Arduino Tutorial 2: Galleria progetti

Per Arduino Software (IDE) / Arduino Web Editor

Release 0.8
31/03/2017

Dopo aver visto le basi di Arduino con il primo tutorial, scopriamo una carrellata di progetti molto vari: si concentreranno di volta in volta su un singolo aspetto o componente, e ci permetteranno di capire le potenzialità di Arduino, la nostra scheda elettronica programmabile.

Trovi questo tutorial, tutti gli sketch di Arduino, gli schemi elettrici ed altre informazioni sul nostro sito:

coderdojotrento.it/arduino2

1. Servomotore controllato da un potenziometro

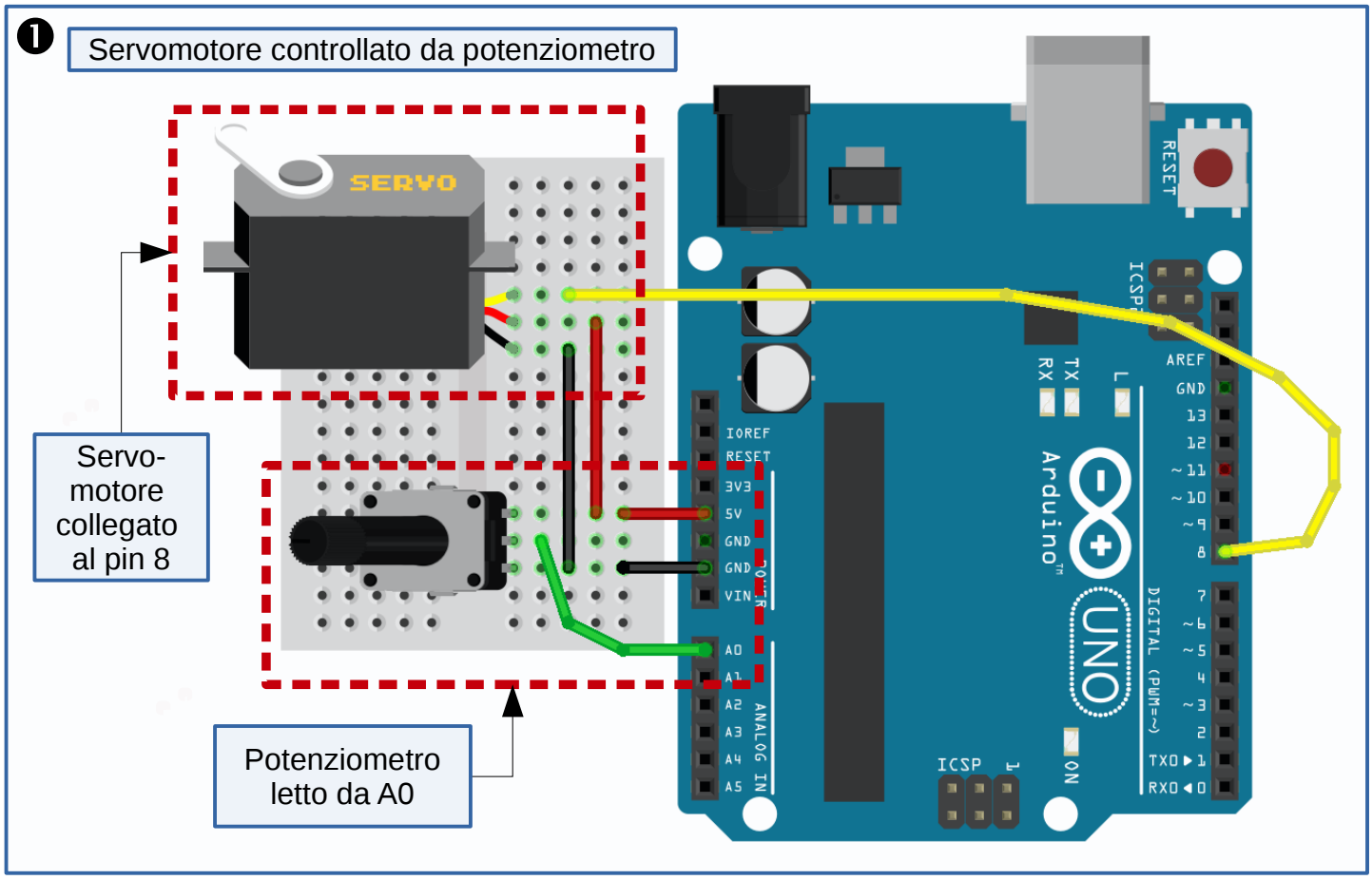
Questo progetto è la rivisitazione di uno degli esempi che si trovano installati assieme alla GUI di Arduino: possiamo caricare il file che troviamo sotto il menù File → Esempi → Servo → Knob (ovvero “Manopola”) e modificarlo per ottenere lo script in figura ❶.

Notiamo l’inclusione della libreria **Servo.h** (riga 1) per controllare il servomotore (inizializzato a riga 10 e mosso a riga 22), l’utilizzo della classe **Serial** per mandare al PC informazioni di debug (inizializzata a riga 11 ed utilizzata alle righe 16, 17, 21 e 22) ed infine l’utilizzo della funzione **map** che permette di trasformare i valori 0...1023 in angoli 0...180 (riga 19).

progetto_6_Servomotore_e_potenziometro ❶

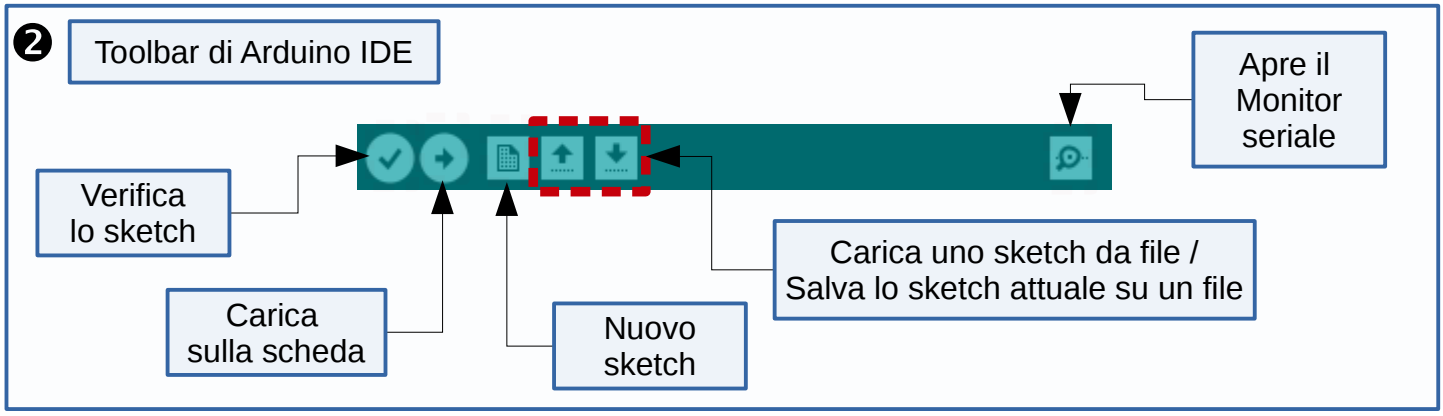
```
1  #include <Servo.h>       // include la libreria che ci permette di pilotare i servomotori
2
3  Servo motore;            // crea una variabile di tipo "Servo" che useremo per controllare il motore
4
5  const int pinPot = A0;   // il numero del pin analogico al quale colleghiamo il potenziometro
6  int valPot;              // variabile per il valore letto dal potenziometro (numeri compresi tra 0 e 1023)
7  int angolo;              // variabile per l'angolo che invieremo al servomotore (sono numeri tra 0 e 180)
8
9  void setup() {
10   motore.attach(8);      // decidiamo di controllare il servomotore dal pin digitale numero 8
11   Serial.begin(9600);    // apre la comunicazione seriale con il computer (alla velocità di 9600 bps)
12 }
13
14 void loop() {
15   valPot = analogRead(pinPot);  // legge il valore dal potenziometro e lo assegna alla variabile valPot
16   Serial.print("valPot: ");     // scrive verso il computer la stringa riportata tra apici
17   Serial.print(valPot);         // scrive verso il computer il valore della variabile valPot
18
19   angolo = map(valPot, 0, 1023, 0, 180);  // trasforma valPot per ottenere un angolo tra 0 e 180 gradi
20
21   Serial.print(", angolo: ");   // scrive verso il computer la stringa riportata tra apici
22   Serial.println(angolo);       // scrive verso il computer il valore della variabile angolo e va a capo
23
24   motore.write(angolo);         // posizione il braccio del servomotore all'angolo indicato
25
26   delay(20);                    // aspetta un cinquantesimo di secondo (20 millisecondi)
27 }
```

In figura ❶ notiamo il **potenziometro** con i due piedini esterni collegati ai 5V ed al GND di Arduino (grazie ai fili rosso e nero); il segnale è prelevato dal cavetto verde e portato sul pin analog input A0. I due spezzoni verticali di cavo rosso e nero portano l'alimentazione al **servomotore**; questo è controllato dal pin ditigale 8 grazie al cavetto giallo.



N.B. Il colore dei cavi non è importante per il funzionamento del circuito in sé, ma aiuta a seguire il percorso quando questo comincia a diventare complesso: per ricordarci meglio possiamo utilizzare cavetti neri o marroni per i collegamenti a terra, mentre possiamo usare colori accesi (rosso o arancione) per collegare i punti che sono a potenziale più alto. Per i cavetti che trasportano dei segnali di controllo, utilizziamo colori ben distinguibili come giallo, azzurro o verde.

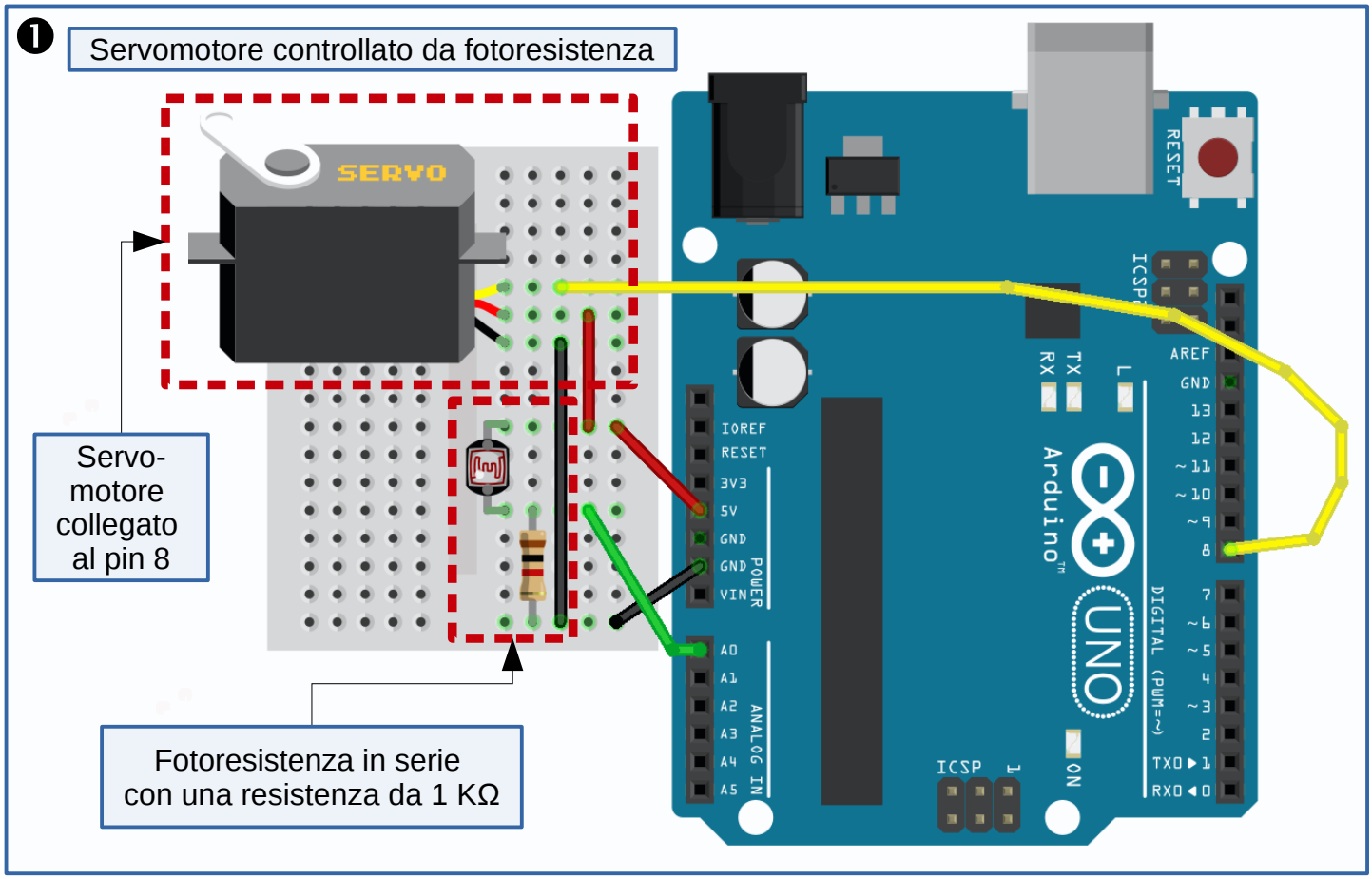
Per la **verifica dello sketch ed il suo caricamento sulla scheda**, utilizziamo i due primi tasti della toolbar dell'IDE di Arduino (vedi figura ❷); per controllare il corretto funzionamento del programma, apriamo il **Monitor seriale** (icona più a destra della toolbar) e vediamo cosa succede se giriamo la manopola del potenziometro.



2. Servomotore controllato da una fotoresistenza

Con lo stesso sketch, possiamo controllare il circuito di figura ❶: al posto del potenziometro, mettiamo due resistenze in serie, una fissa da 1 KΩ ed una **fotoresistenza** (una resistenza che cambia valore in base alla quantità di luce che la colpisce). Le due resistenze si "spartiscono" la tensione di 5 Volt che le alimenta, e sul nodo intermedio (al quale abbiamo collegato il cavetto verde) leggeremo la tensione grazie all'analog input A0.

❶ Servomotore controllato da fotoresistenza



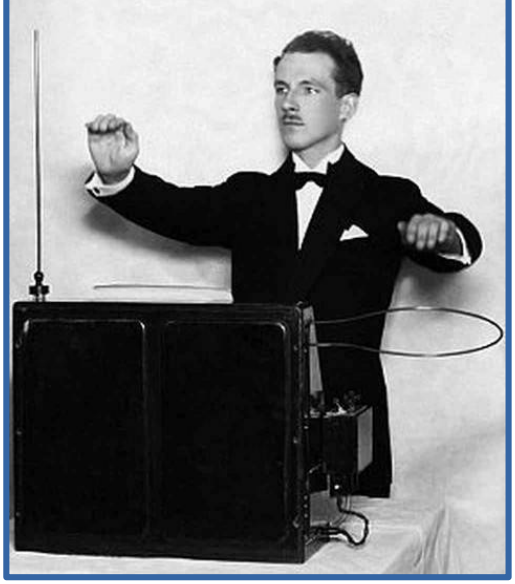
Lo sketch è già caricato su Arduino, quindi appena alimentiamo la scheda col cavetto USB vediamo il servomotore muoversi in base a quanto illuminata è la fotoresistenza: passaci sopra una mano, per far spostare il braccio del servomotore!

3. Theremin a luce

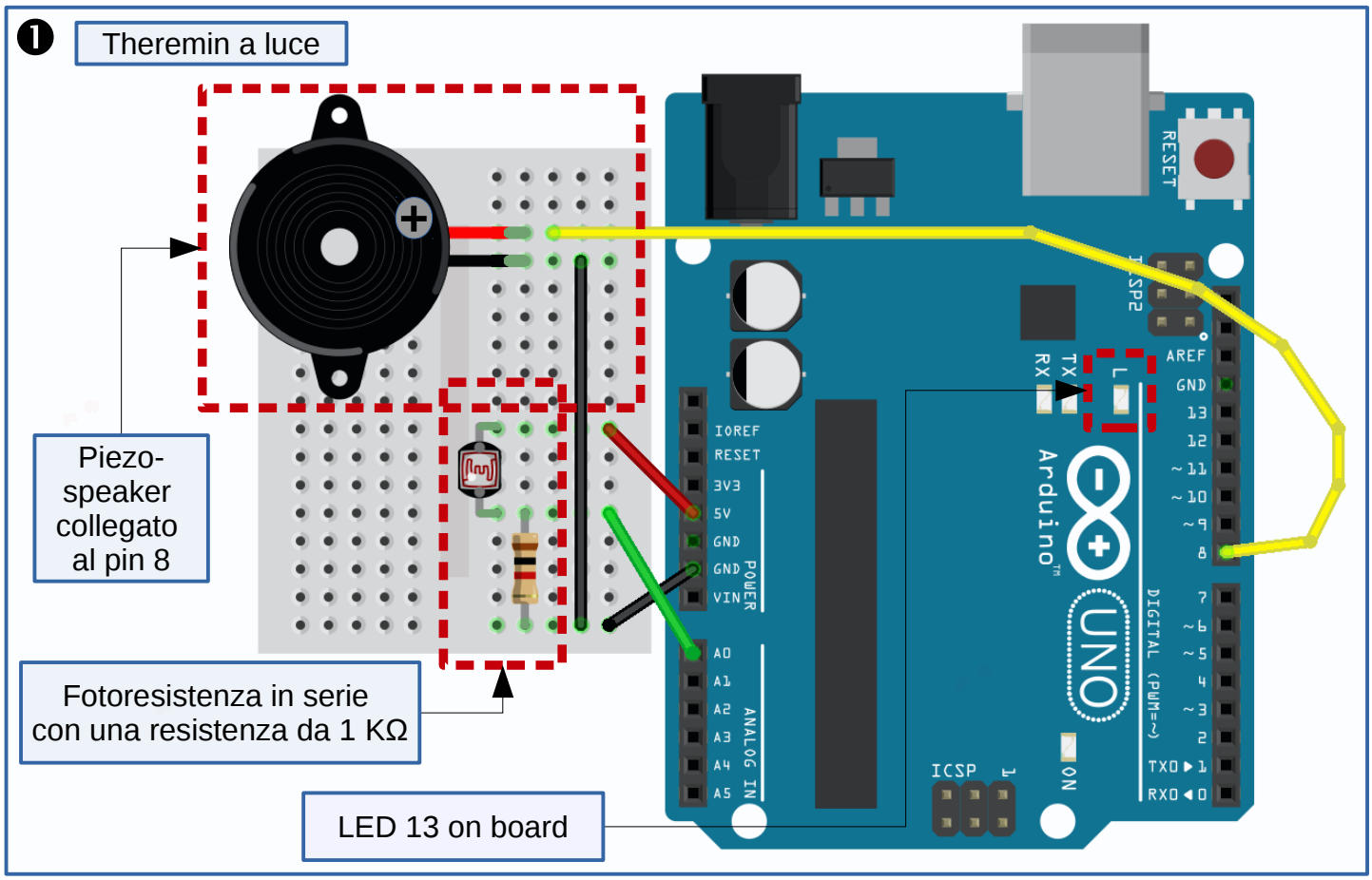
Ma cos'è il ***theremin***? É un particolarissimo strumento musicale che si suona senza toccarlo: basta muovere le mani davanti a due antenne (una regola il volume del suono, l'altra la nota emessa).
Nell'immagine ❷ puoi vedere il suo inventore -Leon Theremin, appunto- all'opera: lo strumento è stato inventato all'inizio del secolo scorso, e noi ne costruiremo una versione semplificata che funziona grazie alla luce.

Costruire il circuito sarà molto semplice: già conosciamo il sensore per la luce (la **fotoresistenza**), ed infatti la parte del nostro circuito che rileva l'intensità luminosa sarà la stessa del circuito descritto nella figura ❶ di questa pagina.
Dalla basetta di prima toglieremo il servomotore e -per la produzione del suono- utilizzeremo poi uno **speaker piezoelettrico** (detto anche *piezo-buzzer*): un aggettivo complicato per descrivere un componente che trasforma l'elettricità in vibrazione, e quindi suono.

❷ Theremin col suo theremin



In figura ❶ abbiamo il circuito per il theremin a luce: nella parte inferiore riconosciamo la serie della fotoresistenza e della resistenza da 1 KΩ, mentre in alto abbiamo aggiunto il piezo-buzzer collegato al pin 8. Le uniche attenzioni che dobbiamo avere riguardano il tipo di buzzer (ne scegliamo uno **passivo**: leggi la scheda in fondo a questa pagina per conoscere i diversi tipi di buzzer) e la polarità dei contatti (sulla capsula di plastica il pin positivo è segnato con un +, e di norma ha la gambetta più lunga).



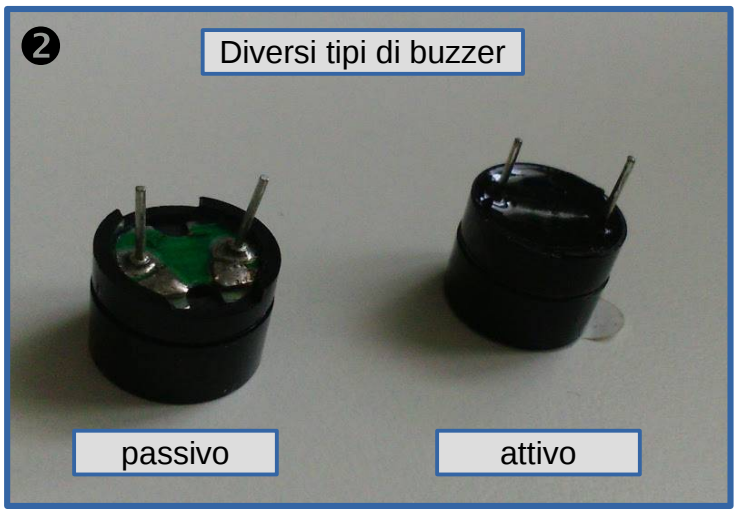
Lo sketch per far funzionare il nostro theremin è riportato nella pagina seguente.

BUZZER ATTIVI E PASSIVI

Nei kit di componenti per Arduino troviamo quasi sempre due tipi di buzzer: il tipo attivo e quello passivo (in figura ❷ è indicato il modo per distinguerli: quelli attivi hanno di solito la base della capsula completamente sigillata).

Quelli **attivi** sono in grado di suonare per conto loro: è sufficiente alimentarli con una tensione di alcuni Volt (i 5 Volt forniti da Arduino vanno benone) ed emetteranno il loro ronzio.

Quelli **passivi** invece non sono in grado di vibrare in autonomia: saremo noi ad inviare un segnale variabile che farà oscillare la parte meccanica che emette il suono.
La funzione **tone()** di Arduino fa proprio questo: manda su un dato pin di output un segnale oscillante (un'onda quadra che varia tra 0 e 5 Volt) di una particolare frequenza (dalla quale dipende la tonalità del suono emesso).



Lo sketch per il theremin a luce (vedi figura ❶) contiene molti elementi interessanti: leggiamolo assieme e poi analizziamo gli aspetti rilevanti.

progetto_7_Theremin_a_luce.ino ❶

```
1  const int pinFoto = A0; // il numero del pin analogico al quale colleghiamo la fotoresistenza
2  const int pinLED = 13;  // il numero del pin digitale al quale colleghiamo il LED
3  const int pinPiezo = 8; // il numero del pin digitale al quale colleghiamo il piezo-speaker
4  int valFoto;            // variabile per il valore letto dalla fotoresistenza (sta tra 0 e 1023)
5  int valFotoMin = 1023;  // variabile per il valore minimo letto durante la fase di calibrazione
6  int valFotoMax = 0;     // variabile per il valore massimo letto durante la fase di calibrazione
7  int nota;               // variabile per l'acutezza della nota che andremo a suonare
8
9  void setup() {
10   pinMode(pinLED, OUTPUT);      // imposta il pin del LED come output
11   digitalWrite(pinLED, HIGH);   // e lo accende per indicare che è iniziata la calibrazione
12
13   while (millis() < 5000) {         // durante i primi 5 secondi dall'avvio dello sketch
14     valFoto = analogRead(pinFoto);  // legge il valore della luminosità
15
16     // controlla se questo valore è il nuovo massimo; eventualmente lo prende come nuovo massimo
17     if (valFoto > valFotoMax) { valFotoMax = valFoto; }
18
19     // controlla se questo valore è il nuovo minimo; eventualmente lo prende come nuovo minimo
20     if (valFoto < valFotoMin) { valFotoMin = valFoto; }
21   }
22
23   digitalWrite(pinLED, LOW);    // spegne il LED per indicare che è finita la calibrazione
24 }
25
26 void loop() {
27   valFoto = analogRead(pinFoto);    // legge il valore della luminosità
28
29   // trasforma valFoto (che sappiamo variare tra valFotoMin e valFotoMax) in un numero
30   // compreso tra 50 e 4000: l'acutezza delle note riprodotte sarà in questo intervallo
31   nota = map(valFoto, valFotoMin, valFotoMax, 50, 4000);
32
33   tone(pinPiezo, nota, 20); // per 20 millisecondi suona la nota sul pin del piezo-speaker
34
35   delay(10);                // attende un poco
36 }
```

a) La funzione setup() è molto grande (righe da 9 a 24): invece di limitarci a settare i pin di input ed output, stavolta effettuiamo tutta la parte di calibrazione degli input dalla fotoresistenza. Per i primi 5 secondi di funzionamento del programma, muoviamo la mano su e giù davanti alla fotoresistenza per far capire ad Arduino quali sono i valori massimi e minimi della luminosità: questi valori ci serviranno poi durante il funzionamento del programma (riga 31) per trasformare la luminosità rilevata in una nota da suonare sul buzzer.

b) Taratura: per trovare i valori minimo e massimo, andiamo a leggere il valore sul pin della fotoresistenza (riga 14) e lo confrontiamo con il valore min e max trovati fino a quel momento (righe 17 e 20). I confronti a per trovare i nuovi min e max andranno a buon fine solo se parto con un min grande e con un max piccolo (guarda come sono stati inizializzati i valori a riga 5 e 6).

c) La funzione millis() (riga 13) ci dice quanti sono i millisecondi passati da quando carichiamo lo sketch (oppure premiamo il bottone di reset); il ciclo while che comincia a riga 13 viene eseguito quindi solo per i primi 5 secondi).

d) Ma il LED del pin 13 c'è o non c'è? Nello sketch utilizziamo un LED collegato al pin 13 e lo teniamo acceso per tutta la fase di taratura, ma come mai il LED non compare nello schema ❶ della pagina precedente? Non serve collegare un LED esterno, perché Arduino ha già un LED "on board"

(cioè saldato direttamente sulla scheda) che è collegato proprio al pin 13 (nella figura ❶ di pagina 4 è evidenziato questo piccolo LED marcato con una "L"). Possiamo quindi fare i nostri primi esperimenti di accensione/spegnimento di un LED, anche non avendone alcuno!

e) La nota da suonare: il valore della frequenza della nota (passato alla funzione **tone()** a riga 33) è calcolato grazie alla funzione **map()** di riga 31; questi valori cambiano con continuità tra 50 e 4000 quando la luminosità passa dal valore minimo al massimo: il suono prodotto è quindi il tipico sibilo del theremin che "scivola su e giù" su tutta la scala, senza fermarsi sulle note DO, RE, MI, FA, etc. Per far suonare ad Arduino delle note specifiche, vediamo il prossimo progetto!

4. Arduino conosce anche le note!

Per questo progetto non serve cambiare nulla nel circuito; anzi, possiamo anche togliere tutta la parte di input (resistenza e fotoresistenza) e tenere solo il piezo-buzzer collegato al pin numero 8. Nessun nuovo componente elettronico, quindi, ma impareremo ad utilizzare i vettori e l'inclusione di nostri files nello sketch. Guardiamo il nuovo sketch riportato in figura ❶.

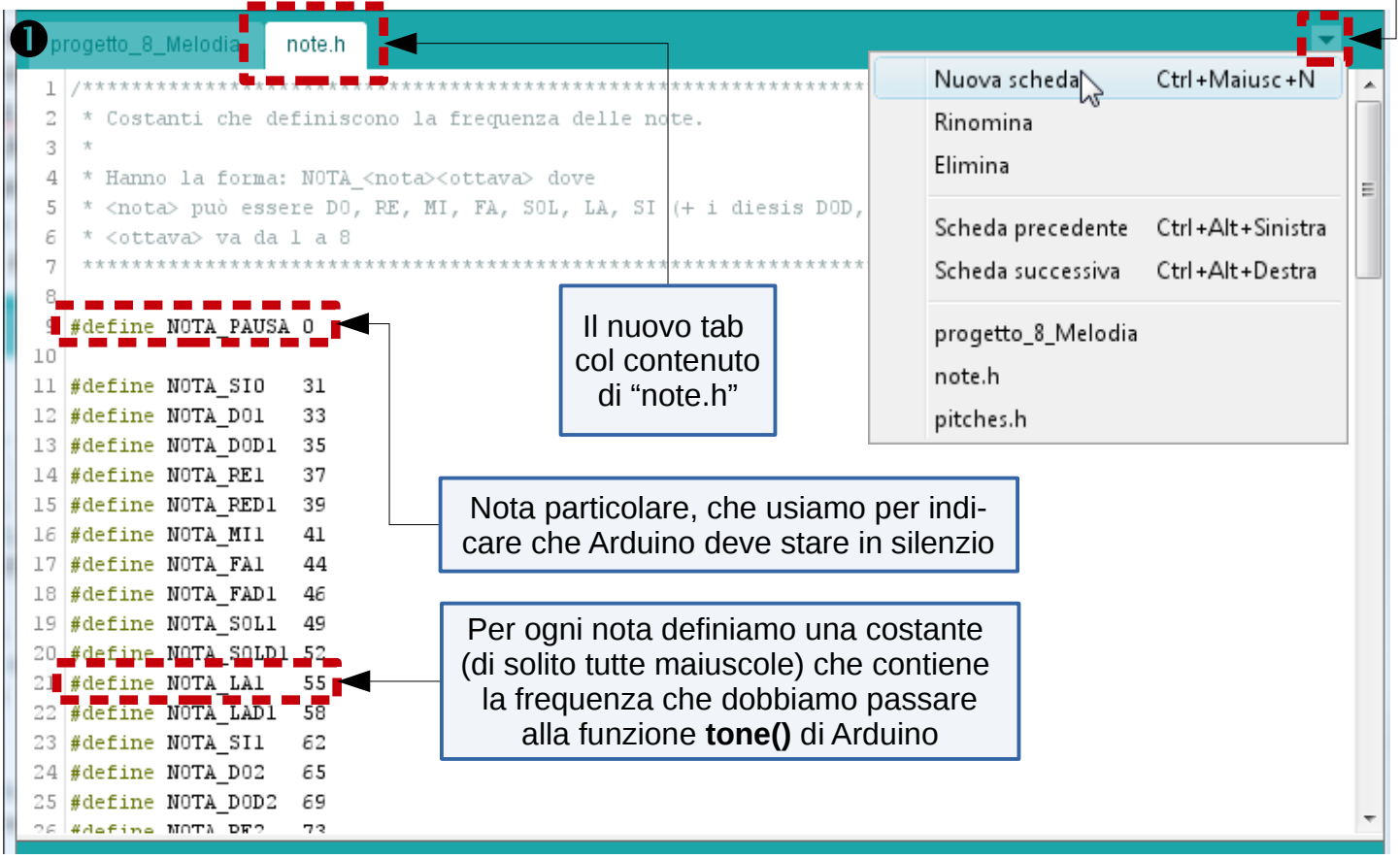
progetto_8_Melodia ❶

```
1  #include "note.h"    // trascrive all'interno dello sketch il contenuto del file note.h
2
3  const int pinPiezo = 8;   // pin digitale al quale colleghiamo il piezo-speaker
4
5  // vettore di interi che contiene la lista delle note: queste sono delle costanti definite all'interno
6  // del file note.h che abbiamo incluso qui sopra
7  int melodia[] = { NOTA_DO5, NOTA_SOL4, NOTA_SOL4, NOTA_LA4, NOTA_SOL4, NOTA_PAUSA, NOTA_SI4, NOTA_DO5 };
8
9  // vettore di interi che contiene la lista dei tipi di note (per poter calcolare la loro durata), che
10 // deve essere lungo come il vettore delle note (i valori sono: 4 = nota da un quarto, 8 = un ottavo, ...):
11 int tipoNote[] = { 4, 8, 8, 4, 4, 4, 4, 4 };
12
13 void setup() {
14   // ciclo che scandisce le note contenute nel vettore melodia: otto note che conteggiamo da 0 fino a 7
15   for (int numeroNota = 0; numeroNota < 8; numeroNota++) {
16
17     // per calcolare la durata della nota, prendo un secondo e lo divido per il tipo di nota;
18     // ad esempio: la nota da un quarto dura 1000/4 millisecondi, la nota da un ottavo dura 1000/8, etc.
19     int durata = 1000 / tipoNote[numeroNota];
20
21     tone(pinPiezo, melodia[numeroNota], durata); // suono la nota con lo speaker
22
23     // per separare le note, attendiamo un po' di tempo tra una e l'altra: un'attesa pari al 30%
24     // della nota appena suonata può andare bene per distinguere i suoni
25     int pausaTraLeNote = durata * 1.30;
26     delay(pausaTraLeNote);
27
28     // finito il ciclo, possiamo passare alla prossima nota (se ce n'è un'altra da suonare)
29   }
30 }
31
32 void loop() {
33   // non serve ripetere la melodia
34 }
```

Analizziamo le novità:

a) La funzione loop() è vuota (righe 32-34): già nel progetto precedente abbiamo visto uno sketch in cui gran parte del lavoro veniva svolto dalla funzione setup() e nella loop() erano presenti poche operazioni, ma stavolta nella loop() non facciamo davvero niente!
Vuol dire che l'esecuzione del nostro programma sta tutta nel setup() e che non c'è alcuna operazione da ripetere all'infinito: quando caricheremo lo sketch su Arduino, questo suonerà una sola volta la melodia; se vogliamo risentirla, sarà sufficiente premere il bottone di reset sulla schedina (il piccolo tastino rosso vicino al connettore della porta USB).

b) La direttiva #include (riga 1): l'abbiamo già usata per includere (cioè per copiare all'interno del nostro sketch) il contenuto di file di libreria come Servo.h: nei file .h di norma è presente tutto il necessario per "far funzionare" una certa cosa, come appunto -ad esempio- un servomotore. L'idea è di mettere tutto in un file separato, così se nel nostro progetto abbiamo esigenza di usare i servomotori, andremo ad includere quel file; se invece nel nostro progetto ci occupiamo di altro, sarà inutile ricordare ad Arduino come fare per controllare i motori… In sostanza queste **librerie esterne** sono dei "*pacchetti di funzionalità*" che andremo a caricare solo all'occorrenza.
Il file Servo.h è esterno al nostro progetto, e per utilizzarlo, dobbiamo impiegare la direttiva #include con il file messo tra parentesi angolari <Servo.h> (così Arduino sa che deve andare a cercare il file Servo.h tra le sue librerie); nella riga 1 dello sketch della pagina precedente, abbiamo però usato le doppie virgolette "note.h": vuol dire che Arduino deve cercare il file note.h dentro il nostro progetto!
La verità infatti è che Arduino non conosce le note, e siamo noi a dovergliele spiegare: è per questo che creiamo un nuovo file all'interno del nostro progetto, utilizzando la voce del menù "Nuova scheda" che troviamo cliccando sull'icona in alto a destra (vedi figura ❶).
Una volta scelto il nome del file, si apre una nuova scheda: qui trascriveremo il contenuto del file che troviamo sul sito di CoderDojo alla pagina del tutorial.



c) L'uso dei vettori: quando dobbiamo memorizzare una serie di valori omogenei tra loro, non ci conviene usare una "sfilza" di variabili, ma un unico oggetto che contiene diverse cellette numerate nelle quali possiamo salvare i nostri valori. **Queste strutture si chiamano vettori**, possono essere del tipo che preferiamo (ad esempio a riga 7 e 11 definiamo due vettori che contengono degli interi) e ci permettono di raggiungere i diversi valori della lista utilizzando un indice racchiuso tra parentesi quadre: ad esempio la prima nota della nostra musichetta sarà melodia[0] (gli indici partono infatti da zero), la seconda nota sarà melodia[1] e così via; in un vettore lungo 8 come la nostra melodia[ ], l'ultimo elemento sarà quindi melodia[7].

A riga 7, oltre a dichiarare che melodia[ ] è un vettore di interi, gli assegniamo anche il contenuto: dopo l'uguale, infatti, troviamo l'elenco degli otto valori che vogliamo inserire nella nostra lista, separati da virgola e racchiusi tra parentesi graffe.

A riga 11 facciamo la stessa cosa per il vettore di interi che si chiama tipoNote[ ]: qui memorizziamo, posizione per posizione, la durata delle note contenute nel vettore melodia[ ].

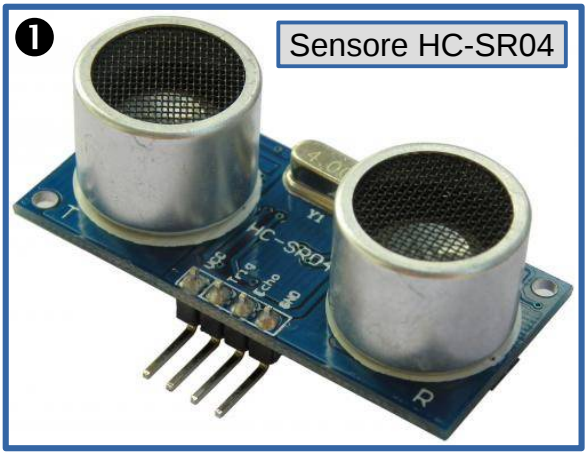
Il modo naturale per usare dei vettori è “scandirne” il contenuto all’interno di un ciclo for; la variabile numeroNota che usiamo nel ciclo for che comincia a riga 15 e finisce a riga 29 serve esattamente a questo: assume via via tutti i valori da 0 a 7, quindi ci permette di raggiungere tutte le note (così: melodia[numeroNota]) e tutte le durate (così: tipoNote[numeroNota]).

Salva lo sketch, caricalo su Arduino e riconosci la musichetta! Se vuoi risentirla, puoi premere il tasto di reset. Se conosci una melodia, memorizza le tue note e la loro lunghezza cambiando i vettori melodia[ ] e tipoNote[ ] e ricordati di cambiare anche la condizione che determina l’uscita dal ciclo for (nel nostro caso di musichetta con 8 note, noi eseguiamo il ciclo finché “numeroNota < 8”).

5. Sensore di distanza ad ultrasuoni

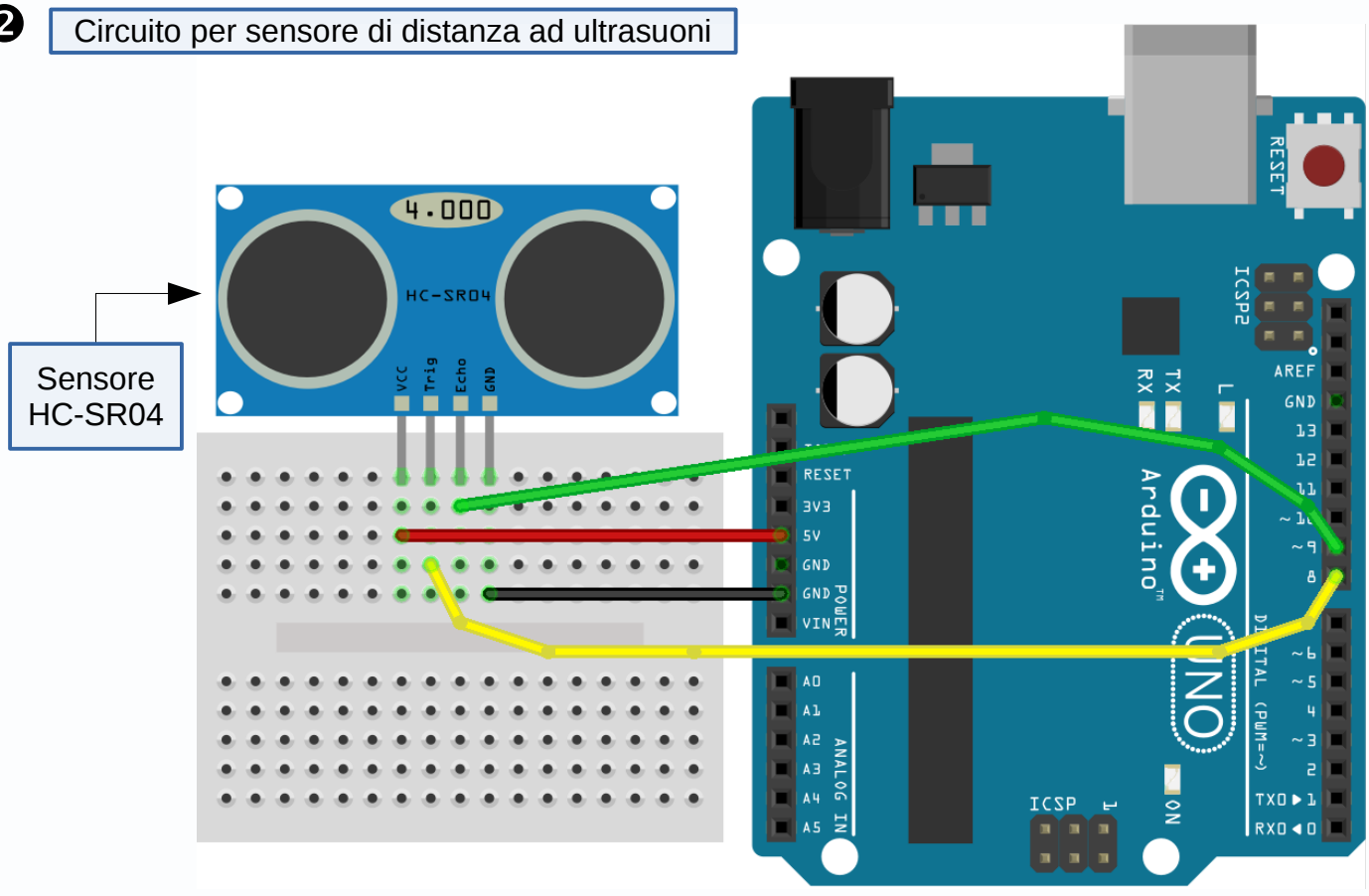
Un componente elettronico molto particolare che troviamo spesso nei kit di Arduino è il **sensore di distanza ad ultrasuoni**: dalla forma (vedi figura ❶) sembrerebbero i due occhi di un robot, in realtà sono piuttosto la bocca ed un orecchio… Infatti uno dei due cilindri emette un impulso ad ultrasuoni ed il secondo rileva il suo eco, dopo che il segnale è rimbalzato contro un ostacolo: è esattamente il modo in cui i **pipistrelli** riescono a misurare le distanza dagli ostacoli anche al buio.

❶ Sensore HC-SR04



Nella figura ❷ è riportato il semplice circuito che dobbiamo preparare per usarlo: dei suoi quattro piedini, i due esterni servono per l’alimentazione (*+5Volt* e *GND*), e gli altri due (*Trigger* -ovvero “grilletto”- ed *Echo*) vanno collegati a due pin digitali.

❷ Circuito per sensore di distanza ad ultrasuoni



Il componente non misura in realtà una distanza, ma il tempo che impiega il segnale sonoro per compiere il tragitto avanti ed indietro dopo essere rimbalzato contro l’ostacolo: per calcolare la distanza dobbiamo quindi solo ricordarci la formula che la lega alla velocità ed al tempo. Nello sketch riportato nella pagina seguente sono descritti tutti i passaggi necessari.

Anche questo sketch (figura ❶) contiene alcune cose che non conoscevamo ancora: guardiamo il listato del programma e poi analizziamo le novità.

progetto_9_Sensore_a_ultrasuoni ❶

```
1  const int pinImpulso = 8;  // pin sul quale scrive l'impulso
2  const int pinEco = 9;      // pin dal quale legge il tempo dopo il quale arriva l'eco
3  const int pinLED = 13;     // pin al quale abbiamo collegato il LED (quello "on board")
4
5  void setup() {
6    pinMode(pinImpulso, OUTPUT);  // setta il pin come canale di output
7    pinMode(pinEco, INPUT);       // setta il pin come canale di input
8    pinMode(pinLED, OUTPUT);      // setta il pin come canale di output
9    Serial.begin(9600);           // inizializza la porta seriale a 9600 baud
10 }
11
12 void loop() {
13   // dopo aver preparato una base pulita, invia un impulso di 10 microsecondi
14   digitalWrite( pinImpulso, LOW );   //
15   delay(60);                         //  1                  +---------+
16   digitalWrite( pinImpulso, HIGH );  //                     |         |
17   delayMicroseconds( 10 );           //  0  -----------------+         +---------- ...
18   digitalWrite( pinImpulso, LOW );   //     <---  60 ms  --->|<- 10 ns ->
19
20   long tempo = pulseIn( pinEco, HIGH ); // legge il tempo di andata e ritorno dell'eco
21
22   long distanza = calcolaDistanza(tempo);  // chiama la funzione per trovare la distanza
23
24   Serial.print(distanza);   // scrive la misura sulla seriale
25   Serial.println(" cm");
26
27   digitalWrite(pinLED, (distanza < 10) ? HIGH : LOW); // se è vicino, accendiamo il LED
28 }
29
30 // funzione che calcola la distanza dell'ostacolo, a partire dal tempo di volo
31 long calcolaDistanza(long microsecondi) {
32   // La velocità del suono è 340 m/s, quindi per coprire un cm ci vogliono 29.4 microsecondi.
33   // Il ping sonoro percorre andata e ritorno, quindi per trovare la distanza dell'ostacolo
34   // dobbiamo prendere la metà della distanza percorsa dal suono.
35   return microsecondi / 29.4 / 2;
36 }
```

Il valore restituito dal **return** e la funzione stessa devono essere dello stesso tipo

a) Il segnale di *trigger* (righe 13-18): dopo aver atteso 60 ms per far riposare un poco il sensore, Arduino manda l'impulso che attiva la misura del sensore ad ultrasuoni; questo segnale è inviato sul canale di *Trigger* proprio perché fa da "grilletto" per la misura, ovvero ne determina l'avvio;

b) La funzione pulseIn() (riga 20): con la chiamata alla funzione pulseIn(), Arduino si mette in ascolto sulla porta *Echo* ed attende il segnale dal sensore. Dopo aver effettuato il lancio delle onde sonore ed aver fatto il calcolo del tempo di volo (andata e ritorno), il sensore HC-SR04 scrive sul pin collegato a Echo un impulso che dura esattamente come il tempo di volo: la funzione pulseIn() ci restituisce proprio la durata dell'impulso in microsecondi (µs), così nel programma possiamo memorizzarla nella variabile che si chiama "tempo" (N.B. è di tipo ***long*** e non *int*).

c) La funzione calcolaDistanza(): se abbiamo dei calcoli o dei compiti particolari da svolgere e questi magari sono *a se' stanti*, cioè possono essere raggruppati in un modulo a parte, possiamo scrivere anche noi le nostre **funzioni**. In questo caso -alle righe da 30 a 36- abbiamo **definito** la funzione calcolaDistanza() che "vive" per conto suo, al di fuori delle altre funzioni che già conosciamo setup() e loop(); guardando cosa c'è tra le parentesi rotonde di riga 31, vediamo che in **ingresso** prende **come argomento** la variabile "microsecondi" che è di tipo long e -dopo aver fatto

alcuni calcoli- restituisce un altro long (l'istruzione return di riga 35 deve far uscire dalla funzione un valore dello stesso tipo che abbiamo usato nella definizione della funzione: è la prima parola della riga 31).
Una volta definita la nostra funzione, possiamo anche **invocarla** (ovvero usarla effettivamente). A riga 22 c'è la chiamata della funzione: la richiamiamo per nome, passandole come argomento (tra parentesi rotonde) la variabile "tempo". Il risultato dell'esecuzione della funzione è un valore di tipo long che assegniamo alla variabile "distanza".
A proposito, come mai le funziono setup() e loop() non hanno nessuna istruzione di return? Semplicemente perché quando sono invocate svolgono dei compiti, ma non devono restituire alcun valore di ritorno: ed è proprio per questo che il loro tipo è **void**.

d) L'operatore ?: (riga 27): in fondo alla funzione loop() abbiamo usato un particolare operatore che si chiama "*operatore di assegnazione condizionale*" e che è definito in molti linguaggi di programmazione; è un operatore ternario (cioè prende in ingresso tre valori) e viene utilizzato per scrivere in modo molto compatto un'assegnazione che però dipende da una condizione.
Quanto trova una scrittura del tipo "**condizione ? A : B**" il processore di Arduino controlla la **condizione** che sta prima del "?": se questa è vera, allora l'espressione vale **A**; se invece è falsa, l'espressione viene valutata come **B**.
In figura ❶ è riportata la riga in cui abbiamo usato l'operatore **?:** ed il codice che avremmo dovuto usare se avessimo voluto scrivere in forma espansa il controllo e le assegnazioni.

❶ L'operatore ?:

Questa espressione vale HIGH o LOW, in base alla condizione (distanza < 10)

```
digitalWrite(pinLED, (distanza < 10) ? HIGH : LOW); // se è vicino, accendiamo il LED
```

Possiamo espandere la riga precedente in questo blocco if-then-else

```
if (distanza < 10) {        // se rileviamo un oggetto vicino (a meno di 10 cm)
  digitalWrite(pinLED, HIGH);    // accendiamo il LED sulla schedina
}
else {
  digitalWrite(pinLED, LOW);     // altrimenti lo teniamo spento
}
```

Come vedi la scrittura che utilizza l'operatore **?:** (detto anche l'operatore di Elvis, dato che sembra un emoticon col ciuffo) è molto compressa, ma è anche meno chiara e leggibile delle istruzioni in cui abbiamo usato l'if-then-else. Impiega quindi quella scrittura a tuo rischio e pericolo: questo sarà il primo passo per diventare un vero hacker! :-D

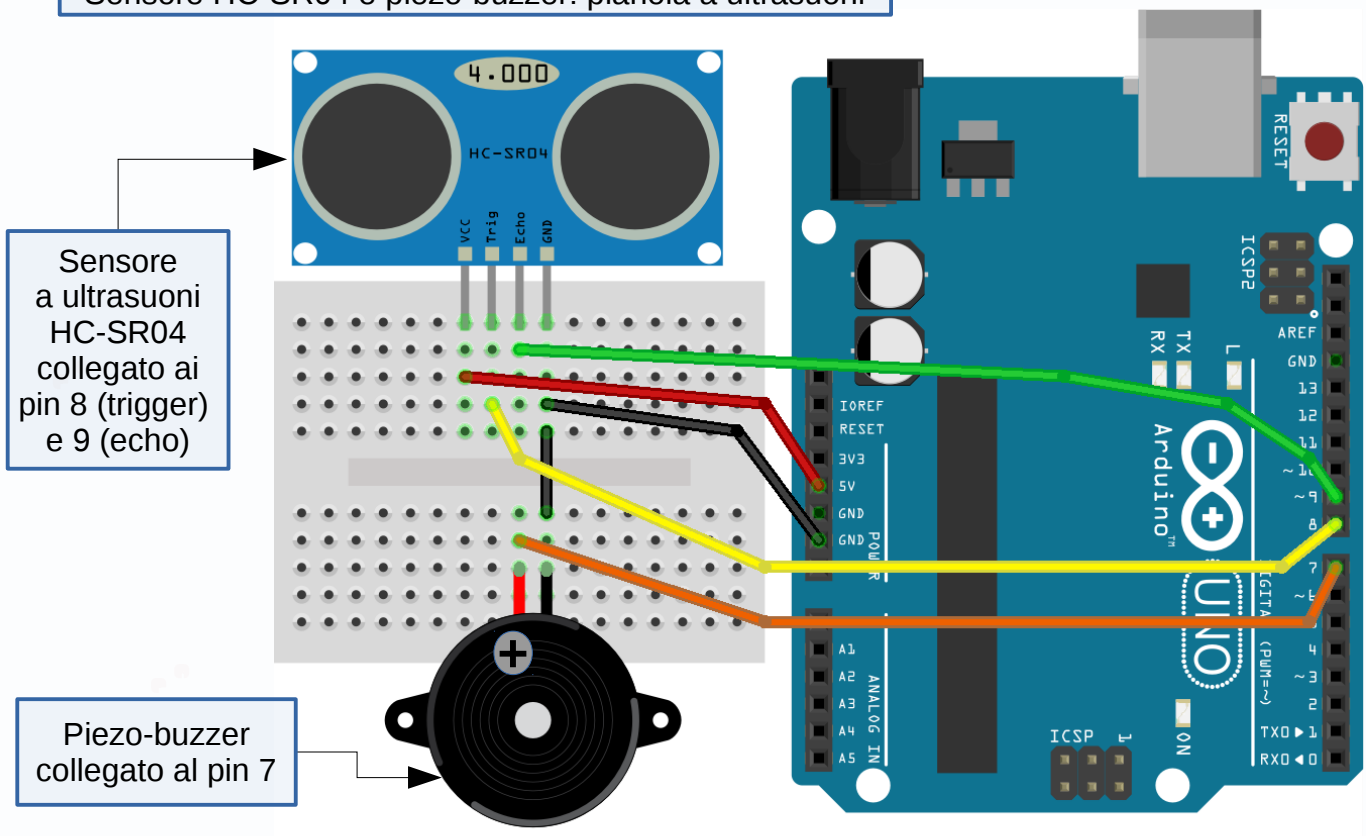
e) I datasheet: se abbiamo un componente che non sappiamo usare bene o del quale non conosciamo le caratteristiche, possiamo cercare aiuto nella documentazione o nei vari tutorial che troviamo in rete; se però ancora mancano delle informazioni che ci permettono di capire come utilizzarlo, dobbiamo cercare il suo "**foglio delle specifiche**" (in inglese: **datasheet**). É un documento scritto di solito dalla casa costruttrice che riassume le caratteristiche tecniche del componente: le dimensioni, la piedinatura, i voltaggi coi quali può essere alimentato, l'intervallo di misurazione, la precisione, i fattori che possono influenzarla, le condizioni ambientali di utilizzo (alcuni componenti ad esempio non lavorano bene, se la temperatura è troppo alta o troppo bassa) e magari degli esempi tipici di utilizzo.
Cercando su internet il datasheet per il sensore HC-SR04 troviamo diversi documenti, ed andando a leggere uno dei primi risultati (http://elecfreaks.com/store/download/HC-SR04.pdf), scopriamo ad esempio la portata del sensore (misura distanze da 2 centimetri fino a 4 metri) e la sua la precisione (fino a 3 mm), come fare per attivare la misura (come dev'essere fatto il segnale che mandiamo come trigger: un impulso da 10 µs), le dimensioni e le caratteristiche dell'ostacolo per assicurare un buon rimbalzo dell'eco, l'angolo di lavoro, la frequenza con la quale può effettuare le misure (utile per capire se dobbiamo magari introdurre dei tempi di attesa come abbiamo fatto a riga 15 dello sketch) e così via.

Sperimenta per bene il comportamento di questo componente: usa una stecca centrimetrata per valutare la sua precisione, perché ci servirà nel prossimo progetto!

6. Una pianola ad ultrasuoni

Non parliamo chiaramente di una pianola che solo i cani potrebbero ascoltare, ma di un piezo-buzzer di Arduino in cui la tonalità è controllata grazie ad un sensore di distanza ad ultrasuoni. Possiamo infatti "fondere" i due progetti precedenti e costruire uno strumento musicale in cui i tasti sono soltanto disegnati, e la nota corrispondente verrà suonata posizionando un ostacolo (la nostra mano, o un cartoncino) sopra il tasto.

❶ Sensore HC-SR04 e piezo-buzzer: pianola a ultrasuoni



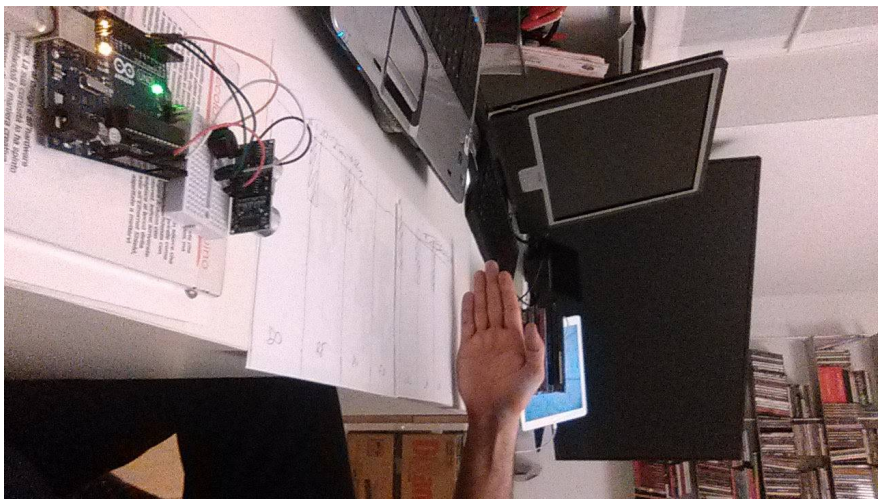
Cominciamo analizzando il circuito di figura ❶: abbiamo semplicemente il sensore ad ultrasuoni collegato ai pin 8 (Trigger) e 9 (Echo) come prima, ed il buzzer è stato collegato al pin 7. Niente di nuovo; la novità sta nell'utilizzo del sensore HC-SR04: lo mettiamo a lato di una tastiera "immaginaria" che possiamo disegnare su un cartoncino lungo più di 80 cm, oppure costruirci con oggetti che ci permettano di distinguere otto bei grandi tasti (la larghezza deve essere di almeno 10 centimetri); se vuoi puoi stampare su un foglio A3 il PDF allegato al tutorial, ritagliarlo e costruire la tastiera incollando i due pezzi ottenuti. La tastiera così ottenuta va messa su un tavolo sgombro, senza oggetti che interferiscono.

Nell'immagine ❷ si vede come conviene disporre il sensore: lo mettiamo vicino al DO più basso (quello a sinistra), in modo che "guardi" lungo tutta la pianola e lo teniamo un po' sollevato dal tavolo appoggiandolo sopra un libro.

Tieni presente che per non interferire con la misura devi stare abbastanza lontano col corpo, in modo che la posizione della mano venga rilevata senza disturbi.

Inoltre ricordati di mettere un grosso ostacolo alla fine della tastiera, altrimenti l'eco si perde su oggetti lontani e la misura si rallenta troppo.

❷ Il sensore a ultrasuoni "vede" che nota stai suonando



Il listato dello sketch (figura ❶) è lungo ma non è complesso, dato che si limita a mettere assieme alcune tecniche che già conosciamo: a riga 1 l'inclusione di un file .h del nostro progetto, a riga 8 l'uso di un vettore per tenere una lista di note (stavolta sono le note di un'ottava completa), l'acquisizione di una misura di distanza (a righe 19-25 l'interrogazione del sensore; a riga 26 l'invocazione della funzione definita dalla riga 48 alla 54, per trasformare in centimetri il

progetto_10_Pianola_a_ultrasuoni | note.h ❶

```
1  #include "note.h"     // trascrive all'interno dello sketch il contenuto del file note.h
2  const int pinPiezo = 7;     // pin digitale al quale colleghiamo il piezo-speaker
3  const int pinImpulso = 8;   // pin sul quale scrive l'impulso
4  const int pinEco = 9;       // pin dal quale legge il tempo dopo il quale arriva l'eco
5  const int pinLED = 13;      // pin al quale abbiamo collegato il LED (quello "on board")
6
7  // vettore di interi che contiene la lista delle 8 note di un'ottava completa:
8  int scala[] = { NOTA_DO5, NOTA_RE5, NOTA_MI5, NOTA_FA5, NOTA_SOL5, NOTA_LA5, NOTA_SI5, NOTA_DO6 };
9
10 void setup() {
11   pinMode(pinPiezo, OUTPUT);     // setta il pin come canale di output
12   pinMode(pinImpulso, OUTPUT);   // setta il pin come canale di output
13   pinMode(pinEco, INPUT);        // setta il pin come canale di input
14   pinMode(pinLED, OUTPUT);       // setta il pin come canale di output
15   Serial.begin(9600);            // inizializza la porta seriale a 9600 baud
16 }
17
18 void loop() {
19   // dopo aver preparato una base pulita, invia un impulso di 10 microsecondi
20   digitalWrite( pinImpulso, LOW );   //
21   delay(60);                         //  1                  +---------+
22   digitalWrite( pinImpulso, HIGH );  //                     |         |
23   delayMicroseconds( 10 );           //  0  -----------------+         +---------- ...
24   digitalWrite( pinImpulso, LOW );   //     <---  60 ms  --->|<- 10 ns ->
25   long tempo = pulseIn( pinEco, HIGH ); // legge il tempo di andata e ritorno dell'eco
26   long distanza = calcolaDistanza(tempo);  // chiama la funzione per trovare la distanza
27
28   // il primo (5) e il secondo numero (55) sono le misure osservate per il primo e
29   // l'ultimo tasto: la funzione map() trasforma così la distanza in una nota
30   int nota = map(distanza, 5, 55, 0, 7);
31
32   Serial.print(distanza);   // scrive la misura sulla seriale
33   Serial.print(" cm ");
34
35   if ((nota >= 0) && (nota <= 7)) {          // se è sopra la tastiera:
36     digitalWrite(pinLED, HIGH);              // accende il LED,
37     tone(pinPiezo, scala[nota], 1000/4);     // suona la nota da un quarto
38     Serial.print(" => nota ");               // e
39     Serial.println(nota);                    // la scrive sulla seriale
40   }
41   else {                                     // altrimenti:
42     digitalWrite(pinLED, LOW);               // spegne il LED,
43     noTone(pinPiezo);                        // spegne il piezo-buzzer
44     Serial.println(" => fuori scala");       // e lo segnala sulla seriale
45   }
46 }
47
48 // funzione che calcola la distanza dell'ostacolo, a partire dal tempo di volo
49 long calcolaDistanza(long microsecondi) {
50   // La velocità del suono è 340 m/s, quindi per coprire un cm ci vogliono 29.4 microsecondi.
51   // Il ping sonoro percorre andata e ritorno, quindi per trovare la distanza dell'ostacolo
52   // dobbiamo prendere la metà della distanza percorsa dal suono.
53   return microsecondi / 29.4 / 2;
54 }
```

Questi due numeri vanno sistemati durante la taratura, descritta all'inizio della pagina seguente

tempo di andata e ritorno dell'eco).

L'unica vera novità sta nell'utilizzo della funzione **map()**: a riga 30 viene infatti impiegata per trasformare una misura di distanza (la posizione dell'ostacolo al cammino degli ultrasuoni) in un indice che ci servirà per suonare una nota (pensando alle posizioni delle note nel vettore scala[ ] definito a riga 8, il primo DO corrisponde allo 0, il RE all'1, … e così via fino al DO più alto che corrisponde al 7).

Dobbiamo quindi capire come scegliere i numeri che -nello script- sono per ora fissati a 5 e 55: corrispondono proprio alle posizione del primo e dell'ultimo tasto e dipendono chiaramente da come prepariamo la nostra tastiera.

Dobbiamo quindi **tarare il programma**, lanciandolo così com'è ed ignorando per ora i suoni che risulteranno; apriamo il Monitor seriale (vedi figura ❷ a pagina 2), posizioniamo la nostra mano sopra al primo DO e prendiamo nota della distanza in centimetri che il sensore misura (potremmo anche usare un righello per vedere la distanza dalla mano al sensore, ma … perché rubare il lavoro al sensore stesso?): scriviamo questo numero al posto del 5 di riga 30.

Ripetiamo poi l'operazione di misura mettendo la mano sopra l'ultimo tasto (il DO più acuto): la distanza in centimetri rilevata dal sensore andrà a sostituire il numero 55 di riga 30. Se ora compiliamo e carichiamo lo sketch, dovremmo riuscire a suonare tutte le note: le due estreme, ma anche quelle intermedie.

La funzione *map()* infatti, fa proprio questo: trasforma il primo numero (il 5, nel caso dello sketch di pagina 12) nel numero 0 ed il secondo numero (il 55) nel numero 7 ed in proporzione le distanze intermedie sono tradotte nei numeri da 1 a 6.

Per capire poi come lo sketch scelga le note, basta leggere le righe da 35 a 45: se il numero calcolato da *map()* sta fra 0 e 7 la nostra mano è sopra la tastiera, quindi il programma prende la nota dal vettore scala[ ] e la suona per una durata pari ad un quarto di battuta (vedi riga 37). Contemporaneamente stampa sulla seriale la misura rilevata e la nota calcolata, ed accende il LED on board. Viceversa (righe 42-44) se siamo fuori dai tasti "suonabili", spegne il LED, zittisce il buzzer e scrive "fuori scala" sulla seriale.

Ora che la pianola è accordata, suona pure tutti i brani che vuoi (purché stiano in una sola ottava e non contengano semitoni…). :-p

Sfida: riguardando quello che è stato fatto per il theremin a luce (vedi lo sketch di pagina 5) riesci ad automatizzare la procedura di taratura? Riesci cioè a scrivere il programma in modo che le posizioni del primo e dell'ultimo tasto vengano acquisite automaticamente senza doverle scrivere a mano in un nuovo sketch?

Trovi questo ed altri tutorial sul nostro sito: coderdojotrento.it/arduino1

Ringraziamenti

Questo tutorial:
* fa parte delle "risorse" di CoderDojo Trento,
* è stato scritto con il supporto di CoderDolomiti.
* è ispirato in gran parte ai tutorial di Arduino presentati in video da Massimo Banzi.

Alcune delle immagini utilizzate in questo tutorial derivano dal sito ufficiale di Arduino (arduino.cc), mentre gli schemi sono stati disegnati con il tool open source Fritzing (fritzing.org).