Corso di programmazione per le scuole con Arduino – PARTE 4

Siamo arrivati all'ultima puntata di questo corso: la volta scorsa abbiamo parlato di come gestire i suoni (tramite microfoni e buzzer). Stavolta presentiamo un singolo progetto che permette di mettere in pratica tutti i concetti di base imparati nelle puntate precedenti. Ma soprattutto, speriamo di stimolare la fantasia degli studenti con un dispositivo che può essere facilmente personalizzato e migliorato. Vedremo, infatti, come realizzare un semplice robot capace di riconoscere le linee disegnate sul pavimento o su un foglio di carta e di muoversi seguendo tali linee. Il dispositivo che progettiamo è molto semplice, ma proprio per questo ogni studente può lavorarci sopra per aggiungere nuove caratteristiche, impiegando in modo creativo le competenze che ha acquisito finora.

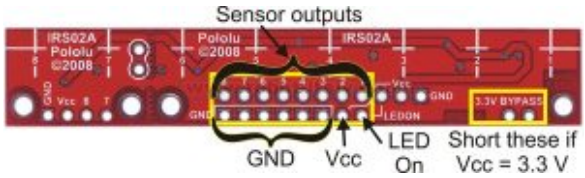
Table of Contents

7 – Un line following robot

Costruire un robot è meno complicato di quanto si possa immaginare, in fondo bastano due servomotori a rotazione continua (continuous rotation servo) e il gioco è fatto. Più

complicato può essere inventarsi un sistema per controllare lo spostamento del robot, ma esiste sempre il meccanismo del "line following", ovvero del seguire le linee. L'idea è semplice: basta disegnare sul pavimento una linea con un buon contrasto (per esempio con un pennarello nero su un foglio di carta bianco). Esiste un metodo piuttosto semplice per permettere ad una scheda Arduino di riconoscere la linea nera: dei sensori infrarossi. Banalmente, basta accoppiare un led a infrarossi con una fotoresistenza (sempre a infrarossi), montandoli sotto al robot. In questo modo, quando la luce infrarossa del led colpisce una zona bianca, la fotoresistenza riceverà un riflesso molto luminoso, mentre quando la luce del led colpisce un punto nero la fotoresistenza riceverà un riflesso molto debole o addirittura nullo (perché il bianco riflette la luce ed il nero la assorbe). Naturalmente il meccanismo funzionerebbe anche con dei normali led colorati, come quelli che abbiamo già utilizzato in precedenza. Però per non avere interferenze dovremmo far correre il robot in una stanza buia, perché la luce del Sole o delle lampade si sovraporrebbe a quella del led. Utilizzando dei led ad infrarossi il problema è risolto, e il vantaggio è che si tratta comunque di banalissimi led, che funzionano come tutti gli altri. Esistono addirittura dei set già pronti con led infrarossi e fotoresistenze, come il QTR-8A, che abbiamo preso come base per il nostro esempio. Il QTR-8A è molto semplice da utilizzare: basta fissarlo sotto al nostro robot, in mezzo ai due servomotori. Il pin Vcc va collegato al pin 5V di Arduino, mentre il pin GND va connesso al GND di Arduino. Poi sono disponibili ben otto pin di segnale: infatti il QTR-8A dispone di 8 coppie di led e fotoresistenze infrarosse, ed ogni fotoresistenza ha un pin che offre a Arduino il proprio segnale analogico, cioè la lettura della luminosità del pavimento. Però Arduino Uno ha soltanto 6 pin analogici: poco male, collegheremo ad Arduino soltanto 6 dei pin del QTR-8A. Per non fare confusione li collegheremo in ordine: il pin 1 del QTR-8A andrà connesso al pin analogico 0 di Arduino, il pin 2 del QTR-8A andrà collegato al pin analogico 1 di

Arduino, e così via. Se avete una scheda più grande, come Arduino Mega, potete utilizzare tutti i pin del QTR-8A, perché un Arduino Mega ha ben 16 pin analogici. In realtà, però, per la maggioranza delle applicazioni 6 coppie di led e fotoresistenze infrarosse sono più che sufficienti.



I contatti della scheda QTR-8A prevedono il Vcc (5V), il GND, e gli input analogici di Arduino

L’idea di base è molto semplice: abbiamo una fila di 6 sensori sotto al robot, ed in teoria vorremmo che la riga nera si trovasse sempre in corrispondenza della metà dei sensori (ovvero tra il terzo ed il quarto). Questo significa che, se facciamo una media della luminosità rilevata dai primi tre sensori ed una media di quella rilevata dagli ultimi tre sensori, è ovvio che dovrebbero essere più o meno uguali. Se la media dei primi tre sensori (che possono essere quelli di sinistra) è più alta significa che la linea nera si trova dalla loro parte, e quindi dovremo far ruotare il robot nella giusta direzione (per esempio destra) per far tornare la linea nera al centro. Infatti, i sensori del QTR-8A funzionano al contrario rispetto ad una normale fotoresistenza: offrono il valore massimo quando la luminosità ricevuta è bassa, e viceversa. Naturalmente, destra e sinistra dipendono dal verso in cui si monta il QTR-8A sul robot: se il robot non si comporta come dovrebbe, basta ruotare di 180° la scheda con tutti i sensori infrarossi.

Utilizzare un radiocomando

Un altro metodo per controllare un robot realizzato con Arduino è utilizzare un radiocomando: i radiocomandi non sono altro che un insieme di potenziometri (le varie leve presenti sul radiocomando sono potenziometri) i cui valori vengono trasmessi a distanza tramite onde radio. Quindi basta collegare il ricevitore del radiocomando ad Arduino, alimentandolo con i pin 5V e GND, e connettendo tutti i vari pin di segnale (ce n'è uno per ciascuna leva del radiocomando) ai pin analogici di Arduino. Arduino può poi leggere i valori dei potenziometri, e dunque capire se sia stata spostata una levetta in tempo reale e reagire di conseguenza (per esempio spegnendo od accendendo uno dei due servomotori).
http://www.instructables.com/id/RC-Control-and-Arduino-A-Complete-Works/?ALLSTEPS

Il codice per il line following

Il codice che fa funzionare questo programma è probabilmente il più complesso che abbiamo analizzato finora, ma è comunque abbastanza semplice da capire:

Prima di tutto si include nel programma la libreria necessaria per il funzionamento dei servomotori.

Dichiariamo poi due variabili speciali, degli "oggetti" (ne avevamo già parlato, sono variabili che possono avere proprietà e funzioni tutte loro), che rappresenteranno i due servomotori. Come abbiamo già accennato, il nostro robot ha un

totale di due ruote motrici, ciascuna mossa da un servomotore: una a destra (right) e l'altra a sinistra (left). Possiamo poi aggiungere una terza ruota centrale, per esempio una rotella delle sedie da ufficio, solo per tenere il robot in equilibrio. Una sorta di triciclo al contrario, visto che le ruote motrici sono due e non una.

Per poter eseguire i nostri calcoli, dovremo tarare i sensori: dovremo capire quale sia il valore massimo (**mx**), minimo (**mn**), e medio (**mid**) di luminosità rilevabile dai vari sensori. Quindi dichiariamo delle variabili che ci serviranno per memorizzare questi valori.

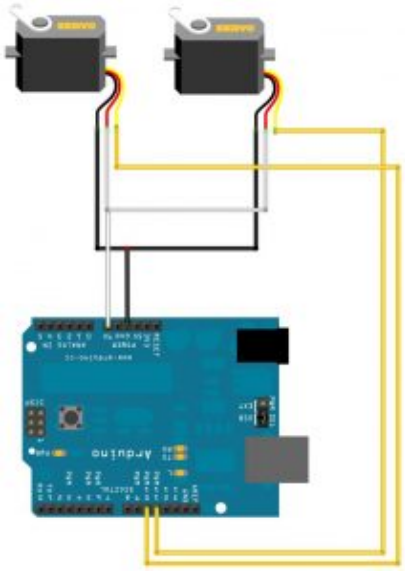
Cominciamo ora a scrivere la funzione **setup**, che viene eseguita all'avvio di Arduino.

Dobbiamo assegnare i due oggetti di tipo servo, **left** e **right**, ai pin digitali di Arduino: abbiamo deciso di collegare il pin **9** al servomotore sinistro ed il pin **10** al servomotore destro. Ricordiamo che devono essere pin PWM, indicati sulla scheda Arduino col simbolo tilde (cioè ~). Indichiamo anche i due valori di minimo e massimo per gli impulsi che faranno muovere il servomotore: di solito non è necessario specificarli (lo fa automaticamente Arduino), ma può essere utile per evitare problemi quando si usano servomotori a rotazione continua come nel nostro caso.

Attiviamo anche la comunicazione sulla porta seriale, così da poter inviare messaggi ad un computer per capire se qualcosa

non stia funzionando nel nostro robot.

Visto che siamo all'inizio, spegniamo il led collegato al pin digitale numero **13** di Arduino (è un led di segnalazione saldato sulla scheda Arduino). Spegniamo anche i due servomotori, così il robot resterà fermo. Con dei servomotori a rotazione continua, lo spegnimento si esegue dando il valore **90** alla funzione write di ciascun oggetto **left** e **right**.



Collegare due servomotori ad Arduino è molto semplice

Calibrare i sensori infrarossi

Attenderemo 5 secondi, ovvero 5000 millisecondi, eseguendo una misura della luminosità di ciascun sensore ogni millisecondo, grazie ad un semplice ciclo **for** che viene ripetuto per 5000 volte.

Accendiamo il led connesso al **pin 13**, quello saldato su Arduino, per avvisare che la calibrazione dei sensori è in atto.

Dobbiamo ora capire quali siano i valori massimo e minimo che si possano misurare con i nostri sensori: per farlo scorriamo tutti i sensori, dal pin analogico **0** al pin analogico **5** di Arduino, leggendo con **analogRead** il valore misurato al momento. Si suppone che il robot si trovi già sopra alla linea, e che almeno alcuni dei sensori siano proprio sopra alla linea nera mentre altri siano sopra al pavimento bianco. Il valore di ogni sensore viene inserito nella variabile **val**. Se tale variabile è maggiore dell'attuale valore massimo (**mx**), allora essa viene considerata il nuovo massimo, assegnando il suo valore a **mx**. Allo stesso modo, se **val** è minore del valore più basso finora registrato **mn**, alla variabile **mn** viene assegnato al valore della variabile **val**.

Prima di terminare il ciclo **for** da 0 a 5000 inseriamo la funzione **delay** chiedendole di attendere 1 millisecondo. Così siamo sicuri che il ciclo durerà in totale almeno 5000 millisecondi, ovvero 5 secondi. In realtà durerà un po' di più, perché le varie operazioni richiedono una certa quantità di tempo, ma sarà probabilmente impercettibile.

Ottenuti i valori massimi e minimi che i sensori possono fornire, possiamo calcolare il valore medio, da memorizzare nella variabile **mid**.

Per segnalare che la calibrazione dei sensori è terminata, spegniamo il led connesso al pin digitale **13** di Arduino.

Finora abbiamo solo calibrato i sensori, ma il robot è ancora fermo. Cominciamo ora la funzione di **loop**, quella che farà muovere il nostro robot.

Per cominciare leggiamo i valori di tutti i sensori, inserendoli in apposite variabili chiamate **s0**, **s1**, **s2**, eccetera.

Iniziamo a muovere il robot: per far girare un servomotore a rotazione continua si può indicare un numero da **0** a **90** oppure da **90** a **180**.
Il valore **180** rappresenta la massima velocità in una direzione, **0** rappresenta la massima velocità nell'altra direzione, e **90** rappresenta la posizione di stallo (quindi il servomotore è fermo).

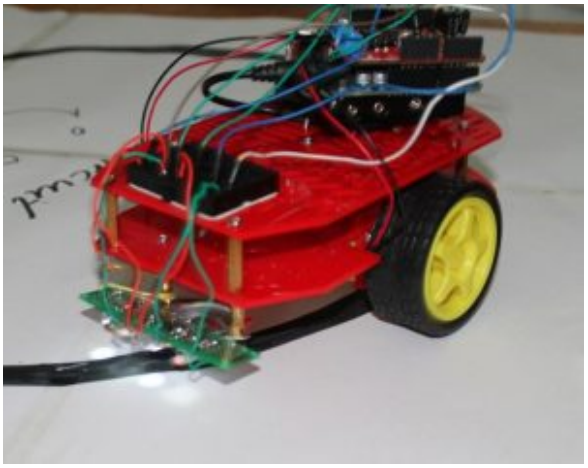
Un semplice miglioramento

Un semplice modo per migliorare il robot può consistere nel rendere variabile la velocità. Se infatti i motori sono sempre impostati a 0 o 180 il robot va sempre alla massima velocità. Si può utilizzare, per esempio, un sensore a ultrasuoni per ridurre la velocità se ci si sta avvicinando a un ostacolo, mappando (funzione **map**) la distanza in un valore da 90 a 0 e da 90 a 180.

Siccome i nostri servomotori sono montati in modo da essere uno speculare all'altro, è ovvio che per far andare il robot avanti uno dei servomotori girerà in una direzione e l'altro nell'altra, così alla fine le due ruote gireranno all'unisono.

Attendiamo un millisecondo, soltanto per essere sicuri che il comando di movimento dei servomotori sia stato applicato.

Abbiamo memorizzato nelle variabili **s0**, **s1**, eccetera, i valori dei vari sensori. Però, come abbiamo detto prima di cominciare a scrivere il programma, noi vogliamo semplicemente comparare la media dei sensori di sinistra con quella dei sensori di destra. Abbiamo deciso che i sensori di **sinistra** siano quelli che sono collegati ai pin analogici **0**, **1**, e **2** di Arduino, mentre quelli di **destra** siano i sensori connessi ai pin analogici **3**, **4**, e **5** (ovviamente dipende da come montiamo il **QTR-8A** sotto al robot). Le due medie si calcolano banalmente con la classica formula matematica: si fa la somma e si divide per 3. Ovviamente, la media potrebbe essere un numero con decimali (con la virgola), ma a noi basta un numero intero: siccome abbiamo definito le due variabili come tipo **int**, Arduino arrotonderà automaticamente i decimali al numero intero più vicino.



Il robot posizionato sopra

la linea nera disegnata su
un pavimento bianco

La linea è a destra o a sinistra?

Normalmente, il robot continua a muoversi in avanti. Però, se la media dei sensori di sinistra è maggiore di quelli dei sensori di destra significa che la linea nera sul pavimento si trova dalla parte sinistra del robot.

Abbiamo indicato anche un fattore correttivo, pari a **240**, per avere un certo lasco: se avessimo scritto soltanto **averageLeft>averageRight** il blocco **if** verrebbe eseguito anche per variazioni minime dei sensori infrarossi tra la parte destra e quella sinistra del robot. Ma vi sarà sempre qualche piccola variazione, anche solo per minime interferenze o oscillazioni nella corrente. Inserendo un fattore correttivo ci assicuriamo che la condizione di **if** venga attivata soltanto se la differenza tra la parte destra e sinistra del robot è notevole.

Ovviamente, se la linea nera è alla sinistra del robot, dovremo ruotare il robot verso sinistra in modo da riportarlo in una posizione in cui la linea nera sia esattamente al centro del robot stesso. E per far ruotare il robot verso sinistra dobbiamo tenere fermo il servomotore di sinistra, dandogli il valore **90**, di modo che faccia da perno, e muovere il servomotore di destra con un valore vicino a **180**. Scegliamo un valore inferiore a 180 perché vogliamo che il servomotore

di destra si muova ma non alla sua massima velocità, così il movimento è più lento e più facile da controllare (se si esagera si rischia di finire al di fuori della linea nera).

Prima di concludere il blocco **if** attendiamo una certa quantità di millisecondi, ottenuta come la metà della differenza delle due medie. L'idea è che in questo modo maggiore è la differenza tra i sensori di destra e quelli di sinistra, maggiore è dunque la dimensione della linea nera, e quindi maggiore è il tempo necessario durante lo spostamento del robot per riuscire ad arrivare ad avere la linea nera al centro del robot stesso.
Siccome la differenza dei due valori potrebbe essere un numero negativo, utilizziamo la funzione **abs** per ottenere il valore assoluto, cioè ottenere la differenza senza segno (in pratica, un eventuale valore **-500** diventerebbe semplicemente **500**).

Se la media dei sensori di sinistra è inferiore a quella dei sensori di destra (tenuto sempre conto del solito fattore correttivo), allora significa che la linea nera è posizionata dalla parte destra del robot. Quindi faremo esattamente l'opposto del precedente **if**: terremo fermo il servomotore destro e muoveremo quello sinistro (anche in questo caso non imposteremo la sua velocità al valore massimo, cioè **0**, ma un po' meno, cioè 40). Anche prima di concludere questo blocco if, attenderemo di nuovo una manciata di millisecondi con lo stesso calcolo precedente.

I percorsi disegnati con nastro adesivo nero su un pavimento chiaro possono anche essere molto lunghi ed elaborati

La fine del percorso

Abbiamo detto al robot cosa fare se la linea nera si trova alla destra oppure alla sinistra del robot stesso. E se invece la linea nera coprisse tutto il pavimento? Significherebbe che il percorso è terminato. Infatti, quando disegniamo il percorso sul pavimento, a meno che non sia un circuito chiuso su se stesso come quelli automobilistici, possiamo indicarne la fine dipingendo di colore nero un bel rettangolo perpendicolare all'ultima parte della linea.

In questo modo quando il robot ci arriva sopra si accorgerà che tutti i suoi sensori, in particolare il sensore **s0** ed il sensore **s5**, che sono il primo e l'ultimo, hanno un valore che è superiore alla media. Questo significa che tutti i sensori sono contemporaneamente sopra alla linea nera, e quindi si è raggiunto il termine del percorso.

In questo caso dobbiamo chiaramente fermare il robot, dando il valore **90** a entrambe i servomotori (come abbiamo già detto,

questo valore provoca l’arresto immediato dei servomotori a rotazione continua).

Per segnalare di avere raggiunto quello che riteniamo essere il termine del percorso (e che quindi il robot non si è fermato per un errore), facciamo lampeggiare rapidamente il led collegato al pin digitale **13** di Arduino, quello saldato sulla scheda. Per farlo lampeggiare 50 volte basta un ciclo **for** che si ripete per l’appunto 50 volte, e ad ogni iterazione non fa altro che accendere il led portando il suo pin al valore **HIGH**, attendere **100** millesimi di secondo, spegnere il led scrivendo il valore **LOW** sul suo pin, ed poi attendere altri **100** millisecondi prima di passare all’iterazione successiva.

Ora attendiamo 5 secondi per essere sicuri che tutte le operazioni necessarie allo spegnimento dei servomotori siano state portate a termine. Durante questi 5 secondi si può tranquillamente spostare il robot, magari posizionandolo di nuovo all’inizio del percorso per farlo ripartire.

La funzione loop si conclude qui, e con essa il programma. Naturalmente, ricordiamo che la funzione loop viene ripetuta di continuo, quindi il robot continuerà a muoversi lungo la linea nera tracciata sul pavimento bianco finché non lo fermiamo facendolo passare sopra ad un rettangolo nero largo tanto quanto l’intera scheda **QTR-8A**, in modo che tutti i sensori infrarossi si trovino contemporaneamente sopra al colore nero.

Costruire un vero Go-Kart con Arduino

Il robot che presentiamo è pensato per essere piccolo e semplice. Ma il bello di Arduino è che si può facilmente salire di scala: basta avere motori più potenti ed un buon telaio, e si può costruire un Go Kart capace di trasportare una persona, controllando il motore elettrico (ne basta uno, visto che la trazione è posteriore) con Arduino. Il progetto che vi suggeriamo utilizza un motore brushless, un Savox BSM5065 450Kv. I motori brushless sono una via di mezzo tra servomotori ed i normali motorini elettrici a spazzole. I motori brushless eseguono rotazioni continue (quindi non si fermano a 180°), e sono controllabili con Arduino (si può controllare la direzione e la velocità). Hanno anche il notevole vantaggio di poter fornire molta potenza e dare dunque una notevole velocità al mezzo su cui vengono montati. Inoltre sono quasi immuni all'usura, rispetto ai motori a spazzole. Ne esistono di piccolissimi, per progetti di modellismo, oppure di enormi (le automobili elettriche usano motori brushless).
http://www.instructables.com/id/Electric-Arduino-Go-kart/?ALLSTEPS

Il codice sorgente

Potete trovare il codice sorgente dei vari programmi presentati in questo corso di introduzione alla programmazione con Arduino nel repository:
https://www.codice-sorgente.it/cgit/arduino-a-scuola.git/tree/
Il file relativo a questo articolo è 7-robot-linefollowing.ino.

Corso di programmazione per le scuole con Arduino – PARTE 2

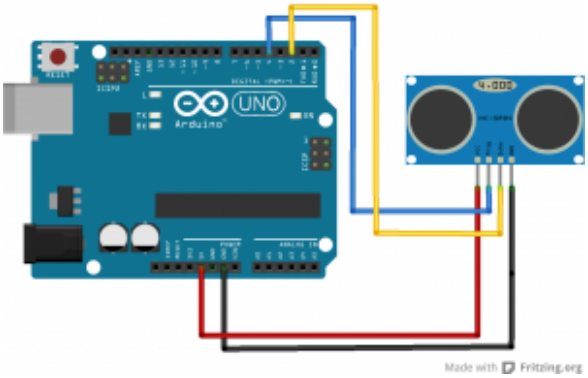
Nella scorsa puntata di questo corso abbiamo accennato ai concetti fondamentali della programmazione, dalle variabili alle funzioni per leggere il valore dei sensori. In questa seconda lezione approfondiamo le funzioni e gli oggetti più utili nel mondo di Arduino. Vedremo come utilizzare i sensori digitali e come accedere a pagine web per estrarre informazioni. È il concetto di API web, molto più semplice nella pratica di quanto la teoria possa far supporre, una abilità fondamentale per realizzare oggetti "intelligenti", in grado di reagire a informazioni che ricevono dall'esterno. Vedremo anche che il controllo di un servomotore non è troppo diverso da quello di un led, quindi gli studenti potranno pensare alla realizzazione di sistemi in movimento.

Naturalmente, qui presentiamo le principali caratteristiche di Arduino con degli esempi pratici, adattabili a vari ambiti, dalla scuola all'arte e il design. Gli insegnanti devono ricordare che si tratta più che altro di spunti utili soprattutto per capire la logica di Arduino e la programmazione. Naturalmente ciascuno dei nostri progetti può essere modificato, per esempio usando sensori diversi. Per imparare davvero come funzioni Arduino, infatti, la cosa migliore consiste proprio nel fare molti tentativi, in modo da prenderci la mano e soprattutto sviluppare la fantasia necessaria a risolvere creativamente i problemi che possono presentarsi.

3 Accendere un led in dissolvenza man mano che ci si avvicina ad Arduino

Una cosa interessante dei led è che possono essere accesi “a dissolvenza”. Cioè, invece di passare in un attimo dal completamente spento al completamente acceso, possono aumentare gradualmente luminosità fino ad accendersi completamente (e anche viceversa, possono diminuire luminosità gradualmente fino a spegnersi). Un tale comportamento è ovviamente **analogico**, non digitale, perché il digitale contempla solo lo stato acceso e quello spento, non lo stato acceso al 30% oppure acceso all’80%. Arduino non ha pin analogici di output, li ha solo di input, ma ha alcuni pin digitali che possono simulare un output analogico. Questi si chiamano **PWM** (Pulse With Modulation), e sono contraddistinti sulla scheda Arduino tra i vari pin digitali dal simbolo tilde (cioè ~). Se vogliamo poter accendere in dissolvenza un led, dobbiamo collegarlo ad uno dei pin digitali indicati dal simbolo ~, e poi assegnare a tale pin un valore compreso tra 0 e 255 utilizzando non la funzione digitalWrite, che abbiamo visto nell’esempio della puntata precedente, ma la funzione **analogWrite**. Naturalmente, per aggiungere una certa interattività, ci serve un sensore: possiamo utilizzare il sensore **HC RS04**. Si tratta di un piccolo sensore ad ultrasuoni capace di leggere la distanza tra il sensore stesso e l’oggetto che ha di fronte (rileva oggetti tra i 2cm ed i 400cm di distanza da esso, con una precisione massima di 3mm). Il suo pin **Vcc** (primo da sinistra) va collegato al 5V di Arduino, mentre il pin **GND** (primo da destra) va collegato al GND di Arduino. Il pin **Trig** (secondo da sinistra) va collegato al pin digitale 9 di Arduino, mentre il suo pin **Echo** (secondo da destra) va collegato al pin digitale 10 di Arduino. Il codice del programma che fa ciò che vogliamo è il seguente:

Come si può ormai immaginare, il programma inizia con la solita dichiarazione delle variabili. La variabile **triggerPort** indica il pin di Arduino cui collegato il pin **Trig** del sensore, così come la **echoPort** indica il pin di Arduino cui è collegato il pin **Echo** del sensore. La variabile **led** indica il pin cui è collegato il led: ricordiamoci che deve essere uno di quelli contrassegnati dal simbolo ~ sulla scheda Arduino.



Il sensore di distanza HCSR04 ha 4 pin da collegare ad Arduino

La funzione **setup** stavolta si occupa di impostare la modalità dei tre pin digitali, chiamando per tre volte la funzione **pinMode** che abbiamo già visto. Stavolta, però, due pin (quello del trigger del sensore e quello del led) hanno la modalità **OUTPUT**, mentre il pin dedicato all'echo del sensore ha modalità **INPUT**. Infatti, il sensore funziona emettendo degli ultrasuoni grazie al proprio pin trigger, e poi ascolta il loro eco di ritorno inviando il segnale ad Arduino tramite il pin echo. Il principio fisico alla base è che più distante è un oggetto, maggiore è il tempo necessario affinché l'eco ritorni indietro e venga rilevato dal sensore: è un sonar, come quello delle navi o dei delfini.

Prima di concludere la funzione setup, provvediamo ad aprire

una comunicazione sulla **porta seriale**, così potremo leggere dal computer il dato esatto di distanza dell'oggetto.

Inizia ora la funzione loop, qui scriveremo il codice che utilizza il sensore a ultrasuoni.

Il sensore a ultrasuoni
Quando si vuole realizzare qualcosa di interattivo ma non troppo invasivo, il sensore a ultrasuoni è una delle opzioni migliori. Noi ci siamo basati sull'HCSR04, uno dei più comuni ed economici, si può trovare su Ebay ed AliExpress per più o meno 2 euro. Misurando variazioni nella distanza il sensore può anche permetterci di capire se la persona posizionata davanti ad esso si stia muovendo, quindi possiamo sfruttarlo anche come sensore di movimento.

L'impulso ad ultrasuoni

Abbiamo detto che il sonar funziona inviando un impulso a ultrasuoni e ascoltandone l'eco. Dobbiamo quindi innanzitutto inviare un impulso: e lo faremo utilizzando la funzione **digitalWrite** per accendere brevemente il pin cui è collegato il trigger del sensore.

Prima di tutto il sensore deve essere spento, quindi impostiamo il pin al valore **LOW**, ovvero 0 Volt. Poi accendiamo il sensore in modo che emetta un suono (nelle frequenze degli ultrasuoni) impostando il pin su **HIGH**, ovvero dandogli 5 Volt. Ci basta un impulso molto breve, quindi dopo avere atteso 10 microsecondi (cioè 0,01 millesimi di secondo) grazie alla funzione **delayMicroseconds**, possiamo spegnere di nuovo il pin

che si occupa di far emettere il suono al sensore portandolo di nuovo agli 0 Volt del valore **LOW**. Insomma, si spegne l'emettitore, lo si accende per una frazione di secondo, e lo si spegne: questo è un impulso.

Ora dobbiamo misurare quanto tempo passa prima che arrivi l'eco dell'impulso. Possiamo farlo utilizzando la funzione **pulseIn**, specificando che deve rimanere in attesa sul pin echo del sensore finché non arriverà un impulso di tipo **HIGH**, ovvero un impulso da 5V (impulsi con voltaggio inferiore potrebbero essere dovuti a normali disturbi nel segnale). La funzione ci fornisce il tempo in millisecondi, quindi per esempio 3 secondi saranno rappresentati dal numero 3000. Possiamo registrare questo numero nella variabile **durata**, che dichiariamo in questa stessa riga di codice. La variabile è dichiarata come tipo **long**: si tratta di un numero intero molto lungo. Infatti, su un Arduino una variabile di tipo **int** arriva al massimo a contenere il numero **32768**, mentre un numero intero di tipo long può arrivare a **2147483647**. Visto che la durata dell'eco può essere un numero molto grande, se utilizzassimo una variabile di tipo int otterremmo un errore. Ci si potrebbe chiedere: perché allora non si utilizza direttamente il tipo long per tutte le variabili? Il fatto è che la memoria di Arduino è molto limitata, quindi è meglio non intasarla senza motivo, e usare **long** al posto di **int** soltanto quando è davvero necessario.

Come abbiamo visto per il sensore di temperatura, anche in questo caso serve una semplice formula matematica per ottenere la distanza (che poi è il dato che ci interessa davvero) sulla base della durata dell'impulso. Si moltiplica per 0,034 e si divide per 2, come previsto dai produttori del sensore che stiamo utilizzando. E anche in questo caso la variabile **distanza** va dichiarata come tipo **long**, perché può essere un numero abbastanza grande.

Ora possiamo scrivere un messaggio sulla porta seriale, per comunicare al computer la distanza rilevata dal sensore.

Naturalmente dobbiamo tenere conto di un particolare: il sensore ha dei limiti, non può misurare la distanza di oggetti troppo lontani. Secondo i produttori, se il sensore impiega più di 38 secondi (ovvero **38000** millisecondi) per ottenere l'eco, significa che l'oggetto che si trova di fronte al sensore è troppo distante. Grazie ad un semplice **if** possiamo decidere di scrivere un messaggio che faccia sapere a chi sta controllando il monitor del computer che siamo fuori portata massima del sensore.

Continuando l'if con un **else**, possiamo dire ad Arduino che se, invece, il tempo trascorso per avere un impulso è inferiore ai 38000 millisecondi, la distanza rilevata è valida. Quindi possiamo scrivere la distanza, in centimetri, sulla porta seriale, affinché possa essere letta dal computer collegato ad Arduino tramite USB.

Mappare i numeri

È finalmente arrivato il momento di eseguire la dissolvenza sul led collegato ad Arduino. Noi vogliamo che la luminosità del led cambi in base alla durata dell'impulso (la quale è, come abbiamo detto, proporzionale alla distanza dell'oggetto più vicino). Però la variabile **durata** può avere un qualsiasi valore tra 0 e **38000**, mentre il led accetta soltanto valori di luminosità che variano tra 0 e **255**.

Poco male: abbiamo già visto la volta scorsa che per risolvere

questo problema possiamo chiamare la funzione **map**, indicando la variabile che contiene il numero da tradurre e i due intervalli. Il risultato viene memorizzato in una nuova variabile di tipo **int** (del resto è comunque un numero piccolo) che chiamiamo **fadeValue**. Questa variabile conterrà, quindi, un numero da 0 a 255 a seconda della distanza del primo oggetto che si trova di fronte al sensore.

Possiamo concludere impostando il valore appena calcolato, memorizzato nella variabile **fadeValue**, come valore del pin cui è collegato il LED (identificato dalla variabile led che avevamo dichiarato all'inizio del programma). Per farlo utilizziamo la funzione **analogWrite**, che per l'appunto non si limita a scrivere "acceso" o "spento" come la **digitalWrite**, ma piuttosto permette di specificare un numero (da 0 a 255, proprio come quello appena calcolato) per indicare la luminosità desiderata del led. Ora il ciclo else è completato, lo chiudiamo con una parentesi graffa.

Prima di concludere definitivamente la funzione loop, con una ultima parentesi graffa chiusa, utilizziamo la funzione **delay** per dire ad Arduino di aspettare **1000** millisecondi, ovvero 1 secondo, prima di ripetere la funzione (e dunque eseguire una nuova lettura della distanza con il sensore ad ultrasuoni).

Controllare anche i led da illuminazione

Arduino fornisce, tramite i suoi pin digitali, al massimo 5Volt e 0,2Watt, che sembrano pochi ma sono comunque sufficienti per accendere i classici led di segnalazione ed alcuni piccoli led da illuminazione che si trovano nei negozi di elettronica. Ma è comunque possibile utilizzare Arduino per

accendere led molto più potenti, anche fino a 90Watt (ed una normale lampadina led domestica ha circa 10W), utilizzando un apposito Shield, ovvero un circuito stampato da montare sopra ad Arduino: https://www.sparkfun.com/products/10618.

4 Leggere l'ora da internet e segnalarla con una lancetta su un servomotore

In questo progetto affrontiamo due temi importanti: il movimento e internet. Arduino, infatti, può manipolare molto facilmente dei servomotori, e dunque si può utilizzare per applicazioni meccaniche di vario tipo, anche per costruire dei robot. Inoltre, Arduino può essere collegato a internet: esistono diversi metodi per farlo, a seconda della scheda che si sta utilizzando. Un Arduino Uno può essere collegato ad internet tramite lo shield Ethernet, acquistabile a parte sia in versione semplice che con PowerOverEthernet (Arduino verrebbe alimentato direttamente dal cavo ethernet). Questo è lo scenario più tipico, ed è quello su cui ci basiamo per il nostro esempio. In alternativa, si può ricorrere a un WemosD1, che è fondamentalmente un Arduino Uno con un chip wifi integrato, abbastanza facile da programmare e economico (costa poco più di un Arduino Uno). Chiaramente, il WiFi va configurato usando le apposite funzioni indicando la password di accesso alla rete, cosa che non serve per le connessioni ethernet. Esiste anche l'ottimo Arduino Yun, che integra sia una porta ethernet che una antenna WiFi, ed ha una comoda interfaccia web per configurare la connessione, senza quindi la necessità di configurare il WiFi nel codice del proprio programma. Arduino Yun è il più semplice da utilizzare, ma è un po' costoso (circa 50 euro), mentre la coppia Arduino Uno + Ethernet shield è molto più economica (la versione ufficiale arriva al massimo a 40 euro, e si può comprare una riproduzione made in China per meno di 10 euro su AliExpress).

Lo shield deve semplicemente essere montato sopra ad Arduino. Il servomotore, invece, va collegato ad Arduino in modo che il polo positivo (**rosso**) sia connesso al pin **5V**, mentre il negativo (**nero**) sia connesso al pin **GND** di Arduino. Infine, il pin del segnale del servomotore (tipicamente **bianco** o in un altro colore) va collegato ad uno dei pin digitali **PWM** di Arduino, quelli contrassegnati dal simbolo **~** che abbiamo già visto per la dissolvenza del led. Nel nostro progetto di prova, realizzeremo un orologio: per semplificare le cose, utilizzeremo il servomotore per muovere soltanto la lancetta delle ore, tralasciando quella dei minuti. Il codice è il seguente:

Sono necessarie tre librerie esterne: **SPI** ed **Ethernet** ci servono per utilizzare la connessione ethernet. Invece, **Servo** è utile per controllare il servomotore.

Proprio per utilizzare il servomotore si deve creare una variabile speciale, che chiamiamo **myservo**, di tipo **Servo**. Questa non è proprio una variabile, è un "oggetto". Gli **oggetti** sono degli elementi del programma che possono avere una serie di proprietà (variabili) e di funzioni tutte loro. Per esempio, l'oggetto **myservo** appena creato avrà tra le sue proprietà la posizione del motore, mentre l'oggetto **client** avrà tra le sue funzioni "personali" una funzione che esegue la connessione a internet. Gli oggetti servono a risparmiare tempo e rendere la programmazione più intuitiva: lo vedremo tra poco.

Una scheda Arduino Uno con lo shield ethernet montato sopra di essa

La pagina web da leggere

Dichiariamo due variabili che ci permettono di impostare il pin cui è collegato il segnale del servomotore, ed il MAC dell'Ethernet shield. Ogni dispositivo ethernet ha infatti un codice MAC che lo identifica: possiamo scegliere qualsiasi cosa, anche se in teoria dovremmo indicare quello che è stampato sull'ethernet shield.

È importante non avere nella propria rete locale due dispositivi con lo stesso codice MAC, altrimenti il router non riesce a distinguerli. Quindi basta cambiare un numero (esadecimale) qualsiasi nel codice dei programmi che si scrivono.

Ora specifichiamo due informazioni: il **nome del server** che vogliamo contattare, e la **pagina** che vogliamo leggere da esso. Per far leggere ad Arduino la pagina web http://codice-sorgente.it/UTCTime.php?zone=Europe/Rome, dobbiamo dividere il nome del server dal percorso della pagina, perché questo è richiesto dal protocollo HTTP. Questa pagina è un classico esempio di **API** web, ovvero una pagina web

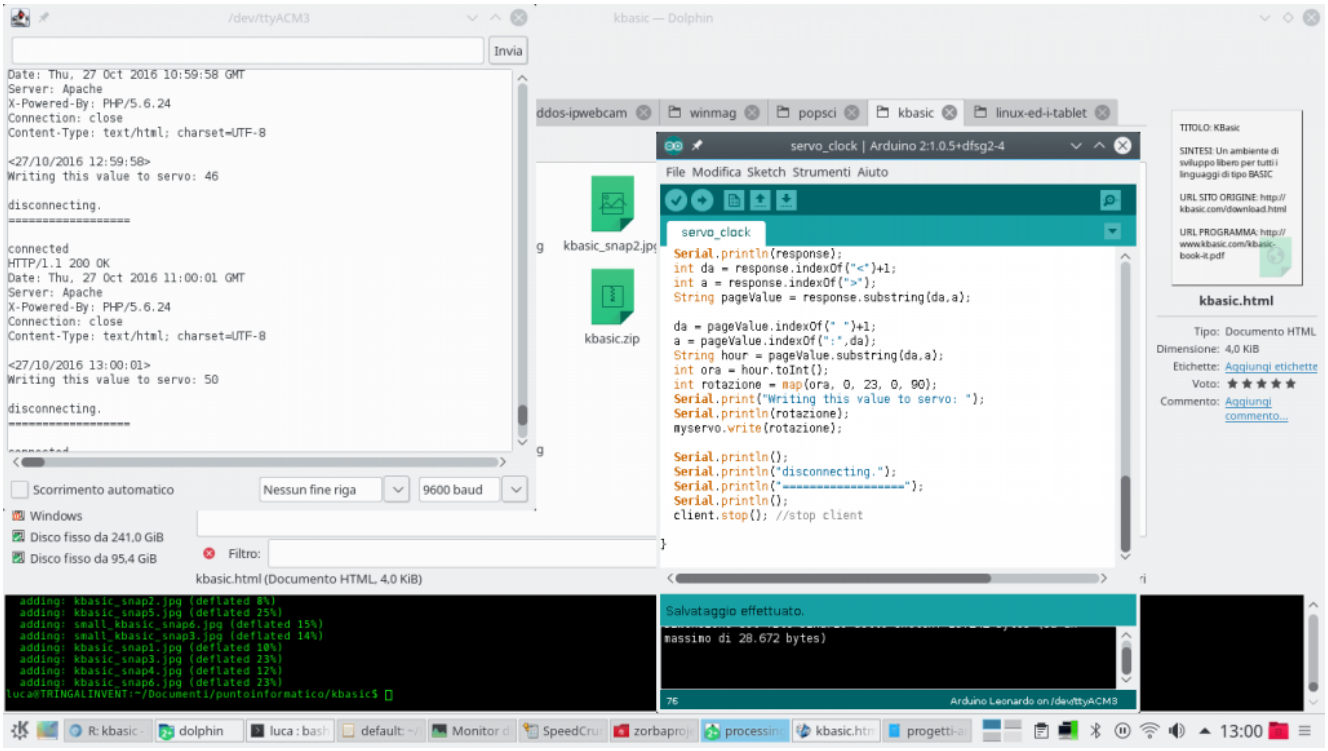
messa a disposizione di programmatori per ottenere informazioni varie. In particolare, questa pagina ci fornisce in tempo reale la data e l'ora per il fuso orario che abbiamo selezionato: nell'esempio scegliamo Roma, ma potremmo indicare Atene scrivendo Europe/Athens.

Altre API web

Abbiamo introdotto il concetto di API web, ovvero di una pagina web che contiene un semplice testo a disposizione dei programmatori. Nel nostro esempio ne utilizziamo una che fornisce l'ora attuale, ma esistono pagine web simili per qualsiasi cosa: uno tra i fornitori più importanti è Google, che offre la possibilità di eseguire ricerche con il suo motore, sfruttare i servizi di Maps, addirittura inviare email. Ma anche Twitter offre API con cui leggere od inviare tweet. Ne esistono migliaia, ed il sito migliore per trovarle è https://www.publicapis.com/. Molte di queste API richiedono un nome utente, che in genere è gratuito ma necessita di una iscrizione al sito web.

Se volete provare a realizzare una vostra pagina web che contenga l'ora attuale, come quella che sfruttiamo nel nostro esempio, dovete soltanto copiare questo file PHP:
https://pastebin.com/JeMf8p84
sul vostro server web (per esempio, Altervista ed Aruba supportano PHP).

La solita funzione **setup** comincia aprendo una comunicazione sulla porta seriale, così potremo leggere i messaggi di Arduino dal nostro computer, se vogliamo. Poi inizia con una condizione **if**: questa ci permette di tentare la connessione al router, utilizzando la funzione **Ethernet.begin**, che richiede come argomento il codice **MAC** che avevamo scritto poco fa.



Con il monitor seriale di Arduino IDE possiamo leggere i messaggi che Arduino invia al nostro computer

La funzione **Ethernet.begin** fornisce una risposta numerica: il numero indica lo stato attuale della connessione, e se lo stato è **0**, significa che non c'è connessione. Quindi l'istruzione **if** può riconoscere questo numero e, nel caso sia **0**, prendere provvedimenti. In particolare, oltre a scrivere un messaggio di errore sulla **porta seriale**, così che si possa leggere dal computer collegato ad Arduino, se non c'è una connessione ethernet viene iniziato un **ciclo while infinito**: questo ciclo dura in eterno, impedendo qualsiasi altra operazione ad Arduino. Lo facciamo per evitare che il programma possa andare avanti se non c'è una connessione. Praticamente, Arduino rimane in attesa di essere spento. Un ciclo è una porzione di codice che viene eseguita più volte: nel caso di un ciclo while (che significa "mentre") il codice contenuto nel ciclo viene ripetuto finché la condizione posta è vera (**true**). In questo caso il ciclo è sempre vero, perché come condizione abbiamo indicato proprio il valore **true**, e non contiene nessun codice, quindi il suo solo effetto è bloccare Arduino. Approfondiremo più avanti i cicli.

Prima di concludere la funzione **setup**, chiamiamo una funzione dell'oggetto **myservo**, che rappresenta il servomotore connesso ad Arduino. La funzione si chiama **attach** e serve a specificare che il nostro servomotore, d'ora in poi rappresentato in tutto e per tutto dal nome **myservo**, è attaccato al pin digitale di Arduino contraddistinto dal numero **3** (numero che avevamo indicato nella variabile **servopin**).

Connettersi al server web

La funzione **loop**, stavolta è molto semplice, perché deleghiamo buona parte del codice ad un'altra funzione: la funzione **sendGET**, che viene chiamata ogni 3 secondi.

In altre parole, la funzione **loop**, che viene eseguita continuamente, non fa altro che aspettare 3000 millisecondi, ovvero 3 secondi, e poi chiamare la funzione **sendGET**: è quest'ultima a fare tutto il lavoro, ed adesso dovremo scriverla.

Le parentesi graffe
Su sistemi Windows, con tastiera italiana, si può digitare una parentesi graffa premendo i tasti **AltGr + Shift + è** oppure **AltGr + Shift + +**. Infatti, i tasti **è** e **+** sono quelli che contengono anche le parentesi quadre. Quindi, premendo solo **AltGr + è** si ottiene la parentesi quadra, mentre premendo **AltGr + Shift + è** si ottiene la parentesi graffa. Su un sistema GNU/Linux, invece, si può scrivere la parentesi graffa premendo i tasti **AltGr + 7** oppure **AltGr + 0**.

Questa è la nostra funzione **sendGET**, nella quale scriveremo il codice necessario per scaricare la pagina web con Arduino e

analizzarla in modo da scoprire l'ora corrente.

Per cominciare si deve aprire una connessione HTTP con il server in cui si trova la pagina web che vogliamo scaricare. Le connessioni HTTP sono (in teoria) sempre eseguite sulla porta **80**, e il nome del server è memorizzato nella variabile **serverName** che avevamo scritto all'inizio del programma: queste due informazioni vengono consegnate alla funzione **connect** dell'oggetto **client**. Se la connessione avviene correttamente, la funzione **connect** fornisce il valore **true** (cioè vero), che viene riconosciuto da **if** e quindi si può procedere con il resto del programma. Come avevamo detto, l'oggetto **client** possiede alcune funzioni (scritte dagli autori di Arduino) che facilitano la connessione a dei server di vario tipo, e per utilizzare le varie funzioni basta utilizzare il punto (il simbolo **.**), posizionandolo dopo il nome dell'oggetto (ovvero la parola **client**).

Per esempio, un'altra funzione molto utile dell'oggetto **client** è **print** (e la sua simile **println**). Esattamente come le funzioni **print** e **println** dell'oggetto **Serial** ci permettono di inviare messaggi sul cavo USB, queste funzioni permettono l'invio di messaggi al server che abbiamo appena contattato. Per richiedere al server una pagina gli dobbiamo inviare un messaggio del tipo **GET pagina HTTP/1.0**. Visto che avevamo indicato la pagina all'inizio del programma, stiamo soltanto inserendo questa informazione nel messaggio con la funzione **print** prima di terminare il messaggio con **println**, esattamente come succede per i messaggi su cavo USB diretti al nostro computer (anch'essi vanno terminati con **println**, altrimenti non vengono inviati).

È poi richiesto un messaggio ulteriore, che specifichi il nome del server da cui vogliamo prelevare la pagina, nella forma **Host: server**. Anche questo messaggio viene inviato come il

precedente.

Ora la connessione è completata ed abbiamo richiesto la pagina al server, quindi il blocco **if** è terminato.
Naturalmente, possiamo aggiungere una condizione **else** per decidere cosa fare se la connessione al server non venisse portata a termine correttamente (ovvero se il risultato della funzione **client.connect** fosse **false**).

In questo caso basta scrivere sul monitor seriale un messaggio per l'eventuale computer collegato ad Arduino, specificando che la connessione è fallita. Il comando **return** termina immediatamente la funzione, impedendo che il programma possa proseguire se non c'è una connessione.

La risposta del server

Il blocco **if** e anche il suo **else** sono terminati. Quindi se il programma sta continuando significa che abbiamo eseguito una connessione al server e abbiamo richiesto una pagina web. Ora dobbiamo ottenere una risposta dal server, che memorizzeremo nella variabile **response**, di tipo **String**. Come abbiamo già detto, infatti, le **stringhe** sono un tipo di variabile che contiene un testo, e la risposta del server è proprio un testo.

La stringa **response** è inizialmente vuota, la riempiremo man mano.

Prima di cominciare a leggere la risposta del server dobbiamo assicurarci che ci stia dicendo qualcosa: potrebbe essere necessario qualche secondo prima che il server cominci a trasmettere. Un ciclo **while** risolve il problema: i cicli while vengono eseguiti continuamente finché una certa condizione è valida. Nel nostro caso, vogliamo due condizioni: una è che il

client deve essere ancora connesso al server, cosa che possiamo verificare chiamando la funzione **client.connected**. L'altra condizione è che il client non sia disponibile, perché se non lo è significa che è occupato dalla risposta del server e quindi la potremo leggere. Possiamo sapere se il client è disponibile con la funzione **client.available**, la quale ci fornisce il valore **true** quando il client è disponibile. Però noi non vogliamo che questa condizione sia vera: la vogliamo falsa, così sapremo che il client non è disponibile. Possiamo negare la condizione utilizzando il **punto esclamativo**. In altre parole, la condizione **client.available** è vera quando il client è disponibile, mentre **!client.available** è vera quando il client non è disponibile. Abbiamo quindi le due condizioni, **client.connected** e **!client.available**. La domanda è: come possiamo unirle per ottenere una unica condizione? Semplice, basta utilizzare il simbolo **&&**, la doppia e commerciale, che rappresenta la congiunzione "e" ("and" in inglese). Quindi la condizione inserita tra le parentesi tonde del ciclo **while** è vera solo se il client è contemporaneamente connesso ma non disponibile. Significa che appena il client diventerà nuovamente disponibile oppure si disconnetterà, il ciclo while verrà fermato ed il programma potrà proseguire.

Ora, se ci troviamo in questo punto del programma, significa che il ciclo precedente si è interrotto, e il client ha quindi registrato tutta la risposta del server alla nostra richiesta di leggere la pagina web. Possiamo leggere la risposta una lettera alla volta utilizzando la funzione **client.read()**. La lettera viene memorizzata in una variabile di tipo **char**, ovvero un carattere, che viene poi aggiunta alla variabile response. Il simbolo **+=**, infatti, dice ad Arduino che il carattere **c** va aggiunto alla fine dell'attuale stringa **response**. Se il carattere è "o" e la stringa è "cia", il risultato dell'operazione sarà la stringa "ciao".
Ovviamente, per leggere tutti i caratteri della risposta del server abbiamo bisogno di un ciclo che ripeta la lettura

finché il client non ha più lettere da fornirci, un ciclo **while**. Per quante volte si deve ripetere il ciclo, cioè con quale condizione lo dobbiamo ripetere? Semplice: dobbiamo ripeterlo finché il client è connesso al server (perché è ovvio che se la connessione è caduta non ha più senso continuare a leggere), oppure finché il client è disponibile (perché se non è più disponibile alla lettura significa che la risposta del server è terminata e non c'è più niente da leggere). Stavolta abbiamo due condizioni, come nel ciclo precedente: una sarà **client.connected** e l'altra **client.available**, ma non le vogliamo necessariamente entrambe vere. Infatti, ci basta che una delle due sia vera per continuare a leggere. Insomma, l'una oppure l'altra. E il simbolo per esprimere la congiunzione "oppure" è **||**, cioè la doppia pipe (il simbolo che sulle tastiere italiane si trova sopra a \).

Se anche questo ciclo while è terminato, significa che tutta la risposta del server è ormai contenuta nella variabile **response**, quindi possiamo inviarla all'eventuale computer connesso ad Arduino tramite cavo USB.

Estrarre l'informazione dalla pagina web

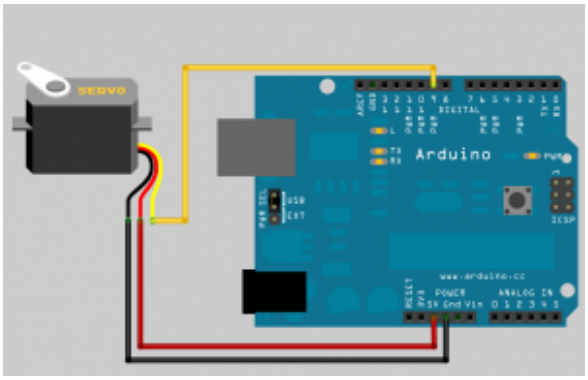
Ora possiamo cominciare a manipolare la risposta del server per estrarre le informazioni che ci interessano.

Innanzitutto, dobbiamo estrarre il contenuto della pagina web: se leggete la risposta completa del server, vi accorgerete che sono presenti anche molte informazioni accessorie che non ci interessano. Dobbiamo identificare il contenuto della pagina web: se provate a visitare la pagina http://codice-sorgente.it/UTCTime.php?zone=Europe/Rome

noterete che il suo contenuto è qualcosa del tipo **<07/03/2019 21:33:56>**. Quindi, il suo contenuto è facilmente riconoscibile in quanto compreso tra il simbolo **<** e il simbolo **>**. Possiamo cercare il simbolo iniziale e quello finale all'interno della stringa **response** utilizzando la funzione **indexOf** tipica di ogni stringa, (funzione cui si accede con il solito simbolo **.**). Per esempio, scrivendo **response.indexOf("<")** ci viene fornita la posizione del simbolo **<** all'interno della stringa **response**. La posizione è un numero, per esempio potrebbe essere **42** se il simbolo **<** fosse il quarantaduesimo carattere dall'inizio del testo. Similmente, si può trovare la posizione del simbolo **>**, ed entrambe le posizioni si memorizzano in due variabili di tipo **int** (visto che sono numeri), che chiamiamo rispettivamente **da** e **a**. Da notare che alla prima posizione abbiamo sommato **1**, altrimenti sarebbe stato considerato anche il simbolo **<**, invece noi vogliamo tenere conto dei caratteri che si trovano dopo di esso. Utilizzando queste due posizioni, possiamo poi sfruttare la funzione **substring** della stessa **stringa** per ottenere tutto il testo compreso tra i due simboli. La funzione **substring** ci fornisce direttamente una stringa, che possiamo inserire in una nuova variabile chiamata **pageValue**.

L'attuale contenuto della stringa **pageValue** è qualcosa del tipo 07/03/2019 21:33:56. Noi siamo interessati soltanto al numero che rappresenta le ore, quindi possiamo ripetere la stessa procedura cambiando i valori delle variabili **da** ed **a**. In particolare, li dobbiamo cambiare affinché ci permettano di identificare il numero delle ore: questo è delimitato a sinistra da uno **spazio**, ed a destra dai **due punti**. Quindi, utilizzando questi due simboli, la funzione **substring** riesce ad estrarre soltanto il numero delle ore, e inserirlo in una nuova variabile. Da notare che, anche se l'ora attuale è ai nostri occhi un numero, per il momento è ancora considerata un testo da parte di Arduino visto che finora lavoravamo con le stringhe. Possiamo trasformare questa informazione in un

numero a tutti gli effetti grazie alla funzione **toInt** della stringa stessa, che traduce il testo “21” nel numero 21. Adesso, la variabile di tipo **int** (un numero intero) chiamata **ora** contiene l’orario attuale (solo le ore, non i minuti ed i secondi).



Collegare un servomotore ad Arduino è facile, basta ricordare che il pin digitale deve essere contrassegnato dal simbolo ~

Muovere il servomotore

Adesso abbiamo l’orario attuale, quindi possiamo spostare il servomotore in modo da direzionare la nostra lancetta. Per muovere un servomotore basta dargli una posizione utilizzando la sua funzione **write**. Quindi, nel nostro caso basta chiamare la funzione **myservo.write**.

Però c’è un particolare, le ore che ci fornisce il sito web vanno da **0** a **23**, mentre le posizioni di un servomotore vanno da **0** a **180**. Infatti, un normale servomotore può ruotare di 180°: se chiamiamo la funzione **myservo.write(0)**, il motore rimarrà nella posizione iniziale, **se chiamiamo myservo.write(45)** verrà posizionato a 45° rispetto alla posizione iniziale. Il problema è: come traduciamo le ore del

giorno in una serie di numeri compresa tra 0 e 180? Semplice, basta utilizzare la funzione **map**, che abbiamo già visto più volte. In questo modo le ore **23** diventerebbero **180**°, le ore **12** diventerebbero **45**°, e così via.

La funzione di mappatura

Abbiamo sempre usato la funzione map predefinita in Arduino. Ma, come esercizio di programmazione, possiamo anche pensare di scrivere una nostra versione della funzione. Che cos'è, quindi questa funzione? È una semplice proporzione, come quelle che si studiano a scuola:

Infatti, basta fare un rapporto tra i due range (quello del valore di partenza e quello del valore che si vuole ottenere). La funzione che abbiamo appena definito fornisce un numero con virgola (double), ma se preferiamo un numero intero ci basta modificare la definizione della funzione come tipo int.

Naturalmente, se vogliamo rappresentare le ore in un ciclo di 12 invece che di 24, basta dividere il numero per 12 tenendo non il quoziente ma il resto. Il resto della divisione si ottiene con l'operatore matematico **%**. Insomma, basta sostituire la variabile **ora** con la formula **(ora%12)**. In questo modo, le 7 rimangono le 7, mentre le 15 diventano le 3 (perché 15 diviso 12 ha il resto di 3).

Prima di completare la funzione, possiamo scrivere qualche messaggio sul cavo USB per far sapere all'eventuale computer connesso ad Arduino che stiamo terminando la connessione al server che ha ospitato la pagina web. Poi la semplice funzione **stop** dell'oggetto **client** ci permette di concludere la connessione, e con la solita parentesi graffa chiudiamo anche la funzione, ed il programma è terminato.

Abbiamo detto che il nostro servomotore ruota di 180°: esistono anche altri servomotori modificati per poter eseguire una rotazione di 360°, chiamati **continuous rotation servo**, ma questi solitamente non si possono posizionare a un angolo preciso, piuttosto ruotano con velocità differenti. Ma sono casi particolari: un normale servomotore ruota di 180°, quindi la nostra lancetta dell'orologio si comporterebbe non tanto come un orologio normale, ma piuttosto come la lancetta di un contachilometri nelle automobili. È anche possibile cercare di modificare un servomotore da 180° per fare in modo che riesca a girare di 360°: http://www.instructables.com/id/How-to-modify-a-servo-motor-for-continuous-rotatio/?ALLSTEPS.
Ovviamente, per migliorare il nostro esempio, con un altro servomotore si può fare la stessa cosa per i minuti, ricordando però che sono 60 e non 24, quindi la funzione **map** va adattata di conseguenza.

Il codice sorgente

Potete trovare il codice sorgente dei vari programmi presentati in questo corso di introduzione alla programmazione con Arduino nel repository:
https://www.codice-sorgente.it/cgit/arduino-a-scuola.git/tree/
I file relativi a questa lezione sono, ovviamente, **dissolvenza-led-distanza-ultrasuoni.ino** e **ethernet-servomotore.ino**.