano circuiti integrati, componenti attivi e display per svolgere funzioni specifiche; chi ha progettato questi shield – dove necessario – ha sviluppato anche la libreria richiesta per gestire con l'immediatezza e la semplicità tipiche di Arduino i vari aspetti dello shield. In alcuni casi, lo shield è l'evoluzione di qualche applicazione nata su breadboard e successivamente trasformata in prodotto commerciale. In questo caso la libreria potrebbe essere già stata sviluppata a supporto del componente alla base dello shield e quindi il produttore si è dedicato solo alla realizzazione dell'hardware lasciando la libreria nelle mani dello sviluppatore originale per il supporto e gli eventuali successivi sviluppi.
Prendiamo ad esempio i vari shield con un display LCD da 2 o 4 righe alfanumeriche, basati su moduli che hanno un controller compatibile con l'Hitachi **HD44780**: sono tutti molto simili fra loro e utilizzano tutti la medesima libreria standard LCD fornita con l'IDE. Grazie alla flessibilità della libreria, i pin di collegamento del display possono essere adattati alla specificità dello shield, mentre le varie funzioni di scrittura dei caratteri e di posizionamento del cursore restano invariate. In pratica si possono usare sketch scritti per applicazioni con display, che includono la libreria LCD, con diversi shield purché questi siano dotati dello schermo richiesto dallo sketch in termini di righe e colonne.

Solo passando a versioni più evolute di shield, con schermi grafici o a colori, la libreria diventa potenzialmente specifica, anche se per i display esistono già numerose librerie dedicate a una grande varietà di pannelli e controller, montati in shield commerciali che anche in questo caso possono far riferimento a queste librerie (**Fig. 4**).
Shield basati su chip poco usuali o particolarmente nuovi sono il caso tipico in cui la libreria è invece parte integrante dello shield e viene sviluppata dal produttore dell'hardware. Anche in questo caso, lo spirito "aperto" di Arduino comporta nella maggior parte dei casi la disponibilità di tutto il materiale informativo e dei sorgenti per modificare e adattare le librerie e l'hardware alle proprie esigenze.
Un esempio di shield con libreria dedicata e realizzata come da indicazioni del team è il **Music Shield** di Seeedstudio che permette la riproduzione di file audio digitali di vario formato e che viene fornito con una libreria già giunta – al momento in cui scriviamo – alla versione 1.2 grazie anche all'aggiornamento per Arduino UNO (**Fig. 5** e **Fig. 6**).
Qualche shield può anche essere realizzato senza la libreria di supporto strutturata come da consuetudine, ma con del **codice di supporto** da includere nel proprio sketch.
Un esempio di questo tipo di approccio è lo shield **FM Radio** di **Curious Inventor**: è basato su un chip **Si4701** - un sintonizzatore FM digitale di recente progettazione – ed è in grado di decodificare anche le informazioni **RDS** per avere il nome della stazione radio o il titolo della canzone. Con lo shield viene fornito uno sketch di esempio e un file "*FMApi.h*" da includere nei propri sketch per controllare lo shield. La soluzione è simile a

Fig. 5
Lo shield musicale di Seeedstudio.



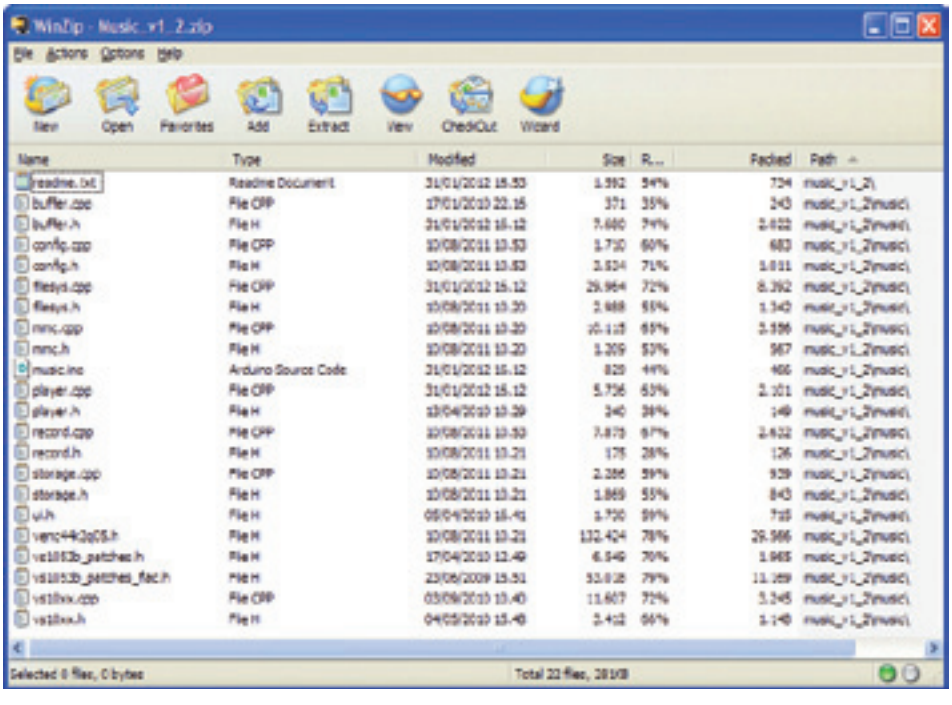


Fig. 6
Il codice fornito come librerie e sorgenti vari per utilizzare lo shield Music, assieme all'esempio "music.ino" visibile nell'elenco.

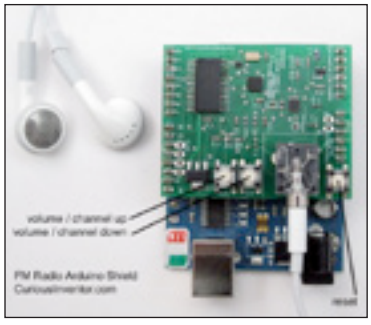


Fig. 7

quella con una libreria normale, ma sono assenti i file con le parole chiave ".txt" e il ".cpp" (**Fig. 7**).
La mancanza di questi due file non compromette la compilazione del codice, ma non comporta ad esempio l'evidenziazione delle **parole riservate** per la funzione nello sketch.

Librerie legate ad hardware specifici

Ci sono numerosi componenti che dispongono di piedini di controllo digitali o che addirittura prevedono dei **registri** in cui scrivere dei valori per il controllo delle funzionalità. Dato che l'evoluzione del mondo dell'elettronica di consumo è stato verso la gestione e l'integrazione con microcontrollori embedded, è quindi abbastanza normale che i chip più disparati siano stati progettati con tutto il necessario per l'interfacciamento con modalità standard come **SPI** o **I²C** o altri protocolli di comunicazione comunque standard e ben documentati.
Se l'interfacciamento è un aspetto tutto sommato semplice, la gestione delle funzioni del chip potrebbe non esserlo per una scarsa documentazione o per una intrinseca complessità del circuito che vogliamo pilotare.
In soccorso arriva il lavoro degli utenti più **esperti** che – dopo ore e ore di sperimentazione e di lettura di data sheet, mettono a disposizione della comunità degli utenti le loro librerie dedicate.
Gli esempi sono numerosi ed è anche possibile che per un medesimo componente si siano attivati in più utenti, ognuno all'insaputa dell'altro, giungendo alla realizzazione di soluzioni tecniche diverse, ma ugualmente funzionanti.
Un esempio sono i chip di controllo per i LED, da montare in strisce o in catene e gestibili singolarmente tramite una interfaccia da due o quattro fili, più o meno conforme allo standard SPI. Se cercate in rete **WS2801** – uno di questi chip per la gestione dei **pixel LED RGB** – troverete varie soluzioni e leggendo i vari commenti nei forum potrete riscontrare come il lavoro sia passato di mano in mano, da un utente all'altro, fino a diventare una summa degli sforzi di ciascuno. Con questo lavoro collaborativo, emergono anche i limiti degli approcci adottati e – conseguentemente – delle soluzioni che portano a salti qualitativi anche importanti. Nello specifico caso della libreria per gestire il WS2801 e gli altri controller pixel LED, si è arrivati alla conclusione che il collo di bottiglia era la trasmissione delle informazioni alla serie di chip, collegati con l'interfaccia SPI. Da qui la decisione di passare da una seriale su qualsiasi pin e gestita interamente via software, alla SPI del microcontrollore, disponibile solo su pin ben definiti, ma capace di prestazioni decisamente superiori.
Nasce quindi **FastSPI**, che è attualmente la libreria più articolata e veloce per gestire le strip LED con chip di controllo di vari tipi; le sue prestazioni arrivano a centinaia di frame al secondo per serie di LED di vari metri di lunghezza. Cercando il nome dei vari controller si trovano però anche delle altre librerie e solo con un po' di studio e di sperimentazione è possibile

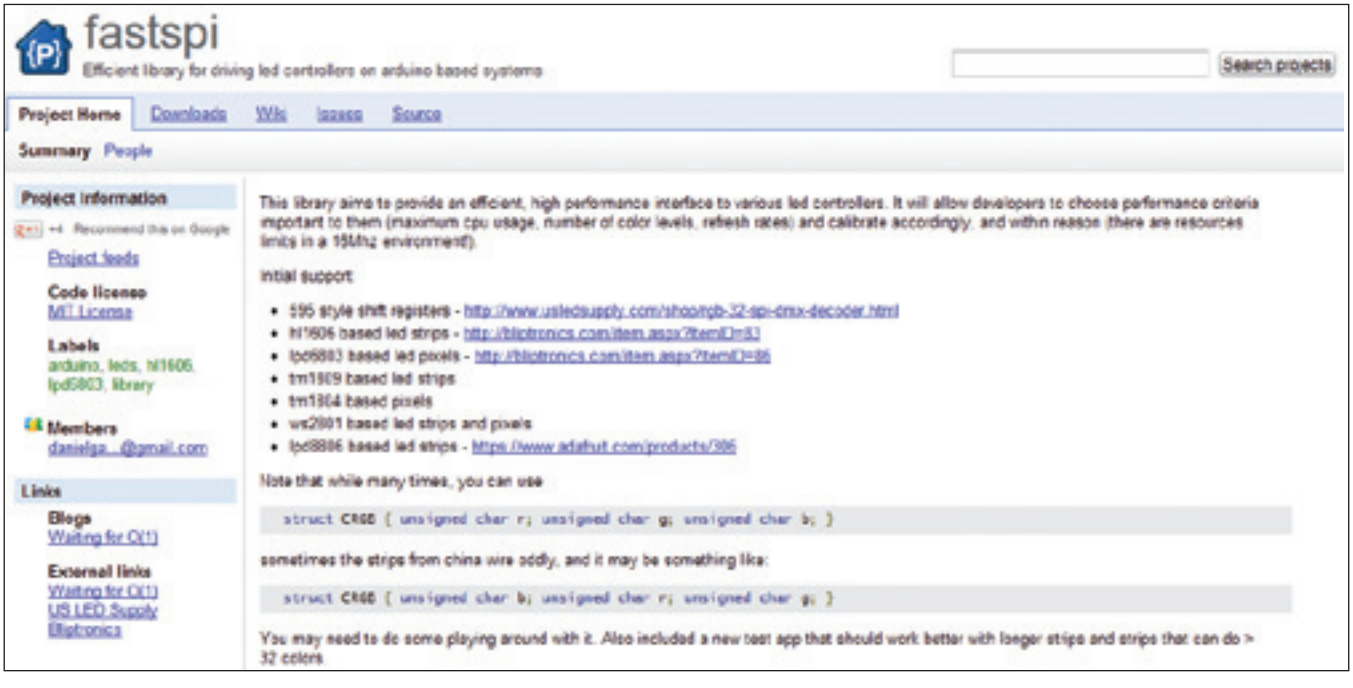
capire quale sia quella più adatta al progetto che si sta portando avanti. In generale, per scoprire se qualche componente che volete utilizzare ha già il supporto di una libreria, basta inserire in **Google** il nome del componente seguito da Arduino. Se qualcuno ha fatto qualcosa, sarà facile trovarlo, altrimenti potrebbe toccare a voi iniziare a supportare il componente con una vostra libreria. Se il componente in questione ha degli equivalenti o dei dispositivi simili, può valer la pena cercare se esiste qualcosa per loro e adattare il codice solo per le differenze funzionali (**Fig. 8**).

Librerie funzionali

Alcune librerie nascono con il solo scopo di aggiungere funzionalità alla piattaforma Arduino senza che questo riguardi del nuovo hardware o uno shield. Nelle Contributed Libraries troviamo diverse librerie **funzionali**, dedicate alle temporizzazioni, alla comunicazione e alla gestione di caratteristiche del microcontrollore che le librerie di base non sfruttano appieno, ma in rete si possono trovare ulteriori librerie funzionali decisamente curiose e interessanti.

TVOut, ad esempio, è una libreria che utilizzando tre pin digitali riesce a creare un segnale televisivo, visualizzabile su qualsiasi televisore dotato dell'ingresso video composito. Con due resistenze i segnali vengono portati all'opportuno livello elettrico ed il software, con le sue funzioni, provvede a generare testi e disegni secondo il programma che si appoggia a questa libreria. Sul sito di distribuzione si trova addirittura un **Tetris** pronto per essere giocato. La risoluzione gestita è molto limitata, ma questo non impedisce di realizzare applicazioni interessanti. Prima che qualcuno

Fig. 8



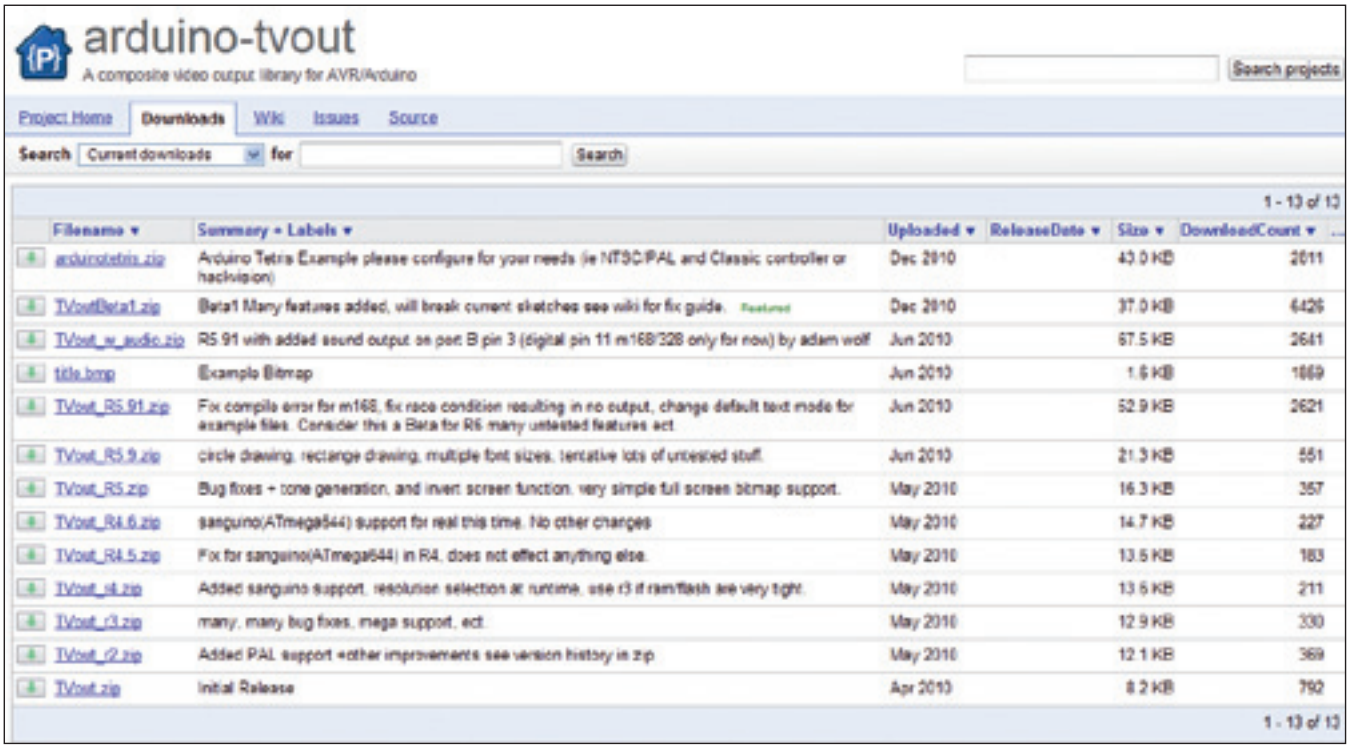


Fig. 9

pensi a qualcosa di sofisticato, vale la pena segnalare che questa libreria riesce solo a generare un'uscita televisiva monocromatica.
Altre librerie sono raccolte di algoritmi per la conversione da un formato all'altro o per la gestione di qualche aspetto specifico. Più in generale, la creazione di una libreria permette di scrivere codice riutilizzabile con facilità e inoltre la libreria può essere condivisa con altri utenti contribuendo alla disponibilità di materiale nella community (**Fig. 9**).

8

La libreria Servo

Pilotare fino a 12 servo significa poter realizzare sistemi robotizzati anche di una certa complessità e la libreria Servo si prende cura dei vari aspetti legati alla generazione dei segnali necessari a gestire il posizionamento degli assi in modo stabile e preciso. Con due servo, un brandeggio che punta i numeri della tombola su un tabellone diventa un gioco da ragazzi.

La vocazione **robotica** della piattaforma Arduino trova un valido supporto nella libreria **Servo**, sviluppata per dare un controllo software più preciso e strutturato sui motori di tipo servo. Con questa libreria, il normale funzionamento del PWM viene rimpiazzato da routine più mirate sulla specifica applicazione e per questo vengono rese disponibili funzioni e metodi che permettono di operare su una serie di motori.
Sulla maggior parte delle schede Arduino questa libreria permette di pilotare fino a **12 motori** in modo autonomo e indipendente, mentre sulle schede di tipo Mega, il numero di motori arriva a 48. Questo utilizzo massiccio dei pin digitali per la generazione del PWM destinato ai servo rende purtroppo alcune funzioni indisponibili, come l'uscita "analogica" in PWM sui pin 9 e 10 anche se ai due pin non sono assegnati dei servo. Nel caso delle schede Mega, fino a 11 motori non comportano la perdita della funzione analogWrite(), mentre il collegamento di 12 o più motori disabilita il PWM di analogWrite() sui pin 11 e 12.

L'anatomia del servomotore

I motori che possono essere collegati ad Arduino sono anche di tipo servo, ovvero degli speciali motori nati anni fa per essere inseriti nei **modelli radiocomandati**. Fra le loro caratteristiche c'è il funzionamento a **5 volt** e il pilotaggio tramite un segnale a **onda quadra** il cui rapporto fra la semionda positiva e quella negativa determinano la posizione del perno del motore. Praticamente si usa un segnale PWM per definire la posizione del perno del motore e i motori solitamente possono ruotare di **180 gradi**; solo in specifici modelli il perno è senza blocco nella rotazione e il segnale PWM come effetto ha la modifica nella **velocità** di rotazione. Anche se a prima vista i servo sembrano dei dispositivi abbastanza semplici, al loro

Fig. 1

interno si cela sia una parte meccanica, sia una parte elettronica di una certa complessità (**Fig. 1**).

La parte meccanica serve a trasformare le veloci rotazioni del motorino elettrico in continua in circa mezzo giro del perno principale, ottenendo una coppia molto maggiore rispetto a quella del perno del motorino. Questo processo di **demoltiplica** si riflette in una certa **lentezza** nella rotazione del perno, ma consente anche di avere una certa precisione negli spostamenti.

La parte elettronica si basa su un potenziometro solidale al perno principale e questo accorgimento permette alla circuiteria presente di confrontare il segnale ricevuto con uno generato internamente. L'uso del potenziometro spiega anche come la rotazione sia limitata a 180 gradi.

Le applicazioni di un servomotore

Nati principalmente per gestire nei radiomodelli le parti meccaniche quali le superfici di controllo di un'ala o di un timone di profondità, piuttosto che uno sterzo in una macchinina, questi motori sono diventati interessanti per il mondo della robotica e dell'hacking grazie alla loro semplicità operativa e alla loro capacità di esercitare una certa forza (o resistenza) con un'alimentazione a 5 volt.

Sono ormai abbastanza comuni i robot realizzati praticamente solo da servomotori fissati fra loro con semplici staffe, dove equilibrio, movimento e capacità di spostarsi dipendono in parte dai **sensori** abbinati alla scheda intelligente che controlla i motori, ma soprattutto dalla complessità e validità degli **algoritmi** che si occupano di comandare in modo **coordinato** e coerente con gli stimoli esterni i motori. Restare in equilibrio o coordinare una camminata su due gambe può sembrare a un adulto qualcosa di semplice, ma basta osservare i problemi che ha un infante che passa dal gattonare ai primi passi per ricordare quanto un semplice passo in avanti richieda non poca coordinazione. La natura dei servomotori, però, rende tutto sommato semplice avere un perfetto controllo della posizione di ciascun motore, senza dover impiegare sofisticati sistemi di feedback: si imposta la posizione e il motore "esegue", mentre la libreria tiene traccia dell'ultimo valore scritto per ciascun motore (**Fig. 2**).

Fig. 2

La libreria Servo

Con l'inclusione della libreria, nell'IDE diventa disponibile la classe Servo e una serie di metodi per gestire i motori collegati.

Servo
attach()
attached()
detach()
write()
writeMicroseconds()
read()

Servo

Con la creazione di un'istanza della classe abbiamo a disposizione un **oggetto** di tipo **servo** con cui eseguire sia le funzioni di impostazione, sia quelle di pilotaggio del motore. La sintassi è:

Servo <<nome istanza>>;

e in questo caso risulta molto utile poter creare delle istanze multiple della medesima classe per gestire contemporaneamente più motori. La scelta del nome dell'oggetto dovrebbe essere fatta in modo oculato così da evitare da un lato confusioni e dall'altro aiuti ad abbinare mentalmente hardware e chiamata software.

attach(pin, min, max)

Questo ci permette di associare un oggetto servo al piedino hardware **fisico**; ovviamente possiamo solo scegliere un pin di tipo digitale, ma non abbiamo il problema della limitazione ai soli pin 3, 5, 6, 9, 10 e 11 che invece avremmo per l'*analogWrite()*. I due parametri, oltre al numero di pin da utilizzare, riguardano la durata minima e quella massima dell'impulso trasmesso al motore per le due posizioni di inizio e fine corsa. Esistono dei valori di default che il software utilizza in caso di parametro non specificato, ovvero 544 microsecondi per il minimo e 2400 microsecondi per il massimo; si tratta di valori non conformi agli standard abituali, ma pensati per garantire il raggiungimento di inizio e fine corsa. Se i motori servo utilizzati prevedono valori diversi, è quindi possibile intervenire tarando in modo fine l'ampiezza dell'impulso di inizio e fine corsa, poi la libreria provvede a suddividere in 180 passi l'intervallo disponibile. La sintassi è

<<nome istanza>>.attach(pin,min,max);

con pin che deve essere uno dei pin digitali, mentre la durata dell'impulso è espressa con un intero. Se dovete agire su questi valori, vi suggeriamo di controllare la documentazione del servomotore e di fare delle verifiche sul campo collegando anche in modo temporaneo e volante il dispositivo ad Arduino. Individuare correttamente i valori di rotazione minima e massima è utile se pensate di dover far lavorare i motori anche i queste due posizioni estreme: se i valori numerici di minimo e massimo portano il motore a posizionarsi in una modalità di "sforzo" costante, ovvero che cerca di superare i blocchi di inizio e fine corsa, assorbe anche molta corrente e rischia di bruciare. Se impostando la rotazione a 0 e 180 gradi sentite i motori ronzare o vedete che anche variando i valori a 1,2,3 e 177, 178 e 179 non ci sono spostamenti, allora è il caso di intervenire sui valori di min e max.

attached(pin)

Questo comando ci permette di sapere se abbiamo un motore collegato a

uno specifico pin. Non si tratta di un test fisico o elettrico sul pin, ma semplicemente del controllo nelle variabili alla ricerca dell'assegnazione di quel pin tramite un comando *attach()*. Ovviamente dobbiamo utilizzare il comando associandolo a un oggetto Servo e quindi potremmo fare un test su un piedino a cui è collegato un servo ancora "attached", ma usando un'istanza di Servo non corrispondente, il risultato sarebbe FALSO.
Il comando ha comunque un suo motivo di esistere in quanto solo scollegando tutti i motori, ovvero **liberando** tutti i piedini **assegnati** con degli *attach()* è possibile tornare ad unsare il PWM e *analogWrite()* sui piendini 9 e 10.

detach(pin)

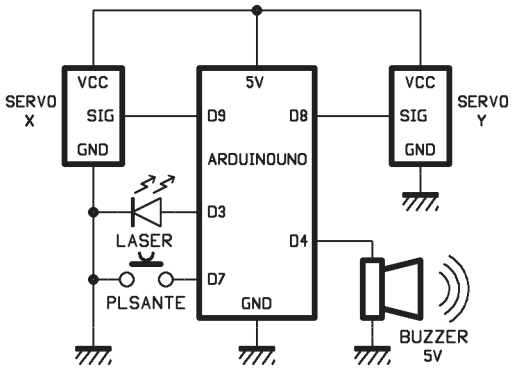
Il contrario di *attach()*, da utilizzare per scollegare un oggetto Servo da un piedino. Poiché un'istanza della classe Servo può gestire un unico piedino alla volta, questa istruzione potrebbe essere utilizzata per scollegare un piedino in preparazione al collegamento ad un altro piedino a cui è connesso un servomotore diverso dal precedente. In pratica possiamo scrivere del codice in cui il motore da pilotare è una delle variabili in gioco, mentre il codice che pilota il motore stesso è sempre quello, magari sotto forma di subroutine all'esterno del loop principale. Come indicato in precedenza, solo quando tutte le variabili relative ai pin sono "detached" è possibile tornare a usare *analogWrite()* sui pin 9 e 10 anche se non sono stati assegnati ad alcun motore.

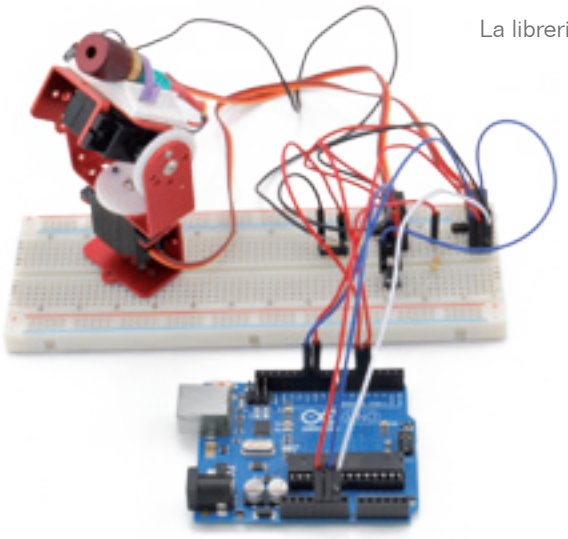
write(angolo)

Inviando questo comando a un motore opportunamente "attached" a un piedino determiniamo la produzione sul pin di un segnale PWM la cui durata è proporzionale al valore *angolo* che può andare da **0** a **180**. Se il servo è conforme agli standard de facto per questi dispositivi, allo 0 corrisponderà un estremo della rotazione, a 90 una posizione centrale e a 180 l'altro estremo. Nel caso si tratti di un servo a rotazione continua, a 0 corrisponde la massima velocità di rotazione in un senso (antiorario) e a 180 il massimo della velocità nell'altro senso (orario), con 90 che corrisponde a uno stallo. Se, con un servo da 180 gradi, ai due valori estremi si nota che il motore "sforza", allora è il caso di intervenire sui parametri *min* e *max* della funzione *attach()* alterandoli per avere un posizionamento corretto e **senza sforzi**. Tutti i motori servo hanno una effettiva rotazione del perno di poco superiore ai 180 gradi per consentire una effettiva rotazione di 180 gradi senza che il motore urti con i due fine corsa. Ricordate che *min* e *max* sono variabili specifiche per ciascuna istanza della classe Servo e quindi ciascun motore può avere la sua taratura.

writeMicrosenconds(uS)

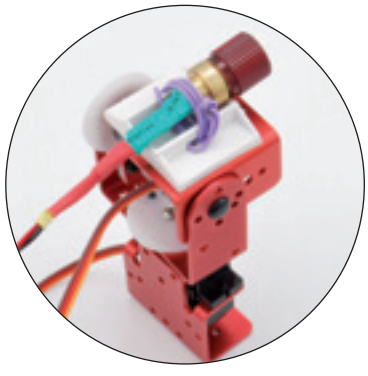
Con questa funzione il valore espresso non è in gradi, ma in microsecondi e permette di gestire il servomotore direttamente con impulsi di durata definita dall'utente e non derivati da un calcolo basato sui valori di *min, max*





e sull'angolo scelto con *write()*. Si tratta di una funzionalità che richiede la conoscenza dei **parametri specifici** del servomotore collegato, così da poter inviare impulsi con durata nell'intervallo consentito dal dispositivo e, soprattutto, ottenendo i risultati attesi. Le specifiche standard parlano di **1000** microsecondi per la posizione di 0 gradi, **1500** microsecondi per la posizione centrale (90 gradi) e **2000** microsecondi per la posizione di 180 gradi. Il tutto in un periodo di 20 millisecondi. Facendo un rapido conto, si evince che non si tratta di un PWM come quello generato con analogWrite(), ma di un treno di impulsi in cui la cadenza è fissa a 20 millisecondi – pari a **500 Hz** – e la parte positiva dell'onda quadra parte da un minimo di 1 millisecondo fino a un massimo di 2 millisecondi, pari a un duty cycle variabile fra il 5 e il 10 %. Il fatto che si esprima il valore in microsecondi è indispensabile per poter pilotare il motore con rotazione del perno di circa **due decimi** di grado per step, pari appunto a un microsecondo. Rispetto a *write()*, questa funzione offre una risoluzione **cinque volte** superiore.

Ecco un puntatore laser adattato per le nostre esigenze di pilotaggio diretto tramite pin digitale di Arduino.



read()

Simile alla funzione *attached()*, con *read()* possiamo sapere qual è l'ultimo valore che è stato passato all'oggetto Servo con una istruzione *write()*. Il valore restituito varia fra 0 e 180 gradi ed è relativo alla sola istruzione *write()*. Se il motore è stato pilotato con *writeMicroseconds()* non potremo utilizzare con successo questa funzione.
Non ci sono altre funzionalità legate alla libreria Servo e quindi possiamo procedere con un'applicazione pratica di questo genere di motori per realizzare qualcosa di originale…

Il progetto: estrattore visuale di numeri della tombola

Un classico delle feste di paese è la tombolata, con ricchi premi legati alla gastronomia locale e l'immancabile opera d'arte del pittore. Il cartellone con i numeri è ormai parte dell'attrezzatura dell'oratorio o della sala comunale delle feste e inventarsi un gadget tecnologico per svecchiare il gioco non è poi così difficile se abbiamo a nostra disposizione un Arduino.

L'hardware

La nostra idea è quella di realizzare un sistema **motorizzato** che punta un fascio di luce concentrato e sagomato sulla casella del numero estratto, con la generazione casuale della sequenza di estrazione e l'avanzamento pilotato da un pulsante da un numero al successivo. Questo ci permette di usare proprio dei servo motori e mettere quindi in pratica quanto abbiamo imparato sulla libreria specifica.
Se non volete usare una torcia, ma qualcosa di più tecnologicamente avanzato, potete smontare un puntatore laser di quelli stile portachiavi – a norma di legge da 1 mW rosso - ed estrarre tutta la parte elettronica, meccanica e ottica. In questo caso, potete usare direttamente un pin digitale per accendere e spegnere il laser dato che l'assorbimento è molto basso e l'alimentazione è a 5V. Nel software il pin scelto è D3.
Per poter utilizzare i motori, abbiamo bisogno di un supporto meccanico e quello che suggeriamo è composto da tre semplici staffe in alluminio, progettate per creare un piccolo **brandeggio** sui due assi **X** e **Y**. Il kit si chiama **9g Servo Bracket** ed è adatto per i servomotori di tipo "micro servo". I due motori sono collegati come **myservoX** al pin **D9** e **myservoY** al pin **D8**.
Nella parte superiore applicheremo il nostro puntatore a led o laser, scegliendo qualcosa che sia in grado di creare un marcatore luminoso adatto alle dimensioni e alla distanza del tabellone.
Sempre nello spirito dell'hacking di oggetti esistenti, quando andate a procurarvi il puntatore laser, cercate anche uno di quei portachiavi che fanno i versi degli animali: con un paio di euro avrete un piccolo **generatore** di effetto audio anche a volume considerevole con cui segnalare l'estrazione di un numero senza dover aggiungere hardware di amplificazione o codice specifico: basta collegarlo a uno dei pin digitali per avere il suo funzionamento dato che anche questo dispositivo funziona circa a 5V e assorbe molto poco. Questo elemento può essere omesso se si preferisce un dispositivo più discreto. Se accettate la nostra proposta, collegate l'elettronica del portachiavi al pin **D4** come positivo e il negativo a **GND**.
Un ultimo elemento del nostro circuito è un **pezzo di filo collegato ad A0**: questo serve a rendere assolutamente instabile la lettura della porta analogica, così da avere un buon punto di partenza per il nostro generatore di numeri casuali.

Il software

Il codice per questo progetto si compone di varie parti funzionali e, per una maggiore pulizia e compattezza, anche di alcune routine all'esterno del loop principale. Oltre alla parte di gestione dei motori per puntare sul tabellone la casella corrispondente al numero estratto, il nostro Sketch dovrà gestire in modo **intelligente** la sequenza di numeri della tombola; non è infatti possibile generare a caso numeri da 1 a 90 perché nel corso della medesima tombolata potrebbe uscire il medesimo numero più di una volta e quindi dobbiamo "mischiare" i numeri come si fa nella realtà con il sacchettino.

Per noi il **sacchettino** sarà un **vettore** con i numeri inizialmente ordinati da 1 a 90, quindi ci appoggeremo a una routine di mescola che scambia ciascuna posizione del vettore dei numeri con il contenuto di una cella scelta a caso nel vettore stesso. Come contorno abbiamo una routine che fa lampeggiare il laser sul tabellone per attirare l'attenzione e una routine che aiuta a regolare la posizione del brandeggio e del tabellone.
Per non creare un programma troppo lungo, abbiamo accettato alcuni compromessi, ma siamo certi che partendo dal nostro codice, alcuni di voi si cimenteranno nello sviluppo di quelle parti che abbiamo intenzionalmente lasciato "irrisolte". Nello specifico va sviluppato il codice per:

- Taratura dei quattro angoli del tabellone tramite due tasti accessori (+ e -)
- Passaggio nella gestione dei motori da *write(gradi)* a *writeMicroseconds(uS)* per aumentare la precisione e poter gestire un posizionamento compensato come da punto successivo.
- Gestione del posizionamento sulle caselle tenendo conto della deformazione dovuta al fatto che il tabellone è piano, mentre il puntatore proietta presupponendo un tabellone porzione di una sfera.

Le dichiarazioni

Nelle prime righe dello sketch abbiamo posizionato l'include per la libreria "*Servo.h*", la creazione delle due istanze della classe **Servo** – chiamate *myservoX* e *myservoY* – e la serie di dichiarazioni di tipi e variabili. Abbiamo predisposto già alcune variabili apparentemente inutili al fine di dare già un aiutino a chi vuole sviluppare il software con le funzionalità appena indicate.
Due parole merita invece il vettore dove abbiamo inserito in maniera molto pedissequa i 90 numeri. Evidentemente è possibile usare un ciclo di *for* per popolare il vettore, ma alla fine anche questa semplice tabella di valori risulta compatta e leggibile. A voi trasformarla in ciclo se volete del codice più "elegante".
La riga immediatamente successiva alla dichiarazione di *Sequenza []* serve per memorizzare in *Elementi* il numero di elementi nel vettore. Sappiamo che sono 90 numeri perché li abbiamo inseriti noi, ma è utile imparare che nel caso di un vettore possiamo usare *sizeof(vettore)* per avere la dimensione del **vettore**, mentre con *sizeof(elemento)* abbiamo le dimensioni del **tipo** di oggetto, che nel nostro caso è un **intero**. Dato che tutti gli elementi del vettore sono interi, dividendo le dimensioni totali del vettore per le dimensioni del tipo intero, avremo il numero di **elementi**.

```
#include <Servo.h>
Servo myservoX;  // Istanza per motore asse x
Servo myservoY;  // Istanza per motore asse y

int Xpos = 0;    // posizione x
int Ypos = 0;    // posizione y
int Tasto=7;     // il tasto di interazione è su D7
```

```
int Laser=3;      // laser o torcia LED sul pin 3
int AudioPin=4;   // gadget audio sul pin 4
int Xmin=100;     // l'angolo in alto a sinistra del tabellone
int Ymin=100;
int Lato=20;      // il lato del nostro tabellone
int Ripeti=30;    // numero di lampeggi del Laser
int Numero;       // qui memorizziamo il numero estratto

int Sequenza[] = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,
                 21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,
                 41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,
                 61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80,
                 81,82,83,84,85,86,87,88,89,90}; // i numeri della tombola
int Elementi = sizeof(Sequenza) / sizeof(int);
```

Il setup()

La parte successiva è quella classica di *setup()* dove inizializziamo i vari pin utilizzati per il tasto di controllo, il laser e il generatore audio. Poi inizializziamo la porta **seriale** sulla quale stamperemo i numeri estratti, permettendo a chi gestisce la postazione di vedere lo **storico** delle estrazioni e anche controllare cosa è uscito in caso di "puntamento approssimato" da parte del sistema meccanico.

Segue il calcolo della variabile *Seme* per il generatore di numeri casuali: fra le tante opzioni possibili, abbiamo scelto la tecnica dell'ingresso analogico lasciato elettricamente aperto e addirittura caricato di **disturbi** e interferenze attraverso un pezzetto di filo in stile antenna.

Il *Seme* determina la sequenza di numeri pseudo casuali e quindi al medesimo seme corrisponde la medesima sequenza. Avere un seme molto variabile non è banale e infatti dopo varie prove abbiamo trovato che la doppia lettura e il pezzetto di filo erano una buona garanzia, mentre una singola lettura portava spesso al 1023 (fondo scala).

Ora che abbiamo un buon punto di partenza per la nostra sequenza di numeri casuali possiamo chiamare la funzione *Mischia()* alla quale passiamo come parametri sia il vettore *Sequenza*, sia il numero di *Elementi* che lo compongono. Come ritorno, avremo il nostro vettore opportunamente "mischiato" e pronto per essere utilizzato.

Le due istanze dei servomotori, prima di poter essere utilizzate vanno collegate ai pin e definite nei valori minimi e massimi degli impulsi ed ecco quindi *myservoX.attach(pin,min, max)* seguito da *myservoY.attach(pin,min, max)* a cui facciamo seguire mezzo secondo di pausa per consentire il posizionamento e l'assestamento dei due servo.

Il passo successivo non riguarda la scheda e l'hardware, ma il tabellone che deve essere posizionato con un ragionevole allineamento. Per questo abbiamo previsto un paio di routine che ci aiutano proprio in questo e sono anche un primo passo verso la regolazione interattiva degli angoli del tabellone.

Innanzitutto il tabellone dovrebbe essere realizzato in modo simile a quello riportato in **Fig. 3**.
Nove righe da dieci caselle spaziate in modo **uniforme**. Non si tratta di un quadrato in quanto abbiamo solo nove righe e di questo terremo conto nel codice, come vedremo fra poco. La taratura della posizione fa riferimento ad *Area()* che si trova più in basso nel codice e che analizzeremo al momento opportuno. Al termine di questa procedura, ritorniamo nel *setup()* e passiamo all'ultimo step di questa sezione: un secondo di segnale acustico generato dal nostro portachiavi cannibalizzato.

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90

Fig. 3

```
void setup()
{
  pinMode (Tasto,INPUT);  // il tastino di controllo
  pinMode (Laser,OUTPUT); // il LASER
  pinMode (AudioPin,OUTPUT); // il generatore AUDIO

  Serial.begin(9600);         // per sapere cosa viene estratto dal monitor seriale
  int Seme = analogRead(0);     // prepariamo un buon seme per la generazioneì
  delay (random(50));           // di una sequenza pseudocasuale con cui
  Seme = Seme + analogRead(0); // mischiare la sequenza; l'ingresso analogico
  randomSeed(Seme);             // a cui colleghiamo un filo è un buon riferimento
                                // per avere valori casuali.

  Mischia (Sequenza, Elementi);// invochiamo la routine di "miscelazione" della sequenza

  myservoX.attach(9,500,2500);  // l'oggetto myservoX è sul pin 9
  myservoY.attach(8,500,2500);  // l'oggetto myservoY è sul pin 8

  delay(500);

  Area ();  // facciamo la taratura della posizione di brandeggio e tabellone

  digitalWrite(AudioPin, HIGH);
  delay (1000);
  digitalWrite(AudioPin, LOW);


  delay (500);

}
```

Il loop principale

Grazie all'impiego di subroutine e al fatto che i numeri sono già stati opportunamente mischiati, quello che resta nel loop principale è tutto sommato semplice e lineare. In pratica avviamo un ciclo da 90 iterazioni e per cia-

scun ciclo svolgiamo due semplici azioni: attendiamo la **pressione** del tasto di controllo una prima volta e nel frattempo visualizziamo "numeri a caso", mentre quando il tasto dell'estrazione viene premuto, **scriviamo** sulla console seriale il **numero estratto**, facciamo un po' di rumore e **posizioniamo** il **puntatore** sulla **casella**, usando un movimento oscillatorio con offset decrescente per aiutare i motorini a trovare la posizione corretta.
Attendiamo continuando ad evidenziare il numero tramite l'oscillazione e quando il tasto viene premuto si torna ai numeri a caso all'inizio del loop.
Abbiamo l'uscita dei numeri su console per aiutare chi gestisce la tombola a non perdersi, dato che è possibile scrollare a ritroso la finestra della console, e anche perché se c'è qualche gioco meccanico nel brandeggio o la posizione del tabellone è imprecisa, ci potrebbero essere dei dubbi sul numero segnalato sul tabellone rispetto all'estratto reale.
Una ultima nota a questa sezione riguarda il fatto che ci sono vari aggiustamenti sui valori, fatti aggiungendo 1, per gestire correttamente sia la posizione dei numeri sul tabellone, sia l'indicizzazione del vettore che parte da zero e non da 1.

```
void loop()
{
  for (int estratto=0;estratto <= 90; estratto++) { // iniziamo a
                                                     // estrarre i numeri
    while (digitalRead(Tasto)!=0)  // facciamo numeri a caso per
                                   // intrattenere il pubblico ;-)
    {
      digitalWrite(Laser,HIGH);
      myservoX.write(Xmin-random(18));  // andiamo a spasso per il tabellone
      myservoY.write(Ymin-random(20));
      Ripeti=10;
      Flash();
    }                          // si continua fino alla pressione del tasto

    Numero = Sequenza[estratto]-1;//la tombola inizia da 1 e noi abbiamo le
                                  //caselle da 0 a 99 come coordinate, quindi
                                  //dobbiamo sottrarre 1 all'estratto per
                                  //trovare la sua posizione
    Xpos= int (Numero/10);    // separiamo decine e unità
    Ypos= Numero-Xpos*10;

    Audio ();  // facciamo capire che è stato estratto un numero
    Serial.print ("estrazione numero: ");
    Serial.print (estratto+1);
    Serial.print (" = ");
    Serial.println (Sequenza[estratto]);
    Audio ();  // ancora un po' di rumore
```

```
    while (digitalRead(Tasto)!=0) //visualizziamo numero estratto
    {
      digitalWrite(Laser,HIGH);
      for(int ofst=3;ofst>-1;ofst--)
      {
        myservoX.write(Xmin-Xpos*2+ofst);//scuotiamo il motore per centrare
        myservoY.write(Ymin-Ypos*2+ofst);
        delay(100);
        myservoX.write(Xmin-Xpos*2-ofst);
        myservoY.write(Ymin-Ypos*2-ofst);
        delay(100);
      }
      Ripeti=30;
      Flash();
    }         //si continua fino alla pressione del tasto
    delay (500); //pausa per consentire di non sovrapporre le pressioni dei tasti
  }
  delay (500);   //pausa per consentire di non sovrapporre le pressioni dei tasti
}
```

Le subroutine

Il codice di questo Sketch contiene quattro subroutine chiamate sia dal *setup()* sia dal *loop()* e predisposte per svolgere funzioni ripetute più volte, oppure da svolgere una sola volta, ma di una certa lunghezza e quindi ingombranti dal punto di vista del codice. Nello specifico trovate:
Flash()
Mischia()
Area()
Audio()

La prima routine – *Flash()* - banalmente accende e spegne il puntatore collegato al pin definito nelle dichiarazioni iniziali (nel nostro caso è D3). La variabile globale "Ripeti" definisce quante volte il ciclo della routine viene ripetuto, così da poterne gestire la durata complessiva prima della chiamata.

```
void Flash()      // qui facciamo lampeggiare il Laser per indicare
{
  for (int i=0;i<Ripeti; i++) {
    digitalWrite (Laser,HIGH);
    delay(30);
    digitalWrite (Laser,LOW);
    delay (30);
    }
}
```

La routine *Mischia()* utilizza un metodo che permette di prendere un vettore lungo a piacere e di mischiarlo con un algoritmo che in sequenza prende ciascuna posizione e la scambia con un'altra posizione definita da un numero casuale. In virtù dei meccanismi di generazione dei numeri casuali, il seme diventa molto importante perché a parità di seme passato alla funzione *randomSeed(seme)* la sequenza si ripete e, conseguentemente, anche il modo con cui il vettore viene mescolato. In questa routine abbiamo anche inserito un piccolo contributo audio di durata diversa ad ogni passaggio per dare la sensazione che sta succedendo qualcosa anche in questa fase. Se volete utilizzare questa subroutine per altre applicazioni, ricordate che quando la chiamate passerete due variabili, ovvero il **vettore** con gli elementi da mischiare e la variabile con il **numero di elementi** contenuti nel vettore.

```
void Mischia(int *list, int elem)  // qui mischiamo i numeri della tombola
{
  for (int a=elem-1; a>0; a--)
  {
    int r = random(a+1);
    if (r != a)
    {
      int temp = list[a];
      list[a] = list[r];
      list[r] = temp;
      digitalWrite(AudioPin,HIGH);
      delay(random(50)+10);
      digitalWrite(AudioPin,LOW);
      delay(30);
    }
  }
}
```

Area() contiene alcune parti di codice relative al posizionamento del tabellone, iniziando dal suo centro che – trattandosi di un rettangolo da nove unità sull'asse X e dieci unità sull'asse Y – ha come coordinate 91 e 90. Anche qui facciamo spostare i motori con un'oscillazione da +/- 3 unità per avere un assestamento con i giochi meccanici ridotti al minimo.
Quando premeremo un tasto, usciremo da questa parte e passeremo alla successiva, che punta il fascio sulla casella corrispondente al numero **1** corrispondente a (100,100) e attende 3 secondi, quindi si sposta alla casella **81**, poi alla **90** e quindi alla **10** per ricominciare poi il giro da **1**. Il ritardo nei passaggi sui tre angoli successivi al primo è di soli 300 millisecondi e questo per dare un effetto "disegnato" al rettangolo. Allontanate e avvicinate il tabellone (o il sistema con Arduino) per ottenere la giusta copertura del tabellone. Premete il tasto quando il puntatore si sta spostando verso l'angolo

in alto a destra e uscirete da questa subroutine.

```
void Area ()     // inquadriamo il tabellone
{
  while (digitalRead(Tasto)!=0)  // siamo in taratura tabellone, premere il tasto per uscire
  {
    digitalWrite(Laser,HIGH);
    for(int ofst=3;ofst>-1;ofst--)
    {
      myservoX.write(Xmin-(Lato/2)+1+ofst);       // scuotiamo il motore per centrare
      myservoY.write(Ymin-(Lato/2)+ofst);
      delay(100);
      myservoX.write(Xmin-(Lato/2)+1-ofst);
      myservoY.write(Ymin-(Lato/2)-ofst);
      delay(100);
    }
    Ripeti=30;
    Flash();
    digitalWrite(Laser,HIGH);
    delay (1000);
  }                                  // si continua fino alla pressione del tasto
  digitalWrite(Laser,LOW);
  Audio();
  while (digitalRead(Tasto)!=0)  // siamo in taratura tabellone, premere il tasto per uscire
  {
    digitalWrite(Laser,HIGH);
    myservoX.write(Xmin);              // 1' angolo
    myservoY.write(Ymin);
    delay(3000);
    myservoX.write(Xmin-Lato+2);       // 2' angolo
    myservoY.write(Ymin);
    delay(300);
    myservoX.write(Xmin-Lato+2);       // 3' angolo
    myservoY.write(Ymin-Lato);
    delay(300);
    myservoX.write(Xmin);              // 4' angolo
    myservoY.write(Ymin-Lato);
    delay(300);
  }                                  // si continua fino alla pressione del tasto
  digitalWrite(Laser,LOW);
}
```

Audio(), infine, non fa che accendere per 300 millisecondi il pin a cui abbiamo collegato il nostro **gadget sonoro** - che potete anche sostituire con un buzzer a 5V -, spegnendolo successivamente. Il ritardo di 30 millisecondi

dopo lo spegnimento serve a dare uno stacco se la subroutine viene ripetutamente chiamata.

```
void Audio ()
{
    digitalWrite(AudioPin,HIGH);
    delay(300);
    digitalWrite(AudioPin,LOW);
    delay(30);
}
```

Sketch

```
// Estrattore di numeri della Tombola da 1 a 90
// utilizza la libreria SERVO
// Realizzato da Simone Majocchi - Marzo 2012
// per il libro di Vispa Edizioni
// Arduino
// Programmazione avanzata e Librerie di sistema

#include <Servo.h>

Servo myservoX;  // Istanza per motore asse x
Servo myservoY;  // Istanza per motore asse y

int Xpos = 0;    // popsizione x
int Ypos = 0;    // posizione y
int Tasto=7;     // il tasto di interazione è su D7
int Laser=3;     // laser o torcia LED sul pin 3
int AudioPin=4;  // gadget audio sul pin 4
int Xmin=100;    // l'angolo in alto a sinistra del tabellone
int Ymin=100;
int Lato=20;     // il lato del nostro tabellone
int Ripeti=30;   // numero di lampeggi del Laser
int Numero;      // qui memorizziamo il numero estratto

int Sequenza[] = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,
                 21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,
                 41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,
                 61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80,
                 81,82,83,84,85,86,87,88,89,90}; // i numeri della tombola
int Elementi = sizeof(Sequenza) / sizeof(int);

void setup()
{
  pinMode (Tasto,INPUT);  // il tastino di controllo
  pinMode (Laser,OUTPUT); // il LASER
  pinMode (AudioPin,OUTPUT); // il generatore AUDIO

  Serial.begin(9600);        // per sapere cosa viene estratto dal monitor seriale
  int Seme = analogRead(0);    // prepariamo un buon seme per la generazioneì
  delay (random(50));          // di una sequenza pseudocasuale con cui
  Seme = Seme + analogRead(0); // mischiare la sequenza; l'ingresso analogico
  randomSeed(Seme);            // a cui colleghiamo un filo è un buon riferimento
                               // per avere valori casuali.

  Mischia (Sequenza, Elementi);// invochiamo la routine di "miscelazione" della sequenza

  myservoX.attach(9,500,2500);  // l'oggetto myservoX è sul pin 9
  myservoY.attach(8,500,2500);  // l'oggetto myservoY è sul pin 8
```

```
  delay(500);
  Area ();  // facciamo la taratura della posizione di brandeggio e tabellone

  digitalWrite(AudioPin, HIGH);
  delay (1000);
  digitalWrite(AudioPin, LOW);

  delay (500);
}

void loop()
{
  for (int estratto=0;estratto <= 90; estratto++) {  // iniziamo a estrarre i  numeri
    while (digitalRead(Tasto)!=0)//facciamo numeri a caso per intrattenere il pubblico ;-)
    {
      digitalWrite(Laser,HIGH);
      myservoX.write(Xmin-random(18));       // andiamo a spasso per il tabellone
      myservoY.write(Ymin-random(20));
      Ripeti=10;
      Flash();
    }                             // si continua fino alla pressione del tasto

    Numero = Sequenza[estratto]-1; //la tombola inizia da 1 e noi abbiamo le caselle da 0 a
                                   //99 come coordinate, quindi dobbiamo sottrarre 1
                                   //all'estratto per trovare la sua posizione
    Xpos= int (Numero/10);  // separiamo decine e unità
    Ypos= Numero-Xpos*10;

    Audio ();  // facciamo capire che è stato estratto un numero

    Serial.print ("estrazione numero: ");
    Serial.print (estratto+1);
    Serial.print (" = ");
    Serial.println (Sequenza[estratto]);
    Audio ();  // ancora un po' di rumonre

    while (digitalRead(Tasto)!=0)  // visualizziamo numero estratto
    {
      digitalWrite(Laser,HIGH);
      for(int ofst=3;ofst>-1;ofst--)
      {
        myservoX.write(Xmin-Xpos*2+ofst);   // scuotiamo il motore per centrare
        myservoY.write(Ymin-Ypos*2+ofst);
        delay(100);
        myservoX.write(Xmin-Xpos*2-ofst);
        myservoY.write(Ymin-Ypos*2-ofst);
        delay(100);
      }
      Ripeti=30;
```

```
      Flash();
    }              // si continua fino alla pressione del tasto
    delay (500);   // pausa per consentire di non sovrapporre le pressioni dei tasti
  }
  delay (500);     // pausa per consentire di non sovrapporre le pressioni dei tasti
}

void Flash()      // qui facciamo lampeggiare il Laser per indicare
{
  for (int i=0;i<Ripeti; i++) {
  digitalWrite (Laser,HIGH);
  delay(30);
  digitalWrite (Laser,LOW);
  delay (30);
  }
}


void Mischia(int *list, int elem)  // qui mischiamo i numeri della tombola
{
  for (int a=elem-1; a>0; a--)
  {
    int r = random(a+1);
    if (r != a)
    {
      int temp = list[a];
      list[a] = list[r];
      list[r] = temp;
      digitalWrite(AudioPin,HIGH);
      delay(random(50)+10);
      digitalWrite(AudioPin,LOW);
      delay(30);
    }
  }
}

void Area ()     // inquadriamo il tabellone
{
  while (digitalRead(Tasto)!=0){ // siamo in taratura tabellone, premere il tasto per uscire
    digitalWrite(Laser,HIGH);
    for(int ofst=3;ofst>-1;ofst--)
    {
      myservoX.write(Xmin-(Lato/2)+1+ofst);   // scuotiamo il motore per centrare
      myservoY.write(Ymin-(Lato/2)+ofst);
      delay(100);
      myservoX.write(Xmin-(Lato/2)+1-ofst);   //
      myservoY.write(Ymin-(Lato/2)-ofst);
      delay(100);
    }
```

```
    Ripeti=30;
    Flash();
    digitalWrite(Laser,HIGH);
    delay (1000);
  }                                   // si continua fino alla pressione del tasto
  digitalWrite(Laser,LOW);
  Audio();
  while (digitalRead(Tasto)!=0)  // siamo in taratura tabellone, premere il tasto per uscire
  {
    digitalWrite(Laser,HIGH);
    myservoX.write(Xmin);             // 1' angolo
    myservoY.write(Ymin);
    delay(3000);
    myservoX.write(Xmin-Lato+2);      // 2' angolo
    myservoY.write(Ymin);
    delay(300);
    myservoX.write(Xmin-Lato+2);      // 3' angolo
    myservoY.write(Ymin-Lato);
    delay(300);
    myservoX.write(Xmin);             // 4' angolo
    myservoY.write(Ymin-Lato);
    delay(300);
  }                                   // si continua fino alla pressione del tasto
  digitalWrite(Laser,LOW);
}

void Audio ()
{
  digitalWrite(AudioPin,HIGH);
  delay(300);
  digitalWrite(AudioPin,LOW);
  delay(30);
}
```