

Maik Schmidt

Il manuale di Arduino

IMPARARE

a programmare
nel linguaggio di Arduino

REALIZZARE

progetti e prototipi interattivi
partendo da zero



"Un robot non può recare danno agli esseri Umani,
né può permettere che, a causa del suo mancato intervento,
gli esseri Umani ricevano danno."

– Isaac Asimov – prima Legge della robotica

APGEO

Maik Schmidt

APOGEO

© Apogeo s.r.l. - socio unico Giangiacomo Feltrinelli Editore s.r.l.

ISBN edizione cartacea: 9788850330447

*A Yvonne.
La sorellina migliore che esiste sulla Terra.*

Prefazione

Benvenuti in Arduino e nel mondo strabiliante del *physical computing*! Arduino (<http://arduino.cc>) è un progetto open source che comprende elementi hardware e software, ideato inizialmente come una soluzione che potesse fornire a progettisti e creativi una piattaforma per la realizzazione interattiva di prototipi di controllo elettronico. Oggi sono molti gli amatori e i professionisti dell'elettronica che utilizzano Arduino nei progetti di physical computing, e anche voi potete imparare facilmente a fare lo stesso.

Arduino permette di modificare il modo di usare il computer come non accadeva dagli anni Ottanta, quando si poteva costruire un computer assemblando in proprio componenti hardware diversi. Arduino semplifica al massimo lo sviluppo di progetti elettronici artigianali che possono riguardare prototipi innovativi e gadget anche molto sofisticati. Sono finiti i tempi in cui si doveva imparare un sacco di teoria dei segnali elettronici e misteriosi linguaggi di programmazione solo per riuscire a far lampeggiare un led. Ora potete realizzare in pochi minuti il vostro primo progetto Arduino senza ricorrere a conoscenze avanzate di ingegneria.

In effetti potete leggere questo libro anche senza conoscere i rudimenti della progettazione elettronica e riuscire a realizzare immediatamente i primi prototipi funzionanti. Fin dalle prime pagine vedrete come utilizzare i componenti hardware fondamentali e imparerete a scrivere il software necessario per dare vita ai vostri progetti.

Questo libro trascura la teoria fine a se stessa e si concentra sugli aspetti pratici del progetto. Verranno illustrate le nozioni fondamentali che permettono di costruire i progetti presentati; inoltre ogni capitolo include un paragrafo che aiuta a risolvere le situazioni più problematiche (“Cosa fare se non funziona?”). Il libro è quindi una guida rapida che vi permette di realizzare velocemente i vostri progetti.

A chi si rivolge questo libro

Questo testo è particolarmente indicato per chi è interessato all'elettronica, in particolare alla realizzazione di giocattoli automatizzati, giochi interattivi e gadget di vario genere. Arduino è uno strumento che può interessare i

progettisti più creativi, e i progettisti software hanno la possibilità di sfruttare al massimo le potenzialità di questo ambiente di progettazione. Chi ha già avuto modo di sviluppare soluzioni software, per esempio in ambiente C/C++ o Java, può ottenere molto dalle indicazioni qui fornite.

Fondamentale è realizzare, provare e modificare i progetti descritti. Fatelo quanto vi pare e non preoccupatevi di commettere errori. I suggerimenti per la soluzione dei problemi di funzionamento e l'esperienza che acquisirete fin dai primi progetti saranno preziosi per migliorare il vostro lavoro. Leggere testi di elettronica senza effettuare alcuna sperimentazione non vale la metà del lavoro. Ricordate sempre il vecchio monito: ognuno di noi ricorda il 5% di quello che ha sentito, il 10% di quello che ha scritto e il 95% di quello che ha provato personalmente. Non abbiate timore, non sono necessarie conoscenze di elettronica pregresse!

RIFERIMENTO

Chi non ha mai scritto una riga di codice può iniziare a seguire un corso di programmazione oppure leggere un testo per principianti, per esempio, in lingua inglese, *Learn to Program* di Chris Pine (The Pragmatic Programmers, LLC, 2006). Potete successivamente imparare a programmare in C studiando un testo come *Il linguaggio C: principi di programmazione e manuale di riferimento* (Pearson Prentice Hall, 2004), oppure in C++ con *The C++ Programming Language*, di Bjarne Stroustrup (Addison Wesley Longman, 2000). Per un'introduzione in italiano potete leggere *C Pocket* di Enrico Amedeo (Apogeo, 2007).

Contenuti del libro

Questo libro è suddiviso in tre parti: “Iniziare a lavorare con Arduino”, “Otto progetti Arduino” e alcune appendici. Nella Parte I verranno illustrate le nozioni fondamentali che permettono di realizzare i progetti illustrati Parte II, pertanto conviene leggere i capitoli così come sono presentati e svolgere tutti gli esercizi proposti. Anche i capitoli della Parte II andrebbero studiati nella sequenza proposta, dato che ogni capitolo riutilizza tecniche di progetto e parti di programma illustrate nei capitoli precedenti.

Di seguito è riportata una sintesi degli argomenti trattati.

- Il libro presenta inizialmente le nozioni fondamentali dello sviluppo di un progetto Arduino. Vedrete come utilizzare l'ambiente IDE (*Integrated Development Environment*) per compilare e caricare i programmi. Realizzerete rapidamente un primo progetto (il dado elettronico) che spiega l'impiego di componenti fondamentali quali led, pulsanti e resistori. L'implementazione di un generatore di codice Morse illustrerà

invece come sia possibile costruire facilmente librerie di Arduino personali.

- Inizierete poi a lavorare con sensori analogici e digitali. Utilizzerete un sensore di temperatura e un sensore a ultrasuoni per realizzare un misuratore di distanze digitale molto preciso. L'impiego di un accelerometro a tre assi permetterà di costruire un controller di giochi sensibile al movimento e di predisporre un tipico gioco di breakout.
- In elettronica non dovete sempre costruire gadget a partire da zero; spesso potete modificare componenti hardware già esistenti. Vedrete pertanto come è facile assumere il controllo del dispositivo Wii Nunchuk di Nintendo e utilizzarlo nelle vostre applicazioni Arduino.
- L'impiego di Nunchuk per controllare le vostre applicazioni o altri dispositivi è una soluzione interessante ma spesso è più comodo predisporre un sistema di controllo senza fili. Per questo motivo vedrete come realizzare un telecomando wireless universale, che potrete controllare anche utilizzando un browser web.
- A proposito di browser web, è facile collegare Arduino a Internet: vedrete come realizzare un sistema di allarme che invia un messaggio di posta elettronica ogni volta che qualcuno si muove nel vostro soggiorno quando siete assenti.
- Infine, utilizzerete un motore per costruire un divertente dispositivo da associare all'ambiente di sviluppo dei vostri progetti. Il dispositivo si potrà collegare al sistema di progettazione e, ogni volta che si manifesta un problema di funzionamento, sposterà un indicatore per segnalare il nome dello sviluppatore su cui far ricadere la responsabilità del problema.
- Nelle appendici potrete studiare gli elementi base dei circuiti elettrici e della saldatura dei componenti hardware. Verranno inoltre fornite indicazioni preziose che riguardano la programmazione di una porta seriale e la programmazione di una scheda Arduino in generale.

Ogni capitolo inizia con un elenco dettagliato dei componenti e degli strumenti necessari per costruire i progetti illustrati, ed è accompagnato da immagini e schemi che illustrano l'assemblaggio delle diverse parti. Alcuni riquadri nel testo descrivono le funzionalità di altri progetti Arduino e forniscono suggerimenti che potete riportare nei vostri progetti professionali.

Non è detto che la realizzazione dei progetti vada immediatamente a buon fine e la ricerca di errori diventa spesso un compito difficile e impegnativo. Per questo motivo ogni capitolo prevede un paragrafo intitolato “Cosa fare se non funziona?” che illustra i problemi più comuni e le strategie da adottare per risolverli. Prima di leggere le indicazioni fornite dal libro conviene provare a risolvere i problemi per conto proprio: è la modalità di apprendimento più efficace. Nel caso poco probabile che non incontriate dei malfunzionamenti, in fondo a ogni capitolo potete trovare una serie di esercizi per verificare le vostre abilità e conoscenze.

Tutti i progetti di questo libro sono stati provati utilizzando Arduino Uno, Arduino Duemilanove e l’ambiente Arduino IDE, versioni da 18 a 21. Utilizzate ove possibile la versione più recente dei componenti hardware e software.

Arduino Uno e la piattaforma Arduino

Il team di Arduino ha deciso di specificare una versione 1.0 della piattaforma di progetto solo dopo aver prodotto diverse versioni delle schede di microcontrollori e dell’ambiente IDE. La piattaforma 1.0 è stata annunciata a capodanno del 2010 (<http://arduino.cc/blog/2010/01/01/uno-punto-zero/>) e costituisce un punto di riferimento per lo sviluppo successivo di nuovi componenti hardware e software. In tempi più recenti è stato rilasciato il microcontrollore Arduino Uno e sono stati perfezionati l’ambiente IDE e il supporto puntuale delle sue librerie.

Al momento non è ancora completamente chiara la fisionomia della piattaforma Arduino 1.0. Il team di Arduino sembra voler garantirne la piena compatibilità con le versioni precedenti dei controllori; questo libro è aggiornato in base alle nuove schede Arduino Uno ma tutti i progetti sono in grado di funzionare anche con i microcontrollori Arduino meno recenti, per esempio le schede Duemilanove o Diecimila. Questo libro fa inoltre riferimento alla versione 21 della piattaforma Arduino. Potete seguire gli sviluppi della piattaforma collegandovi alla pagina

<http://code.google.com/p/arduino/issues/list?q=milestone=1.0>.

Esempi di codice e convenzioni adottate

Nonostante questo libro si occupi prevalentemente di progetti open source hardware e di componenti elettronici, nelle prossime pagine troverete molti esempi di codice software. Le istruzioni da programma sono necessarie per dare vita all'hardware e per fargli eseguire le operazioni che volete e nel modo desiderato.

I programmi da eseguire con i microcontrollori Arduino sono scritti in C/C++. Le applicazioni che richiedono l'utilizzo del PC fanno riferimento all'ambiente di programmazione Processing (<http://processing.org>), ma nell'Appendice C si vedrà come impiegare altri linguaggi di programmazione per comunicare con le schede Arduino.

Risorse online

Questo libro dispone di una pagina web sul sito dell'editore inglese (<http://pragprog.com/titles/msard>) dove potete scaricare il codice relativo a tutti gli esempi. La pagina web permette al lettore di partecipare alle discussioni dei forum e di incontrare altri lettori e l'autore del libro stesso. Siete inoltre invitati a segnalare eventuali errori tipografici e bug dei programmi collegandovi alla pagina

<http://www.pragprog.com/titles/msard/errata>.

La pagina web dedicata al libro include un collegamento a un album di foto Flickr (<http://bit.ly/foto-arduino>), che contiene tutte le immagini del libro in alta risoluzione e dove potete anche trovare foto di progetti realizzati da altri lettori. Sono ovviamente gradite anche quelle relative ai vostri progetti.

È venuto il momento di iniziare!

Componenti necessari

Di seguito è riportato un elenco dei componenti necessari per realizzare tutti i progetti inclusi in questo libro. Ogni capitolo contiene inoltre un elenco aggiuntivo di altri componenti richiesti dai singoli progetti, in modo da affrontare la realizzazione dei progetti capitolo dopo capitolo senza acquistare preventivamente tutti gli elementi hardware.

L'elenco delle voci è piuttosto lungo, ma si tratta prevalentemente di componenti di poco costo che potete acquistare spendendo complessivamente una cifra non superiore ai 200 euro.

Starter Pack

Sono molti i negozi online che vendono componenti elettronici e kit relativi ai microcontrollori Arduino, per esempio Makershed (<http://makershed.com>) e Adafruit (<http://adafruit.com>). Questi siti offrono diverse soluzioni appositamente studiate per iniziare la realizzazione di progetti completi (Starter Pack): si raccomanda di acquistare almeno una di queste soluzioni.

L'offerta migliore e più economica è costituita dal kit Arduino Projects Pack di Makershed (codice prodotto MSAPK), che contiene quasi tutti i componenti necessari per realizzare i progetti di questo libro e altro materiale utile. Oltre ad acquistare il kit Arduino Projects Pack dovete anche procurarvi i componenti indicati di seguito.

- Sensore a ultrasuoni Parallax PING)).
- Sensore di temperatura TMP36 di Analog Devices.
- Accelerometro ADXL335 Breakout Board.
- Header standard a 6 pin 0.1".
- Controller Nintendo Nunchuk.
- Sensore passivo a infrarossi.
- Un led a infrarossi.
- Un ricevitore a infrarossi.
- Un Ethernet shield.

In alternativa, anche Adafruit offre un Arduino Starter Pack (ID prodotto 170). Questa soluzione è più economica ma non contiene molti componenti presenti nel kit Makershed; per esempio, non include un circuito stampato millefori e l'accelerometro (*tilt sensor*).

Tutti i negozi aggiornano l'offerta di starter pack, pertanto si suggerisce di verificare con attenzione le ultime novità presenti nei cataloghi online.

Elenco completo dei componenti

Chi preferisce acquistare i componenti elettronici singolarmente, oppure in base a quanto richiesto dai singoli progetti, piuttosto che affidarsi ai kit starter pack può fare riferimento all'elenco che segue per conoscere tutti i componenti utilizzati in questo libro. Ogni capitolo è inoltre corredata di un elenco componenti e di immagini che riproducono tutto il materiale necessario per realizzare i vari progetti. I siti suggeriti di seguito rappresentano solo un'indicazione di massima per l'acquisto dei componenti elettronici. Non dovete fare altro che eseguire alcune ricerche online per individuare i negozi che vi risultano più comodi e convenienti.

Tra i negozi che permettono di acquistare singoli componenti elettronici si ricordano RadioShack (<http://radioshack.com>), Digi-Key (<http://digkey.com>), Sparkfun (<http://sparkfun.com>) e Mouser (<http://mouser.com>).

- Una scheda microcontrollore (o *microcontroller*) Arduino, per esempio Uno, Duemilanove o Diecimila, disponibile presso Adafruit (ID prodotto 50) o Makershed (codice prodotto MKSP4).
- Un cavo USB standard A-B per porta USB 1.1 o 2.0. È probabile che abbiate già a disposizione un cavo di questo genere, altrimenti lo potete ordinare dal sito di RadioShack (numero catalogo 55011289).
- Una breadboard, da acquistare, per esempio, presso Makershed (codice prodotto MKKN2) o Adafruit (ID prodotto 64).
- Tre led (altri quattro sono richiesti per svolgere uno degli esercizi facoltativi). In genere non conviene acquistare un led alla volta: è preferibile acquistare una confezione di almeno 20 componenti, simile a quella offerta da RadioShack (numero catalogo 276-1622).
- Un resistore da 100Ω , due da $10k\Omega$ e tre da $1k\Omega$. Non conviene mai acquistare i resistori in quantità così ridotte; procuratevi un kit di resistori

- simile al value pack offerto da RadioShack (numero catalogo 271-308).
- Due tasti in miniatura. Non conviene acquistare tasti in confezione singola; potete per esempio acquistare una confezione che contiene quattro tasti in miniatura presso RadioShack (numero catalogo 275-002).
 - Cavi di collegamento per breadboard, che potete acquistare per esempio da Makershed (codice prodotto MKSEED3) o Adafruit (ID prodotto 153).
 - Un sensore Parallax PING))) (codice prodotto MKPX5 presso Makershed).
 - Un sensore passivo a infrarossi (codice prodotto MKPX6 presso Makershed).
 - Un sensore di temperatura TMP36 di Analog Devices (<http://www.analog.com/en/sensors/digital-temperature-sensors/tmp36/products/product.html>). Potete trovare questo componente collegandovi al sito di Adafruit (ID prodotto 165).
 - Una scheda con accelerometro ADXL335 (*breakout board*) che potete trovare sul sito di Adafruit (ID prodotto 163).
 - Un header standard a 6 pin 0.1". Lo trovate incluso alla scheda ADXL335 che acquistate da Adafruit; in alternativa potete ordinare lo stesso connettore presso Sparkfun (cercate i prodotti *breakaway headers*). In genere potete acquistare strisce di connettori che hanno più pin del necessario; in questo caso dovete accorciare la striscia di pin in base alle vostre esigenze.
 - Un controller Nintendo Nunchuk, che potete acquistare in qualsiasi negozio di videogiochi o collegandovi al sito <http://www.amazon.com/>.
 - Una scheda Arduino Ethernet shield (codice prodotto MKSP7 presso Makershed).
 - Un sensore a infrarossi, per esempio il PNA4602, che potete trovare sul sito di Adafruit (ID prodotto 157) o di Digi-Key (cercate il prodotto PNA4602).
 - Un led a infrarossi. Potete trovarlo tra le offerte di RadioShack (numero catalogo 276-143) o presso Sparkfun (cercate "infrared led").
 - Un servomotore a 5V, per esempio il componente Hitec HS-322HD oppure Vigor Hextronic. Potete trovarlo presso Adafruit (ID prodotto 155) o da Sparkfun. Procuratevi servomotori standard che abbiano una tensione di alimentazione compresa tra 4,8V e 6V.

Alcuni esercizi inclusi nel libro richiedono i componenti elencati di seguito.

- Una scheda per prototipi Arduino Proto Shield di Adafruit (ID prodotto 51) oppure di Makershed (codice prodotto MKAD6). Dovete procurarvi anche una piccola breadboard (codice prodotto MKKN1 da Makershed). Si raccomanda vivamente di utilizzare questo tipo di scheda!
- Un altoparlante piezo o un buzzer. Cercate “piezo buzzer” nel sito di RadioShack o procuratevi questo componente da Adafruit (ID prodotto 160).
- Un sensore accelerometro (*tilt sensor*) simile a quello offerto da Adafruit (ID prodotto 173) oppure da Mouser (numero catalogo 107-2006-EV).

Per le operazioni di saldatura dovete avere a disposizione il materiale indicato di seguito.

- Un saldatore da 25W-30W a punta fine (preferibilmente da 1/16") e un portasaldatore.
- Una bobina di stagno 60/40 (con pasta antiossidante) per la saldatura di componenti elettronici; in genere il filo di stagno ha un diametro di 0.031".
- Una spugna.

Potete trovare il materiale di saldatura presso qualsiasi negozio di componenti elettronici, dove potete anche acquistare utili kit per principianti che includono altri strumenti utili per eseguire saldature accurate. Consultate per esempio il catalogo di Adafruit (ID prodotto 136) o di Makershed (codice prodotto MKEE2).

Parte I

Iniziare a lavorare con Arduino

Capitolo 1 Benvenuti in Arduino

Capitolo 2 Le funzioni di Arduino

Benvenuti in Arduino

Il progetto Arduino venne ideato per progettisti e creativi con poca esperienza tecnica. Anche chi non conosceva la programmazione software poteva fare riferimento al progetto Arduino per realizzare sofisticati prototipi e apparecchiature interattive. Non deve pertanto stupire che i primi passi con Arduino siano molto semplici, soprattutto per tecnici esperti di controllo e automazione.

È fondamentale conoscere gli elementi di base di questo sistema di controllo. Si potrà sfruttarne al massimo le potenzialità solo dopo aver appreso il funzionamento della scheda Arduino, dell'ambiente di sviluppo e di altre tecniche particolari, per esempio la comunicazione seriale tra i dispositivi.

Prima di iniziare è bene comprendere il significato di *physical computing*. Chi ha già lavorato nella programmazione dei computer può rimanere perplesso di fronte a questo termine: dopotutto, i computer sono oggetti fisici che accettano segnali (*input*) da altri oggetti fisici, per esempio da tastiera e mouse, e producono segnali (*output*) audio e video che vengono riprodotti da altoparlanti e display, anche questi oggetti puramente fisici. Ma allora, perché non affermare che la scienza dei computer è nel suo complesso “*physical computing*”?

In teoria, la programmazione convenzionale è intesa come un sottoinsieme del *physical computing*: tastiera e mouse sono *sensori* di segnali di input del mondo reale, mentre display e stampanti sono veri e propri *attuatori*. D'altra parte, è difficile impiegare un computer convenzionale per controllare direttamente il funzionamento di sensori e attuatori, mentre l'utilizzo di una scheda Arduino rende quasi banale il controllo di dispositivi sofisticati e per certi versi misteriosi. I prossimi capitoli illustreranno le modalità di utilizzo delle schede Arduino, mentre nei prossimi paragrafi verrà introdotto il concetto di *physical computing* studiando il funzionamento di una scheda di controllo, gli strumenti di lavoro necessari per metterla in funzione e l'installazione e la configurazione di una scheda. Al termine di questo capitolo sarete in grado

di affrontare la parte più interessante della trattazione, ovvero lo sviluppo di un primo progetto Arduino.

Cosa serve

- Una scheda Arduino, per esempio un modello Arduino Uno, Duemilanove o Diecimila.
- Un cavo USB per collegare la scheda Arduino al computer.
- Un led.
- L'IDE Arduino, l'ambiente di sviluppo software che verrà spiegato più avanti in questo capitolo. L'IDE è necessario per realizzare i programmi di tutti i progetti illustrati in questo libro, pertanto la sua presenza non verrà più richiamata esplicitamente.

Cos'è esattamente un progetto Arduino?

A prima vista può non essere chiara la definizione di “progetto Arduino”, dato che a un esame preliminare scopriamo nomi stravaganti, per esempio Arduino Uno, Duemilanove, Diecimila, LilyPad oppure Seeduino. Il problema nasce dal fatto che non esiste un unico oggetto che possiamo chiamare “Arduino”.

Un paio di anni fa il team di Arduino produsse una scheda a microcontrollore che venne rilasciata con una licenza open source. I negozi di elettronica offrono schede di questo tipo già assemblate, ma è sempre possibile scaricare lo schema elettronico di questa scheda (<http://arduino.cc/en/uploads/Main/arduino-uno-schematic.pdf>) e realizzare per conto proprio il sistema a microcontrollore.

Il team di Arduino ha successivamente perfezionato il progetto hardware e ha rilasciato svariate versioni della scheda di controllo, differenti una dall'altra. In genere queste schede sono identificate da un nome italiano, per esempio Uno, Duemilanove o Diecimila; potete trovare un elenco completo delle schede realizzate dal team di Arduino all'indirizzo

<http://arduino.cc/en/Main/Boards>.

La Figura 1.1 illustra alcune schede Arduino, che risultano diverse nell'aspetto ma che presentano anche molti componenti comuni e possono essere programmate utilizzando gli stessi strumenti e le stesse librerie.

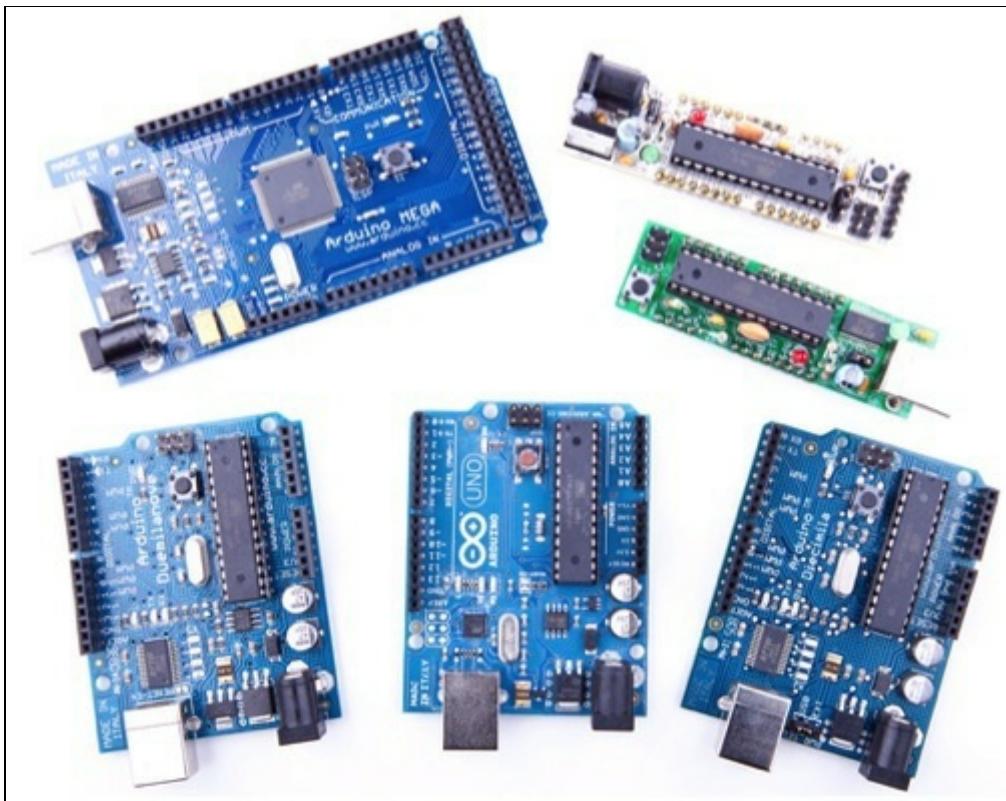


Figura 1.1 Potete scegliere tra diversi microcontrollori Arduino.

Il team di Arduino non si è limitato a migliorare il progetto hardware delle schede a microcontrollore ma ha realizzato anche alcune schede dedicate a un utilizzo particolare; per esempio, la scheda Arduino LilyPad (<http://arduino.cc/en/Main/ArduinoBoardLilyPad>) permette di incorporare la scheda di controllo in un tessuto. Potete così utilizzarla per realizzare t-shirt interattive.

Sul Web potete trovare le schede Arduino “ufficiali” e altri dispositivi cloni. L'utilizzo e la modifica del progetto originale sono consentiti a chiunque e sono molte le persone che hanno realizzato una propria versione di una particolare scheda Arduino.

Potete per esempio trovare schede Freeduino, Seeduino, Boarduino e l'interessante dispositivo Paperduino (<http://lab.guilhermemartins.net/2009/05/06/paperduino-prints/>), un clone Arduino privo di circuito stampato nel quale tutti i componenti sono incollati a un foglio di carta.

Arduino è un marchio registrato e solo le schede ufficiali sono denominate con questo nome, mentre i diversi cloni hanno in genere un nome che termina con “duino”. Per realizzare i progetti illustrati in questo libro siete

liberi di utilizzare una qualsiasi scheda clone che sia compatibile con le specifiche originali del progetto Arduino.

Esaminare la scheda Arduino

La Figura 1.2 mostra l'immagine di una scheda Arduino Uno e i suoi componenti principali. In primo luogo è visibile un connettore USB, che permette di collegare Arduino a un computer tramite cavo USB. La connessione tra scheda e computer permette di effettuare le operazioni indicate di seguito.

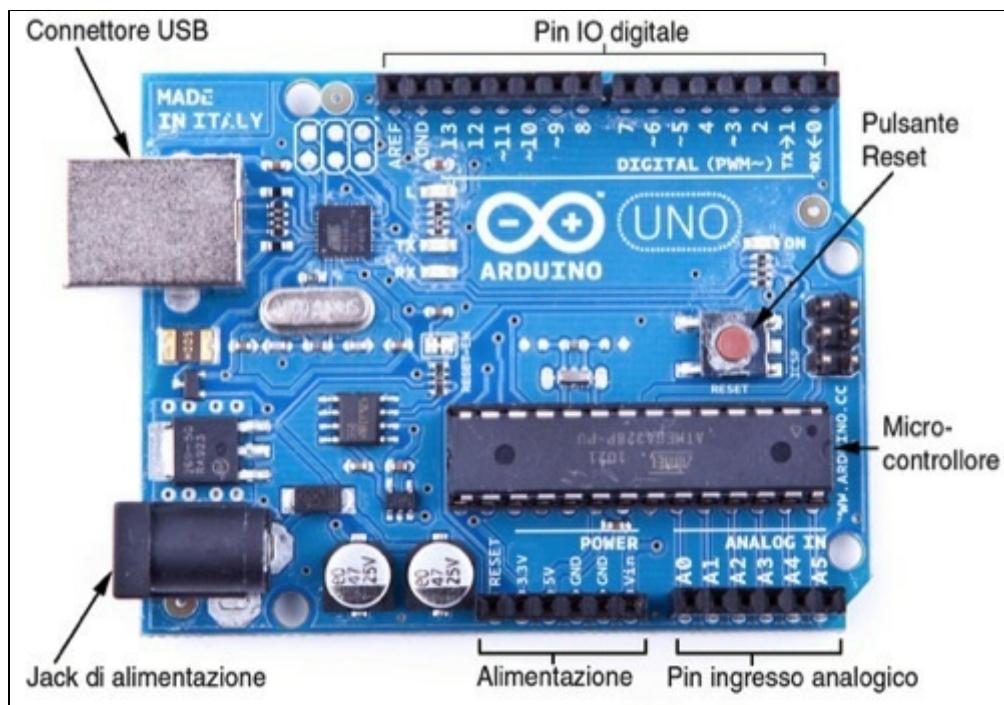


Figura 1.2 I componenti principali di una scheda Arduino.

- Caricare un nuovo programma nella scheda del microcontrollore (come verrà spiegato più avanti in questo capitolo).
- Consentire la comunicazione tra scheda Arduino e computer (si veda il Capitolo 2).
- Fornire alimentazione elettrica alla scheda Arduino.

Arduino è un dispositivo elettronico e in quanto tale deve essere alimentato con energia elettrica. Potete fornire alimentazione collegando per esempio la scheda a un computer tramite la porta USB, anche se questa soluzione non è sempre conveniente. Alcuni progetti non richiedono necessariamente il collegamento della scheda con un computer e sarebbe quantomeno stravagante utilizzare un computer solo per fornire alimentazione elettrica

alla scheda del microcontrollore. Occorre anche tenere presente che la porta USB mette a disposizione 5V di alimentazione, ma alcuni progetti richiedono anche tensioni diverse.

In questi casi conviene allora utilizzare un alimentatore in CA da 9V (Figura 1.3); in effetti si può impiegare un alimentatore qualsiasi in grado di fornire una tensione compresa tra 7V e 12V

(<http://www.arduino.cc/playground/Learning/WhatAdapter>). L'alimentatore deve avere un connettore a punta cilindrica di 2,1 mm con positivo centrale: non è necessario comprendere il significato di queste specifiche, potete semplicemente chiedere un alimentatore di questo tipo al vostro fornitore di materiale elettrico. Collegate l'alimentatore al jack corrispondente della scheda Arduino e potete immediatamente mettere in funzione il sistema di controllo, anche se non avete collegato la scheda al computer. La tensione esterna fornita dall'alimentatore è utilizzata dalla scheda anche se collegate Arduino a un computer via USB.



Figura 1.3 Potete alimentare una scheda Arduino con un alimentatore in CA.

Ricordate però che le prime versioni di schede Arduino (Arduino-NG e Diecimila) non commutano automaticamente l'alimentazione elettrica tra alimentatore esterno e tensione USB, ma prevedono un apposito ponticello (*PWR_SEL*) che dovete impostare rispettivamente nella posizione EXT oppure USB, come si può vedere nella Figura 1.4.

Ora conoscete due modi per fornire alimentazione elettrica a una scheda Arduino, che peraltro non è avida e permette di condividere con altri dispositivi la tensione che ne alimenta il funzionamento. Nella Figura 1.2 si

può vedere che in una scheda sono presenti più morsetti (o *pin*, in quanto sono collegati internamente ai pin del microcontrollore) relativi a diverse tensioni di alimentazione, come indicato di seguito.

- I pin *3V3* e *5V* permettono di alimentare dispositivi esterni con tensioni di 3,3V oppure di 5V.
- I due pin *Gnd (Ground)* condividono il collegamento a massa/terra con la scheda Arduino.
- I progetti che dovranno essere portatili richiedono una tensione di alimentazione fornita da batterie. Potete collegare alla scheda Arduino una sorgente esterna di alimentazione, per esempio una serie di pile o batterie, utilizzando i pin *Vin* e *Gnd*.

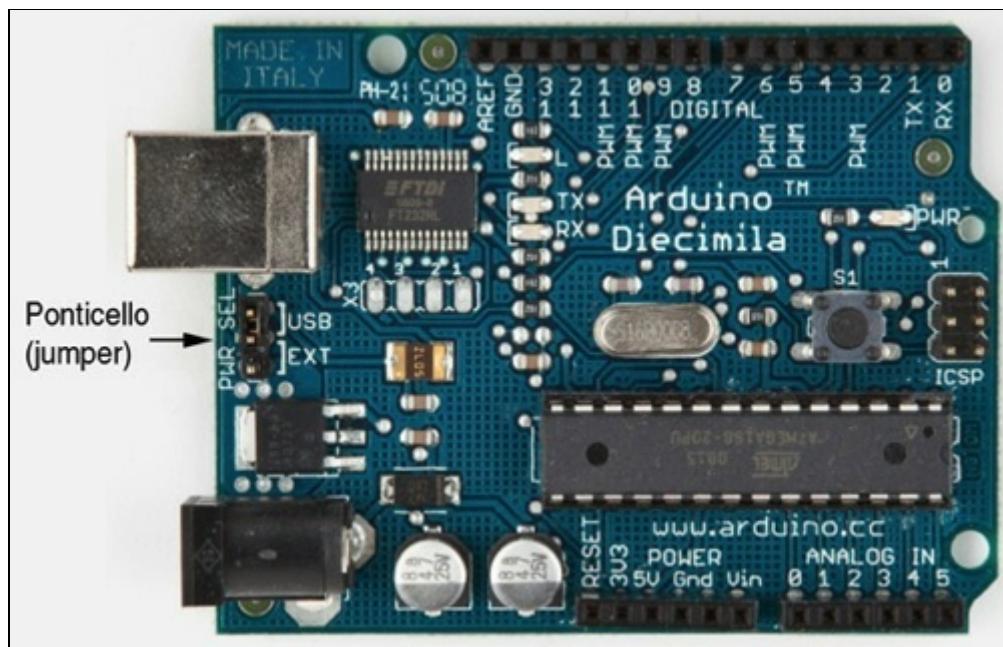


Figura 1.4 Le schede Arduino meno recenti hanno un ponticello per la selezione della tensione di alimentazione.

Il collegamento di un alimentatore fornisce la tensione di alimentazione della scheda tramite il pin connesso al jack.

Nella parte inferiore destra della scheda potete vedere 6 pin relativi ad altrettanti ingressi analogici denominati da A0 ad A5. Questi pin permettono di collegare più sensori analogici alla scheda Arduino. I dati dei sensori vengono convertiti in un numero compreso tra 0 e 1023. Nel Capitolo 5 si vedrà come utilizzare questa funzionalità per collegare Arduino con un sensore di temperatura.

Nella parte superiore della scheda si possono vedere 14 pin di IO digitale denominati da D0 a D13. In base alle esigenze del progetto questi pin possono controllare segnali di input o di output digitale; potete per esempio leggere lo stato ON e OFF di un pulsante (input digitale) o commutare lo stato del pin da ON a OFF per accendere e spegnere un led (output digitale), come si vedrà nel Capitolo 3. Sei di questi pin (D3, D5, D6, D9, D10, D11) possono anche essere utilizzati come output analogici; in questo caso i pin presentano una tensione analogica che dipende da un valore numerico compreso tra 0 e 255.

Segnali analogici e digitali

Quasi tutti i fenomeni fisici hanno a che fare con grandezze analogiche. Ogni volta che si osserva un evento naturale, per esempio elettrico o acustico, si sta in realtà ricevendo un segnale analogico. Una delle proprietà fondamentali dei segnali analogici è che sono continui. In un determinato istante di tempo è sempre possibile misurare l'intensità del segnale e da un punto di vista teorico si può rilevare una variazione minima di qualsiasi segnale analogico.

Se la Natura è sostanzialmente analogica, è altrettanto vero che stiamo vivendo nell'era digitale. Fin dalla comparsa dei primi computer, qualche decina di anni fa, ci si rese immediatamente conto che è molto più semplice elaborare le informazioni del mondo reale dopo averle rappresentate in forma numerica (digitale) piuttosto che come segnale analogico di tensione o di volume. Per esempio, è molto più semplice elaborare segnali audio con un computer dopo che le onde sonore sono state memorizzate come sequenza di numeri, dove ogni numero della sequenza descrive l'intensità del segnale in un determinato istante di tempo. Invece di memorizzare il segnale analogico in modo continuo (come veniva fatto per i dischi in vinile), la "misura" del segnale è eseguita solo in particolari istanti di tempo (Figura 1.5); questa tecnica di registrazione prende il nome di campionamento e i valori memorizzati prendono il nome di campioni. La frequenza di rilevazione dei campioni è chiamata sampling rate. Nel caso di un CD audio il sampling rate è di 44,1 kHz, ovvero si memorizzano 44.100 campioni al secondo.

La digitalizzazione di un segnale analogico richiede inoltre la limitazione della misura dei campioni entro un certo intervallo di valori. In un CD audio ogni campione è rappresentato da 16 bit; nella Figura 1.5 si può notare che l'intervallo di campionamento è definito da due linee tratteggiate orizzontali: nell'esempio è stato "tagliato" un picco del segnale visibile all'inizio del campionamento.

Arduino permette di effettuare il collegamento con dispositivi analogici e digitali, anche se in genere non ci si deve occupare più di tanto della forma del segnale da controllare, dato che la scheda Arduino è in grado di eseguire automaticamente la conversione tra segnali analogici e digitali o viceversa.

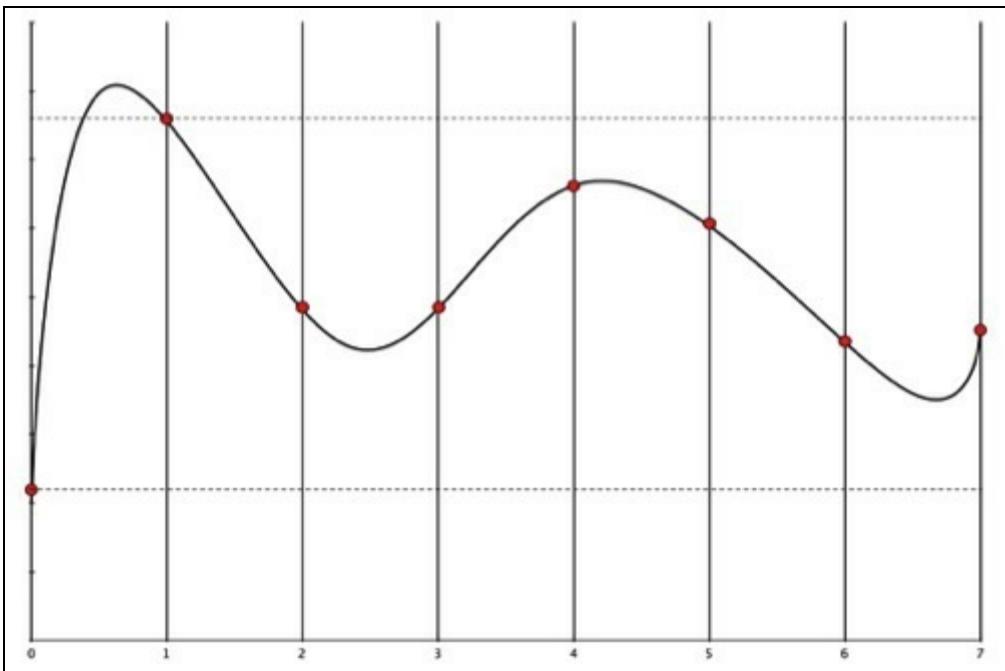


Figura 1.5 Digitalizzazione di un segnale analogico.

I pin descritti finora sono collegati direttamente a un microcontrollore. Questo componente elettronico combina le funzioni tipiche di una CPU e di svariati dispositivi periferici, che riguardano per esempio il controllo dei canali IO. Esistono molti tipi di microcontrollore, ma in genere Arduino propone schede che montano un ATmega328 oppure ATmega168. Entrambi questi componenti sono a 8 bit e vengono prodotti da Atmel.

I computer caricano i programmi da eseguire leggendoli da un disco fisso, mentre i microcontrollori devono essere programmati direttamente; ciò significa che dovete caricare il software nel microcontrollore tramite un collegamento via cavo. Dopo aver caricato un programma, questo rimane installato nel microcontrollore fintanto che non viene sovrascritto da un nuovo programma.

È sufficiente fornire alimentazione a una scheda Arduino per mandare in esecuzione il programma attualmente memorizzato nel microcontrollore presente nella scheda. A volte si richiede di avviare Arduino dalla prima istruzione del programma, operazione che si può effettuare utilizzando il pulsante *Reset* visibile lungo il lato destro della scheda. Premete *Reset* per inizializzare il funzionamento della scheda e avviare dall'inizio il programma memorizzato nella scheda, come si vedrà nel Capitolo 3.

In questo paragrafo si fa riferimento alla scheda Arduino Uno, la versione più recente tra quelle disponibili. Ricordate però che esistono altri tipi di schede, il cui funzionamento è teoricamente identico, anche se alcuni

dettagli risultano differenti. Arduino Mega2560 (<http://arduino.cc/en/Main/ArduinoBoardMega2560>) ha un numero maggiore di pin IO rispetto ad altre schede e utilizza il potente microcontrollore ATmega2560, mentre Arduino Nano (<http://arduino.cc/en/Main/ArduinoBoardNano>) è stato ideato per essere collegato direttamente a una breadboard, pertanto non presenta alcun pin di IO. Si suggerisce di iniziare con una scheda “standard”, come Arduino Uno o Duemilanove.

Installazione della IDE Arduino

I responsabili del progetto Arduino hanno realizzato un ambiente di sviluppo, o IDE (*Integrated Development Environment*), che facilita l'apprendimento delle funzioni che si possono svolgere con questo genere di microcontrollore. L'IDE può essere eseguita utilizzando diversi sistemi operativi e deve essere installata rispettando le indicazioni illustrate di seguito.

Installare la IDE Arduino in Windows

L'IDE funziona con tutte le versioni più recenti di Windows, per esempio Windows XP, Windows Vista e Windows 7. L'installazione è un'operazione molto semplice in quanto l'archivio ZIP da scaricare (<http://arduino.cc/en/Main/Software>) non richiede la presenza di un apposito installer. Scaricate da Internet l'archivio compresso ed estraete i file contenuti in una directory sul vostro disco fisso.

Prima di avviare l'IDE dovete installare i driver richiesti dalla porta USB di Arduino. La procedura di installazione dipende dal tipo di scheda che avete a disposizione e dalla versione di Windows, ma in ogni caso dovete innanzitutto collegare Arduino a una porta USB del computer per avviare la procedura.

Windows Vista dovrebbe avviare automaticamente l'installazione del driver. Non dovete far altro che osservare i messaggi visualizzati dalla procedura e attendere che il computer segnali la possibilità di utilizzare il nuovo hardware USB.

Windows XP e Windows 7 possono non rilevare automaticamente i driver tra quelli disponibili nei siti di aggiornamento di Microsoft. A un certo

punto la procedura di installazione chiede di indicare il percorso da impostare per individuare i driver USB, dopo aver escluso l'installazione automatica dei driver da Internet. La posizione dei driver dipende dal tipo di scheda Arduino; per esempio, nel caso di Arduino Uno e Arduino Mega 2560 dovete rispettivamente selezionare i file `Arduino UNO.inf` e `Arduino MEGA 2560.inf` nella directory *drivers*. Per quanto riguarda le schede meno recenti, per esempio Duemilanove, Diecimila o Nano, selezionate la directory *drivers/FTDI USB Drivers*.

Dopo aver installato i driver potete avviare con un doppio clic il file eseguibile `Arduino` che trovate nella directory principale dell'archivio estratto in precedenza. Seguite le istruzioni fornite a schermo per installare la IDE. Tenete presente che i driver USB non vengono modificati spesso, a differenza delle versioni di IDE, che possono essere aggiornate più frequentemente. Ogni volta che installate una nuova versione dell'ambiente di sviluppo è bene verificare se sono presenti anche nuovi driver USB da installare, operazione che in genere non è però necessaria.

Installare la IDE Arduino in Mac OS X

La IDE Arduino IDE è disponibile come immagine disco per la versione più recente di Mac OS X (<http://arduino.cc/en/Main/Software>). Scaricate il file immagine, fate doppio clic sulla sua icona e trascinate l'icona Arduino nella vostra cartella *Applicazioni*.

Chi utilizza Arduino Uno oppure Arduino Mega 2560 può avviare direttamente l'IDE. Dovete invece installare i driver relativi alla porta seriale di Arduino nel caso in cui stiate utilizzando una scheda meno recente, per esempio Duemilanove, Diecimila o Nano. L'immagine disco include un file binario universale: fate doppio clic sul file

`FTDIUSBSerialDriver_10_4_10_5_10_6.pkg` e seguite le istruzioni di installazione fornite a video.

L'installazione di una versione della IDE non richiede in genere l'installazione di nuovi driver FTDI, che va effettuata solo quando diventa disponibile una versione aggiornata degli stessi.

Installare la IDE Arduino in Linux

In Linux le procedure di installazione possono risultare differenti. L'IDE Arduino funziona correttamente con quasi tutte le versioni recenti di Linux, ma la procedura di installazione dipende dalla distribuzione che si sta impiegando. Va inoltre ricordato che spesso è necessario installare elementi software aggiuntivi (per esempio, la macchina virtuale Java), che sono invece preinstallati in altri sistemi operativi.

Si suggerisce di verificare la documentazione ufficiale del progetto (<http://www.arduino.cc/playground/Learning/Linux>) per individuare le istruzioni che riguardano la distribuzione Linux che vi interessa. Dopo aver installato driver e IDE si può iniziare a esaminare le caratteristiche di questo ambiente di sviluppo.

Conoscere l'IDE Arduino

L'IDE Arduino offre un ambiente particolarmente semplice e chiaro, in particolare rispetto ad altri ambienti di sviluppo quali Eclipse, Xcode oppure Microsoft Visual Studio. L'IDE è costituita sostanzialmente da un editor, un compilatore, un loader e un monitor seriale, come si può vedere nella Figura 1.6 o meglio ancora avviando l'IDE sul vostro computer.



Figura 1.6 L'IDE Arduino si presenta in modo chiaro.

Non sono presenti altre funzioni avanzate, per esempio un debugger o strumenti di completamento automatico delle istruzioni. Potete modificare una piccola serie di preferenze e, analogamente alle applicazioni Java, l'IDE Arduino non si integra pienamente con le funzioni della scrivania Mac. L'ambiente può comunque essere utilizzato a piacere e mette a disposizione un supporto adeguato per la gestione dei progetti Arduino. La Figura 1.7 mostra la barra degli strumenti dell'IDE che permette di accedere alle funzioni principali.

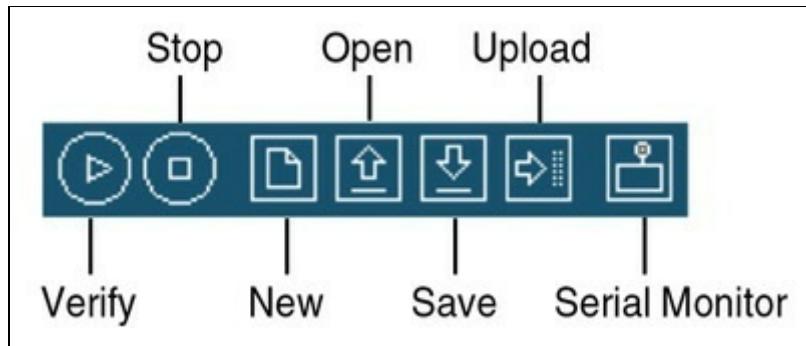


Figura 1.7 La barra degli strumenti dell'IDE permette di accedere velocemente alle funzioni principali.

- Il pulsante *Verify* compila il programma attualmente presente nell'editor. Da un certo punto di vista il nome del pulsante è improprio, dato che l'attivazione di questa funzione non si limita a verificare la sintassi del programma ma lo converte in una forma compatibile con la scheda Arduino.
- Il pulsante *New* crea un nuovo programma svuotando il contenuto della finestra dell'editor. Prima di eseguire questa operazione l'IDE offre la possibilità di memorizzare le modifiche apportate dopo l'ultimo salvataggio.
- Con il pulsante *Open* selezionate e aprite un programma già memorizzato nel computer.
- Il pulsante *Save* memorizza il programma presente nell'editor.
- Fate clic sul pulsante *Upload* per compilare il programma e caricarlo sulla scheda Arduino che avete configurato tramite il menu *Tools > Serial Port*, come si vedrà più avanti in questo capitolo.
- La scheda Arduino può comunicare con un computer attraverso la porta seriale. Fate clic sul pulsante *Serial Monitor* per aprire una finestra omonima che permette di esaminare i dati inviati da Arduino e trasmettere dati alla scheda.

- Il pulsante *Stop* interrompe il funzionamento della finestra *Serial Monitor*.

L'utilizzo dell'IDE è abbastanza semplice, ma si possono sempre verificare problemi di funzionamento oppure si può voler eseguire operazioni particolari. In questi casi conviene fare riferimento al menu *Help* per consultare le risorse utili offerte dal sito web del progetto Arduino e studiare le soluzioni proposte per risolvere non solo i problemi più comuni ma, anche per avere a disposizione altra documentazione e tutorial.

Per iniziare a familiarizzare con le funzioni principali dell'IDE provate a realizzare un semplice programma che fa lampeggiare un led (*Light-Emitting Diode*). Un led è una sorgente di luce efficiente ed economica, già presente anche su ogni scheda Arduino, dove un led segnala l'alimentazione elettrica della scheda, mentre altri due lampeggiano per segnalare la trasmissione o la ricezione di dati attraverso la connessione seriale (Figura 1.8).



Figura 1.8 La scheda Arduino presenta una serie di led.

Questo primo progetto si propone di far lampeggiare il led di stato della scheda Arduino. Il led di stato è collegato al pin di IO digitale 13. I pin digitali possono essere considerati una sorta di interruttore che si può trovare in uno dei due stati HIGH oppure LOW. Lo stato HIGH corrisponde alla presenza di 5V di tensione sul pin, il che permette di far scorrere una corrente sul led che si può così accendere. Lo stato LOW interrompe il passaggio di corrente e il led si spegne. Non è necessario approfondire ora lo studio del comportamento elettrico di un led, ma se vi interessa l'argomento potete consultare direttamente l'Appendice A.

Aprite l'IDE e digitate nell'editor le istruzioni riportate di seguito.

[welcome/Helloworld/Helloworld.pde](#)

```
Riga 1 const unsigned int LED_PIN = 13;
```

```

- const unsigned int PAUSE = 500;
-
- void setup() {
5   pinMode(LED_PIN, OUTPUT);
-
- }
-
- void loop() {
-   digitalWrite(LED_PIN, HIGH);
10  delay(PAUSE);
-   digitalWrite(LED_PIN, LOW);
-   delay(PAUSE);
- }

```

Conviene studiare il funzionamento del programma esaminando il significato delle singole istruzioni. Nelle prime due righe si definiscono le costanti `int` tramite la parola chiave `const`. `LED_PIN` indica il pin digitale IO che si sta utilizzando, mentre `PAUSE` imposta l'intervallo di tempo (in millisecondi) di accensione del led.

Ogni programma Arduino richiede la presenza di una funzione `setup()`, che in questo primo progetto ha inizio alla riga 4. Di seguito è riportata la sintassi tipica che definisce una funzione:

```
<tipo del valore restituito> <nome funzione> '(' <elenco di parametri>
')'
```

In questo programma la funzione è denominata `setup()` e il valore restituito è di tipo `void` in quanto non viene restituito alcun valore. L'istruzione `setup()` non si aspetta alcun argomento e per questo motivo l'elenco dei parametri è vuoto. Prima di proseguire è bene saperne di più sui tipi di dati accettati da Arduino.

Tipi di dati in Arduino

Qualsiasi dato presente in un programma Arduino deve essere associato a un tipo di dato. Scegliete il tipo che meglio soddisfa le esigenze del programma di controllo tra le opzioni indicate di seguito.

- I tipi `boolean` occupano un byte di memoria e assumono il valore `true` oppure `false`.
- Le variabili `char` occupano un byte di memoria e memorizzano valori numerici compresi tra -128 e 127. I numeri corrispondono a caratteri espressi nel codice ASCII; per esempio, di seguito sono riportate due

variabili `c1` e `c2` che hanno lo stesso valore:

```
char c1 = 'A';
char c2 = 65;
```

Osservate l'utilizzo della virgoletta singola per indicare letterali di tipo `char`.

- Le variabili `byte` occupano un byte e memorizzano valori numerici compresi tra 0 e 255.
- Una variabile `int` occupa due byte di memoria e permette di memorizzare valori numerici compresi tra -32.768 e 32.767. Anche la variabile senza segno `unsigned int` occupa due byte e può memorizzare valori compresi tra 0 e 65.535.
- Valori numerici più elevati richiedono l'impostazione di tipo `long`. Questa variabile occupa quattro byte di memoria e memorizza valori compresi tra -2.147.483.648 e 2.147.483.647. La variabile senza segno `unsigned long` richiede sempre quattro byte e accetta valori compresi tra 0 e 4.294.967.295.
- Le variabili `float` e `double` assumono al momento il medesimo significato e le potete utilizzare per memorizzare numeri a virgola mobile. Queste variabili occupano quattro byte di memoria e accettano valori compresi tra -3,4028235E+38 e 3,4028235E+38.
- Utilizzate il tipo `void` solo nelle dichiarazioni delle funzioni. Questo tipo imposta una funzione che non restituisce alcun valore.
- Gli array memorizzano insiemi di valori dello stesso tipo:

```
int values[2];           // Un array di due elementi
int values[0] = 42;       // Imposta il primo elemento
int values[1] = -42;      // Imposta il secondo elemento
int more_values[] = { 42, -42 };
int first = more_values[0]; // first == 42
```

Gli array `values` e `more_values` contengono gli stessi elementi e le istruzioni evidenziano semplicemente due modalità diverse di inizializzazione di un array. Ricordate che l'indice di un array inizia con il valore 0 e che eventuali elementi di un array privi di inizializzazione contengono valori casuali.

- Una stringa è un array di valori `char`. L'ambiente Arduino supporta la

creazione di stringhe con una sintassi piuttosto “elastica”; tutte le dichiarazioni riportate di seguito impostano stringhe che hanno lo stesso contenuto:

```
char string1[8] = { 'A', 'r', 'd', 'u', 'i', 'n', 'o', '\0' };
char string2[] = "Arduino";
char string3[8] = "Arduino";
char string4[] = { 65, 114, 100, 117, 105, 110, 111, 0 };
```

Le stringhe devono sempre concludersi con un byte a zero. L'inserimento del byte a zero è automatico nel caso in cui creiate una stringa utilizzando le virgolette doppie; dovete però aggiungere un byte quando determinate la lunghezza reale dell'array corrispondente a una determinata stringa.

Nel Capitolo 8 si vedrà come utilizzare la nuova classe `String` di Arduino.

Arduino chiama la funzione `setup()` quando viene collegata l'alimentazione elettrica e impiega la medesima funzione per inizializzare la scheda e l'hardware collegato a questa. Nel programma si utilizza il metodo `pinMode()` per configurare il pin 13 come pin di output. In questo modo il pin è in grado di fornire la corrente necessaria per accendere il led. Lo stato di default di un pin è `INPUT`; le parole *INPUT* e *OUTPUT* sono costanti predefinite, come si può vedere consultando la documentazione ufficiale del progetto (<http://arduino.cc/en/Tutorial/DigitalPins>).

Un'altra funzione obbligatoria è chiamata `loop()` e nel primo progetto di esempio ha inizio alla riga 8. Questa funzione contiene la logica principale del programma e viene utilizzata da Arduino per creare un loop infinito. La logica di questo programma deve in primo luogo accendere il led collegato al pin 13. Per effettuare questa operazione si utilizza il metodo `digitalWrite()` passando come parametri il numero del pin e la costante `HIGH`. In questo modo il pin fornisce in uscita 5V fino alla modifica successiva e il led collegato al pin risulta acceso.

A questo punto il programma chiama il metodo `delay()` e attende 500 millisecondi senza eseguire alcuna operazione. Durante questo intervallo di tempo il pin 13 rimane in stato `HIGH` e il led è sempre acceso. Il led si spegne quando si imposta di nuovo a `LOW` lo stato del pin utilizzando ancora una volta il metodo `digitalWrite()`. Il programma attende ancora per 500

millisecondi prima di concludere la funzione `loop()`. Il programma Arduino viene avviato di nuovo e il led torna ad accendersi.

Nel prossimo paragrafo si vedrà come mettere in funzione il programma e trasferirlo nella scheda Arduino.

Compilare e caricare i programmi

Prima di compilare e caricare un programma è necessario configurare due impostazioni dell'IDE, che riguardano il tipo di scheda Arduino che si vuole impiegare e la porta seriale di collegamento tra scheda e computer.

È facile identificare il tipo di scheda Arduino, dato che questa informazione è stampata sulla scheda. I tipi più diffusi sono chiamati Uno, DueMilanove, Diecimila, Nano, Mega, Mini, NG, BT, LilyPad, Pro oppure Pro Mini. In alcuni casi dovete anche verificare il tipo di microcontrollore installato sulla scheda, che in genere corrisponde a un componente ATmega168 oppure ATmega328. Il modello di microcontrollore è stampato sul corpo del componente. Identificate il tipo esatto di Arduino a disposizione e selezionate il modello corrispondente nel menu *Tools > Board*.

A questo punto dovete scegliere la porta seriale di collegamento tra Arduino e computer, impostando il menu *Tools > Serial Port*. In Mac OS X il nome delle porta seriale inizia con `/dev/cu.usbserial` oppure `/dev/cu.usbmodem` (nel MacBook Pro dell'autore di questo libro la porta è identificata come `/dev/cu.usbmodemfa141`). Nei sistemi Linux si dovrebbe identificare la porta `/dev/ttyUSB0`, `/dev/ttyUSB1` o una porta analoga; il numero finale dipende dalla quantità di porte USB disponibili nel computer.

In Windows la procedura è leggermente più complessa; per esempio, in Windows XP dovete aprire la finestra *Gestione periferiche (Device Manager)* e individuare la voce *USB Serial Port* nel menu *Ports (COM & LPT)*, come si può vedere nella Figura 1.9. In genere la porta seriale è denominata *COM1*, *COM2* o qualcosa di simile.

Dopo aver selezionato correttamente la porta seriale fate clic sul pulsante *Verify* del menu IDE, in modo da visualizzare un messaggio simile a quello che segue (ricordate che l'IDE Arduino identifica i programmi chiamandoli *sketch*):

```
Binary sketch size: 1010 bytes (of a 32256 byte maximum)
```

Questo messaggio segnala che l'IDE ha compilato con successo il codice sorgente convertendolo in 1010 byte di codice macchina da caricare nella scheda Arduino. Se compare un messaggio di errore dovete controllare di aver digitato correttamente le istruzioni; nel dubbio, potete scaricare il codice dei programmi collegandovi alla pagina web dedicata al libro (<http://www.pragprog.com/titles/msard>). Il numero massimo di byte riportato nel messaggio dipende dal tipo di scheda Arduino a disposizione; per esempio, Arduino Duemilanove indica in genere 14336 byte.

A questo punto fate clic sul pulsante *Upload*. Dopo alcuni secondi dovrebbe comparire un messaggio simile a questo:

```
Binary sketch size: 1010 bytes (of a 32256 byte maximum)
```

Si tratta dello stesso messaggio visualizzato al termine della compilazione del programma, ma questa volta segnala che 1010 byte di codice macchina sono stati trasferiti con successo sulla scheda Arduino. In caso di errore dovete controllare nel menu *Tools* di aver selezionato correttamente il tipo di scheda e di porta seriale.

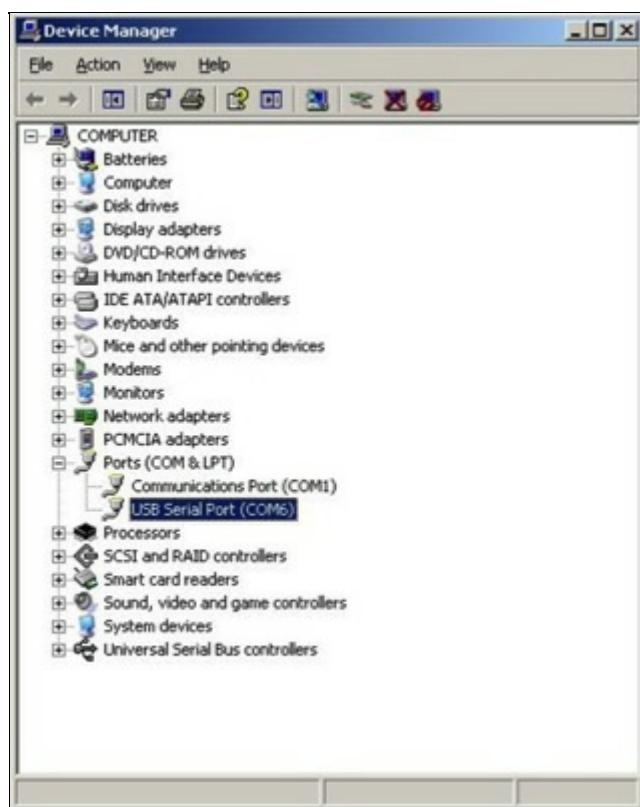


Figura 1.9 Individuazione della porta seriale di collegamento tra scheda Arduino e computer Windows XP.

In fase di caricamento del programma i led TX ed RX lampeggiano per alcuni secondi; ciò è normale e si verifica ogni volta che Arduino e

computer comunicano attraverso la porta seriale. Il led TX si accende quando Arduino invia informazioni al computer, mentre si accende il led RX quando Arduino riceve bit di dati. La comunicazione è decisamente rapida, pertanto i led lampeggiano e non è possibile identificare la trasmissione di ogni singolo byte.

Il programma è eseguito non appena si conclude la trasmissione del codice su Arduino. In questo programma ciò diventa evidente quando il led inizia a lampeggiare e rimane acceso per mezzo secondo, spento per il successivo mezzo secondo e così via.

La Figura 1.10 mostra un grafico che rappresenta la tensione fornita dal pin durante l'esecuzione del programma. Il pin si trova inizialmente in stato LOW e non genera alcuna corrente in uscita. Lo stato HIGH è stabilito dal software utilizzando il metodo `digitalWrite()` e fornisce 5V in uscita per 500 millisecondi. Successivamente, il software imposta lo stato LOW per 500 millisecondi, poi ripete la medesima procedura.

In effetti l'accensione e lo spegnimento del led di stato non producono effetti particolarmente interessanti. Nel prossimo paragrafo si vedrà come collegare un led “reale” alla scheda Arduino.

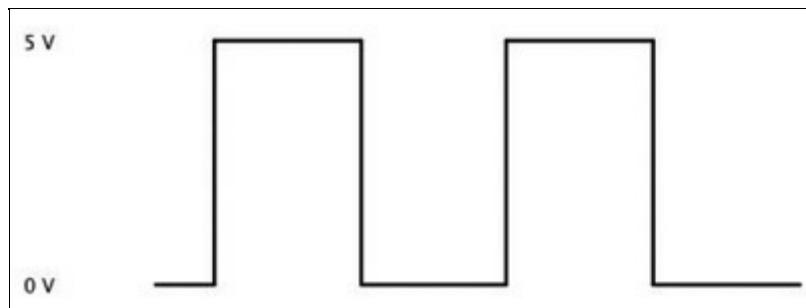


Figura 1.10 Cosa succede nel pin 13 quando il led lampeggia.

Lavorare con i led

I led della scheda Arduino sono utili per verificare il funzionamento dell'apparecchiatura ma non dovrebbero essere utilizzati nei progetti elettronici veri e propri. Sono stati previsti per uno scopo particolare e non devono essere adattati per funzionare in un contesto differente. Va inoltre ricordato che si tratta di componenti molto piccoli e poco luminosi. In definitiva, conviene aggiungere led all'esterno della scheda e imparare come collegarli al microcontrollore. Si tratta di un'operazione molto semplice.

Non dovete utilizzare lo stesso tipo di led montato sulla scheda Arduino: sono componenti SMD (*Surface-Mounted Device*) piuttosto delicati, che richiedono strumenti di cablaggio particolari e molta destrezza. Permettono di risparmiare denaro quando si devono produrre grandi quantità di dispositivi elettronici dello stesso tipo, ma gli hobbisti appassionati di elettronica possono tranquillamente farne a meno.

I led da utilizzare nei progetti di questo libro sono del tipo passante (*through-hole*), simili a quelli mostrati nella Figura 1.11. Il termine “passante” identifica il fatto che vengono montati su circuito stampato facendoli “passare” da un lato all’altro della scheda; per questo motivo sono dotati di un terminale metallico piuttosto lungo.

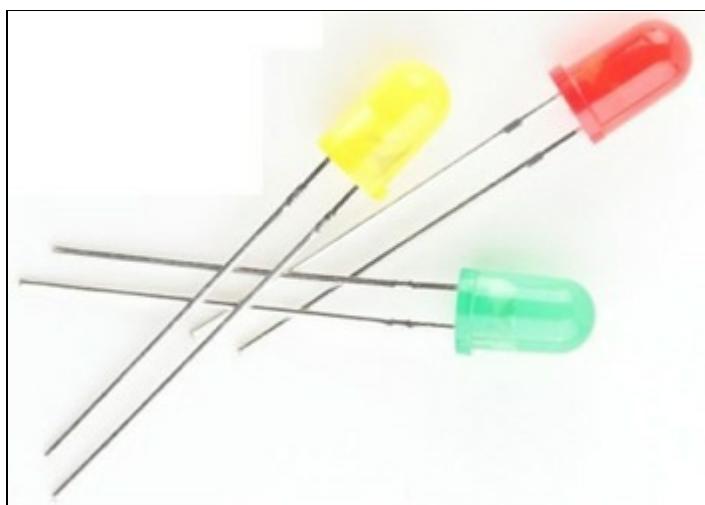


Figura 1.11 Led per circuito stampato.

In primo luogo dovete inserire i terminali del led in un paio di fori del circuito stampato, poi saldate il terminale alla pista di collegamento e tagliate la parte eccedente dei terminali. I led possono anche essere collegati a morsetti simili a quelli disponibili nella scheda Arduino oppure agganciati a una breadboard, come si vedrà nel Capitolo 3.

La Figura 1.12 mostra il collegamento di un led alla morsettiera di una scheda Arduino. Inserite il terminale più corto del led al terminale di massa/terra (GND) e il terminale più lungo al pin 13. Potete eseguire questa operazione anche quando il programma è in esecuzione. Il led di stato e il led esterno devono accendersi e spegnersi simultaneamente.

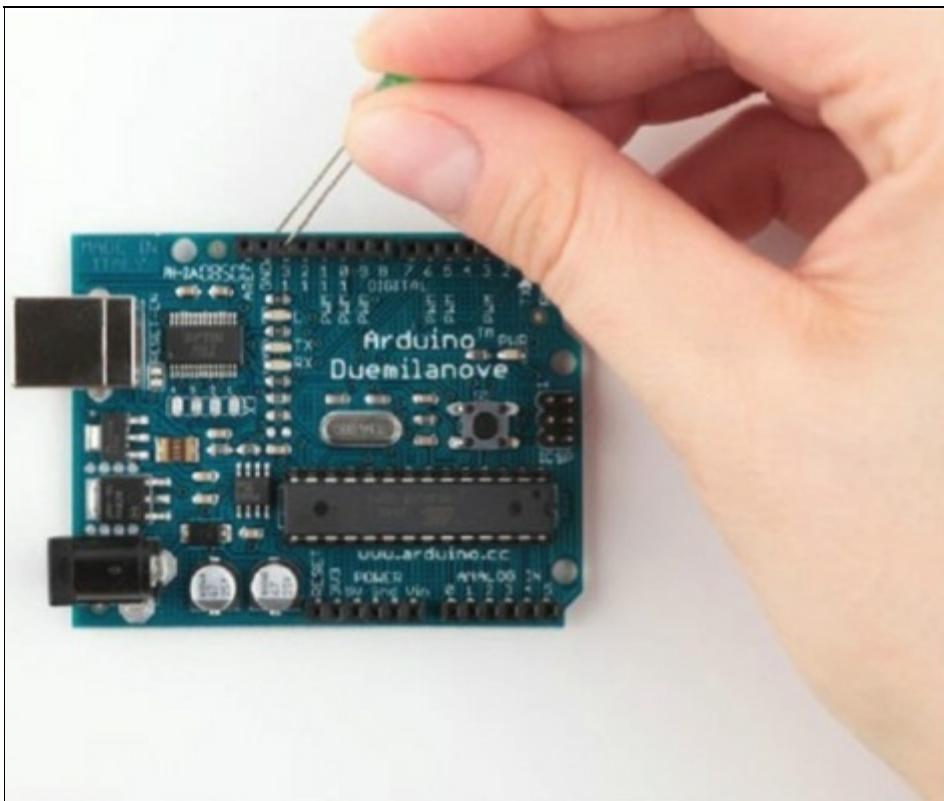


Figura 1.12 Collegamento di un led alla scheda Arduino.

Verificate di collegare correttamente il led al pin 13: se collegate il led a un altro pin rischiate probabilmente di bruciare il componente, dato che solo questo pin ha un resistore interno di protezione, come si vedrà nel Capitolo 3.

Ecco fatto! Avete appena aggiunto ad Arduino un primo componente esterno e avete realizzato il vostro primo progetto di physical computing.

I concetti e le operazioni illustrate in questo capitolo sono utili per realizzare quasi tutti i progetti Arduino. Nel prossimo capitolo vedrete come controllare in modo più accurato il funzionamento dei led e imparerete a utilizzare altre funzioni dell'IDE.

Cosa fare se non funziona?

Niente paura! Se non funziona è probabile che abbiate collegato il led in malo modo. L'assemblaggio di un circuito elettronico distingue tra componenti che possono essere collegati a piacere e componenti che presentano una sola modalità di inserimento nel circuito. Un led ha due terminali chiamati anodo (positivo) e catodo (negativo). È facile confondere i due terminali, così fin da piccolo ho imparato che “il catodo è negativo”. È possibile distinguere i due terminali tenendo presente tra

l’altro che il catodo è in genere più corto dell’anodo. Potete identificare i terminali anche osservando il contenitore plastico del componente: dalla parte del catodo il contenitore è piatto, mentre in corrispondenza dell’anodo il contenitore è arrotondato.

Anche l’impostazione non corretta della porta seriale è un errore comune. Un messaggio “Serial port already in use” in fase di caricamento di un programma significa che dovete controllare la porta seriale che avete selezionato nel menu *Tools > Serial Port*. Messaggi quali “Problem uploading to board” oppure “Programmer is not responding” richiedono invece di controllare l’impostazione della scheda Arduino nel menu *Tools > Board*.

Analogamente a qualsiasi altro codice informatico, anche i programmi Arduino possono contenere bug, mentre gli errori di scrittura e di sintassi possono essere rilevati dal compilatore. Nella Figura 1.13 è riportato un tipico messaggio di errore segnalato dal compilatore.

The screenshot shows the Arduino IDE interface with the title bar "HelloWorld | Arduino 0021". Below the title bar is a toolbar with various icons. The main area displays a sketch named "HelloWorld" with the following code:

```
const int LED_PIN = 13;
const int PAUSE = 500;

void setup() {
    pinMod(LED_PIN, OUTPUT);
}

void loop() {
    digitalWrite(LED_PIN, HIGH);
    delay(PAUSE);
    digitalWrite(LED_PIN, LOW);
    delay(PAUSE);
}
```

A red box highlights the line "pinMod(LED_PIN, OUTPUT);" in the setup() function. Below the code area, a red bar displays the error message "'pinMod' was not declared in this scope". At the bottom of the IDE window, the status bar shows the error message "HelloWorld.cpp: In function 'void setup()'".

Figura 1.13 L’IDE Arduino visualizza esplicativi messaggi di errore.

Invece di digitare **pinMode()** è stato scritto **pinMod()**: dato che il compilatore non ha trovato una funzione denominata in questo modo, la procedura si è interrotta ed è stato emesso un messaggio di errore. L’IDE Arduino evidenzia la riga che ha bloccato la compilazione, mostra

l’istruzione su sfondo in colore giallo e visualizza un messaggio che permette di identificare la causa dell’errore.

RIFERIMENTO

Altri bug possono risultare più complessi da identificare e richiedono di studiare accuratamente il codice o di utilizzare sofisticate tecniche di debugging. Il libro di Paul Butcher *Debug It! Find, Repair, and Prevent Bugs in Your Code* (The Pragmatic Programmers, LCC, 2009) fornisce ampie delucidazioni su questo argomento.

Può perfino succedere, ma l’eventualità è piuttosto rara, che stiate utilizzando un led danneggiato. Se nessuno dei suggerimenti precedenti risolve il problema di funzionamento, procuratevi un nuovo led.

Esercizi

- Generate diverse modalità di accensione/spegnimento del led impostando più intervalli di pausa e modificate la durata delle singole pause, che non devono necessariamente essere identiche tra loro. Provate a impostare pause molto brevi che facciano lampeggiare il led a frequenza elevata. Sapete spiegare l’effetto che si produce in questi casi?
- Fate in modo che il led visualizzi il vostro nome in codice Morse (http://en.wikipedia.org/wiki/Morse_code oppure http://it.wikipedia.org/wiki/Codice_Morse).

Le funzioni di Arduino

Le nozioni illustrate nel capitolo precedente sono sufficienti per realizzare applicazioni poco complesse. Non appena i progetti iniziano a diventare più complicati nasce l'esigenza di separare le singole applicazioni in file distinti da gestire in una sola raccolta complessiva. In questo capitolo si vedrà come l'IDE Arduino controlla il funzionamento di progetti di grandi dimensioni.

In genere i progetti più complessi non richiedono solo una grande quantità di software ma anche componenti hardware aggiuntivi; è raro che la scheda Arduino venga utilizzata da sola. Potete per esempio collegare molti più sensori di quanto riuscite a immaginare ora e che dobbiate trasmettere i dati misurati dai sensori a un computer. Per scambiare dati con Arduino si utilizza la porta seriale della scheda. Questo capitolo spiega tutto ciò che si deve conoscere sulla comunicazione seriale. Per rendere la trattazione più concreta, si vedrà poi come trasformare il computer in un complesso interruttore digitale che permette di controllare da tastiera l'accensione di un led.

Cosa serve

Di seguito sono indicati i componenti necessari per svolgere gli esempi di questo capitolo.

- Una scheda Arduino, per esempio il modello Uno, Duemilanove o Diecimila.
- Un cavo USB per collegare la scheda Arduino al computer.
- Un led (facoltativo).
- Un software di monitoraggio delle comunicazioni seriali (terminale), per esempio Putty (in ambiente Windows) oppure il comando `screen` in Linux e Mac OS X (facoltativo).

Gestire progetti e programmi

Chi sviluppa software oggi può scegliere tra una grande quantità di strumenti di sviluppo che rendono automatica l'esecuzione di operazioni ripetitive e noiose; ciò è vero anche nel caso di sistemi di controllo integrati come Arduino. Potete utilizzare ambienti di sviluppo IDE anche per gestire il funzionamento dei programmi. L'IDE più diffusa è stata ideata dagli stessi progettisti del team di Arduino.

L'IDE Arduino è in grado di gestire tutti i file che riguardano un determinato progetto e mette a disposizione tutti gli strumenti necessari per generare i file binari da mandare in esecuzione nella scheda Arduino. La soluzione è comoda e poco “invadente” nei comportamenti.

Potete per esempio avere già notato che l'IDE Arduino memorizza tutte le istruzioni non appena queste vengono digitate; in questo modo si esclude la possibilità che un progettista alle prime armi perda incidentalmente una serie di dati o di comandi. (A dire il vero, anche i professionisti più esperti rischiano di tanto in tanto di perdere dati preziosi.)

L'organizzazione automatizzata dei file che appartengono a un progetto è una delle caratteristiche fondamentali dell'IDE, che genera in modo trasparente una directory relativa a ogni nuovo progetto e memorizza nella medesima directory tutti i file relativi a uno stesso progetto. Per aggiungere nuovi file in un progetto dovete fare clic sul pulsante *Tabs* visibile a destra e aprire il menu corrispondente, poi selezionare il comando *New Tab* (Figura 2.1). Per aggiungere un file esistente dovete semplicemente scegliere il comando *Sketch > Add File*.

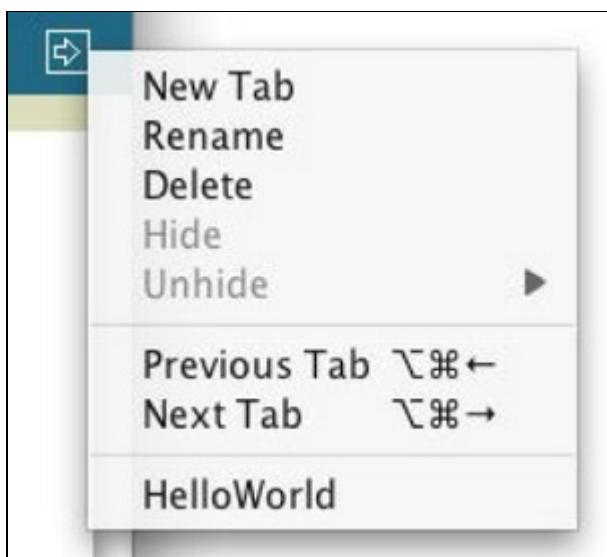


Figura 2.1 Il menu Tabs.

Le voci dei menu ricordano che l'IDE Arduino chiama *sketch* i progetti del sistema di controllo. Se non impostate direttamente il nome del progetto, l'IDE assegna un nome che inizia con `sketch_`. Potete modificare a piacere il nome semplicemente utilizzando il comando *Save As*. Se non salvate il progetto in modo esplicito l'IDE memorizza i file in una cartella predefinita, la cui posizione può essere individuata nel menu delle preferenze. Potete modificare questa impostazione in modo che l'IDE richieda sempre il nome da assegnare a un nuovo progetto. Se necessario, il comando *Sketch > Show Sketch Folder* permette di verificare quale cartella è impiegata dal progetto corrente.

L'IDE utilizza le directory non solo per organizzare i file dei progetti Arduino, dato che memorizza altri elementi interessanti nelle directory indicate di seguito.

- La directory *examples* contiene esempi di progetti da utilizzare come punto di partenza per le prove personalizzate della scheda. Potete accedere a questi progetti tramite la finestra di dialogo *File > Open*. Dedicate un po' di tempo allo studio di questi esempi, anche se al momento non comprendete completamente il significato di tutte le operazioni.
- La directory *libraries* contiene librerie di file che riguardano attività e dispositivi diversi tra loro; per esempio, ogni volta che utilizzate un nuovo sensore avete la possibilità di copiare da questa directory una libreria che supporta proprio il nuovo dispositivo.

L'IDE Arduino semplifica il lavoro di sviluppo adottando impostazioni di default che si possono adattare a molti progetti. Permette altresì di modificare la maggior parte di queste opzioni, come verrà illustrato nel prossimo paragrafo.

Le preferenze dell'IDE

Nei progetti più semplici le impostazioni di default dell'IDE sono più che adeguate, ma presto o tardi si avrà la necessità di modificare qualche opzione. La Figura 2.2 evidenzia come l'IDE permette di modificare direttamente solo alcune preferenze; la finestra di dialogo fa comunque riferimento a un file chiamato `preferences.txt` che contiene molte altre

preferenze. Si tratta di un file di proprietà Java costituito da coppie chiave/valore, come si può vedere negli esempi che seguono:

```
...  
editor.external.bgcolor=#168299  
preproc.web_colors=true  
editor.font.macosx=Monaco,plain,10  
sketchbook.auto_clean=true  
update.check=true  
build.verbose=true  
upload.verbose=true  
...
```

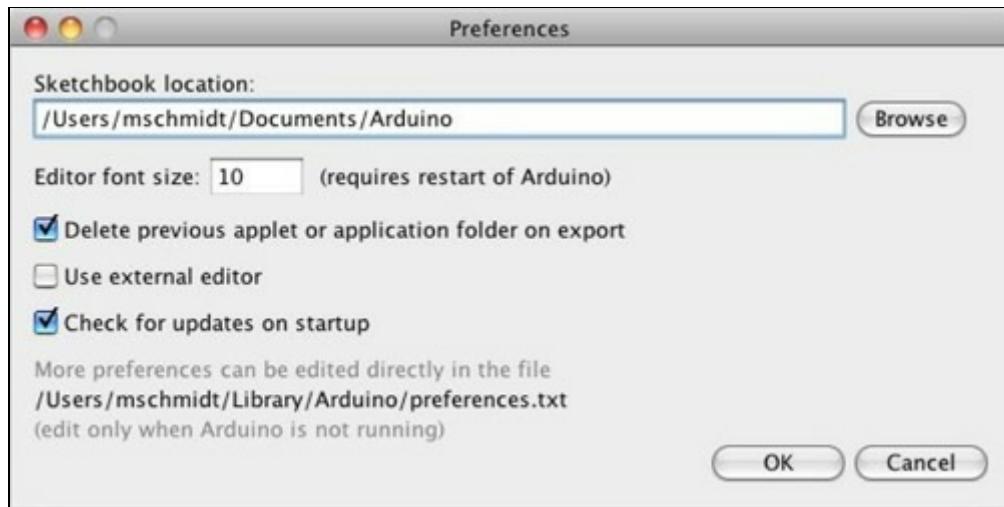


Figura 2.2 L'IDE permette di modificare alcune preferenze.

La maggior parte di queste proprietà riguarda l'interfaccia utente, ovvero impostazioni che modificano i caratteri, i colori e così via. Alcune impostazioni modificano invece il comportamento dell'applicazione. Potete per esempio attivare l'output più dettagliato di operazioni quali la compilazione o il caricamento di un progetto. Modificate il file `preferences.txt` e impostate `build.verbose` e `upload.verbose` con il valore `true`. Provate a caricare il programma sviluppato nel Capitolo 1 ed eseguite una nuova compilazione, in modo da visualizzare un pannello di messaggi simile a quello mostrato nella Figura 2.3. Le versioni più recenti dell'IDE permettono di ottenere lo stesso risultato tenendo premuto il tasto Maiusc quando fate clic sul pulsante *Verify/Compile* o sul pulsante *Upload*.

ATTENZIONE

Ricordate che l'IDE aggiorna alcune preferenze solo quando si chiude l'applicazione. Dovete pertanto interrompere l'esecuzione dell'IDE prima di modificare le preferenze agendo direttamente sul file `preferences.txt`.

Dopo aver acquisito familiarità con l'IDE Arduino potete iniziare a programmare. È venuto il momento di far dialogare la scheda di controllo con il mondo esterno.

Utilizzare le porte seriali

Arduino offre la possibilità di realizzare applicazioni di tipo *standalone*, ovvero progetti che non richiedono l'utilizzo di computer aggiuntivi; in questi casi è necessario collegare una sola volta la scheda Arduino al computer, allo scopo di caricare il software di controllo. Dopo aver eseguito questa operazione il funzionamento della scheda è garantito semplicemente dalla presenza della tensione di alimentazione elettrica. Si verifica spesso però il caso in cui i progettisti utilizzano la scheda Arduino per migliorare le prestazioni di un computer impiegando uno o più sensori oppure offrendo accesso a dispositivi hardware aggiuntivi. In genere il controllo di hardware esterno è gestito attraverso la porta seriale; conviene pertanto conoscere le caratteristiche delle comunicazioni seriali che si possono realizzare con Arduino.



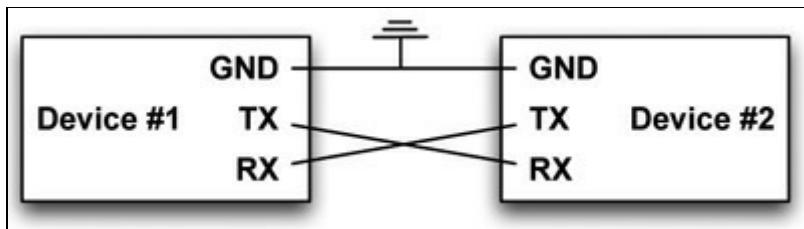
Figura 2.3 L'IDE in modalità testo mostra l'output degli strumenti da riga di comando.

Il linguaggio di programmazione di Arduino

A volte si incontrano persone che provano irritazione quando hanno a che fare con il linguaggio di programmazione delle schede Arduino; questo perché si ritiene erroneamente che il linguaggio in questione sia stato inventato solo per programmare queste schede di controllo. Si tratta di una convinzione totalmente

fuori luogo: il linguaggio da utilizzare è C++, il che sottintende che Arduino supporta anche il linguaggio C. Ogni scheda Arduino si basa su un microcontrollore AVR prodotto da Atmel. La stessa Atmel sostiene che la sigla AVR non ha alcun significato particolare. Questi microcontrollori sono molto diffusi e sono veramente tanti i progetti hardware che si basano su questo genere di componente. Uno dei motivi di successo dei microcontrollori AVR è dato dall'ottimo corredo di strumenti che li accompagna, basato a sua volta sul compilatore C++ GNU e ottimizzato per generare il codice di controllo di questi componenti. Queste considerazioni fanno sì che non dobbiate scrivere codice C++ da tradurre in codice macchina per il vostro computer ma esclusivamente per il microcontrollore AVR. Questa tecnica di progettazione prende il nome di *cross-compiling*, ed è comunemente adottata per programmare dispositivi incorporati.

Le specifiche standard della comunicazione seriale sono cambiate nel tempo; per esempio, ora si utilizzano porte USB e i computer non dispongono più di connettori RS232. Nonostante questo i principi di funzionamento della comunicazione seriale non sono stati modificati. Nei casi più semplici il collegamento tra due dispositivi richiede la presenza di tre soli fili: una linea di massa comune, una linea per la trasmissione dati (TX) una linea per la ricezione dati (RX).



La comunicazione di tipo seriale può sembrare ormai fuori moda, ma costituisce ancora oggi la tecnica preferita per mettere in collegamento dispositivi hardware diversi tra loro. La lettera "S" della sigla USB indica proprio il termine "seriale"; d'altra parte, quanto tempo è passato dall'ultima volta che avete visto una porta parallela? (Forse è venuto il momento di disfarsi definitivamente di quel vecchio PC che tenete in cantina e che avreste sempre voluto trasformare in un media center!)

La porta seriale di Arduino serve innanzitutto a caricare il software di controllo, ma la potete utilizzare anche per collegare Arduino con altri dispositivi. Nel Capitolo 1 avete visto come analizzare il funzionamento della porta seriale di una scheda Arduino; nei prossimi paragrafi si utilizzerà la porta seriale per controllare il led di stato tramite la tastiera del computer. Il led verrà acceso quando premete il tasto 1 e verrà spento quando premete il tasto 2. Di seguito è riportato il codice necessario per realizzare questo progetto.

[welcome/LedSwitch/LedSwitch.pde](#)

```
Riga 1  const unsigned int LED_PIN = 13;
      - const unsigned int BAUD_RATE = 9600;
```

```

-
- void setup() {
5   pinMode(LED_PIN, OUTPUT);
-   Serial.begin(BAUD_RATE);
-
- }
-
- void loop() {
10  if (Serial.available() > 0) {
-    int command = Serial.read();
-    if (command == '1') {
-      digitalWrite(LED_PIN, HIGH);
-      Serial.println("LED on");
15  } else if (command == '2') {
-      digitalWrite(LED_PIN, LOW);
-      Serial.println("LED off");
-    } else {
-      Serial.print("Unknown command: ");
20    Serial.println(command);
-    }
-  }
- }

```

Analogamente agli esempi già incontrati, anche in questo programma si definisce una costante per indicare il pin ove risulta collegato il led e si imposta la sua modalità `OUTPUT` tramite la funzione `setup()`. Alla riga 6 si inizializza la porta seriale utilizzando la funzione `begin()` della classe `Serial`, impostando un baud rate di 9600. (Il baud rate verrà definito nell'Appendice C.) Queste istruzioni permettono al programma di inviare e ricevere dati attraverso la porta seriale.

A questo punto il programma deve leggere e interpretare correttamente i dati. La funzione `loop()` ha inizio con la chiamata del metodo `available()` della classe `Serial` visibile alla riga 10. Il metodo `available()` restituisce il numero di byte che attendono di essere letti sulla porta seriale. Se è disponibile almeno un dato lo si può leggere utilizzando il metodo `Serial.read();` questo metodo restituisce il primo byte dei dati in input, se presente, altrimenti restituisce il valore -1.

LED alla moda

I prodotti di alta tecnologia da indossare come vestiti erano piuttosto popolari fino a qualche anno fa; le t-shirt corredate di equalizzatore grafico perfettamente funzionante sono ancora intriganti ma non sono più considerate strabilianti (<http://www.thinkgeek.com/tshirts-apparel/interactive/8a5b/>). D'altra parte, il semplice utilizzo di alcuni led permette di realizzare accessori femminili sorprendenti; per esempio, i giapponesi hanno creato ciglia a led (http://blog.makezine.com/archive/2009/10/led_eyelashes.html). Questo prodotto molto particolare non utilizza Arduino, anche se con LilyPad

(<http://www.arduino.cc/en/Main/ArduinoBoardLilyPad>) potete creare oggetti molto simili a questo. Dovete però prestare una certa attenzione, dato che la maggior parte dei led è molto luminosa e può danneggiare pericolosamente gli occhi!

Se il byte letto dal programma corrisponde al carattere 1 si accende il led e si trasmette sulla porta seriale il messaggio “LED on”. Il metodo `Serial.println()` aggiunge al messaggio di testo un carattere di “ritorno carrello” (*carriage return*, codice ASCII 13) seguito da una carattere “nuova riga” (*newline*, codice ASCII 10).

Il led si spegne quando il programma riceve il carattere 2; se si riceve un carattere diverso da 1 e 2 il programma trasmette un messaggio che segnala la presenza di un comando che non riesce a comprendere. La funzione `Serial.print()` è identica a quella di `Serial.println()`, ma non include nel messaggio i caratteri carriage return e newline.

A questo punto si può verificare il funzionamento del programma. Compilate il progetto, caricatelo nella scheda Arduino e avviate lo strumento *Serial Monitor*. Potete anche collegare un led al pin 13, altrimenti controllate l'accensione e lo spegnimento del led di stato. A prima vista sembra che nulla stia succedendo; questo perché non avete ancora inviato un comando. Digitate 1 nella casella di testo e fate clic sul pulsante *Send*. Si devono verificare due eventi: il led si accende e compare il messaggio “LED on” nella finestra di *Serial Monitor*, come si può vedere nella Figura 2.4. Il funzionamento del led è proprio controllato dalla tastiera del computer!



Figura 2.4 La funzione Serial Monitor dell'IDE Arduino.

Provate più volte a inviare il comando 1 oppure 2 e osservate anche ciò che succede quando inviate un comando sconosciuto. Se digitate per esempio una A maiuscola dovete ricevere il messaggio “Unknown command: 65”. Il numero 65 corrisponde al codice ASCII della lettera A, e Arduino rappresenta il dato in output nella sua forma più diretta.

Queste operazioni corrispondono al comportamento di default del metodo `print()` della classe `Serial`, che potete modificare impostando un indicatore di formato nella chiamata della funzione. Per verificare le opzioni disponibili provate a sostituire la riga 20 con queste istruzioni:

```
Serial.println(command, DEC);  
Serial.println(command, HEX);  
Serial.println(command, OCT);  
Serial.println(command, BIN);  
Serial.println(command, BYTE);
```

L’invio della lettera A produce un output simile al seguente:

```
Unknown command: 65  
41  
101  
1000001  
A
```

L’impostazione dell’indicatore di formato determina il tipo di conversione che `Serial.println()` effettua per rappresentare il byte di dati. L’opzione `DEC` visualizza il byte come numero decimale, `HEX` lo converte in numero esadecimale e così via. È interessante osservare che questa conversione modifica in genere la lunghezza dei dati trasmessi; per esempio, la rappresentazione binaria del singolo byte 65 richiede la trasmissione di 7 byte, in quanto contiene sette caratteri.

Sistemi di numerazione

Il nostro sistema di numerazione quotidiano ha base 10 per una ragione evoluzionistica; è probabile che se avessimo quattro dita per ogni mano conteremmo in base 8 e avremmo inventato i computer molti secoli prima.

Da migliaia di anni siamo abituati a utilizzare sistemi di numerazione posizionali e rappresentiamo numeri quali 4711 in base alla convenzione

indicata di seguito:

$$4 \times 10^3 + 7 \times 10^2 + 1 \times 10^1 + 1 \times 10^0$$

Questa notazione semplifica le operazioni aritmetiche. L'elaborazione di dati con il computer, in grado di interpretare solo numeri binari, risulta però molto più efficiente quando si adotta un sistema di numerazione in base 2 (binario), in base 8 (ottale) oppure in base 16 (esadecimale).

Si prenda per esempio di nuovo il numero decimale 4711; di seguito è riportata la rappresentazione di questo valore nel sistema di numerazione ottale e in quello esadecimale:

- $1 \times 8^4 + 1 \times 8^3 + 1 \times 8^2 + 4 \times 8^1 + 7 \times 8^0 = 011147$
- $1 \times 16^3 + 2 \times 16^2 + 6 \times 16^1 + 7 \times 16^0 = 0x1267$

Nei programmi Arduino potete esprimere valori utilizzando il sistema di numerazione che preferite, come negli esempi che seguono:

```
int decimal = 4711;
int binary = B1001001100111;
int octal = 011147
int hexadecimal = 0x1267;
```

I numeri binari iniziano con il carattere B, i numeri ottali iniziano con uno 0, mentre i numeri esadecimali iniziano con 0x.

Utilizzare altri sistemi di monitoraggio delle comunicazioni seriali

Nelle applicazioni più semplici è sufficiente utilizzare la funzione *Serial Monitor* dell'IDE Arduino, ma questa non può essere associata facilmente ad altre applicazioni di monitoraggio del sistema ed è carente in alcune funzionalità di controllo; per esempio, le versioni dell'IDE meno recenti non permettono di inviare caratteri newline. Ciò significa che dovete avere a disposizione una soluzione alternativa da impiegare come terminale di comunicazione seriale per inviare dati, alternativa che dipende dal sistema operativo con cui state lavorando.

Sistemi di monitoraggio per Windows

Putty (<http://www.chiark.greenend.org.uk/~sgtatham/putty/>) è un'ottima soluzione per chi utilizza Windows. È un'applicazione gratuita ed è costituita da un file eseguibile che non richiede alcuna installazione. La Figura 2.5 mostra la finestra di configurazione della comunicazione relativa a una porta seriale.



Figura 2.5 Configurazione di Putty per utilizzare questo sistema di monitoraggio con Arduino.

Potete avviare una comunicazione seriale con Arduino immediatamente dopo aver configurato il funzionamento di Putty. La Figura 2.6 mostra la finestra di dialogo che gestisce le comunicazioni. Fate clic su *Open* per visualizzare una finestra di terminale vuota.

A questo punto è sufficiente premere più volte 1 e 2 per accendere e spegnere il led. La Figura 2.7 mostra una tipica sessione di comunicazione seriale.

Sistemi di monitoraggio per Linux e Mac OS X

Gli utenti Linux e Mac possono utilizzare il comando `screen` per comunicare con Arduino attraverso la porta seriale. Verificate quale porta

seriale collega Arduino con il computer (per esempio tramite il menu IDE *Tools > Board*) ed eseguite un comando simile a quello riportato di seguito. Le schede più recenti possono indicare il nome della porta seriale con una indicazione analoga a `/dev/cu.usbserial-A9007LUY`, mentre nei sistemi Linux può essere necessario impostare una stringa simile a `/dev/ttyUSB1`:

```
$ screen /dev/cu.usbmodemfa141 9600
```

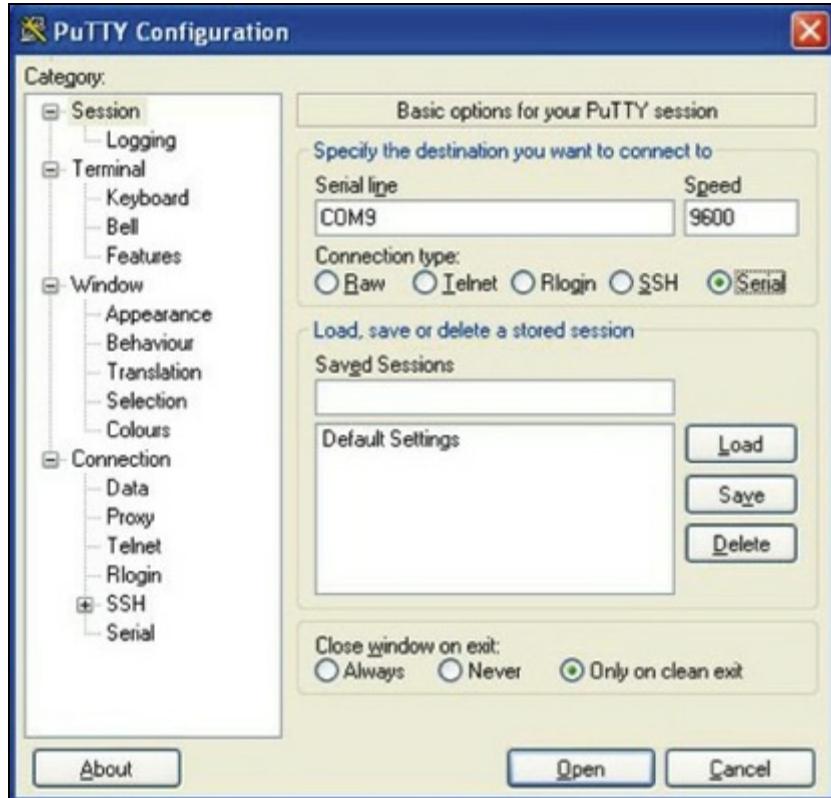


Figura 2.6 Apertura di una sessione di comunicazione seriale tra Arduino e Putty.



Figura 2.7 Putty sta comunicando con Arduino.

I parametri del comando `screen` definiscono il nome della porta seriale e il baud rate della trasmissione. La Figura 2.8 mostra una tipica sessione di

lavoro. Per interrompere l'esecuzione di screen è sufficiente premere Ctrl+A seguito da Ctrl+K.

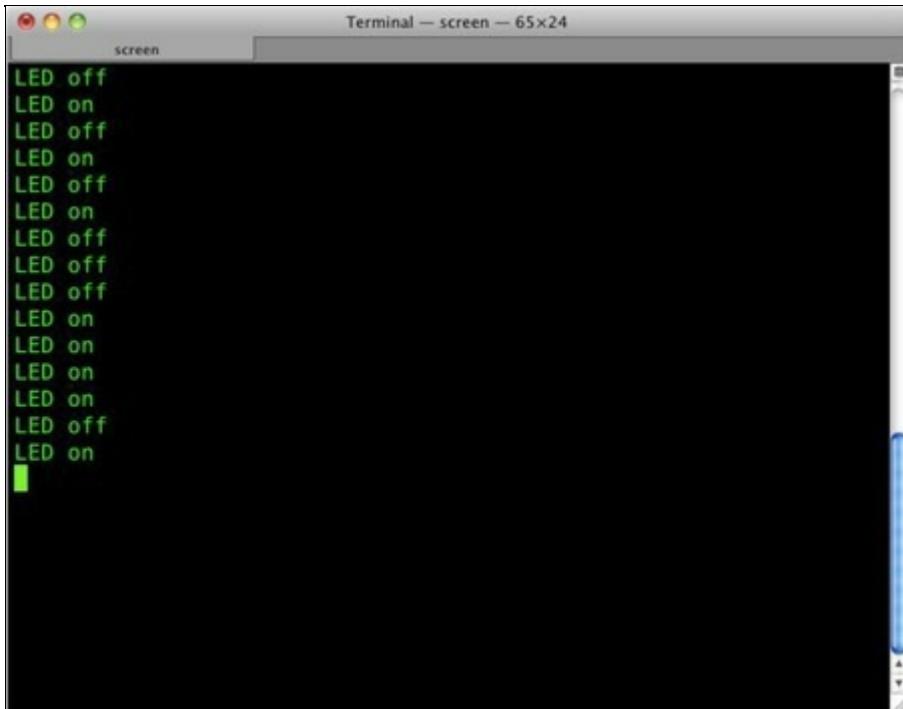


Figura 2.8 Il comando screen sta comunicando con Arduino.

A questo punto siete in grado di comunicare con Arduino; di conseguenza tutto ciò che è controllato da Arduino può essere anche controllato dal vostro computer e viceversa. L'accensione e lo spegnimento di led non sono controlli particolarmente sofisticati, ma possono servire per immaginare le potenzialità offerte dal sistema di gestione della comunicazione seriale. In effetti potete muovere sistemi robotizzati, automatizzare i servizi di casa oppure creare giochi altamente interattivi.

Di seguito sono riportate alcune interessanti osservazioni legate alla comunicazione seriale dei dati.

- La porta seriale di Arduino riceve buffer di dati che possono contenere fino a 128 byte. La trasmissione di una quantità maggiore di dati ad alta frequenza richiede di sincronizzare il sistema di trasmissione e di ricezione per evitare la perdita di dati. In genere il ricevitore invia al trasmettitore un segnale di “acknowledgment” per annunciare che è pronto a gestire un nuovo invio di dati.

Rendere più eccitanti i progetti con i led

Quanto esposto nei paragrafi precedenti porta a pensare che i led siano componenti utili ma non particolarmente eccitanti. Possono certamente essere impiegati per segnalare lo stato di funzionamento di un

dispositivo o perfino per realizzare lo schermo di un televisore, ma queste sono applicazioni note a tutti. In effetti i led sono componenti fondamentali di progetti molto spettacolari, tra i quali si vuole ricordare per esempio Bedazzler (<http://www.instructables.com/id/Bedazzler-DIY-non-lethal-weapony/>). Il progetto Bedazzler si occupa della realizzazione di un'arma non letale a led lampeggianti che provoca nausea, vertigini, mal di testa, cecità temporanea, dolore agli occhi e vomito. In origine aveva scopi prettamente militari, ma al momento è diventato un progetto open source (<http://www.ladyada.net/make/bedazzler/>). A prescindere dall'ovvia curiosità a carattere scientifico, dovete sempre ricordare che Bedazzler è un'arma pericolosa, che non deve mai essere considerata alla stregua di un giocattolo e non va mai rivolta contro persone o animali.

- La comunicazione seriale può controllare il funzionamento di più dispositivi, ma le schede Arduino mettono a disposizione in genere una sola porta seriale. Se dovete predisporre un numero maggiore di porte prendete in considerazione l'impiego di Arduino Mega 2560, una scheda che ne ha quattro (<http://arduino.cc/en/Main/ArduinoBoardMega2560>).
- Arduino ammette la presenza di un componente UART (*Universal Asynchronous Receiver/Transmitter*, <http://en.wikipedia.org/wiki/UART> o <http://it.wikipedia.org/wiki/UART>) per gestire le comunicazioni seriali. Questo componente integrato è in grado di svolgere le operazioni di trasmissione/ricezione liberando risorse della CPU della scheda di controllo, che può in questo modo elaborare altre forme di dati. Una soluzione di questo genere migliora sensibilmente le prestazioni del sistema complessivo. Il componente UART utilizza i pin 0 (RX) e 1 (TX); ciò significa che questi pin della scheda non possono essere impiegati per scopi diversi dalla comunicazione della porta seriale. Se dovete sfruttarli potete sempre disattivare la comunicazione seriale impostando un comando `Serial.end()`.
- La libreria SoftwareSerial (<http://www.arduino.cc/en/Reference/SoftwareSerial>) permette di utilizzare i pin digitali per la comunicazione seriale. Ricordate però che questa soluzione presenta limitazioni significative in merito alla velocità e affidabilità delle trasmissioni, oltre a non supportare tutte le funzioni offerte da una porta seriale vera e propria.

In questo capitolo avete visto come comunicare con Arduino tramite la porta seriale, una tecnica che spalanca le porte a innovative soluzioni di physical computing. I dettagli della comunicazione seriale verranno ripresi nell'Appendice C. Nei prossimi capitoli si studierà la raccolta delle informazioni tramite l'uso di sensori e si imparerà a modificare l'ambiente reale muovendo una serie di oggetti. La comunicazione seriale costituisce la

base di partenza che permette di controllare questi eventi impiegando Arduino e il vostro computer.

Cosa fare se non funziona?

Se si manifestano problemi di funzionamento nell'esecuzione degli esempi di questo capitolo conviene innanzitutto fare riferimento alle indicazioni fornite nel Capitolo 1. Se i problemi persistono, controllate il funzionamento della comunicazione seriale; per esempio, potreste aver impostato non correttamente il baud rate; la Figura 2.9 mostra ciò che può succedere in un caso del genere.

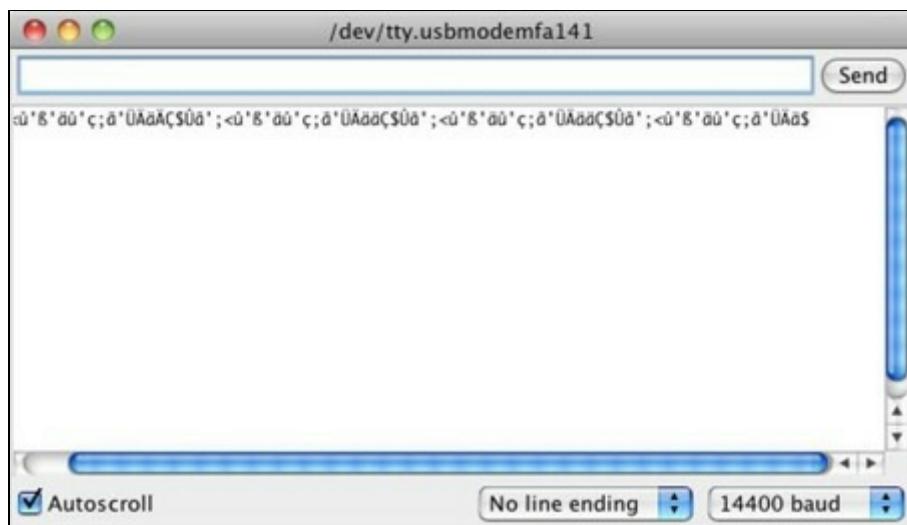


Figura 2.9 Un baud rate non corretto genera un mucchio di dati senza senso.

Verificate che il baud rate impostato nella chiamata della funzione `Serial.begin()` coincida con quello configurato in *Serial Monitor*.

Esercizi

- Aggiungete nuovi comandi al programma di questo capitolo; per esempio, il comando 3 potrebbe far lampeggiare il led per un certo intervallo di tempo.
- Provate a rendere più comprensibili i comandi del programma; per esempio, al posto di 1 impostate il comando on, mentre al posto di 2 configurate il comando off.

Se avete problemi nel risolvere questi esercizi consultate le indicazioni fornite nel Capitolo 4 a proposito delle librerie di programma.

Parte II

Otto progetti Arduino

Capitolo 3 Il dado binario

Capitolo 4 Libreria per la generazione di un codice Morse

Capitolo 5 Misurare con i sensori il mondo che ci circonda

Capitolo 6 Game controller sensibile al movimento

Capitolo 7 Giocherellare con il controller Wii Nunchuk

Capitolo 8 Networking con Arduino

Capitolo 9 Telecomando universale

Capitolo 10 Controllo dei motori con Arduino

Il dado binario

La conoscenza dei fondamenti dell’ambiente di sviluppo Arduino rende sempre più intrigante lo studio di nuovi progetti. Ora avete la possibilità di creare le prime applicazioni complesse e standalone. Al termine di questo capitolo saprete come utilizzare led, pulsanti, breadboard e resistori. La combinazione tra questi componenti e la scheda Arduino offre potenzialità pressoché infinite per la realizzazione di progetti innovativi e molto interessanti.

Il primo progetto riguarda la creazione di un dado binario. Un dado visualizza il proprio valore mostrando una faccia dove sono riprodotti da uno a sei punti colorati mentre il dado di questo progetto rappresenta il proprio valore tramite una serie di led. Nel primo esempio dei capitoli precedenti è stato sufficiente impiegare un solo componente, ma il dado richiede la presenza di più led, da collegare esternamente alla scheda Arduino.

Dato che i componenti esterni non possono essere collegati direttamente alla scheda, dovete cablare il circuito esterno su una breadboard. Per collegare pulsanti e led alla scheda Arduino dovete inoltre impiegare un altro componente elettronico fondamentale: il resistore. Alla fine di questo capitolo saprete pertanto maneggiare molti utili strumenti di lavoro.

Cosa serve

1. Una breadboard di piccole dimensioni.
2. Tre led (per svolgere gli esercizi in fondo al capitolo è necessario avere a disposizione un numero maggiore di led).
3. Due resistori da $10\text{k}\Omega$ (si veda l’Appendice A per saperne di più sui resistori).
4. Tre resistori da $1\text{k}\Omega$.
5. Due pulsanti.
6. Cavi di collegamento per il cablaggio su breadboard.
7. Una scheda Arduino, per esempio un modello Arduino Uno, Duemilanove o Diecimila.

8. Un cavo USB per collegare la scheda Arduino al computer.
9. Un tilt sensor (facoltativo).

La Figura 3.1 mostra i componenti necessari per realizzare il progetto illustrato in questo capitolo. Foto simili a questa verranno incluse nella maggior parte dei prossimi capitoli; i numeri visibili nelle foto corrispondono al numero riportato nell'elenco componenti. Le foto non riportano invece altri componenti standard dei progetti, per esempio la scheda Arduino e il cavo USB.



Figura 3.1 I componenti necessari per realizzare il progetto di questo capitolo.

Cablaggio di circuiti con le breadboard

Il collegamento diretto tra led e scheda Arduino è una soluzione da adottare solo nei progetti più semplici. In genere il prototipo di un progetto è cablato su una breadboard da connettere alla scheda di controllo del microcontrollore. Una breadboard “emula” i collegamenti di un circuito stampato e non richiede la saldatura di componenti; dovete semplicemente inserire i componenti nei fori predisposti sulla breadboard.

Le breadboard sono disponibili in diverse forme e dimensioni, come si può vedere per esempio nella Figura 3.2, ma il loro utilizzo rimane sostanzialmente identico. In ogni caso sono presenti più fori nei quali

inserire i terminali dei componenti oppure i cavi di collegamento; l'inserimento nei fori definisce il cablaggio del circuito tenendo presente la connessione interna tra i fori della breadboard. La Figura 3.3 mostra le piste che collegano internamente i fori di una breadboard.

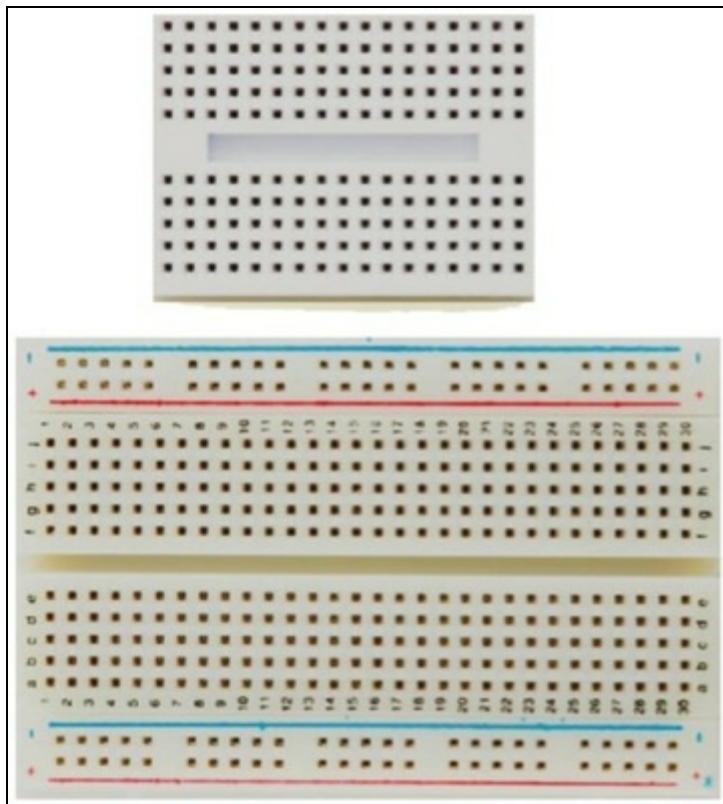


Figura 3.2 Esempi di breadboard.

La maggior parte dei fori costituisce una colonna di collegamenti. La breadboard più grande visibile nella foto mostra anche la presenza di quattro righe orizzontali di fori collegati tra loro; queste connessioni interne sono particolarmente utili quando si devono cablare circuiti con molti componenti e in genere si collega una riga orizzontale al morsetto positivo di alimentazione e un'altra riga alla massa del circuito. In questo modo si distribuisce l'alimentazione elettrica lungo tutta la lunghezza della breadboard. A questo punto si può provare a collegare i componenti su una breadboard.

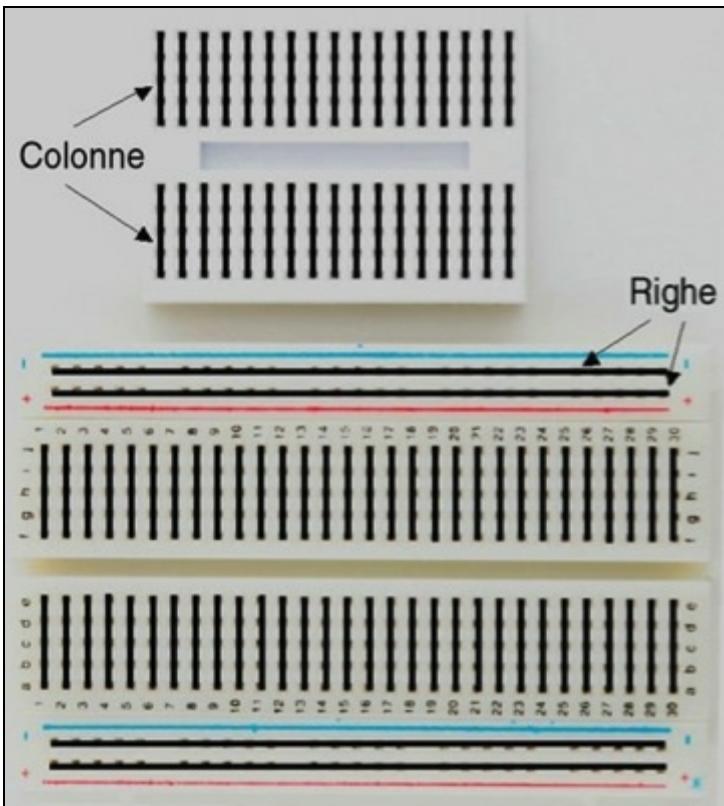


Figura 3.3 Collegamenti interni alla breadboard.

Collegare un led su breadboard

I led impiegati finora erano installati sulla scheda Arduino e un solo led esterno è stato collegato direttamente ai morsetti della scheda di controllo. In questo paragrafo verrà invece inserito un led su breadboard e questo verrà successivamente connesso alla scheda tramite la stessa breadboard.

La Figura 3.4 mostra l’immagine del circuito completo e vi si può riconoscere la scheda Arduino, la breadboard, un led, tre cavi di collegamento e un resistore da $1k\Omega$; l’intero cablaggio richiede pochi minuti di lavoro. Collegate Arduino alla breadboard tramite due cavi; per esempio, collegate il pin 12 con la nona colonna della breadboard e il pin di massa con la decima colonna di fori. In questo modo collegate automaticamente il pin 12 con tutti i fori della nona colonna e il pin di massa con i fori della colonna 10. La scelta delle colonne di fori è assolutamente arbitraria.

Inserite il terminale negativo del led (quello più corto) nella colonna 10 e il terminale positivo nella colonna 9. Spingete con una certa decisione il terminale nel foro per farlo scivolare nella posizione più stabile. Provate a effettuare l’inserimento in punti diversi della breadboard; a volte può risultare comodo accorciare i terminali dei componenti prima di inserirli nella breadboard. Verificate che si riesca comunque a identificare il

connettore negativo e positivo anche dopo averli accorciati. Potete per esempio accorciare il terminale negativo in modo che risulti più corto di quello positivo. Ricordate anche di indossare occhiali protettivi quando tagliate i terminali dei componenti elettronici.

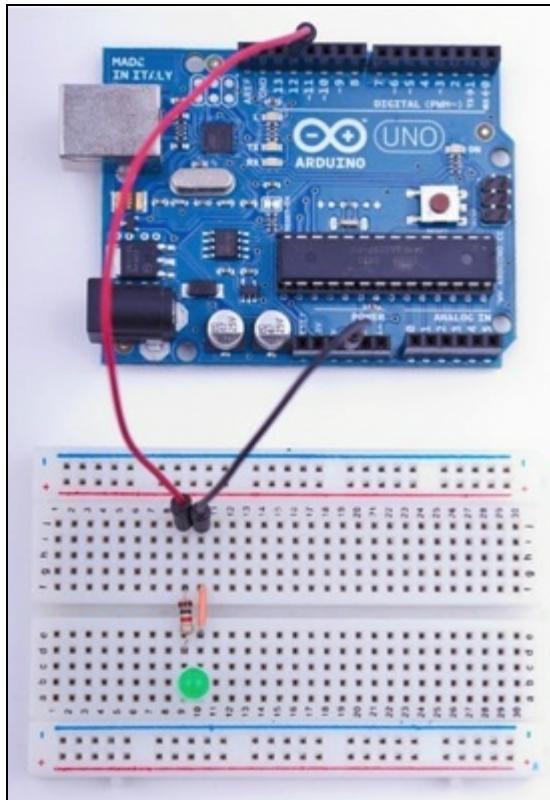


Figura 3.4 Collegamento tra un led su breadboard e la scheda Arduino.

Le operazioni svolte finora sono abbastanza semplici. In sintesi, avete semplicemente aumentato le possibilità di collegare componenti esterni al pin di massa e al pin 12 di IO della scheda. Perché occorre aggiungere un resistore e, soprattutto, cos'è un resistore? Si tratta di un componente che limita la quantità di corrente che scorre lungo una connessione elettrica. In questo progetto il resistore protegge il led limitando la potenza elettrica sul componente, per evitare che venga distrutto. Dovete sempre utilizzare un resistore di limitazione quando collegate un led esterno! Nell'Appendice A verranno fornite altre indicazioni sull'uso dei resistori e sul codice a colori che ne permette l'identificazione. Nella Figura 3.5 potete vedere un resistore i cui terminali sono predisposti in modi differenti, ovvero in posizione normale, piegati e accorciati.

È lecito chiedersi perché non sia stato utilizzato un resistore nel collegamento diretto tra led e scheda Arduino. La risposta è semplice: il pin 13 presenta un resistore interno di limitazione da $1k\Omega$. In questo progetto si

utilizza invece il pin 12, che richiede l'inserimento dall'esterno di un resistore di limitazione della corrente.



Figura 3.5 Disposizioni diverse di un resistore.

Non perdiamo altro tempo nella descrizione dei collegamenti di cablaggio: è sufficiente fare riferimento all'immagine mostrata nella Figura 3.6 per comprendere come realizzare il circuito corretto. In questa immagine sono state utilizzate entrambe le colonne di fori affacciati sulla breadboard; da una parte un terzo cavo molto corto ha connesso le due estremità delle colonne, dall'altra parte il resistore è stato cablato in modo da realizzare un ponte tra le due colonne adiacenti.

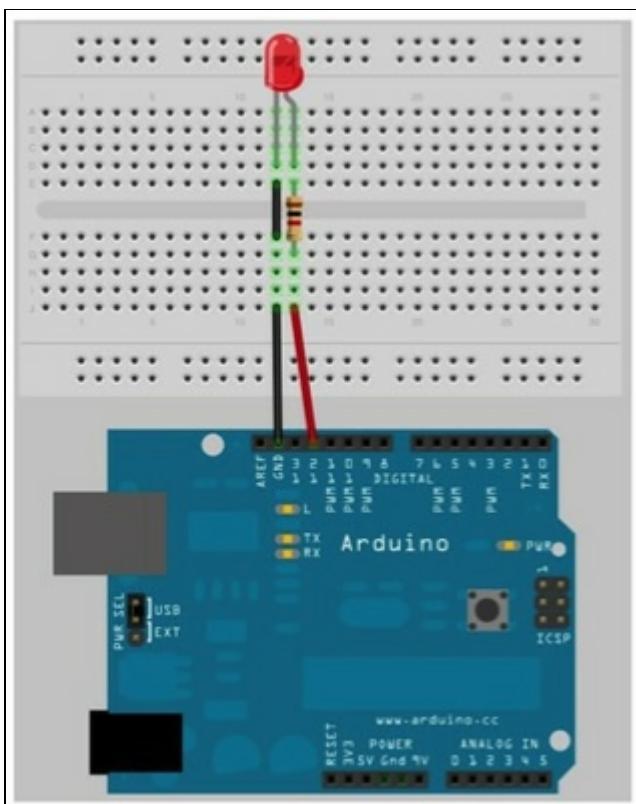


Figura 3.6 Potete utilizzare l'intera estensione di una breadboard.

Prima versione del dado binario

Tutti sanno che il valore di un dado è espresso da un numero compreso tra 1 e 6. Per emulare un dado di questo genere con un dispositivo elettronico sarebbe necessario impiegare sei led e predisporre un circuito piuttosto complesso. Vale la pena allora semplificare almeno in prima battuta la soluzione e visualizzare il valore del dado in forma binaria.

Un dado binario richiede solo tre led per indicare il valore decimale da 1 a 6. Questo valore è convertito nel sistema di numerazione binario e ogni bit di valore 1 corrisponde a un led acceso. Di seguito è riportato uno schema che mostra la corrispondenza tra valore del dado e sua rappresentazione binaria; un triangolo in nero identifica un led acceso.

	=			=	
	=			=	
	=			=	

Sapete già come controllare l'accensione di un singolo led collegato alla breadboard. Il controllo di tre led è abbastanza simile e richiede solo qualche collegamento in più, i led, alcuni resistori da $1k\Omega$ e i pin della scheda di controllo. La Figura 3.7 mostra una prima versione funzionante del dado binario.

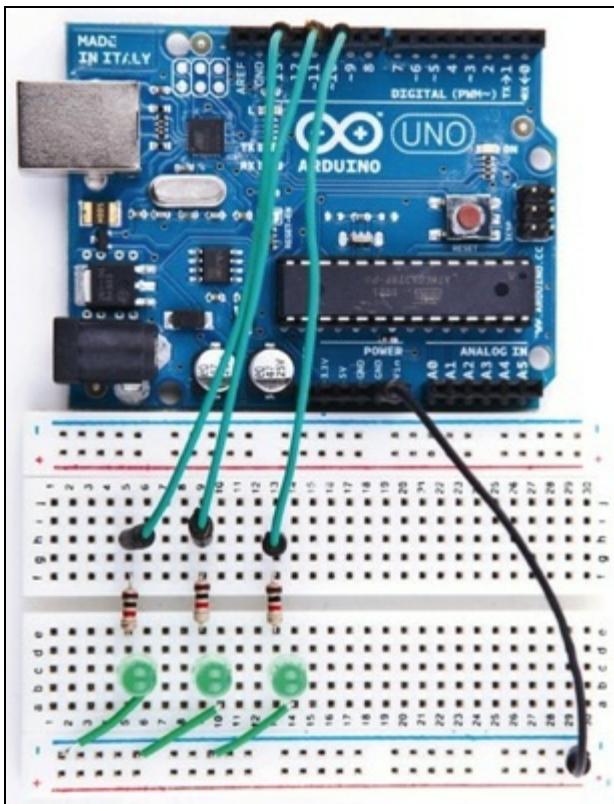


Figura 3.7 Prima versione funzionante del dado binario.

La differenza più significativa tra questo circuito e i precedenti riguarda la connessione a massa. Un singolo led da collegare a massa può essere cablato inserendo direttamente il terminale negativo a massa, mentre nel caso di tre led da collegare a massa conviene utilizzare le righe orizzontali della breadboard. Collegate la riga contrassegnata da un trattino (-) al pin di massa della scheda Arduino; in questo modo tutti i fori della riga sono connessi al pin di massa. Potete collegare a questo punto i fori della riga ai led utilizzando fili molto corti.

Il cablaggio del circuito non presenta novità particolari, in quanto si tratta semplicemente di replicare il collegamento di un led illustrato nel paragrafo precedente. Tenete presente che i tre led risultano connessi ai pin 10, 11 e 12. A questo punto potete digitare le istruzioni del programma qui riportate.

[BinaryDice/BinaryDice.pde](#)

```
Riga 1  const unsigned int LED_BIT0 = 12;
-  const unsigned int LED_BIT1 = 11;
-  const unsigned int LED_BIT2 = 10;
-
5  void setup() {
-      pinMode(LED_BIT0, OUTPUT);
-      pinMode(LED_BIT1, OUTPUT);
-      pinMode(LED_BIT2, OUTPUT);
```

```

10    randomSeed(analogRead(A0));
11    long result = random(1, 7);
12    output_result(result);
13 }

14
15 void loop() {
16 }

17
18 void output_result(const long result) {
19     digitalWrite(LED_BIT0, result & B001);
20     digitalWrite(LED_BIT1, result & B010);
21     digitalWrite(LED_BIT2, result & B100);
22 }

```

Più led, lettori di dadi e cubi a led

La realizzazione del dado binario è interessante ma costituisce comunque un progetto decisamente semplice, adatto per chi è alle prime armi. Cosa si può dire del progetto contrario, ovvero di un progetto che riproduca un dado reale? Steve Hoefer (<http://grathio.com/cgi-bin/mt5/mt-search.cgi?search=dice&IncludeBlogs=1&limit=20>) ha per esempio realizzato un lettore di dadi piuttosto impressionante.

Il progetto utilizza cinque coppie di emettitori e ricevitori a infrarossi per “misurare” la superficie della faccia di un dado. Si tratta di un progetto molto avanzato, interessante da studiare per apprendere molte informazioni utili.

Un altro progetto degno di attenzione è rappresentato dal cubo a led, le cui facce mostrano diverse serie di led (<http://arduino.cc/blog/2010/04/14/cubeduino/>). In questo caso il controllo di un numero così grande di led diventa un compito abbastanza complesso, ma il risultato è veramente sbalorditivo.

Queste sono le istruzioni richieste per implementare la prima versione del dado binario. Come nei progetti precedenti, anche in questo caso il software definisce alcune costanti relative ai pin di output ove sono collegati i led. La funzione `setup()` imposta tutti i pin in modalità `OUTPUT`. Il dado binario richiede la generazione di numeri casuali compresi tra 1 e 6. La funzione `random()` restituisce numeri casuali che appartengono a un determinato intervallo tramite un generatore di numeri pseudocasuali. Alla riga 10 si inizializza il generatore con un valore di rumore prelevato dal pin A0 di input analogico (si veda il prossimo riquadro per maggiori dettagli sulla generazione di numeri pseudocasuali). Fin dalla versione 19 l’IDE Arduino definisce costanti che corrispondono ai pin analogici e denominate `A0`, `A1` e così via. A questo punto il programma genera un nuovo numero casuale utilizzando la funzione `output_result()`. Il parametro 7 è corretto in quanto la funzione si aspetta l’indicazione del limite superiore dell’intervallo aumentato di uno.

La funzione `output_result()` accetta un valore numerico e restituisce i tre bit meno significativi del valore attivando in corrispondenza i tre led di

output. Per stabilire la corrispondenza si utilizza l'operatore `&` e una serie di valori binari. L'operatore `&` prende in considerazione due valori numerici e li confronta un bit alla volta. Se due bit che si trovano nella stessa posizione valgono 1 anche il risultato del confronto vale 1. Il prefisso `B` permette di impostare direttamente valori binari nel codice sorgente; per esempio, `B11` equivale in binario a indicare il valore decimale `3`.

La funzione `loop()` è vuota e il funzionamento del programma può essere descritto come segue: ogni volta che si avvia questo progetto sulla scheda Arduino viene visualizzato un nuovo numero; per lanciare di nuovo il dado occorre premere il pulsante *Reset*.

Generatore di numeri casuali

Alcuni problemi di elaborazione dati sono più complessi di altri e la generazione di numeri casuali appartiene sicuramente alla prima categoria. Dopotutto, va ricordato che una delle proprietà fondamentali dei computer è il comportamento deterministico, ovvero non casuale, dei risultati. Nonostante questo, spesso è necessario ottenere proprio un comportamento casuale (almeno in apparenza) dei risultati elaborati da un computer, allo scopo di esaudire scopi anche molto diversi tra loro, che vanno dall'impostazione di giochi all'implementazione di complessi algoritmi di crittografia.

Una delle tecniche più diffuse, adottata per esempio dalla funzione `random()`, consiste proprio nella creazione pseudocasuale di numeri (http://en.wikipedia.org/wiki/Pseudo-random_numbers) o

http://it.wikipedia.org/wiki/Numeri_pseudo-casuali). La generazione sembra casuale anche se deriva dal calcolo di una formula matematica ben precisa. Esistono diversi algoritmi di calcolo, ma in genere ogni nuovo numero pseudocasuale è calcolato a partire dal numero ottenuto da un calcolo precedente; ciò implica che dovete impostare un valore di inizializzazione o *seme casuale* per creare il primo numero pseudocasuale della sequenza. Potete creare sequenze diverse di numeri pseudocasuali semplicemente impostando semi casuali differenti tra loro.

La generazione di numeri pseudocasuali è un'operazione poco dispendiosa, ma è sufficiente conoscere l'algoritmo di calcolo e il seme casuale per sapere in anticipo tutti i numeri della sequenza. Questa forma di generazione non può pertanto essere impiegata nei sistemi di crittografia.

In Natura è possibile osservare infiniti processi casuali che possono essere misurati facilmente con la scheda Arduino per creare numeri realmente casuali. Spesso è sufficiente misurare il rumore elettrico presente sul pin analogico 0 e impostare il valore di questa misura come seme casuale della funzione `randomSeed()`.

Potete anche utilizzare questa forma di rumore per generare numeri completamente casuali; avete perfino a disposizione una libreria di programmi che esegue questo genere di operazioni (<http://code.google.com/p/tinkerit/wiki/TrueRandom>).

Arduino è ideale quando si tratta di realizzare un dispositivo hardware per la generazione di numeri altamente casuali. Potete trovare molti progetti che misurano processi naturali per questo unico scopo; uno dei più conosciuti utilizza per esempio una clessidra collegata a una scheda Arduino (<http://www.circuitlake.com/usb-hourglass-sand-timer.html>).

Compilate il codice, caricatelo su Arduino e provate a lanciare più volte il dado binario. Avete appena realizzato il vostro primo progetto elettronico avanzato!

Per ottenere un nuovo risultato dovete semplicemente premere il pulsante *Reset* della scheda di controllo. Questa soluzione è molto drastica e per il momento può essere accettata come semplice prototipo iniziale del dado

binario. Spesso è necessario predisporre più di un pulsante e ad ogni modo è più elegante fare in modo che il pulsante da utilizzare sia esterno alla scheda. Le modifiche da apportare a questo proposito saranno descritte nel prossimo paragrafo.

Lavorare con i pulsanti

In questo paragrafo verrà aggiunto un pulsante esterno al dado binario, per cui non si dovrà più ricorrere al pulsante *Reset* di Arduino per lanciarlo. Conviene però iniziare a conoscere i pulsanti collegando un circuito che controlla l'accensione di un led da pulsante esterno.

Innanzitutto è lecito chiedersi cosa sia un pulsante. Di seguito sono mostrate tre viste differenti di un pulsante in miniatura che può essere impiegato al posto del pulsante *Reset* di Arduino.



Il pulsante presenta quattro connettori che si possono inserire senza fatica su una breadboard (al massimo dovete sforzare i connettori utilizzando delicatamente ma in modo deciso un paio di pinzette da orologiaio). Due terminali opposti sono collegati elettricamente tra loro quando premete il pulsante, altrimenti la connessione è aperta.

La Figura 3.8 mostra un semplice circuito che utilizza un pulsante. Collegate il pin 7 (la scelta è arbitraria) al pulsante, poi collegate a massa il pulsante passando attraverso un resistore da $10k\Omega$. A questo punto collegate l'alimentazione a 5V della scheda all'altro terminale del pulsante.

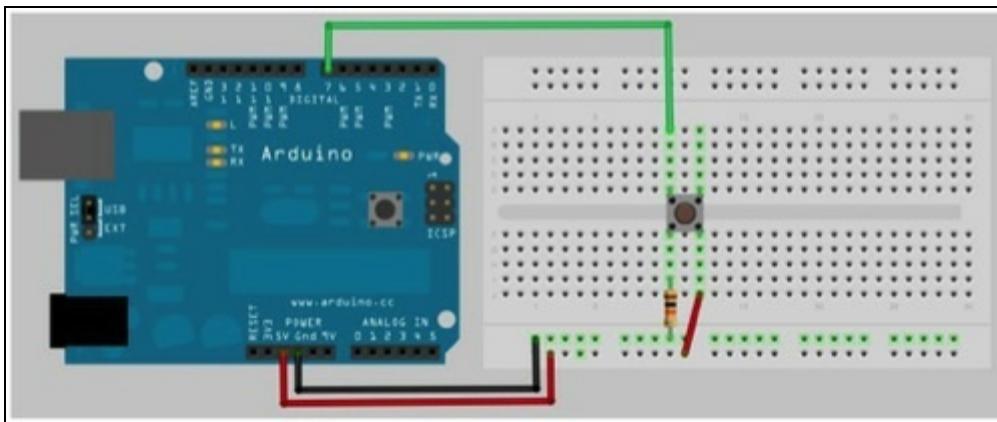


Figura 3.8 Un semplice circuito per il cablaggio di un pulsante.

Anche in questo caso il collegamento è abbastanza semplice, ma a cosa serve il resistore? Occorre considerare che quando il pulsante non viene premuto la scheda si aspetta di “misurare” un valore di default (`LOW`). Il pulsante rilasciato non è collegato direttamente a massa e pertanto la scheda può rilevare elettricità statica e interferenze di altra natura. Nel circuito descritto in precedenza, invece, scorre una piccola corrente nel resistore verso massa e questa corrente esclude che un eventuale rumore elettrico possa alterare la tensione misurata sul pin di input della scheda.

La pressione del pulsante riporta i 5V di alimentazione sul pin digitale di Arduino, mentre a pulsante rilasciato sul pin di input è presente un collegamento a massa. Questa configurazione prende il nome di *resistore pull-down*; un *resistore pull-up* funziona esattamente in modo opposto. In altre parole, per questo progetto dovete collegare il pin di segnale direttamente al morsetto positivo dell’alimentazione e al pulsante, mentre l’altro pin del pulsante va collegato a massa attraverso un resistore.

Dopo aver eliminato le fluttuazioni di segnale dovute a collegamenti non corretti del pulsante si può tornare al software. Di seguito è riportato il programma che controlla la pressione del pulsante e accende di conseguenza un led.

[BinaryDice/SimpleButton/SimpleButton.pde](#)

```

const unsigned int BUTTON_PIN = 7;
const unsigned int LED_PIN = 13;

void setup() {
    pinMode(LED_PIN, OUTPUT);
    pinMode(BUTTON_PIN, INPUT);
}

```

```

void loop() {
    const int BUTTON_STATE = digitalRead(BUTTON_PIN);

    if (BUTTON_STATE == HIGH)
        digitalWrite(LED_PIN, HIGH);
    else
        digitalWrite(LED_PIN, LOW);
}

```

Collegate il pulsante al pin 7 e il led al pin 13, poi inizializzate i pin in base alle indicazioni della funzione `setup()`. La funzione `loop()` rileva lo stato attuale del pin collegato al pulsante: se il pin è `HIGH` si accende il led, altrimenti il led rimane spento.

Caricate il programma su Arduino e osservate che il led si accende ogni volta che tenete premuto il pulsante, mentre si spegne non appena lo rilasciate. Tutto sembra pronto per controllare il funzionamento del dado binario tramite il pulsante. Prima di proseguire conviene però perfezionare il progetto e trasformare il pulsante in un vero interruttore della luce.

Si può iniziare con la soluzione più semplice. Non modificate il circuito esterno e caricate su Arduino il programma qui riportato.

[BinaryDice/UnreliableSwitch/UnreliableSwitch.pde](#)

```

Riga 1  const unsigned int BUTTON_PIN = 7;
      - const unsigned int LED_PIN = 13;
      -
      -
      - void setup() {
5       pinMode(LED_PIN, OUTPUT);
      -     pinMode(BUTTON_PIN, INPUT);
      -   }
      -
      - int led_state = LOW;
10
      - void loop() {
      -     const int CURRENT_BUTTON_STATE = digitalRead(BUTTON_PIN);
      -
      -     if (CURRENT_BUTTON_STATE == HIGH) {
15         led_state = (led_state == LOW) ? HIGH : LOW;
      -     digitalWrite(LED_PIN, led_state);
      -   }
      - }

```

Le prime istruzioni inizializzano le costanti relative ai pin della scheda, mentre la funzione `setup()` imposta le modalità di utilizzo dei pin. Alla riga 9 si definisce la variabile globale `led_state` per memorizzare lo stato attuale del led, che deve risultare `LOW` quando il led è acceso e `HIGH` quando è spento. La funzione `loop()` verifica lo stato attuale del pulsante. Se premete il pulsante lo stato diventa `HIGH` e il programma commuta il contenuto di `led_state`. In altre parole, se `led_state` era `HIGH` ora viene impostato con il valore `LOW` e viceversa. Alla fine del programma si impone lo stato fisico del led in modo che coincida con lo stato corrente previsto dal software.

La soluzione è semplice ma purtroppo non funziona. Provate a eseguirla più volte di seguito e noterete che sorgono comportamenti imprevisti e quasi indecifrabili.

Se premete il pulsante, per esempio, il led si accende e poi si spegne immediatamente. Anche quando rilasciate il pulsante lo stato del led può diventare arbitrario; in altre parole, a volte rimane acceso e altre volte si spegne.

Il problema è dovuto al fatto che Arduino esegue il metodo `loop()` con continuità. La CPU di Arduino è abbastanza lenta ma l'esecuzione del metodo si ripete comunque più volte, a prescindere dalla pressione o meno del pulsante. Quando premete il pulsante e lo tenete premuto anche solo per qualche istante, il suo stato rimane `HIGH` per un tempo sufficiente a provocare più volte la commutazione dello stato del led (la commutazione avviene in modo così rapido da far apparire il led sempre acceso). Quando rilasciate il pulsante il led si può pertanto trovare in uno stato arbitrario.

La soluzione può essere perfezionata memorizzando non solo lo stato corrente del led ma anche lo stato precedente del pulsante.

[BinaryDice/MoreReliableSwitch/MoreReliableSwitch.pde](#)

```
const unsigned int BUTTON_PIN = 7;
const unsigned int LED_PIN = 13;

void setup() {
    pinMode(LED_PIN, OUTPUT);
    pinMode(BUTTON_PIN, INPUT);
}

int old_button_state = LOW;
```

```

int led_state = LOW;

void loop() {
    const int CURRENT_BUTTON_STATE = digitalRead(BUTTON_PIN);
    if (CURRENT_BUTTON_STATE != old_button_state &&
        CURRENT_BUTTON_STATE == HIGH)
    {
        led_state = (led_state == LOW) ? HIGH : LOW;
        digitalWrite(LED_PIN, led_state);
    }
    old_button_state = CURRENT_BUTTON_STATE;
}

```

Dopo aver inizializzato i pin relativi a pulsante e led il programma dichiara due nuove variabili: `old_button_state` memorizza lo stato precedente del pulsante, mentre `led_state` memorizza lo stato attuale del led. Entrambe queste variabili possono assumere il valore `HIGH` oppure `LOW`.

La funzione `loop()` deve anche in questo caso leggere lo stato corrente del pulsante; ora non si deve limitare a verificare se si trova in stato `HIGH`, deve anche verificare se lo stato corrente è cambiato rispetto all'ultima rilevazione effettuata. Solo quando entrambe le condizioni sono vere il programma commuta lo stato del led. Questa soluzione non accende e spegne più volte il led nell'intervallo di tempo in cui il pulsante rimane premuto. Alla fine del programma si memorizza lo stato corrente del pulsante nella variabile `old_button_state`.

Caricate la nuova versione del programma e osservate come questa soluzione sembra funzionare meglio della precedente. Si possono però verificare ancora situazioni limite nelle quali il pulsante non funziona esattamente come dovrebbe. Ora i problemi si manifestano quando rilasciate il pulsante.

Il malfunzionamento è dovuto al fatto che i pulsanti meccanici rimbalzano per alcuni millisecondi quando vengono premuti. La Figura 3.9 mostra il segnale prodotto in genere da un pulsante meccanico. Immediatamente dopo aver premuto il pulsante non si riscontra l'emissione di un segnale “pulito”: per risolvere il problema occorre aggiungere una funzione “antirimbalzo” del pulsante. Di solito è sufficiente attendere per un piccolo intervallo di tempo e lasciare che il contatto elettrico del pulsante si stabilizzi. La funzione antirimbalzo garantisce che il programma risponda in modo corretto alla pressione del pulsante. Anche in questo caso, oltre alla

funzione antirimbalzo è necessario memorizzare lo stato corrente del led in una variabile. Vediamo il programma che esegue queste operazioni.

BinaryDice/DebounceButton/DebounceButton.pde

```
Riga 1  const unsigned int BUTTON_PIN = 7;
-      const unsigned int LED_PIN = 13;
-
-      void setup() {
5       pinMode(LED_PIN, OUTPUT);
-      pinMode(BUTTON_PIN, INPUT);
-    }
-
-      int old_button_state = LOW;
10     int led_state = LOW;
-
-      void loop() {
-        const int CURRENT_BUTTON_STATE = digitalRead(BUTTON_PIN);
-        if (CURRENT_BUTTON_STATE != old_button_state &&
15          CURRENT_BUTTON_STATE == HIGH)
-        {
-          led_state = (led_state == LOW) ? HIGH : LOW;
-          digitalWrite(LED_PIN, led_state);
-          delay(50);
20        }
-        old_button_state = CURRENT_BUTTON_STATE;
-      }
```

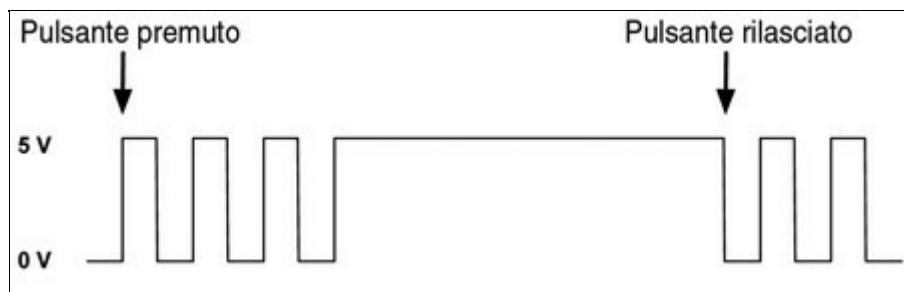


Figura 3.9 Rimbalzi del segnale elettrico prodotti da un pulsante.

La versione definitiva dell'interruttore a led è diversa da quella precedente per una sola riga di istruzioni: alla riga 19 si realizza l'antirimbalzo del pulsante con una pausa di 50 millisecondi che precede l'accesso al ciclo principale del programma.

Per il momento queste informazioni sono sufficienti per utilizzare i pulsanti con Arduino. Nel prossimo paragrafo si vedrà come impiegare due pulsanti per trasformare il dado binario in un divertente gioco.

Inserimento di un pulsante esterno

Il controllo del dado binario è stato finora impostato grazie a un uso improprio del pulsante *Reset*. La soluzione è poco efficiente ed è pertanto necessario aggiungere pulsanti esterni. Nella Figura 3.10 potete osservare le modifiche da apportare al circuito; in effetti, non dovete modificare i componenti già collegati al circuito, ma aggiungere nuovi componenti esterni. Collegate in primo luogo un pulsante sulla breadboard e collegate un suo terminale al pin 7, poi collegate il pulsante a massa attraverso un resistore da $10k\Omega$ e aggiungete un piccolo cavo per collegare questo resistore a 5V. Dopo aver sistemato l'hardware potete digitare le istruzioni software riportate di seguito.

[BinaryDice/DiceWithButton/DiceWithButton.pde](#)

```
const unsigned int LED_BIT0 = 12;
const unsigned int LED_BIT1 = 11;
const unsigned int LED_BIT2 = 10;
const unsigned int BUTTON_PIN = 7;

void setup() {
    pinMode(LED_BIT0, OUTPUT);
    pinMode(LED_BIT1, OUTPUT);
    pinMode(LED_BIT2, OUTPUT);
    pinMode(BUTTON_PIN, INPUT);
    randomSeed(analogRead(A0));
}

int current_value = 0;
int old_value = 0;

void loop() {
    current_value = digitalRead(BUTTON_PIN);
    if (current_value != old_value && current_value == HIGH) {
        output_result(random(1, 7));
        delay(50);
    }
    old_value = current_value;
}

void output_result(const long result) {
    digitalWrite(LED_BIT0, result & B001);
    digitalWrite(LED_BIT1, result & B010);
    digitalWrite(LED_BIT2, result & B100);
}
```

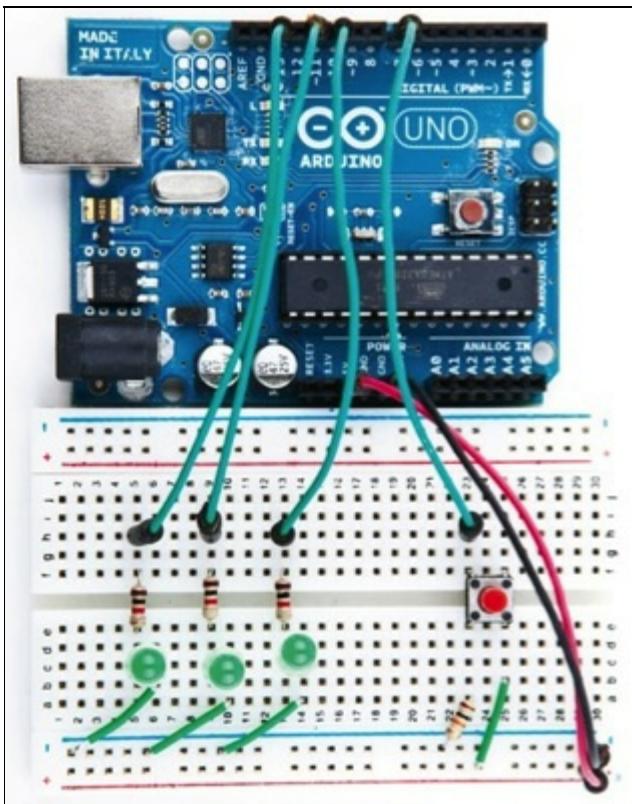


Figura 3.10 Il dado binario con pulsante esterno.

Il programma combina correttamente la soluzione precedente e il codice necessario per svolgere la funzione di antirimbalzo. Anche in questo caso si inizializzano alcuni pin della scheda: tre pin di output relativi ai led e un pin di input per il pulsante. Si inizializza anche il seme casuale, mentre la funzione `loop()` attende una nuova pressione del pulsante. Ogni volta che premete il pulsante si avvia un nuovo “lancio” del dado binario e si visualizza il risultato del lancio tramite l'accensione e lo spegnimento dei led. Ora però il pulsante *Reset* è stato sostituito da un pulsante esterno!

Ora che sapete come è facile aggiungere un pulsante potete leggere il prossimo paragrafo che spiega come trasformare il semplice dado binario in un gioco divertente.

Il gioco del dado

È necessario aggiungere ancora un pulsante per avere in funzione un gioco con tutte le carte in regola. Il primo pulsante permette di “lanciare” il dado, mentre un secondo pulsante consentirà di impostare il risultato su cui si vuole scommettere: sarà sufficiente lanciare di nuovo il dado e ottenere un risultato uguale a quello della giocata per vedere lampeggiare i tre led presenti sul dado, altrimenti i led rimarranno spenti.

Per impostare la giocata dovete premete il secondo pulsante per un numero adeguato di volte. Volete giocare il numero 3? Premete il pulsante della giocata per tre volte successive, poi premete il pulsante di lancio del dado.

Eseguite le stesse operazioni svolte in precedenza per aggiungere un secondo pulsante esterno. Nella Figura 3.11 potete osservare il cablaggio finale del circuito. Questa volta il nuovo pulsante è collegato al pin 5.

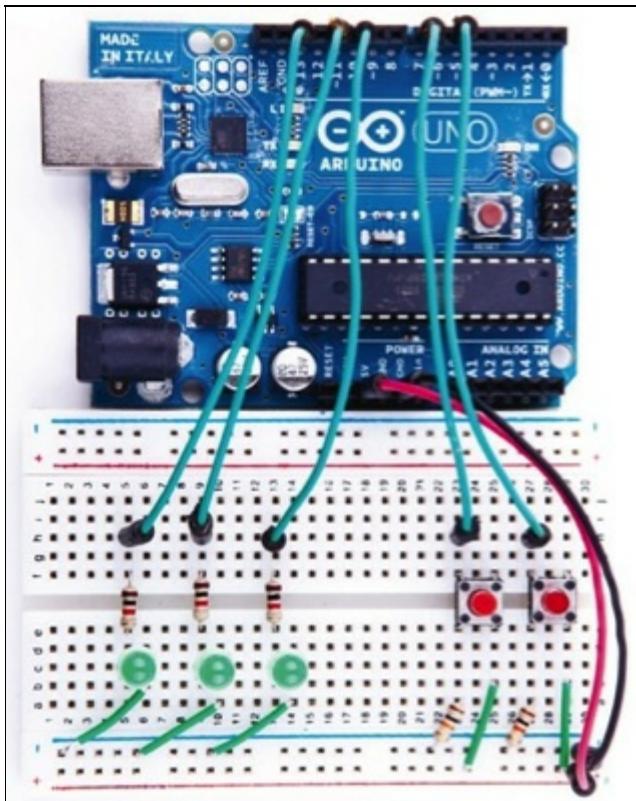


Figura 3.11 Ora il dado binario ha anche il pulsante che imposta la giocata su cui scommettere.

Ora si deve aggiungere il codice di controllo del secondo pulsante. Si potrebbe essere tentati di copiare le istruzioni definite nel programma precedente; dopotutto, il progetto hardware è stato copiato. Nel mondo fisico dell'hardware una ridondanza di questo genere è più che ammissibile, dato che in effetti occorre impiegare due componenti elettronici identici tra loro. Nel mondo virtuale del software, invece, una tale ridondanza è perfettamente inutile, pertanto si preferisce evitare di copiare la funzione antiribalzo per ricorrere all'utilizzo di una libreria scritta proprio a questo proposito (<http://www.arduino.cc/playground/Code/Bounce>). Scaricate la libreria e copiate il suo contenuto non compresso nella directory */Documents/Arduino/libraries* (in un computer Mac) oppure nella directory *My Documents\Arduino\libraries* (in un computer Windows). In genere

questa operazione è l'unica richiesta per avere a disposizione la libreria, anche se è sempre bene consultare le istruzioni di installazione e la documentazione presenti nella pagina web da cui l'avete scaricata. Di seguito è riportata la versione definitiva del software relativo al dado binario.

[BinaryDice/DiceGame/DiceGame.pde](#)

```
Riga 1 #include <Bounce.h>
-
- const unsigned int LED_BIT0 = 12;
- const unsigned int LED_BIT1 = 11;
5 const unsigned int LED_BIT2 = 10;
- const unsigned int START_BUTTON_PIN = 5;
- const unsigned int GUESS_BUTTON_PIN = 7;
- const unsigned int BAUD_RATE = 9600;
-
10 void setup() {
-     pinMode(LED_BIT0, OUTPUT);
-     pinMode(LED_BIT1, OUTPUT);
-     pinMode(LED_BIT2, OUTPUT);
-     pinMode(START_BUTTON_PIN, INPUT);
15     pinMode(GUESS_BUTTON_PIN, INPUT);
-     randomSeed(analogRead(A0));
-     Serial.begin(BAUD_RATE);
- }
-
20 const unsigned int DEBOUNCE_DELAY = 20;
- Bounce start_button(START_BUTTON_PIN, DEBOUNCE_DELAY);
- Bounce guess_button(GUESS_BUTTON_PIN, DEBOUNCE_DELAY);
- int guess = 0;
-
25 void loop() {
-     handle_guess_button();
-     handle_start_button();
- }
-
30 void handle_guess_button() {
-     if (guess_button.update()) {
-         if (guess_button.read() == HIGH) {
-             guess = (guess % 6) + 1;
-             output_result(guess);
35             Serial.print("Guess: ");
-             Serial.println(guess);
-         }
-     }
- }
```

40

```
- void handle_start_button() {
-     if (start_button.update()) {
-         if (start_button.read() == HIGH) {
-             const int result = random(1, 7);
-             output_result(result);
-             Serial.print("Result: ");
-             Serial.println(result);
-             if (guess > 0) {
-                 if (result == guess) {
-                     Serial.println("You win!");
-                     hooray();
-                 } else {
-                     Serial.println("You lose!");
-                 }
-             }
-             delay(2000);
-             guess = 0;
-         }
-     }
- }
-
- void output_result(const long result) {
-     digitalWrite(LED_BIT0, result & B001);
-     digitalWrite(LED_BIT1, result & B010);
-     digitalWrite(LED_BIT2, result & B100);
- }
-
- void hooray() {
-     for (int i = 0; i < 3; i++) {
-         output_result(7);
-         delay(500);
-         output_result(0);
-         delay(500);
-     }
- }
```

In effetti il programma è piuttosto lungo ma si tratta perlopiù di istruzioni già note e le parti nuove sono abbastanza semplici. Alla prima riga si include la libreria Bounce che verrà impiegata più avanti per eseguire la funzione antirimbalzo dei due pulsanti. Il programma prosegue impostando le costanti relative a tutti i pin utilizzati sulla scheda, mentre il metodo `setup()` inizializza i pin e definisce il seme casuale. Le istruzioni inizializzano anche la porta seriale, dato che verranno inviati in output alcuni messaggi di debug.

La libreria `Bounce` dichiara una classe chiamata `Bounce` e il programma deve creare un oggetto `Bounce` per ogni pulsante su cui eseguire l'antirimbalo: queste operazioni sono svolte alle righe 21 e 22. Il costruttore della classe `Bounce` richiede come parametri il numero del pin cui è collegato il pulsante e la pausa dell'antirimbalo espressa in millisecondi. Il programma deve infine dichiarare e inizializzare una variabile `guess` per memorizzare la giocata che si intende effettuare.

La funzione `loop()` si riduce in questo programma a due chiamate di funzioni. Una prima chiamata è responsabile della gestione della giocata effettuata con il pulsante corrispondente, l'altra chiamata controlla la pressione del pulsante di lancio del dado. La funzione

`handle_guess_button()` utilizza per la prima volta la classe `Bounce`. Il suo metodo `update()` stabilisce lo stato attuale dell'oggetto `guess_button`. Successivamente il programma legge lo stato corrente del medesimo oggetto utilizzando il metodo `read()`.

Premete il pulsante per impostare il suo stato con il valore `HIGH` e incrementare la variabile `guess`. L'operatore modulo (%) della riga 33 garantisce che il valore della variabile sia compreso tra 1 e 6; questo operatore esegue la divisione tra due valori e restituisce il resto. Se il divisore è 6 il valore restituito è compreso tra 0 e 5, dato che la divisione tra un numero qualsiasi e 6 produce sempre un resto compreso tra 0 e 5. Il programma incrementa di 1 il valore del resto e si ottengono così solo valori compresi tra 1 e 6. Il programma visualizza infine in output la giocata attuale utilizzando i tre led; lo stesso valore è inviato anche attraverso la porta seriale.

La gestione del pulsante di lancio in `handle_start_button()` è identica a quella prevista per il pulsante che effettua la giocata. Premete il pulsante di lancio del dado per calcolare un nuovo risultato, che viene visualizzato in output sulla porta seriale. A questo punto il programma controlla se l'utente ha impostato una giocata (in questo caso la variabile `guess` ha un valore maggiore di 0) e se l'utente ha ottenuto un risultato che coincide con la giocata. In ogni caso viene inviato un messaggio sulla porta seriale; se l'utente ha vinto si chiama il metodo `hooray()`, che fa lampeggiare più volte i tre led.

Al termine dell'esecuzione di questo metodo il programma attende due secondi prima di avviare un nuovo gioco e impostare a 0 il valore della variabile `guess`.

Caricate il software in Arduino e avviate la finestra *Serial Monitor* dell'IDE, che deve visualizzare il nuovo valore attuale della variabile `guess` ogni volta che premete il pulsante della giocata. Premete il pulsante di lancio per visualizzare il risultato corrispondente. La Figura 3.12 mostra un esempio di output che si ottiene giocando con il dado binario.

In questo capitolo avete portato a termine un primo progetto complesso di Arduino, che ha richiesto l'impiego di una breadboard e di led, pulsanti, resistori e cavi di collegamento. Avete dovuto inoltre digitare un codice software non banale per mettere in funzione l'hardware come si deve. Nel prossimo capitolo scriverete un programma ancora più sofisticato che riguarda la generazione di codice Morse. Vedrete inoltre come creare librerie Arduino da condividere con altri utenti della stessa scheda di controllo.



Figura 3.12 Hai vinto!

Cosa fare se non funziona?

Sono molte le possibilità che qualcosa vada storto quando si lavora per la prima volta con una breadboard. In genere il problema più grave riguarda la connessione non corretta tra i componenti. Dedicate un po' di tempo a fare pratica di cablaggio di led, cavi di collegamento, resistori e pulsanti da inserire nella breadboard. Dovete premere i terminali in modo deciso ma senza esercitare troppa pressione, altrimenti rischiate di piegarli, e non

potrebbero più essere inseriti adeguatamente. In genere è più facile inserire componenti che hanno terminali abbastanza corti. Ricordate di indossare occhiali di protezione quando tagliate i terminali dei componenti elettronici.

Quando manipolate i componenti elettronici dovete anche badare al fatto che alcuni di questi (per esempio i led) devono essere inseriti rispettando la polarità dei terminali. Anche i pulsanti possono manifestare problemi di inserimento nella breadboard; verificate con attenzione la posizione dei terminali dei contatti normalmente aperti del circuito elettrico.

Perfino i cavi di collegamento possono diventare un problema, in particolare quando la loro lunghezza non è adeguata. Sostituite immediatamente i cavi troppo corti, che potrebbero uscire dai fori della breadboard. I cavi di connessione sono troppo economici per sprecare il vostro tempo prezioso con inutili e seccanti operazioni di “debugging”.

Esercizi

- Il dado binario può funzionare bene quando giocate a Monopoli con i vostri amici, ma la maggior parte delle persone sembra preferire i più convenzionali dadi decimali. Provate a trasformare il dado da binario a decimale utilizzando sei led. Disponete i led così come li vedete sulla faccia di un dado qualsiasi.
- I resistori da $1k\Omega$ impiegati in questo capitolo per proteggere i led hanno un valore di resistenza piuttosto elevato. Leggete l'Appendice A e sostituite questi componenti con resistori che abbiano una resistenza più bassa. Si nota la differenza di luminosità dei led?
- I led possono essere impiegati per visualizzare dati ben più complessi di quelli relativi al dado binario. Se disponete di un numero sufficiente di led potete provare a realizzare altri progetti, per esempio un orologio binario (<http://www.instructables.com/id/LED-Binary-Clock/>). A questo punto dovreste saperne abbastanza di circuiti elettronici e di programmi Arduino per riuscire a creare un orologio binario. Provate a realizzare un progetto di questo tipo oppure un altro tipo di display che utilizzi più led.
- L'impiego di un pulsante per “lanciare” il dado è abbastanza scomodo, non vi sembra? In genere, per lanciare i dadi questi vanno tenuti in una mano e agitateli prima di buttarli sul tavolo da gioco. Potete facilmente simulare questo effetto utilizzando un tilt sensor. Questo genere di sensori rileva l'oscillazione di un oggetto ed è quindi particolarmente indicato per

simulare l'effetto di “agitazione” di un dado. Da un punto di vista teorico il funzionamento di un tilt sensor è analogo a quello di un pulsante ma, a differenza di questo, non dovete premere il sensore bensì scuotere lo rispetto alla posizione di riposo. Provate ad aggiungere uno di questi sensori al dado binario dopo aver studiato le indicazioni fornite sul sito web del progetto Arduino
(<http://www.arduino.cc/en/Tutorial/TiltSensor>).

Libreria per la generazione di un codice Morse

Ora che conoscete abbastanza bene l'IDE Arduino e sapete come accendere i led potete affrontare un progetto più significativo. In questo capitolo verrà sviluppato un generatore di codice Morse che legge un testo inviato alla porta seriale e lo trasmette in output sotto forma di segnali luminosi che accendono e spengono un led.

La realizzazione di questo progetto permette di approfondire la conoscenza della trasmissione seriale tra Arduino e computer. L'approfondimento riguarda anche il funzionamento della stessa IDE di sviluppo, in quanto si vedrà come utilizzare librerie esistenti e come strutturare progetti di grandi dimensioni nelle proprie librerie di programmi. Al termine di questo capitolo sarete in grado di realizzare una libreria che potrà essere pubblicata su Internet.

Cosa serve

- Una scheda Arduino, per esempio un modello Arduino Uno, Duemilanove o Diecimila.
- Un cavo USB per collegare la scheda Arduino al computer.
- Un led.
- Un altoparlante o un buzzer (facoltativi).

Concetti di base del codice Morse

Il codice Morse fu inventato per convertire un testo in segnali audio

(http://en.wikipedia.org/wiki/Morse_Code o

http://it.wikipedia.org/wiki/codice_Morse). Da un punto di vista teorico la codifica è simile a quella che converte un insieme di caratteri nel codice ASCII. La differenza è che il codice ASCII rappresenta i caratteri in forma numerica, mentre il codice Morse è costituito da sequenza di punti e linee (chiamati in gergo *di* e *dah*). I punti hanno una durata inferiore rispetto a quella delle linee.

Una lettera A è codificata dalla sequenza · – (*di dah*), mentre la lettera Z corrisponde a – – · · (*dah dah di di*).

Il codice Morse è caratterizzato anche da una temporizzazione che stabilisce la durata dei punti e delle linee e deve anche specificare la durata delle pause tra simboli differenti e tra parole. L'unità di base del codice Morse è costituita dalla durata di un punto, da cui deriva per esempio che una linea corrisponde a tre punti. Tra due simboli consecutivi la pausa deve durare quanto un punto, mentre tra due lettere la pausa è di tre punti. Tra due parole si deve avere una pausa di sette punti.

La trasmissione di un messaggio in codice Morse richiede una modalità di trasmissione di segnali caratterizzati da lunghezze differenti l'uno dall'altro. Una tecnica classica è rappresentata dall'utilizzo di segnali sonori, ma in questo progetto si impiegherà un led che verrà acceso e spento con diversi intervalli di tempo. In Marina è ancora in uso la trasmissione Morse basata su segnali luminosi.

Ora potete implementare il generatore di codice Morse!

Realizzare un generatore di codice Morse

L'elemento principale della libreria del generatore è costituito dalla classe C++ chiamata `Telegraph`. In questo paragrafo verrà definita la sua interfaccia a partire dallo schema di principio indicato di seguito.

[Telegraph/Telegraph.pde](#)

```
void setup() {  
}  
  
void loop() {  
}
```

Si tratta ovviamente di un programma Arduino ridotto ai minimi termini, che non svolge alcuna operazione a eccezione della dichiarazione delle funzioni obbligatorie, ancora prive di istruzioni. Questa tecnica di sviluppo consente però di compilare il lavoro progressivamente verificando di volta in volta la presenza di errori sintattici. Salvate il programma con il nome `Telegraph`, in modo che l'IDE possa creare una cartella chiamata `Telegraph` e vi includa il file `Telegraph.pde`. Tutti i file e le directory richiesti dalla libreria saranno memorizzati nella cartella `Telegraph`.

A questo punto aprite una nuova scheda e indicate il nome di file `telegraph.h`. Proprio così, avete appena creato un file header in C; anzi, per essere precisi è un file header in C++. Di seguito è riportato l'elenco delle istruzioni di questo file.

Telegraph/telegraph.h

```
#ifndef __TELEGRAPH_H__
#define __TELEGRAPH_H__


class Telegraph {
public:
    Telegraph(const int output_pin, const int dit_length);
    void send_message(const char* message);

private:
    void dit();
    void dah();
    void output_code(const char* code);
    void output_symbol(const int length);

    int _output_pin;
    int _dit_length;
    int _dah_length;
};

#endif
```

Ricordate sempre che la programmazione orientata agli oggetti non è più una esclusiva delle CPU di grandi capacità! In questo caso il programma descrive l'interfaccia della classe `Telegraph` da impiegare nel vostro primo progetto importante, a condizione ovviamente che siate interessati alla trasmissione di informazioni in codice Morse.

La prima riga del programma imposta un meccanismo di esclusione della doppia inclusione; in altre parole, il corpo del file header definisce una macro di preprocessore denominata `__TELEGRAPH_H__`. Il corpo del file (che contiene la definizione della macro) è incluso in una istruzione `#ifndef`, pertanto le sue istruzioni vengono compilate solo se la macro non è stata ancora definita. In questo modo potete includere l'header più volte nel progetto e garantire che le istruzioni dell'header saranno compilate una sola volta.

L'interfaccia della classe `Telegraph` è caratterizzata da una parte pubblica cui hanno accesso gli utenti della classe e da una parte privata che può essere utilizzata solo dai membri della classe. Nella parte pubblica sono presenti due elementi: un costruttore per la creazione di nuovi oggetti `Telegraph` e un metodo chiamato `send_message()` che invia un messaggio tramite l'emissione di un segnale in codice Morse. Nelle applicazioni del vostro progetto potrete utilizzare questa classe C++ come segue:

```
Telegraph telegraph(13, 200);
telegraph.send_message("Hello, world!");
```

Nella prima riga viene creato un nuovo oggetto `Telegraph` che comunica con il pin 13 ed emette “punti” che hanno una durata di 200 millisecondi. La seconda istruzione trasmette il messaggio “Hello, world!” in codice Morse. Questa soluzione permette di inviare un messaggio qualsiasi e di modificare facilmente l'impostazione che riguarda il pin di trasmissione e la durata di un punto.

Dopo aver definito l'interfaccia della classe potete implementare il programma in base alle indicazioni del prossimo paragrafo.

Perfezionare l'interfaccia del generatore

La dichiarazione delle interfacce è una fase importante dello sviluppo di un programma, almeno quanto lo è la sua implementazione. Create una nuova scheda, immettete il nome di file `telegraph.cpp` e riportate le istruzioni che seguono.

NOTA

Le versioni meno recenti di Arduino presentano un bug che non consente la creazione di un nuovo file in questo modo, dato che l'IDE segnala la presenza di un file che ha lo stesso nome. Per sapere come risolvere il problema consultate la pagina web all'indirizzo <http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1251245246>.

`Telegraph/telegraph.cpp`

```
#include <ctype.h>
#include <WProgram.h>
#include "telegraph.h"

char* LETTERS[] = {
  ".-", "-...", "-.-.", "-..", ".-", // A-E
  "....", "--.", "...", "...", ".---", // F-J
  "-.-", "-...-", "--", "-.", "---", // K-O
  "
```

```

"---.", "--.-", ".-.", "...", "--",      // P-T
"-.-", "...-", ".--", "-..-", "-.--", // U-Y
"--.."
};

char* DIGITS[] = {
    "----", ".----", "....-", "...--", // 0-3
    ".....", ".....", "-....", "--....", // 4-7
    "----.", "----."                      // 8-9
};

```

Analogamente alla maggior parte dei programmi in C++, anche questo importa in primo luogo alcune librerie. Dato che più avanti verranno impiegate funzioni particolari, per esempio `toupper()`, il programma include `ctype.h` cui si aggiunge `telegraph.h` per rendere disponibile la dichiarazione della classe illustrata nel paragrafo precedente e le sue funzioni. A cosa serve però la libreria `WProgram.h`?

Finora non è stato necessario occuparsi della provenienza di costanti quali `HIGH`, `LOW` oppure `OUTPUT`. Queste costanti sono definite in diversi file header messi a disposizione dall'IDE Arduino e che potete trovare nella directory `hardware/cores/arduino` della stessa IDE. Esaminando la libreria `WProgram.h` vedete che nel file `wiring.h` sono contenute tutte le costanti impiegate finora e molte altre ancora. Questa libreria dichiara molte macro e gran parte delle funzioni principali di Arduino.

Fintanto che modificate programmi scritti a partire da zero non dovete preoccuparvi di includere file header standard, in quanto l'IDE svolge questa operazione dietro le quinte. Non appena iniziate a elaborare progetti più complessi che includono codice C++ vero e proprio, nasce l'esigenza di gestire l'inclusione di tutte le librerie. Dovete in altre parole importare tutte le librerie richieste dal programma, come quelle delle costanti di Arduino.

Dopo aver importato i file header il programma determina due array di tipo stringa chiamati `LETTERS` e `DIGITS`. Questi array contengono il codice Morse di lettere e cifre che useremo più avanti per convertire il testo del messaggio in codice Morse. Prima di effettuare questa operazione il programma definisce il costruttore che deve creare e inizializzare i nuovi oggetti

`Telegraph`.

Telegraph/telegraph.cpp

```
Telegraph::Telegraph(const int output_pin, const int dit_length) {
```

```

    _output_pin = output_pin;
    _dit_length = dit_length;
    _dah_length = dit_length * 3;
    pinMode(_output_pin, OUTPUT);
}

```

Il costruttore richiede due argomenti: il numero del pin ove inviare il codice Morse e la lunghezza di un punto espresso in millisecondi. Il programma memorizza questi valori in variabili istanza corrispondenti, calcola la durata di una linea e imposta il pin di comunicazione come pin di output.

Avete probabilmente notato che le variabili istanza di tipo privato hanno un nome che inizia con un underscore (_), una convenzione che non è peraltro imposta dal linguaggio C++ o dall'IDE Arduino.

Output dei simboli del codice Morse

Al termine delle inizializzazioni il programma può iniziare a trasmettere in output i simboli del codice Morse. La leggibilità del codice è facilitata dalla presenza di svariati metodi helper.

[Telegraph/telegraph.cpp](#)

```

void Telegraph::output_code(const char* code) {
    for (int i = 0; i < strlen(code); i++) {
        if (code[i] == '.')
            dit();
        else
            dah();
    }
}

void Telegraph::dit() {
    Serial.print(".");
    output_symbol(_dit_length);
}

void Telegraph::dah() {
    Serial.print("-");
    output_symbol(_dah_length);
}

void Telegraph::output_symbol(const int length) {
    digitalWrite(_output_pin, HIGH);
    delay(length);
    digitalWrite(_output_pin, LOW);
}

```

}

La funzione `output_code()` accetta una sequenza in codice Morse costituita da punti e linee e la converte in chiamate delle funzioni `dit()` e `dah()`. I metodi `dit()` e `dah()` inviano un punto e una linea alla porta seriale e lasciano il resto dell'elaborazione al metodo `output_symbol()`, a cui passano la lunghezza del simbolo in codice Morse da trasmettere. La funzione `output_symbol()` imposta il pin di output nello stato `HIGH` per tutta la lunghezza del simbolo, dopodiché viene ripristinato lo stato `LOW`. Le operazioni vengono effettuate in base allo schema di temporizzazione del codice Morse; manca solo l'implementazione del metodo `send_message()`, riportata di seguito.

[Telegraph/telegraph.cpp](#)

```
Riga 1 void Telegraph::send_message(const char* message) {
-    for (int i = 0; i < strlen(message); i++) {
-        const char current_char = toupper(message[i]);
-        if (isalpha(current_char)) {
5            output_code(LETTERS[current_char - 'A']);
-            delay(_dah_length);
-        } else if (isdigit(current_char)) {
-            output_code(DIGITS[current_char - '0']);
-            delay(_dah_length);
10       } else if (current_char == ' ') {
-           Serial.print(" ");
-           delay(_dit_length * 7);
-       }
-    }
15   Serial.println();
- }
```

Questa funzione trasmette un carattere del messaggio alla volta, all'interno di un ciclo di istruzioni. Alla riga 3 il carattere iniziale è convertito in una lettera maiuscola, dato che le minuscole non sono riconosciute dal codice Morse (questo è il motivo per cui non è possibile implementare un client per chat da trasmettere in codice Morse). A questo punto il programma verifica se il carattere attuale è costituito da una lettera ricavata dalla funzione `isalpha()` del linguaggio C. In questo caso si utilizza la funzione per stabilire la rappresentazione in codice Morse memorizzata nell'array `LETTERS`. Per eseguire questa operazione si adotta un vecchio stratagemma: nella tabella ASCII tutte le lettere (e le cifre) compaiono in modo

sequenziale, ovvero si ha A=65, B=66 e così via. Il carattere attuale può pertanto essere trasformato nell’indice dell’array `LETTERS` sottraendo 65 ('A') dal valore del codice ASCII. Il valore corrente del codice Morse è poi trasferito al metodo `output_symbol()` e si ritarda in questo modo il programma per un tempo pari alla durata di una linea.

L’algoritmo prevede un funzionamento analogo per la trasmissione di cifre. Occorre semplicemente ricavare l’indice della array `DIGITS` e non dell’array `LETTERS` e poi sottrarre il valore ASCII del carattere 0.

Alla riga 10 il programma verifica la presenza di un carattere vuoto; in questo caso si deve inviare alla porta seriale un carattere vuoto e attendere per un tempo pari a sette punti. Tutti gli altri caratteri vengono ignorati dal programma, che può elaborare solo lettere, cifre e caratteri vuoti. Al termine dell’esecuzione del metodo il programma invia alla porta seriale un carattere newline per identificare la fine del messaggio.

Installare e utilizzare la classe `Telegraph`

La classe `Telegraph` è veramente completa e può essere impiegata per realizzare altri esempi di programma. Questa considerazione è importante per due motivi: potete verificare le funzionalità del codice presente nella libreria e gli utenti della classe possono ricavarne informazioni utili per studiarne l’utilizzo.

L’IDE Arduino ricerca le librerie in due posizioni differenti, ovvero nella cartella globale *libraries* relativa alla directory di installazione dell’IDE e nella directory locale dei progetti dell’utente dell’applicazione. In fase di sviluppo conviene utilizzare la directory dei progetti, la cui posizione può essere individuata tra le preferenze dell’IDE, come si può osservare nella Figura 4.1. Create una nuova directory di nome *libraries* nella directory locale dei progetti.

Per rendere disponibile la classe `Telegraph` dovete creare una sottocartella `Telegraph` nella directory *libraries*, poi copiate in questa cartella i file `telegraph.h` e `telegraph.cpp`. Non copiate il file `Telegraph.pde`. Riavviate l’IDE.

Ora potete provare la madre di tutti i messaggi: “Hello, world!”. Create un nuovo progetto denominato `HelloWorld` e digitate le istruzioni che seguono.

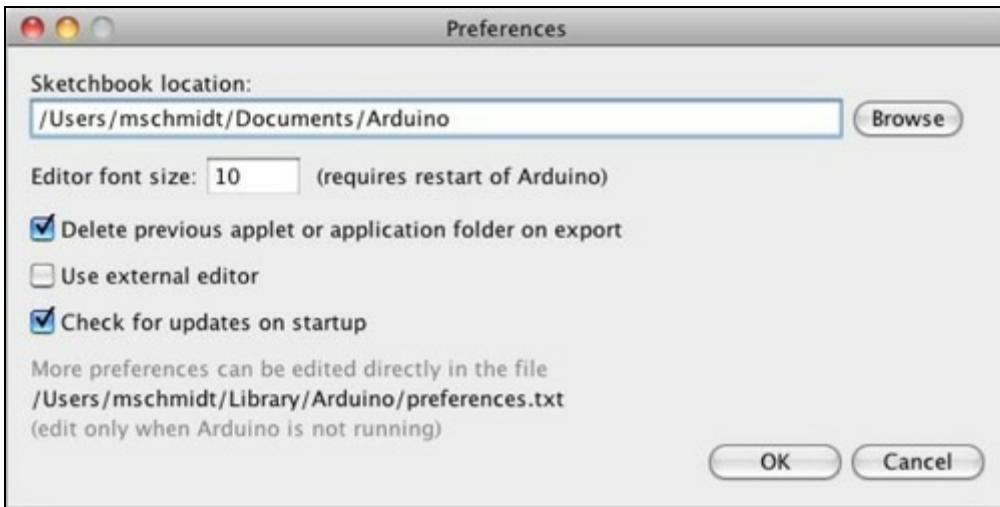


Figura 4.1 Individuate nelle preferenze la directory in cui memorizzare i progetti.

[Telegraph/examples/HelloWorld>HelloWorld.pde](#)

```
#include "telegraph.h"
const unsigned int OUTPUT_PIN = 13;
const unsigned int DIT_LENGTH = 200;

Telegraph telegraph(OUTPUT_PIN, DIT_LENGTH);

void setup() {}

void loop() {
    telegraph.send_message("Hello, world!");
    delay(5000);
}
```

Questo programma trasmette ogni cinque secondi la stringa “Hello, world!” in codice Morse. Le istruzioni includono la definizione della classe `Telegraph` e delle costanti che riguardano il pin ove è collegato il led e la durata di un punto. A questo punto il programma crea un oggetto globale `Telegraph` e una funzione vuota `setup()`. Nella funzione `loop()` si chiama ogni cinque secondi il metodo `send_message()` sull’istanza `Telegraph`.

La compilazione di questo programma implica anche la compilazione automatica della libreria `Telegraph`. Un eventuale errore di sintassi nella libreria verrà pertanto segnalato in questa fase di compilazione. Correggete gli errori assicurandovi di modificare i file originali del codice sorgente. Dopo aver corretto gli errori copiate nuovamente i file nella cartella `libraries`; non dimenticare di riavviare l’IDE.

La conversione di una stringa statica in codice Morse è un'operazione interessante ma non sarebbe meglio fare in modo che il programma elaborasse stringhe arbitrarie? Per ottenere questo risultato dovete studiare un esempio di programma più sofisticato del precedente. Questa volta si scriverà un codice che legge messaggi dalla porta seriale e li riporta verso un'istanza `Telegraph`. Create un nuovo progetto di nome `MorseCodeGenerator` e digitate le istruzioni qui riportate.

[Telegraph/examples/MorseCodeGenerator/MorseCodeGenerator.pde](#)

```
#include "telegraph.h"

const unsigned int OUTPUT_PIN = 13;
const unsigned int DIT_LENGTH = 200;
const unsigned int MAX_MESSAGE_LEN = 128;
const unsigned int BAUD_RATE = 9600;
const int LINE_FEED = 13;

char message_text[MAX_MESSAGE_LEN];
int index = 0;

Telegraph telegraph(OUTPUT_PIN, DIT_LENGTH);

void setup() {
    Serial.begin(BAUD_RATE);
}

void loop() {
    if (Serial.available() > 0) {
        int current_char = Serial.read();
        if (current_char == LINE_FEED || index == MAX_MESSAGE_LEN - 1) {
            message_text[index] = 0;
            index = 0;
            telegraph.send_message(message_text);
        } else {
            message_text[index++] = current_char;
        }
    }
}
```

Anche in questo caso si include il file header della classe `Telegraph` e come al solito si definiscono alcune costanti: `OUTPUT_PIN` stabilisce il pin di collegamento del led, mentre `DIT_LENGTH` contiene la durata di un punto espresso in millisecondi. La costante `LINE_FEED` è impostata con il valore

ASCII che corrisponde al carattere linefeed; questo carattere è necessario per stabilire la fine del messaggio da trasmettere. Infine, la costante `MAX_MESSAGE_LEN` stabilisce la lunghezza massima dei messaggi che il programma consente di trasmettere in codice Morse.

A questo punto si definiscono tre variabili globali: `message_text` è il buffer di caratteri da riempire con i dati ricevuti dalla porta seriale, `index` tiene traccia della posizione attuale nel buffer e `telegraph` è l'oggetto `Telegraph` da impiegare per convertire il messaggio in “segnali luminosi”. La funzione `setup()` inizializza la porta seriale, mentre `loop()` verifica l'arrivo di nuovi dati chiamando il metodo `Serial.available()`. Il programma legge il byte successivo di dati, se disponibile, e verifica se si tratta di un carattere linefeed oppure dell'ultimo byte da inserire nel buffer di caratteri. In entrambi i casi si imposta l'ultimo byte di `message_text` con il valore 0, dato che in C/C++ le stringhe sono terminate dal carattere null.

Ora dovete compilare e caricare il programma. Aprite *Serial Monitor* e selezionate *Carriage return* come terminazione di riga nel menu a comparsa visibile nella parte inferiore della finestra. Questa opzione permette a *Serial Monitor* di aggiungere automaticamente un carattere newline a ogni riga inviata ad Arduino. Digitate un messaggio qualsiasi, per esempio il vostro nome, e osservate come Arduino converte il messaggio in segnali luminosi.

L'incapsulamento della logica del codice Morse nella classe `Telegraph` comporta il vantaggio che il programma principale è breve e conciso. La creazione di software per dispositivi miniaturizzati non significa che si debba rinunciare ai vantaggi offerti dalla programmazione orientata agli oggetti.

Non resta che aggiungere alcuni dettagli meno significativi per trasformare il progetto in una libreria di prima classe, in base alle indicazioni del prossimo paragrafo.

Ritocchi finali

Una delle funzioni più interessanti dell'IDE Arduino riguarda l'uso dei colori per distinguere la sintassi dei comandi. I nomi delle classi, i nomi delle funzioni, le variabili e altro ancora sono visualizzati con colori differenti nella finestra dell'editor. In questo modo il codice sorgente è più semplice da leggere ed è possibile anche colorare la sintassi delle librerie.

Dovete semplicemente aggiungere al progetto un file chiamato `keywords.txt`.

Telegraph/keywords.txt

```
# Sintassi a colori della libreria Telegraph
```

```
Telegraph      KEYWORD1
send_message  KEYWORD2
```

La riga che inizia con un carattere # è di commento e sarà ignorata dal compilatore. Le righe successive contengono il nome di uno dei membri della libreria e il tipo di membro. Separate i due parametri con un carattere di tabulazione. Le classi sono di tipo `KEYWORD1`, mentre le funzioni sono di tipo `KEYWORD2`. Per quanto riguarda le costanti impostate il tipo `LITERAL1`.

Attivate la sintassi a colori per la libreria `Telegraph` copiando il file `keywords.txt` nella cartella *libraries* e riavviate l'IDE. A questo punto il nome della classe `Telegraph` compare in arancione, mentre `send_message()` è in colore marrone.

Prima di terminare la pubblicazione della libreria dovete effettuare le operazioni descritte di seguito.

- Memorizzate gli esempi di programma in una cartella chiamata *examples* e copiate questa cartella nella directory *libraries*. Ogni programma deve avere una cartella in questa directory.
- Stabilite i termini della licenza d'uso del progetto e trascriveteli in un file chiamato `LICENSE`. All'indirizzo <http://www.opensource.org/> trovate molte informazioni al riguardo e licenze di tipo standard. Ricordate che questa opzione consente di aumentare la fiducia dei potenziali utenti del progetto.
- Aggiungete istruzioni per l'installazione e altra documentazione di progetto. In genere gli utenti si aspettano di trovare la documentazione in un file chiamato `README` e cercano le istruzioni di installazione nel file `INSTALL`. Provate a installare la libreria utilizzando sistemi operativi diversi tra loro e per ogni sistema fornite adeguate istruzioni di installazione.

Al termine di queste operazioni la cartella della libreria dovrebbe avere un aspetto simile a quello mostrato nella Figura 4.2.

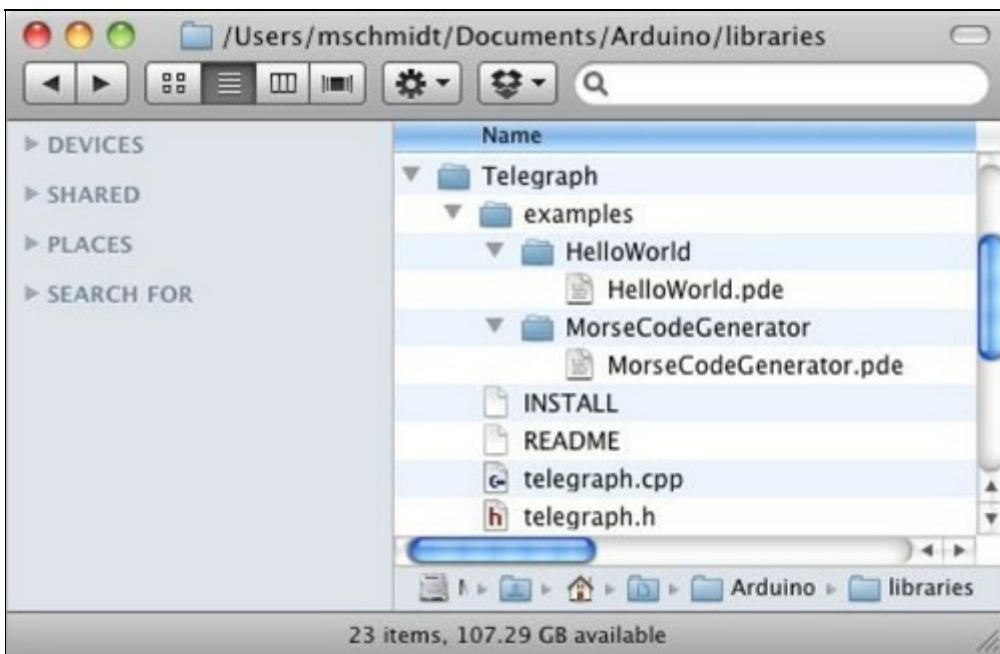


Figura 4.2 Esempio tipico del contenuto di una libreria Arduino.

Ora non rimane che creare un archivio ZIP che contenga tutti i file del progetto Arduino. La maggior parte dei sistemi operativi richiede semplicemente di fare clic con il tasto destro del mouse sulla directory in *Esplora risorse*, nel *Finder* o in un'altra funzionalità analoga a queste e trasformare la directory in un archivio ZIP. I sistemi Linux e Mac consentono di creare un archivio ZIP anche utilizzando una delle istruzioni da riga di comando qui riportate:

```
maik> zip -r Telegraph Telegraph  
maik> tar cfvz Telegraph.tar.gz Telegraph
```

La prima istruzione crea un file chiamato `Telegraph.zip`, mentre la seconda crea l'archivio `Telegraph.tar.gz`. Entrambi i formati sono molto diffusi ed è sempre bene mettere a disposizione degli utenti entrambi i formati compressi.

La creazione di una libreria Arduino è abbastanza semplice, anche se dovete eseguire più di una operazione manuale. In definitiva, non avete scusanti: ogni volta che pensate di aver realizzato qualcosa di interessante dovete renderlo pubblico.

I progetti illustrati finora hanno comunicato con l'esterno tramite led (output) e pulsanti (input). Nel prossimo capitolo si vedrà come impiegare dispositivi di input più sofisticati, per esempio sensori a ultrasuoni.

Studierete anche la possibilità di visualizzare i dati trasmessi da Arduino tramite programmi eseguiti dal computer.

Cosa fare se non funziona?

L'IDE Arduino considera fondamentale la denominazione di file e directory ed è stata ideata per realizzare progetti, non librerie. Per questo motivo dovete effettuare alcune operazioni manuali sui file per organizzare l'archivio di una libreria. Nella Figura 4.2 potete osservare l'aspetto finale della directory di una libreria Arduino. Se avete installato più di una versione dell'IDE Arduino ricordate di verificare che state impiegando la corretta cartella *libraries*.

Tenete presente che dovete spesso riavviare l'IDE e comunque dovete farlo ogni volta che modificate uno dei file della libreria.

Se la sintassi a colori non funziona dovete controllare che il file delle parole chiave sia denominato `keywords.txt`. Verificate con molta attenzione che gli oggetti e gli indicatori di tipo siano separati da un carattere di tabulazione, quindi riavviate l'IDE.

Esercizi

- Il codice Morse supporta non solo le lettere ma anche i numeri. Definisce inoltre simboli di uso comune, per esempio le virgole. Modificate la classe `Telegraph` in modo che il programma possa interpretare l'intero codice Morse.
- L'accensione dei led produce un effetto interessante ma il codice Morse è in genere associato alla ricezione di segnali audio. Sostituite il led con un altoparlante piezo, di basso costo e semplice da utilizzare. La Figura 4.3 mostra il collegamento di un altoparlante alla scheda Arduino. Gli altoparlanti hanno un pin di massa e un pin di segnale: collegate tra loro i pin di massa dell'altoparlante e di Arduino, mentre il pin di segnale dell'altoparlante deve essere collegato al pin 13 della scheda. Sostituite il metodo `output_symbol()` con le istruzioni qui riportate:

```
void Telegraph::output_symbol(const int length) {
    const int frequency = 131;
    tone(_output_pin, frequency, length);
}
```

Questo metodo invia un'onda quadra all'altoparlante e riproduce un tono a frequenza di 131 Hz. L'esempio di progetto "Melody" incluso nell'IDE Arduino spiega come suonare note particolari con un altoparlante piezo.

- Modificate il progetto della libreria in modo da supportare dispositivi di output differenti tra loro. Potete per esempio impostare un oggetto `Output-Device` del costruttore `Telegraph` e derivare un oggetto `LedDevice` e `SpeakerDevice` da `OutputDevice`. Osservate il codice che segue.

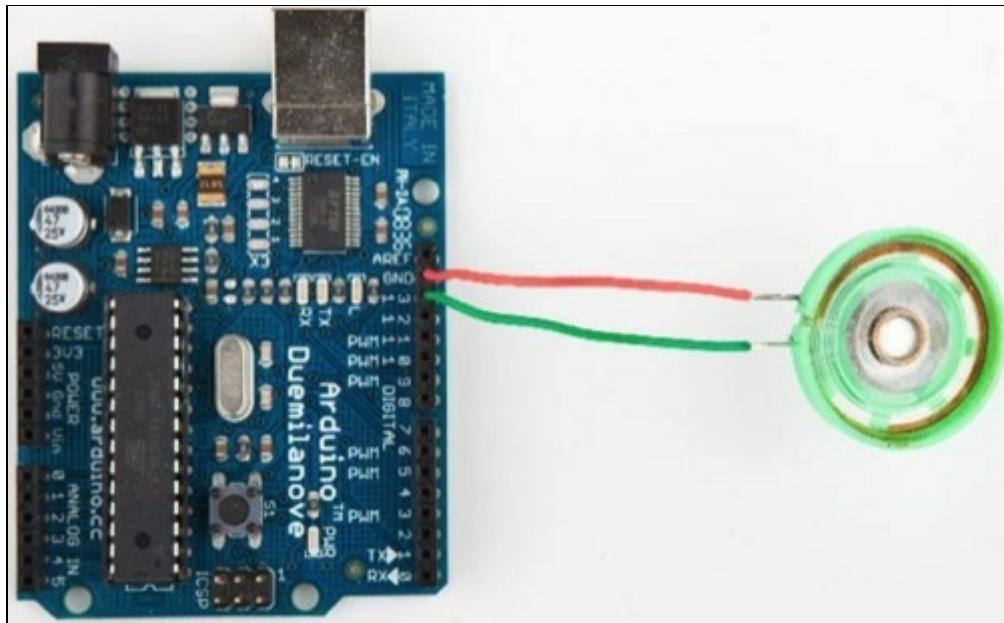


Figura 4.3 Collegare un altoparlante alla scheda Arduino è facile.

```
class OutputDevice {
public:
    virtual void output_symbol(const int length);
};

class Led : public OutputDevice {
public:
    void output_symbol(const int length) {
        // ...
    }
};

class Speaker : public OutputDevice {
public:
    void output_symbol(const int length) {
        // ...
    }
};
```

Potete utilizzare queste classi come segue:

```
Led led;  
Speaker speaker;  
OutputDevice* led_device = &led;  
OutputDevice* speaker_device = &speaker;  
  
led_device->output_symbol(200);  
speaker_device->output_symbol(200);
```

Il resto del programma è lasciato come esercizio da svolgere per conto vostro.

- Imparate il codice Morse. Fate trasmettere da un vostro amico alcuni messaggi che vengano visualizzati nel terminale seriale e provate a interpretare il testo del messaggio che state leggendo. Questo esercizio non è necessario per studiare lo sviluppo dei programmi Arduino ma può essere molto divertente!

Misurare con i sensori il mondo che ci circonda

La scheda Arduino può rilevare le modifiche che avvengono nell'ambiente circostante tramite l'impiego di sensori particolari, che escludono l'utilizzo di mouse e tastiera tipici dei computer. Esistono sensori che misurano la temperatura, l'accelerazione o la distanza di un oggetto collocato nelle vicinanze della scheda.

I sensori sono un elemento essenziale del physical computing e Arduino ne semplifica enormemente l'impiego. In questo capitolo studierete il funzionamento di sensori digitali e analogici che rilevano grandezze fisiche reali e che ricavano misure impiegando solo alcuni cavi e poche istruzioni di programma.

In particolare si approfondirà la conoscenza di due tipi di sensori: uno a ultrasuoni per misurare distanze e uno di temperatura. Il sensore a ultrasuoni permetterà di realizzare un righello digitale per la misura di distanze. I sensori a ultrasuoni sono in grado di effettuare misure abbastanza accurate, che possono essere perfezionate ricorrendo a semplici stratagemmi. È interessante notare che il sensore di temperatura servirà proprio a questo scopo, e alla fine del capitolo vedrete come realizzare un righello digitale molto preciso. Verrà proposta anche un'applicazione grafica che visualizza i dati misurati con i sensori.

Arduino non si limita a semplificare l'impiego di sensori, ma stimola anche la creazione di circuiti elettronici e di soluzioni software sempre efficaci. Per esempio, i due sensori verranno utilizzati insieme, ma le rispettive funzioni rimarranno definite nel software in modo indipendente. In altre parole, tutti i programmi presentati in questo capitolo verranno eseguiti senza alcuna modifica sul circuito definitivo del progetto.

Cosa serve

1. Un sensore a ultrasuoni Parallax PING)).
2. Un sensore di temperatura Analog Devices TMP36.
3. Una breadboard.

4. Cavi di collegamento.
5. Una scheda Arduino, per esempio un modello Uno, Duemilanove o Diecimila.
6. Un cavo USB per collegare Arduino al computer.
7. Una versione del linguaggio di programmazione Processing (<http://processing.org>).

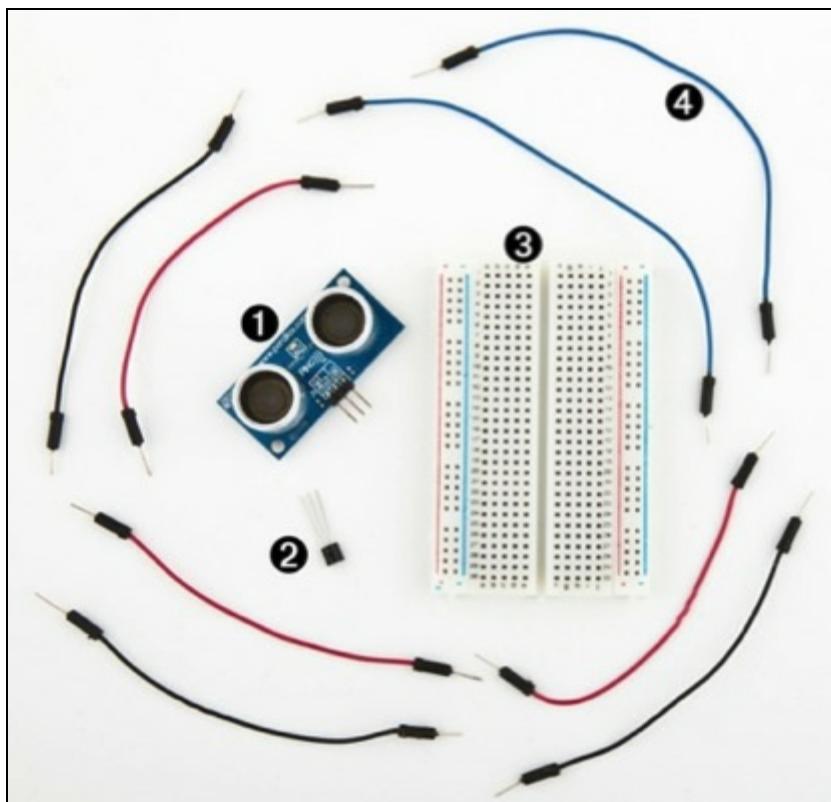


Figura 5.1 Componenti necessari per realizzare i progetti di questo capitolo.

Misurare le distanze con un sensore a ultrasuoni

La misura automatica e continua delle distanze può risultare utile in molte occasioni. Immaginate per esempio un sistema robotizzato che deve trovare il percorso su cui muoversi oppure un sistema anti-intrusione che deve far suonare un allarme o chiamare la polizia ogni volta che qualcuno si avvicina alla vostra casa oppure al ritratto della *Gioconda*. Potete realizzare tutte queste applicazioni con Arduino. Prima di creare l'allarme o il robot dovete però conoscere alcune nozioni fondamentali.

Esistono molti tipi di sensori che misurano distanze e Arduino è in grado di funzionare con la maggior parte di questi componenti. Alcuni di essi si basano su ultrasuoni, altri lavorano con luce a infrarossi o con luce laser. Dal punto di vista teorico tutti questi sensori funzionano comunque allo stesso modo: emettono un segnale, attendono il ritorno dell'eco e misurano

il tempo trascorso tra emissione e ricezione del segnale. È sufficiente conoscere la velocità del suono o della luce nell'aria per convertire il tempo misurato in una distanza.

Il primo progetto consiste nella creazione di un dispositivo che misura la distanza tra sensore e l'oggetto più vicino e visualizza in output il risultato della misura sulla porta seriale. In questo progetto verrà impiegato il sensore a ultrasuoni Parallax PING)))

(<http://www.parallax.com/StoreSearchResults/tabid/768/txtSearch/28015/ProductID/92/Default.aspx>). La scelta di questo componente è dettata dal fatto che è semplice da utilizzare, è corredata di una ricca documentazione e offre interessanti funzionalità. È in grado di rilevare oggetti a distanze variabili tra 2 centimetri e 3 metri, può essere cablato direttamente su una breadboard senza alcuna saldatura. È anche un ottimo esempio di componente che fornisce dati in forma di impulsi a larghezza variabile, come verrà spiegato più avanti in questo capitolo. Il sensore PING))) permette di realizzare facilmente un sonar oppure un robot che trova la via d'uscita da un labirinto senza toccare nemmeno una parete.

È già stato detto che i sensori a ultrasuoni non forniscono direttamente la misura della distanza dell'oggetto più vicino. Questi sensori misurano il tempo che impiega un segnale sonoro emesso dal sensore a raggiungere l'oggetto e ritornare sul sensore. Il componente PING))) adotta questa tecnica, come si può vedere nella Figura 5.2, e il suo circuito interno è abbastanza complesso; fortunatamente, il sensore presenta all'esterno tre soli pin: alimentazione, massa e segnale.

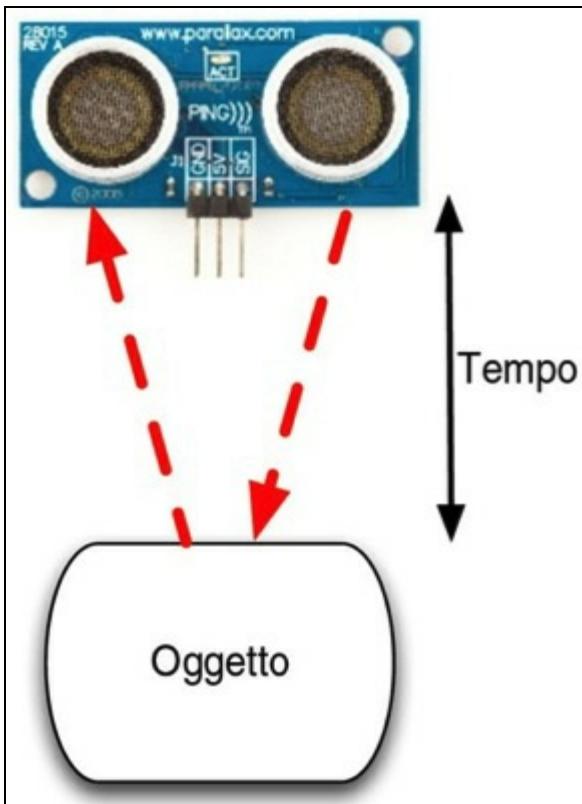


Figura 5.2 Principio di funzionamento del sensore PING))).

La disposizione dei terminali semplifica il collegamento tra sensore e Arduino. Collegate innanzitutto i morsetti di massa e di 5V ai corrispondenti pin del sensore PING))), poi collegate il terzo pin del sensore (segnale) a uno dei pin digitali di IO. Nel progetto di esempio si è scelto di impiegare il pin 7, senza un motivo particolare. La Figura 5.3 mostra lo schema di cablaggio del circuito, che è riprodotto in foto nella Figura 5.5.

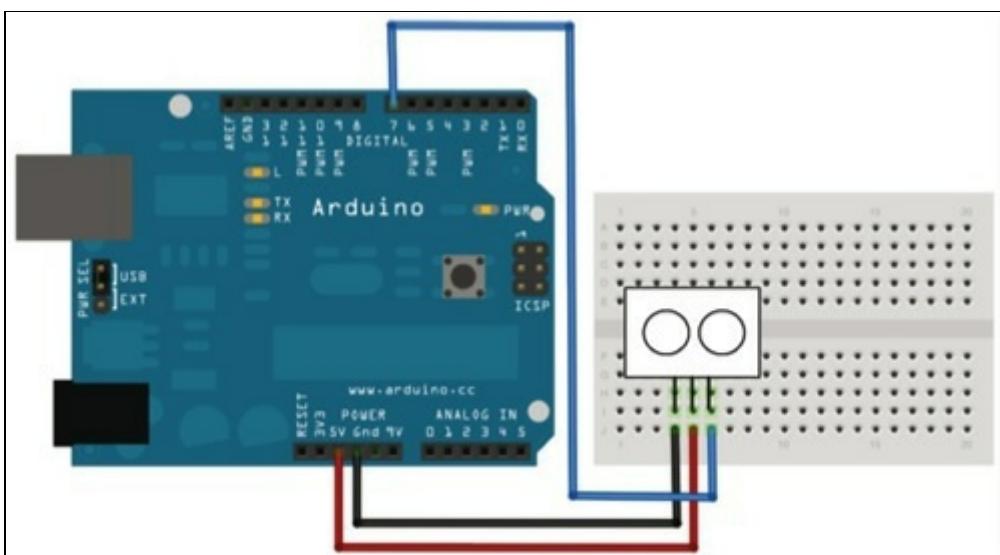


Figura 5.3 Circuito di base per il collegamento del sensore PING))).

Per mettere in funzione il circuito dovete istruire la scheda in modo che possa comunicare con il sensore PING))) tramite le istruzioni che seguono.

ultrasonic/simple/simple.pde

```
Riga 1  const unsigned int PING_SENSOR_IO_PIN = 7;
-  const unsigned int BAUD_RATE = 9600;
-
-  void setup() {
5    Serial.begin(BAUD_RATE);
-  }
-
-  void loop() {
-    pinMode(PING_SENSOR_IO_PIN, OUTPUT);
10   digitalWrite(PING_SENSOR_IO_PIN, LOW);
-    delayMicroseconds(2);
-
-    digitalWrite(PING_SENSOR_IO_PIN, HIGH);
-    delayMicroseconds(5);
15   digitalWrite(PING_SENSOR_IO_PIN, LOW);
-
-    pinMode(PING_SENSOR_IO_PIN, INPUT);
-    const unsigned long duration = pulseIn(PING_SENSOR_IO_PIN,
HIGH);
-    if (duration == 0) {
20      Serial.println("Warning: We did not get a pulse from
sensor.");
-    } else {
-      Serial.print("Distance to nearest object:");
-      Serial.print(microseconds_to_cm(duration));
-      Serial.println(" cm");
25    }
-
-    delay(100);
-  }
-
30 unsigned long microseconds_to_cm(const unsigned long
microseconds) {
-    return microseconds / 29 / 2;
- }
```

Il programma definisce in primo luogo una costante relativa al pin di IO cui è collegato il sensore PING))). Per collegare il sensore a un altro pin digitale dovete semplicemente modificare la prima riga del programma. Il metodo `setup()` imposta a 9600 il baud rate della porta seriale, in quanto i dati del sensore dovranno essere visualizzati in *Serial Monitor*.

Il funzionamento concreto del sensore ha luogo nel ciclo `loop()` che implementa il protocollo di funzionamento del componente PING))). In base alle sue specifiche (<http://www.parallax.com/dl/docs/prod/acc/28015-PING-v1.5.pdf>), si può controllare il funzionamento del sensore tramite una serie di impulsi; anche i risultati della misura corrispondono a impulsi a larghezza variabile.

Le righe da 9 a 11 impostano il pin di segnale del sensore allo stato `LOW` per 2 microsecondi, in modo da configurare lo stato iniziale della misura e garantire la presenza di impulsi `HIGH` puliti da impiegare nei passi successivi della misurazione. Ricordate che i segnali elettronici sono sempre soggetti a picchi di tensione dovuti all'alimentazione elettrica.

A questo punto si può avviare la misurazione vera e propria. Le righe da 13 a 15 impostano il pin di segnale del sensore allo stato `HIGH` per 5 microsecondi e avviano in questo modo una nuova misurazione. Al termine di questo intervallo di tempo il pin di segnale torna allo stato `LOW`, dato che il sensore risponde con un impulso `HIGH` a larghezza variabile utilizzando il medesimo pin.

Un pin digitale offre poche opzioni per la trasmissione delle informazioni. Potete impostare il pin a uno stato `HIGH` oppure `LOW` e controllare per quanto tempo il medesimo rimane in uno stato particolare. In molti progetti questo tipo di funzionamento è più che sufficiente e ciò vale anche per il progetto che si sta elaborando. Quando il sensore PING))) emette il suo “fischio” a 40 kHz, si imposta il pin di segnale allo stato `HIGH` e lo si riporta allo stato `LOW` quando il sensore riceve l'eco del fischio. In altre parole, il pin di segnale rimane allo stato `HIGH` per un intervallo di tempo esattamente pari al tempo impiegato dal segnale sonoro per raggiungere l'oggetto più vicino e tornare sul sensore. In sintesi si può dire che si utilizza un pin digitale per misurare un segnale analogico. La Figura 5.4 mostra un diagramma tipico che riporta l'attività di un pin digitale collegato a un sensore PING))).

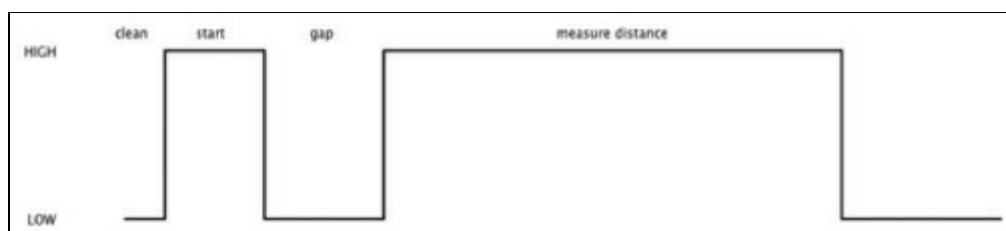


Figura 5.4 Diagramma temporale del segnale del sensore PING))).

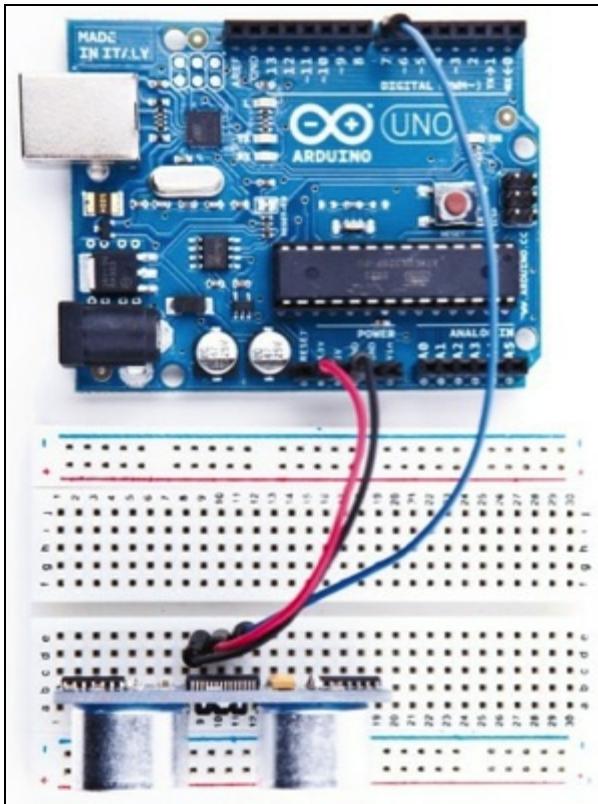


Figura 5.5 Foto del circuito di base per il collegamento del sensore PING)).

Si potrebbe misurare manualmente per quanto tempo il pin rimane nello stato `HIGH`, ma il metodo `pulseIn()` esegue questa operazione al posto nostro alla riga 18 del programma, dopo aver impostato di nuovo il pin di segnale in modalità input. Il metodo `pulseIn()` accetta tre parametri.

- *pin*: numero del pin ove leggere la durata dell'impulso.
- *type*: tipo di impulso da leggere; il tipo può essere `HIGH` oppure `LOW`.
- *timeout*: tempo di attesa dell'impulso, espresso in microsecondi; se non viene rilevato un impulso entro questo intervallo di tempo la funzione `pulseIn()` restituisce il valore 0. Questo parametro è facoltativo e il suo valore di default è pari a un secondo.

È interessante osservare che l'intera procedura utilizza un solo pin per comunicare con il sensore PING)). Presto o tardi vi renderete conto che i pin di IO sono una risorsa limitata delle schede Arduino e apprezzerete ancora di più che il sensore PING)) impieghi un solo pin digitale.

Cercate di occupare sempre il numero minimo di pin quando scegliete i componenti hardware da collegare alla scheda di controllo.

A questo punto non rimane che convertire l'intervallo di tempo misurato in una lunghezza. Il suono viaggia a una velocità di 343 metri al secondo, il che corrisponde a dire che impiega 29,155 microsecondi per percorrere un centimetro. La conversione della misura richiede pertanto di dividere l'intervallo di tempo misurato per 29 e successivamente per 2, dato che il suono ha percorso due volte la distanza da misurare. Ricordate che il segnale viaggia dal sensore all'oggetto per tornare di nuovo al sensore. Il metodo `microseconds_to_cm()` esegue il calcolo previsto per la conversione.

Le specifiche del sensore PING))) dichiarano un tempo di attesa di almeno 200 microsecondi tra due misure successive.

Nel caso di misure che si ripetono velocemente si potrebbe calcolare la durata di una pausa in modo accurato misurando il tempo di esecuzione del programma di controllo, ma nel progetto che si sta studiando questa verifica è superflua in quanto le istruzioni che il metodo `loop()` esegue tra due misure successive impiegano sicuramente un tempo superiore a 200 microsecondi, tenendo conto che l'output dei dati è un'operazione piuttosto dispendiosa in termini di tempo. Nonostante questo il programma aggiunge un piccolo ritardo di 100 microsecondi per posticipare l'esecuzione della misura successiva.

È lecito chiedersi perché si utilizza spesso nel programma la parola chiave `const`. Il software Arduino si basa sul linguaggio C/C++, che raccomanda di dichiarare i valori delle costanti come `const`.

RIFERIMENTO

Consultate a questo proposito il testo di Scott Meyers *Effective C++: 50 Specific Ways to Improve Your Programs* (Addison Wesley Longman, 1997).

La dichiarazione di tipo `const` rende più chiaro il programma e limita la possibilità di commettere errori logici, oltre a consentire al compilatore di ridurre la dimensione del programma.

Nonostante la maggior parte dei programmi Arduino sia costituita da poche istruzioni, lo sviluppo software dei progetti Arduino deve comunque aderire alle buone pratiche di programmazione; pertanto la definizione di un valore costante deve essere dichiarata come tale, ovvero usando `const` e non `#define`. Questo vale anche per altri linguaggi di programmazione e vi dovrete abituare a utilizzare dichiarazioni `final` nei programmi Processing e Java, come verrà ribadito più avanti in questo capitolo.

A questo punto potete esplorare il sensore e studiare vantaggi e svantaggi dell'uso di questo componente. Compilate il programma, caricatelo nella scheda Arduino e aprite *Serial Monitor*. Ricordate di impostare il baud rate a 9600. Dovreste ottenere un output simile al seguente:

```
Distance to nearest object: 42 cm  
Distance to nearest object: 33 cm  
Distance to nearest object: 27 cm  
Distance to nearest object: 27 cm  
Distance to nearest object: 29 cm  
Distance to nearest object: 36 cm
```

Questo genere di output è accompagnato dal led del sensore PING))) che si accende ogni volta che viene effettuata una nuova misurazione.

Verificate le prestazioni del sensore puntando oggetti di dimensioni molto grandi o molto piccole. Provate a rilevare la presenza di oggetti da diverse angolazioni e oggetti che si trovano sopra o sotto il sensore e rilevare la distanza di oggetti che non presentano una superficie omogenea. Provate per esempio a puntare il sensore contro animali di peluche; vedrete che non vengono rilevati come oggetti solidi. (Questo è probabilmente il motivo per il quale i pipistrelli non attaccano gli orsi: non riescono a vederli!)

Sono sufficienti alcuni cavi di collegamento e poche righe di codice per realizzare una prima versione del righello digitale. Il progetto è in grado a questo punto di riportare una misura espressa da un valore intero in centimetri; nel prossimo paragrafo verrà aumentata la sensibilità della misura modificando il software e aggiungendo altri componenti esterni.

Aumentare la sensibilità utilizzando numeri a virgola mobile

Le specifiche del sensore PING))) dichiarano un'accuratezza delle misure per distanze variabili tra 2 centimetri e 3 metri. Questi valori dipendono dalla lunghezza dell'impulso generato dal sensore: la durata minima dell'impulso è di 115 microsecondi, mentre la sua lunghezza massima è di 18,5 millisecondi. La procedura descritta per questo progetto non sfrutta completamente la precisione del sensore, dato che i calcoli vengono effettuati riportando risultati interi. In questo modo è possibile ricavare misure con l'accuratezza di un centimetro. Per ottenere una sensibilità della misura nell'ordine del millimetro occorre impiegare valori numerici a virgola mobile.

In generale l'adozione di valori interi è dettata dalle scarse risorse di memoria e dalle prestazioni della CPU della scheda Arduino, tenendo conto che i calcoli a virgola mobile richiedono un certo dispendio di risorse hardware. A volte, però, è utile sfruttare i valori a virgola mobile e Arduino supporta calcoli di questo genere. Perfezionate il progetto con il programma qui riportato.

ultrasonic/float/float.pde

```
Riga 1  const unsigned int PING_SENSOR_IO_PIN = 7;
      - const unsigned int BAUD_RATE = 9600;
      - const float MICROSECONDS_PER_CM = 29.155;
      - const float MOUNTING_GAP = 0.2;
      5 const float SENSOR_OFFSET = MOUNTING_GAP * MICROSECONDS_PER_CM
* 2;

      -
      -
      - void setup() {
      -     Serial.begin(BAUD_RATE);
      - }
10
      - void loop() {
      -     const unsigned long duration = measure_distance();
      -     if (duration == 0)
      -         Serial.println("Warning: We did not get a pulse from
sensor.");
      15    else
      -         output_distance(duration);
      -     }
      -
      -
      - const float microseconds_to_cm(const unsigned long
microseconds) {
20      const float net_distance = max(0, microseconds -
SENSOR_OFFSET);
      -     return net_distance / MICROSECONDS_PER_CM / 2;
      - }

      -
      - const unsigned long measure_distance() {
25      pinMode(PING_SENSOR_IO_PIN, OUTPUT);
      -     digitalWrite(PING_SENSOR_IO_PIN, LOW);
      -     delayMicroseconds(2);

      -
      -     digitalWrite(PING_SENSOR_IO_PIN, HIGH);
30      delayMicroseconds(5);
      -     digitalWrite(PING_SENSOR_IO_PIN, LOW);
      -
      -     pinMode(PING_SENSOR_IO_PIN, INPUT);
```

```

-     return pulseIn(PING_SENSOR_IO_PIN, HIGH);
35 }

-
-
- void output_distance(const unsigned long duration) {
-   Serial.print("Distance to nearest object: ");
-   Serial.print(microseconds_to_cm(duration));
40   Serial.println(" cm");
- }

```

Questo programma non è molto diverso da quello precedente. In primo luogo si imposta il valore 29,155 per indicare il tempo in microsecondi impiegato dal segnale per percorrere la distanza di un centimetro. Il calcolo della distanza tiene conto dello spazio che esiste tra sensore e circuito che contiene il componente.

Il collegamento del sensore alla breadboard introduce in genere un piccolo gap dovuto alla distanza tra sensore e bordo della breadboard. Questo gap è definito alla riga 5 del programma e viene ripreso nel calcolo successivo della distanza. Il gap è misurato in centimetri e il suo valore è moltiplicato per due, dato che il segnale sonoro percorre lo stesso gap in avanti e all'indietro.

Il metodo `loop()` risulta ora più chiaro, in quanto le operazioni principali del programma sono state riportate in funzioni distinte tra loro. La logica di controllo del sensore è impostata dal metodo `measure_distance()`, mentre il metodo `output_distance()` elabora i valori di output da inviare alla porta seriale. Le modifiche più significative di questa versione del programma riguardano la funzione `microseconds_to_cm()`, che restituisce un valore a virgola mobile ed esegue la sottrazione tra misura del sensore e gap. La funzione `max()` esclude che si possano ottenere valori negativi del risultato finale.

Compilate e caricate il programma. Nella finestra *Serial Monitor* dovreste visualizzare dati di output simili ai seguenti:

```

Distance to nearest object: 17.26 cm
Distance to nearest object: 17.93 cm
Distance to nearest object: 17.79 cm
Distance to nearest object: 18.17 cm
Distance to nearest object: 18.65 cm
Distance to nearest object: 18.85 cm
Distance to nearest object: 18.78 cm

```

Questo esempio è molto più preciso della prima versione del programma. Chi ha già lavorato con numeri in virgola mobile e altri linguaggi di programmazione può chiedersi per quale motivo Arduino arrotondi automaticamente il risultato a due cifre decimali. Il problema è legato al metodo `print()` della classe `Serial`: le ultime versioni della piattaforma Arduino sono in grado di elaborare tutti i tipi di dati, ma il risultato di un calcolo effettuato da questa funzione viene arrotondato a due cifre decimali prima di generare l'output nel caso in cui la variabile considerata sia `float`. Per modificare l'impostazione di default si prenda per esempio l'istruzione `Serial.println(3.141592, 4)`, il cui risultato è riportato come `3.1416`.

Solo la visualizzazione del dato in output è influenzata da questa impostazione e la variabile interna rimane di tipo `float`; sempre a proposito di questo tipo di dati dovete ricordare che al momento in Arduino i valori `float` e `double` hanno lo stesso significato.

In definitiva, qual è il prezzo da pagare quando si impiegano variabili `float`? Il consumo di memoria è di 4 byte, una quantità di memoria pari a quella richiesta da variabili `long`; d'altra parte, i calcoli a virgola mobile sono piuttosto esigenti in termini di CPU e vanno evitati nelle parti del software più critiche in termini di tempi di esecuzione. L'impegno della CPU aumenta a causa delle funzioni di libreria aggiuntive che devono essere collegate al programma per supportare il metodo `float`. La funzione `Serial.print(3.14)` può sembrare innocua ma aumenta in modo considerevole la dimensione del programma. Togliete il commento visibile alla riga 39 e compilate di nuovo il programma per osservare l'effetto di questa funzione. Negli esempi svolti con questo progetto si è riscontrato che il programma richiede 3192 byte in assenza del supporto `float` per il metodo `Serial.print()`, altrimenti sono necessari 4734 byte. È una differenza di ben 1542 byte!

In alcuni casi si può ottimizzare la soluzione supportando l'opzione `float` senza pagare in termini di memoria aggiuntiva. Potete risparmiare una quantità significativa di byte convertendo i valori `float` in interi prima di inviarli a una connessione seriale. Potete per esempio trasferire valori che volete rappresentare con una precisione di due cifre decimali moltiplicando prima i valori per 100, senza dimenticare di dividere successivamente per 100 quando i dati sono stati ricevuti. Questo stratagemma verrà impiegato più avanti, insieme all'arrotondamento dei risultati.

Aumentare la precisione utilizzando un sensore di temperatura

Il supporto di numeri a virgola mobile è senza dubbio un miglioramento del programma, anche se l'aumento di precisione è legato essenzialmente al dato inviato in output; si può ottenere un risultato simile anche modificando i calcoli che producono i risultati numerici. Per ora aumenteremo la sensibilità delle misure senza ricorrere ad artifici software, ma impiegando un nuovo componente hardware, ovvero un sensore di temperatura.

Nei paragrafi precedenti si è partiti dall'ipotesi che il suono viaggia nell'aria a una velocità di 343 m/s, un valore che non è particolarmente accurato in quanto la velocità del suono non è costante e dipende, tra l'altro, dalla temperatura dell'aria. Trascurare la temperatura può comportare un errore abbastanza significativo, che può raggiungere il 12% rispetto al valore esatto. Di seguito è riportata la relazione matematica che lega la velocità effettiva del suono (C) e la temperatura dell'aria (t):

$$C = 331,5 + (0,6 * t)$$

L'applicazione di questa relazione richiede la conoscenza del valore di temperatura espresso in gradi Celsius, una misura che può essere ricavata tramite il sensore temperatura/tensione TMP36 di Analog Devices (<http://tinyurl.com/msard-analog>), un componente economico e semplice da usare.

Il collegamento tra sensore TMP36 e Arduino consiste semplicemente nel connettere i morsetti di massa e di alimentazione con i pin corrispondenti del componente hardware, dopodiché dovete collegare il pin di segnale del sensore con il pin A0, ovvero il pin analogico (ANALOG IN) identificato dal numero 0, come si può vedere nella Figura 5.6.

Il nome dell'azienda che produce il sensore di temperatura suggerisce il fatto che il componente è di tipo analogico: la tensione presente sul pin di segnale dipende linearmente dalla temperatura rilevata. Maggiore è la temperatura, più alto è il valore di tensione fornito tra pin di segnale e massa. Questo componente offre un'ottima opportunità per imparare a utilizzare i pin di IO analogico della scheda Arduino. Innanzitutto conviene studiare il codice che definisce il funzionamento del sensore.

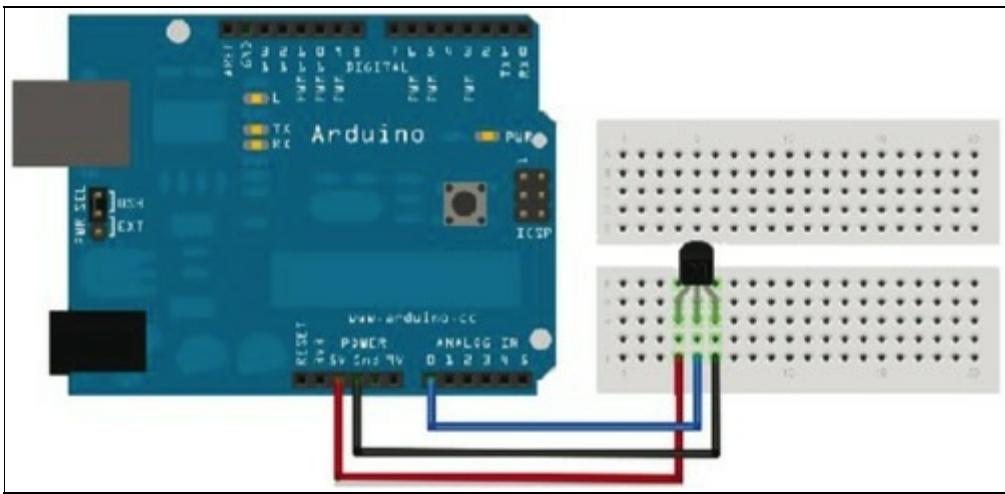


Figura 5.6 Collegamento tra sensore di temperatura e scheda Arduino.

[temperature/sensortest/sensortest.pde](#)

```

Riga 1  const unsigned int TEMP_SENSOR_PIN = 0;
-  const float SUPPLY_VOLTAGE = 5.0;
-  const unsigned int BAUD_RATE = 9600;
-
5   void setup() {
-     Serial.begin(BAUD_RATE);
-   }
-
-   void loop() {
10    Serial.print(get_temperature());
-    Serial.println(" C");
-    delay(1000);
-  }
-
15  const float get_temperature() {
-    const int sensor_voltage = analogRead(TEMP_SENSOR_PIN);
-    const float voltage = sensor_voltage * SUPPLY_VOLTAGE / 1024;
-    return (voltage * 1000 - 500) / 10;
-  }

```

Le prime due righe del programma impostano le costanti relative al pin analogico cui è collegato il sensore e la tensione di alimentazione fornita dalla scheda Arduino. Le istruzioni successive sono il consueto metodo `setup()` seguito da un metodo `loop()` che riporta in output la temperatura coerente una volta al secondo. La logica di funzionamento del sensore è encapsulata nel metodo `get_temperature()`.

Il sensore PING))) necessita di un solo pin digitale che si trova allo stato `HIGH` oppure `LOW`. I pin analogici si comportano in modo diverso e

presentano una tensione che può variare da 0V a un valore di tensione pari all'alimentazione della scheda (in genere 5V).

Potete leggere la tensione presente sui pin analogici di Arduino utilizzando il metodo `analogRead()`, che restituisce un valore numerico compreso tra 0 e 1023, dato che i pin analogici hanno una risoluzione di 10 bit e $1024 = 2^{10}$. La riga 16 del programma legge il valore attuale della tensione fornita dal sensore TMP36.

Rimane da affrontare ancora un problema: dovete convertire il valore restituito da `analogRead()` in un valore di tensione che dipende dalla tensione di alimentazione della scheda Arduino. In genere questa tensione vale 5V ma esistono modelli di scheda Arduino che prevedono un'alimentazione di 3,3V, per esempio la Arduino Pro. Ricordate di modificare la costante `SUPPLY_VOLTAGE` in modo da impostare il valore che corrisponde alla vostra scheda.

Conoscendo la tensione di alimentazione, potete convertire l'output del pin analogico in una tensione dividendo il valore numerico per 1024 e moltiplicando il risultato per il valore di tensione. Questa operazione è eseguita dalle istruzioni alla riga 17.

A questo punto dovete convertire la tensione fornita dal sensore in gradi Celsius. Le specifiche tecniche del sensore suggeriscono di applicare la seguente relazione matematica:

$$T = ((\text{output del sensore in mV}) - 500) / 10$$

Dovete sottrarre 500 millivolt alla tensione di output perché il sensore fornisce sempre una tensione positiva di questo valore, anche quando non sta rilevando la temperatura. La risoluzione del sensore è pari a 10mV, pertanto dovete dividere il risultato per 10. Supponete per esempio che una tensione di 750 millivolt corrisponda a una temperatura di $(750 - 500) / 10 = 25^{\circ}\text{C}$. Questa operazione di calcolo è implementata alla riga 18.

Compilate il programma e caricatelo su Arduino per visualizzare in *Serial Monitor* una serie di dati simile a questa:

```
10.06 C  
26.64 C  
28.62 C
```

```
28.50 C  
28.50 C  
29.00 C  
29.00 C  
28.50 C  
29.00 C
```

L'esecuzione del programma permette di notare che il sensore ha bisogno di un po' di tempo per calibrare la misura; ciononostante i risultati diventano stabili abbastanza rapidamente. A proposito di tempi di lettura, ricordate di inserire un breve ritardo tra due chiamate successive di `analogRead()`, in quanto il sistema di conversione analogico/digitale interno alla scheda di controllo richiede un certo tempo (0,0001 secondi) per eseguire una nuova lettura del segnale.

Il programma prevede peraltro un tempo di attesa pari a un secondo per rendere più agevole la lettura dei dati e perché non ci si aspetta una variazione significativa della temperatura in tempi inferiori a questi. In casi particolari è possibile impostare un tempo di attesa minimo pari a un millisecondo.

A questo punto avete a disposizione due circuiti distinti, uno per la misura della distanza e l'altro per la misura della temperatura. La Figura 5.7 mostra il cablaggio di entrambi i circuiti su una sola breadboard, soluzione riprodotta in foto nella Figura 5.8. Di seguito è indicato il programma che mette in funzione entrambi i sensori.

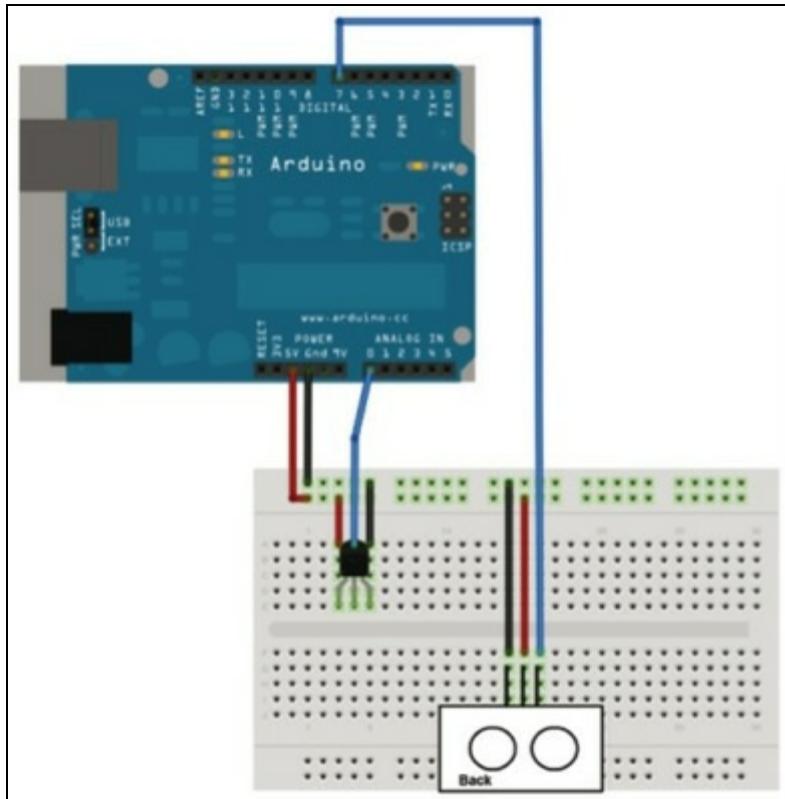


Figura 5.7 I sensori TMP36 e PING)) lavorano all'unisono.

ultrasonic/PreciseSensor/PreciseSensor.pde

```
Riga 1 const unsigned int TEMP_SENSOR_PIN = 0;
-
- const float SUPPLY_VOLTAGE = 5.0;
-
- const unsigned int PING_SENSOR_IO_PIN = 7;
-
- const float SENSOR_GAP = 0.2;
5 const unsigned int BAUD_RATE = 9600;
-
-
- float current_temperature = 0.0;
-
- unsigned long last_measurement = millis();
-
-
10 void setup() {
    Serial.begin(BAUD_RATE);
}
-
-
- void loop() {
```

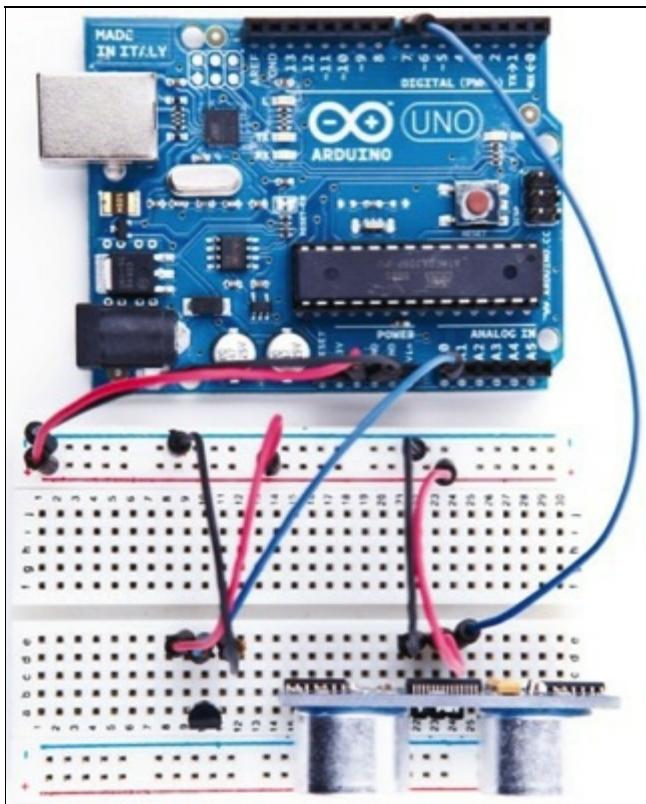


Figura 5.8 Foto del circuito definitivo.

```

15     unsigned long current_millis = millis();
-       if (abs(current_millis - last_measurement) >= 1000) {
-           current_temperature = get_temperature();
-           last_measurement = current_millis;
-       }
20       Serial.print(scaled_value(current_temperature));
-       Serial.print(",");
-       const unsigned long duration = measure_distance();
-       Serial.println(scaled_value(microseconds_to_cm(duration)));
-   }
25
-   long scaled_value(const float value) {
-       float round_offset = value < 0 ? -0.5 : 0.5;
-       return (long) (value * 100 + round_offset);
-   }
30
-   const float get_temperature() {
-       const int sensor_voltage = analogRead(TEMP_SENSOR_PIN);
-       const float voltage = sensor_voltage * SUPPLY_VOLTAGE / 1024;
-       return (voltage * 1000 - 500) / 10;
35   }
-
-   const float microseconds_per_cm() {
-       return 1 / ((331.5 + (0.6 * current_temperature)) / 10000);
-   }
40

```

```

- const float sensor_offset() {
-   return SENSOR_GAP * microseconds_per_cm() * 2;
-
-
45 const float microseconds_to_cm(const unsigned long
microseconds) {
-   const float net_distance = max(0, microseconds -
sensor_offset());
-   return net_distance / microseconds_per_cm() / 2;
- }
-
-
50 const unsigned long measure_distance() {
-   pinMode(PING_SENSOR_IO_PIN, OUTPUT);
-   digitalWrite(PING_SENSOR_IO_PIN, LOW);
-   delayMicroseconds(2);
-
-
55   digitalWrite(PING_SENSOR_IO_PIN, HIGH);
-   delayMicroseconds(5);
-   digitalWrite(PING_SENSOR_IO_PIN, LOW);
-
-
-   pinMode(PING_SENSOR_IO_PIN, INPUT);
60   return pulseIn(PING_SENSOR_IO_PIN, HIGH);
- }

```

Questo codice deriva dalla combinazione dei programmi impiegati per mettere in funzione separatamente i due sensori PING))) e TMP36. È stato necessario apportare poche modifiche.

- La costante `MICROSECONDS_PER_CM` è stata sostituita dalla funzione `microseconds_per_cm()` che calcola il tempo in millisecondi impiegato dal segnale sonoro per percorrere un centimetro; il calcolo dipende dalla temperatura corrente misurata dal sistema di controllo.
- La temperatura non cambia con continuità o rapidamente, pertanto si può evitare di misurarla in modo continuo ma solo una volta al secondo. La funzione `millis()` visibile alla riga 8 rileva quanti millisecondi sono trascorsi da quando è stata avviata la scheda Arduino. Alle righe da 15 a 19 il programma controlla se è trascorso più di un secondo dall'ultima misura di temperatura: in caso affermativo si effettua una nuova misura.
- In questa versione non si trasferiscono più sulla porta seriale i dati del sensore come numeri in virgola mobile ma si preferisce convertire i valori numerici in interi a scalare. Questa operazione è eseguita dalla funzione `scaled_value()`, che arrotonda un valore `float` con due cifre decimali e lo

converte in un valore `long` dopo averlo moltiplicato per 100. Ogni dato visualizzato deve poi essere interpretato dividendo il valore per 100.

Caricate il programma su Arduino e avvicinate la mano al sensore di temperatura per visualizzare una serie di dati simile a questa:

```
1940,2818  
2914,3032  
3045,34156  
3005,2843  
3045,2476  
3085,2414
```

L'output è costituito da un elenco di valori separati da virgole nel quale il primo valore di ogni coppia rappresenta la temperatura rilevata in gradi Celsius mentre il secondo è la distanza tra Arduino e l'oggetto più vicino, misurata in centimetri. Entrambi i valori devono essere divisi per 100 per ottenere il risultato della misura effettuata dai sensori.

Il progetto include ora due sensori, uno dei quali è collegato a un pin digitale e l'altro a uno analogico. Nel prossimo paragrafo vedrete come trasferire i dati dei sensori su un computer e come impiegare questi dati per realizzare applicazioni che tengano conto delle rilevazioni eseguite nell'ambiente esterno alla scheda di controllo.

Codificare i dati del sensore

La codifica dei dati di un sensore è un problema che deve spesso essere affrontato nei progetti Arduino, dato che in genere le misure dei sensori devono essere raccolte e interpretate da applicazioni installate sui normali computer.

La definizione di un formato dati deve pertanto tenere conto di svariate considerazioni; per esempio, il formato dati non deve sprecare le risorse di memoria della scheda Arduino. In questo progetto si potrebbe tra l'altro utilizzare istruzioni XML per codificare i dati dei sensori:

```
<sensor-data>  
  <temperature>30.05</temperature>  
  <distance>51.19</distance>  
</sensor-data>
```

Ovviamente questa non è una soluzione efficiente in quanto spreca memoria dati per creare la struttura del formato del file XML. Va inoltre considerato che l'applicazione finale deve utilizzare un programma di parsing XML per interpretare i dati da elaborare.

In ogni caso non è necessario passare da un estremo all'altro; in altre parole, dovete utilizzare i formati binari solo quando è assolutamente necessario oppure quando l'applicazione finale deve ricevere comunque dati in forma binaria.

In definitiva, spesso la soluzione migliore è costituita proprio dai formati dati più semplici, per esempio il formato CSV (*Character-Separated Values*).

Trasferire i dati al computer utilizzando Processing

I progetti svolti in questo capitolo prevedono il trasferimento dei dati verso il computer tramite porta seriale. Finora il trasferimento dei dati è stato osservato solo facendo riferimento alla finestra *Serial Monitor* dell'IDE Arduino, e i dati non sono mai stati impiegati in altre applicazioni presenti nel computer.

Sensori sonar per salvare l'ambiente

I ricercatori della Northwestern and University of Michigan hanno realizzato un sistema sonar che impiega microfono e altoparlanti di un computer per rilevare se un computer è in uso o meno

(http://blog.makezine.com/archive/2009/10/using_sonar_to_save_power.html). Il computer spegne automaticamente lo schermo se il sistema sonar rileva il mancato utilizzo del computer, contribuendo in questo modo a risparmiare energia e salvare l'ambiente.

Invece di ricorrere a microfono e altoparlanti siete già in grado di utilizzare un sensore PING))) per realizzare un dispositivo simile a questo. Provate!

In questo paragrafo verrà costruita un'applicazione che visualizza in modo grafico i dati rilevati da un sensore. Il programma implementa una sorta di sonar “invertito” che traccia un piccolo punto sullo schermo che si muove per mostrare la distanza misurata rispetto all’oggetto più vicino, come si può vedere nell’ultima immagine di questo capitolo.

L’applicazione verrà implementata utilizzando il linguaggio di programmazione Processing, e la Figura 5.9 mostra uno schema a blocchi che illustra la struttura del progetto. Il codice Processing è eseguito dal computer, mentre il codice di funzionamento del sensore PING))) è eseguito dalla scheda Arduino. La comunicazione tra Processing e Arduino avviene tramite la porta seriale.

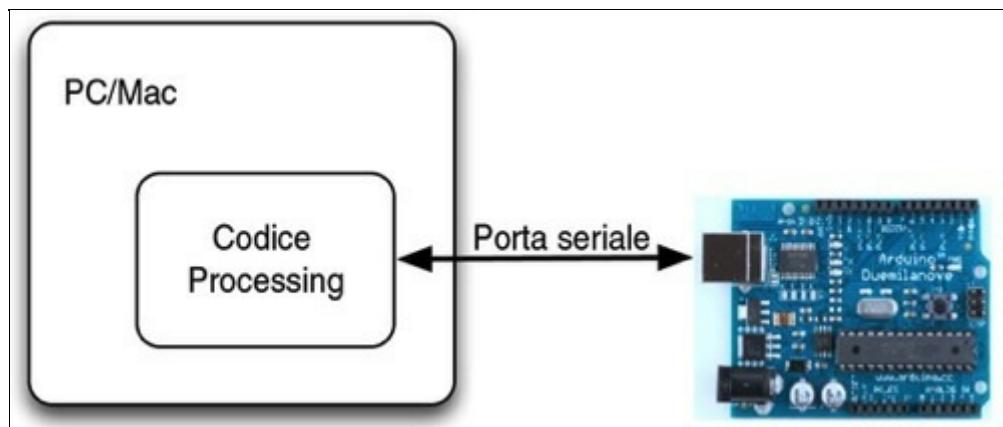


Figura 5.9 Architettura del sistema per il progetto del sonar invertito.

RIFERIMENTO

Consultate per esempio il volume di Ira Greenberg *Processing: Creative Coding and Computational Art*, Apress, 2007.

Processing è un'estensione del linguaggio di programmazione Java dedicato in particolare alla computer grafica. Le istruzioni di questo linguaggio semplificano la creazione di applicazioni multimediali, ovvero di progetti che producono audio e includono immagini animate 2D e 3D. Processing supporta in modo efficiente le interazioni con l'utente ed è disponibile una ricca documentazione di progetto.

Processing fu ideato inizialmente per aiutare gli studenti che non hanno esperienze dirette di programmazione ma desiderano utilizzare computer e altri dispositivi elettronici per realizzare prodotti interattivi. Proprio per questo motivo è semplice da apprendere e molto intuitivo. Sono molti gli utenti che impiegano questo linguaggio di programmazione anche per attività avanzate, che possono riguardare in particolare le presentazioni grafiche di dati.

Potete scaricare liberamente Processing all'indirizzo (<http://processing.org/download/>). Il programma è costituito da un semplice file di installazione disponibile per tutti i sistemi operativi più diffusi e si presenta con una finestra simile a quella mostrata nella Figura 5.10. Ha un aspetto familiare, vero? Non si tratta di una coincidenza, visto che l'IDE di Arduino è derivata proprio dall'IDE di Processing. Invece di scrivere un nuovo ambiente di sviluppo software a partire da zero il team di Arduino ha preferito modificare l'IDE di Processing e per questo motivo i due ambienti di programmazione risultano così simili. Le due IDE condividono per esempio l'estensione dei file ._{pde} (*Processing Development Environment*) per i progetti.



Figura 5.10 L'IDE di Processing è la base dell'IDE di Arduino.

L'impiego di Processing come punto di partenza per lo sviluppo di software Arduino permette di avere a disposizione un'ottima IDE ben collaudata; la combinazione tra Processing e Arduino presenta ulteriori vantaggi, che possono essere sintetizzati dalle considerazioni che seguono.

- Arduino semplifica l'elaborazione dati delle schede di controllo, mentre Processing semplifica la realizzazione di applicazioni multimediali. Potete pertanto visualizzare i dati rilevati da sensori con soluzioni veramente spettacolari.
- Processing è facile da imparare, in particolare se conoscete il linguaggio Java.
- Processing offre un ottimo supporto alle comunicazioni seriali.

In definitiva, sono molte le ragioni che giustificano l'interesse che si manifesta attorno al linguaggio Processing, che diventano ancora più evidenti per chi lavora con Arduino. Ecco spiegato perché Processing verrà impiegato in molti esempi di progetto illustrati in questo libro.

Rappresentazione dei dati del sensore

Conviene innanzitutto prendere in considerazione una classe Processing che rappresenta i dati correnti che un sensore ha riportato da una scheda Arduino e reso disponibili sulla porta seriale. Aprite un nuovo file nell'IDE Processing e scrivete le istruzioni riportate di seguito.

[ultrasonic/InvertedSonar/SensorData.pde](#)

```
class SensorData {  
    private float temperature;  
    private float distance;  
  
    SensorData(float temperature, float distance) {  
        this.temperature = temperature;  
        this.distance = distance;  
    }  
  
    float getTemperature() {  
        return this.temperature;  
    }  
  
    float getDistance() {  
        return this.distance;  
    }  
}
```

Chi conosce Java o C++ avrà già riconosciuto le caratteristiche della classe `SensorData`. Questa classe incapsula un valore di temperatura e una distanza come valori numerici in virgola mobile e permette di accedere a questi dati tramite i metodi accessori `getTemperature()` e `getDistance()`. Potete creare nuovi oggetti `SensorData` utilizzando il costruttore corrispondente e passando come parametri la temperatura e la distanza rilevate dai sensori.

Processing è un linguaggio orientato agli oggetti che permette di definire nuove classi utilizzando la parola `class`. Le classi sono identificate da un nome e contengono dati (chiamati spesso *attributi* o *proprietà*) e funzioni (chiamate anche *metodi*). La classe `SensorData` `class` contiene due attributi denominati `temperature` e `distance`, entrambi di tipo `float` e dichiarati, inoltre, di tipo `private`. In questo modo possono accedere a questi dati solo i membri della classe `SensorData`. Questa impostazione corrisponde alle convenzioni adottate da un tipico linguaggio di programmazione a oggetti ed esclude la possibilità di avere effetti indesiderati, oltre a semplificare le

modifiche successive di un programma. Una classe non deve mai esporre i propri elementi interni.

La classe prevede metodi pubblici per impostare e ricavare i valori degli attributi e include tre metodi: `SensorData()`, `getTemperature()` e `getDistance()`. Chi programma in Java e in C++ avrà notato che in Processing ogni elemento è pubblico, a meno che venga specificato diversamente. Un metodo che ha lo stesso nome della classe si chiama *costruttore* e può essere impiegato per creare e inizializzare nuovi oggetti di una determinata classe. I costruttori non prevedono la restituzione di valori ma possono specificare una serie di parametri; per esempio, in questo progetto i costruttori accettano due argomenti, che utilizzano per inizializzare due attributi.

Occorre però non trascurare un piccolo problema: i parametri del metodo hanno gli stessi nomi degli attributi della classe. Cosa succede allora se i parametri dei metodi vengono assegnati agli attributi tramite istruzioni simili a quelle che seguono?

```
temperature = temperature;  
distance = distance;
```

Semplice, ogni parametro del metodo è assegnato a se stesso, il che equivale a non effettuare alcuna operazione concreta. Per questo motivo si utilizza la parola chiave `this` che fa riferimento alla classe in modo da distinguere tra parametri dei metodi e attributi delle classi. In alternativa si possono utilizzare nomi differenti per indicare parametri e attributi, ma in genere si preferisce aggiungere la parola chiave `this`.

Dopo il costruttore, il programma definisce i metodi `getTemperature` e `getDistance`, le cui istruzioni sono abbastanza simili in quanto si devono dichiarare tra parentesi il tipo di dati restituito dal metodo (`float`), il nome del metodo e un elenco di parametri. In questo progetto l'elenco dei parametri è vuoto e i metodi restituiscono il valore corrente degli attributi corrispondenti grazie all'utilizzo della parola chiave `return`. Questa parola chiave interrompe l'esecuzione del metodo e restituisce i suoi argomenti alla funzione che ha chiamato il metodo.

A questo punto il programma crea e inizializza nuovi oggetti `SensorData`:

```
SensorData sensorData = new SensorData(31.5, 11.76);
```

Questa istruzione crea un nuovo oggetto `SensorData` chiamato `sensorData` che imposta `temperature` a 31.5 e `distance` a 11.76. Per leggere questi valori è sufficiente utilizzare i corrispondenti metodi “get”:

```
sensorData.getTemperature(); // -> 31.5  
sensorData.getDistance(); // -> 11.76
```

Dato che `getTemperature()` e `getDistance()` sono membri della classe `SensorData`, è possibile chiamarli solo utilizzando una istanza della stessa classe. L’istanza in questione si chiama `sensorData`, e per chiamare i metodi “get” si deve scrivere il nome dell’istanza seguito da un punto e dal nome del metodo.

Dopo aver memorizzato i dati dei sensori, il programma procede con le istruzioni indicate nel prossimo paragrafo.

Realizzare le fondamenta dell’applicazione

In questo paragrafo verrà scritto il codice richiesto dall’applicazione per importare una serie di librerie e per definire costanti e variabili. Le istruzioni sono riportate di seguito.

[ultrasonic/InvertedSonar/InvertedSonar.pde](#)

```
import processing.serial.*;  
  
final int WIDTH = 1000;  
final int HEIGHT = 1000;  
final int xCenter = WIDTH / 2;  
final int yCenter = HEIGHT / 2;  
final int LINE_FEED = 10;  
  
Serial arduinoPort;  
SensorData sensorData;  
int degree = 0;  
int radius = 0;
```

La comunicazione di Arduino tramite porta seriale richiede l’importazione del supporto Processing per la comunicazione seriale, definita nelle prima riga del programma. L’istruzione `import` importa tutte le classi del pacchetto `processing.serial` e le mette a disposizione del programma.

L'applicazione del progetto prevede l'uso di una schermata di 1000×1000 pixel, risoluzione che deve essere definita impostando larghezza, altezza e centro della schermata. La costante `LINE_FEED` contiene il valore ASCII di un carattere linefeed, valore che verrà utilizzato più avanti nel programma per interpretare i dati inviati dalla scheda Arduino.

A questo punto il programma imposta le variabili globali indicate di seguito, un'opzione gradita a chi programma in Java: proprio così, Processing accetta la definizione di variabili globali!

- `arduinoPort`: istanza della classe Processing `Serial` che si ricava dal pacchetto `processing.serial` importato in precedenza e che incapsula le comunicazioni seriali con la scheda Arduino.
- `sensorData`: dati correnti del sensore che sono trasferiti da Arduino all'applicazione. Si fa riferimento alla classe `SensorData` definita in precedenza in questo capitolo.
- `degree`: questa variabile visualizza su un cerchio la distanza corrente dell'oggetto più vicino memorizzando l'angolazione ove si trova l'oggetto rispetto al cerchio tracciato sullo schermo. I valori ammessi sono compresi tra 0 e 359.
- `radius`: distanza corrente dell'oggetto più vicino, valore interpretato come raggio del cerchio su cui riportare la posizione dell'oggetto.

Implementare la comunicazione seriale in Processing

Di seguito sono riportate le funzioni che leggono i dati dalla porta seriale dove è collegata la scheda Arduino e che interpretano i dati inviati dalla stessa.

[ultrasonic/InvertedSonar/InvertedSonar.pde](#)

```
Riga 1 void setup() {  
    -   size(WIDTH, HEIGHT);  
    -   println(Serial.list());  
    -   String arduinoPortName = Serial.list()[0];  
    5    arduinoPort = new Serial(this, arduinoPortName, 9600);  
    -   arduinoPort.bufferUntil(LINE_FEED);  
    - }  
    -  
    - void serialEvent(Serial port) {  
    10    sensorData = getSensorData();  
    -    if (sensorData != null) {
```

```

-
-     println("Temperature: " + sensorData.getTemperature());
-     println("Distance: " + sensorData.getDistance());
-     radius = min(300, int(sensorData.getDistance() * 2));
15    }
-
-
-    SensorData getSensorData() {
-        SensorData result = null;
20        if (arduinoPort.available() > 0) {
-
-            final String arduinoOutput =
arduinoPort.readStringUntil(LINE_FEED);
-            result = parseArduinoOutput(arduinoOutput);
-
-        }
-
-        return result;
25    }
-
-
-    SensorData parseArduinoOutput(final String arduinoOutput) {
-        SensorData result = null;
-        if (arduinoOutput != null) {
30        final int[] data = int(split(trim(arduinoOutput), ','));
-
-        if (data.length == 2)
-            result = new SensorData(data[0] / 100.0, data[1] /
100.0);
-
-    }
-
-    return result;
35}

```

La funzione `setup()` è uno dei metodi standard di Processing e assume lo stesso significato dell'omonima funzione di Arduino. L'ambiente runtime di Processing chiama una sola volta questa funzione in fase di avvio dell'applicazione, allo scopo di inizializzarla. Il metodo `size()` genera una nuova schermata caratterizzata da larghezza e altezza definite dal programma; a questo proposito potete consultare una ricca documentazione online che spiega il funzionamento di tutte le classi Processing (<http://processing.org/reference/>).

Dopo aver inizializzato la schermata dell'applicazione si procede all'impostazione delle comunicazioni seriali. In primo luogo si visualizza un elenco dei dispositivi seriali collegati al computer utilizzando l'istruzione `Serial.list()`, poi si imposta il nome della porta seriale da utilizzare in base alla prima voce dell'elenco. Questa configurazione potrebbe non corrispondere alla scelta corretta, pertanto dovete impostare direttamente nel codice il nome della porta seriale impiegata dal sistema di controllo

oppure esaminare l'elenco delle porte disponibili e stabilire la posizione cui corrisponde la porta che vi interessa.

Alla riga 5 viene creato un nuovo oggetto `Serial` relativo all'applicazione (motivo per il quale compare l'attributo `this`). Questa istruzione richiede come parametri il nome della porta seriale ricavato in precedenza dall'elenco di porte disponibili e imposta il baud rate a 9600. Potete impostare comunicazioni più rapide semplicemente modificando il parametro di baud rate di questa istruzione e nel progetto Arduino.

Infine, il programma dice all'oggetto `Serial` che deve notificare la presenza di nuovi dati seriali solo quando rileva un nuovo carattere di linefeed, cui corrisponde la trasmissione di un nuovo insieme di dati da parte della scheda Arduino.

In questa applicazione si è scelto di adottare un modello asincrono di elaborazione dei dati; in altre parole, non esiste un ciclo di istruzioni che ricerca la presenza di nuovi dati ma il programma riceve sulla porta seriale una notifica relativa alla presenza di nuovi dati, o più precisamente riceve la notifica che è stato ricevuto un nuovo carattere di linefeed. In questo modo potete modificare lo stato dell'applicazione in tempo reale ed escludete eventuali ritardi tra l'arrivo dei dati e l'aggiornamento della grafica a video.

Non appena arrivano nuovi dati, il programma chiama automaticamente il metodo `serialEvent()` e sulla porta seriale vengono trasferiti i dati. Il progetto fa riferimento a una sola porta seriale, pertanto si può ignorare il parametro di questa istruzione. La funzione `getSensorData()` legge i dati e, se rilevati correttamente, il programma li visualizza su console affinché possano essere esaminati, oltre a impostare il raggio che corrisponde alla distanza misurata dalla scheda di controllo. La visualizzazione dei dati è ottimizzata moltiplicando la distanza per due ed escludendo i valori maggiori di 300 centimetri.

L'implementazione di `getSensorData()` è abbastanza semplice. Alla riga 20 si controlla innanzitutto che siano disponibili nuovi dati sulla porta seriale. Questo controllo può sembrare ridondante, dato che il metodo viene chiamato solo quando sono disponibili nuovi dati, ma la ridondanza permette di riutilizzare il programma anche in un contesto che prevede dati sincroni, quando il controllo è assolutamente necessario. Il programma

legge i dati fino a trovare un carattere di linefeed e trasferisce i risultati alla funzione `parseArduinoOutput()`.

L'analisi dei dati di output è facilitata dalla presenza del metodo `split()` di Processing. Alla riga 20 si può notare l'istruzione che distingue i singoli elementi dei dati Arduino separati da una virgola; il metodo `trim()` si occupa di rimuovere i caratteri vuoti all'inizio e alla fine della serie di dati. Questa istruzione restituisce un array costituito da due elementi che contengono la rappresentazione di testo di due valori interi. Le stringhe di testo sono convertite in numeri interi utilizzando la funzione `int()`. È interessante notare che in questo programma la funzione `int()` riceve un array che contiene due stringhe e restituisce un array che contiene due valori `int`.

Esiste sempre la possibilità di ricevere una riga di testo incompleta da Arduino, per esempio perché la comunicazione seriale ha inizio in una posizione arbitraria dei byte; è quindi preferibile verificare l'esistenza di due valori accettabili come dati del sensore. In caso affermativo il programma crea un nuovo oggetto `SensorData` inizializzato con il valore di temperatura e di distanza, dopo aver diviso per 100 entrambi i dati.

Queste sono le istruzioni richieste per leggere in modo asincrono i dati della scheda Arduino. A questo punto i dati del sensore sono letti ogni volta che si rendono disponibili e le variabili `sensorData` e `radius` vengono aggiornate automaticamente.

Visualizzare i dati del sensore

Dopo aver predisposto la comunicazione seriale tra computer e scheda Arduino si può visualizzare la distanza misurata rispetto all'oggetto più vicino al sensore utilizzando il programma indicato di seguito.

[ultrasonic/InvertedSonar/InvertedSonar.pde](#)

```
Riga 1 void init_screen() {  
-     background(255);  
-     stroke(0);  
-     strokeWeight(1);  
5      int[] radius_values = { 300, 250, 200, 150, 100, 50 };  
-      for (int r = 0; r < radius_values.length; r++) {  
-          final int current_radius = radius_values[r] * 2;  
-          ellipse(xCenter, yCenter, current_radius, current_radius);  
-      }  
}
```

```

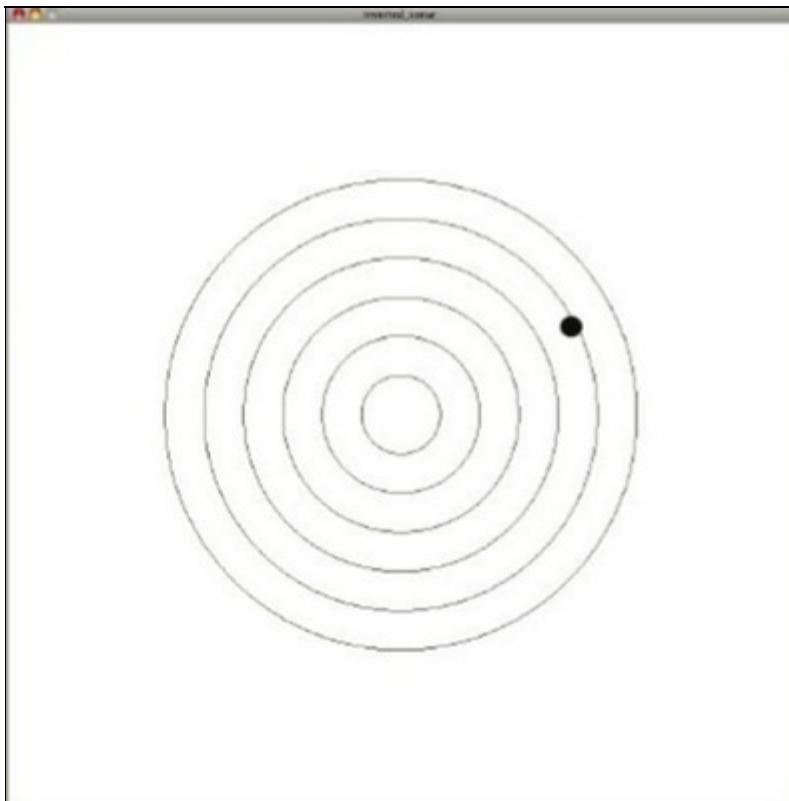
10     strokeWeight(10);
-
-
-
- void draw() {
-     init_screen();
15     int x = (int)(radius * Math.cos(degree * Math.PI / 180));
-     int y = (int)(radius * Math.sin(degree * Math.PI / 180));
-     point(xCenter + x, yCenter + y);
-     if (++degree == 360)
-         degree = 0;
20 }

```

La funzione `init_screen()` pulisce la schermata, dopodiché il programma imposta nella riga 2 uno sfondo bianco. La funzione `stroke(0)` definisce il colore del tratto da disegnare (nero) mentre lo spessore del tratto è di 1 pixel. A questo punto il programma traccia sei cerchi concentrici rispetto al centro della schermata. Questi cerchi facilitano la lettura della distanza rilevata dal sensore PING)). Il programma imposta poi lo spessore del tratto da disegnare a 10 pixel, in modo da visualizzare il sensore come un singolo punto abbastanza marcato ed evidente.

Processing chiama automaticamente il metodo `draw()` con un determinato valore di frame rate; la frequenza di aggiornamento di default corrisponde a 60 frame al secondo.

Questo metodo è equivalente alla funzione `loop()` di Arduino. In questo progetto si inizializza la schermata e si calcolano le coordinate per un cerchio. Il raggio di questo cerchio dipende dalla distanza dell'oggetto da Arduino, pertanto si visualizza un punto che si muove lungo il cerchio. La distanza dal centro del cerchio dipende dalla misurazione effettuata con il sensore PING)).



Alla fine di questo capitolo sapete che esistono due tipi di sensori: digitali e analogici. Avete imparato a collegare entrambi i tipi di sensori alla scheda Arduino e sapete come trasferire al computer le misure rilevate dai sensori. Lavorare con questi due tipi di IO è fondamentale nel physical computing e quasi tutti i progetti si basano sugli elementi che avete studiato in questo capitolo, a prescindere dalla complessità del progetto.

Divertirsi con i sensori

Un sensore a ultrasuoni permette di rilevare con facilità la presenza di una persona vicina. Le applicazioni utili sono molte: per esempio, potete aprire automaticamente una porta non appena qualcuno vi si avvicina. In alternativa, potete sfruttare la tecnologia più avanzata per puro divertimento. Cosa ne dite di giochi per Halloween, per esempio una zucca che “spara” stelle filanti da una bomboletta spray ogni volta che qualcuno attraversa una linea invisibile (<http://www.instructables.com/id/Arduino-controlled-Silly-String-shooter/>)? Potrebbe essere una bella sorpresa per la prossima festa tra amici che potete realizzare utilizzando il sensore PING)).

Cosa fare se non funziona?

Verificate di aver collegato correttamente i componenti sulla breadboard (si veda il Capitolo 3). Prestate particolare attenzione al collegamento dei sensori PING))) e TMP36, dato che si tratta probabilmente della prima volta che utilizzate questo genere di componenti.

Se si manifestano errori nel software, a prescindere che si tratti di codice Processing oppure Arduino, potete sempre scaricare il codice dal sito web dedicato al libro e verificare di nuovo il funzionamento del progetto.

Per risolvere problemi di comunicazione seriale controllate con molta attenzione la configurazione della porta seriale e del tipo di scheda Arduino. Può capitare di dover collegare Arduino a una porta diversa da quella prevista; in questo caso dovete modificare l'indice 0 nell'istruzione

`arduinoPort = new Serial(this, Serial.list()[0], 9600)` in base alla nuova configurazione. Verificate inoltre che il parametro di baud rate nel codice Processing e in *Serial Monitor* coincida con il baud rate utilizzato nel codice Arduino. Controllate che la porta seriale non venga bloccata da un'altra applicazione, per esempio da un'altra finestra *Serial Monitor* che avete dimenticato di chiudere.

Esercizi

- Costruite un sistema automatico di allarme anti-intrusione che visualizza un cartello di “Stop” ogni volta che qualcuno si avvicina al vostro computer (potete trovare un cartello di questo genere all’indirizzo http://en.wikipedia.org/wiki/File:Stop_sign_MUTCD.svg). Fate in modo che l’applicazione risulti il più possibile precisa; per esempio, prevedete un ritardo di attivazione per evitare che il cartello venga visualizzato non appena si avvia l’applicazione.
- La velocità del suono non dipende solo dalla temperatura ma anche da umidità e pressione atmosferica. Eseguite qualche ricerca per scoprire le formule di calcolo più corrette e i sensori più adeguati (consultate per esempio il sito <http://parallax.com>). Sfruttate i risultati della ricerca per realizzare un circuito in grado di ottenere misure di distanza molto accurate.
- Utilizzate una tecnologia alternativa per misurare le distanze, per esempio un sensore a infrarossi. Cercate un sensore adeguato, studiatene le caratteristiche tecniche e realizzate un circuito in grado di visualizzare la distanza tra porta seriale e l’oggetto più vicino a questa.

Game controller sensibile al movimento

È la velocità sbalorditiva con la quale ci si abitua alle nuove tecnologie. Se una decina di anni fa poche persone avrebbero immaginato che saremmo riusciti a sfruttare tecnologie che reagiscono ai nostri movimenti più naturali, oggi ci sembra assolutamente normale ruotare uno smartphone per orientare la visualizzazione di un'immagine da orizzontale a verticale, e anche i bambini più piccoli sanno utilizzare in modo del tutto intuitivo i game controller sensibili al movimento per giocare con la console Nintendo Wii. In questo capitolo vedrete come realizzare con una scheda Arduino dispositivi di questo tipo.

Si utilizzerà uno dei componenti sensibili al movimento più diffusi: l'*accelerometro*. Gli accelerometri rilevano i movimenti in tutte le direzioni; in altre parole comprendono quando vengono mossi verso l'alto, il basso, in avanti, all'indietro, a sinistra oppure a destra. Molti apparecchi elettronici popolari contengono accelerometri, per esempio l'iPhone e i game controller della Nintendo Wii. La loro diffusione massiccia li ha resi anche molto economici.

Gli accelerometri sono sfruttati per realizzare progetti divertenti e altamente professionali. Se pensate a un computer la mente va immediatamente ai game controller oppure ad altri dispositivi di input che gestiscono controlli molto sofisticati, ma potete nello stesso tempo immaginare di usarli in simulazioni o per districarvi in giochi-labirinto o per misurare l'accelerazione in modo più o meno diretto, per esempio quando guidate un'automobile.

Vedrete anche come interpretare correttamente i dati rilevati dall'accelerometro e come ottenere risultati molto accurati. Utilizzerete un accelerometro per realizzare un game controller sensibile al movimento e infine progetterete un gioco che sfrutta questo nuovo dispositivo di controllo.

Cosa serve

1. Una mini breadboard oppure, meglio ancora, una scheda Arduino Prototyping shield corredata di una breadboard di dimensioni ridotte.
2. Un accelerometro ADXL335.
3. Un pulsante.
4. Un resistore da $10k\Omega$.
5. Cavi di collegamento su breadboard.
6. Una scheda Arduino, per esempio un modello Arduino Uno, Duemilanove o Diecimila.
7. Un cavo USB per collegare la scheda Arduino al computer.
8. Un connettore standard da 0,1" a 6 pin.

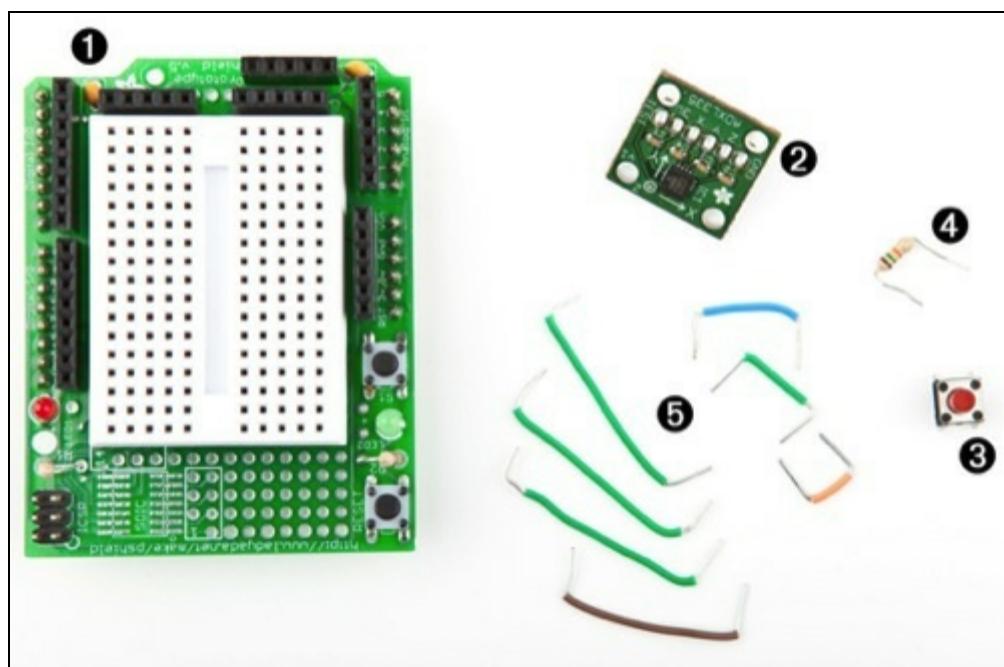


Figura 6.1 Componenti necessari per eseguire i progetti di questo capitolo.

Cablaggio dell'accelerometro

Esistono in commercio molti accelerometri che si contraddistinguono fondamentalmente per il numero di assi spaziali supportati; in genere si trovano componenti che riconoscono due o tre assi spaziali. Nel progetto di questo capitolo verrà impiegato l'accelerometro ADXL335 di Analog Devices, semplice da cablare e da reperire

(<http://www.analog.com/en/sensors/inertial-sensors/adxl335/products/product.html>).

In questo paragrafo vedrete come collegare il componente ADXL335 sulla scheda Arduino per realizzare un programma demo che illustra i dati rilevati direttamente dal sensore. In questo modo avrete modo di

apprendere le caratteristiche tecniche di funzionamento dell'accelerometro e saprete interpretare i dati acquisiti dalla scheda di controllo. In basso a destra nella Figura 6.2 potete vedere un circuito stampato in miniatura su cui è evidente il piccolo circuito integrato (in colore nero) del sensore ADXL335; gli altri componenti del circuito configurano il collegamento tra sensore e scheda Arduino. Nella stessa figura è mostrato in alto un connettore standard da 0,1" a 6 pin. Il sensore presenta sei morsetti di connessione etichettati GND, Z, Y, X, 3V e TEST. Per cablare il sensore su una breadboard dovete saldare il connettore standard ai connettori del circuito stampato; la connessione fisica tra sensore e breadboard è molto semplice e permette di stabilizzare meccanicamente il sensore, che in questo modo non subisce spostamenti accidentali. A sinistra nella foto potete osservare il risultato che si ottiene dopo aver saldato il connettore standard e il circuito stampato del sensore, tenendo presente però che questo circuito stampato non è identico a quello mostrato a destra nella figura, anche se molto simile. Per saperne di più sulla saldatura dei componenti elettronici si veda l'Appendice A.

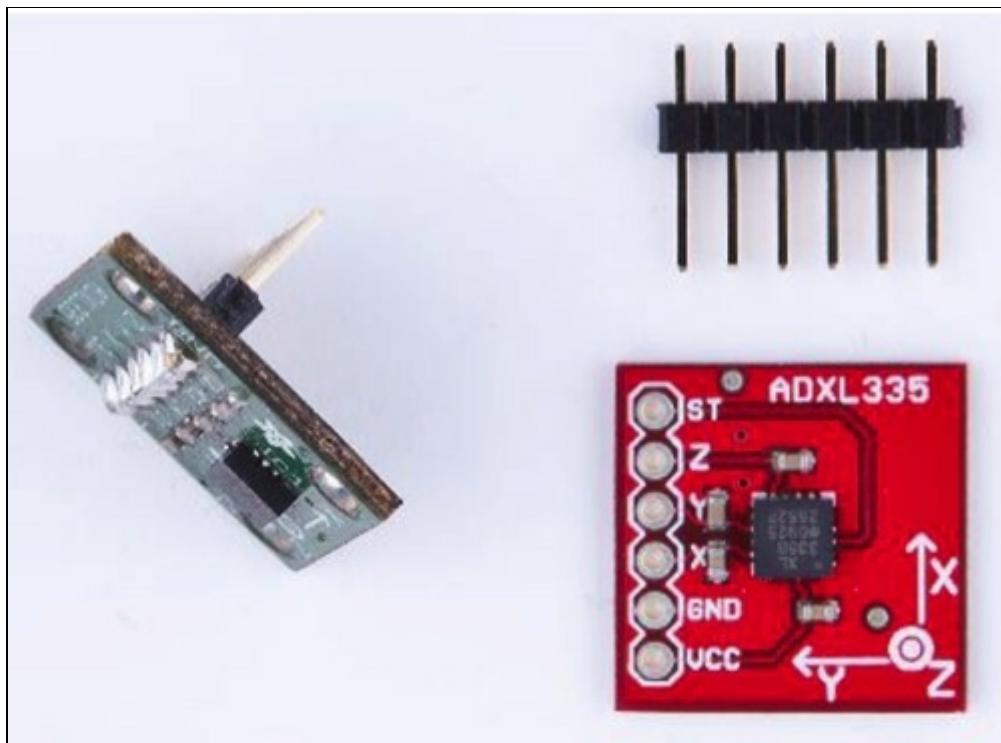


Figura 6.2 Sensore ADXL335 montato su circuito stampato in miniatura.

Potete ignorare per questo progetto il connettore TEST, mentre il significato degli altri connettori dovrebbe risultare ovvio. L'alimentazione del sensore richiede il collegamento tra GND del sensore e pin GND della scheda Arduino, così come il pin 3V del sensore va collegato alla tensione di 3,3 V

fornita da Arduino. I connettori X, Y e Z riportano i dati di accelerazione relativi rispettivamente agli assi x, y e z.

Analogamente al sensore di temperatura TMP36 impiegato nel Capitolo 5, anche l'accelerometro ADXL335 è un dispositivo analogico, pertanto i suoi connettori X, Y e Z devono essere collegati a tre pin analogici della scheda Arduino. Per questo progetto si stabilisce di collegare Z al pin 0, Y al pin 1 e X al pin 2, come si può vedere nella Figura 6.3. Ricordate di verificare con cura il cablaggio corretto di questi pin. Completate il cablaggio del circuito che collega Arduino e accelerometro prima di proseguire.

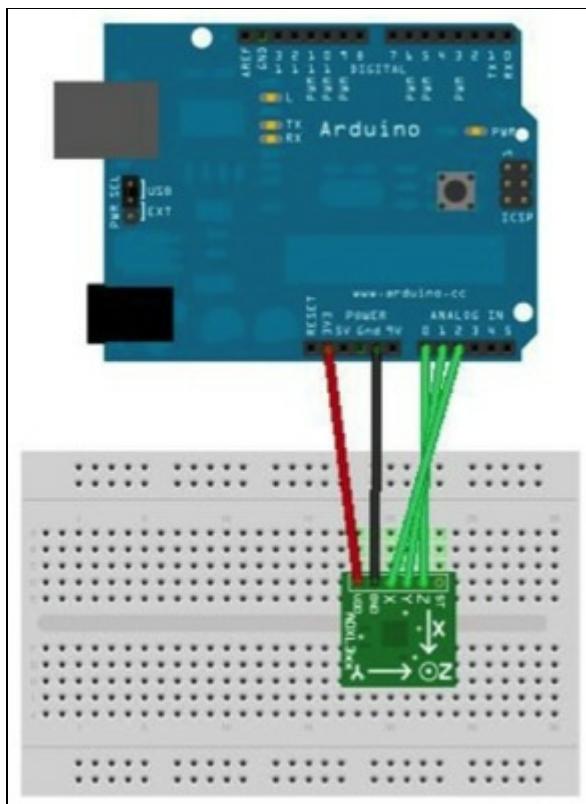


Figura 6.3 Collegamenti tra sensore ADXL335 e scheda Arduino.

Mettere in funzione l'accelerometro

Una strategia decisamente pragmatica che permette di conoscere un nuovo dispositivo di controllo consiste nel metterlo in funzione e osservare i dati che fornisce. Di seguito è riportato un programma che legge i valori di input dell'accelerometro relativi ai tre assi spaziali e li riporta in output sulla porta seriale.

[MotionSensor/SensorTest/SensorTest.pde](#)

```
const unsigned int X_AXIS_PIN = 2;
const unsigned int Y_AXIS_PIN = 1;
const unsigned int Z_AXIS_PIN = 0;
```

```

const unsigned int BAUD_RATE = 9600;

void setup() {
    Serial.begin(BAUD_RATE);
}

void loop() {
    Serial.print(analogRead(X_AXIS_PIN));
    Serial.print(" ");
    Serial.print(analogRead(Y_AXIS_PIN));
    Serial.print(" ");
    Serial.println(analogRead(Z_AXIS_PIN));
    delay(100);
}

```

Questo semplice programma di prova definisce una funzione `setup()` che imposta tre costanti cui corrispondono i tre pin analogici della scheda e inizializza la porta seriale. È interessante osservare che i pin analogici non sono stati impostati a `INPUT` in modo esplicito, dato che questa è la loro impostazione di default.

La funzione `loop()` riporta con continuità i valori letti dai pin analogici in output sulla porta seriale. Aprite la finestra *Serial Monitor* e spostate il sensore inclinandolo lungo i tre assi. Dovreste visualizzare un output simile al seguente:

```

344 331 390
364 276 352
388 286 287
398 314 286
376 332 289
370 336 301
379 338 281

```

Questi valori rappresentano i dati misurati lungo gli assi x, y e z. Provate per esempio a traslare il sensore esclusivamente lungo l'asse x: in questo caso i valori dell'asse x cambiano in funzione del movimento subito dall'accelerometro. Nel prossimo paragrafo verrà studiato il significato di questi valori.

Individuazione e ottimizzazione dei valori limite

La misura delle grandezze fisiche è ben lungi dall'essere perfetta, in particolare quando si fa riferimento alle rilevazioni effettuate da svariati tipi di sensori, compresi gli accelerometri. Questi componenti generano sempre dati compresi tra un minimo e un massimo, ma tali valori possono variare nel tempo. I dati rilevati possono inoltre manifestare una certa oscillazione anche quando si riferiscono a una posizione fissa, evidenziando una serie di “vibrazioni” (*jitter*) che modificano i valori di output anche quando non muovete il sensore. Dovete anche tenere conto che le misure possono cambiare di valore in modo non sempre corretto. In questo paragrafo vedrete come stabilire il valore minimo e massimo misurato da un sensore e come ridurre l’oscillazione delle singole rilevazioni (*anti-jittering*).

È abbastanza facile individuare i valori limite, anche se questa forma di taratura non può avvenire automaticamente. In sostanza dovete leggere in modo continuo l’output del sensore mentre lo state muovendo. La lettura dei dati può essere effettuata utilizzando per esempio il programma indicato di seguito.

[MotionSensor/SensorValues/SensorValues.pde](#)

```
const unsigned int X_AXIS_PIN = 2;
const unsigned int Y_AXIS_PIN = 1;
const unsigned int Z_AXIS_PIN = 0;
const unsigned int BAUD_RATE = 9600;

int min_x, min_y, min_z;
int max_x, max_y, max_z;

void setup() {
  Serial.begin(BAUD_RATE);
  min_x = min_y = min_z = 1000;
  max_x = max_y = max_z = -1000;
}

void loop() {
  const int x = analogRead(X_AXIS_PIN);
  const int y = analogRead(Y_AXIS_PIN);
  const int z = analogRead(Z_AXIS_PIN);

  min_x = min(x, min_x); max_x = max(x, max_x);
  min_y = min(y, min_y); max_y = max(y, max_y);
  min_z = min(z, min_z); max_z = max(z, max_z);

  Serial.print("x(");
  Serial.print(min_x);
```

```

Serial.print("/");
Serial.print(max_x);
Serial.print(", y(");
Serial.print(min_y);
Serial.print("/");
Serial.print(max_y);
Serial.print(", z(");
Serial.print(min_z);
Serial.print("/");
Serial.print(max_z);
Serial.println(")");
}

```

Questo programma dichiara le variabili che contengono il valore minimo e massimo per i tre assi e le inizializza impostando numeri (-1000 e 1000) che sono sicuramente esterni all'intervallo di misura del sensore. La funzione `loop()` misura con continuità l'accelerazione lungo i tre assi e regola di conseguenza i rispettivi valori minimi e massimi.

Compilate e caricate il programma, poi spostate la breadboard su cui è montato il sensore in tutte le direzioni, facendola anche ruotare lungo i tre assi. Traslate il sensore lentamente, poi in modo più rapido, inclinatelo prima piano e poi in modo più deciso. Utilizzate cavi di collegamento piuttosto lunghi e fate attenzione che durante i diversi movimenti la breadboard non perda incidentalmente alcuna connessione.

Proseguite le prove fino a quando ottenete valori stabili sia per il minimo sia per il massimo. Dovreste visualizzare un output simile a questo:

`x(247/649), y(253/647), z(278/658)`

Trascrivete su carta questi valori, che vi serviranno più avanti, quando dovrete eseguire nuove misure con l'accelerometro.

A questo punto ci si può occupare dell'oscillazione delle singole misure. In linea teorica la modifica da effettuare è semplice: invece di leggere direttamente ogni misura rilevata dal sensore, tenete conto di una serie di misure successive e riportate la media. In questo modo le piccole variazioni sono escluse dalla misura riportata dal sensore. Ecco il programma che esegue questi calcoli.

MotionSensor/Buffering/Buffering.pde

```

Riga 1  const unsigned int X_AXIS_PIN = 2;
        - const unsigned int Y_AXIS_PIN = 1;
        - const unsigned int Z_AXIS_PIN = 0;

```

```

- const unsigned int NUM_AXES = 3;
5 const unsigned int PINS[NUM_AXES] = {
- X_AXIS_PIN, Y_AXIS_PIN, Z_AXIS_PIN
- };
- const unsigned int BUFFER_SIZE = 16;
- const unsigned int BAUD_RATE = 9600;
10
- int buffer[NUM_AXES][BUFFER_SIZE];
- int buffer_pos[NUM_AXES] = { 0 };

-
- void setup() {
15   Serial.begin(BAUD_RATE);
- }

-
- int get_axis(const int axis) {
-   delay(1);
20   buffer[axis][buffer_pos[axis]] = analogRead(PINS[axis]);
-   buffer_pos[axis] = (buffer_pos[axis] + 1) % BUFFER_SIZE;
-

-   long sum = 0;
-   for (int i = 0; i < BUFFER_SIZE; i++)
25     sum += buffer[axis][i];
-   return round(sum / BUFFER_SIZE);
- }

-
- int get_x() { return get_axis(0); }
30 int get_y() { return get_axis(1); }
- int get_z() { return get_axis(2); }

-
- void loop() {
-   Serial.print(get_x());
35   Serial.print(" ");
-   Serial.print(get_y());
-   Serial.print(" ");
-   Serial.println(get_z());
- }

```

In primo luogo si definiscono come al solito le costanti relative ai pin di input, cui si aggiunge questa volta anche la costante `NUM_AXES`, che indica il numero di assi da misurare. È presente inoltre un array `PINS` che contiene l'elenco dei pin utilizzati dalla scheda di controllo. Queste impostazioni semplificano eventuali interventi successivi di modifica del programma.

La riga 11 dichiara i buffer per le misure di tutti gli assi, che saranno compilati con i dati rilevati dal sensore in modo da calcolare la media dei valori delle singole misure. È necessario memorizzare la posizione attuale di

ogni buffer, pertanto alla riga 12 si definisce un array di posizioni dei buffer.

La funzione `setup()` deve semplicemente inizializzare la porta seriale, mentre le operazioni principali del programma sono svolte dalla funzione `get_axis()`, che inizialmente prevede un breve ritardo per consentire all'hardware della scheda Arduino di ottenere fisicamente i dati da un pin analogico al successivo, poi legge l'accelerazione misurata sugli assi esaminati dalla scheda e memorizza i valori rilevati nella posizione corrente del buffer di ogni asse. La funzione incrementa la posizione del buffer e la riporta a zero quando si raggiunge la fine del buffer. Il programma restituisce infine il valore medio dei dati rilevati fino a questo punto per ogni asse.

Questo passaggio è fondamentale per il calcolo della misura. Potete valutarne l'effetto collocando il sensore su un tavolo, senza muoverlo. Eseguite il programma più volte cambiando la dimensione del buffer; se non toccate l'output del programma dovrebbe risultare sempre identico. Provate però a impostare `BUFFER_SIZE` con il valore 1: in questo caso dovreste rilevare piccole variazioni dei dati misurati dal sensore, oscillazioni che scompaiono solo quando aumentate in modo significativo la dimensione del buffer.

A questo punto le misure dell'accelerometro sono abbastanza accurate, al punto da permettervi di costruire un game controller che non provocherà fastidi con inattesi e imponderabili variazioni dei dati rilevati.

Realizzare il game controller personalizzato

La realizzazione di un game controller richiede di aggiungere un solo pulsante alla breadboard. Nella Figura 6.4 potete osservare il cablaggio dei componenti hardware. Ricordate di verificare con cura la correttezza dei collegamenti tra i pin della scheda e quelli dell'accelerometro.

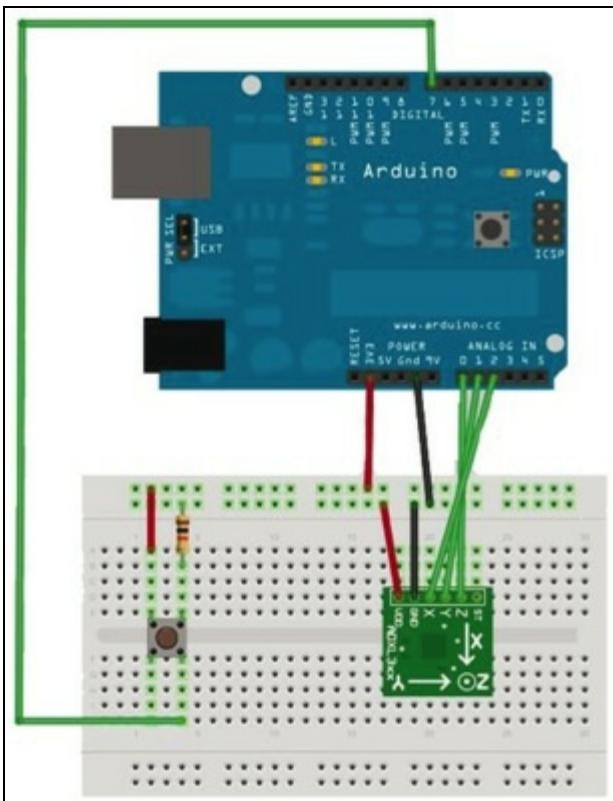


Figura 6.4 Game controller con accelerometro e pulsante.

Questo cablaggio corrisponde allo schema di principio dei collegamenti che fanno funzionare un tipico game controller. Il progetto non prevede di realizzare un contenitore apposito per il game controller, ma fin dall'inizio si può tenere conto in qualche modo dell'ergonomia complessiva dei componenti hardware. Il cablaggio su breadboard è decisamente fragile e non potete pensare di giocare con l'accelerometro collegato in questo modo alla scheda di controllo Arduino. Presto o tardi vi accorgerete che alcuni cavi si scollegano dalla breadboard quando muovete l'accelerometro, che smette così di funzionare.

Per risolvere il problema potete provare a fissare la breadboard alla scheda Arduino con alcuni elastici, una soluzione che può funzionare dal punto di vista della tenuta meccanica dell'hardware, ma poco elegante da vedere e molto difficile da maneggiare.

Una soluzione migliore è data dall'impiego di una scheda Arduino Prototyping shield (Figura 6.5), ovvero di una breadboard in miniatura che può essere collegata in modo sicuro “a cavallo” di una scheda Arduino per realizzare velocemente i prototipi di circuiti anche complessi. La breadboard è circondata da pin che corrispondono a quelli della scheda Arduino, pertanto la connessione della breadboard non richiede più l'inserimento con cavi lunghi e scomodi. Le schede shield permettono di

migliorare la facilità d'uso dei progetti Arduino e sono disponibili per soddisfare svariate esigenze. Potete per esempio trovare schede Ethernet, schede audio o che montano display di vario tipo. Consultate il sito <http://shieldlist.org/> per conoscere l'elenco completo dei modelli Arduino shield.

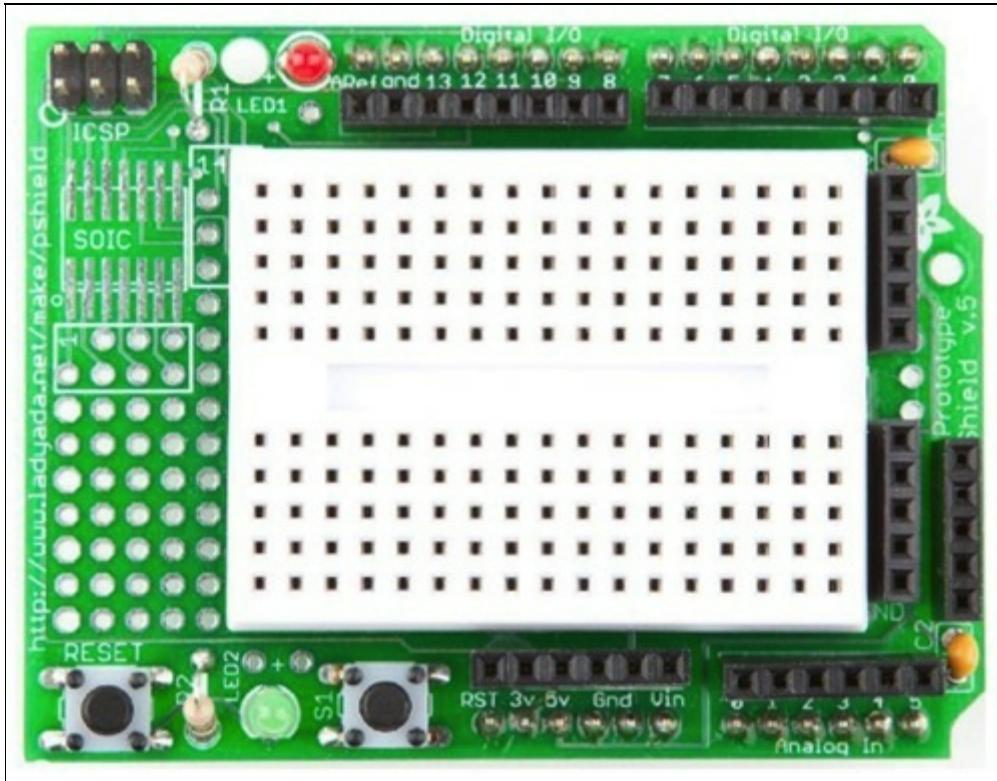


Figura 6.5 Esempio di scheda Arduino Prototyping shield.

Il cablaggio completo di Proto Shield per realizzare il game controller è mostrato nella Figura 6.6.

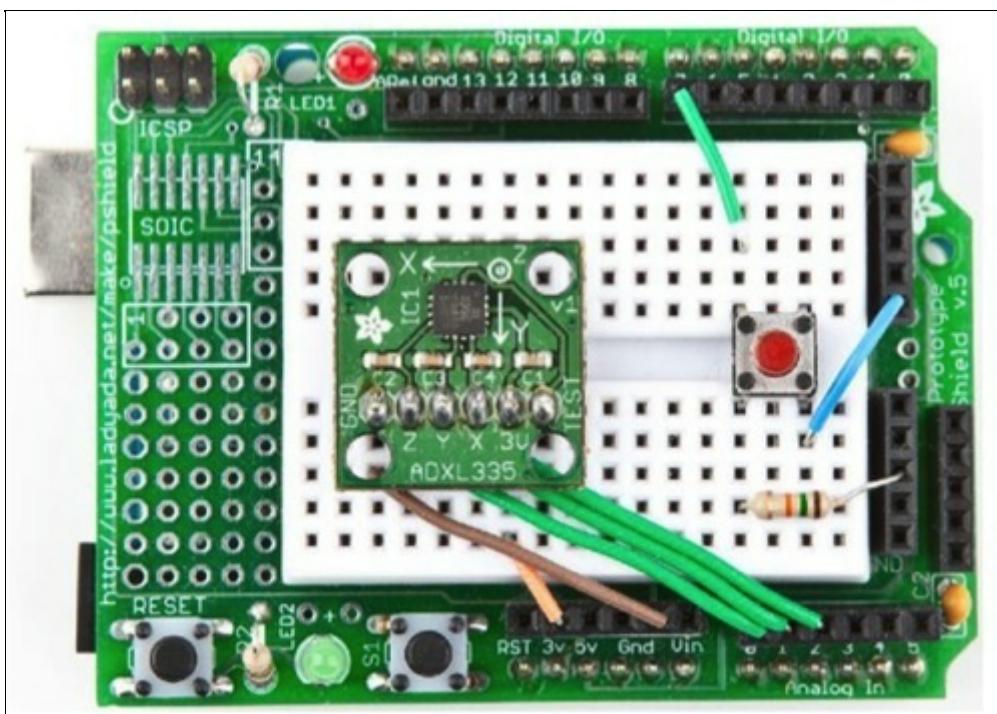


Figura 6.6 Prototipo completo del game controller su una scheda Proto shield.

Dopo aver completato la predisposizione hardware del progetto potete dedicarvi alla versione definitiva del software necessario, che deve supportare la presenza del pulsante esterno e svolgere le operazioni di anti-jittering indicate in precedenza.

MotionSensor/Controller/Controller.pde

```
#include <Bounce.h>

const unsigned int BUTTON_PIN = 7;
const unsigned int X_AXIS_PIN = 2;
const unsigned int Y_AXIS_PIN = 1;
const unsigned int Z_AXIS_PIN = 0;
const unsigned int NUM_AXES = 3;
const unsigned int PINS[NUM_AXES] = {
    X_AXIS_PIN, Y_AXIS_PIN, Z_AXIS_PIN
};
const unsigned int BUFFER_SIZE = 16;
const unsigned int BAUD_RATE = 19200;

int buffer[NUM_AXES][BUFFER_SIZE];
int buffer_pos[NUM_AXES] = { 0 };

Bounce button(BUTTON_PIN, 20);

void setup() {
    Serial.begin(BAUD_RATE);
    pinMode(BUTTON_PIN, INPUT);
}

int get_axis(const int axis) {
    delay(1);
    buffer[axis][buffer_pos[axis]] = analogRead(PINS[axis]);
    buffer_pos[axis] = (buffer_pos[axis] + 1) % BUFFER_SIZE;

    long sum = 0;
    for (int i = 0; i < BUFFER_SIZE; i++)
        sum += buffer[axis][i];
    return round(sum / BUFFER_SIZE);
}

int get_x() { return get_axis(0); }
int get_y() { return get_axis(1); }
int get_z() { return get_axis(2); }
```

```

void loop() {
    Serial.print(get_x());
    Serial.print(" ");
    Serial.print(get_y());
    Serial.print(" ");
    Serial.print(get_z());
    Serial.print(" ");
    if (button.update())
        Serial.println(button.read() == HIGH ? "1" : "0");
    else
        Serial.println("0");
}

```

La funzione antirimbalzo del pulsante è implementata dalla classe `Bounce`, come è stato illustrato nel Capitolo 3. Le altre istruzioni del programma dovrebbero risultare abbastanza chiare; è interessante notare che il baud rate è impostato a 19200 per garantire un trasferimento sufficientemente veloce dei dati del game controller.

Compilate e caricate il codice, aprirete il terminale seriale e provate a giocare con il controller. Muovete il sensore, premete il pulsante e osservate l'output visualizzato dal programma, che dovrebbe essere simile al seguente:

```

324 365 396 0
325 364 397 0
325 364 397 1
325 364 397 0
325 365 397 0
325 365 397 1
326 364 397 0

```

Siete riusciti a costruire un game controller, ma non sarebbe meglio avere a disposizione un videogioco da utilizzare con questo dispositivo? Questo è proprio l'argomento del prossimo paragrafo.

Progettare un nuovo gioco

Per provare il game controller realizzato nel paragrafo precedente potete progettare in Processing un clone del tipico videogame Breakout (http://en.wikipedia.org/wiki/Breakout_%28arcade_game%29 o [http://it.wikipedia.org/wiki/Breakout_\(videogioco\)](http://it.wikipedia.org/wiki/Breakout_(videogioco)) per saperne di più). Obiettivo del gioco è abbattere con una palla il muro di mattoni visibile

nella parte superiore della schermata. Il gioco permette di muovere il game controller lungo l'asse x per spostare la barra orizzontale su cui rimbalza la palla quando raggiunge la parte inferiore della schermata. Di seguito è riportato un esempio di funzionamento del gioco.



Nonostante questo libro non si occupi prevalentemente di videogiochi, è interessante studiare gli elementi interni del programma, perché questo genere di programmazione diventa molto divertente quando si lavora con Processing! Scaricate il software del progetto dal sito del libro (<http://www.pragprog.com/titles/msard>) e giocate con Arduino prima di affrontare lo studio del codice di controllo.

Dato che il game controller deve essere collegato alla porta seriale, il programma si preoccupa innanzitutto di inizializzare la porta di comunicazione.

MotionSensor/Game/Game.pde

```
import processing.serial.*;  
Serial arduinoPort;
```

Le istruzioni successive definiscono alcune costanti che permettono di personalizzare facilmente le funzionalità del gioco.

MotionSensor/Game/Game.pde

```
final int COLUMNS = 7;
```

```

final int ROWS = 4;
final int BALL_RADIUS = 8;
final int BALL_DIAMETER = BALL_RADIUS * 2;
final int MAX_VELOCITY = 8;
final int PADDLE_WIDTH = 60;
final int PADDLE_HEIGHT = 15;
final int BRICK_WIDTH = 40;
final int BRICK_HEIGHT = 20;
final int MARGIN = 10;
final int WIDTH = COLUMNS * BRICK_WIDTH + 2 * MARGIN;
final int HEIGHT = 300;
final int X_AXIS_MIN = 252;
final int X_AXIS_MAX = 443;
final int LINE_FEED = 10;
final int BAUD_RATE = 19200;

```

La maggior parte di questi valori dovrebbe risultare abbastanza chiara, dato che impostano la dimensione degli oggetti che compaiono nella schermata. La costante `PADDLE_WIDTH` definisce la larghezza della barra in pixel, mentre `COLUMNS` e `ROWS` riguardano la disposizione dei mattoni. Sostituite il valore di `X_AXIS_MIN` e di `X_AXIS_MAX` con i valori di minimo e di massimo rilevati con il sensore, in base alle indicazioni fornite nei paragrafi precedenti.

A questo punto il programma stabilisce la modalità di rappresentazione degli oggetti che compongono il gioco.

[MotionSensor/Game/Game.pde](#)

```

int px, py;
int vx, vy;
int xpos = WIDTH / 2;
int[][] bricks = new int[COLUMNS][ROWS];

```

Le coordinate x e y correnti della palla sono memorizzate nelle variabili `px` e `py`, mentre le componenti x e y della velocità sono memorizzate nelle variabili `vx` e `vy`. La posizione x della barra è memorizzata in `xpos`.

`bricks` è un array bidimensionale che contiene lo stato corrente dei mattoni visibili nella schermata. Se un elemento dell'array vale 1 significa che il mattone corrispondente è visualizzato nella schermata, mentre un valore 0 corrisponde a un mattone che è stato abbattuto. Il programma deve infine memorizzare gli stati possibili del gioco.

[MotionSensor/Game/Game.pde](#)

```

boolean buttonPressed = false;
boolean paused = true;
boolean done = true;

```

Prevedibilmente, la variabile `buttonPressed` vale `true` quando si preme il pulsante del game controller, mentre in caso contrario assume il valore `false`. La variabile `paused` segnala se il gioco si trova in pausa; `done` vale `true` quando si termina il livello corrente, ovvero tutti i mattoni sono stati abbattuti.

Ogni programma Processing richiede una funzione `setup()`, che in questo progetto contiene le istruzioni riportate di seguito.

MotionSensor/Game/Game.pde

```

void setup() {
    size(WIDTH, HEIGHT);
    noCursor();
    textFont(loadFont("Verdana-Bold-36.vlw"));
    initGame();
    println(Serial.list());
    arduinoPort = new Serial(this, Serial.list()[0], BAUD_RATE);
    arduinoPort.bufferUntil(LINE_FEED);
}

void initGame() {
    initBricks();
    initBall();
}

void initBricks() {
    for (int x = 0; x < COLUMNS; x++)
        for (int y = 0; y < ROWS; y++)
            bricks[x][y] = 1;
}

void initBall() {
    px = width / 2;
    py = height / 2;
    vx = int(random(-MAX_VELOCITY, MAX_VELOCITY));
    vy = -2;
}

```

La funzione `setup()` inizializza la schermata, nasconde il puntatore del mouse grazie all'opzione `noCursor()` e imposta il font da utilizzare per scrivere i messaggi di output. Ricordate che potete creare un font

utilizzando il menu *Tools > Create Font* di Processing. A questo punto il programma chiama `initGame()` per inizializzare l'array `bricks`, la posizione e la velocità della palla. Per vivacizzare il gioco si è stabilito di impostare con un valore casuale la componente x della velocità, mentre la componente y della velocità vale -2, in modo da far “cadere” la palla con una velocità ragionevole.

Dopo aver effettuato queste inizializzazioni il programma può implementare il ciclo delle istruzioni principali del gioco. Il metodo `draw()` di Processing è la soluzione migliore per gestire tutte le operazioni; per approfondire lo studio delle classi di Processing potete consultare la ricca documentazione offerta da <http://processing.org/reference/>.

MotionSensor/Game/Game.pde

```
void draw() {
    background(0);
    stroke(255);
    strokeWeight(3);

    done = drawBricks();
    if (done) {
        paused = true;
        printWinMessage();
    }

    if (paused)
        printPauseMessage();
    else
        updateGame();

    drawBall();
    drawPaddle();
}
```

L'istruzione `background()` pulisce la schermata e la colora di nero. Il programma prosegue impostando un tratto in colore bianco e con lo spessore di tre pixel, dopodiché disegna i mattoni che devono ancora essere abbattuti. Se non ci sono più mattoni da abbattere il programma mette in pausa il gioco e visualizza il messaggio “You Win!”.

Una pausa di gioco è evidenziata da un messaggio; se il gioco non è in pausa il programma aggiorna lo stato corrente. Infine, il programma

disegna la palla e la barra nelle rispettive posizioni utilizzando le istruzioni che seguono.

MotionSensor/Game/Game.pde

```
boolean drawBricks() {  
    boolean allEmpty = true;  
    for (int x = 0; x < COLUMNS; x++) {  
        for (int y = 0; y < ROWS; y++) {  
            if (bricks[x][y] > 0) {  
                allEmpty = false;  
                fill(0, 0, 100 + y * 8);  
                rect(  
                    MARGIN + x * BRICK_WIDTH,  
                    MARGIN + y * BRICK_HEIGHT,  
                    BRICK_WIDTH,  
                    BRICK_HEIGHT  
                );  
            }  
        }  
    }  
    return allEmpty;  
}  
  
void drawBall() {  
    strokeWeight(1);  
    fill(128, 0, 0);  
    ellipse(px, py, BALL_DIAMETER, BALL_DIAMETER);  
}  
  
void drawPaddle() {  
    int x = xpos - PADDLE_WIDTH / 2;  
    int y = height - (PADDLE_HEIGHT + MARGIN);  
    strokeWeight(1);  
    fill(128);  
    rect(x, y, PADDLE_WIDTH, PADDLE_HEIGHT);  
}
```

È interessante notare che la palla non è altro che un cerchio, mentre mattoni e barra orizzontale sono rettangoli. Il loro aspetto è reso più gradevole dalla presenza di un bordo marcato.

Anche la visualizzazione dei messaggi previsti dal gioco è abbastanza semplice, come si può vedere di seguito.

MotionSensor/Game/Game.pde

```
void printWinMessage() {
```

```

    fill(255);
    textSize(36);
    textAlign(CENTER);
    text("YOU WIN!", width / 2, height * 2 / 3);
}

void printPauseMessage() {
    fill(128);
    textSize(16);
    textAlign(CENTER);
    text("Press Button to Continue", width / 2, height * 5 / 6);
}

```

La funzione `update()` è molto importante dato che aggiorna lo stato del gioco, ovvero controlla le collisioni tra i diversi elementi, muove la palla e così via.

MotionSensor/Game/Game.pde

```

void updateGame() {
    if (ballDropped()) {
        initBall();
        paused = true;
    } else {
        checkBrickCollision();
        checkWallCollision();
        checkPaddleCollision();
        px += vx;
        py += vy;
    }
}

```

La palla esce dal campo e il gioco si interrompe quando il giocatore non la colpisce con la barra orizzontale. A questo punto il giocatore può proseguire solo dopo aver premuto il pulsante. La versione finale del gioco deve prevedere un contatore delle “vite” del giocatore da decrementare ogni volta che questi preme il pulsante; quando il contatore arriva a zero il programma deve visualizzare il messaggio “Game Over”.

Con la palla ancora in gioco il programma deve controllare le collisioni tra questa e gli altri elementi presenti nella schermata. Il controllo riguarda il fatto che la palla abbia colpito uno o più mattoni, oppure una parete o la barra orizzontale. In ogni caso il programma deve calcolare la posizione della palla. Il controllo delle collisioni può sembrare un’operazione complessa ma in realtà abbastanza semplice, poiché deve solo confrontare

le coordinate della palla con le coordinate degli altri elementi presenti nella schermata.

MotionSensor/Game/Game.pde

```
boolean ballDropped() {
    return py + vy > height - BALL_RADIUS;
}

boolean inXRange(final int row, final int v) {
    return px + v > row * BRICK_WIDTH &&
           px + v < (row + 1) * BRICK_WIDTH + BALL_DIAMETER;
}

boolean inYRange(final int col, final int v) {
    return py + v > col * BRICK_HEIGHT &&
           py + v < (col + 1) * BRICK_HEIGHT + BALL_DIAMETER;
}

void checkBrickCollision() {
    for (int x = 0; x < COLUMNS; x++) {
        for (int y = 0; y < ROWS; y++) {
            if (bricks[x][y] > 0) {
                if (inXRange(x, vx) && inYRange(y, vy)) {
                    bricks[x][y] = 0;
                    if (inXRange(x, 0)) // Colpita parte superiore o inferiore di un mattone.
                        vy = -vy;
                    if (inYRange(y, 0)) // Colpita parte sinistra o destra di un mattone.
                        vx = -vx;
                }
            }
        }
    }
}

void checkWallCollision() {
    if (px + vx < BALL_RADIUS || px + vx > width - BALL_RADIUS)
        vx = -vx;

    if (py + vy < BALL_RADIUS || py + vy > height - BALL_RADIUS)
        vy = -vy;
}

void checkPaddleCollision() {
    final int cx = xpos;
    if (py + vy >= height - (PADDLE_HEIGHT + MARGIN + 6) &&
```

```

px >= cx - PADDLE_WIDTH / 2 &&
px <= cx + PADDLE_WIDTH / 2)
{
vy = -vy;
vx = int(
    map(
        px - cx,
        -(PADDLE_WIDTH / 2), PADDLE_WIDTH / 2,
        -MAX_VELOCITY,
        MAX_VELOCITY
    )
);
}
}

```

Più divertimento con le tecnologie sensibili al movimento

Le tecnologie sensibili al movimento sono sempre più diffuse ed economiche; da tempo vengono impiegate per realizzare progetti originali e curiosi. Un esempio molto divertente è dato da Brushduino (<http://camelpunch.blogspot.com/2010/02/blog-post.html>), un dispositivo costruito da un padre per aiutare i propri figli a utilizzare correttamente lo spazzolino da denti. Il componente fondamentale di questo progetto, a prescindere dalla scheda di controllo Arduino, è costituito proprio da un accelerometro a tre assi. Una serie di led collegati a Brushduino indica quale parte della bocca deve essere ancora “passata” con lo spazzolino; ogni volta che uno dei bambini finisce di pulire una parte della bocca viene invece riprodotto un brano musicale del videogioco Super Mario Bros.

Ricordate ad ogni modo che l'accelerometro non è l'unico sensore che permette di rilevare il movimento e di realizzare giochi elettronici innovativi e fantasiosi. Potete per esempio impiegare un tilt sensor per costruire una palla da far saltare in un gioco Hacky Sack interattivo

(http://blog.makezine.com/archive/2010/03/arduino-powered_hacky-sack_game.html). La palla si illumina ed emette un bip ogni volta che la colpite; il gioco si conclude eseguendo una melodia dopo aver colpito la palla per 30 volte.

È interessante notare che il controllo delle collisioni modifica anche la velocità della palla, se necessario.

Dopo aver impostato il movimento della palla il programma deve far muovere anche la barra orizzontale. Il movimento della barra dipende dallo spostamento del game controller lungo l'asse x ed è definito dalle istruzioni che seguono. I dati del controller sono rilevati tramite la porta seriale.

MotionSensor/Game/Game.pde

```

Riga 1 void serialEvent(Serial port) {
-
-    final String arduinoData = port.readStringUntil(LINE_FEED);
-
-
-    if (arduinoData != null) {
5        final int[] data = int(split(trim(arduinoData), ' '));
-
-        if (data.length == 4) {
-
-            buttonPressed = (data[3] == 1);
-
-            if (buttonPressed) {
-
-                paused = !paused;
-
```

```

10         if (done) {
-
-             done = false;
-
-             initGame();
-
-         }
-
15         if (!paused)
-
-             xpos = int(map(data[0], X_AXIS_MIN, X_AXIS_MAX, 0,
WIDTH));
-
-         }
-
-     }
-
20 }

```

Processing chiama la funzione `serialEvent()` ogni volta che si rendono disponibili nuovi dati sulla porta seriale. Il controller trasmette i propri dati una riga alla volta e ogni riga contiene l'accelerazione attuale lungo gli assi x, y e z, cui si aggiunge lo stato del pulsante. I singoli dati sono separati da spazi vuoti. Il metodo `serialEvent()` legge una nuova riga di dati, distingue i singoli elementi grazie agli spazi vuoti e converte le stringhe risultanti in valori `int`. Tutte queste operazioni sono definite nella riga 5 del programma riportato in precedenza.

Le istruzioni verificano la presenza dei quattro attributi richiesti, poi controllano se il giocatore ha premuto il pulsante del game controller. In caso affermativo il programma commuta lo stato di pausa: se il gioco si trova in pausa lo si può riprendere, altrimenti resta fermo. Il programma deve anche verificare se è stata raggiunta la fine del gioco e in questo caso avvia un nuovo gioco dall'inizio.

Il programma deve infine leggere alla riga 17 il valore attuale dell'accelerazione X e confrontare questo valore con le posizioni x ammesse dalla barra. Questa è l'operazione che permette di spostare la barra orizzontale utilizzando il game controller. Anche in questo progetto l'uso del controller prescinde dal fatto che si debba gestire un semplice gioco o elaborare un genere completamente diverso di software: il codice di programmazione deve in ogni caso leggere quattro valori interi dalla porta seriale.

In questo paragrafo avete appreso molte informazioni che riguardano prevalentemente la programmazione di videogiochi piuttosto che i componenti hardware e software di un progetto Arduino. Ora dovrebbe essere evidente che è abbastanza semplice integrare un qualsiasi progetto

elettronico ben strutturato con istruzioni software che ne controllano il funzionamento. In questo capitolo avete analizzato i dati analogici rilevati dall'accelerometro e avete eliminato le vibrazioni indesiderate adottando una tecnica che verrà riproposta in altri progetti elettronici, come si vedrà nel prossimo capitolo.

Creare nuovi giochi con Arduino

Arduino permette di costruire molto più di un game controller; grazie a questa scheda di controllo potete infatti realizzare videogiochi molto interessanti. È sufficiente impiegare adeguate schede di estensione hardware per trasformare Arduino in una console ricca di gadget curiosi

(<http://antipastohw.blogspot.com/2009/02/getting-started-with-gamepack-in-3.html>). I progetti diventano presto piuttosto costosi, ma pensate a quello che potete ricavare da una scheda Arduino collegata a un touch screen OLED da 320×200 pixel, un joystick analogico, un paio di pulsanti e addirittura un motore che riproduce le vibrazioni prodotte dal feedback (http://it.wikipedia.org/wiki/Force_feedback).

Se pensate a una soluzione più economica potete fare riferimento a un clone di Super Mario Bros realizzato con componenti hardware ridotti al minimo (<http://arduino.cc/blog/2010/03/10/super-mario-brothers-with-an-arduino/>); quest'ultimo progetto è un esempio perfetto dell'incredibile libertà creativa offerta dalle schede Arduino.

Altri progetti interessanti

È sufficiente tenere gli occhi aperti per scoprire più applicazioni dell'accelerometro di quelle che potreste mai immaginare. Di seguito sono riportati alcuni esempi di prodotti commerciali e di progetti distribuiti liberamente.

- Il kit iPod Sport di Nike può esservi di aiuto per eseguire i vostri esercizi di ginnastica quotidiani ed è basato sostanzialmente sull'uso di un accelerometro. Potete apprendere molto dallo studio dei suoi componenti interni (http://www.runnerplus.com/read/1-how_does_the_nike_ipod_sport_kit_accelerometer_work/).
- È divertente realizzare per il computer un gioco tipo Marble Maze e controllarne il funzionamento impiegando il game controller illustrato in questo capitolo. Ancora più divertente può essere costruire un vero e proprio labirinto (http://www.electronicsinfoonline.com/New/Everything_Else/marble-maze-that-is-remote-controlled-using-an-accelerometer.html).
- In questo capitolo l'accelerometro è stato utilizzato per ottenere misure dirette dell'accelerazione; in altre parole, viene tenuto in mano e le misure hanno a che fare con lo spostamento effettivo del sensore. Potete però immaginare progetti che richiedano la misura indiretta dell'accelerazione, da impiegare per esempio quando si guida un'automobile

Cosa fare se non funziona?

Il progetto di questo capitolo deve tenere conto di tutte le indicazioni fornite nel Capitolo 5, cui si devono aggiungere le attenzioni particolari richieste dai componenti aggiuntivi, in primo luogo dalla scheda Proto shield. Verificate con cura che venga collocata correttamente a cavallo della scheda Arduino e che nessun connettore scivoli al di fuori del morsetto corrispondente di fissaggio. A volte i connettori risultano leggermente fuori misura, pertanto i collegamenti vanno controllati uno per uno.

Esaminate con cura le saldature del connettore dei pin. Utilizzate una lente di ingrandimento e controllate con molta attenzione i singoli punti di saldatura. Avete usato abbastanza stagno? Ne avete usato troppo e avete provocato un cortocircuito tra due punti che non si devono collegare tra loro?

Esercizi

- Realizzate un mouse personalizzato e originale utilizzando l'accelerometro ADXL335. Il dispositivo deve funzionare sospeso per aria e deve trasmettere segnali relativi all'accelerazione misurata in tempo reale lungo l'asse x e lungo l'asse y. Il progetto deve anche prevedere un tasto sinistro e un tasto destro. Scrivete il codice Processing (potete anche utilizzare un altro linguaggio di programmazione, se preferite) che gestisce il movimento del puntatore del mouse sullo schermo del computer.

Giocherellare con il controller Wii Nunchuk

Una delle attività più intriganti della progettazione elettronica consiste nel prendere un prodotto esistente e giocherellare con hardware e software fino a farlo diventare qualcosa di differente oppure utilizzarlo per uno scopo diverso da quello originale. A volte si deve accedere ai circuiti elettronici interni (annullando la garanzia del prodotto); altre volte potete impiegare il dispositivo integro in un nuovo progetto.

In questo capitolo vedrete come appropriarvi di un controller Nintendo Nunchuk, un’ottima soluzione per giocherellare con l’elettronica dato che il prodotto include un accelerometro a tre assi, un joystick analogico e due pulsanti, oltre a essere economico (dovrebbe costare meno di 20 euro). Tenete anche presente che il design del controller e il facile accesso ai connettori permettono di integrare Nunchuk senza difficoltà nei vostri progetti più sofisticati (Figura 7.1).

Il controller Nunchuk sarà impiegato con Arduino per trasferire al computer i dati rilevati dai suoi sensori. Vedrete come collegarlo alla scheda Arduino e scrivere il software che legge lo stato attuale del controller. Sarete infine in grado di muovere, ruotare e cambiare le dimensioni di un cubo tridimensionale visualizzato sullo schermo del computer agendo su Nunchuk. Queste operazioni non richiedono la presenza di una console Nintendo Wii: l’unico componente hardware necessario è il controller Nunchuk.

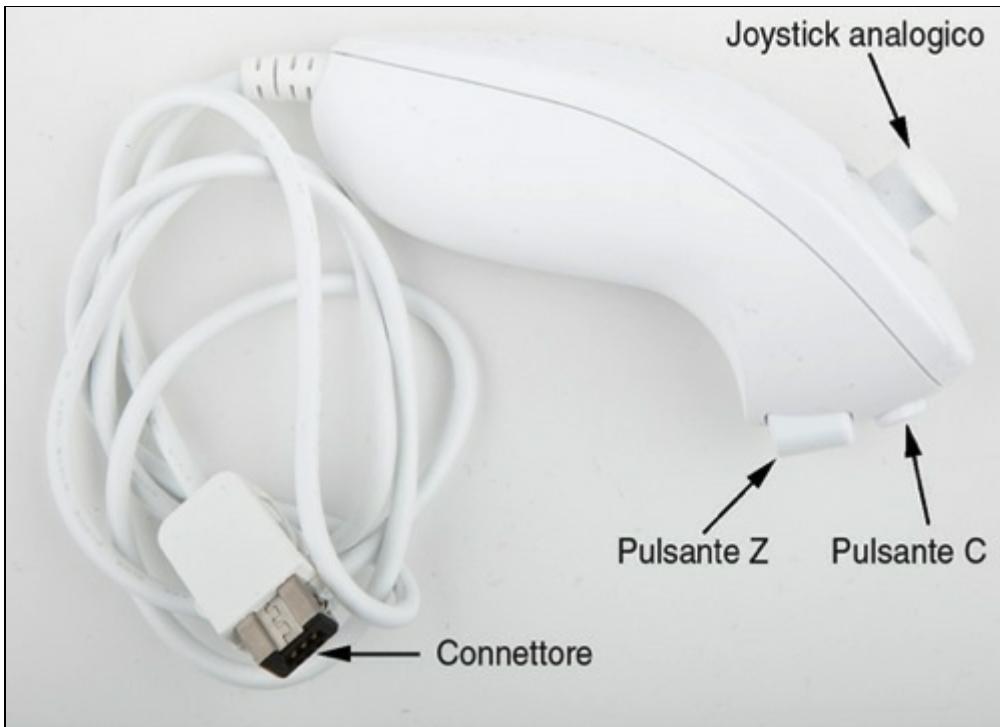


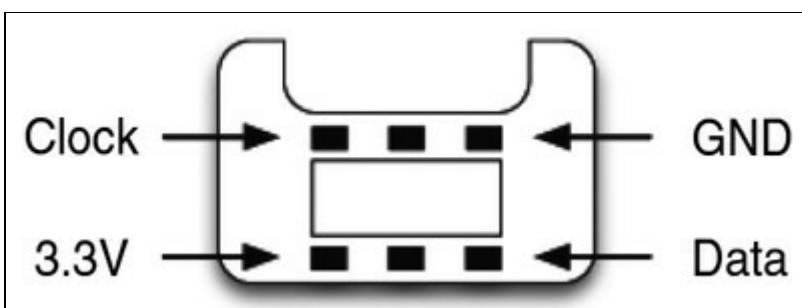
Figura 7.1 Il controller Nintendo Nunchuk.

Cosa serve

- Una scheda Arduino, per esempio un modello Arduino Uno, Duemilanove o Diecimila.
- Un cavo USB per collegare la scheda Arduino al computer.
- Un controller Nintendo Nunchuk.
- Quattro cavi di collegamento.

Cablaggio del controller Wii Nunchuk

Il collegamento tra Nunchuk e Arduino è veramente semplice. Non dovete aprire il controller né modificarlo in alcun modo. È sufficiente aggiungere quattro cavi al suo connettore e collegarli alla scheda Arduino. Di seguito è illustrato lo schema di cablaggio di una presa Nunchuk.



Il controller presenta sei connettori, anche se ne vengono impiegati solo quattro: GND, 3.3 V, Data e Clock. Inserite un cavo in ogni connettore e

collegate i cavi alla scheda Arduino rispettando la corrispondenza tra il pin Data e il pin analogico 4 di Arduino, mentre il pin Clock va collegato al pin analogico 5. Il pin GND deve essere connesso con il pin di massa di Arduino e il pin 3.3 V con l'omonimo pin Arduino.

Queste indicazioni concludono il cablaggio tra controller Nunchuk e scheda Arduino. Nel prossimo paragrafo studierete l’interfaccia tra controller e scheda di controllo, costituita dal punto di vista hardware esclusivamente dai cavi che avete collegato ai pin 4 e 5 di Arduino.

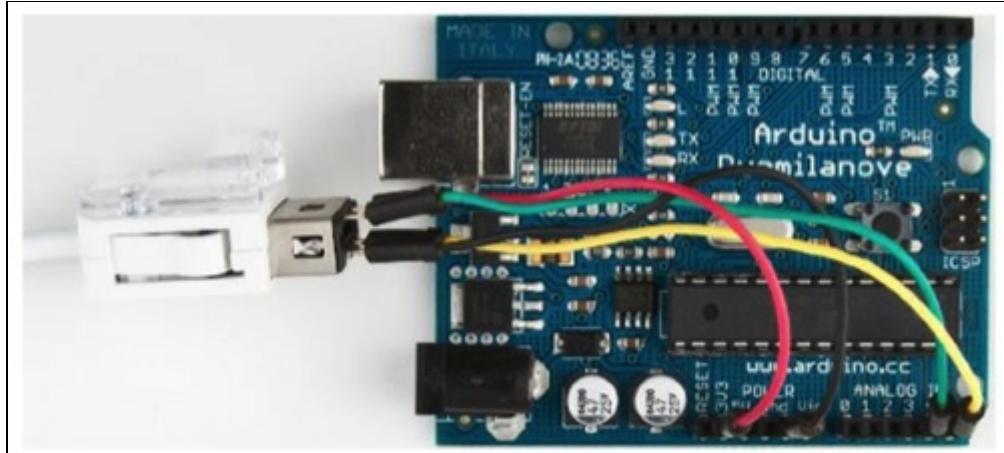


Figura 7.2 Collegamento tra Nunchuk e scheda Arduino.

Comunicare con il controller Nunchuk

Non è disponibile alcuna documentazione ufficiale che illustri lo schema elettronico interno del controller Nunchuk o come utilizzare il dispositivo in un ambiente diverso da Nintendo Wii. Su Internet si possono però trovare indicazioni fornite da hacker e altri appassionati di elettronica che spiegano il funzionamento interno del controller (<http://www.windmeadow.com/node/42>).

Tutto sommato il funzionamento del controller è veramente semplice, dato che Nunchuk si basa su una interfaccia TWI (*Two-Wire Interface*), nota anche come protocollo I2C (*Inter-Integrated Circuit*), le cui specifiche sono disponibili all’indirizzo <http://en.wikipedia.org/wiki/I2C>.

L’interfaccia determina una modalità di comunicazione master/slave dei dati che richiede due soli cavi di collegamento, uno per la trasmissione dei dati (DATA) e l’altro per la sincronizzazione della comunicazione (CLOCK).

L’IDE Arduino comprende la libreria Wire che implementa il protocollo I2C, in base al quale il controller si aspetta un collegamento della linea dati al pin analogico 4, mentre la linea di clock è connessa al pin analogico 5.

Questo protocollo è utilizzato per comunicare con Nunchuk; prima di procedere con queste operazioni occorre però conoscere i comandi ammessi dal controller. All’indirizzo

<http://todbot.com/blog/2010/09/25/softi2cmaster-add-i2c-to-any-arduino-pins/> potete trovare una libreria che permette di impiegare una coppia qualsiasi di pin nella comunicazione I2C.

Migliorare la vita delle persone giocherellando con l’elettronica

La diffusione delle moderne console per videogiochi ha abbassato drasticamente il prezzo delle periferiche, anche tenendo conto che i dispositivi non comprendono solo i controller di tipo classico, ma perfino simulatori di snowboard, fotocamere e altro ancora. Non deve pertanto sorprendere che qualcuno particolarmente creativo sia riuscito a realizzare progetti interessanti sfruttando componenti hardware ideati inizialmente per le console dei videogiochi.

Uno dei progetti più di effetto e utili è Eyewriter (<http://www.eyewriter.org/>), un’apparecchiatura elettronica che utilizza il dispositivo PlayStationEye (una fotocamera per la PlayStation 3 di Sony) per tenere traccia dei movimenti degli occhi umani.

Questo progetto è stato realizzato da un gruppo di hacker per aiutare un loro amico con difficoltà motorie a disegnare graffiti. A causa di una malattia questo artista (molto noto nel settore) è quasi completamente paralizzato ed è in grado di muovere solo gli occhi. Grazie ad Eyewriter ora può creare di nuovo disegni straordinari. Vale la pena studiare il progetto anche se non è un vero e proprio progetto Arduino.

A essere onesti il controller Nunchuk comprende il solo comando “Passami tutti i dati”. Ogni volta che riceve questo comando restituisce sei byte, il cui significato è indicato di seguito e illustrato nella Figura 7.3.

Bit	7	6	5	4	3	2	1	0
Byte 1	Joystick x position							
Byte 2	Joystick y position							
Byte 3	X acceleration bits 9..2							
Byte 4	Y acceleration bits 9..2							
Byte 5	Z acceleration bits 9..2							
Byte 6	Z accel. bits 1..0	Y accel. bits 1..0	X accel. bits 1..0	C status	Z status			

Figura 7.3 Nunchuk restituisce sempre 6 byte di dati.

- Il byte 1 contiene il valore dell’asse x del joystick analogico, mentre il byte 2 contiene il valore dell’asse y. Questi dati sono numeri a 8 bit con valori compresi tra 29 e 225.

- I valori di accelerazione lungo gli assi x, y e z sono numeri a 10 bit. I byte 3, 4 e 5 contengono i rispettivi otto bit più significativi, mentre i due bit meno significativi sono inclusi nel byte 6.
- Il byte 6 deve essere interpretato un bit alla volta. Il bit 0 è il bit meno significativo e contiene lo stato del pulsante Z. Questo bit vale 0 se il pulsante è premuto, altrimenti vale 1. Il bit 1 contiene lo stato del pulsante C.

Gli altri sei bit contengono i restanti bit meno significativi dei valori di accelerazione. I bit 2 e 3 riguardano l'asse X, i bit 4 e 5 l'asse Y, mentre i bit 6 e 7 sono relativi all'asse Z.

Dopo aver visto come interpretare i dati rilevati da Nunchuk potete iniziare a costruire la corrispondente classe `Nunchuk`.

Applicazioni scientifiche del controller Wii

L'accuratezza della Wii e i costi contenuti suggeriscono l'impiego scientifico delle apparecchiature Wii, ovvero per scopi diversi dai videogiochi; potete per esempio trovare idrologi che misurano l'evaporazione su una massa d'acqua (<http://www.wired.com/wiredscience/2009/12/wiimote-science/>). In genere l'apparecchiatura che effettua queste misurazioni ha costi che superano ampiamente i 500 dollari.

Alcuni medici dell'Università di Melbourne hanno utilizzato una Wii Balance Board come dispositivo a basso costo nelle terapie riabilitative di chi ha subito un ictus

(<http://www.newscientist.com/article/mg20527435.300-wii-board-helps-physios-strikeabalance-after-strokes.html>). Le loro pubblicazioni documentano come i dati della Balance Board siano clinicamente paragonabili a quelli ottenuti da una piattaforma stabilometrica professionale, a un costo decisamente inferiore.

La classe Nunchuk

Di seguito è riportata l'interfaccia della classe `Nunchuk` e la parte principale della sua implementazione.

`MotionSensor/NunchukDemo/nunchuk.h`

```
Riga 1 #ifndef __NUNCHUK_H__
- #define __NUNCHUK_H__
-
- #define NUNCHUK_BUFFER_SIZE 6
5
- class Nunchuk {
- public:
-     void initialize();
-     bool update();
10
-     int joystick_x() const { return _buffer[0]; }
-     int joystick_y() const { return _buffer[1]; }
```

```

-
-     int x_acceleration() const {
15      return ((int) (_buffer[2]) << 2) | (( _buffer[5] >> 2) &
0x03);
-
-     int y_acceleration() const {
-
-       return ((int) (_buffer[3]) << 2) | (( _buffer[5] >> 4) &
0x03);
20     }
-
-     int z_acceleration() const {
-
-       return ((int) (_buffer[4]) << 2) | (( _buffer[5] >> 6) &
0x03);
-
-     }
25
-     bool z_button() const { return !(_buffer[5] & 0x01); }
-     bool c_button() const { return !(_buffer[5] & 0x02); }
-
-
-   private:
30   void request_data();
-   char decode_byte(const char);
-
-
-   unsigned char _buffer[NUNCHUK_BUFFER_SIZE];
-
- };
35
- #endif

```

Questa piccola classe C++ contiene tutto quello che serve per utilizzare un controller Nunchuk in un progetto Arduino. Le prime istruzioni della classe definiscono un meccanismo che evita la doppia inclusione: tramite `#ifndef` si controlla l'esistenza di una macro preprocessor chiamata `_NUNCHUK_H_`. Si imposta la macro, se questa operazione non è ancora stata eseguita, e si prosegue con la dichiarazione della classe `Nunchuk`; in caso contrario la funzione preprocessor salta la dichiarazione. In questo modo potete includere il file header più volte nella stessa applicazione.

La riga 4 genera una costante che riguarda la dimensione dell'array necessario per memorizzare i dati restituiti da Nunchuk. L'array è definito alla riga 33 e in questo progetto la costante è impostata dalla funzione preprocessor e non tramite la parola chiave `const`, dato che C++ prevede che le costanti degli array siano note in fase di compilazione.

A questo punto ha inizio la dichiarazione della classe `Nunchuk`. Per avviare il canale di comunicazione tra Arduino e Nunchuk si deve chiamare una volta il metodo `initialize()`, poi si chiama `update()` ogni volta che Nunchuk deve inviare dati. Questi metodi saranno implementati nei prossimi paragrafi.

Il programma include metodi pubblici che impostano tutti gli attributi restituiti da Nunchuk: posizione x e y del joystick analogico, stato dei pulsanti e valori di accelerazione lungo gli assi x, y e z. Questi metodi elaborano i dati rilevati direttamente nel buffer alla riga 33. La loro implementazione è abbastanza banale e richiede una sola riga di istruzioni; solo l'assemblaggio dei valori a 10 bit dell'accelerazione richiede una serie di operazioni sui bit (si veda a tal proposito l'Appendice B).

Alla fine della dichiarazione della classe potete trovare due metodi helper privati che devono ancora essere implementati come indicato di seguito: `initialize()` e `update()`.

MotionSensor/NunchukDemo/nunchuk.cpp

```
Riga 1 #include <WProgram.h>
- #include <Wire.h>
- #include "nunchuk.h"
-
5 #define NUNCHUK_DEVICE_ID 0x52
-
- void Nunchuk::initialize() {
-     Wire.begin();
-     Wire.beginTransmission(NUNCHUK_DEVICE_ID);
10    Wire.send(0x40);
-     Wire.send(0x00);
-     Wire.endTransmission();
-     update();
- }
15
- bool Nunchuk::update() {
-     delay(1);
-     Wire.requestFrom(NUNCHUK_DEVICE_ID, NUNCHUK_BUFFER_SIZE);
-     int byte_counter = 0;
20     while (Wire.available() && byte_counter <
NUNCHUK_BUFFER_SIZE)
-         _buffer[byte_counter++] = decode_byte(Wire.receive());
-     request_data();
-     return byte_counter == NUNCHUK_BUFFER_SIZE;
- }
```

```

25
- void Nunchuk::request_data() {
-     Wire.beginTransmission(NUNCHUK_DEVICE_ID);
-     Wire.send(0x00);
-     Wire.endTransmission();
30 }
-
- char Nunchuk::decode_byte(const char b) {
-     return (b ^ 0x17) + 0x17;
- }

```

Dopo aver incluso le librerie necessarie, il programma definisce la costante `NUNCHUK_DEVICE_ID`. Il protocollo *I2C* è di tipo master/slave; in questo progetto Arduino è l'elemento master e Nunchuk è lo slave. Il controller Nunchuk registra se stesso sul bus dati utilizzando un ID particolare (0x52), in modo da potervi fare riferimento ogni volta che si rende necessario.

Il metodo `initialize()` imposta la connessione tra Arduino e Nunchuk inviando un segnale di handshake. Alla riga 8 si chiama la funzione `begin()` della classe `wire`, in modo che Arduino si unisca al protocollo *I2C* come master; se passate alla funzione `begin()` un ID il dispositivo corrispondente si unisce alle trasmissioni *I2C* come elemento slave. A questo punto ha inizio una nuova trasmissione verso il dispositivo identificato da `NUNCHUCK_DEVICE_ID`, nel nostro caso la scheda di controllo Nunchuk.

La trasmissione invia due byte (0x40 e 0x00) a Nunchuk, dopodiché si conclude. Questa comunicazione determina l'intera procedura di handshake, poi il programma chiede a Nunchuk lo stato corrente dei suoi sensori tramite la funzione `update()`. Nella Figura 7.4 si può vedere il flusso dei messaggi tra Arduino e un controller Nunchuk.

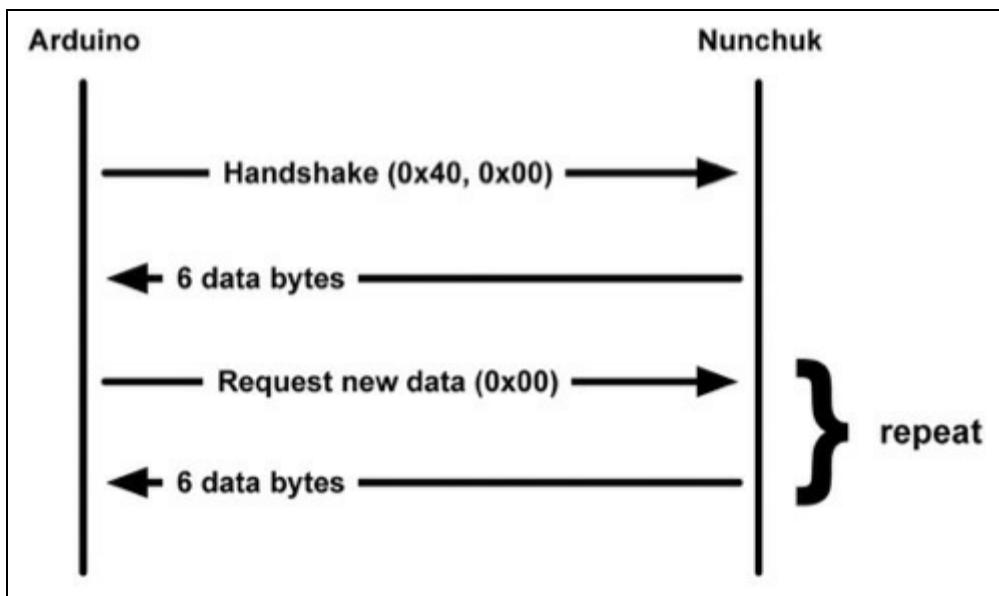


Figura 7.4 Flusso dei messaggi tra Arduino e Nunchuk.

La funzione `update()` imposta in primo luogo una pausa di un millisecondo per lasciare che si stabilizzino tutti i segnali, poi richiede sei bit da Nunchuk chiamando `Wire.requestFrom()`. Questo metodo non restituisce i byte dei sensori, che devono essere letti da un ciclo di istruzioni per riempire un buffer. Il metodo `Wire.available()` restituisce il numero di byte disponibili nel bus dati, mentre `Wire.receive()` restituisce il byte corrente. Non potete utilizzare direttamente i byte trasmessi da Nunchuk, dato che il controller ne nasconde leggermente il significato. La “decodifica” dei dati è abbastanza semplice, come si può notare studiando la funzione `decode_byte()`.

Il programma chiama infine il metodo `request_data()` per dire a Nunchuk di predisporre una nuova serie di dati. Si trasmette a Nunchuk un singolo byte zero, il cui significato è “prepara i prossimi sei byte”.

Prima di utilizzare la classe `Nunchuk` in base alle indicazioni del prossimo paragrafo vale la pena esaminare la documentazione della libreria `Wire`. Selezionate nel menu dell’IDE di Arduino il comando *Help > Reference* e fate clic sul link *Libraries*.

Impiego della classe Nunchuk

A questo punto potete utilizzare la classe `Nunchuk` per esaminare i dati rilevati dal controller.

[MotionSensor/NunchukDemo/NunchukDemo.pde](#)

```
#include <Wire.h>
```

```

#include "nunchuk.h"

const unsigned int BAUD_RATE = 19200;

Nunchuk nunchuk;

void setup() {
    Serial.begin(BAUD_RATE);
    nunchuk.initialize();
}

void loop() {
    if (nunchuk.update()) {
        Serial.print(nunchuk.joystick_x());
        Serial.print(" ");
        Serial.print(nunchuk.joystick_y());
        Serial.print(" ");
        Serial.print(nunchuk.x_acceleration());
        Serial.print(" ");
        Serial.print(nunchuk.y_acceleration());
        Serial.print(" ");
        Serial.print(nunchuk.z_acceleration());
        Serial.print(" ");
        Serial.print(nunchuk.z_button());
        Serial.print(" ");
        Serial.println(nunchuk.c_button());
    }
}

```

Queste istruzioni non presentano novità di sorta: si definisce un oggetto globale `Nunchuk` e lo si inizializza nella funzione `setup()`. In `loop()` si chiama `update()` per richiedere lo stato corrente del controller e trasmettere in output tutti gli attributi tramite la porta seriale.

Compilate e caricate il programma, poi aprite *Serial Monitor* e provate a giocherellare con Nunchuk. Muovete il joystick, muovete il controller e premete i pulsanti. Dovreste osservare dati simili ai seguenti:

```

46 109 428 394 651 1 1
49 132 414 380 656 1 0
46 161 415 390 651 1 0
46 184 429 377 648 1 0
53 199 404 337 654 1 0
53 201 406 359 643 1 0

```

A questo punto avete collegato correttamente un controller Nunchuk alla scheda Arduino. Non è ancora un progetto strabiliante, ma nel prossimo paragrafo vedrete come controllare gli oggetti presenti sullo schermo del computer utilizzando Nunchuk.

Rotazione di un cubo colorato

Il controller Nunchuk nasce per i videogiochi che devono trasformare i movimenti fisici reali in movimenti virtuali di un elemento visualizzato sullo schermo di un computer. In questo paragrafo effettuerete proprio un'operazione di questo genere: grazie a Nunchuk farete muovere un cubo tridimensionale sullo schermo (Figura 7.5).

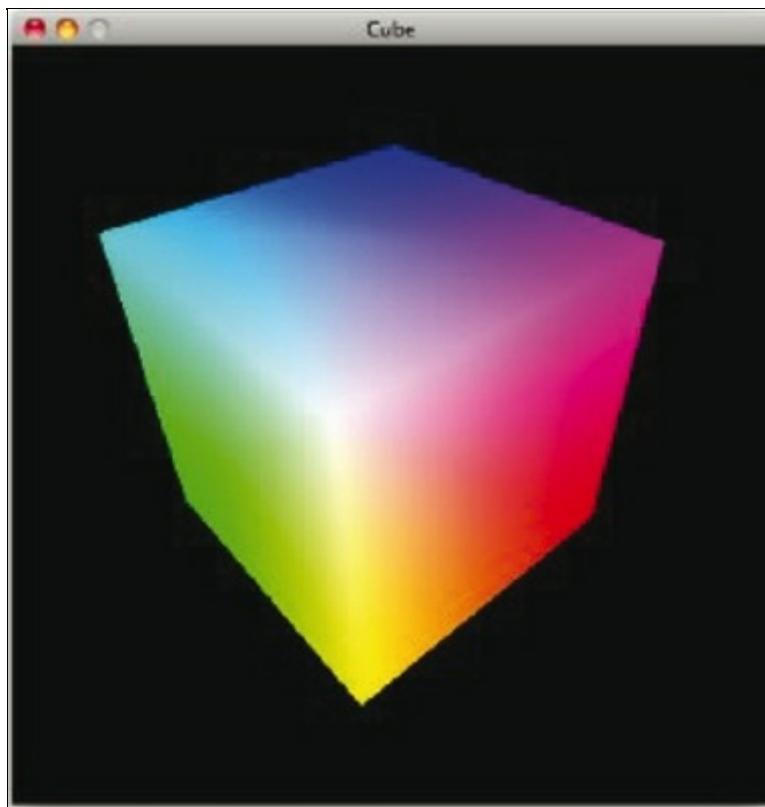


Figura 7.5 Controllo della rotazione di un cubo tramite Nunchuk.

Prima di iniziare a disegnare il cubo e impiegare il controller dovete studiare un aspetto del progetto finora trascurato, ovvero la presenza di vibrazioni del segnale rilevato dai sensori o *jitter*. Analogamente al controller illustrato nel Capitolo 6 anche i dati di accelerazione rilevati da Nunchuk devono essere stabilizzati. Verrà adottata la stessa tecnica mostrata nel capitolo precedente, ma questa volta la soluzione sarà implementata nel codice Processing invece che nel programma della scheda Arduino.

```

Riga 1  class SensorDataBuffer {
-      private int _maxSamples;
-      private int _bufferIndex;
-      private int[] _xBuffer;
5       private int[] _yBuffer;
-      private int[] _zBuffer;
-
-
-      public SensorDataBuffer(final int maxSamples) {
-          _maxSamples = maxSamples;
10         _bufferIndex = 0;
-          _xBuffer = new int[_maxSamples];
-          _yBuffer = new int[_maxSamples];
-          _zBuffer = new int[_maxSamples];
-      }
15
-      public void addData(final int x, final int y, final int z) {
-          if (_bufferIndex >= _maxSamples)
-              _bufferIndex = 0;
-
20          _xBuffer[_bufferIndex] = x;
-          _yBuffer[_bufferIndex] = y;
-          _zBuffer[_bufferIndex] = z;
-          _bufferIndex++;
-      }
25
-      public int getX() {
-          return getAverageValue(_xBuffer);
-      }
-
30      public int getY() {
-          return getAverageValue(_yBuffer);
-      }
-
-      public int getZ() {
35          return getAverageValue(_zBuffer);
-      }
-
-      private int getAverageValue(final int[] buffer) {
-          int sum = 0;
40          for (int i = 0; i < _maxSamples; i++)
-              sum += buffer[i];
-          return (int) (sum / _maxSamples);
-      }
-  }

```

SensorDataBuffer incapsula i tre buffer dei dati di accelerazione lungo gli assi x, y e z, oltre a memorizzare un indice di buffer che contiene la

posizione attuale nei tre buffer. Il costruttore definito a partire dalla riga 8 si aspetta come parametro il numero massimo di campioni (ovvero la dimensione del buffer) inizializza il buffer e il suo indice.

Il metodo `addData()` riceve i nuovi valori relativi ai tre assi spaziali e li aggiunge nei buffer corrispondenti. Se un buffer è pieno, si eliminano le voci meno recenti. I metodi `getX()`, `getY()` e `getZ()` permettono di richiedere il valore attuale dell'accelerazione media su ciascun asse. Questi tre metodi delegano l'elaborazione dati al metodo `getAverageValue()`.

A questo punto potete iniziare a disegnare un cubo tridimensionale. In primo luogo il programma inizializza gli elementi relativi alla comunicazione seriale della scheda Arduino che controlla Nunchuk, come indicato di seguito.

[MotionSensor/Cube/Cube.pde](#)

```
import processing.serial.*;  
  
final int LINE_FEED = 10;  
final int MAX_SAMPLES = 16;  
  
Serial arduinoPort;  
SensorDataBuffer sensorData = new SensorDataBuffer(MAX_SAMPLES);
```

Anche in questo caso si importano le librerie della comunicazione seriale e si inizializza un oggetto globale `Serial`; in questo progetto si deve anche creare un oggetto `SensorDataBuffer`.

Ora si devono impostare le costanti relative alle dimensioni della schermata, gli intervalli dei valori rilevati da Nunchuk e si devono eseguire alcuni calcoli “tridimensionali”.

[MotionSensor/Cube/Cube.pde](#)

```
final int WIDTH = 500;  
final int HEIGHT = 500;  
final int BAUD_RATE = 19200;  
final int X_AXIS_MIN = 300;  
final int X_AXIS_MAX = 700;  
final int Y_AXIS_MIN = 300;  
final int Y_AXIS_MAX = 700;  
final int Z_AXIS_MIN = 300;  
final int Z_AXIS_MAX = 700;  
final int MIN_SCALE = 5;
```

```

final int MAX_SCALE = 128;
final float MX = 2.0 / (X_AXIS_MAX - X_AXIS_MIN);
final float MY = 2.0 / (Y_AXIS_MAX - Y_AXIS_MIN);
final float MZ = 2.0 / (Z_AXIS_MAX - Z_AXIS_MIN);
final float BX = 1.0 - MX * X_AXIS_MAX;
final float BY = 1.0 - MY * Y_AXIS_MAX;
final float BZ = 1.0 - MZ * Z_AXIS_MAX;

```

`X_AXIS_MIN` e `X_AXIS_MAX` impostano il valore minimo e massimo dell'accelerazione rilevata da Nunchuk sull'asse x. Valori analoghi sono definiti da `Y_AXIS_MIN` e così via. Il programma deve anche impostare le altre costanti (`MX`, `BX` e così via) che più avanti serviranno per convertire i valori di accelerazioni in angoli.

A questo punto vanno definite le variabili che memorizzano lo stato attuale del cubo: la sua posizione, l'angolo di rotazione rispetto ai singoli assi e il fattore di scala.

MotionSensor/Cube/Cube.pde

```

int xpos = WIDTH / 2;
int ypos = HEIGHT / 2;
int scale = 90;

float xrotate = 0.0;
float yrotate = 0.0;
float zrotate = 0.0;

```

Il metodo `setup()` inizializza la schermata e la porta seriale.

MotionSensor/Cube/Cube.pde

```

Riga 1 void setup() {
  2   size(WIDTH, HEIGHT, P3D);
  3   noStroke();
  4   colorMode(RGB, 1);
  5   background(0);
  6   println(Serial.list());
  7   arduinoPort = new Serial(this, Serial.list()[0], BAUD_RATE);
  8   arduinoPort.bufferUntil(LINE_FEED);
  9 }

```

L'unica istruzione da studiare con attenzione è la chiamata di `colorMode()` alla riga 4. Questa funzione stabilisce che i colori sono valori RGB

compresi tra 0 e 1; il cubo ha così un aspetto molto vivace. La sagoma disegnata in 3D deriva da uno degli esempi standard di Processing.

Potete disegnare il cubo utilizzando il programma riportato di seguito.

MotionSensor/Cube/Cube.pde

```
Riga 1 void draw() {  
-     background(0);  
-     pushMatrix();  
  
-  
5      translate(xpos, ypos, -30);  
-      rotateX(yrotate);  
-      rotateY(xrotate);  
-      rotateZ(zrotate);  
-      scale(scale);  
10  
-      beginShape(QUADS);  
-      fill(0, 1, 1); vertex(-1, 1, 1);  
-      fill(1, 1, 1); vertex( 1, 1, 1);  
-      fill(1, 0, 1); vertex( 1, -1, 1);  
15      fill(0, 0, 1); vertex(-1, -1, 1);  
  
-  
-      fill(1, 1, 1); vertex( 1, 1, 1);  
-      fill(1, 1, 0); vertex( 1, 1, -1);  
-      fill(1, 0, 0); vertex( 1, -1, -1);  
20      fill(1, 0, 1); vertex( 1, -1, 1);  
  
-  
-      fill(1, 1, 0); vertex( 1, 1, -1);  
-      fill(0, 1, 0); vertex(-1, 1, -1);  
-      fill(0, 0, 0); vertex(-1, -1, -1);  
25      fill(1, 0, 0); vertex( 1, -1, -1);  
  
-  
-      fill(0, 1, 0); vertex(-1, 1, -1);  
-      fill(0, 1, 1); vertex(-1, 1, 1);  
-      fill(0, 0, 1); vertex(-1, -1, 1);  
30      fill(0, 0, 0); vertex(-1, -1, -1);  
  
-  
-      fill(0, 1, 0); vertex(-1, 1, -1);  
-      fill(1, 1, 0); vertex( 1, 1, -1);  
-      fill(1, 1, 1); vertex( 1, 1, 1);  
35      fill(0, 1, 1); vertex(-1, 1, 1);  
  
-  
-      fill(0, 0, 0); vertex(-1, -1, -1);  
-      fill(1, 0, 0); vertex( 1, -1, -1);  
-      fill(1, 0, 1); vertex( 1, -1, 1);  
40      fill(0, 0, 1); vertex(-1, -1, 1);  
-      endShape();  
-
```

```
-     popMatrix();
-
```

`draw()` definisce e riempie le sei facce del cubo utilizzando le funzioni `fill()` e `vertex()`. Si definiscono i vertici a partire dalla coordinate di base, dato che alla riga 9 verrà cambiata la scala di visualizzazione del cubo impostando una dimensione più ragionevole. Lo spostamento e la rotazione del cubo sono operazioni gestite dalla riga 5 alla riga 8.

Dato che il metodo Processing `draw()` annulla tutte le elaborazioni effettuate da `translate()` e dai metodi di rotazione, si utilizzano le funzioni `pushMatrix()` e `popMatrix()` per memorizzare e ripristinare queste informazioni.

Il programma deve infine recuperare i dati Nunchuk e convertirli in argomenti da passare alle funzioni di elaborazione vettoriale.

MotionSensor/Cube/Cube.pde

```
Riga 1 void serialEvent(Serial port) {
-
-     final String arduinoData = port.readStringUntil(LINE_FEED);
-
-     if (arduinoData != null) {
5         final int[] data = int(split(trim(arduinoData), ' '));
-
-         if (data.length == 7) {
-
-             xpos = int(map(data[0], 0x1e, 0xe1, 0, WIDTH));
-             ypos = int(map(data[1], 0xd1, 0xdf, HEIGHT, 0));
-
-
10            if (data[5] == 1) scale++;
-            if (data[6] == 1) scale--;
-
-            if (scale < MIN_SCALE) scale = MIN_SCALE;
-            if (scale > MAX_SCALE) scale = MAX_SCALE;
-
-
15            sensorData.addData(data[2], data[3], data[4]);
-
-
-            final float gx = MX * sensorData.getX() + BX;
-            final float gy = MY * sensorData.getY() + BY;
-            final float gz = MZ * sensorData.getZ() + BZ;
20
-            xrotate = atan2(gx, sqrt(gy * gy + gz * gz));
-            yrotate = atan2(gy, sqrt(gx * gx + gz * gz));
-            zrotate = atan2(sqrt(gx * gx + gy * gy), gz);
-
-        }
25    }
-
```

Le operazioni di lettura, riconoscimento e conversione dei dati trasmessi dalla porta seriale sono analoghe a quelle viste nei capitoli precedenti. La parte più interessante di questo programma inizia alla riga 7, dove si mappa la posizione x del joystick analogico sulla nuova coordinata x del cubo. La riga successiva esegue la stessa operazione per la coordinata y.

Le istruzioni dalla riga 10 alla riga 13 gestiscono lo stato dei pulsanti Nunchuk. Premete il pulsante Z per aumentare la dimensione del cubo; premete il pulsante C per rimpicciolirlo.

Le altre istruzioni del metodo `serialEvent()` convertono i valori di accelerazione del controller in angoli. I calcoli matematici sono piuttosto complessi ed esulano dalla trattazione principale di questo capitolo.

Avviate il programma e provate a giocherellare con il cubo. Facile, vero? Avete utilizzato solo quattro cavi e alcune istruzioni software per impiegare l'hardware Nunchuk in un vostro progetto di controllo hardware e software. Potete sfruttare questo dispositivo per controllare un robot; qualcuno lo ha addirittura impiegato per creare musica (<http://www.youtube.com/watch?v=J4GPS83Rm6M>).

La prossima volta che acquisterete una nuova periferica hardware provate a immaginare un suo impiego in un contesto diverso da quello tradizionale. Spesso la soluzione è più semplice di quanto sembri. Ricordate inoltre che quando create una classe simile alla classe `Nunchuk` di questo progetto vale la pena considerare la possibilità di convertirla in libreria e renderla disponibile su Internet, come è stato spiegato nel Capitolo 4.

Cosa fare se non funziona?

Dal punto di vista di chi deve realizzare il progetto questa volta la soluzione è decisamente semplice, ma può esserci comunque qualcosa che va storto, specialmente nel cablaggio hardware. Verificate di aver collegato correttamente i pin di Arduino e quelli di Nunchuk. Controllate inoltre che i cavi siano ben fissati ai connettori di Arduino e del controller Nunchuk. Se necessario, utilizzate cavi di diametro maggiore.

Esercizi

- Riscrivete il gioco implementato nel Capitolo 6 in modo da supportare il

controller Nunchuk. Il gioco deve supportare tanto il joystick analogico quanto l'accelerometro. Riuscite a commutare tra joystick e accelerometro utilizzando i pulsanti del controller Nunchuk?

- Elaborare la Nintendo Wii Motion è un po' più complicato (<http://randomhacksfboredom.blogspot.com/2009/07/motion-plus-and-nunchucktogetheron.html>) ma può essere una strada intrigante per migliorare significativamente le vostre abilità di progetto.

Networking con Arduino

Una scheda Arduino permette di realizzare innumerevoli progetti sempre utili e divertenti, ma il suo collegamento in rete (*networking*) spalanca le porte a nuove opportunità di utilizzo dei progetti di controllo.

L'accesso alle informazioni via Internet consente per esempio di trasformare Arduino in una stazione meteo che riporta i dati rilevati da un servizio di previsioni meteorologiche. Potete perfino fare in modo che Arduino diventi un web server che mette a disposizione i dati dei suoi sensori ad altri dispositivi e computer, sempre tramite funzioni di networking.

In questo capitolo verrà realizzato un sistema di allarme anti-intrusione che rileva la presenza di movimenti nel vostro soggiorno. La scheda Arduino trasmette un messaggio di posta elettronica ogni volta che rileva un movimento in vostra assenza. Si tratta di un progetto decisamente sofisticato, il cui studio deve essere preceduto dall'analisi di progetti più semplici che permettano di apprendere le tecniche e le abilità necessarie per affrontare il progetto più avanzato.

Si può iniziare con una scheda Arduino “nuda”, che non include funzionalità di networking ma che viene comunque collegata a Internet sfruttando la connessione tra Arduino e computer.

La situazione migliorerà radicalmente nel secondo progetto, che prevede l'impiego di una scheda Ethernet shield. La scheda di controllo diventa un dispositivo di networking che accede direttamente a servizi IP, per esempio a un servizio DAYTIME, e trasforma Arduino in un orologio molto preciso.

L'accesso ai servizi IP permette di studiare l'invio di messaggi di posta elettronica direttamente tramite Arduino ed Ethernet shield. Per quanto riguarda il sistema di allarme occorre solo sapere come rilevare il movimento nella stanza utilizzando un sensore passivo a infrarossi. Nei prossimi paragrafi conoscerete le principali tecnologie di networking della scheda Arduino e imparerete a sfruttare soluzioni di networking in un sistema di allarme con sensore a infrarossi.

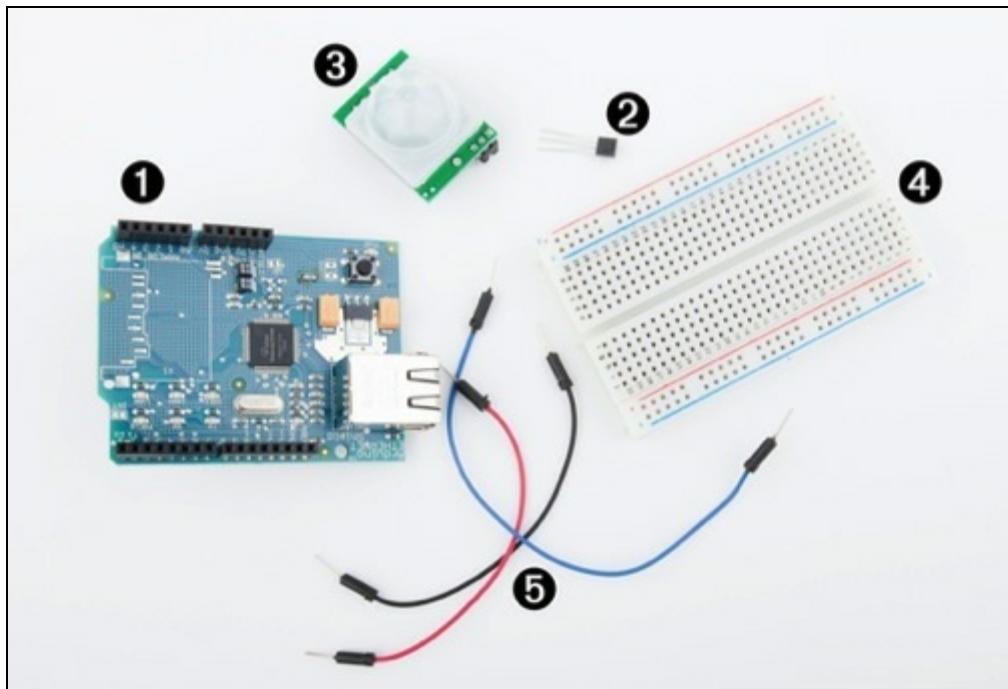


Figura 8.1 Componenti necessari per eseguire i progetti di questo capitolo.

Le conoscenze acquisite in questo capitolo vi permettono di realizzare un sistema di allarme anti-intrusione che invia messaggi di posta elettronica. Una volta che sarà attivo vi sentirete più al sicuro!

Cosa serve

1. Una scheda Ethernet shield per Arduino.
2. Un sensore di temperatura TMP36.
3. Un sensore di movimento a infrarossi di tipo PIR (*Passive Infrared Sensor*).
4. Una breadboard.
5. Cavi di collegamento su breadboard.
6. Una scheda Arduino, per esempio un modello Arduino Uno, Duemilanove o Diecimila.
7. Un cavo USB per collegare la scheda Arduino al computer.

Utilizzare il computer per trasferire i dati dei sensori su Internet

Ricordate come si faceva a collegare un computer a Internet, diciamo più o meno quindici anni fa? Tutto partiva da un modem a 38400 baud, Netscape Navigator 3 e da un floppy disk o CD che avevate ricevuto per posta. Oggi è molto probabile che abbiate a disposizione un accesso a banda larga via

ADSL, via cavo o satellitare e che in casa sia presente una stazione WiFi. Il progetto di questo capitolo prevede proprio di sfruttare inizialmente la connessione che avete a disposizione per collegare Arduino a Internet.

La Figura 8.2 mostra lo schema di principio di questa connessione. Un programma eseguito dal computer comunica con Arduino tramite la porta seriale ed è questo programma che gestisce le funzioni di networking ogni volta che l'applicazione della scheda di controllo ha bisogno di collegarsi a Internet. Questa architettura di sistema consente per esempio di inviare a Twitter (<http://twitter.com>) i dati rilevati dai sensori Arduino.

In particolare, in questo progetto il sistema di controllo invia a Twitter un messaggio ogni volta che la temperatura del vostro soggiorno o del vostro ufficio supera una determinata soglia, nello specifico i 32 gradi centigradi o Celsius (che corrispondono a 90 gradi Fahrenheit). Dovete innanzitutto realizzare un sensore di temperatura simile a quello illustrato nel Capitolo 5, poi dovete caricare in Arduino il programma riportato di seguito.

Ethernet/TwitterTemperature/TwitterTemperature.pde

```
Riga 1 #define CELSIUS
-
- const unsigned int TEMP_SENSOR_PIN = 0;
- const unsigned int BAUD_RATE = 9600;
5 const float SUPPLY_VOLTAGE = 5.0;
-
- void setup() {
-     Serial.begin(BAUD_RATE);
- }
10
- void loop() {
-     const int sensor_voltage = analogRead(TEMP_SENSOR_PIN);
-     const float voltage = sensor_voltage * SUPPLY_VOLTAGE /
1024;
-     const float celsius = (voltage * 1000 - 500) / 10;
15 #ifdef CELSIUS
-     Serial.print(celsius);
-     Serial.println(" C");
- #else
-     Serial.print(9.0 / 5.0 * celsius + 32.0);
20     Serial.println(" F");
- #endif
-     delay(5000);
- }
```

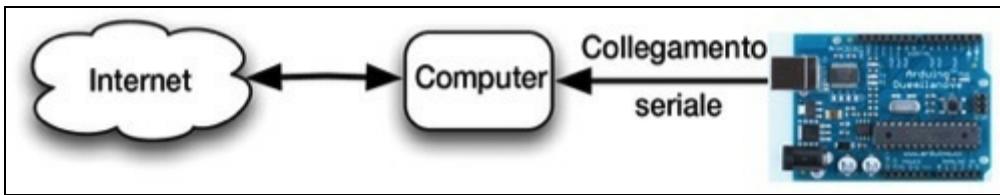


Figura 8.2 Collegamento tra Arduino e Internet tramite computer.

Queste istruzioni sono quasi identiche a quelle del progetto precedente. Alla riga 5 ricordate di impostare `SUPPLY_VOLTAGE` con il valore `3.3` se state utilizzando una scheda Arduino che prevede una tensione di alimentazione di `3,3V` e non di `5V`. Il programma supporta valori di temperatura espressi in gradi Celsius o Fahrenheit; l'unità di misura dipende dalla costante definita prima dell'elaborazione dei dati. L'applicazione visualizza in output la temperatura in gradi Celsius se impostate la costante `Celsius` nella prima riga del programma; se questa riga viene rimossa o indicata come commento, il programma esprime la temperatura in gradi Fahrenheit.

Il comportamento del programma dipende dalla direttiva preprocessor `#ifdef`, che verifica se è stata impostata una determinata costante e compila il codice tenendo conto della condizione rilevata. In questo programma si calcola il risultato della conversione da Celsius a Fahrenheit (riga 19) solo se non è stata impostata la costante `Celsius`.

Caricate il programma; dovreste visualizzare in output una nuova misura della temperatura corrente ogni cinque secondi, come nell'esempio che segue:

```

27.15 C
26.66 C
27.15 C

```

A questo punto è necessario scrivere un programma da eseguire a computer per leggere questi dati di output e inviare a Twitter un messaggio non appena la temperatura supera i 32 gradi Celsius (valore che corrisponde a 90 gradi Fahrenheit). Potete utilizzare qualsiasi linguaggio di programmazione in grado di leggere da porta seriale e di supportare Twitter. Come nei progetti precedenti anche in questo caso verrà impiegato Processing.

Servizi web per la pubblicazione dei dati del sensore

La diffusione di componenti hardware e sensori open source ha favorito negli anni la comparsa di molti servizi web che permettono di pubblicare dati rilevati da misure elettroniche. Questo genere di servizi consente di

pubblicare, leggere e analizzare i dati misurati da sensori di vario tipo e che provengono da ogni parte del mondo. Si tratta di dati ricavati da stazioni meteorologiche, sensori ambientali e altro ancora, messi gratuitamente a disposizione su Internet. I servizi web più conosciuti sono Pachube (<http://pachube.com>) e Sensorpedia (<http://sensorpedia.com/>). Da un punto di vista teorico l'utilizzo di questi siti è identico: dovete registrare un vostro account e ricevere una chiave API da impiegare per autenticare l'accesso ai servizi e caricare i dati dei sensori.

Registrazione di un'applicazione Twitter

Prima di scrivere il programma dovete registrarvi sul sito Twitter per ottenere le vostre credenziali di accesso OAuth

(<http://en.wikipedia.org/wiki/Oauth> o

<http://it.wikipedia.org/wiki/OAuth>). Il protocollo aperto OAuth permette alle applicazioni di utilizzare risorse di un'altra applicazione. In questo progetto dovete acquisire il privilegio che permetta alla vostra applicazione di aggiornare i vostri messaggi di Twitter senza richiedere ogni volta l'immissione di nome utente e password per l'accesso a questo social network.

Per molto tempo Twitter ha supportato il sistema di autenticazione HTTP Basic Authentication

(http://en.wikipedia.org/wiki/Basic_authentication). In questo caso i servizi web automatici richiedevano solo un nome utente e una password per impostare o aggiornare i messaggi di Twitter. A partire da agosto 2010 Twitter ha però rimosso il supporto del servizio Basic Authentication e ora impiega OAuth.

Potete ottenere un token di accesso OAuth dopo aver registrato la nuova applicazione nella sezione di sviluppo del sito web di Twitter

(<http://dev.twitter.com>). Effettuate il login e fate clic sul link *Register an app*, poi compilate il form che potete osservare nella Figura 8.3. Verificate che il tipo di applicazione sia *Client* e che l'accesso di default sia *Read & Write*. Il nome dell'applicazione è arbitrario e sarà riportato nel canale di Twitter ogni volta che utilizzerete l'applicazione per inviare nuovi messaggi. Se indicate per esempio il nome RescueMeFromWork, i vostri messaggi verranno pubblicati come RescueMeFromWork.

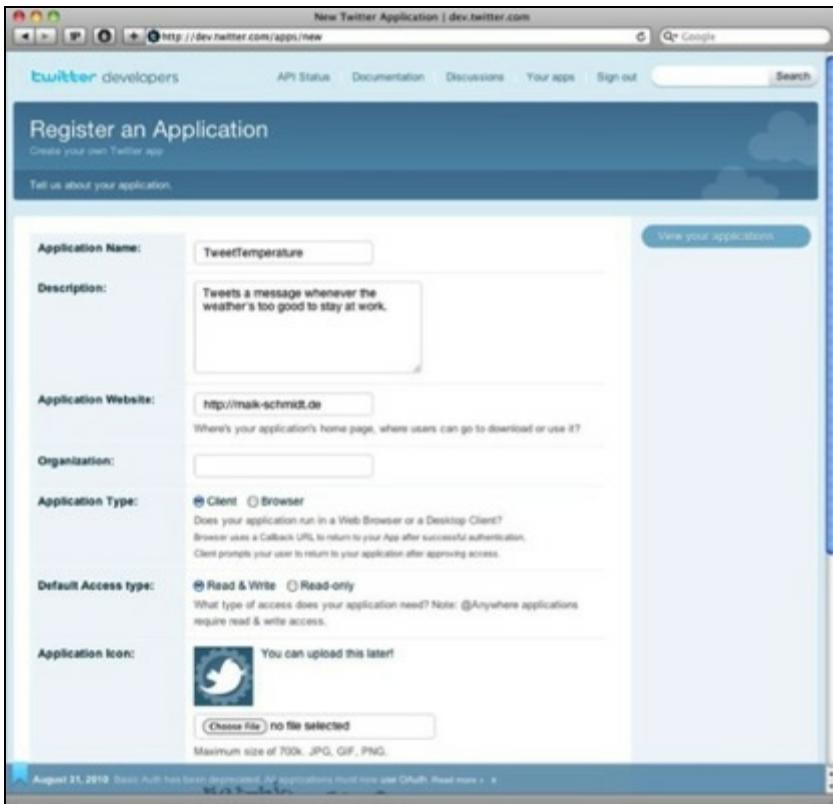


Figura 8.3 Registrazione di una nuova applicazione client per Twitter.

Dopo aver registrato la nuova applicazione visualizzate la pagina delle impostazioni dell'applicazione e individuate le voci *Consumer key* e *Consumer secret* in una finestra simile a quella mostrata nella Figura 8.4. La combinazione tra queste voci e i dati OAuth token e OAuth secret permettono alla vostra applicazione di modificare lo stato di Twitter. Per individuare le chiavi token e secret dovete fare clic sul link *My Access Token*.

Copiate tutte queste chiavi di accesso, che nel prossimo paragrafo verranno impiegate per inviare nuovi messaggi a Twitter tramite Processing.

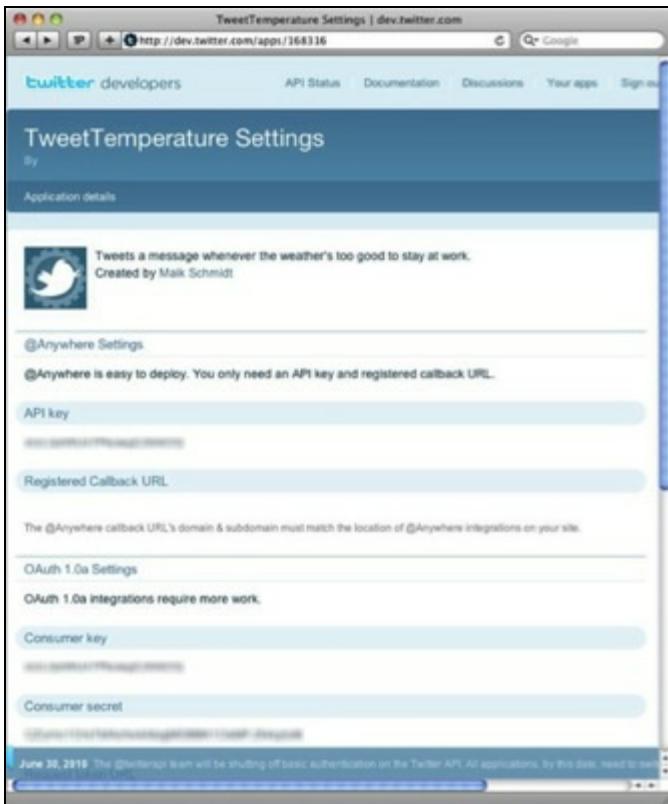


Figura 8.4 I privilegi di accesso sono riportati nella pagina delle impostazioni.

Messaggi Twitter con Processing

Processing non supporta direttamente la trasmissione di messaggi a Twitter ma i programmi Processing hanno accesso diretto alle librerie Java, tra le quali ve ne sono molte che si occupano della gestione di Twitter. Una delle librerie più note è `twitter4j` (<http://twitter4j.org/>), il cui funzionamento è consolidato da tempo e offre un ottimo supporto del protocollo OAuth.

Scaricate la libreria dal sito web

<http://twitter4j.org/en/index.html#download> e copiate l'archivio non compresso in una directory temporanea, dove potete trovare un file chiamato `twitter4j-core-x.y.z.jar` oppure `twitter4j-core-x.y.z-SNAPSHOT.jar`. Aprite l'IDE Processing, create un nuovo progetto e trascinate il file `.jar` nell'IDE; in questo modo il file `.jar` viene copiato direttamente nella cartella locale *code*. Queste operazioni permettono alla vostra applicazione di accedere alla libreria `twitter4j`.

Di seguito è riportato il codice principale del programma.

Ethernet/TweetTemperature/TweetTemperature.pde

```
import processing.serial.*;  
  
final float MAX_WORKING_TEMP = 32.0;
```

```

final int LINE_FEED = 10;
final int BAUD_RATE = 9600;
final String CONSUMER_KEY = "<YOUR CONSUMER KEY>";
final String CONSUMER_SECRET = "<YOUR CONSUMER SECRET>";
final String ACCESS_TOKEN = "<YOUR ACCESS TOKEN>";
final String ACCESS_TOKEN_SECRET = "<YOUR ACCESS TOKEN SECRET>";

Serial arduinoPort;

void setup() {
    println(Serial.list());
    arduinoPort = new Serial(this, Serial.list()[0], BAUD_RATE);
    arduinoPort.bufferUntil(LINE_FEED);
}

void draw() { }

```

Anche in questo programma si importano le librerie della comunicazione seriale con Arduino e si definiscono alcune costanti che saranno impiegate più avanti, la maggior parte delle quali riguarda le credenziali di accesso a Twitter. La costante `MAX_WORKING_TEMP` stabilisce la soglia di temperatura superata la quale l'applicazione invia un messaggio a Twitter. Il valore di questa costante può essere espresso in gradi Celsius o Fahrenheit.

Il metodo `setup()` visualizza un elenco dei dispositivi seriali a disposizione e inizializza la variabile `serialPort` con la prima periferica dell'elenco, nell'ipotesi che si tratti della scheda Arduino. Il programma potrebbe sfogliare automaticamente l'elenco fino a individuare un nome di porta che corrisponde a quello di una scheda Arduino, ma questo genere di soluzione è poco affidabile e si preferisce evitarla. L'applicazione del progetto non richiede alcuna visualizzazione grafica, pertanto il metodo `draw()` rimane vuoto.

A questo punto si può implementare la logica effettiva del sistema di allarme, che vi invita ad andare tranquillamente in spiaggia (“Someone, please, take me to the beach”).

[Ethernet/TweetTemperature/TweetTemperature.pde](#)

```

void serialEvent(Serial port) {
    final String arduinoData = port.readStringUntil(LINE_FEED);

    if (arduinoData != null) {
        final String[] data = split(trim(arduinoData), ' ');
        if (data.length == 2 &&

```

```

        (data[1].equals("C") || data[1].equals("F")))
    {
        float temperature = float(data[0]);
        println(temperature);
        int sleepTime = 5 * 60 * 1000;
        if (temperature > MAX_WORKING_TEMP) {
            tweetAlarm();
            sleepTime = 120 * 60 * 1000;
        }
        try {
            Thread.sleep(sleepTime);
        }
        catch(InterruptedException ignoreMe) { }
    }
}

void tweetAlarm() {
    TwitterFactory factory = new TwitterFactory();
    Twitter twitter = factory.getInstance();
    twitter.setOAuthConsumer(CONSUMER_KEY, CONSUMER_SECRET);
    AccessToken accessToken = new AccessToken(
        ACCESS_TOKEN,
        ACCESS_TOKEN_SECRET
    );
    twitter.setOAuthAccessToken(accessToken);
    try {
        Status status = twitter.updateStatus(
            "Someone, please, take me to the beach!"
        );
        println(
            "Successfully updated status to '" + status.getText() + "'."
        );
    }
    catch (TwitterException e) {
        e.printStackTrace();
    }
}
}

```

Nel Capitolo 5 avete imparato a implementare le comunicazioni seriali in Processing. Ogni volta che arrivano nuovi dati sulla porta seriale, il programma in runtime chiama il metodo `serialEvent()` che tenta di leggere una riga di testo e verifica se questa contiene un numero decimale seguito da uno spazio vuoto e da un carattere *C* o *F*. Questa forma di validazione dei dati assicura che è stato letto un dato set relativo a una misura di temperatura corretta.

Il dato validato è convertito in un oggetto `float` e il programma verifica se il suo valore è maggiore di `MAX_WORKING_TEMP` (nessuno dovrebbe lavorare quando fa così caldo!). In caso affermativo si chiama il metodo `tweetAlarm()` e si invia un messaggio a Twitter con una richiesta di aiuto che verrà letta da chi legge i vostri tweet (*follower*); a questo punto il programma attende due ore prima di eseguire una nuova misurazione della temperatura. Se la temperatura è più bassa si attendono solo cinque minuti prima di una nuova misurazione.

Il metodo `tweetAlarm()` aggiorna il canale Twitter in modo molto semplice. Aderendo alle convenzioni tipiche del linguaggio Java il programma crea una nuova istanza `Twitter` tramite `TwitterFactory` e imposta le credenziali di accesso chiamando la funzione `setOAuthConsumer()`. A questo punto si impostano le credenziali OAuth tramite `setOAuthAccessToken()` e infine si chiama la funzione `updateStatus()`. Se tutte queste operazioni vanno a buon fine, il programma visualizza a console un messaggio che segnala l'avvenuto successo. Se al contrario si manifesta un problema, il metodo `updateStatus()` genera un'eccezione e il programma visualizza la traccia di stack per facilitare il debugging delle istruzioni.

Dopo aver scritto questo codice non dovete far altro che collegare Arduino al computer e provare il programma. Nella Figura 8.5 potete osservare ciò che succede quando la temperatura della stanza di Maik supera i 32 gradi Celsius. Per eseguire le prime prove conviene impostare una soglia di temperatura inferiore. (Se non dovete modificare il valore massimo di temperatura si può sapere perché non siete già in spiaggia?)

L'impiego di un computer come “ripetitore” Internet per Arduino può costituire una soluzione comoda ma nella maggior parte delle applicazioni è anche eccessiva. Nel prossimo paragrafo vedrete come trasformare Arduino in un dispositivo di networking.



Figura 8.5 Si spera che qualcuno risponda alla richiesta di aiuto.

Comunicazione in rete tramite una scheda Ethernet shield

Nel paragrafo precedente avete imparato a realizzare applicazioni Arduino di networking usando la connessione a Internet del vostro computer. Questa tecnica funziona egregiamente ma presenta alcuni svantaggi. Il problema fondamentale è dato dalla necessità di avere a disposizione un computer solo per gestire la connessione a Internet, mentre in molte applicazioni Arduino è sufficiente impiegare le risorse hardware della scheda di controllo. In questo paragrafo vedrete come aggiungere una scheda Ethernet shield per risolvere questo problema.

Tweet dei progetti Arduino

Uno dei kit hardware più noti è Botanicall (<http://www.botanicalls.com/>), una soluzione che permette di sapere quando le vostre piante hanno bisogno di essere annaffiate grazie alla trasmissione di un messaggio a Twitter. Dopo aver annaffiato le piante l'applicazione invia un messaggio di ringraziamento. La versione ufficiale di Botanicall propone un hardware specifico dedicato al progetto, ma potete realizzare un kit che svolge le medesime funzioni anche a partire da una scheda Arduino (http://www.botanicalls.com/archived_kits/twitter/).

Non c'è dubbio che Botanicall semplifichi la vita di chiunque, mentre si può discutere sull'utilità concreta di Twitwee Clock (<http://www.haroonbaig.com/projects/TwitweeClock/>). Questo orologio a cucù tiene costantemente sotto controllo i canali di Twitter tramite una connessione wireless a Internet. Ogni volta che la scheda di controllo rileva la presenza di un nuovo tweet, l'applicazione visualizza il messaggio corrispondente su un display e il cucù emette un segnale sonoro. Ricordate di chiedere il permesso a chi condivide con voi l'appartamento prima di realizzare questo progetto e appendere l'orologio alle pareti del soggiorno!

Non potete collegare direttamente in rete una scheda Arduino, non solo a causa delle risorse hardware limitate ma soprattutto perché manca una porta Ethernet. Ciò significa che non potete collegare un cavo Ethernet alla scheda di controllo ma per fare questo dovete aggiungere una apposita scheda Ethernet shield. Queste schede shield includono chip e connettori Ethernet che trasformano la scheda di controllo in un dispositivo di

networking. Dovete semplicemente montare la scheda Ethernet a cavallo della scheda di controllo.

Avete a disposizione svariati modelli Ethernet, tutti efficaci per gli scopi richiesti dai progetti tipici di controllo; potete per esempio valutare le proposte disponibili all'indirizzo <http://www.ladyada.net/make/eshield/>.

Nella realizzazione di prototipi si suggerisce di procurarsi la scheda "ufficiale" (<http://www.arduino.cc/en/Main/ArduinoEthernetShield>), che è dotata di connettori relativi a tutti i pin ed è visibile a sinistra nella Figura 8.6. Al momento il team di Arduino ha annunciato anche l'uscita di una scheda Arduino Ethernet, una scheda di controllo che include direttamente una porta Ethernet e non richiede l'inserimento di una scheda shield aggiuntiva.

L'hardware non è il solo elemento da considerare per trasformare Arduino in un dispositivo di networking, dato che la comunicazione in rete richiede anche un apposito software. L'IDE Arduino dispone di una comoda libreria Ethernet che contiene classi in grado di gestire le funzioni di networking. Utilizzerete questa libreria per accedere a un servizio DAYTIME in Internet.

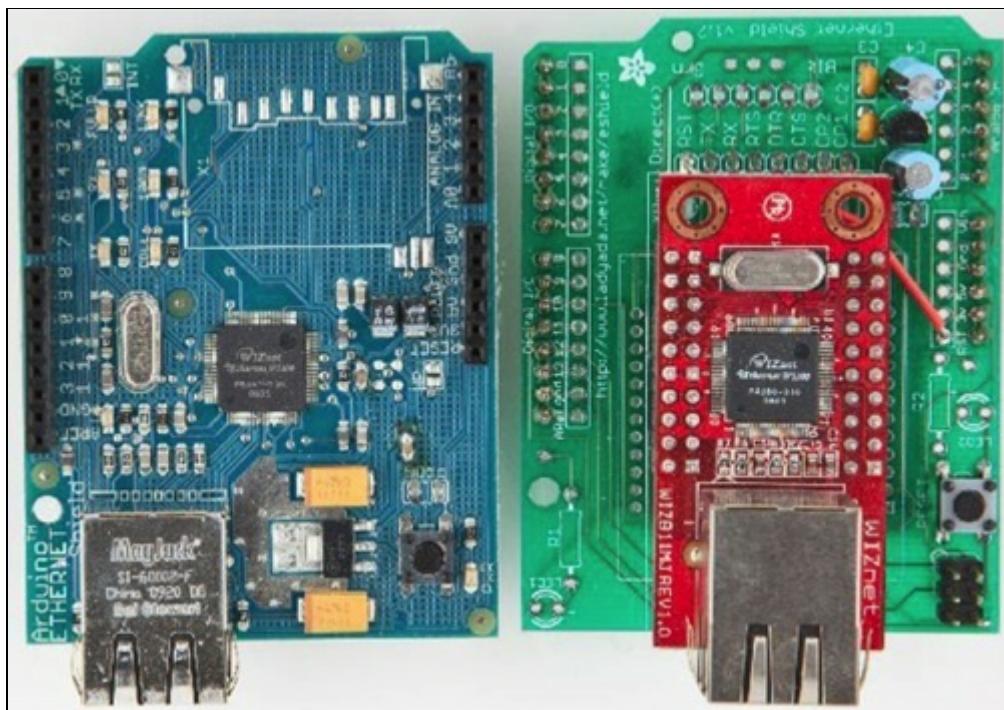


Figura 8.6 Due schede Ethernet shield per Arduino.

Un servizio web DAYTIME (<http://en.wikipedia.org/wiki/DAYTIME>) restituisce data e ora correnti in una stringa ASCII. I server DAYTIME rimangono in ascolto sulla porta 13 TCP o UDP. Potete trovare molti servizi DAYTIME su Internet; uno di questi è offerto da time.nist.gov

(<http://it.wikipedia.org/wiki/NIST>). Prima di impiegare il servizio via software dovete studiarne il funzionamento, utilizzando per esempio questo comando telnet:

```
maik> telnet time.nist.gov 13
Trying 192.43.244.18...
Connected to time.nist.gov.
Escape character is '^]'.
55480 10-10-11 13:25:35 28 0 0 138.5 UTC(NIST) *
Connection closed by foreign host.
```

Il comando telnet si collega al server DAYTIME, che risponde trasmettendo data e ora correnti. La connessione con il server si chiude immediatamente dopo la trasmissione.

Di seguito è riportata l'implementazione delle stesse operazioni da eseguire con una scheda Arduino su cui è montata la scheda Ethernet shield.

Ethernet/TimeServer/TimeServer.pde

```
Riga 1 #include <SPI.h>
      - #include <Ethernet.h>
      -
      -
      - const unsigned int DAYTIME_PORT = 13;
      5 const unsigned int BAUD_RATE = 9600;
      -
      -
      - byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
      - byte my_ip[] = { 192, 168, 2, 120 };
      - byte time_server[] = { 192, 43, 244, 18 }; // time.nist.gov
10
      - Client client(time_server, DAYTIME_PORT);
      -
      -
      - void setup() {
      -   Ethernet.begin(mac, my_ip);
15   Serial.begin(BAUD_RATE);
      - }
      -
      -
      - void loop() {
      -   delay(1000);
20   Serial.print("Connecting... ");
      -
      -
      - if (!client.connect()) {
      -   Serial.println("connection failed.");
      - } else {
25   Serial.println("connected.");
      -   delay(1000);
      - }
```

```

-
-     while (client.available()) {
-         char c = client.read();
30         Serial.print(c);
-
-         Serial.println("Disconnecting.");
-         client.stop();
35     }
-
```

Il programma include in primo luogo la libreria Ethernet e definisce una costante per la porta del servizio DAYTIME; è necessario includere anche la libreria SPI, dato che la libreria Ethernet dipende da questa. Le istruzioni definiscono poi i tre byte array.

- `mac` contiene l'indirizzo MAC della scheda Ethernet shield. Questo indirizzo è un codice numerico di 48 bit che identifica in modo univoco un dispositivo di rete (http://en.wikipedia.org/wiki/Mac_address o http://it.wikipedia.org/wiki/Indirizzo_MAC). In genere è il produttore della scheda che imposta questo codice identificatore, ma nel caso di Ethernet shield la scelta dell'indirizzo è arbitraria e deve essere fatta dall'utente.

ATTENZIONE

È importante ricordare che l'indirizzo MAC deve essere univoco: se collegate più schede Arduino in rete dovete verificare che siano presenti indirizzi MAC diversi uno dall'altro.

- Ogni volta che collegate il computer a Internet è molto probabile che si debba impostare un nuovo indirizzo IP definito dal protocollo DHCP (*Dynamic Host Configuration Protocol*, http://en.wikipedia.org/wiki/Dynamic_Host_Configuration_Protocol o <http://it.wikipedia.org/wiki/DHCP>). Nella maggior parte delle applicazioni Arduino l'implementazione del protocollo DHCP è piuttosto dispendiosa, pertanto si preferisce in genere assegnare manualmente l'indirizzo IP della scheda. L'indirizzo locale presenta di solito un valore che appartiene all'intervallo 192.168.x.y; questo codice va memorizzato nell'array `my_ip`.
- La conversione di nomi di dominio, per esempio `www.nist.gov`, in un indirizzo IP richiede l'accesso al protocollo DNS (*Domain Name System*). La libreria standard di Arduino non supporta però il servizio DNS e

l'indirizzo IP deve essere identificato in modo manuale. Il suo valore va riportato in `time_server`. Il comando `telnet` già incontrato in precedenza converte il nome di dominio del servizio DAYTIME nel corrispondente indirizzo IP. In alternativa, potete utilizzare uno dei comandi riportati di seguito per scoprire l'indirizzo IP relativo a un determinato nome di dominio:

```
maik> host time.nist.gov
time.nist.gov has address 192.43.244.18
maik> dig +short time.nist.gov
192.43.244.18
maik> resolveip time.nist.gov
IP address of time.nist.gov is 192.43.244.18
maik> ping -c 1 time.nist.gov
PING time.nist.gov (192.43.244.18): 56 data bytes
64 bytes from 192.43.244.18: icmp_seq=0 ttl=48 time=173.598 ms

--- time.nist.gov ping statistics ---
1 packets transmitted, 1 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 173.598/173.598/173.598/0.000 ms
```

Alla riga 11 il programma crea un nuovo oggetto `client`. Questa classe fa parte della libreria Ethernet e permette di generare client di networking da collegare a un certo indirizzo IP e a una porta.

A questo punto si deve inizializzare la scheda Ethernet shield tramite la funzione `setup()` visibile alla riga 14. Il programma deve chiamare il metodo `Ethernet.begin()`, cui deve passare gli indirizzi MAC e IP della scheda, poi inizializza la porta seriale in modo da poter visualizzare i messaggi di output. Dopo aver inizializzato i componenti necessari per la comunicazione in rete si può finalmente effettuare la connessione con il server DAYTIME e ottenere da questo i dati desiderati.

Ricordate che la funzione `Ethernet.begin()` accetta anche i parametri relativi all'indirizzo IP del gateway di rete e della subnet mask. Questi dati sono richiesti nel caso in cui la connessione di Arduino a Internet non è diretta ma avviene passando attraverso un router oppure un cable modem. In questo genere di reti potete per esempio passare l'indirizzo di gateway utilizzando queste istruzioni:

```
// ...
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
byte my_ip[] = { 192, 168, 2, 120 };
```

```

byte time_server[] = { 192, 43, 244, 18 }; // time.nist.gov
// Inserite qui sotto l'indirizzo IP del cable modem o del router
byte gateway[] = { 192, 168, 13, 254 };

Client client(time_server, DAYTIME_PORT);

void setup() {
    Ethernet.begin(mac, my_ip, gateway);
    Serial.begin(BAUD_RATE);
}
// ...

```

La funzione `loop()` del progetto Arduino prevede innanzitutto una breve pausa che consente di inizializzare correttamente tutti i componenti hardware. La pausa è richiesta dalla scheda Ethernet shield in quanto si tratta di un dispositivo autonomo che può lavorare in parallelo con la scheda di controllo Arduino. L'istruzione della riga 22 tenta di effettuare una connessione con il servizio DAYTIME. Il programma visualizza un messaggio di errore se non si riesce a stabilire la connessione, altrimenti attende un secondo per dare tempo al servizio web di predisporre i dati, poi legge i dati e li visualizza in output carattere dopo carattere.

È interessante notare che l'interfaccia client è simile all'interfaccia della classe `Serial`. La funzione `available()` permette di verificare se sono disponibili altri dati, mentre `read()` restituisce il byte successivo. Alla fine di questa parte del programma si chiama la funzione `stop()` per scollegare la scheda dal servizio web e consentire l'avvio di una nuova connessione.

Compilate e caricate il programma in Arduino, poi apriete *Serial Monitor* per visualizzare una schermata simile alla seguente:

Connecting...connected.

55480 10-10-11 13:32:23 28 0 0 579.9 UTC(NIST) *

Disconnecting.

Connecting...connected.

55480 10-10-11 13:32:26 28 0 0 34.5 UTC(NIST) *

Disconnecting.

Progetti divertenti di networking con Arduino

I progetti elettronici da indossare come vestiti o *e-textile* (<http://en.wikipedia.org/wiki/E-textiles>) si stanno diffondendo sempre più e sono un ottimo modo per impressionare colleghi e amici. Potete trovare svariati tipi di magliette interattive, in grado per esempio di visualizzare la presenza di reti WiFi o di mostrare l'intensità del rumore ambientale tramite un semplice equalizzatore grafico.

La scheda Arduino Lilypad (<http://arduino.cc/en/Main/ArduinoBoardLilyPad>), una chiavetta Bluetooth e un telefonino Android sono i componenti necessari per realizzare una t-shirt che mostri il numero di messaggi che non avete ancora letto nella vostra casella di posta elettronica (http://blog.makezine.com/archive/2010/03/email-counting_t-shirt.html).

È possibile visualizzare su una maglietta, praticamente in tempo reale, non solo i messaggi di posta non letti ma perfino il vostro stato d'animo, a condizione di predisporre un apposito sensore sulla vostra scrivania e di comunicare il vostro umore in un canale IRC che monitorate con una scheda Arduino (http://blog.makezine.com/archive/2010/01/arduino_powered_mood_meter.html).

I progetti Luminet (<http://openmaterials.org/2010/09/08/luminet/> e <http://luminet.cc>) sono decisamente affascinanti, anche quando non sono sviluppati direttamente dal team di Arduino. Un gruppo di pixel luminosi interconnessi permette di realizzare, per esempio, una giacca interattiva dagli effetti sbalorditivi.

A questo punto avete collegato direttamente Arduino a Internet e siete anche riusciti a trasformare la scheda di controllo in qualcosa di veramente utile: un orologio molto preciso!

Tutto sommato, il networking con Arduino non è poi così diverso dal networking con un computer, a condizione di utilizzare una scheda Ethernet shield. Nel prossimo paragrafo imparerete a inviare messaggi di posta elettronica tramite Arduino.

Librerie di networking

La libreria Ethernet dell'IDE Arduino è abbastanza limitata e non è particolarmente utile, dato che, per esempio, non supporta i protocolli DNS o DHCP. Nei progetti più sofisticati vale la pena considerare la libreria Ethernet messa a disposizione dal progetto Arduino (<http://gkaindl.com/software/arduino-ethernet>).

Per trasformare Arduino in un web server potete invece ricorrere alla libreria Webduino (<http://code.google.com/p/webduino/>), che offre funzionalità interessanti e rappresenta una soluzione ormai affidabile.

Tenete però presente che queste librerie occupano molta memoria e lasciano poche risorse a disposizione dell'applicazione di controllo. Inoltre sono piuttosto instabili, poiché si basano spesso sui componenti interni della libreria Ethernet ufficiale che vengono modificati con una certa frequenza; può quindi succedere che le librerie manifestino problemi di funzionamento con la versione più recente dell'IDE Arduino.

Posta elettronica da riga di comando

Dopo aver appreso come accedere ai servizi di networking si può pensare alla realizzazione di un progetto più ambizioso: un sistema di allarme anti-intrusione che invii un messaggio di posta elettronica se qualcuno si sta muovendo nel vostro soggiorno. Ma come si inviano messaggi di posta elettronica da Arduino?

La posta elettronica è un servizio di networking fondamentale, anche se poche persone ne conoscono a fondo le caratteristiche di funzionamento. Per inviare messaggi da Arduino potete scegliere una soluzione semplice che implica l'utilizzo di un computer per svolgere le operazioni di networking, come è stato fatto in precedenza in questo capitolo. Lo spirito più genuino dell'hacking suggerisce però di adottare una soluzione più

complessa che permetta l'implementazione sulla scheda del protocollo SMTP (*Simple Mail Transfer Protocol*, <http://en.wikipedia.org/wiki/Smtp> o <http://it.wikipedia.org/wiki/SMTp>) che regola il servizio e-mail su Internet. Questo protocollo utilizza solo testo e si basa sostanzialmente su istruzioni da riga di comando; in altre parole, lo scambio di informazioni avviene una riga alla volta. Un messaggio e-mail è composto in genere da un piccolo insieme di attributi: il mittente, il destinatario, l'oggetto e il corpo del messaggio. L'invio di un messaggio richiede la trasmissione di una richiesta a un server SMTP; la richiesta deve essere conforme alle specifiche del protocollo SMTP.

Prima di inviare un messaggio di posta elettronica con Arduino collegato a una scheda Ethernet shield dovete imparare a inviare un messaggio da riga di comando utilizzando il comando `telnet`. In primo luogo dovete conoscere l'indirizzo di un server SMTP da impiegare per l'invio di posta elettronica. Le istruzioni illustrate di seguito si basano sull'ipotesi di avere a disposizione un account di posta (in questo esempio Google Mail, <http://gmail.com>). Ricordate di modificare i nomi dei domini in funzione del provider di posta elettronica a vostra disposizione. In ogni caso non dovete abusare del servizio di posta e dovete leggere preventivamente i termini e le condizioni di utilizzo. Aprite una finestra di terminale e digitate quanto segue:

```
maik> nslookup
> set type=mx
> gmail.com
Server: 192.168.2.1
Address: 192.168.2.1#53
Non-authoritative answer:
gmail.com mail exchanger = 5 gmail-smtp-in.l.google.com.
gmail.com mail exchanger = 10 alt1.gmail-smtp-in.l.google.com.
gmail.com mail exchanger = 20 alt2.gmail-smtp-in.
> exit
```

Questo comando restituisce un elenco dei server MX (*Mail eXchange*) di Google Mail che si possono raggiungere dal vostro computer. Riportate su carta il nome del primo server e apriete una connessione SMTP standard sulla porta 25 come indicato di seguito; ricordate di sostituire il nome del server `gmail-smtp-in.l.google.com` e gli indirizzi di posta elettronica in base alle vostre esigenze:

```
=> maik> telnet gmail-smtp-in.l.google.com 25
<= Trying 74.125.77.27...
Connected to gmail-smtp-in.l.google.com.
Escape character is '^].
220 mx.google.com ESMTP q43si10820020eeh.100
=> HELO
<= 250 mx.google.com at your service
=> MAIL FROM: <arduino@example.com>
<= 250 2.1.0 OK q43si10820020eeh.100
=> RCPT TO: <info@example.com>
<= 250 2.1.5 OK q43si10820020eeh.100
=> DATA
<= 354 Go ahead q43si10820020eeh.100
=> from: arduino@example.com
=> to: info@example.com
=> subject: This is a test
=>
=> Really, this is a test!
=> .
<= 250 2.0.0 OK 1286819789 q43si10820020eeh.100
=> QUIT
<= 221 2.0.0 closing connection q43si10820020eeh.100
Connection closed by foreign host.
```

Questa sessione di comunicazione è simile a quella svolta in precedenza con il servizio DAYTIME, anche se in questo caso le fasi della trasmissione sembrano molto più complesse. Ricordate che non potete scrivere i comandi in lettere maiuscole. In primo luogo si deve inviare il comando HELO (con una sola lettera “L”) per stabilire una connessione con il server SMTP, poi si dice al server che si intende inviare un messaggio e-mail tramite il comando MAIL FROM:. L’indirizzo riportato nel comando sarà impiegato dal server qualora il messaggio dovesse ritornare al mittente. È interessante notare che il server invia una riga di risposta a seguito di ogni richiesta. Le risposte del server iniziano sempre con un codice di stato a tre cifre.

Il comando RCPT TO: imposta l’indirizzo di posta del destinatario. Per inviare un messaggio a più destinatari è sufficiente ripetere questo comando per ogni destinatario.

Il comando DATA segnala al server che si sta per iniziare la trasmissione degli attributi del messaggio di posta. Questi attributi sono composti principalmente da una coppia chiave/valore, dove chiave e valore sono

separati da due punti. Nelle prime tre righe di comando si impostano gli attributi tipici di un messaggio di posta, ovvero mittente (`from`), destinatario (`to`) e oggetto (`subject`).

Gli attributi sono separati dal corpo del messaggio grazie all'inserimento di una riga vuota, mentre la fine del corpo del messaggio è indicata da una riga che contiene solo un punto. Il comando `QUIT` chiude la sessione di comunicazione con il server SMTP.

A questo punto dovreste ricevere un nuovo messaggio nella vostra casella di posta elettronica. Se ciò non si verifica provate a utilizzare un altro server MX. Il protocollo SMTP è abbastanza semplice da un punto di vista teorico ma in pratica può manifestare problemi di ogni genere. Spesso i server SMTP riportano comunque utili messaggi di errore che possono aiutare a risolvere i problemi di trasmissione via Telnet.

Proseguite con il progetto Arduino solo dopo essere riusciti a inviare un messaggio di posta da riga di comando: questa operazione è la base su cui si poggia il programma illustrato nel prossimo paragrafo e che permette di inviare posta elettronica tramite la scheda Arduino.

Posta elettronica tramite la scheda Arduino

In sostanza Arduino trasmette messaggi di posta elettronica implementando riga per riga la sessione `telnet` che avete studiato nel paragrafo precedente. Invece di scrivere direttamente gli attributi del messaggio di posta nel codice di networking vedrete come realizzare una soluzione più sofisticata.

Il programma prevede innanzitutto di definire la classe `Email`, come indicato di seguito.

`Ethernet/Email/email.h`

```
#ifndef __EMAIL_H__
#define __EMAIL_H__

class Email {
    String _from, _to, _subject, _body;

public:
    Email(
        const String& from,
        const String& to,
```

```

const String& subject,
const String& body
) : _from(from), _to(to), _subject(subject), _body(body) {}

const String& getFrom()      const { return _from; }
const String& getTo()       const { return _to; }
const String& getSubject()   const { return _subject; }
const String& getBody()     const { return _body; }

};

#endif

```

Questa classe incapsula i quattro attributi fondamentali di un messaggio di posta, ovvero gli indirizzi di mittente e destinatario, l'oggetto e il corpo del messaggio. Questi attributi sono memorizzati in oggetti nel formato `String`.

Alt, fermi... una classe `String`? Proprio così! A partire dalla versione 19 l'IDE Arduino propone una classe `String`, che non è così ricca di funzioni come le classi omonime dei linguaggi C++ o Java, ma che rappresenta comunque una soluzione migliore rispetto all'utilizzo di più puntatori `char`. L'utilizzo di questa classe verrà spiegato nei prossimi paragrafi.

Le altre istruzioni della classe `Email` dovrebbero risultare abbastanza chiare. Il costruttore di questa classe inizializza le variabili di istanza e sono presenti metodi che permettono di ricavare ogni attributo. A questo punto ci occorre la classe `SmtpService` che invia gli oggetti `Email`.

Ethernet/Email/smtp_service.h

```

Riga 1 #ifndef __SMTP_SERVICE__H__
- #define __SMTP_SERVICE__H__

-
-
- #include "email.h"
5
-
- class SmtpService {
-     byte* _smtp_server;
-     unsigned int _port;
-
10    void read_response(Client& client) {
-        delay(4000);
-        while (client.available()) {
-            const char c = client.read();
-            Serial.print(c);
15        }
-    }
-
```

```

-
-     void send_line(Client& client, String line) {
-         const unsigned int MAX_LINE = 256;
20        char buffer[MAX_LINE];
-         line.toCharArray(buffer, MAX_LINE);
-         Serial.println(buffer);
-         client.println(buffer);
-         read_response(client);
25    }
-
-     public:
-
-     SmtpService(
30        byte*                 smtp_server,
-         const unsigned int port) : _smtp_server(smtp_server),
-                                     _port(port) {}
-
-     void send_email(const Email& email) {
35        Client client(_smtp_server, _port);
-         Serial.print("Connecting...");
-
-         if (!client.connect()) {
-             Serial.println("connection failed.");
40        } else {
-             Serial.println("connected.");
-             read_response(client);
-             send_line(client, String("hello"));
-             send_line(
45            client,
-                 String("mail from: <") + email.getFrom() + String(">")
-             );
-             send_line(
-                 client,
-                 String("rcpt to: <") + email.getTo() + String(">")
50            );
-             send_line(client, String("data"));
-             send_line(client, String("from: ") + email.getFrom());
-             send_line(client, String("to: ") + email.getTo());
55            send_line(client, String("subject: ") +
email.getSubject());
-             send_line(client, String(""));
-             send_line(client, email.getBody());
-             send_line(client, String("."));
-             send_line(client, String("quit"));
60            client.println("Disconnecting.");
-             client.stop();
-         }
-     }
- };

```

- **#endif**

Si tratta in effetti di un numero consistente di istruzioni, ma il programma è abbastanza semplice da comprendere. In primo luogo la classe `SmtpService` incapsula l'indirizzo IP e la porta del server SMTP.

La comunicazione con il server SMTP richiede di leggere le sue risposte e questa operazione è eseguita dal metodo privato `read_response()` definito a partire dalla riga 10. Il programma attende per 10 secondi (in genere i server SMTP sono molto impegnati, dato che devono inviare molti messaggi spam), poi legge i dati inviati dal server e li riporta in output sulla porta seriale per semplificare il debugging delle operazioni svolte.

Prima di elaborare le risposte occorre inviare le richieste utilizzando il metodo `send_line()`, che alla riga 18 invia un singolo comando al server SMTP. Dovete passare la connessione con il server come istanza `client`, mentre la riga di testo da inviare come messaggio è definita come oggetto `String`.

L'invio dei dati memorizzati in un oggetto `String` richiede di accedere ai dati (in formato carattere) cui fa riferimento l'oggetto. Al momento le specifiche Arduino dicono di utilizzare semplicemente `toCharArray()` oppure `getBytes()` per recuperare queste informazioni, ma questa indicazione non è corretta. In altre parole, questi due metodi non restituiscono un puntatore e si aspettano l'impostazione di un array `char` abbastanza grande e l'indicazione della sua dimensione. Per questo motivo il programma copia il contenuto di `line` in `buffer` prima di visualizzare l'output su porta seriale e sulla porta Ethernet, dopodiché legge la risposta del server e la visualizza sulla porta seriale.

L'interfaccia pubblica non presenta sorprese di sorta. Il costruttore si aspetta l'indirizzo IP del server SMTP e la sua porta di connessione. Il metodo `send_email()` è l'elemento più consistente del codice software, ma corrisponde anche alla parte più semplice del programma. Questo metodo riproduce esattamente la sessione `telnet`. Vale la pena sottolineare la gestione delle stringhe, che utilizza la nuova classe `String` di Arduino e l'operatore di concatenamento (+) che trasforma ogni stringa in un oggetto `String`.

A questo punto potete utilizzare le classe per inviare un messaggio e-mail.

Ethernet/Email/Email.pde

```
Riga 1 #include <SPI.h>
- #include <Ethernet.h>
- #include "smtp_service.h"
-
5 const unsigned int SMTP_PORT = 25;
- const unsigned int BAUD_RATE = 9600;
-
- byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
- byte my_ip[] = { 192, 168, 2, 120 };
10
- // Inserite qui sotto l'indirizzo IP del server SMTP!
- byte smtp_server[] = { 0, 0, 0, 0 };
-
- SmtpService smtp_service(smtp_server, SMTP_PORT);
15
- void setup() {
-     Ethernet.begin(mac, my_ip);
-     Serial.begin(BAUD_RATE);
-     delay(1000);
20     Email email(
-         "arduino@example.com",
-         "info@example.net",
-         "Yet another subject",
-         "Yet another body"
25     );
-     smtp_service.send_email(email);
- }
-
- void loop() {}
```

Niente di nuovo. Il programma definisce una serie di costanti, l'indirizzo MAC e altro ancora, poi crea un'istanza `SmtpService`. La funzione `setup()` inizializza la porta seriale e la scheda Ethernet shield, poi attende per un secondo in modo da stabilizzare la configurazione del sistema. Alla riga 20 si crea un nuovo oggetto `Email` e si chiama il suo metodo `send_email()`.

A questo punto sapete come inviare messaggi di posta elettronica utilizzando una scheda Arduino; per realizzare il sistema di allarme anti-intrusione dovete ora imparare a rilevare il movimento in una stanza.

Rilevare il movimento tramite un sensore passivo a infrarossi

Rilevare il movimento è una tecnica utile per molte applicazioni e probabilmente conoscete già dispositivi che accendono la luce in giardino oppure davanti a una porta non appena qualcuno vi si avvicina. La maggior parte di questi dispositivi si basa sulle misure effettuate con un sensore passivo a infrarossi, o PIR (*Passive Infrared Sensor*,

http://en.wikipedia.org/wiki/Passive_infrared_sensor, o più in generale http://it.wikipedia.org/wiki/Radiazione_infrarossa).

Quasi tutti gli oggetti emettono radiazioni a infrarossi e un sensore PIR (Figura 8.7) è in grado di misurare proprio questa porzione di radiazioni. Il controllo del movimento diventa un'operazione abbastanza semplice quando si è in grado di ricevere la radiazione a infrarossi emessa da oggetti che si trovano nel campo di osservazione del sensore. Si supponga per esempio che il sensore riceva la radiazione a infrarossi emessa da una parete e che a un certo punto un essere umano oppure un animale si sposti di fronte alla parete: anche un movimento minimo modifica il segnale a infrarossi ricevuto dal sensore.

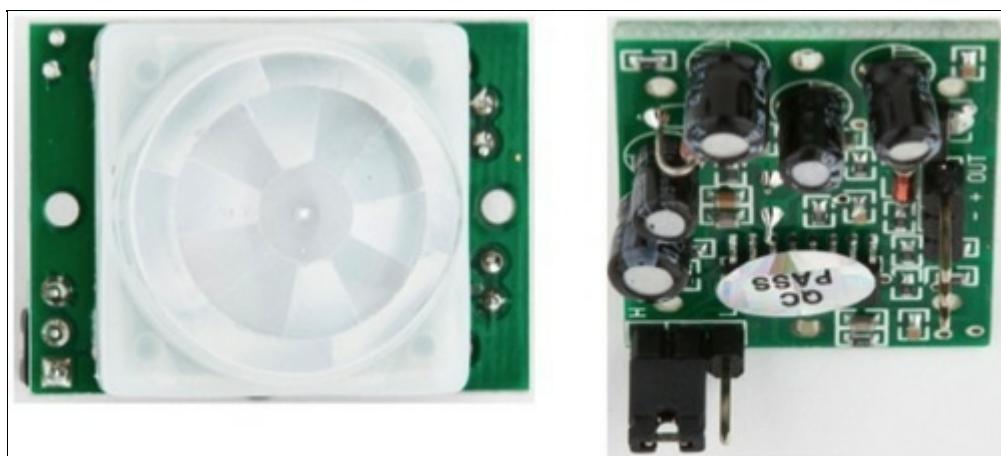


Figura 8.7 Vista dall'alto e dal basso di un sensore PIR.

I sensori di questo tipo non richiedono di conoscere i dettagli tecnici del loro funzionamento e potete utilizzare un solo pin digitale per controllare se qualcuno si sta muovendo nel campo “visivo” del sensore. PIR di Parallax (<http://www.parallax.com/Store/Sensors/ObjectDetection/tabid/176/Product>) è un ottimo esempio di sensore a infrarossi e sarà impiegato come componente hardware principale del progetto illustrato in questo paragrafo.

Il sensore PIR ha tre pin: Power, Ground e Signal. Collegate il terminale Power alla tensione di 5V fornita da Arduino, Ground a uno dei pin GND di Arduino e Signal al pin digitale 2, come potete vedere nella Figura 8.8. Il

sensore presenta anche un ponticello (*jumper*) che potete utilizzare per modificare il funzionamento del componente hardware. In questo progetto dovete collocare il ponticello in corrispondenza della lettera H sul corpo del componente. Per saperne di più sui sensori a infrarossi potete consultare le pagine disponibili all'indirizzo

<http://www.ladyada.net/learn/sensors/pir.html>.

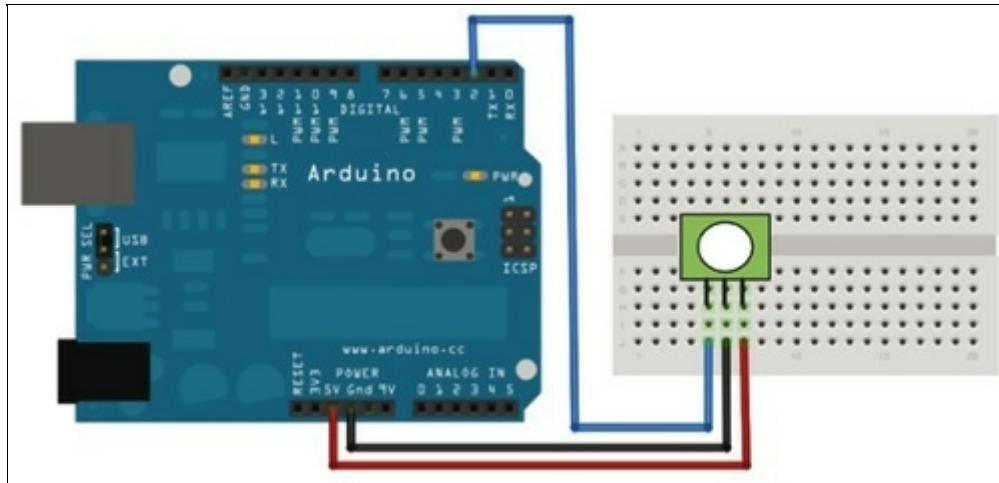


Figura 8.8 Schema di principio del circuito di cablaggio di un sensore PIR.

Di seguito è riportato il codice che dovete scrivere nell'IDE Arduino.

Ethernet/MotionDetector/MotionDetector.pde

```
Riga 1  const unsigned int PIR_INPUT_PIN = 2;
-  const unsigned int BAUD_RATE = 9600;
-
-  class PassiveInfraredSensor {
5    int _input_pin;
-
-  public:
-
-    PassiveInfraredSensor(const int input_pin) {
10      _input_pin = input_pin;
-      pinMode(_input_pin, INPUT);
-    }
-
-    const bool motion_detected() const {
15      return digitalRead(_input_pin) == HIGH;
-    }
-  };
-
-  PassiveInfraredSensor pir(PIR_INPUT_PIN);
20
-  void setup() {
-    Serial.begin(BAUD_RATE);
```

```

-    }

-
25 void loop() {
-    if (pir.motion_detected()) {
-        Serial.println("Motion detected");
-    } else {
-        Serial.println("No motion detected");
30    }
-    delay(200);
-}

```

La costante `PIR_INPUT_PIN` definisce il pin digitale di collegamento con il sensore PIR. Alla riga 4 inizia invece la definizione della classe `PassiveInfraredSensor`, che incapsula tutti gli elementi software che hanno a che fare con i sensori PIR.

La variabile membro `_input_pin` memorizza il numero del pin digitale ove risulta collegato il sensore, poi il programma definisce un costruttore che accetta come argomento il numero di pin e lo assegna alla variabile membro.

Non resta che definire il metodo `motion_detected()`, che restituisce `true` se è stato rilevato un movimento, altrimenti vale `false`. Il programma deve pertanto verificare semplicemente se lo stato corrente del pin digitale collegato al sensore è `HIGH` oppure `LOW`.

Compilate il programma e caricatelo in Arduino; dovreste visualizzare un output simile a quello mostrato nella Figura 8.9 non appena iniziate a muovere una mano di fronte al sensore in funzione. Dopo aver predisposto i due componenti principali del sistema di allarme dovrete assemblare i vari elementi del progetto, in base alle indicazioni fornite nel prossimo paragrafo.



Figura 8.9 Output tipico di un sensore PIR.

Assemblare il progetto

Avendo a disposizione le classi `PassiveInfraredSensor` e `SmtpService` diventa abbastanza semplice realizzare il sistema di allarme anti-intrusione. Collegate il sensore PIR alla Ethernet shield come mostrato nella Figura 8.10 e caricate il programma che segue in Arduino.

Ethernet/BurglarAlarm/burglar_alarm.h

```
Riga 1 #ifndef __BURGLAR_ALARM_H__
- #define __BURGLAR_ALARM_H__
-
- #include "pir_sensor.h"
5 #include "smtp_service.h"
-
- class BurglarAlarm {
-     PassiveInfraredSensor _pir_sensor;
-     SmtpService _smtp_service;
10
-     void send_alarm() {
-         Email email(
-             "arduino@example.com",
-             "info@example.net",
15         "Intruder Alert!",
-             "Someone's moving in your living room!"
-         );
-         _smtp_service.send_email(email);
-     }
20
- public:
-
-     BurglarAlarm(
```

```

-
-     const PassiveInfraredSensor& pir_sensor,
25      const SmtpService&           smtp_service) :
-
-         _pir_sensor(pir_sensor),
-         _smtp_service(smtp_service)
-
-     {
-
-     }
30
-     void check() {
-         Serial.println("Checking");
-         if (_pir_sensor.motion_detected()) {
-             Serial.println("Intruder detected!");
35             send_alarm();
-         }
-     }
- };
-
40 #endif

```

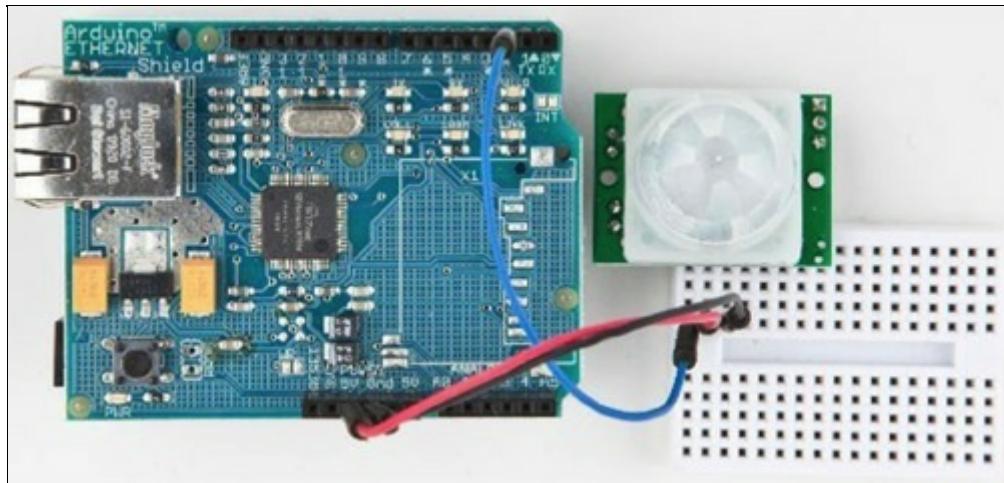


Figura 8.10 Il sistema di allarme anti-intrusione.

Questo programma definisce una classe `BurglarAlarm` che aggrega il codice scritto nei paragrafi precedenti. Il programma incapsula una istanza `SmtpService` e un oggetto `PassiveInfraredSensor`. Il metodo più complesso di questa classe è `send_alarm()`, che permette di inviare un messaggio e-mail preconfezionato.

Le altre istruzioni della classe `BurglarAlarm` dovrebbero risultare abbastanza chiare. Alla riga 23 si definisce il costruttore che inizializza i membri privati della classe, mentre il metodo `check()` verifica se il sensore PIR ha rilevato un movimento e in caso affermativo invia un messaggio di posta elettronica. Ecco il programma che utilizza la classe `BurglarAlarm`.

```

#include <SPI.h>
#include <Ethernet.h>
#include "burglar_alarm.h"

const unsigned int PIR_INPUT_PIN = 2;
const unsigned int SMTP_PORT = 25;
const unsigned int BAUD_RATE = 9600;

byte mac[]      = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
byte my_ip[]    = { 192, 168, 2, 120 };

// Inserite l'IP del server SMTP nella prossima istruzione!
byte smtp_server[] = { 0, 0, 0, 0 };

PassiveInfraredSensor pir_sensor(PIR_INPUT_PIN);
SmtpService           smtp_service(smtp_server, SMTP_PORT);
BurglarAlarm          burglar_alarm(pir_sensor, smtp_service);

void setup() {
  Ethernet.begin(mac, my_ip);
  Serial.begin(BAUD_RATE);
  delay(20 * 1000);
}

void loop() {
  burglar_alarm.check();
  delay(3000);
}

```

In primo luogo si definiscono le librerie richieste dal progetto, poi le costanti relative al pin del sensore PIR e l'indirizzo MAC. A questo punto si impostano gli oggetti `SmtpService` e `PassiveInfraredSensor`, che vanno utilizzati per definire un'istanza `BurglarAlarm`.

Il metodo `setup()` configura la porta seriale e la scheda Ethernet shield. Il programma aggiunge una pausa di 20 secondi, un tempo sufficiente per lasciare la stanza prima che il sistema di allarme inizi a funzionare.

La funzione `loop()` deve semplicemente delegare le operazioni di controllo al metodo `check()` della classe `BurglarAlarm`. Nella Figura 8.11 potete vedere ciò che succede quando il sistema di allarme rileva la presenza di un intruso.



Figura 8.11 Output del sistema anti-intrusione.

Avete notato la semplicità della programmazione orientata agli oggetti in un dispositivo incorporato? La complessità delle operazioni che riguardano l'invio del messaggio di posta e il funzionamento del sensore sono accuratamente nascosti in due piccole classi. Per costruire il sistema di allarme è sufficiente aggiungere il codice che unisce i vari pezzi.

Una considerazione sulla privacy: non abusate del progetto descritto in questo capitolo per spiare le persone senza il loro permesso. È un comportamento sbagliato dal punto di vista etico e in molti paesi è un reato perseguitabile per legge.

In questo capitolo avete imparato modi diversi per collegare a Internet la scheda Arduino. Alcune soluzioni richiedono la presenza di un computer, altre si basano sulla Ethernet shield, ma in ogni caso spalancano le porte allo studio di interessanti applicazioni della scheda di controllo integrata.

Le operazioni di networking sono soluzioni tecniche che hanno una conseguenza immediata nel mondo reale. Nel prossimo capitolo imparerete a utilizzare un'altra tecnica che produce effetti analoghi: vedrete infatti come controllare da remoto il funzionamento di dispositivi elettronici.

Cosa fare se non funziona?

Le reti di trasmissione sono elementi hardware e software decisamente complessi e sono molte le cose che possono andare male quando provate a realizzare gli esempi illustrati in questo capitolo. Vediamo alcuni problemi tipici che potete incontrare.

- Avete indicato una porta seriale non corretta nell'applicazione Processing. L'impostazione di default utilizza la prima porta seriale individuata dal programma, ma può essere che abbiate collegato Arduino a una porta diversa da questa. Se necessario, modificate l'indice 0 nell'istruzione
`arduinoPort = new Serial(this, Serial.list()[0], BAUD_RATE);` in base alla configurazione del vostro sistema.
- Avete dimenticato di collegare il cavo Ethernet alla scheda Ethernet shield.
- Il router di rete prevede una whitelist MAC che permette solo a determinati indirizzi MAC di accedere alle funzioni di networking. Verificate che l'indirizzo MAC della vostra scheda sia compreso nella whitelist. Studiate con attenzione la documentazione del router.
- Avete impostato per due volte lo stesso indirizzo MAC per identificare due dispositivi differenti.
- Avete utilizzato un indirizzo IP non consentito dalla vostra connessione a Internet o che avete già impostato per un altro dispositivo. Verificate con molta cura gli indirizzi IP di tutti i dispositivi.
- Avete configurato credenziali di accesso non corrette per accedere a un servizio web, per esempio a Twitter. Verificate di aver impostato correttamente i token OAuth.
- Twitter non permette di duplicare i messaggi. Ogni volta che l'applicazione non riesce a inviare un messaggio a Twitter verificate di non avere appena trasmesso un messaggio identico.
- Da un paio di decenni le operazioni di networking sono diventate molto affidabili, ma a volte possono essere ancora piuttosto instabili. Può capitare che le connessioni abbiano problemi di funzionamento o che i tempi di trasmissione superino i limiti consentiti. Aumentate le pause dei programmi tenendo conto di queste considerazioni.

Tecnologie di networking alternative

Ethernet è una delle tecnologie di networking più diffuse e potenti. L'impiego di una scheda Ethernet shield permette di collegare a Internet la scheda Arduino come dispositivo client e server.

Le esigenze di progetto possono richiedere l'utilizzo di una connessione wireless. Grazie alla presenza di una scheda WiFi shield (per esempio il modello WiShield, <http://www.asynclabs.com/>, oppure WiFly, <http://www.sparkfun.com/products/9954>) potete facilmente trasformare Arduino in un dispositivo di networking senza fili.

Spesso i progetti non richiedono le funzionalità offerte da Ethernet, in particolare se le comunicazioni avvengono a breve distanza e nell'ambito di una rete personale. Potete scegliere tra soluzioni diverse, ma le tecnologie Bluetooth (<http://it.wikipedia.org/wiki/Bluetooth>) e ZigBee (<http://en.wikipedia.org/wiki/Zigbee> o <http://it.wikipedia.org/wiki/Zigbee>) sono probabilmente le più diffuse. Arduino propone schede in grado di sfruttare entrambe queste tecnologie di networking.

Tenete infine presente che i progetti Arduino possono sfruttare perfino la rete di telefonia cellulare. Collegate una scheda GSM shield (<http://www.hwkitchen.com/products/gsm-playground/>) e una SIM card per iniziare a

Esercizi

- Consultate il Web per trovare altri progetti che utilizzano le schede Ethernet e realizzatene almeno uno; potete per esempio trovare client chat per Arduino all'indirizzo

<http://rapplogic.blogspot.com/2009/11/chatduino-aim-client-for-arduino-wiznet.html>.

- Costruite un progetto simile a quello dell'allarme anti-intrusione ma utilizzate un altro tipo di sensore. Potete trovare suggerimenti utili all'indirizzo

<http://www.tigoe.net/pcomp/code/category/arduinowiring/873>.

- Aggiungete l'indicazione del giorno e dell'ora nel messaggio di posta elettronica trasmesso dall'allarme anti-intrusione. Potete ottenere i dati che vi interessano impiegando un servizio DAYTIME.

Telecomando universale

I telecomandi sono una grande comodità, anche se possono provocare più di una seccatura. A volte non hanno proprio quella funzione che vi piacerebbe avere a disposizione, per esempio un timer per lo spegnimento programmato. Sembrano inoltre riprodursi come conigli e vanno a occupare completamente il tavolino del salotto; perdipiù, i telecomandi richiedono l'uso di batterie, che in genere si scaricano proprio durante un'importante trasmissione sportiva (e naturalmente non avete mai in casa la batteria da sostituire).

I telecomandi universali risolvono alcuni di questi fastidi ma nemmeno i dispositivi più costosi sono perfetti.

I telecomandi vengono utilizzati tutti i giorni ma poche persone ne conoscono il funzionamento. In questo capitolo studierete le caratteristiche di un telecomando e realizzerete un telecomando universale migliore di quello che potete trovare in un qualsiasi negozio di elettronica, visto che potrete personalizzarlo in base alle vostre esigenze. Potrete aggiungere le funzioni che preferite e comandi che gli altri telecomandi non sono in grado di offrire. Questo telecomando permette di aggiungere facilmente nuovi protocolli di controllo e potrà supportare non solo i segnali a infrarossi ma perfino altre modalità di trasmissione senza fili, per esempio le tecnologie Bluetooth o WiFi.

Nei primi paragrafi studierete le caratteristiche dei segnali a infrarossi e realizzerete un primo progetto con un sensore a infrarossi per controllare i codici di qualsiasi telecomando disponiate. Dopo aver acquisito questi codici di controllo potrete emettere lo stesso segnale utilizzando un led a infrarossi, il che rappresenta il primo passo per la realizzazione del telecomando universale.

A questo punto svilupperete ulteriormente l'idea di telecomando “universale”. Dopo aver costruito un primo dispositivo di questo genere vedrete come controllare il telecomando Arduino da porta seriale o tramite una connessione Ethernet. In questo modo sarete in grado di pilotare il

telecomando Arduino utilizzando un browser web e di controllare da Internet il funzionamento del televisore o del lettore DVD (Figura 9.1).

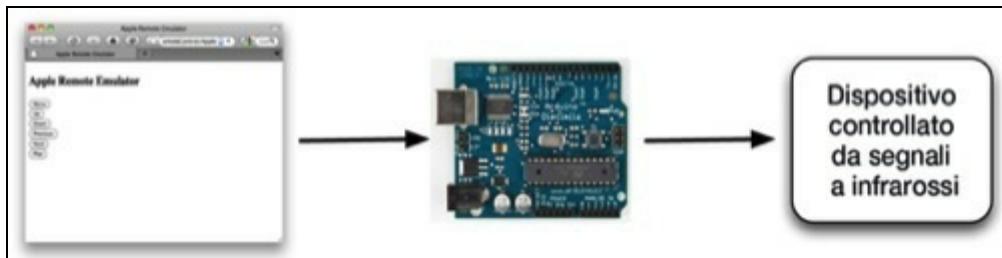


Figura 9.1 Architettura del “proxy” a infrarossi.

Cosa serve

1. Una scheda Ethernet shield per Arduino.
2. Una breadboard.
3. Un ricevitore a infrarossi, preferibilmente il sensore PNA4602.
4. Un resistore da 100Ω .
5. Un led a infrarossi.
6. Cavi di collegamento su breadboard.
7. Uno o più telecomandi a infrarossi. Potete utilizzare per esempio il telecomando del televisore, del lettore DVD o del computer Mac. Gli esempi di questo capitolo fanno riferimento al telecomando Apple Remote; se non lo avete a disposizione, dovrete modificare il nome del protocollo, la lunghezza di un bit e i codici di controllo a seconda del dispositivo che utilizzate nel progetto. Il telecomando di un televisore Sony, per esempio, richiede di impostare il protocollo con il nome SONY, come vedrete più avanti in questo capitolo.
8. Una scheda Arduino, per esempio un modello Arduino Uno, Duemilanove o Diecimila.
9. Un cavo USB per collegare la scheda Arduino al computer.

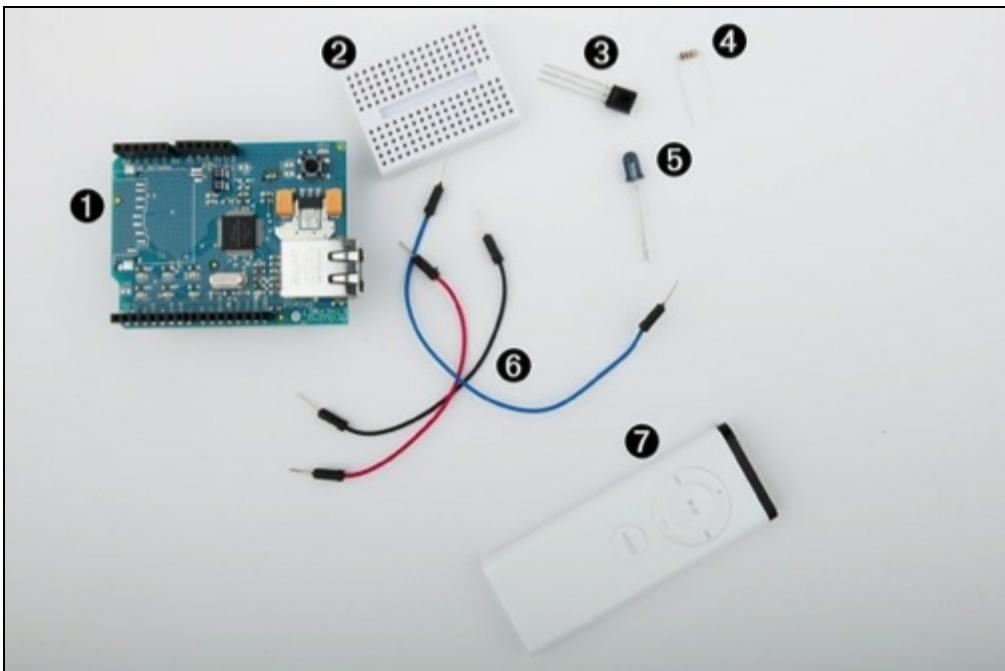


Figura 9.2 Componenti necessari per eseguire i progetti di questo capitolo.

Il telecomando a infrarossi

Il controllo remoto o senza fili di un dispositivo elettronico, per esempio un televisore, si basa sulla presenza di un trasmettitore e di un ricevitore. Il ricevitore è in genere assemblato all'interno dell'apparecchiatura elettronica, mentre il trasmettitore fa parte del telecomando. La trasmissione dei segnali può impiegare svariate tecnologie wireless, per esempio Bluetooth o WiFi, ma la maggior parte dei telecomandi si affida a comunicazioni che utilizzano la luce a infrarossi.

La trasmissione di segnali a infrarossi è una tecnica che presenta molti vantaggi. Questo tipo di segnale è invisibile all'occhio umano, pertanto non disturba chi utilizza il telecomando. Può essere generato a poco prezzo con led a infrarossi, facilmente integrabili nei circuiti elettronici. In sintesi, la scelta dell'infrarosso è una soluzione ottima soprattutto nel controllo dei dispositivi che si trovano solitamente in un ambiente domestico.

La tecnologia dell'infrarosso presenta peraltro una serie di svantaggi. Il segnale non riesce ad attraversare le porte o le pareti e la distanza tra telecomando e dispositivo da controllare deve essere abbastanza ridotta. È ancora più importante sottolineare che il segnale a infrarossi è soggetto a interferenze con altre sorgenti luminose.

Le distorsioni provocate da altre sorgenti luminose possono essere ridotte al minimo producendo un segnale a infrarossi modulato; ciò significa che il

led viene acceso e spento a una certa frequenza, in genere compresa tra 36kHz e 40kHz.

La modulazione del segnale è solo uno dei problemi che complicano la realizzazione di un telecomando a infrarossi veramente affidabile. Il problema più significativo è che i produttori di apparecchiature elettroniche hanno inventato molteplici protocolli di modulazione del segnale, ognuno dei quali è incompatibile con gli altri.

I diversi protocolli utilizzano frequenze differenti e interpretano i dati del segnale in un modo specifico: alcuni interpretano “luce accesa” come bit a 1, altri la associano al bit 0, e ogni protocollo definisce comandi che corrispondono a diverse lunghezze del segnale. In conclusione, è necessario conoscere tutte le proprietà del segnale di un determinato telecomando per riuscire a utilizzare con successo i diversi tipi di protocollo di trasmissione wireless.

Le informazioni sui segnali possono essere acquisite solo adottando una precisa tecnica di analisi del funzionamento di un telecomando. Nei prossimi paragrafi imparerete a leggere i segnali a infrarossi prodotti da un telecomando qualsiasi e vedrete anche come è possibile riprodurli.

I codici di controllo del telecomando

Dato che i telecomandi delle diverse aziende utilizzano raramente lo stesso protocollo o gli stessi comandi, prima di iniziare a trasmettere i codici di controllo è necessario sapere quale tipo di segnale occorre inviare per ottenere un certo risultato; occorre cioè ricavare la maggior quantità possibile di informazioni relative al funzionamento del telecomando che si intende modulare.

Per studiare il funzionamento di un particolare dispositivo a infrarossi vi sono due strategie alternative: la prima comporta l'utilizzo di uno dei database di telecomandi disponibile su Internet, per esempio quello offerto dal progetto Linux Infrared Remote Control (<http://www.lirc.org/>), mentre la seconda implica l'utilizzo di un ricevitore a infrarossi per leggere direttamente i segnali che provengono dal telecomando che si vuole emulare. In questo capitolo si preferisce adottare la seconda soluzione, in quanto permette di apprendere molte informazioni tecniche che si riveleranno utili per i vostri progetti.

I ricevitori a infrarossi (Figura 9.3) sono componenti integrati che hanno un circuito interno molto complesso, ma nonostante ciò sono semplici da utilizzare. I sensori rilevano automaticamente lo spettro di luce a infrarossi a una determinata frequenza, in genere compresa tra 36 kHz e 40 kHz, e riportano la misura effettuata su un solo pin di segnale. L'impiego dei sensori non richiede lo studio dei dettagli tecnici della propagazione della luce e si può concentrare l'attenzione sulla lettura e sull'interpretazione dei segnali provenienti dal telecomando.

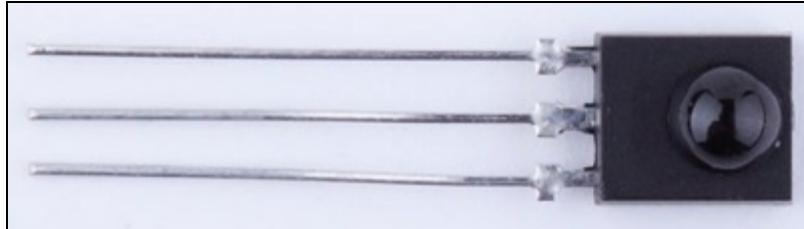


Figura 9.3 Un sensore a infrarossi PNA4602.

Nella Figura 9.4 è mostrato lo schema dei collegamenti tra Arduino e un ricevitore PNA4602. Questo componente è semplice da utilizzare e funziona alla frequenza di 38kHz, pertanto è in grado di rilevare i segnali emessi da un gran numero di dispositivi elettronici. Collegate il connettore di massa del sensore a uno dei pin GND di Arduino, il pin di alimentazione al pin 5V di Arduino e il pin di segnale al pin digitale 11.

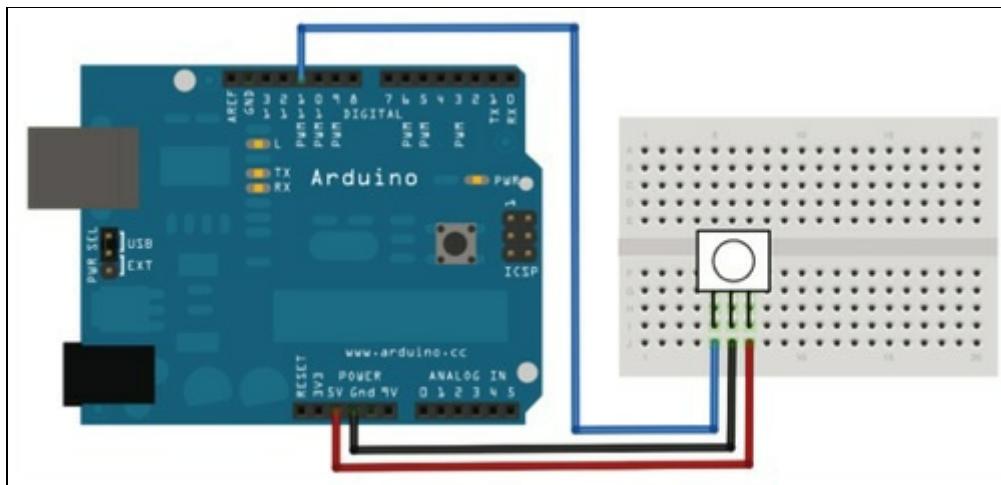


Figura 9.4 Collegare Arduino con il ricevitore a infrarossi è semplice.

Si potrebbe pensare di scrivere un programma che legge e riporta in output i dati ricevuti dal pin 11, ma è bene abbandonare presto questa idea. Provate a chiamare la funzione `digitalRead()` nel metodo `loop()` e visualizzate il risultato sulla porta seriale. Puntate il telecomando del

televisore verso il sensore a infrarossi e osservate ciò che viene visualizzato.

È molto probabile che vi risulti difficile interpretare i dati rilevati dal sensore. Il problema è che la decodifica del segnale è decisamente ardua. Anche se il ricevitore ha già elaborato i dati del segnale luminoso, questi devono essere convertiti e interpretati in base a regole molto complesse. Dovete anche tenere presente che il metodo `digitalRead()` di Arduino non è sempre accurato nella gestione di tutti i tipi di segnale in input. Per ottenere i risultati più efficaci dovreste intervenire direttamente a livello di microcontrollore.

Fortunatamente non dovete effettuare questa operazione in modo manuale perché la libreria IRremote svolge autonomamente le operazioni più specifiche. Questa libreria supporta i protocolli più diffusi per la trasmissione a infrarossi e gestisce l'invio e la ricezione di dati. Scaricate ed estraete il file ZIP della libreria, poi copiate la directory *IRremote* in *~/Documenti/Arduino/libraries* (in un computer Mac) oppure in *Documenti\Arduino\libraries* (in un computer Windows). Infine riavviate l'IDE Arduino.

Di seguito è riportato il programma che permette di decodificare i segnali rilevati dal sensore a infrarossi, a condizione che la libreria IRremote supporti la codifica adottata dal trasmettitore.

[RemoteControl/InfraredDumper/InfraredDumper.pde](#)

```
Riga 1 #include <IRremote.h>
-
- const unsigned int IR_RECEIVER_PIN = 11;
- const unsigned int BAUD_RATE = 9600;
5
- IRrecv ir_receiver(IR_RECEIVER_PIN);
- decode_results results;
-
- void setup() {
10    Serial.begin(BAUD_RATE);
-    ir_receiver.enableIRIn();
- }
-
- void dump(const decode_results* results) {
15    const int protocol = results->decode_type;
-    Serial.print("Protocol: ");
-    if (protocol == UNKNOWN) {
-        Serial.println("not recognized.");
-
```

```

-     } else {
20      if (protocol == NEC) {
-        Serial.println("NEC");
-      } else if (protocol == SONY) {
-        Serial.println("SONY");
-      } else if (protocol == RC5) {
25      Serial.println("RC5");
-      } else if (protocol == RC6) {
-        Serial.println("RC6");
-      }
-      Serial.print("Value: ");
30      Serial.print(results->value, HEX);
-      Serial.print(" (");
-      Serial.print(results->bits, DEC);
-      Serial.println(" bits)");
-    }
35  }

-
-  void loop() {
-    if (ir_receiver.decode(&results)) {
-      dump(&results);
40      ir_receiver.resume();
-    }
-  }

```

In primo luogo si definisce un oggetto `IRrecv` chiamato `ir_receiver` che legge i dati presenti sul pin 11. Il programma definisce anche un oggetto `decode_result` che memorizza gli attributi dei segnali a infrarossi rilevati in input. Il metodo `setup()` inizializza la porta seriale, mentre il ricevitore a infrarossi è inizializzato dal metodo `enableIRIn()`.

A questo punto si definisce il metodo `dump()` che imposta il formato e trasmette in output sulla porta seriale il contenuto di un oggetto `decode_result`. Questo oggetto corrisponde a uno dei tipi di dati fondamentali della libreria IRremote e incapsula una serie di informazioni, per esempio il tipo di protocollo, la lunghezza di un codice di controllo e il codice stesso. Alla riga 15 il programma legge il tipo di protocollo impiegato per decodificare il segnale presente in input. Ogni volta che si riceve un nuovo segnale il programma riporta su porta seriale gli attributi del protocollo corrispondente.

Il metodo `loop()` è abbastanza semplice, in quanto chiama la funzione `decode()` per verificare se è stato ricevuto un nuovo segnale. In caso

affermativo si chiama la funzione `dump()` per riportare in output su porta seriale il segnale ricevuto; la funzione `resume()` fa in modo che il programma attenda la ricezione di un nuovo segnale.

Compilate e caricate il programma in Arduino, poi avviate *Serial Monitor* e puntate un telecomando verso il ricevitore. Premete alcuni tasti del telecomando e osservate ciò che succede. Nella Figura 9.5 potete per esempio vedere ciò che si rileva quando si punta un telecomando Apple Remote verso il ricevitore e si premono i tasti menu, up, down, previous, next e play. A volte può comparire il codice `0xffffffff`; ciò significa che avete tenuto premuto un tasto per troppo tempo, dato che questo segnale corrisponde al “repeat code” che imposta la ripetizione dell’ultimo comando.

Dopo aver acquisito i codici di controllo del telecomando potete sfruttare queste informazioni per realizzare un nuovo telecomando con Arduino, come verrà illustrato nel prossimo paragrafo.

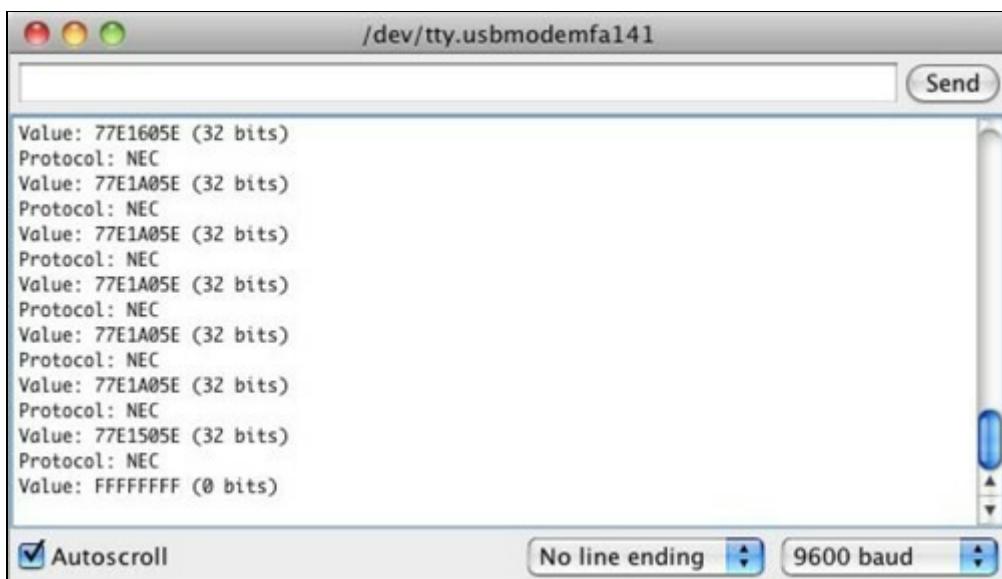


Figura 9.5 Individuazione dei codici di controllo del telecomando Apple Remote.

Realizzare un telecomando personalizzato

Ora che conoscete il protocollo e i codici di controllo impiegati dal telecomando Apple Remote per comunicare con un computer Mac potete passare alla realizzazione di un telecomando Apple Remote personalizzato. Avete solo bisogno di un led a infrarossi, un componente non molto diverso dai led impiegati nei capitoli precedenti; l’unica differenza è data dal fatto che questo led emette luce “invisibile”. Nella Figura 9.6 potete vedere lo schema di cablaggio di un led a infrarossi con il pin 3 di Arduino;

la scelta del pin dipende dalla libreria software utilizzata; in questo progetto si aspetta la connessione tra pin 3 e led a infrarossi. Ricordate che non è possibile collegare un led senza resistore, come verrà spiegato nell'Appendice A.

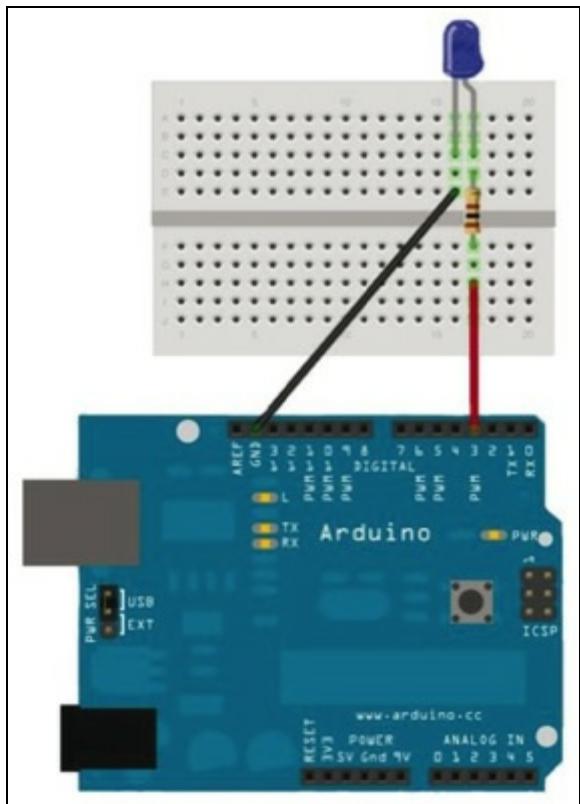


Figura 9.6 Collegamento tra Arduino e led a infrarossi.

Si potrebbe riprodurre direttamente il segnale a infrarossi, ma questa operazione risulterebbe noiosa e sarebbe facile commettere errori. Conviene sfruttare l'implementazione già pronta della libreria IRremote, che permette di creare una classe `AppleRemote` che incapsula i dettagli più insidiosi del protocollo.

`RemoteControl/AppleRemote/AppleRemote.pde`

```
#include <IRremote.h>

class AppleRemote {

    enum {
        CMD_LEN = 32,
        UP = 0x77E15061,
        DOWN = 0x77E13061,
        PLAY = 0x77E1A05E,
        PREV = 0x77E1905E,
        NEXT = 0x77E1605E,
        MENU = 0x77E1C05E
    };
}
```

```

};

IRsend mac;

void send_command(const long command) {
    mac.sendNEC(command, CMD_LEN);
}

public:

void menu() { send_command(MENU); }
void play() { send_command(PLAY); }
void prev() { send_command(PREV); }
void next() { send_command(NEXT); }
void up() { send_command(UP); }
void down() { send_command(DOWN); }
};

```

Queste istruzioni iniziano con una enumerazione che contiene tutte le costanti necessarie: lunghezza di ciascun codice di controllo e codice di controllo vero e proprio. Si prosegue con la definizione di un oggetto `IRsend` chiamato `mac` che sarà utilizzato per inviare i comandi tramite il metodo `send_command()`. Questo metodo fa riferimento alla funzione `sendNEC()` di `IRsend`, dato che il telecomando Apple Remote adotta il protocollo NEC.

Dopo aver impostato gli elementi di base si possono implementare i comandi tramite la chiamata di una sola funzione, per cui il programma continua con l'implementazione dei metodi `menu()`, `play()` e così via.

Anche la classe `AppleRemote` è semplice da utilizzare. Di seguito è riportato un programma che sfrutta questa classe per controllare un computer Mac tramite una schermata *Serial Monitor* di Arduino.

[RemoteControl/AppleRemote/AppleRemote.pde](#)

```

AppleRemote apple_remote;
const unsigned int BAUD_RATE = 9600;

void setup() {
    Serial.begin(BAUD_RATE);
}

void loop() {
    if (Serial.available()) {

```

```
const char command = Serial.read();
switch(command) {
    case 'm':
        apple_remote.menu();
        break;
    case 'u':
        apple_remote.up();
        break;
    case 'd':
        apple_remote.down();
        break;
    case 'l':
        apple_remote.prev();
        break;
    case 'r':
        apple_remote.next();
        break;
    case 'p':
        apple_remote.play();
        break;
    default:
        break;
}
}
```

Questo programma definisce un oggetto globale `AppleRemote` chiamato `apple_remote`, mentre la funzione `setup()` inizializza la porta seriale. Il metodo `loop()` attende l'arrivo di nuovi dati sulla porta seriale e ogni volta che questo si verifica il programma stabilisce se si tratta di un carattere *m*, *u*, *d*, *l*, *r* oppure *p*. Il carattere rilevato corrisponde rispettivamente a un codice di controllo del comando menu, up, down, previous, next oppure play.

Compilate e caricate il programma per controllare un computer Mac tramite *Serial Monitor*, il che rappresenta già un risultato significativo. L’interfaccia è decisamente scomoda per chi non è abituato a maneggiare circuiti elettronici; nel prossimo paragrafo vedrete come realizzare un’interfaccia molto più intuitiva.

Controllo a distanza di dispositivi dal browser

Avete già realizzato molti progetti controllabili tramite *Serial Monitor*. Chi è solito programmare in campo informatico può ritenere che questa

interfaccia sia comoda e pratica da usare, ma è sufficiente mostrare uno di questi progetti a un amico poco abituato a manovrare congegni tecnologici per suscitare perplessità e perché reclami l'esigenza di un'interfaccia molto più intuitiva e usabile.

Il progetto Seriality (<http://www.zambetti.com/projects/seriality/>) propone un componente aggiuntivo (*plug-in*) del browser web che facilita la realizzazione di un'interfaccia intuitiva. Il plug-in aggiunge il supporto delle comunicazioni seriali al motore JavaScript del browser web.

Al momento il plug-in è disponibile solo per Firefox, Safari e Chrome su computer Mac OS X, ma si sta studiando la versione per i browser Windows. L'immagine disco di Seriality si può scaricare all'indirizzo <http://code.google.com/p/seriality/downloads/list>. L'installazione non prevede operazioni particolari.

Dopo aver installato Seriality potete trasformare il browser web in un emulatore del telecomando Apple Remote utilizzando il programma riportato di seguito, che nasce dalla combinazione di istruzioni HTML e JavaScript.

[RemoteControl/AppleRemoteUI/ui.html](#)

```
Riga 1 <html>
-   <title>Apple Remote Emulator</title>
-   <head>
-     <script type="text/javascript">
5       var serial;
-
-
-     function setup() {
-       serial =
(document.getElementById("seriality")).Seriality();
-         alert(serial.ports.join("\n"));
10        serial.begin(serial.ports[0], 9600);
-
-       }
-     </script>
-   </head>
-
-
15   <body onload="setup();">
-     <object type="application/Seriality"
-             id="seriality"
-             width="0"
-             height="0">
20     </object>
-     <h2>Apple Remote Emulator</h2>
```

```

-
- <form>
-   <button type="button" onclick="serial.write('m');">
-     Menu
25   </button>
-   <br/>
-   <button type="button" onclick="serial.write('u');">
-     Up
-   </button>
30   <br/>
-   <button type="button" onclick="serial.write('d');">
-     Down
-   </button>
-   <br/>
35   <button type="button" onclick="serial.write('l');">
-     Previous
-   </button>
-   <br/>
-   <button type="button" onclick="serial.write('n');">
40     Next
-   </button>
-   <br/>
-   <button type="button" onclick="serial.write('p');">
-     Play
45   </button>
-   <br/>
-   </form>
-   </body>
- </html>

```

La pagina HTML è abbastanza semplice; conviene concentrare l'attenzione sulle istruzioni in JavaScript. Alle righe da 4 a 12 si definiscono due elementi: la variabile globale `serial` e la funzione `setup()`. Questa funzione inizializza `serial` e le assegna un oggetto `Seriality`. L'oggetto `Seriality` è incorporato nella pagina web utilizzando il tag `<object>`, il suo `id` è “`seriality`” e pertanto vi si può accedere tramite la funzione

`getElementById()`.

Dopo aver impostato il riferimento dell'oggetto si può chiamare la funzione JavaScript `alert()` e visualizzare in output tutte le porte seriali rilevate nel sistema di controllo. Dovete individuare l'indice della porta seriale ove è collegata la scheda Arduino e utilizzare questo dato nella successiva chiamata del metodo `begin()`. Per maggiore chiarezza, in questo progetto si passa come parametro il primo dispositivo seriale e si imposta un baud rate

di 9600. L'indicazione della prima porta seriale va modificata in base alla configurazione del vostro computer, come è già stato spiegato nei progetti dei capitoli precedenti.

A questo punto si chiama il metodo `setup()` nell'handler di eventi `.onload` dell'elemento `<body>`, che permette di accedere all'oggetto `Seriality` presente negli handler `onclick` dei sei elementi `<button>`.

Caricate il programma che nei paragrafi precedenti ha definito l'emulazione del telecomando Apple Remote e puntate il browser sulla pagina HTML del paragrafo precedente. Fate clic sul pulsante *OK* nella finestra che visualizza le porte seriali per ottenere una pagina web simile a quella mostrata nella Figura 9.7. Fate clic su uno dei pulsanti e controllate l'azione che ne deriva. Questa volta l'interfaccia è decisamente più semplice, vero?

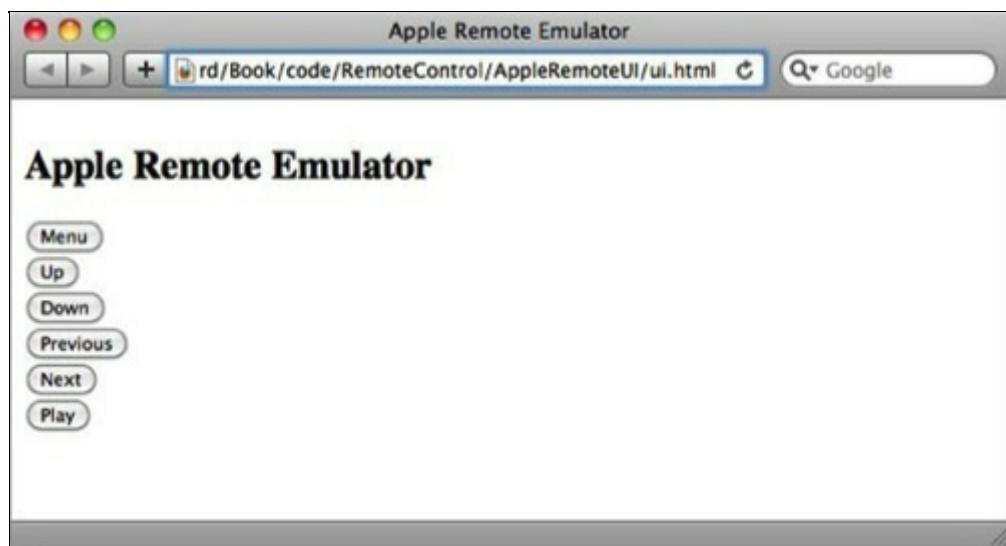


Figura 9.7 Apple Remote Emulator in funzione.

È interessante notare che non potete accedere direttamente all'hardware Arduino tramite Seriality, ma potete solo accedere alla porta seriale. In questo modo tutto ciò che avviene in Arduino deve essere reso accessibile attraverso una comunicazione seriale. In effetti si tratta di una pratica piuttosto comune, pertanto Seriality può essere considerato uno strumento utile per migliorare l'interfaccia utente di qualsiasi progetto Arduino.

A questo punto il progetto prevede di collegare Arduino alla porta seriale del computer per gestire il controllo da browser web. Nel prossimo paragrafo vedrete come superare questo problema e controllare Arduino senza collegamento seriale.

Realizzare un proxy a infrarossi

I progetti di telecomando dei paragrafi precedenti hanno il difetto di dipendere dal collegamento seriale con un computer. In questo paragrafo vedrete come sostituire questa connessione con un collegamento Ethernet, il che non richiede più la presenza di un computer ma solo dell'accesso a Internet. Potrete così collegare direttamente il cavo Ethernet alla scheda Ethernet shield collegata ad Arduino (Figura 9.8) e rendere disponibile il controllo attraverso Internet.

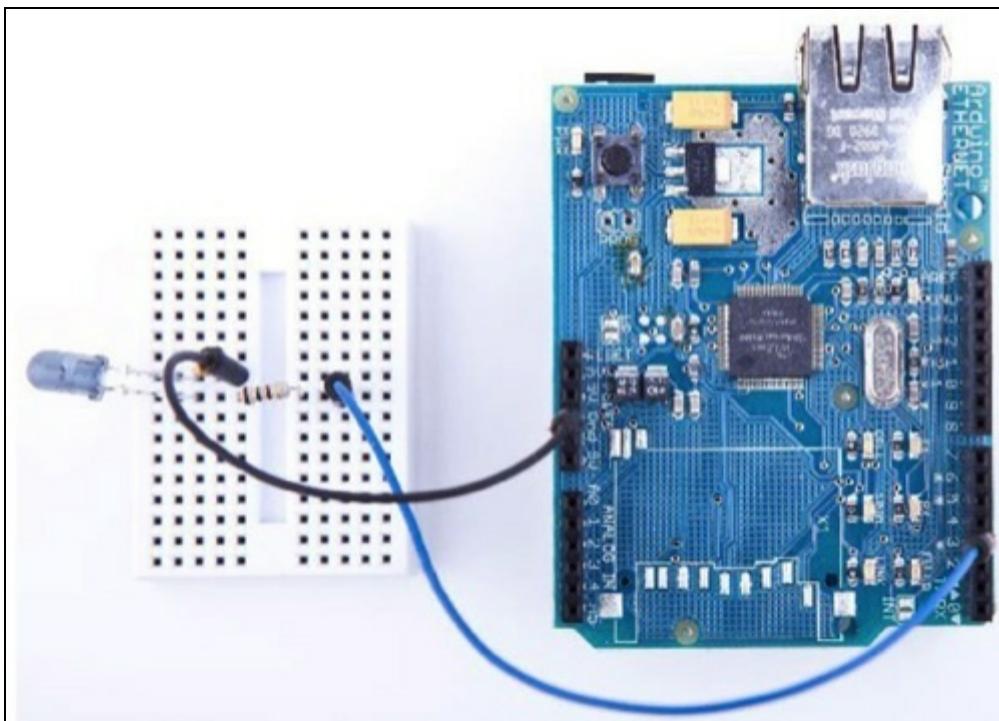


Figura 9.8 Telecomando controllato via Ethernet.

Questa soluzione non significa che dovete utilizzare il browser web del computer per accedere alla scheda Arduino. In effetti potete utilizzare il browser della PlayStation portatile, dell'iPhone o della console Nintendo DS. Proprio così, ora potete controllare la televisione utilizzando le console per videogiochi oppure uno smartphone. Potete anche sostituire la scheda Ethernet shield con una scheda WiFi shield; in questo modo non dovete nemmeno collegare fisicamente Arduino al router Internet.

Prima di studiare il software di questo progetto conviene soffermarsi un attimo e valutare gli obiettivi che si vogliono raggiungere. In sintesi, si vuole realizzare un proxy a infrarossi, ovvero un dispositivo elettronico in grado di ricevere comandi via Ethernet e di convertirli in segnali a infrarossi, come illustrato nella Figura 9.1. L'integrazione del proxy nelle funzioni di networking richiede che questo dispositivo sia accessibile

tramite protocollo HTTP, in modo da poterne controllare il funzionamento utilizzando un browser web.

In questo progetto sarà implementata solo una piccola parte delle istruzioni HTTP standard per la scheda Arduino; in altre parole, il progetto supporterà solo un determinato schema di comandi. Questa è la sintassi dell'URL supportata dal programma:

`http://<ip-arduino>/<nome-protocollo>/<lunghezza-comando>/<codice-comando>`

L'elemento `<ip-arduino>` deve indicare l'indirizzo IP della scheda Ethernet shield di Arduino. L'elemento `<nome-protocollo>` fa riferimento a uno dei protocolli supportati, ovvero "NEC", "SONY", "RC5" oppure "RC6". L'elemento `<lunghezza-comando>` indica la lunghezza del comando espressa in bit, mentre `<codice-comando>` contiene il codice del comando, espresso con un numero decimale.

Si supponga per esempio di inviare il codice relativo al tasto "menu" di un telecomando Apple Remote e che la scheda Arduino abbia l'indirizzo IP 192.168.2.42. In questo caso nel browser web deve essere impostato il seguente URL:

`http://192.168.2.42/NEC/32/2011283550`

Il nome del protocollo in questo esempio è NEC, la lunghezza del codice di comando è di 32 bit e il codice vale 2011283550, numero decimale che corrisponde al valore esadecimale 0x77E1C05E.

Nel Capitolo 8 avete già utilizzato Arduino come web client, mentre in questo progetto vedrete l'impiego di Arduino come web server. Il server attende nuove richieste HTTP simili a quella indicata nell'esempio, analizza l'URL ed emette il segnale a infrarossi corrispondente al comando rilevato dalla scheda.

I dettagli tecnici delle operazioni sono racchiusi nella classe `InfraredProxy` e la soluzione software è semplificata e concisa grazie all'utilizzo delle librerie Ethernet e IRremote. La classe `InfraredProxy` è uno degli esempi di implementazione software più complessi tra quelli illustrati in questo libro, come si può vedere di seguito.

[RemoteControl/InfraredProxy/InfraredProxy.pde](#)

```
Riga 1 #include <SPI.h>
      - #include <Ethernet.h>
      - #include <IRremote.h>
```

```

-
5  class InfraredProxy {
-
-    IRsend _infrared_sender;
-
-    void read_line(Client& client, char* buffer, const int
buffer_length) {
-
-        int buffer_pos = 0;
10     while (client.available() && (buffer_pos < buffer_length - 1)) {
-
-            const char c = client.read();
-
-            if (c == '\n')
-
-                break;
-
-            if (c != '\r')
15         buffer[buffer_pos++] = c;
-
-        }
-
-        buffer[buffer_pos] = '\0';
-
-    }
-
-
20    bool send_ir_data(const char* protocol, const int bits, const long value) {
-
-        bool result = true;
-
-        if (!strcasecmp(protocol, "NEC"))
-
-            _infrared_sender.sendNEC(value, bits);
-
-        else if (!strcasecmp(protocol, "SONY"))
25
-            _infrared_sender.sendSony(value, bits);
-
-        else if (!strcasecmp(protocol, "RC5"))
-
-            _infrared_sender.sendRC5(value, bits);
-
-        else if (!strcasecmp(protocol, "RC6"))
-
-            _infrared_sender.sendRC6(value, bits);
30
-    else
-
-        result = false;
-
-    return result;
-
-}
-
-
35    bool handle_command(char* line) {
-
-    strsep(&line, " ");
-
-    char* path = strsep(&line, " ");
-
-
-    char* args[3];
40
-    for (char** ap = args; (*ap = strsep(&path, "/")) != NULL; )
-
-        if (**ap != '\0')
-
-            if (++ap >= &args[3])
-
-                break;
-
-    const int bits = atoi(args[1]);
45
-    const long value = atol(args[2]);
-
-    return send_ir_data(args[0], bits, value);
-
-}
-
-
```

```

- public:
50
- void receive_from_server(Server server) {
-     const int MAX_LINE = 256;
-     char line[MAX_LINE];
-     Client client = server.available();
55     if (client) {
-         while (client.connected()) {
-             if (client.available()) {
-                 read_line(client, line, MAX_LINE);
-                 Serial.println(line);
-                 if (line[0] == 'G' && line[1] == 'E' && line[2] == 'T')
60                     handle_command(line);
-                 if (!strcmp(line, ""))
-                     client.println("HTTP/1.1 200 OK\n");
-                     break;
-             }
-             }
-             delay(1);
-             client.stop();
70         }
-     }
- };

```

Dopo aver incluso le librerie richieste dal software il programma dichiara la classe `InfraredProxy` e definisce una variabile membro chiamata `_infrared_sender` per memorizzare l'oggetto `IRsend` che va impiegato per emettere i codici di controllo tramite segnali a infrarossi.

Alla riga 8 si definisce il metodo `read_line()`, che legge una riga di dati inviata dal client. La riga si conclude con un carattere newline (`\n`) oppure con un carattere carriage return seguito da newline (`\r\n`). La funzione `read_line()` si aspetta di leggere i dati da un oggetto Ethernet `Client`, un buffer di caratteri in cui memorizzare i dati (`buffer`) e la lunghezza massima del buffer di caratteri (`buffer_length`). Il metodo ignora qualsiasi carattere di newline o carriage return e imposta l'ultimo carattere della riga di dati con `\0`, pertanto il buffer viene riempito con una stringa che si conclude con il carattere null.

Il metodo `send_ir_data()` ha inizio alla riga 20 ed emette un comando a infrarossi in base alle specifiche dettate da tipo di protocollo (`protocol`), lunghezza del codice in bit (`bits`) e valore del codice da inviare (`value`). Il

nome del protocollo stabilisce il metodo delegato a svolgere le operazioni vere e proprie sull’istanza `IRsend`.

Il metodo `handle_command()` implementa uno degli aspetti più complessi della classe `InfraredProxy` in quanto deve analizzare l’URL riportato dalla richiesta HTTP. Per comprendere le operazioni svolte da questo metodo è necessario conoscere il funzionamento delle richieste HTTP. Si supponga di digitare nella barra indirizzi del browser web l’URL

`http://192.168.2.42/NEC/32/2011283550`; il browser invia in questo caso una richiesta HTTP come la seguente:

```
GET /NEC/32/2011283550 HTTP/1.1  
host: 192.168.2.42
```

La prima riga imposta una richiesta GET e il metodo `handle_command()` si aspetta proprio una stringa che contiene una richiesta di questo genere. Le informazioni codificate nel percorso della richiesta (`/NEC/32/2011283550`) sono impiegate per emettere il segnale a infrarossi. L’analisi di queste informazioni è abbastanza complessa, e si semplifica solo utilizzando la funzione C `strsep()`. Questa funzione separa le stringhe delimitate da caratteri speciali, si aspetta come parametri una stringa che contiene alcuni di questi caratteri di separazione e una stringa che definisce quali sono i caratteri da interpretare come tali. La funzione `strsep()` sostituisce ogni occorrenza del carattere di separazione con un carattere `\0` e restituisce un puntatore relativo alla stringa originale. Prima di effettuare questa operazione la funzione sostituisce il puntatore della stringa da suddividere con un puntatore che indica la prima stringa.

La funzione `strsep()` è impiegata i due contesti differenti. Nel primo caso la funzione estrae il percorso dal comando GET, ovvero elimina la stringa “GET” e la stringa “HTTP/1.1”. Entrambe le stringhe sono separate dal percorso grazie alla presenza di un carattere vuoto. Queste operazioni sono effettuate alle righe 36 e 37. Si supponga per esempio di passare l’URL `http://192.168.2.42/NEC/32/2011283550` alla funzione `handle_command()`; in questo caso `path` contiene `/NEC/32/2011283550`.

A questo punto si dispone di una stringa composta da tre stringhe separate da un carattere barra (/) e si può utilizzare di nuovo la funzione `strsep()`, come si può vedere alle righe da 40 a 43, in un modo che dovrebbe essere

familiare a chi conosce la programmazione in C. Alla fine di queste operazioni l'array `args` contiene i tre elementi inclusi nel percorso. Potete passare il nome del protocollo direttamente a `send_ir_data()`, ma prima occorre convertire la lunghezza di un bit e il valore del codice in valori `int` e `long`. La conversione è effettuata dalle funzioni `atoi()` e `atol()`.

Dopo aver definito i metodi helper richiesti dall'elaborazione dati non rimane che implementare l'interfaccia pubblica della classe `InfraredProxy`, che contiene il solo metodo `receive_from_server()`. Questo metodo si occupa dell'implementazione della logica fondamentale della classe `InfraredProxy` e si aspetta un'istanza della classe `Server` definita nella libreria Ethernet. La funzione si aspetta alla riga 54 che un client si colleghi utilizzando il metodo `available()` della classe `Server`. Ogni volta che il server si collega a un client il programma verifica alla riga 57 la presenza di nuovi dati utilizzando il metodo `available()` della classe `Client`.

La funzione `receive_from_server()` legge i dati inviati dal client riga per riga utilizzando il metodo `read_line()` e visualizza ogni riga sulla porta seriale per il debugging delle operazioni; per ogni riga si verifica inoltre la presenza del prefisso "GET". In caso affermativo si chiama `handle_command()`, altrimenti si controlla se la riga è vuota, dato che tutti i messaggi HTTP si concludono con una riga vuota. Se questo è vero, la funzione `receive_from_server()` rimanda una risposta "OK", attende per un millisecondo per dare tempo al client di elaborare la risposta e infine si scollega dal client chiamando la funzione `stop()`.

Il programma è composto da molte istruzioni ma lo sforzo è compensato dalla semplicità di utilizzo di `InfraredProxy`.

RemoteControl/InfraredProxy/InfraredProxy.pde

```
const unsigned int PROXY_PORT = 80;
const unsigned int BAUD_RATE = 9600;

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
byte ip[] = { 192, 168, 2, 42 };

Server server(PROXY_PORT);
InfraredProxy ir_proxy;

void setup() {
    Serial.begin(BAUD_RATE);
```

```

Ethernet.begin(mac, ip);
server.begin();
}

void loop() {
    ir_proxy.receive_from_server(server);
}

```

Analogamente ai progetti precedenti, anche in questo caso si devono impostare gli indirizzi MAC e IP della scheda di controllo, poi si definisce un oggetto `Server` cui si deve passare la porta che trasmette i dati, ovvero la porta 80 che corrisponde alla porta HTTP standard. Il programma deve anche inizializzare un nuovo oggetto `InfraredProxy`.

Il metodo `setup()` inizializza la porta seriale per consentire il debugging del programma. Si deve inizializzare anche la scheda Ethernet shield, poi si chiama il metodo `begin()` della classe `Server` per avviare la trasmissione di dati da parte del server. Il metodo `loop()` deve semplicemente chiamare la funzione `receive_from_server()` della classe `InfraredProxy`, cui deve passare l'istanza `Server`.

Finalmente potete provare il funzionamento del programma! Collegate la scheda Ethernet shield ad Arduino, poi collegate il circuito con il led a infrarossi alla scheda Arduino. Configurate gli indirizzi MAC e IP, compilate e caricate il programma in Arduino. Puntate il browser web all'indirizzo `http://192.168.2.42/NEC/32/2011283550` ricordando di modificare l'URL in base alle impostazioni locali. Osservate ciò che avviene nel computer Mac oppure nel dispositivo che volete controllare a distanza. Nella Figura 9.9 potete vedere un esempio di output prodotto dal proxy a infrarossi e visualizzato da *Serial Monitor*.

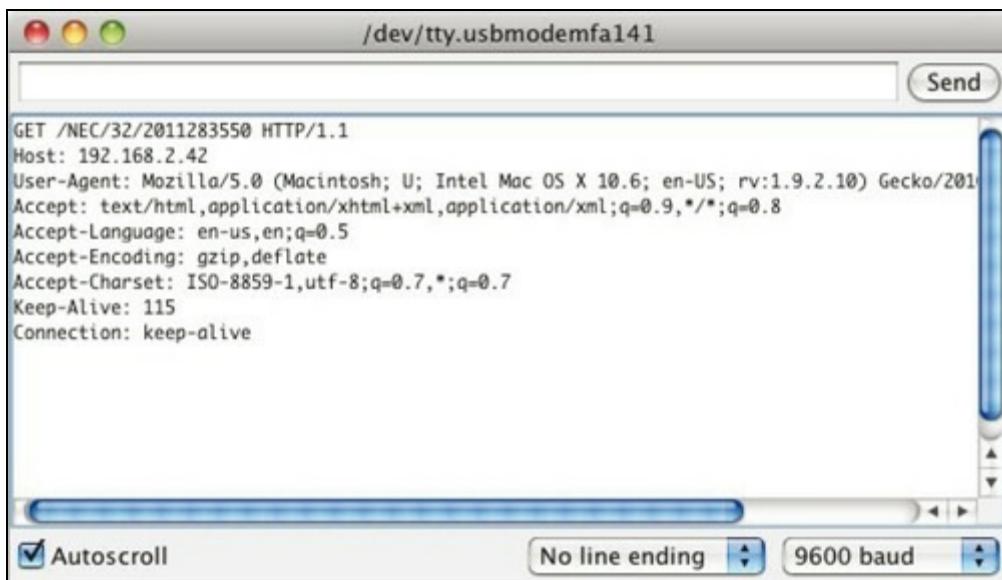


Figura 9.9 Accedere al proxy a infrarossi tramite Firefox.

I progetti di questo capitolo hanno impiegato una quantità minima di componenti hardware (in pratica, solo un piccolo ed economico led a infrarossi), ma i risultati ottenuti sono molto utili e sofisticati, almeno da un punto di vista di sviluppo software. Ora sapete non solo controllare qualsiasi dispositivo in grado di ricevere segnali a infrarossi, ma sapete anche svolgere questa forma di controllo a distanza utilizzando la porta seriale di un computer o perfino un browser web.

Avete anche appreso che non è più necessario collegare Arduino alla porta USB di un computer; il proxy a infrarossi richiede per esempio il collegamento alla porta USB solo per l'alimentazione elettrica delle schede. Collegate un alimentatore in CA ad Arduino e potrete fare a meno del cavo USB.

Per la prima volta siete riusciti a controllare dispositivi elettronici reali utilizzando la scheda Arduino. Nel prossimo capitolo si andrà avanti su questa strada e vedrete come controllare il funzionamento di un motore elettrico.

Cosa fare se non funziona?

In questo capitolo avete utilizzato prevalentemente una serie di led e una scheda Ethernet shield, per cui valgono anche per questi progetti i suggerimenti riportati nel Capitolo 3, dove si sono impiegati gli stessi componenti hardware.

Dovete inoltre prestare attenzione a molti altri aspetti del progetto, che riguardano per esempio la distanza tra un led a infrarossi e il ricevitore.

Controllate tutto

I progetti di questo capitolo si basano su dispositivi che potete già controllare tramite un normale telecomando a infrarossi. Tenete però presente che potete aggiungere un ricevitore a infrarossi a un qualsiasi dispositivo che riceve segnali di controllo oppure realizzare nuovi gadget a partire proprio da un ricevitore a infrarossi. In teoria, per esempio, potreste controllare con un telecomando il funzionamento di un frigorifero o del forno a microonde. Avete mai pensato di realizzare un tosaerba controllato da remoto (<http://www.instructables.com/id/Arduino-RC-Lawnmower/>)? È venuto il momento di farlo!

La soluzione migliore prevede di posizionare il led vicino al ricevitore. I due componenti dovrebbero trovarsi proprio uno di fronte all'altro e dovete anche verificare che la luce ambiente non sia così intensa da disturbare la ricezione del segnale a infrarossi.

In fase di test del progetto si suggerisce di sostituire di tanto in tanto il led a infrarossi (il cui segnale è invisibile) con un led normale (la cui luce è visibile a occhio nudo). In questo modo potete verificare più semplicemente il principio di funzionamento della scheda di controllo.

Se state controllando da remoto un computer Mac dovete ricordare di disabilitare altri telecomandi impostando correttamente le opzioni della sezione *Sicurezza* delle *Preferenze di sistema*.

Ricordate infine che il vostro dispositivo potrebbe utilizzare un protocollo non supportato dalla libreria IRremote. In questo caso dovete aggiungere una nuova classe di funzioni, un'operazione anche molto complessa, ma IRremote è una libreria open source che rende possibile questo genere di modifica.

Esercizi

- Realizzate l'emulatore di uno dei telecomandi che trovate in casa. Fate in modo che suoi comandi siano trasmessi via porta seriale e via Ethernet.
- Invece di controllare Arduino da *Serial Monitor* o browser web, realizzate un controllo che utilizzi Nintendo Nunchuk. Potete per esempio muovere il joystick analogico verso l'alto e verso il basso per controllare il volume del televisore, oppure muovere Nunchuk a sinistra e a destra per cambiare canale.
- Progettate un telecomando universale con Arduino. Procuratevi uno schermo *touch screen*, una tastiera, una scheda SD e un modulo

Bluetooth. Probabilmente non avrete mai bisogno di costruire un dispositivo così articolato, ma adesso sapete almeno che sareste in grado di realizzarlo con Arduino.

Controllo dei motori con Arduino

Finora avete realizzato progetti che hanno un impatto concreto nel mondo reale. Avete fatto lampeggiare dei led e avete controllato il funzionamento di dispositivi tramite segnali a infrarossi. In questo capitolo verrà illustrata un'esperienza dagli effetti ancora più tangibili, ovvero il controllo di motori che spostano oggetti fisici.

Lo sviluppo del progetto non arriverà a proporre la realizzazione di un robot completamente automatico, ma sarete in grado di costruire un piccolo dispositivo che esegue operazioni utili e perfino divertenti.

Prima di iniziare è tuttavia necessario conoscere gli elementi di base dei diversi tipi di motore elettrico e i rispettivi vantaggi e svantaggi. Oggigiorno potete scegliere tra svariati tipi di motore e questo capitolo illustrerà in sintesi una descrizione delle differenze che esistono tra un tipo di motore e un altro.

L'attenzione si soffermerà poi sui servomotori, dato che questo tipo di motore può essere impiegato in molti progetti. I servomotori sono componenti a basso costo e semplici da usare. Imparerete a impiegare la libreria Arduino dedicata a questi componenti hardware e utilizzerete la porta seriale per controllarne il funzionamento.

La conoscenza dei fondamenti vi permetterà di realizzare un progetto più sofisticato, ovvero un dispositivo che sfrutta praticamente gli stessi componenti hardware del primo progetto di questo capitolo ma con un software di controllo decisamente più complesso. Il dispositivo si dimostrerà molto utile anche nel vostro ufficio!

Cosa serve

1. Un servomotore, per esempio un servo Hitec HS-322HD.
2. Cavi di collegamento su breadboard.
3. Un sensore di temperatura TMP36 (facoltativo; è richiesto solo per gli esercizi).
4. Una scheda Arduino, per esempio un modello Arduino Uno,

Duemilanove o Diecimila.

5. Un cavo USB per collegare la scheda Arduino al computer.

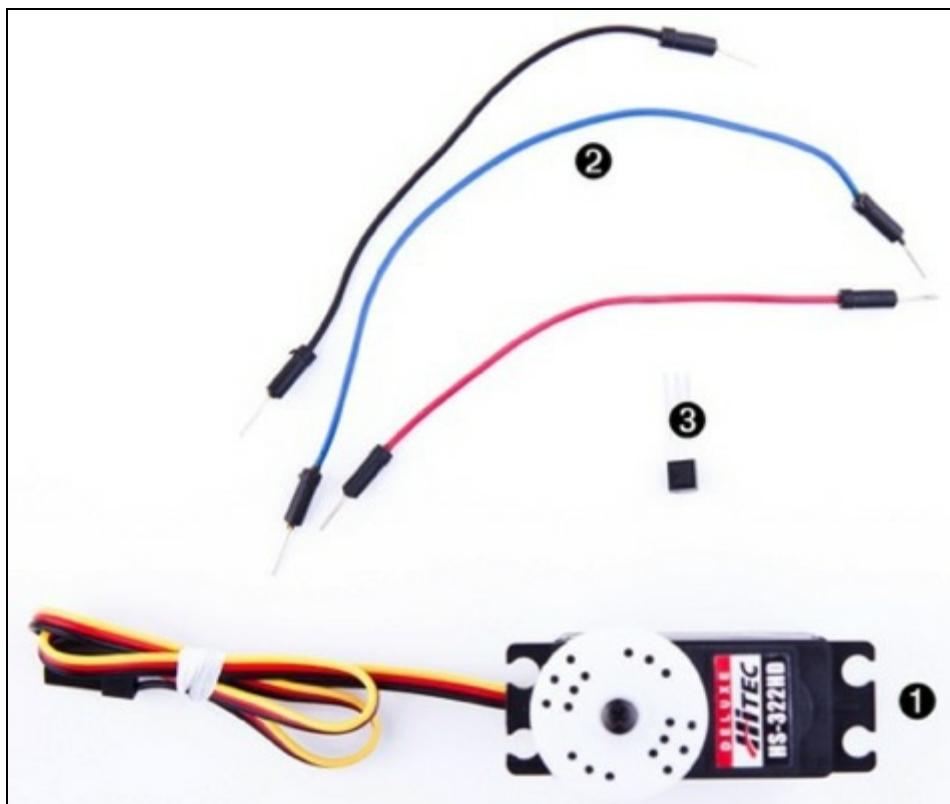


Figura 10.1 Componenti necessari per eseguire i progetti di questo capitolo.

I motori per il controllo elettronico

Le specifiche di progetto portano a scegliere un determinato tipo di motore piuttosto che un altro. Nei progetti di elettronica amatoriale si utilizzano in genere motori in continua, servomotori o motori passo-passo. Nella Figura 10.2 sono visibili alcuni tipi di motore, a esclusione del più semplice motore in continua. I motori elettrici si differenziano per velocità, precisione del controllo, consumo di energia elettrica, affidabilità e prezzo.

I motori in continua sono veloci ed efficienti, pertanto sono in genere impiegati nei trapani elettrici, nelle biciclette elettriche e nelle automobili telecomandate. Potete controllare facilmente il funzionamento di un motore in continua, dato che questo componente ha due soli connettori.



Figura 10.2 Tipi di motore (da sinistra a destra): servomotore standard, motore a rotazione continua, motore passo-passo.

Collegate uno dei connettori alla tensione elettrica e l’altro a massa per mettere in moto il meccanismo. Invertite le connessioni e vedrete il motore ruotare in senso contrario. Aumentate la tensione di alimentazione e vedrete il motore ruotare più velocemente; abbassate la tensione e la velocità di rotazione diminuirà.

I motori in continua non sono la scelta ideale quando si vuole ottenere una regolazione precisa del funzionamento. In questi casi conviene impiegare un motore passo-passo, che permette di realizzare un controllo accurato in un intervallo di 360 gradi. Forse è difficile rendersene conto, ma in effetti siamo circondati da motori passo-passo, per esempio nella stampante, nello scanner e nel disco fisso del computer. Il controllo dei motori passo-passo non è roba da fantascienza, anche se richiede l’esecuzione di operazioni più complesse di quelle di un motore in continua o di un servomotore.

I servomotori sono i componenti preferiti tra chi si occupa di elettronica per hobby, dato che offrono un buon compromesso tra i motori in continua e quelli passo-passo. Hanno un prezzo accessibile, sono affidabili e semplici da controllare. Il movimento di un servomotore può essere regolato solo in un intervallo di 180 gradi, che è però sufficiente per realizzare molte applicazioni. I motori a rotazione continua aumentano

l'intervallo di regolazione a 360 gradi, ma il controllo del funzionamento diventa più complesso.

Nel prossimo paragrafo vedrete come è facile controllare con Arduino il funzionamento di un servomotore standard.

Utilizzo di base di un servomotore

L'IDE Arduino offre una libreria per il controllo dei servomotori da impiegare nei primi progetti. Nella Figura 10.3 è visibile lo schema di cablaggio che permette di collegare Arduino con un servomotore. Collegate il cavo di massa a uno dei pin GND di Arduino, il cavo di alimentazione al pin 5V e il cavo di controllo al pin 9.

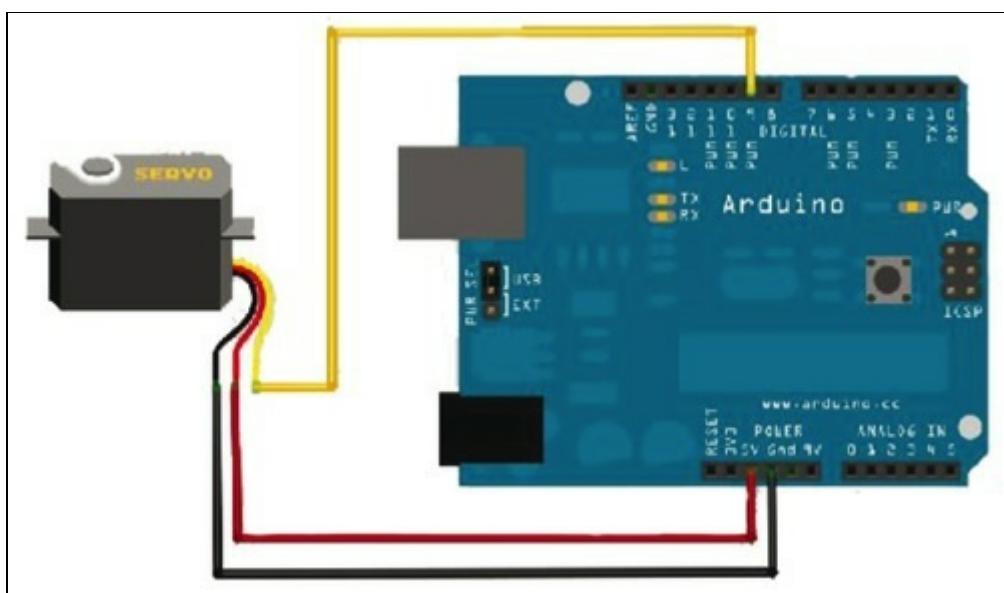


Figura 10.3 Schema di cablaggio per il collegamento di un servomotore a 5V.

ATTENZIONE

Ricordate che questo schema può essere utilizzato solo con servo a 5V! Molti servomotori economici utilizzano 9V di tensione e richiedono pertanto un'alimentazione esterna; in questo caso non potete collegare l'alimentazione del servo al pin 5V di Arduino. Se avete a disposizione un servomotore a 9V, collegate l'alimentatore esterno (per esempio un alimentatore da CA in CC oppure una batteria da 9V) al jack di alimentazione della scheda Arduino. A questo punto collegate il servomotore al pin Vin; per saperne di più consultate le indicazioni fornite all'indirizzo

<http://www.arduino.cc/playground/Learning/WhatAdapter>. Leggete con attenzione anche le specifiche della scheda Arduino; per esempio, non potete utilizzare la scheda Arduino BT (<http://arduino.cc/en/Main/ArduinoBoardBluetooth>) per controllare i motori, dato che questa scheda può essere alimentata con una tensione massima di 5,5V.

La Figura 10.4 mostra la connessione via cavo tra un servomotore e una scheda Arduino. Potete sfruttare i connettori dei pin, ma l'impiego dei cavi elettrici garantisce una maggiore flessibilità del collegamento.

Il controllo dei servomotori è un'operazione abbastanza comoda, dato che potete impostare la posizione dell'albero motore con un angolo compreso tra 0 e 180 gradi. Di seguito è riportato un programma che trasmette da porta seriale un valore in gradi e posiziona il servomotore in base al valore trasmesso.

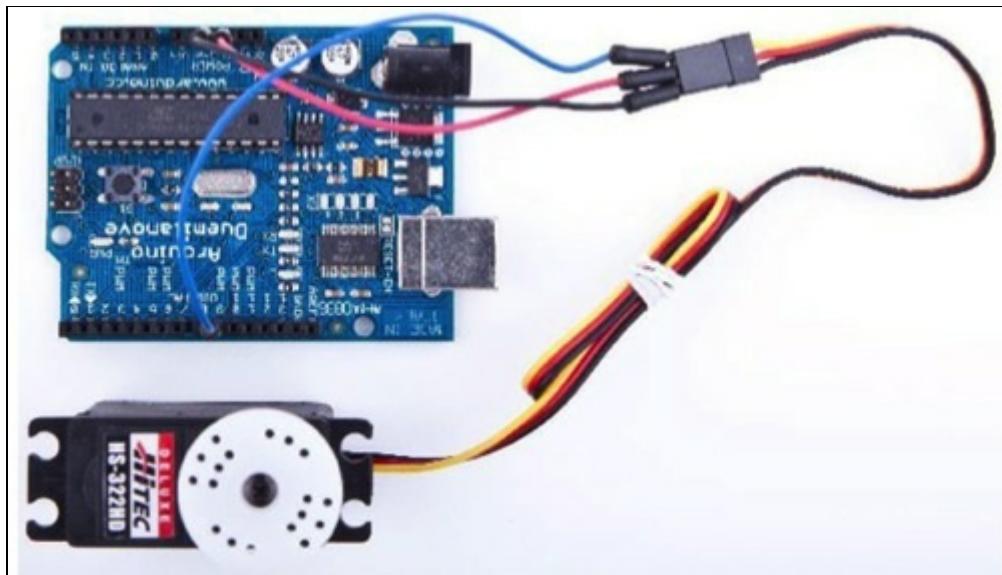


Figura 10.4 Dovete inserire tre cavi nel connettore del servo per collegare il motore ad Arduino.

Motors/SerialServo/SerialServo.pde

```
Riga 1 #include <Servo.h>
-
-
- const unsigned int MOTOR_PIN = 9;
- const unsigned int MOTOR_DELAY = 15;
5 const unsigned int SERIAL_DELAY = 5;
- const unsigned int BAUD_RATE = 9600;
-
-
- Servo servo;
-
-
10 void setup() {
-     Serial.begin(BAUD_RATE);
-     servo.attach(MOTOR_PIN);
-     delay(MOTOR_DELAY);
-     servo.write(1);
15     delay(MOTOR_DELAY);
- }
-
-
- void loop() {
-     const int MAX_ANGLE = 3;
20
-     char degrees[MAX_ANGLE + 1];
```

```

-     if (Serial.available()) {
-         int i = 0;
25       while (Serial.available() && i < MAX_ANGLE) {
-             const char c = Serial.read();
-             if (c != -1 && c != '\n')
-                 degrees[i++] = c;
-                 delay(SERIAL_DELAY);
30       }
-       degrees[i] = 0;
-       Serial.print(degrees);
-       Serial.println(" degrees.");
-       servo.write(atoi(degrees));
35       delay(MOTOR_DELAY);
-   }
- }
```

Il programma include la libreria Servo e alla riga 8 definisce un nuovo oggetto `Servo`. La funzione `setup()` inizializza la porta seriale, mentre la funzione `attach()` collega l'oggetto `Servo` al pin definito in `MOTOR_PIN`. Al termine di queste operazioni il programma attende per 15 millisecondi, in modo da garantire il tempo necessario affinché il motore esegua il comando, poi chiama la funzione `write()` per riportare il servomotore nella posizione corrispondente a 1 grado. Si potrebbe riportare il servomotore a 0 gradi, ma alcuni componenti utilizzati nelle prove di questo progetto hanno manifestato problemi di rumore quando sono stati collocati in questa posizione.

Lo scopo principale della funzione `loop()` è la lettura sulla porta seriale di nuovi valori numerici espressi in gradi. Questi valori sono compresi nell'intervallo tra 0 e 180 e sono letti in formato ASCII. Il programma deve pertanto prevedere una stringa che contiene fino a quattro caratteri, tenendo presente che in C le stringhe devono sempre terminare con un carattere null. Per questo motivo alla riga 21 si dichiara una stringa `degrees` di lunghezza 4.

A questo punto il programma attende l'arrivo sulla porta seriale di nuovi dati, che vengono letti carattere per carattere fino a quando non sono più disponibili nuovi dati oppure fino a quando sono stati letti dati sufficienti per proseguire. La stringa si conclude con un byte zero e si visualizza il valore letto sulla porta seriale. Il programma deve infine convertire la stringa in un valore intero tramite la funzione `atoi()`, dopodiché il valore è passato al metodo `write()` dell'oggetto `Servo`, come si può vedere alla riga

34. Il programma attende di nuovo che il servomotore svolga l'operazione indicata dal comando.

Compilate e caricate il programma, poi apriete una finestra *Serial Monitor*. Dopo aver inizializzato il servomotore provate a inviare alcuni valori, per esempio 45, 180 oppure 10. Osservate che il motore ruota fino a raggiungere la posizione indicata dal comando. Per valutare meglio l'effetto di rotazione ritagliate in forma di freccia un pezzo di cavo o di carta e fissatelo all'albero del motore.

È abbastanza facile controllare la posizione di un servomotore tramite la porta seriale, e il circuito appena realizzato costituisce la base di partenza per lo sviluppo di progetti utili e divertenti. Nel prossimo paragrafo vedrete come realizzare un dispositivo automatico particolarmente curioso.

Realizzare il Blaminatr

Puntare il dito per dare la colpa a qualcuno non è mai piacevole, anche se a volte dà grandi soddisfazioni. In questo paragrafo vedrete come realizzare un Blaminatr (dall'inglese *to blame*, ovvero incolpare qualcuno), che permette di lasciare al dispositivo la responsabilità di “dare la colpa” al posto nostro.

Nella Figura 10.5 si può osservare il dispositivo in azione: il Blaminatr ha deciso di “puntare il dito” nei confronti di Maik, l'autore di questo libro.

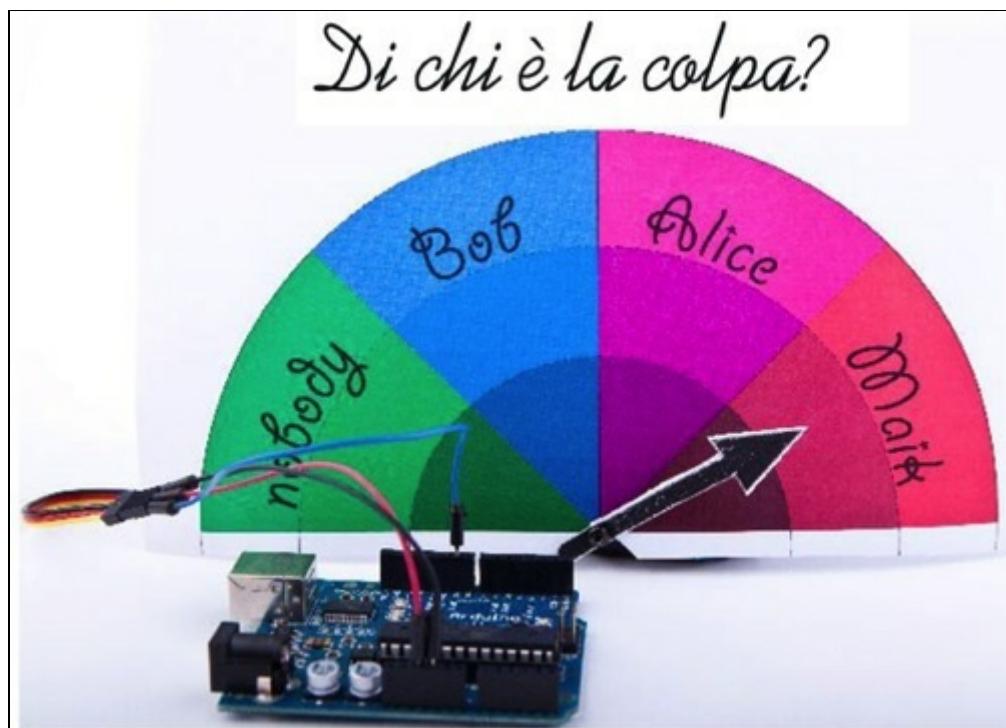


Figura 10.5 Blaminatr: dare la colpa a qualcuno non è mai stato così semplice.

L'arte di Arduino

Potete utilizzare Arduino non solo per realizzare gadget più o meno divertenti ma anche per progettare soluzioni significative da un punto di vista artistico. Potete trovare proposte intriganti di progetto in particolare nelle creazioni che si rifanno ai nuovi media. Uno dei progetti più interessanti è Anthros (<http://www.richgilbank.ca/anthros>), un ambiente che osserva un'area tramite una webcam. L'area tenuta sotto controllo da Anthros contiene una serie di "tentacoli" che si muovono verso chiunque l'attraversa. I servomotori muovono i tentacoli e una scheda Arduino controlla il funzionamento dei servomotori.

RIFERIMENTO

Chi è interessato ad approfondire il tema dei nuovi media nelle creazioni artistiche deve assolutamente leggere la tesi di Alicia Gibb intitolata *New Media Art, Design, and the Arduino Microcontroller: A Malleable Tool* (<http://aliciagibb.com/thesis/>).

I Blaminatr sono gadget che potete utilizzare nei vostri uffici in molte situazioni. Chi sviluppa software a livello professionale potrebbe per esempio aggiungere un Blaminatr al proprio sistema CI (*Continuous Integration*) per compilare il software ed eseguire i test in automatico (http://en.wikipedia.org/wiki/Continuous_integration); tra i sistemi CI più noti si ricordano CruiseControl.rb (<http://cruisecontrolrb.thoughtworks.com/>) e Luntbuild (<http://luntbuild.javaforge.com/>).

Ogni volta che uno sviluppatore aggiunge una modifica, il sistema CI compila automaticamente la soluzione software ed esegue una serie completa di test, di cui pubblica i risultati via posta elettronica oppure utilizzando un feed RSS. Potete scrivere facilmente un'applicazione che sottoscriva questo genere di feed RSS. Ogni volta che qualcuno blocca lo sviluppo del progetto viene notificato un nuovo feed RSS e Blaminatr permette di puntare con la freccia il nome dell'ultimo sviluppatore che ha confermato le ultime modifiche. All'indirizzo

http://urbanhonking.com/ideasfordozens/2010/05/19/the_github_stoplight/ potete trovare un progetto alternativo a questo che prevede l'impiego di un semaforo per segnalare lo stato corrente dello sviluppo del progetto.

Nel paragrafo precedente avete imparato ciò che serve per utilizzare i servomotori nella realizzazione di Blaminatr. A questo punto dovete solo ricorrere a un po' di creatività per costruire il display del dispositivo e aggiungere il programma che svolga le operazioni necessarie. Le istruzioni devono innanzitutto definire una classe `Team` che rappresenti i membri del team del vostro ufficio, in altri termini i potenziali "colpevoli" cui riversare ogni responsabilità.

```

Riga 1 const unsigned int MAX_MEMBERS = 10;
-
- class Team {
-     char** _members;
5     int _num_members;
-     int _positions[MAX_MEMBERS];
-
- public:
-
10    Team(char** members) {
-        _members = members;
-
-        _num_members = 0;
-        char** member = _members;
15        while (*member++)
-            _num_members++;
-
-        const int share = 180 / _num_members;
-        int pos = share / 2;
20        for (int i = 0; i < _num_members; i++) {
-            _positions[i] = pos;
-            pos += share;
-        }
-    }
25
-    int get_position(const char* name) const {
-        int position = 0;
-        for (int i = 0; i < _num_members; i++) {
-            if (!strcmp(_members[i], name)) {
30                position = _positions[i];
-                break;
-            }
-        }
-        return position;
35    }
- };

```

Queste istruzioni definiscono molte variabili membro: `_members` contiene un elenco che comprende fino a dieci membri del team, `_num_members` contiene il numero corrente di persone nel team, mentre nella variabile `_positions` si memorizza la posizione (angolo) del nome di ciascun membro del team visualizzato dal display di Blaminatr.

Il costruttore si aspetta un array di stringhe che contiene i nomi dei membri del team e che deve terminare con un puntatore NULL. Il programma

memorizza il riferimento a un elenco, poi calcola il numero di membri del team. L'operazione si ripete in un ciclo fino a quando si incontra il puntatore NULL. Questa elaborazione è impostata nelle righe da 13 a 16.

A questo punto il programma calcola la posizione di ciascun nome nel display di Blaminatr. Ogni membro del team occupa un intervallo di valori nel display a 180 gradi; Blaminatr deve puntare ciascun intervallo al centro, pertanto si divide ogni intervallo per 2. Le diverse posizioni sono memorizzate nell'array `_positions` che corrisponde all'array `_members`; ciò significa che la prima voce di `_positions` contiene la posizione del primo membro del team e così via.

Il metodo `get_position()` permette di ricavare la posizione che appartiene a un determinato nome. Il programma scorre l'array `_members` fino a individuare il membro indicato dalla funzione `strcmp()`, e dopo aver individuato la corrispondenza restituisce la voce associata a questa nell'array `_positions`. Se il programma non trova alcuna corrispondenza il valore restituito è 0. Di seguito è riportata l'implementazione della classe Blaminatr.

Motors/Blaminatr/Blaminatr.pde

```
#include <Servo.h>

const unsigned int MOTOR_PIN = 9;
const unsigned int MOTOR_DELAY = 15;

class Blaminatr {
    Team _team;
    Servo _servo;

public:
    Blaminatr(const Team& team) : _team(team) {}

    void attach(const int sensor_pin) {
        _servo.attach(sensor_pin);
        delay(MOTOR_DELAY);
    }

    void blame(const char* name) {
        _servo.write(_team.get_position(name));
        delay(MOTOR_DELAY);
    }
};
```

L’oggetto `Blaminatr` aggrega un oggetto `Team` e un oggetto `Servo`. Il costruttore inizializza l’istanza `Team` mentre l’istanza `Servo` è inizializzata dal metodo `attach()`.

Il metodo più interessante del programma è `blame()`. Questa funzione si aspetta il nome del membro del team da “colpevolizzare”, calcola la sua posizione e sposta il motore in base a questi dati. Di seguito è riportato il programma che gestisce tutte le operazioni.

Motors/Blaminatr/Blaminatr.pde

```
Riga 1  const unsigned int MAX_NAME = 30;
-  const unsigned int BAUD_RATE = 9600;
-  const unsigned int SERIAL_DELAY = 5;
-
5   char* members[] = { "nobody", "Bob", "Alice", "Maik", NULL };
-  Team team(members);
-  Blaminatr blaminatr(team);
-
-  void setup() {
10    Serial.begin(BAUD_RATE);
-    blaminatr.attach(MOTOR_PIN);
-    blaminatr.blame("nobody");
-  }
-
15  void loop() {
-    char name[MAX_NAME + 1];
-    if (Serial.available()) {
-      int i = 0;
-      while (Serial.available() && i < MAX_NAME) {
20        const char c = Serial.read();
-        if (c != -1 && c != '\n')
-          name[i++] = c;
-          delay(SERIAL_DELAY);
-      }
25    name[i] = 0;
-    Serial.print(name);
-    Serial.println(" is to blame.");
-    blaminatr.blame(name);
-  }
30 }
```

Queste istruzioni definiscono un elenco di nomi terminato da un puntatore `NULL`. La prima voce dell’elenco è `"nobody"`, il che permette di gestire in

modo automatico perfino la possibilità molto rara che non ci sia qualcuno cui dare la colpa.

Il programma utilizza poi `members` per inizializzare un nuovo oggetto `Team` da passare al costruttore di `Blaminatr`.

La funzione `setup()` inizializza la porta seriale e collega il servomotore di `Blaminatr` al pin definito in `MOTOR_PIN`. La stessa funzione inizializza anche `Blaminatr` con il valore "*nobody*".

La funzione `loop()` è molto simile a quella introdotta nel progetto precedente di questo capitolo; l'unica differenza è che in questo caso non si controlla direttamente il servomotore ma si chiama la funzione `blame()`, come si può vedere alla riga 28.

Ecco fatto! Ora potete iniziare a disegnare il display e costruire la freccia di puntamento. Collegate display e freccia direttamente al motore o meglio ancora disponete il tutto in una scatola. Compilate e caricate il programma per iniziare a dare la colpa a qualcuno.

Ovviamente potete utilizzare i motori elettrici in progetti molto più seri. Potete per esempio impiegarli per costruire robot che si muovono su ruote o altre apparecchiature simili a questa. Ricordate però che non potete collegare un numero eccessivo di motori a una scheda Arduino, perché questa non è stata ideata per sopportare un carico eccessivo. Se state pensando a un progetto che richiede la presenza di più motori valutate la possibilità di acquistare una scheda Motor shield (ne potete trovare collegandovi al sito <http://adafruit.com> oppure <http://makershed.com>); in alternativa potete utilizzare come scheda di controllo un modello Roboduino (<http://store.curiousinventor.com/roboduino.html>).

Cosa fare se non funziona?

Lavorare con i motori è decisamente semplice ma c'è sempre qualcosa che può andare storto. Il problema principale è legato al fatto che i motori consumano molta energia elettrica e per questo motivo non possono essere collegati direttamente a una scheda Arduino. Inoltre è difficile far funzionare più di un motore alla volta, in particolare quando si ha a disposizione la piccola quantità di energia elettrica che potete ottenere da una porta USB. Se il motore non gira come desiderato conviene controllare

le sue specifiche tecniche e se necessario collegare la scheda Arduino a un alimentatore in CA oppure in CC.

Altri progetti con i motori

I motori sono affascinanti. Cercate su Internet altri progetti che usano una scheda Arduino per controllare il funzionamento di un motore. Uno dei più interessanti è Arduino Hypnodisk (<http://www.flickr.com/photos/kevino/4583084700/in/pool-make>), un dispositivo che sfrutta un servomotore per far ruotare un disco ipnotico, ovvero un disco su cui è disegnata una spirale che produce un effetto ipnotico. Un telemetro a infrarossi modifica la velocità del motore: è sufficiente avvicinarsi al disco per far aumentare la velocità di rotazione del motore.

Un progetto utile e stuzzicante è la clessidra USB (<http://home.comcast.net/~hourglass/>), che usa un dispositivo basato su scheda Arduino e servomotore per girare una clessidra. La caduta della sabbia nella clessidra è rilevata da un sensore ottico; ogni volta che la sabbia cade completamente nella parte inferiore della clessidra il dispositivo la gira automaticamente.

Lo scopo principale del dispositivo ha a che fare con la generazione di numeri casuali: la discesa dei granelli di sabbia è un esempio ideale di generazione della casualità a partire dai segnali rilevati dal sensore ottico (si veda il Capitolo 3); i numeri casuali sono inviati su porta seriale.

Non dovete collegare un peso eccessivo al motore. Se lo spostamento di una freccia in carta non costituisce un problema, potreste riscontrare dei malfunzionamenti quando provate a collegare il motore a un peso molto più consistente. Dovete anche fare attenzione a non collocare ostacoli che possano bloccare la rotazione del motore; l'albero del motore deve sempre ruotare liberamente.

Alcuni motori devono essere tarati di tanto in tanto e in genere questa operazione richiede l'impiego di un piccolo cacciavite. Consultate le specifiche del motore per saperne di più.

Esercizi

- Aggiungete a Blaminatr una scheda Ethernet shield, in modo da poter dare la colpa alle persone via Internet e non solo attraverso la porta seriale. Fate in modo, per esempio, di puntare il browser web all'indirizzo <http://192.168.1.42/blame/Maik> per dare la colpa all'autore di questo libro.
- Realizzate un termometro basato su un sensore di temperatura TMP36 e su un servomotore. Il display del dispositivo potrebbe essere simile a quello mostrato nella Figura 10.6; in sintesi, il progetto deve muovere una freccia in modo che indichi la temperatura misurata dal sensore.
- Utilizzate un ricevitore a infrarossi per controllare il funzionamento di Blaminatr. Potete per esempio sfruttare il tasto dei canali del telecomando di un televisore per far muovere Blaminatr da un nome a quello

successivo.

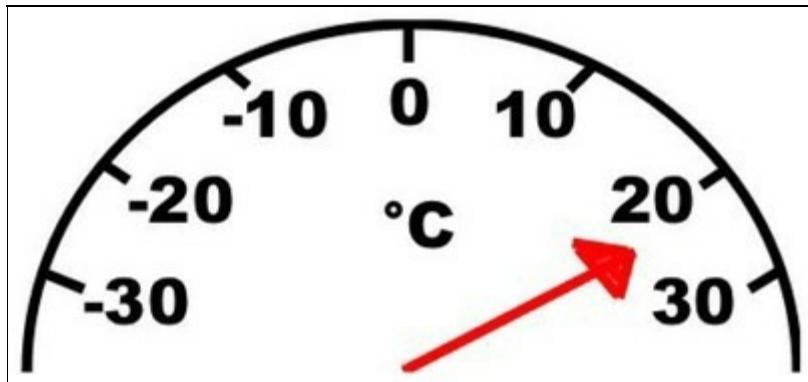


Figura 10.6 Un termometro a motore.

Parte III

Appendici

Appendice A Elettronica di base

Appendice B Programmazione avanzata di Arduino

Appendice C Programmazione seriale avanzata

Appendice D Bibliografia

Elettronica di base

La realizzazione dei primi progetti Arduino non richiede la conoscenza approfondita dell'elettronica, ma conviene avere familiarità con i concetti fondamentali di elettrotecnica e di saldatura dei componenti elettronici per affrontare progetti più sofisticati.

In questa appendice saranno illustrati gli elementi di base dei circuiti elettrici, a partire dalla Legge di Ohm, che costituisce probabilmente la relazione fondamentale tra le grandezze fisiche sia in elettrotecnica sia in elettronica. Farete anche la conoscenza dei resistori e scoprirete che la saldatura dei componenti elettronici non è un'operazione così difficile come sembra.

Corrente, tensione e resistenza

I primi progetti Arduino hanno ben poco a che fare con lo studio dei circuiti elettrici. A un certo punto però diventa necessario capire il significato delle grandezze fisiche di corrente, tensione e resistenza. Per esempio, sapete già che l'impiego di un led richiede sempre l'inserimento di un resistore di limitazione della corrente; è lecito pertanto chiedersi il motivo della presenza del resistore e, perché no, provare a capire come si può calcolare il valore di resistenza che questo componente deve avere per accendere il led nel modo opportuno. È venuto il momento di affrontare la questione.

Un circuito elettrico è analogo per molti versi a un circuito idraulico. Nella Figura A.1 si può osservare a sinistra un circuito idraulico e a destra l'equivalente circuito elettrico. È veramente interessante l'analogia tra i due circuiti: esiste una stretta relazione tra le due situazioni fisiche, se si considera per esempio una dinamo messa in funzione dalla corrente idraulica e impiegata come generatore della tensione elettrica. Vale la pena studiare più da vicino i singoli componenti presenti nei due circuiti.

L'acqua scorre nei tubi idraulici, mentre gli elettroni scorrono nei cavi elettrici. La tensione elettrica è misurata in Volt (V) ed è analoga alla pressione esercitata dalla pompa idraulica. La tensione elettrica provoca il

passaggio di corrente; maggiore è la tensione, maggiore è la velocità con la quale gli elettroni scorrono nel circuito.

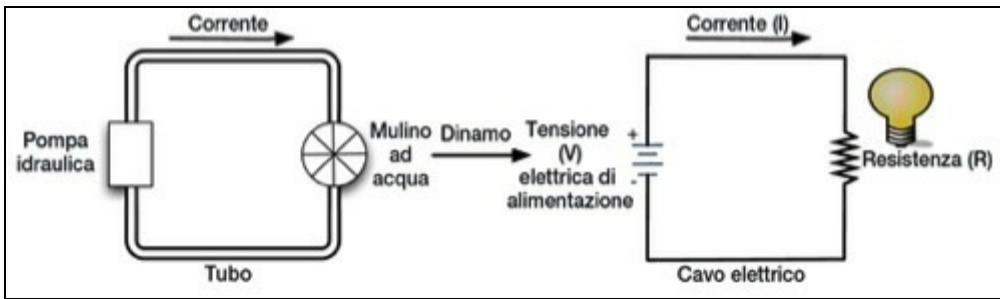


Figura A.1 Analogia tra circuiti idraulici e circuiti elettrici.

Nei circuiti elettronici la corrente fornisce una misura dell’energia elettrica che sta scorrendo in una linea elettrica e può essere ritenuta equivalente alla quantità di acqua che scorre in un circuito idraulico. Il flusso di acqua è misurato in litri al minuto, mentre la corrente si misura in quantità di carica elettrica al minuto, ovvero in Ampere (A). La corrente di 1A significa che nel cavo elettrico scorrono circa $6,24 \times 10^{18}$ elettroni al secondo.

Ogni componente di un circuito elettrico o idraulico oppone una certa resistenza al passaggio della corrente. In un circuito idraulico i componenti principali sono costituiti dai tubi nei quali scorre l’acqua e da utilizzatori di vario genere, per esempio un mulino le cui pale sono messe in movimento dal flusso dell’acqua. In un circuito elettrico i componenti principali sono i fili di collegamento e utilizzatori quali una lampadina a incandescenza. La resistenza è una grandezza fisica strettamente legata alla corrente e alla tensione presenti in un circuito elettrico. La resistenza si misura in Ohm (Ω). Nell’impiego dei led è necessario per esempio dimensionare correttamente il valore della resistenza di limitazione da inserire nel circuito di alimentazione.

La relazione tra corrente I, tensione V e resistenza R è una legge sperimentale introdotta dal fisico tedesco Georg Ohm ed è oggi conosciuta come Legge di Ohm. Il simbolo I per indicare la corrente risale storicamente a quando l’osservazione dei fenomeni elettrici faceva ancora riferimento alla *Intensità* di corrente:

- I (corrente) = V (tensione) / R (resistenza)

L'espressione fondamentale della Legge di Ohm è equivalente alle due relazioni che seguono.

- R (resistenza) = V (tensione) / I (corrente)
- V (tensione) = R (resistenza) \times I (corrente)

È noto dall'algebra che dati due valori si può sempre calcolare il valore della terza grandezza. Da questo punto di vista la Legge di Ohm è in assoluto l'unica relazione imprescindibile nello studio dell'elettrotecnica e dell'elettronica.

In genere il data sheet di un led fornisce due valori: la tensione diretta di alimentazione e il valore massimo di corrente. La tensione diretta è compresa tra 1,8V e 3,6V, mentre la corrente massima è di 20mA (milliampere). Si supponga per esempio di avere a disposizione un led la cui tensione diretta vale 2,5V e la corrente massima è di 20mA; si consideri inoltre una tensione di alimentazione del circuito pari a 5V, un valore che corrisponde alla tensione di alimentazione delle schede Arduino. Che resistenza deve avere il resistore da collegare in serie al led per garantire il corretto funzionamento del circuito?

In questo caso si deve fare in modo che ai capi del resistore sia presente una tensione di $5 - 2,5 = 2,5$ V, così che ai capi del led ci sia una tensione di 2,5V. La tensione sul resistore prende il nome di *caduta di tensione*. La corrente massima di 20mA (che corrisponde a 0,02A) deve scorrere attraverso il led e attraverso il resistore.

Dopo aver stabilito che il resistore deve funzionare con una tensione di 2,5V e una corrente di 0,02A è possibile calcolare come segue il valore di resistenza R del componente:

$$R = V / I$$

In questo caso si ricava il valore indicato di seguito:

$$R = 2,5V / 0,02A = 125\Omega$$

Il circuito richiede in definitiva l'inserimento di un resistore da 125Ω collegato in serie al led. Se non disponete di un resistore di questo tipo potete sempre utilizzarne uno caratterizzato da una resistenza maggiore, per esempio 150Ω o 220Ω . Un resistore di questo valore svolge comunque il ruolo di protezione del led anche se ne diminuisce la luminosità, dato che la corrente è inferiore a quella prevista nel calcolo precedente:

$$I = 2,5V / 150\Omega = 17mA$$

$$I = 2,5V / 220\Omega = 11mA$$

Resistori

È molto difficile trovare un progetto elettronico che non richieda l'uso di resistori. Dovete pertanto conoscere questo genere di componente e sapere, per esempio che in genere sono costruiti utilizzando un impasto di carbone oppure in metallo. I resistori in metallo sono più precisi e non introducono una quantità eccessiva di rumore elettrico, ma i resistori a carbone sono molto più economici. Nei circuiti più semplici potete utilizzare un tipo qualsiasi di resistore. Il parametro fondamentale di un resistore è il suo valore di resistenza, misurata in Ohm (Ω). Esistono pochi resistori che riportano sul corpo del componente il valore di resistenza, dato che questi componenti hanno dimensioni molto ridotte e la lettura di un valore numerico risulterebbe praticamente impossibile. In genere i resistori riportano una serie di strisce che permettono di identificare il valore di resistenza tramite un codice a colori.

Colore	Codice	Zeri da aggiungere
Nero	0	-
Marrone	1	0
Rosso	2	00
Arancione	3	000
Giallo	4	0000
Verde	5	00000
Blu	6	000000
Viola	7	0000000
Grigio	8	00000000
Bianco	9	000000000

Figura A.2 Codice a colori per il riconoscimento del valore di un resistore.

Un resistore è identificato di solito da quattro o cinque strisce colorate, almeno per quanto riguarda quelli a terminali passanti. I resistori SMD (*Surface Mounting Device*) prevedono altre modalità di identificazione ma sono componenti particolari che non vengono impiegati per realizzare progetti a livello amatoriale. Una delle strisce colorate è staccata dalle altre (Figura A.3) e si trova all'estremità destra del codice a colori; indica la tolleranza della resistenza rispetto al valore nominale. Una striscia in colore oro corrisponde a una tolleranza del $\pm 5\%$; in argento si ha una tolleranza del $\pm 10\%$; se la striscia non è presente il resistore ha una tolleranza del $\pm 20\%$. Le altre strisce colorate permettono di ricavare il valore (nominale) della resistenza offerta dal resistore.

Le strisce colorate devono essere lette da sinistra a destra e ogni striscia corrisponde a un numero, come riportato nella Figura A.2. La striscia più a destra (a prescindere dal fatto che si tratti della terza o della quarta) indica il numero di zeri da aggiungere alle cifre ricavate dalle altre strisce.

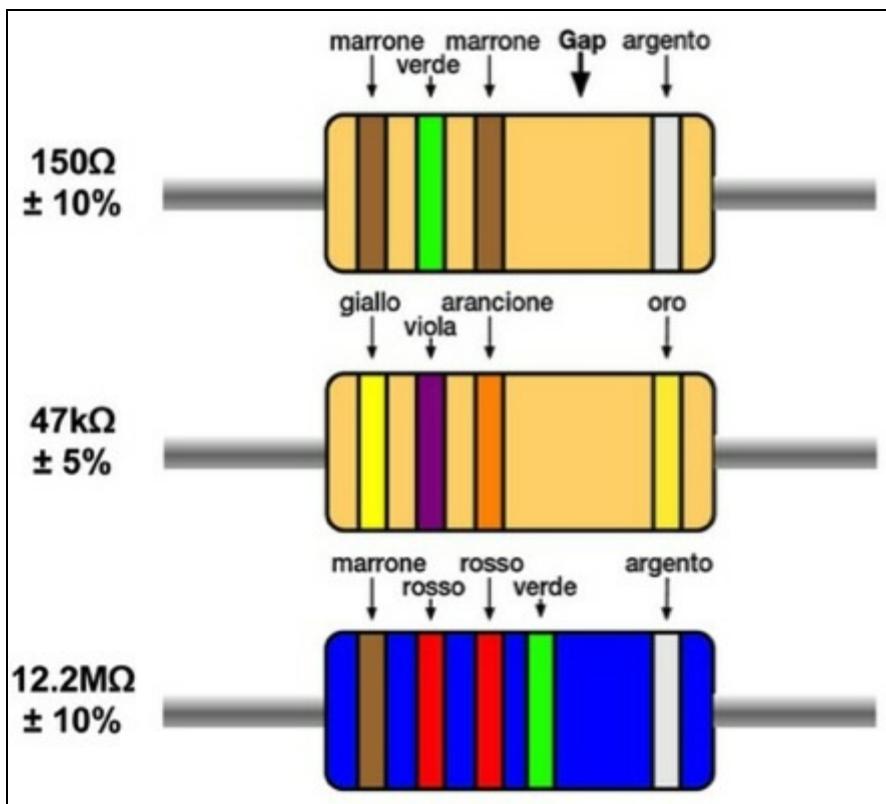


Figura A.3 Le strisce colorate indicano il valore dei resistori.

Nella Figura A.3 sono visibili tre esempi di resistori.

- Il primo resistore presenta quattro strisce colorate, nell'ordine marrone (1), verde (5), marrone (1 zero) e argento ($\pm 10\%$). Il valore di resistenza è 150Ω con una tolleranza del $\pm 10\%$.
- Anche il secondo resistore riporta quattro strisce, in questo caso giallo (4), viola (7), arancione (3 zeri) e oro ($\pm 5\%$). Il valore di resistenza è $47000 = 47k\Omega$ con una tolleranza del $\pm 5\%$.
- Sul terzo resistore sono disegnate cinque strisce colorate: marrone (1), rosso (2), rosso (2), verde (5 zeri) e argento ($\pm 10\%$); il suo valore di resistenza è $12.200.000 = 12.2M\Omega$ con una tolleranza del $\pm 10\%$.

All'inizio il codice a colori può sembrare complicato, ma ci si accorge presto che la lettura del valore di resistenza è un'operazione abbastanza rapida. Anche in Internet si trovano molti strumenti che facilitano l'identificazione dei resistori e di altri componenti elettronici di uso comune (<http://harkopen.com/tutorials/using-wolfram-alpha-electric-circuits>).

Si conclude così la teoria dei circuiti elettronici che è necessario conoscere per realizzare i progetti Arduino illustrati in questo libro. Per saperne di più conviene studiare un testo di elettrotecnica o di elettronica, oppure

consultare un sito Internet dedicato a questo argomento, per esempio

<http://1camtuf.coredump.cx/electronics/>.

La saldatura dei componenti elettronici

I progetti di questo libro possono essere quasi tutti realizzati eseguendo il cablaggio dei componenti hardware su una breadboard o direttamente sulla scheda Arduino. Presto o tardi dovete però affrontare il problema della saldatura di componenti elettronici se volete diventare esperti progettisti delle schede di controllo, in primo luogo perché la realizzazione pratica di progetti destinati a durare nel tempo prevede la saldatura di qualche componente elettronico.

Chi ritiene che la saldatura sia un'operazione complessa o che richieda apparecchiature costose è portato a evitare perfino di provare a saldare un semplice resistore. In realtà la saldatura è economica e abbastanza semplice; richiede una certa pratica, ma è sufficiente provare a saldare pochi componenti per comprendere che non si tratta di ingegneria aerospaziale!

In questo libro è illustrato un progetto che richiede di saldare un connettore standard a 6 pin per collegare il circuito stampato di un sensore ADXL335, componente necessario per realizzare il game controller presentato nel Capitolo 6. Nei prossimi paragrafi vedrete come effettuare la saldatura del connettore su circuito stampato utilizzando l'apparecchiatura elencata di seguito (Figura A.4).

- Un saldatore da 25-30W (W sta per Watt) con una punta sottile (preferibilmente da 1/16") e un portasaldatore.
- Una bobina di stagno 60/40 (con pasta anti-ossidante) per circuiti elettronici. In genere il filo di stagno ha un diametro di circa 0.031".
- Una spugna imbevuta di acqua per pulire la punta calda del saldatore.



Figura A.4 Strumenti per la saldatura di componenti elettronici.

Prima di iniziare a saldare dovete predisporre l'apparecchiatura necessaria. Collocate su un tavolo i vari elementi e fate in modo di potervi accedere comodamente evitando che gocce di stagno fuso possano sporcare o danneggiare il tavolo di lavoro. Indossate sempre occhiali protettivi! Ricordate che perfino l'operazione più banale, per esempio il taglio dei terminali dei componenti elettronici, può provocare infortuni anche gravi.

Posizionate correttamente i componenti elettronici da saldare: inserite il connettore standard nel circuito stampato e controllate che i due elementi rimangano ben fissati tra loro durante le operazioni di saldatura.

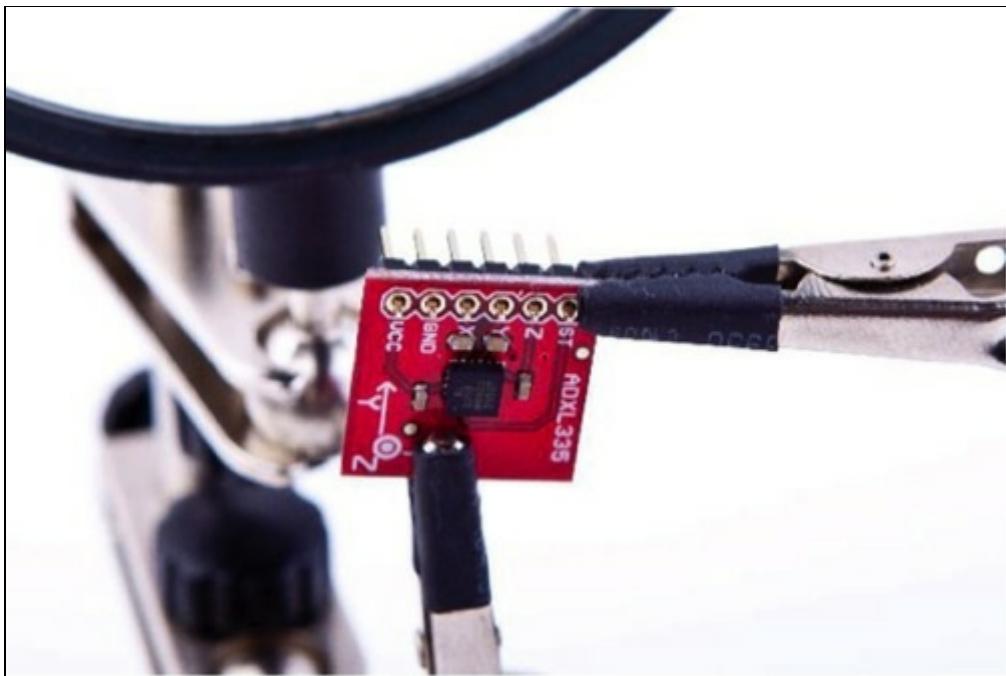


Figura A.5 Dovete collegare il piccolo circuito stampato al connettore standard.

Le persone diventano molto creative quando devono inventarsi un modo per bloccare meccanicamente i componenti elettronici da saldare, ma dovete sempre fare molta attenzione. Per esempio, non utilizzate materiale infiammabile per mantenere il contatto tra connettore e circuito stampato, né impiegate oggetti conduttori di calore, in particolare se questi toccano direttamente i punti di saldatura. In alcuni casi potete ricorrere a del nastro isolante ma anche questo va usato con molta attenzione.

Procuratevi un pezzo di legno o di altro materiale isolante che abbia uno spessore uguale a quello del connettore standard. Appoggiate il circuito stampato sopra al pezzo di legno e collegate il connettore in modo che l'intera struttura rimanga ben fissata sul tavolo di lavoro.

Se avete intenzione di eseguire molte saldature e di realizzare molti progetti elettronici dovete procurarvi alcuni strumenti che vi possano aiutare nel predisporre più facilmente sul tavolo di lavoro i componenti da saldare.

Nella Figura A.6 i componenti sono mantenuti in posizione tramite un apposito supporto per saldatura, utile per bloccare meccanicamente gli oggetti che devono essere saldati tra loro. In genere il supporto dispone anche di una lente di ingrandimento e non è un'apparecchiatura molto costosa. Se pensate di dover effettuare molte saldature valutate la possibilità di acquistarne uno simile.



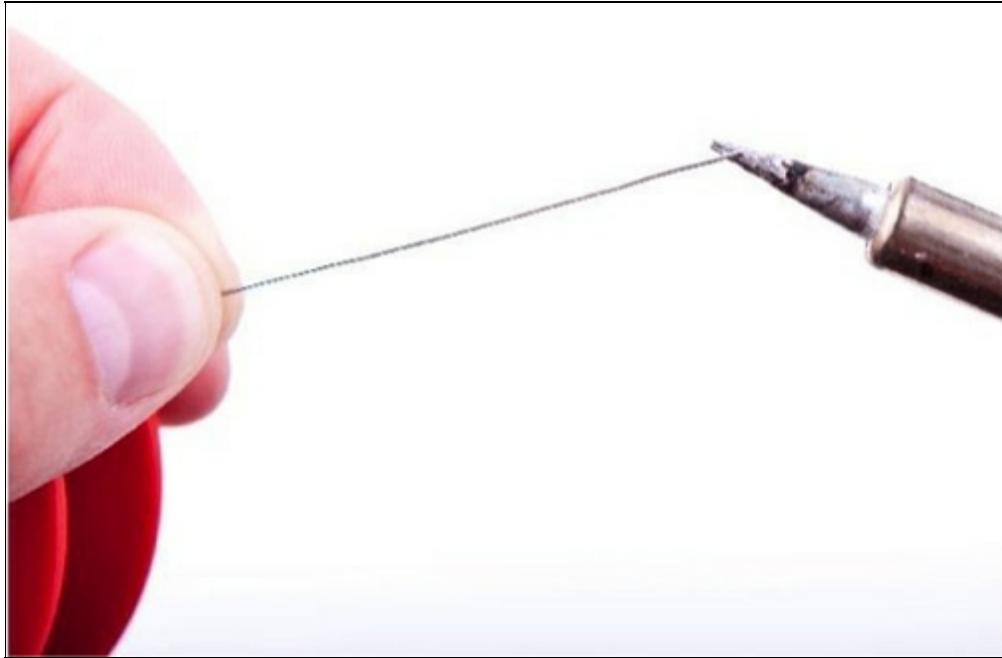
Figura A.6 Un supporto per saldatura a stagno può risultare molto utile.

A questo punto potete mettere in funzione il saldatore fino a scaldare la punta. L'obiettivo principale della saldatura è la connessione meccanica ed elettrica tra due superfici metalliche; in questo caso, dovete unire la superficie di un pin del connettore con il metallo della pista del circuito stampato. L'operazione manuale richiede quindi di scaldare le due parti metalliche e di “saldarle” interponendo tra loro una certa quantità di stagno fuso.

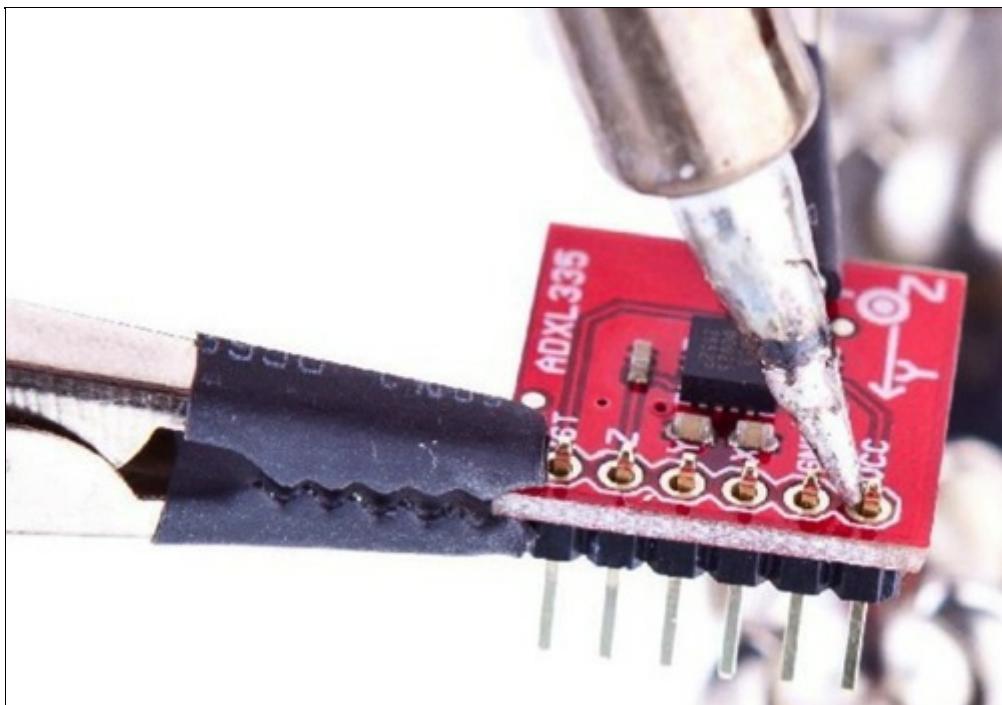
La qualità della saldatura dipende dalla temperatura di riscaldamento delle parti metalliche e di fusione dello stagno; in genere la temperatura raggiunta dal saldatore è uno dei problemi più comuni da affrontare. Se la temperatura è troppo bassa i punti di saldatura sono troppo fragili, e se il saldatore rimane collegato alle parti da saldare troppo a lungo le può danneggiare irreparabilmente. Una temperatura eccessiva del saldatore può danneggiare i componenti anche dopo un contatto di breve durata. Si può discutere a lungo su quale sia la temperatura “giusta” per eseguire una saldatura elettronica: in genere si suggerisce di portare la punta del saldatore a un calore compreso tra 315 e 350 °C.

Bagnate la spugna (senza esagerare) e pulite la punta del saldatore passandola alcune volte sulla spugna bagnata, poi “stagnate” la punta facendo scivolare su di essa una certa quantità di stagno fuso. In questo

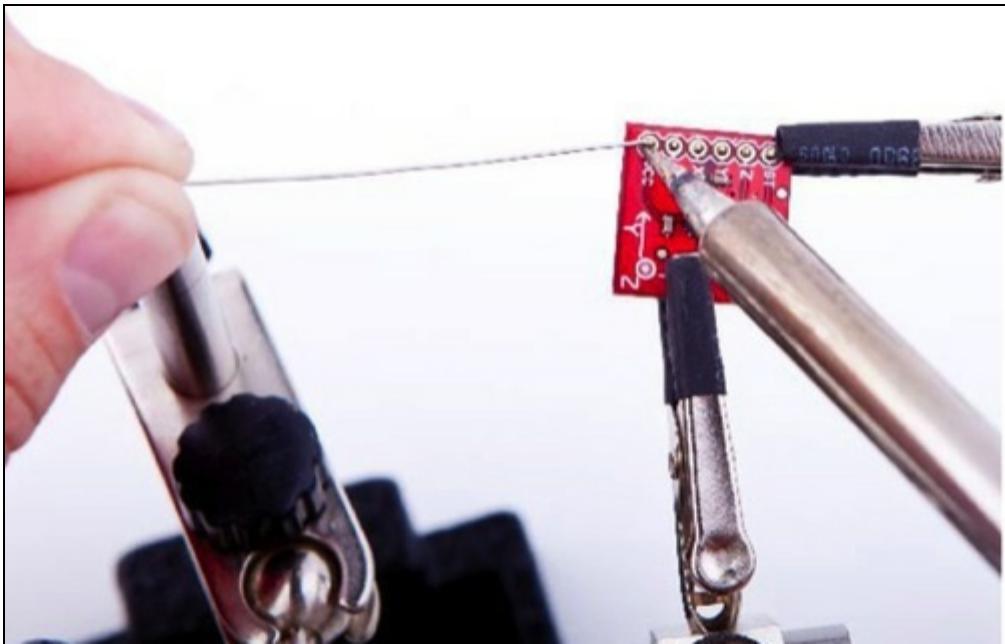
modo la proteggete dal calore e favorite il trasferimento di questo sui componenti da saldare.



La saldatura è in sostanza un trasferimento di calore e a questo punto dovete scaldare il punto di saldatura. Fate in modo che la punta del saldatore tocchi contemporaneamente il pin del connettore e la piazzola sul circuito stampato.



Mantenete il contatto per un secondo circa, poi aggiungete una piccola quantità di stagno fuso tra punta e pin del connettore.



Lo stagno fuso inizia a fluire immediatamente e distribuisce il calore tra le parti in modo naturale. Aggiungete un po' di stagno (non troppo!) fino a ottenere un punto di saldatura uniforme e brillante. L'intera operazione non deve richiedere più di due o tre secondi. Al termine della saldatura allontanate la punta del saldatore e aspettate qualche secondo per lasciar raffreddare lo stagno e i componenti saldati.

Ripetete la medesima operazione per gli altri pin del connettore in modo da ottenere un risultato simile a quello mostrato di seguito.



Verificate la bontà delle saldature costruendo il game controller e giocate con un videogame per rilassarvi un po'.

Congratulazioni! Avete appena concluso il vostro primo lavoro di saldatura elettronica!

Queste istruzioni sono un punto di partenza che vi permette di affrontare le prime saldature più semplici. Ora sapete però che la saldatura non è un'operazione particolarmente complessa: potete affrontare la realizzazione dei kit per principianti, disponibili in qualsiasi negozio di componenti elettronici e corredati di istruzioni dettagliate per la saldatura. Inoltre potete consultare altre guide e perfino video disponibili su Internet che spiegano come saldare i componenti elettronici
(http://store.curiousinventor.com/guides/How_to_Solder).

Programmazione avanzata di Arduino

Il linguaggio di programmazione delle schede Arduino non è altro che il linguaggio C++ con alcune limitazioni e utilizza una suite di strumenti di programmazione molto particolare. In questa appendice saranno illustrate tali limitazioni e studierete il significato delle operazioni sui bit, impiegate diffusamente nei progetti che includono l'uso di sensori e di altri dispositivi hardware.

Il linguaggio di programmazione di Arduino

I primi programmi che realizzerete per Arduino possono sembrare scritti in un linguaggio Arduino “speciale”, anche se in effetti si utilizzano di solito semplici istruzioni in C/C++. Il codice sorgente deve poi essere convertito in un codice macchina (operazione di *cross-compiling*) adatto per essere eseguito dal microcontrollore della scheda di controllo Arduino. I microcontrollori Arduino appartengono alla famiglia AVR prodotta dall’azienda Atmel, che ha messo a punto un gruppo di strumenti basati su compilatori GNU allo scopo di semplificare lo sviluppo software dei microcontrollori AVR. In altre parole, gli strumenti di sviluppo funzionano come quelli originali ma sono stati ottimizzati per generare il codice macchina dei microcontrollori AVR.

Quasi tutti gli strumenti di sviluppo GNU, per esempio `gcc`, `ld` o `as`, prevedono una corrispondente variante AVR, ovvero `avr-gcc`, `avr-ld` e così via, che potete trovare nella directory *hardware/tools/bin* dell’IDE Arduino.

L’IDE è in pratica un ambiente grafico che evita di utilizzare direttamente gli strumenti da riga di comando. Non dovete far altro che delegare le operazioni agli strumenti AVR ogni volta che dovete compilare o caricare un programma tramite l’IDE. Analogamente ai lavori di sviluppo più professionali anche voi dovete riportare in output il maggior numero di informazioni possibile, in modo da esaminare tutte le chiamate effettuate dagli strumenti della riga di comando. Modificate il file `preferences.txt` in base alle indicazioni fornite nel Capitolo 2 e impostate le operazioni `build.verbose` e `upload.verbose` con il valore `true`. Caricate e compilate per

esempio il programma che fa lampeggiare un led. Dovreste ottenere un messaggio di output simile a quello mostrato nella Figura 2.3 nel Capitolo 2.

Le chiamate dei comandi possono sembrare piuttosto misteriose all'inizio, data la presenza di un certo numero di file temporanei. Dovreste comunque essere in grado di identificare tutte le fasi di compilazione e di link che si richiedono per realizzare anche un programma semplice come quello che fa accendere un led. Questo è l'obiettivo principale che è stato raggiunto dal team di Arduino: sono riusciti a nascondere tutti i dettagli tecnici svolti dall'IDE e hanno consentito così di programmare le schede Arduino anche a persone che non hanno esperienza diretta nel campo della programmazione informatica. D'altra parte, i programmatore esperti preferiscono lavorare in modalità testo, dato che in questo modo hanno la possibilità di studiare in dettaglio il funzionamento di tutti gli strumenti AVR.

Caricate il programma in Arduino ed esamineate il funzionamento di avrdude. Questo strumento si occupa del caricamento del codice in Arduino e può essere utilizzato per programmare anche altri tipi di dispositivi. È interessante osservare che gli strumenti AVR possono essere impiegati per sfruttare l'IDE Arduino nell'implementazione di software relativo a progetti non Arduino, per esempio il progetto Meggy Jr. (<http://www.evilmadscientist.com/article.php/meggyjr>).

Qualcuno potrebbe dire: “Attenzione, io sono un programmatore in C/C++ e non vedo la funzione `main()`!”. Questa è un'altra differenza tra la programmazione Arduino e il codice tipico del linguaggio C++. La programmazione della schede Arduino non richiede la definizione della funzione `main()` perché questa è già definita dalle librerie messe a disposizione dagli sviluppatori del team di Arduino. Come è lecito aspettarsi, questa funzione chiama `setup()` ed esegue un ciclo definito da `loop()`.

La programmazione in C++ dei microcontrollori AVR prevede ulteriori limitazioni, illustrate di seguito. Per saperne di più consultate l'indirizzo (http://www.nongnu.org/avr-libc/user-manual/FAQ.html#faq_cplusplus).

- Non potete utilizzare la libreria STL (*Standard Template Library*) perché

- le sue dimensioni sono eccessive per i piccoli microcontrollori AVR.
- Non è supportata la gestione delle eccezioni e per questo motivo si attiva spesso la funzione `--fnoexceptions` quando si richiama il compilatore `avr-gcc`.
- Non è supportata la gestione dinamica della memoria tramite funzioni `new()` e `delete()`.

Oltre a queste considerazioni dovete sempre tenere presente le prestazioni del sistema. Il linguaggio C++ genera per esempio in modo automatico un gran numero di funzioni, quali i costruttori di copia, gli operatori di assegnamento e così via. Queste funzioni sono utilizzate raramente dalle schede di controllo Arduino, ma nonostante queste restrizioni Arduino è in grado di supportare un insieme consistente di istruzioni del linguaggio di programmazione C++. In conclusione non avete scuse, iniziate a scrivere il vostro primo programma!

Operazioni sui bit

L'elaborazione dei dati rilevati da componenti elettronici integrati richiede spesso di avere a che fare con i bit digitali. A volte dovete per esempio leggere i singoli bit riportati dalle misurazioni di un determinato sensore, altre volte dovete impostare i bit in modo da configurare lo stato di funzionamento di un dispositivo oppure per eseguire una certa azione.

L'elaborazione dei bit necessita di impostare alcune operazioni elementari, la più semplice delle quali è *not*, che commuta lo stato del bit da 0 a 1 oppure da 1 a 0. La maggior parte dei linguaggi di programmazione implementa l'operazione *not* utilizzando il simbolo `!`:

```
int x = 42; // In binario corrisponde a x = 101010
int y = !x; // y == 010101
```

Oltre a questa dovete imparare il significato delle tre operazioni binarie chiamate *AND*, *OR* e *XOR* (*eXclusive OR*). La maggior parte dei linguaggi di programmazione definisce gli operatori corrispondenti con i simboli `&`, `|` e `^`, mentre lo schema che segue riporta il risultato dell'operazione effettuata su una coppia di bit *a* e *b*.

a	b	a AND b	a OR b	a XOR b
		a & b	a b	a ^ b
0	0	0	0	0
1	0	0	1	1
0	1	0	1	1
1	1	1	1	0

Questi operatori permettono di effettuare la mascheratura dei bit di un numero binario. Potete per esempio estrarre una serie di bit da una stringa più lunga di bit. Si supponga per esempio di voler estrarre i due bit meno significativi di un numero binario. L'operazione da eseguire è questa:

```
int x = 42; // In binario corrisponde a x = 101010
int y = x & 0x03; // y == 2 == B10
```

L'operazione *OR* permette invece di impostare o azzerare uno o più bit di un numero binario. Di seguito sono riportate le istruzioni che impostano a *x* il bit di ordine 5 del valore iniziale, a prescindere dal fatto che questo bit valesse inizialmente 0 oppure 1:

```
int x = 42; // In binario corrisponde a x = 101010
int y = x | 0x10; // y == 58 == B111010
```

Gli operatori *<<* e *>>* di scorrimento (*shift*) dei bit permettono di traslare i bit in una determinata posizione prima di elaborarne il valore. Il primo operatore fa scorrere i bit verso sinistra, il secondo operatore trasla i bit verso destra:

```
int x = 42; // In binario corrisponde a x = 101010
int y = x << 1; // y == 84 == B1010100
int z = x >> 2; // z == 10 == B1010
```

L'operazione di scorrimento è abbastanza intuitiva ma dovete prestare attenzione quando la applicate a un valore numerico con segno (http://en.wikipedia.org/wiki/Arithmetic_shift). L'operazione sui bit è molto simile ma il significato del risultato va interpretato tenendo presente che gli operatori binari non sono uguali agli operatori booleani, per

esempio `&&` e `||`, che non eseguono calcoli a livello di bit ma implementano le regole dell'algebra booleana

(http://en.wikipedia.org/wiki/Boolean_algebra_%28logic%29 o [http://it.wikipedia.org/wiki/Booleano_\(informatica\)](http://it.wikipedia.org/wiki/Booleano_(informatica))).

Programmazione seriale avanzata

Quasi tutti i progetti Arduino di questo libro si basano sull'utilizzo della porta seriale per la comunicazione tra computer e scheda di controllo. A volte la comunicazione riguarda semplicemente l'emissione di messaggi di debug per monitorare lo stato corrente dei progetti, ma il più delle volte la comunicazione seriale serve a trasmettere informazioni di output o comandi alla scheda. In questi progetti si è fatto riferimento alla classe `Serial` senza spiegare effettivamente il funzionamento della comunicazione seriale, che sarà illustrato in questa appendice.

Le comunicazioni con Arduino prevedono l'utilizzo del linguaggio di programmazione Processing e nei progetti di questo libro sono impiegate istruzioni scritte in JavaScript. Dato che molti programmati preferiscono altri linguaggi in questa appendice si vedrà come utilizzare C/C++, Java, Ruby, Python e Perl per comunicare con Arduino.

Saperne di più sulle comunicazioni seriali

Nel Capitolo 2 avete appreso che le trasmissioni seriali richiedono solo tre fili: una massa comune, una linea (TX) per la trasmissione di dati e una linea (RX) per la ricezione di dati, come si può vedere nella Figura C.1.

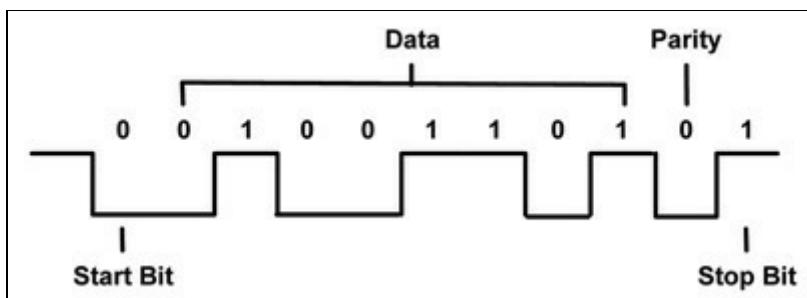


Figura C.1 Comunicazione seriale a livello di bit.

I dati sono trasmessi da impulsi elettrici e i dispositivi che partecipano alla comunicazione (trasmettitore e ricevitore) devono avere un livello di riferimento comune per la tensione elettrica dei segnali, stabilito dal collegamento di massa. La linea di trasmissione (del trasmettitore) è impiegata per inviare segnali al ricevitore e deve essere collegata alla linea

di ricezione (del ricevitore). In questo modo si dice che la trasmissione avviene in modalità *full-duplex*, ovvero entrambi i dispositivi possono simultaneamente trasmettere e ricevere dati.

Ora sapete come impostare il collegamento hardware tra i due dispositivi coinvolti in una trasmissione seriale, ma dovete ancora comprendere come vengono inviati i segnali elettrici corrispondenti. In sintesi, i due dispositivi devono stabilire un protocollo di trasmissione. Nella Figura C.1 potete vedere i segnali coinvolti nella trasmissione tipica di una serie di bit. Gli stati differenti di un bit sono rappresentati da due livelli di tensione ben distinti tra loro. In genere un bit a 0 è rappresentato da una tensione di 0V, mentre la tensione di 5V corrisponde a un bit che vale 1; altri protocolli di trasmissione utilizzano invece i livelli di tensione -12V e 12V.

Di seguito sono indicati i parametri che controllano una trasmissione seriale.

- Il bit di start o *start bit* indica l'inizio di una parola di dati ed è utilizzato per sincronizzare il funzionamento di trasmettitore e ricevitore. Corrisponde sempre a un bit 0.
- Il bit di stop o *stop bit* segnala l'invio dell'ultimo bit e separa la trasmissione consecutiva di due parole di dati. Il protocollo può prevedere la presenza di più stop bit, ma questo avviene di rado nelle comunicazioni seriali.
- Le informazioni sono trasmesse come *data bit* binari; in altre parole, la trasmissione della lettera *M* richiede di convertire la lettera in un formato numerico. La trasmissione seriale può adottare diversi sistemi di codifica dei segnali; i progetti Arduino suggeriscono in genere di utilizzare il codice ASCII. In questo caso, la lettera *M* maiuscola corrisponde al valore numerico 77, ovvero al numero binario 01001101, che è la sequenza di bit da trasmettere per inviare la lettera.
- Il bit di parità o *parity bit* indica se il numero di bit trasmessi è pari o dispari. Questo bit permette di configurare un semplice test di controllo della trasmissione, che viene implementato raramente nelle trasmissioni odierne e risale ai tempi in cui le connessioni in rete erano meno affidabili di quanto lo siano oggi. Il controllo di parità può essere di tipo "nullo" (non è inviato alcun bit di parità), "dispari" (il bit di parità vale 1 se è dispari il numero di bit che valgono 1 nella parola dati, altrimenti vale 0) oppure "pari" (il bit di parità vale 1 se è pari il numero di bit a 1 della

parola dati, altrimenti vale 0). Nell'esempio della Figura C.1 la parità è dispari e il bit di parità vale 0 perché ci sono 4 bit a 1 nella parola 01001101.

- Il *baud rate* determina la velocità di trasmissione ed è un parametro espresso in bit al secondo. Le schede Arduino ammettono in genere valori di baud rate pari a 9600, 14400, 19200 o perfino 115200. Ricordate che il baud rate non definisce la quantità totale di dati trasmessi ogni secondo, in quanto dovete tenere conto della presenza dei bit di controllo. Si prenda per esempio una trasmissione configurata in modo da avere 1 start bit, 1 stop bit, parità di tipo nullo e 8 bit per byte; in questo caso la trasmissione di un dato completo richiede $1 + 1 + 8 = 10$ bit per trasferire un singolo byte. Se il baud rate è 9600 potete in teoria inviare $9600 / 10 = 960$ byte al secondo, a condizione che ogni bit venga trasmesso utilizzando esattamente un periodo del segnale presente sulla linea di comunicazione seriale.

La comunicazione seriale nei linguaggi di programmazione

In questo libro avete già utilizzato diversi linguaggi di programmazione per accedere alla scheda Arduino collegata alla porta seriale di un computer. Nel Capitolo 6 avete utilizzato Processing, mentre nel Capitolo 9 le istruzioni sono state scritte in JavaScript.

Le schede Arduino richiedono spesso di programmare le porte seriali; in questo paragrafo vedrete come impostare una trasmissione utilizzando diversi linguaggi di programmazione. Per semplificare la trattazione si farà riferimento allo stesso programma Arduino, riportato di seguito.

[SerialProgramming/AnalogReader/AnalogReader.pde](#)

```
const unsigned int BAUD_RATE = 9600;
const unsigned int SERIAL_DELAY = 5;
const unsigned int NUM_PINS = 6;

void setup() {
    Serial.begin(BAUD_RATE);
}

void loop() {
    const int MAX_PIN_NAME = 3;

    char pin_name[MAX_PIN_NAME + 1];
```

```
if (Serial.available()) {
    int i = 0;
    while (Serial.available() && i < MAX_PIN_NAME) {
        const char c = Serial.read();
        if (c != -1 && c != '\n')
            pin_name[i++] = c;
        delay(SERIAL_DELAY);
    }
    pin_name[i] = 0;
    if (strlen(pin_name) > 1 &&
        (pin_name[0] == 'a' || pin_name[0] == 'A'))
    {
        const int pin = atoi(&pin_name[1]);
        if (pin < NUM_PINS) {
            Serial.print(pin_name);
            Serial.print(": ");
            Serial.println(analogRead(pin));
        } else {
            Serial.print("Unknown pin: ");
            Serial.println(pin);
        }
    } else {
        Serial.print("Unknown pin name: ");
        Serial.println(pin_name);
    }
}
```

Questo programma attende il nome di un pin analogico (a0, a1... a5) e restituisce il valore corrente del pin indicato. In questo modo i client del programma devono inviare un dato alla scheda Arduino (il nome del pin) e ricevere il risultato corrispondente, come si può vedere nell'esempio illustrato nella Figura C.2, che fa riferimento all'utilizzo di *Serial Monitor* dell'IDE Arduino.



Figura C.2 Il programma di prova restituisce i valori correnti misurati sui pin analogici.

I client funzionano in modo analogo tra loro: si aspettano che il nome della porta seriale venga indicato come argomento della riga di comando, inviano in modo continuo la stringa “a0” ad Arduino per riportare il valore corrente del pin analogico 0 e visualizzano il risultato sulla console. I client impostano un baud rate costante di 9600 e attendono due secondi prima di aprire la porta seriale, dato che molte schede Arduino si riavviano per aprire una comunicazione seriale. Per saperne di più sulle caratteristiche dei segnali coinvolti nella trasmissione seriale si veda il paragrafo precedente di questa appendice.

Alcuni client richiedono l’installazione di librerie aggiuntive e in alcuni casi dovete accedere al sistema come amministratore del computer. Queste operazioni non verranno descritte esplicitamente. Dovete inoltre verificare che non siano aperte altre finestre di controllo delle comunicazioni seriali quando provate ad eseguire gli esempi illustrati nei prossimi paragrafi.

C/C++

Nonostante la scheda Arduino sia programmata con istruzioni scritte in C++, non dovete configurare direttamente i client che comunicano con Arduino in C++ o C. Potete comunque effettuare questa operazione in modo abbastanza semplice utilizzando come riferimento la classe `arduino_serial.c1` messa a disposizione da Tod E. Kurt (<http://todbot.com/blog/2006/12/06/arduino-serial-c-code-to-talk-to-arduino/>).

Il programma originale implementa uno strumento da riga di comando completo che offre opzioni veramente utili. Molte di queste funzioni vanno ben oltre gli scopi dei progetti Arduino; di seguito potete vedere le quattro funzioni principali della classe `arduino_serial.c1`, riportate in un file header scritto in C.

[SerialProgramming/c/arduino-serial.h](#)

```
#ifndef __ARDUINO_SERIAL__
#define __ARDUINO_SERIAL__

#include <fcntl.h>
#include <sys/ioctl.h>
#include <termios.h>
#include <stdint.h>
#include <string.h>

int serialport_init(const char* serialport, int baud);
int serialport_writebyte(int fd, uint8_t b);
int serialport_write(int fd, const char* str);
int serialport_read_until(int fd, char* buf, char until);

#endif
```

Vediamo il significato di queste istruzioni.

- `serialport_init()` apre una connessione su porta seriale. La funzione si aspetta il nome della porta seriale da aprire e il baud rate da configurare. La funzione restituisce un parametro file descriptor se le operazioni vanno a buon fine, altrimenti restituisce il valore -1.
- `serialport_writebyte()` permette di inviare un singolo byte alla scheda Arduino collegata alla porta seriale del computer. È sufficiente passare come parametri il file descriptor restituito da `serialport_init()` e il byte da trasmettere. La funzione restituisce il valore -1 in caso di errore, altrimenti restituisce 0.
- `serialport_write()` scrive una stringa intera sulla porta seriale. È sufficiente passare il file descriptor e la stringa da scrivere. Restituisce -1 in caso di errore, altrimenti restituisce 0.
- `serialport_read_until()` permette di leggere dati dalla porta seriale. Dovete passare come parametri il file descriptor e un buffer da riempire con i dati letti sulla porta seriale. Il metodo si aspetta anche l'indicazione del carattere di delimitazione dei dati. `serial_port_read_until()` continua

a leggere dati fino a quando incontra il carattere di delimitazione.
Restituisce sempre il valore 0.

Per completezza della trattazione, di seguito sono riportate le implementazioni delle quattro funzioni.

[SerialProgramming/c/arduino-serial.c](#)

```
#include "arduino-serial.h"

int serialport_writebyte(int fd, uint8_t b) {
    int n = write(fd, &b, 1);
    return (n != 1) ? -1 : 0;
}

int serialport_write(int fd, const char* str) {
    int len = strlen(str);
    int n = write(fd, str, len);
    return (n != len) ? -1 : 0;
}

int serialport_read_until(int fd, char* buf, char until) {
    char b[1];
    int i = 0;
    do {
        int n = read(fd, b, 1);
        if (n == -1)
            return -1;
        if (n == 0) {
            usleep(10 * 1000);
            continue;
        }
        buf[i++] = b[0];
    } while (b[0] != until);

    buf[i] = 0;
    return 0;
}

int serialport_init(const char* serialport, int baud) {
    int fd = open(serialport, O_RDWR | O_NOCTTY | O_NDELAY);
    if (fd == -1) {
        perror("init_serialport: Unable to open port");
        return -1;
    }

    struct termios toptions;
```

```

if (tcgetattr(fd, &toptions) < 0) {
    perror("init_serialport: Couldn't get term attributes");
    return -1;
}

speed_t brate = baud;
switch(baud) {
    case 4800:    brate = B4800;    break;
    case 9600:    brate = B9600;    break;
    case 19200:   brate = B19200;   break;
    case 38400:   brate = B38400;   break;
    case 57600:   brate = B57600;   break;
    case 115200:  brate = B115200;  break;
}
cfsetispeed(&toptions, brate);

toptions.c_cflag &= ~PARENB;
toptions.c_cflag &= ~CSTOPB;
toptions.c_cflag &= ~CSIZE;
toptions.c_cflag |= CS8;
toptions.c_cflag &= ~CRTSCTS;
toptions.c_cflag |= CREAD | CLOCAL;
toptions.c_iflag &= ~(IXON | IXOFF | IXANY);
toptions.c_lflag &= ~(ICANON | ECHO | ECHOE | ISIG);
toptions.c_oflag &= ~OPOST;

toptions.c_cc[VMIN] = 0;
toptions.c_cc[VTIME] = 20;

if (tcsetattr(fd, TCSANOW, &toptions) < 0) {
    perror("init_serialport: Couldn't set term attributes");
    return -1;
}

return fd;
}

```

Queste istruzioni dovrebbero essere familiari a chi conosce la gestione dei file in ambiente Unix; in ogni caso avete qui a disposizione il codice che vi permette di accedere alla scheda Arduino collegata alla porta seriale del computer. Di seguito è riportato un programma che imposta la comunicazione seriale richiesta dal programma di lettura dei pin analogici illustrato in precedenza. Ricordate che questo programma è eseguito dal computer, non dalla scheda Arduino.

```

Riga 1 #include <stdio.h>
- #include <unistd.h>
- #include "arduino-serial.h"
-
5 #define MAX_LINE 256
-
-
- int main(int argc, char* argv[]) {
-     if (argc == 1) {
-         printf("You have to pass the name of a serial port.\n");
10     return -1;
- }
-
-
- int baudrate = B9600;
- int arduino = serialport_init(argv[1], baudrate);
15 if (arduino == -1) {
-     printf("Could not open serial port %s.\n", argv[1]);
-     return -1;
- }
- sleep(2);
20
- char line[MAX_LINE];
- while (1) {
-     int rc = serialport_write(arduino, "a0\n");
-     if (rc == -1) {
25     printf("Could not write to serial port.\n");
- } else {
-     serialport_read_until(arduino, line, '\n');
-     printf("%s", line);
- }
30 }
- return 0;
- }

```

In primo luogo si importano le librerie necessarie e si definisce una costante relativa alla lunghezza massima delle righe che devono essere lette sulla scheda Arduino, dopodiché si imposta la funzione `main()`.

Dopo aver verificato che da riga di comando sia stato trasmesso il nome della porta seriale, il programma inizializza la porta seriale (riga 14), poi attende 2 secondi per dare tempo alla scheda Arduino di entrare in funzione. Al termine di questa pausa ha inizio un ciclo di istruzioni che invia costantemente la stringa “a0” alla scheda Arduino (riga 23). Il programma verifica il risultato della funzione `serialport_write()` e se l’operazione ha avuto successo le riga 27 permette di leggere il risultato

inviai dalla scheda Arduino. Ora potete compilare il programma utilizzando il comando che segue:

```
maik> gcc arduino-serial.c analog_reader.c -o analog_reader
```

Identificate la porta seriale del computer ove è collegata la scheda Arduino (nell'esempio è connessa a /dev/tty.usbmodemfa141) ed eseguite il programma impostando un comando simile a questo:

```
maik> ./analog_reader /dev/tty.usbmodemfa141
a0: 495
a0: 376
a0: 368
^C
```

Le operazioni vengono effettuate come previsto, e l'accesso a una porta seriale in C non è poi così difficile. Per inserire questo codice in un programma C++ dovete includere le funzioni in una classe chiamata `SerialPort` o con un nome simile a questo.

Java

La piattaforma Java propone svariate soluzioni standard che includono anche la Java Communications API che definisce l'accesso a una porta seriale (<http://java.sun.com/products/javacomm/>). Ricordate però che l'API è semplicemente una raccolta di specifiche che devono essere implementate nel programma di controllo. Potete studiare una buona implementazione dell'API esaminando il progetto RXTX (<http://rxtx.qbang.org/>).

Scaricate la versione più recente del software e seguite le istruzioni di installazione relative alla vostra piattaforma. Verificate che `RXTXcomm.jar` faccia parte del percorso delle classi, poi scrivete le istruzioni qui riportate utilizzando l'IDE o un editor di testi qualsiasi.

[SerialProgramming/java/AnalogReaderTest.java](#)

```
import java.io.InputStream;
import java.io.OutputStream;
import gnu.io.CommPortIdentifier;
import gnu.io.SerialPort;

class AnalogReader {
    private InputStream _input;
    private OutputStream _output;
```

```

public AnalogReader(
    final String portName,
    final int baudRate) throws Exception
{
    final int timeout = 1000;
    final String appName = "analog reader client";
    CommPortIdentifier portId =
        CommPortIdentifier.getPortIdentifier(portName);
    SerialPort port = (SerialPort)portId.open(
        appName,
        timeout
    );
    _input = port.getInputStream();
    _output = port.getOutputStream();
    port.setSerialPortParams(
        baudRate,
        SerialPort.DATABITS_8,
        SerialPort.STOPBITS_1,
        SerialPort.PARITY_NONE
    );
}

public void run() throws Exception {
    byte[] buffer = new byte[255];
    Thread.sleep(2000);
    while (true) {
        _output.write("a0\n".getBytes());
        Thread.sleep(100);
        if (_input.available() > 0) {
            _input.read(buffer);
            System.out.print(new String(buffer));
        }
    }
}

public class AnalogReaderTest {
    public static void main(String[] args) throws Exception {
        if (args.length != 1) {
            System.out.println(
                "You have to pass the name of a serial port."
            );
            System.exit(1);
        }
        AnalogReader analogReader = new AnalogReader(args[0], 9600);
        analogReader.run();
    }
}

```

Questo file definisce le due classi `AnalogReader` e `AnalogReaderTest`.

`AnalogReader` incapsula l'accesso alla scheda Arduino, memorizza un oggetto `InputStream` in `_input` per ricevere dati dalla scheda uno `OutputStream` in `_output` per inviare dati alla scheda.

Il costruttore inizializza il collegamento con la porta seriale e assegna il passaggio di dati in input e in output alle variabili membro. Il collegamento con la porta seriale richiede in primo luogo di impostare un oggetto

`CommPortIdentifier`, mediante il quale si può creare un oggetto `SerialPort`.

Questo oggetto garantisce l'accesso al flusso di dati e permette di impostare i parametri della porta, per esempio il baud rate.

Il protocollo del programma Arduino è implementato nel metodo `run()`. Il programma attende per due secondi prima di avviare un ciclo di istruzioni che invia la stringa “a0” alla porta seriale utilizzando il metodo `write()` della classe `OutputStream`. Prima di inviare la stringa occorre convertire i dati in un byte array tramite la funzione `getBytes()`. Il programma attende ancora per 100 millisecondi, in modo da lasciare alla scheda Arduino il tempo necessario per produrre il risultato, dopodiché si verifica se questo è valido e lo si legge chiamando il metodo `read()` della classe `InputStream`.

`AnalogReaderTest` è una piccola classe che implementa un metodo `main()` che permette di creare un oggetto `AnalogReader` sul quale si chiama la funzione `run()`. Di seguito sono indicate le istruzioni che permettono di compilare e utilizzare il programma:

```
maik> javac AnalogReaderTest.java
maik> java AnalogReaderTest /dev/tty.usbmodemfa141
Experimental:      JNI_OnLoad called.
Stable Library
=====
Native lib Version = RXTX-2.1-7
Java lib Version   = RXTX-2.1-7
a0: 496
a0: 433
a0: 328
a0: 328
^C
```

Dopo aver effettuato il debug delle librerie impiegate dal software, il programma `AnalogReaderTest` esegue esattamente le operazioni desiderate:

visualizza in modo continuo i valori rilevati sul pin analogico 0. L'accesso alla porta seriale in Java diventa un problema semplice da risolvere se si utilizzano correttamente le librerie software.

Ruby

Anche i linguaggi di programmazione dinamici, per esempio Ruby (<http://it.wikipedia.org/wiki/Ruby>), consentono di accedere direttamente alla porta seriale del computer e a una scheda Arduino collegata alla medesima porta. Prima di eseguire qualsiasi operazione è necessario installare la Ruby gem serialport utilizzando questo comando:

```
maik> gem install serialport
```

Questa libreria Ruby permette di collegarsi alla scheda Arduino tramite le 30 righe di codice qui riportate.

[SerialProgramming/ruby/analog_reader.rb](#)

```
Riga 1  require 'rubygems'  
-   require 'serialport'  
  
-  
-   if ARGV.size != 1  
5     puts "You have to pass the name of a serial port."  
-     exit 1  
-   end  
  
-  
-   port_name = ARGV[0]  
10  baud_rate = 9600  
-   data_bits = 8  
-   stop_bits = 1  
-   parity = SerialPort::NONE  
  
-  
15  arduino = SerialPort.new(  
-     port_name,  
-     baud_rate,  
-     data_bits,  
-     stop_bits,  
20  parity  
- )  
  
-  
-   sleep 2  
-   while true  
25  arduino.write "a0"  
-   line = arduino.gets.chomp  
-   puts line  
- end
```

Il programma crea alla riga 15 un nuovo oggetto `SerialPort`, cui vengono passati i parametri necessari per configurare la porta seriale. Dopo un’attesa di 2 secondi il programma esegue un ciclo di istruzioni e chiama la funzione `write()` sull’oggetto `SerialPort`. Il metodo `gets()` permette di ricavare i risultati elaborati dalla scheda Arduino, dopodiché il programma li visualizza su console. Di seguito è riportato un esempio di utilizzo di queste istruzioni:

```
maik> ruby analog_reader.rb /dev/tty.usbserial-A60061a3
a0: 496
a0: 456
a0: 382
^C analog_reader.rb:21:in 'gets': Interrupt
from analog_reader.rb:21
```

Può essere conveniente affidarsi a Ruby per accedere alla scheda di controllo Arduino, dato che questa soluzione permette di concentrare l’attenzione sull’applicazione che si sta progettando. I dettagli di configurazione della comunicazione seriale che altri linguaggi di programmazione richiedono di gestire direttamente sono in questo caso demandati alle funzioni della libreria Ruby.

Python

Python è un altro linguaggio di programmazione dinamico che permette di realizzare rapidamente client software della scheda di controllo Arduino. La programmazione di una porta seriale richiede di scaricare e installare in primo luogo la libreria pyserial

(<http://sourceforge.net/projects/pyserial/files/>). In ambiente Windows è disponibile un programma di installazione particolare, ma in generale è sufficiente impostare un comando simile a quello indicato di seguito per eseguire l’installazione della libreria:

```
maik> python setup.py install
```

Dopo aver installato pyserial potete utilizzare la libreria per creare il client relativo al programma che legge la porta analogica di Arduino.

[SerialProgramming/python/analog_reader.py](#)

```
Riga 1 import sys
```

```

- import time
- import serial
-
5 if len(sys.argv) != 2:
-     print "You have to pass the name of a serial port."
-     sys.exit(1)
-
-     serial_port = sys.argv[1]
10 arduino = serial.Serial(
-         serial_port,
-         9600,
-         serial.EIGHTBITS,
-         serial.PARITY_NONE,
15         serial.STOPBITS_ONE)
-     time.sleep(2)
-
-     while 1:
-         arduino.write('a0')
20     line = arduino.readline().rstrip()
-         print line

```

Il programma verifica innanzitutto che sulla riga di comando sia stato impostato il nome di una porta seriale, poi crea un nuovo oggetto `serial` alla riga 10, cui passa i parametri necessari per configurare la comunicazione seriale.

Dopo un'attesa di 2 secondi il programma avvia un ciclo di istruzioni che trasmette la stringa “a0” alla porta seriale chiamando la funzione `write()`. I risultati restituiti da Arduino sono letti tramite il metodo `readline()` e vengono riportati in output su console. Di seguito è riportato un esempio di utilizzo del programma Python:

```

maik> python analog_reader.py /dev/tty.usbserial-A60061a3
a0: 497
a0: 458
a0: 383
^C

```

Interessante, vero? Bastano 20 righe di codice Python per avere il pieno controllo della scheda Arduino. Python è un'altra ottima soluzione per scrivere client Arduino.

Perl

Perl è ancora oggi uno dei linguaggi di programmazione dinamici più diffusi e supporta egregiamente le comunicazioni seriali. Alcune distribuzioni includono le librerie di programmazione della porta seriale ma in genere è necessario installare un modulo apposito.

Gli utenti Windows possono fare riferimento alla libreria Win32::SerialPort (<http://search.cpan.org/dist/Win32-SerialPort/>) per gli altri sistemi operativi si può utilizzare la libreria Device::SerialPort. L'installazione richiede di impostare un comando simile al seguente:

```
maik> perl -MCPAN -e 'install Device::SerialPort'
```

A questo punto si può scrivere il programma Perl.

[SerialProgramming/perl/analog_reader.pl](#)

```
Riga 1  use strict;
-      use warnings;
-      use Device::SerialPort;
-
5   if ($#ARGV != 0) {
-        die "You have to pass the name of a serial port.";
-    }
-
-    my $serial_port = $ARGV[0];
10  my $arduino = Device::SerialPort->new($serial_port);
-    $arduino->baudrate(9600);
-    $arduino->databits(8);
-    $arduino->parity("none");
-    $arduino->stopbits(1);
15  $arduino->read_const_time(1);
-    $arduino->read_char_time(1);
-
-    sleep(2);
-    while (1) {
20    $arduino->write("a0\n");
-      my ($count, $line) = $arduino->read(255);
-      print $line;
-    }
```

Questo programma verifica la presenza sulla riga di comando del nome della porta seriale, poi crea una nuova istanza `Device::SerialPort` alla riga 10. A questo punto si configurano i parametri della porta seriale e alla riga 15 si imposta il valore di timeout relativo alle chiamate della funzione `read()`. Se non si imposta un timeout il metodo `read()` restituisce immediatamente i risultati, senza lasciare ad Arduino il tempo necessario

per rispondere. Il metodo `read_char_time()` imposta un timeout di pausa tra un carattere e il successivo.

Il programma attende per 2 secondi prima di avviare un ciclo di istruzioni che invia la stringa “a0” alla porta seriale e legge la risposta di Arduino tramite il metodo `read()`. Questo metodo richiede come argomento il numero massimo di byte da leggere e restituisce il numero corrente di byte letti e i dati ricevuti. Il programma visualizza infine i dati in output su console.

Di seguito è riportato un esempio di utilizzo del programma:

```
maik> perl analog_reader.pl /dev/tty.usbserial-A60061a3
a0: 496
a0: 366
a0: 320
^C
```

Tutto qui! Bastano una ventina di istruzioni Perl per realizzare un client di lettura dei dati analogici rilevati dalla scheda Arduino. Anche Perl è una scelta eccellente per programmare i client software di Arduino.

Bibliografia

Amedeo E., *C Pocket*, Apogeo, 2007.

Butcher P., *Debug It!: Find, Repair, and Prevent Bugs in Your Code*, The Pragmatic Programmers, LLC, Raleigh, NC e Dallas, TX, 2009.

Greenberg I., *Processing: Creative Coding and Computational Art*, Apress, Berkeley, CA, USA, 2007.

Kernighan B.W. e Ritchie D., *Il linguaggio C: principi di programmazione e manuale di riferimento*, Pearson Prentice Hall, 2004 (ed. orig.: *The C Programming Language*, Prentice Hall PTR, Englewood Cliffs, NJ, seconda edizione, 1998).

Meyers S., *Effective C++: 50 Specific Ways to Improve Your Programs and Designs*, Addison Wesley Longman, Reading, MA, seconda edizione, 1997.

Pine C., *Learn to Program*, The Pragmatic Programmers, LLC, Raleigh, NC e Dallas, TX, 2006.

Platt C., *Make: Electronics*, O'Reilly Media, Inc., Sebastopol, CA, 2010.

Stroustrup B., *The C++ Programming Language*, Addison Wesley Longman, Reading, MA, 2000.

Indice