Benvenuti a questo corso pratico di introduzione alla programmazione delle schede Arduino e compatibili. Queste dispense si pongono l'obiettivo di veicolare alcuni concetti base necessari alla programmazione di Arduino tramite l'introduzione di esercizi via via più complessi. Queste dispense non si pongono l'obiettivo di insegnare a programmare in generale, obiettivo ben più ampio e trattato in una miriade di testi più specifici. Le dispense si pongono come una guida per ottenere dei risultati pratici immediati e come una leva che smuova la curiosità del lettore.

Nella maggior parte degli esercizi il lettore si troverà di fronte ad una qualche periferica di output ed una di input, come possono essere LED ed interruttori. Questa architettura permetterà al lettore di interagire con la scheda e sapere che la sua interazione è andata a buon fine. Gli esercizi sono pensati come una catena dove ciascuno di essi è in qualche modo legato al successivo.

Questo schema sarà vero per una prima metà del testo, introduttivo ad alcune caratteristiche base della scheda. In una seconda parte saranno presentati alcuni esercizi più avanzati che faranno uso di algoritmi (programmi) più articolati, librerie di terze parti di cui non vedremo il sorgente e concetti, sia di elettronica che di programmazione, più complessi.

Divertitevi!

# ZERESIMA PARTE

Gli esercizi sono strutturati in questa maniera:

## Numero. Titolo

### Introduzione

Descrizione dell'obiettivo dello sketch per Arduino ed eventualmente differenze rispetto ai precedenti.

### Lista della spesa

- un elenco dell'elettronica necessaria;
- solitamente la scheda Arduino ed il necessario per programmarla non sono elencati.

### Il codice

```
// Tutto il testo tra "Il codice:" e "Qualche link utile:" è il programma
// per la scheda.
```

```
// Tutto il testo tra il doppio slash e la fine della riga è un commento al
// codice e viene ignorato dal compilatore.
// Quando eseguite questo programma di prova, attivate la console
// seriale... ;)

#define LED_PIN 13

// I due seguenti blocchi di codice, identificati dalle parentesi graffe
// aperte e chiuse, sono "funzioni" e sono eseguite dal controller ATMEGA
// della scheda.

// Questo blocco di codice si chiama "setup" e serve a preparare la scheda,
// nel caso fosse necessario.
void setup() {
  Serial.begin(9600);  // Inizializzazione del canale di comunicazione di
                       // tipo "seriale"
  // ...che cominciamo ad utilizzare scrivendo qualche sciocchezza!
  Serial.println("Preparazione della scheda all'accensione del led
      "integrato e collegato al pin 13:");
  Serial.println("3...");
  delay(1000);  // attesa di 1000 millisecondi, ovvero 1 secondo
  pinMode(LED_PIN, OUTPUT);
  Serial.println("2...");
  delay(1000);
  digitalWrite(LED_PIN, LOW);
  Serial.println("1...");
  delay(1000);
}

// Questo blocco di codice si chiama "loop" ed è eseguito ripetutamente
// fino a quando il microcontrollore non viene riprogrammato, resettato
// tramite il preposto pulsante o spento.
void loop() {
  if ( digitalRead(LED_PIN) == HIGH ) {
    delay(500);
    Serial.println("ATMEGA was here [cit.]");
    return;  // questa istruzione permette di terminare l'esecuzione e di
             // ripartire dall'inizio
  }
  Serial.println("0!");
  delay(700);
  Serial.println("Bzzzp...");
  delay(500);
  Serial.println("...blip-blip-blip...");
  delay(250);
  Serial.println("Ftannnngh!");
  delay(100);
  Serial.println("...zzzzzzzzzzz... ...click.");
  delay(50);
  digitalWrite(LED_PIN, HIGH);
  Serial.println("Flash!");
}
```

## Qualche link utile

- https://www.arduino.cc/en/Tutorial/BareMinimum
- un elenco di alcuni link dell'Internet dove poter andare per continuare la propria esperienza.

Notate che non fornisco indicazioni sullo schema elettrico da utilizzare per la costruzione dei circuiti: se lo volete abbozzatene uno e speditemelo, e poi vediamo dove metterlo.

# PRIMA PARTE

## 1. LED lampeggiante

### Introduzione

Con questo esercizio accendiamo e spegniamo un led ad intervalli regolari di 500 millisecondi senza mai fermare l'esecuzione del programma.

Terremo traccia dell'istante temporale in cui il led è stato spento od acceso e faremo in modo di non eseguire quelle istruzioni fino a quando non saranno passati almeno 500 millisecondi. Utilizzeremo una variabile chiamata "cronometro" ed una istruzione di nome "if" che permette, prima di essa, di eseguire un qualsiasi altro comando mentre, dopo di essa, di seguire due percorsi differenti. Se saranno passati i 500 millisecondi, "l'interruttore" del led verrà premuto, altrimenti no.

Avrei potuto mostrarvi la più comoda istruzione "delay(INTERVALLO_IN_MILLISECONDI)", che interrompe l'esecuzione del programma per la quantità di millisecondi specificati tramite INTERVALLO_IN_MILLISECONDI, ma vi sareste trovati con un programma letteralmente "bloccato" in attesa che l'intervallo finisca. Con qualche istruzione in più, ora potete far fare altre cose alla schda, mentre il led continuerà a lampeggiare.

### Lista della spesa

- 1 led di qualsiasi colore (sconsiglio quelli ad alta luminosità, che vi possono accecare parti della retina per qualche minuto);
- 1 resistenza da almeno 100 Ohm.

### Il codice

```
int LED_PIN = 9;  // se usate un pin diverso tra quelli digitali,
                  // aggiornate questo valore
int MEZZO_SECONDO = 500;  // 500 millisecondi sono mezzo secondo

int led_stato = LOW;  // LOW e HIGH sono costanti predefinite della
                      // piattaforma Arduino
unsigned long cronometro = 0;

void setup() {
  Serial.begin(9600);
```

```
  pinMode(LED_PIN, OUTPUT);
  digitalWrite(LED_PIN, LOW);
}

void loop() {
  if ( millis() - cronometro < MEZZO_SECONDO )
    return;

  cronometro = millis();
  led_stato = digitalRead(LED_PIN);

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

  Serial.println(digitalRead(LED_PIN));
}
```

### Qualche link utile

- https://www.arduino.cc/en/Tutorial/Blink
- https://www.arduino.cc/en/Tutorial/BlinkWithoutDelay

# 2. Lampeggìo a comando

### Introduzione

Riprendiamo l'esercizio di prima e aggiungiamo un bottone grazie al quale sospenderemo il lampeggìo del led. Utilizzeremo la funzione della libreria "digitalRead" che chiede al microcontrollore di sapere se su un certo pin è applicata una tensione elettrica. Se la tensione è bassa il valore restituito sarà il medesimo associato alla costante LOW, altrimenti sarà il medesimo della costante HIGH: "digitalRead" restituisce solo questi due possibili valori.

### Lista della spesa

- 1 led e 1 resistenza di almeno 100 Ohm;
- 1 interruttore a scatto;
- 1 resistenza da 10 KOhm.

### Il codice

```
int LED_PIN = 9;
int INTERRUTTORE_PIN = 8;  // se usate un pin diverso tra quelli digitali,
                           // cambiate questo valore numerico
int MEZZO_SECONDO = 500;

int led_stato = LOW;
int interruttore_stato = LOW;
```

```
unsigned long cronometro = 0;

void setup() {
  Serial.begin(9600);
  pinMode(LED_PIN, OUTPUT);
  digitalWrite(LED_PIN, LOW);
  // all'avvio della scheda, i pin sono già in modo INPUT
  // non c'è necessità di impostare lo stato LOW perché è già così in
  // modo INPUT
}

void loop() {
  interruttore_stato = digitalRead(INTERRUTTORE_PIN);
  if ( millis() - cronometro < MEZZO_SECONDO and interruttore_stato == LOW )
    return;

  cronometro = millis();
  led_stato = digitalRead(LED_PIN);

  if ( led_stato == LOW or interruttore_stato == HIGH ) {
    digitalWrite(LED_PIN, HIGH);
  } else {
    digitalWrite(LED_PIN, LOW);
  }

  Serial.println(digitalRead(LED_PIN));
}
```

### Qualche link utile

- https://www.arduino.cc/en/Tutorial/Button

# 3. Variamo la luminosità

### Introduzione

In questo esercizio cominciamo ad utilizzare gli input analogici, chiamati anche ADC (analog-to-digital converter). La funzione principale "loop" eseguirà continuamente alla massima velocità disponibile, molto superiore alla percezione visiva umana, permettendoci di godere dell'effetto ottico di aumentare o diminuire la luminosità di un led istantaneamente.

Un potenziometro, una resistenza variabile, ci permetterà di alterare il livello della tensione elettrica che applicheremo ad uno dei pin analogici del microcontrollore. Il compito di un ADC è di campionare la tensione elettrica (mondo del continuo) in un valore numerico di una scala con un minimo ed un massimo (mondo del discreto). La scala dell'ATMEGA va da 0 a 1023.

Questo valore numerico verrà trasformato in un altro valore numerico appartenente ad una scala più corta, di soli 256 valori: da 0 a 255. Questo numero verrà utilizzato per generare, su di un certo pin, un segnale PWM (pulse-width modulation), "un'onda quadra" che si sposta, moltissime volte in un

secondo, tra due soli possibili valori di tensione elettrica: 0 e 5 volt. La quantità di tempo passata a 0 e poi a 5 volt permette di "simulare" in media una tensione intermedia tra 0 e 5 volt. Il nuovo valore numerico tra 0 e 255 serve ad indicare per quanto tempo, nella singola ripetizione dell'onda (una frazione di secondo, ripetuta tante volte in un secondo), verrà fornita una tensione fissa a 5 volt.

Ci vengono incontro "analogRead" e "analogWrite".

### Lista della spesa

- 1 led e 1 resistenza di almeno 100 Ohm;
- 1 potenziometro da 10 KOhm.

### Il codice

```
int LED_PIN = 9;  // il pin del led deve supportare il segnale PWM
int POTENZIOMETRO_PIN = 0;  // aggiornare questo valore numerico, nel caso
                            // si cambi input analogico

int potenziometro_stato = 0;

void setup() {
  Serial.begin(9600);
  pinMode(LED_PIN, OUTPUT);
  digitalWrite(LED_PIN, LOW);
}

void loop() {
  potenziometro_stato = analogRead(POTENZIOMETRO_PIN);

  analogWrite(LED_PIN, potenziometro_stato / 4);  // il valore tra 0 e 1023
                                                  // passa alla scala tra
                                                  // 0 e 255
}
```

### Qualche link utile

- https://www.arduino.cc/en/Tutorial/ReadAnalogVoltage

## 4. Anche con una fotoresistenza

### Introduzione

Adesso vediamo lo stesso esercizio di prima applicato ad una fotoresistenza. Il principio è analogo, ma in questo caso abbiamo a che fare con un massimo teorico di 1023 difficilmente raggiungibile in condizioni ambientali comuni, come una stanza illuminata da una lampada. Quindi, per ottenere un effetto godibile, trasformerò una scala più piccola con un massimo empirico accettabile di 550 nella consueta tra 0 e 255. Per farlo utilizzerò la comoda funziona "map"

(per altri motivi, è un pessimo nome) che trasforma un valore di una certa scala in un altro di un'altra scala.

### Lista della spesa

- 1 led e 1 resistenza di almeno 100 Ohm;
- 1 fotoresistenza;
- 1 resistenza da 1 KOhm.

### Il codice

```
int LED_PIN = 9;
int FOTORESISTENZA_PIN = 0;

int fotoresistenza_stato = 0;

void setup() {
  Serial.begin(9600);
  pinMode(LED_PIN, OUTPUT);
  digitalWrite(LED_PIN, LOW);
}

void loop() {
  fotoresistenza_stato = analogRead(FOTORESISTENZA_PIN);

  Serial.println(fotoresistenza_stato);

  // massimo teorico di 1023 difficilmente raggiungibile in una stanza,
  // quindi sto a 550
  int luminosita = map(fotoresistenza_stato, 0, 550, 0, 255);
  analogWrite(LED_PIN, luminosita);
}
```

### Qualche link utile

- https://www.arduino.cc/en/Tutorial/ReadAnalogVoltage

## 5. Un po' di suoni

### Introduzione

Quì ci divertiamo un pochino di più, grazie all'emissione di qualche suono. Ri-utilizzando speaker di vecchi computer o buzzer piezoelettrici presenti anche nelle più scrause radiosveglie possiamo permettere ad Arduino di emettere anche della musica. Ovviamente non sarà di grande qualità, ma per dei progetti o dei giochi molto semplici, potrebbe bastare.

Per ottenere questo effetto la libreria è fornita di una comoda funzione "tone" alla quale basterà passare il pin a cui lo strumento di output è collegato, la frequenza a cui vogliamo farlo vibrare e la durata della vibrazione. Attenzione:

questa funzione è bloccante e quindi il microcontrollore non eseguirà altre istruzione fino a quando non avrà terminato l'esecuzione di questa funzione.

### Lista della spesa

- 1 speaker o buzzer piezoelettrico;
- 1 fotoresistenza e 1 resistenza da 1 KOhm.

### Il codice

```
int SPEAKER_PIN = 8;  // puo anche non supportare il PWM
int FOTORESISTENZA_PIN = 0;

int fotoresistenza_stato = 0;

void setup() {
  Serial.begin(9600);
  pinMode(SPEAKER_PIN, OUTPUT);
  digitalWrite(SPEAKER_PIN, LOW);
}

void loop() {
  fotoresistenza_stato = analogRead(FOTORESISTENZA_PIN);

  // nella prossima istruzione ci poniamo tra i 120 Hz ed i 1700 Hz
  int frequenza = map(fotoresistenza_stato, 0, 1023, 120, 1700);
  tone(SPEAKER_PIN, frequenza, 5);
}
```

### Qualche link utile

- https://www.arduino.cc/en/Tutorial/toneMelody

## 6. Input selezionabile

### Introduzione

Questo progetto riassume un po' tutto quello che abbiamo visto fin'ora: un pulsante che permette di decidere con quale periferica di input, potenziometro o fotoresistenza, controllare lo speaker. Due led indicheranno l'input attivo in quel momento. Quì introduciamo la tecnica del debouncing, che permette di stabilizzare il segnale proveniente dal pulsante. Si tratta di attendere un po' di tempo prima di uscire dal blocco di istruzioni associato alla pressione del pulsante. In questa maniera tutte le piccole variazioni elettriche dovute al termine della pressione del pulsante da parte dell'utente, per esempio, vengono eliminate e non si rischia di ri-entrare troppo presto nel blocco di codice.

### Lista della spesa

- 2 led e 2 resistenze da almeno 100 Ohm;
- 1 speaker o buzzer piezoelettrico;
- 1 fotoresistenza e 1 resistenza da 1 KOhm;
- 1 potenziometro da 10 KOhm;
- 1 interruttore a scatto e 1 resistenza da 10 KOhm;
- 1 breadboard e almeno una decina di cavetti doppio maschio.

## Il codice

```
int SPEAKER_PIN = 9;
int POTENZIOMETRO_PIN = 0;
int FOTORESISTENZA_PIN = 1;
int INTERRUTTORE_PIN = 2;
int LED_POT_PIN = 3;
int LED_FOTO_PIN = 4;

int POT_COME_INPUT = 0;
int FOTO_COME_INPUT = 1;

int input_selezionato = 0;
int int_stato_precedente = LOW;

void setup() {
  Serial.begin(9600);
  pinMode(LED_POT_PIN, OUTPUT);
  digitalWrite(LED_POT_PIN, HIGH);
  pinMode(LED_FOTO_PIN, OUTPUT);
  digitalWrite(LED_FOTO_PIN, LOW);
  pinMode(SPEAKER_PIN, OUTPUT);
  digitalWrite(SPEAKER_PIN, LOW);
}

void loop() {
  int int_stato = digitalRead(INTERRUTTORE_PIN);
  if ( int_stato != int_stato_precedente ) {
    if ( int_stato == HIGH ) {
      if ( input_selezionato == POT_COME_INPUT ) {
        input_selezionato = FOTO_COME_INPUT;
        digitalWrite(LED_POT_PIN, LOW);
        digitalWrite(LED_FOTO_PIN, HIGH);
      } else {
        input_selezionato = POT_COME_INPUT;
        digitalWrite(LED_POT_PIN, HIGH);
        digitalWrite(LED_FOTO_PIN, LOW);
      }
    }
    int_stato_precedente = int_stato;
    delay(50);  // tecnica per evitare il debouncing e stabilizzare la
                // percezione della pressione dell'interruttore
  }

  int frequenza = 0;
  if ( input_selezionato == POT_COME_INPUT ) {
    frequenza = analogRead(POTENZIOMETRO_PIN);
  } else {
    frequenza = map(analogRead(FOTORESISTENZA_PIN), 0, 1023, 120, 1700);
  }
```

```
  tone(SPEAKER_PIN, frequenza, 5);
}
```

### Qualche link utile

- https://www.arduino.cc/en/Tutorial/Debounce
- https://www.arduino.cc/en/Tutorial/StateChangeDetection

# SECONDA PARTE

## A. Se t'avvicini, mi gira!

### Introduzione

In questo progetto, in cui controlleremo la velocità di un elica attaccata ad un motore elettrico, vedremo due chip molto interessanti: l'L293D e l'SN74HC.

Il primo permette di controllare correnti elettriche e tensioni molto maggiori a quelli che possono uscire dai pin dell'ATMEGA e solitamente è utilizzato per controllare motori elettrici come quelli delle mini 4WD. Si applica la tensione d'uscita desiderata ad un certo pin, che puo essere appunto tranquillamente superiore ai 5 volt dell'Arduino, e questa sarà veicolata verso altri quattro pin d'uscita. Questi saranno conessi ai contatti dei motori. Altri pin del chip, in contatto coi pin del microcontrollore dell'Arduino, diverranno i pulsanti di accensione e spegnimento di questi circuiti separati. I pin dell'ATMEGA possono essere anche quelli col segnale PWM, cosa che faremo in questo esercizio. Un altro vantaggio di questo chip è di proteggere l'ATMEGA dai disturbi elettrici di ritorno dei motori elettrici, che potrebbero danneggiarlo.

L'altro, l'SN74HC, è un multiplexer: i pin d'uscita controllabili singolarmente nei due stati LOW e HIGH sono maggiori di quelli richiesti per controllarli. A fronte di tre pin connessi in entrata dall'ATMEGA, possiamo controllare indipendentemente ben otto pin in uscita dal chip nei due stati di tensione presente ed assente.

### Lista della spesa

- 1 sensore ultrasonico di distanza SEN136B5B;
- 1 L293D;
- 4 pile stilo AA;
- 1 portapile per 4 stilo AA;
- 1 motore da mini 4WD;
- 1 elica che si incastri sull'asse del motore;
- 1 SN74HC;

- 8 led e 8 resistenze da almeno 100 Ohm;
- 1 breadboard e vari cavi doppio maschio.

## Il codice

```
#define DISTANZA_TRIG_PIN 7
#define DISTANZA_ECHO_PIN 6

#define CAMPIONI_DISTANZE_PRECEDENTI 5
float distanze_precedenti[CAMPIONI_DISTANZE_PRECEDENTI];
int indice_distanze_precedenti = 0;
float ultima_distanza_rilevata = 0;

#define ACCEL_DATA_PIN 11  // Pin connected to DS (SER) of 74HC595 (pin 14)
#define ACCEL_LATCH_PIN 8  // Pin connected to ST_CP (RCLK) of 74HC595
                           // (pin 12)
#define ACCEL_CLOCK_PIN 12  // Pin connected to SH_CP (SRCLK) of 74HC595
                            // (pin 11)

unsigned long accel_ultimo_aggiornamento = 0;
int accel_gradi[] = {
  1+2+4+8,
  2+4+8,
  2+4+8,
  4+8,
  8,
  0,
  0,
  0,
  16,
  16+32,
  16+32+64,
  16+32+64,
  16+32+64+128
};
int accel_grado = 0;

#define MOTORE_PIN 5

void setup() {
  Serial.begin (9600);

  pinMode(DISTANZA_TRIG_PIN, OUTPUT);
  pinMode(DISTANZA_ECHO_PIN, INPUT);

  pinMode(ACCEL_LATCH_PIN, OUTPUT);
  pinMode(ACCEL_CLOCK_PIN, OUTPUT);
  pinMode(ACCEL_DATA_PIN, OUTPUT);

  pinMode(MOTORE_PIN, OUTPUT);
}

void loop() {
  float distanza = campiona_distanza();
  float accelerazione = ultima_distanza_rilevata - distanza;

  if ( millis() - accel_ultimo_aggiornamento > 10 ) {
    accel_ultimo_aggiornamento = millis();
```

```
    int accel_nuovo_grado = map(accelerazione * 100, -3 * 100, 3 * 100,
        -6, 6);
    if ( accel_grado > 0 && ( accel_nuovo_grado > accel_grado ||
                                                accel_nuovo_grado < 0 ) ) {
      accel_grado = accel_nuovo_grado;
    } else if ( accel_grado < 0 && ( accel_nuovo_grado < accel_grado ||
                                                accel_nuovo_grado > 0 ) ) {
      accel_grado = accel_nuovo_grado;
    } else if ( accel_grado == 0 ) {
      accel_grado = accel_nuovo_grado;
    }

    digitalWrite(ACCEL_LATCH_PIN, LOW);
    shiftOut(ACCEL_DATA_PIN, ACCEL_CLOCK_PIN, MSBFIRST,
        accel_gradi[accel_grado + 6]);
    digitalWrite(ACCEL_LATCH_PIN, HIGH);

    if ( accel_grado > 0 ) accel_grado--;
    if ( accel_grado < 0 ) accel_grado++;
  }

  /*Serial.print(distanza);
  Serial.print(" cm\t");
  Serial.print(accelerazione);
  Serial.println();*/

  int motore_velocita = map(distanza, 0, 120, 0, 255);
  analogWrite(MOTORE_PIN, 255 - motore_velocita);

  ultima_distanza_rilevata = distanza;
  delay(1);
}

float campiona_distanza() {
  if ( indice_distanze_precedenti >= CAMPIONI_DISTANZE_PRECEDENTI ) {
    indice_distanze_precedenti = 0;
  }
  float distanza = misura_distanza();
  if ( distanza != 0 ) {
    distanze_precedenti[indice_distanze_precedenti] = distanza;
    indice_distanze_precedenti++;
  }
  float distanza_media = 0;
  for ( int i = 0 ; i < CAMPIONI_DISTANZE_PRECEDENTI ; i++ ) {
    distanza_media += distanze_precedenti[i];
  }
  distanza_media = distanza_media / CAMPIONI_DISTANZE_PRECEDENTI;
  return distanza_media;
}

float misura_distanza() {
  long durata;
  float distanza;
  digitalWrite(DISTANZA_TRIG_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(DISTANZA_TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(DISTANZA_TRIG_PIN, LOW);
  durata = pulseIn(DISTANZA_ECHO_PIN, HIGH, 5000);
```

```
  distanza = ( durata / 2 ) / 29.1;
  return distanza;
}
```

### Qualche link utile

- http://arduinobasics.blogspot.it/2012/11/arduinobasics-hc-sr04-ultrasonic-sensor.html

# B. Controllo ambientale

## Introduzione

In questo esercizio vedremo come mostrare su di un display grafico monocolore (in realtà sono due, lo sfondo ed il nero...) dei numeri e del testo, in particolare dei valori con parte decimale e alcuni caratteri di contorno.

Il display, che potete trovare in varie fogge da Olimex o da AdaFruit, è quello che solitamente si poteva trovare su di un cellulare della Nokia anni fa, il modello 3110. E' composto da 84 colonne e 48 righe, controllate da sequenze "verticali" di 6 byte per 84 colonne. Ovvero, i 48 punti delle righe sono raccolti in sequenze da 8 (6 x 8 = 48) e per ciascun punto si puo indicare la presenza del colore o la sua assenza (ovvero la presenza del colore dello sfondo). Per questo ragioniamo facilmente in termini di byte (otto bit), di 1 e di 0, e quindi di sequenze di 6 byte per ciascuna colonna, per un totale di 8 byte x 6 righe x 84 colonne = 4032 punti che nella memoria del microcontrollore diventano 504 byte.

Tutta questa complessità ci verrà nascosta da una libreria abbastanza facile da utilizzare e che includo in qualche modo con queste dispense: OxLCD. Questa libreria è in realtà un wrapper, una libreria per un'altra libreria, che ho scritto per un problema che ho individuato con la libreria sottostante e che non sono riuscito a correggere.

L'altro elemento che vedremo è il sensore di temperatura ed umidità DHT11, quello economico della sua famiglia. Tanto economico. Troppo economico. Meglio provare il DHT22. Un sensore che aggiunge, ma per via della libreria che calcola delle medie, anche un rallentamento nell'esecuzione del codice.

In questo programma il pulsante serve per mostrare i dati sul display e poi per nasconderli quando non sono più necessari, altrimenti verranno cancellati dopo 5 secondi.

## Lista della spesa

- 1 display grafico monocolore stile Nokia3110;
- 1 DHT11 come sensore di temperatura e umidità;

- 1 fotoresistenza e 1 resistenza da 1 KOhm;
- 1 interruttore a scatto e 1 resistenza da 10 KOhm.

## Il codice

```
#define INTERRUTTORE_PIN 8
#define DURATA_VISUALIZZAZIONE 5000

int int_stato_precedente = LOW;
int dati_visualizzati = LOW;
unsigned long ultima_visualizzazione_dati = 0;

#define FOTO_PIN 0
#define FOTO_RESISTOR 10.0

#include <DHT.h>

#define DHT_PIN 9  // non necessita di essere analogico od un pwm

// decommentate la define del modello che state usando
#define DHT_TIPO DHT11   // DHT 11
// #define DHT_TIPO DHT22   // DHT 22  (AM2302)
// #define DHT_TIPO DHT21   // DHT 21 (AM2301)

DHT dht(DHT_PIN, DHT_TIPO);

#include "OxLCD.h"

// Reset                          #LCD RES
// Data/Command                   LCD D/#C
// Data input (master output)     MOSI
// Clock                          SCK
// Chip enable                    #CS
OxLCD oxlcd(2, 3, 5, 6, 7);

void setup() {
  Serial.begin(9600);
  dht.begin();
  mostra_testo("Sistema", "attivo", ";)");
  delay(1000);
  mostra_testo("", "", "");
}

void loop() {
  int luminosita = analogRead(FOTO_PIN);
  float lumen = 255.84 * pow(
    ( FOTO_RESISTOR * 1024.0 ) / ( 1024 - luminosita ) - FOTO_RESISTOR,
    -10.0 / 9.0
  );
  float umidita = dht.readHumidity();
  float temperatura = dht.readTemperature();

  int int_stato = digitalRead(INTERRUTTORE_PIN);
  if ( int_stato != int_stato_precedente ) {
    if ( int_stato == HIGH and dati_visualizzati == LOW ) {
      mostra_dati(temperatura, umidita, lumen);
      ultima_visualizzazione_dati = millis();
      dati_visualizzati = HIGH;
```

```
    } else if ( int_stato == HIGH and dati_visualizzati != LOW ) {
      ultima_visualizzazione_dati = 0;
    }
    int_stato_precedente = int_stato;
    delay(50);
  }

  if ( dati_visualizzati != LOW and
      ( millis() - ultima_visualizzazione_dati ) > DURATA_VISUALIZZAZIONE )
  {
    Serial.println("Clear");
    mostra_testo("", "", "");
    dati_visualizzati = LOW;
  }
}

void mostra_dati(float temperatura, float umidita, float lumen) {
  char temp[6];  // variabile di supporto per le trasformazioni
  // preparo le "stringhe" vuote in cui copiare i valori numerici
  char u[] = "      %";
  char t[] = "      '";
  char l[] = "      lux";
  // trasformo dei valori numerici con i decimali in sequenze di caratteri
  // copiandoli nelle "stringhe" dopo la conversione
  dtostrf(umidita, 6, 0, temp);
  strncpy(u, temp, 6);
  dtostrf(temperatura, 6, 0, temp);
  strncpy(t, temp, 6);
  dtostrf(lumen, 6, 0, temp);
  strncpy(l, temp, 6);

  Serial.println(u);
  Serial.println(t);
  Serial.println(l);

  mostra_testo(u, t, l);
}

void mostra_testo(char riga1[], char riga2[], char riga3[]) {
  oxlcd.clearBuffer();  // ripulisce, ponendo tutti i valori a 0, un'area
                        // di memoria chiamata "buffer"
  sfondo_display();  // disegna qualcosa nel buffer
  oxlcd.drawString(riga1, 8, 10);
  oxlcd.drawString(riga2, 8, 20);
  oxlcd.drawString(riga3, 8, 30);
  oxlcd.sendBuffer();  // invia il buffer cos'ì com'è al display
}

void sfondo_display() {
  oxlcd.drawRect(0, 0, 83, 47, 0x00);
}
```

### Qualche link utile

- http://forum.arduino.cc/index.php?topic=161313.0

# C. Oh toh, i servomotori sul web!

## Introduzione

In questo esercizio vedremo come utilizzare i servomotori, una tipologia di motori elettrici, e la comunicazione seriale in ingresso alla scheda. In aggiunta, sotto forma di archivio compresso, è presente un'interfaccia web di controllo dei due motori, che saranno indicati come due "ali" (per il modo con cui ve li presenterò).

I servomotori verranno resi accessibili tramite una libreria solitamente presente con l'IDE di Arduino, richiamata tramite l'inclusione di "Servo.h". Ad essa si indicheranno i gradi a cui vorremo che l'asse centrale del servomotore si trovi rispetto ad una sua posizione iniziale. Rispetto ad altri tipi di motori elettrici, come quelli per mini 4WD, il servomotore mantiene la posizione che gli viene assegnata opponendo una discreta resistenza, finché sotto tensione, alla rotazione forzata del suo perno. Per poterli utilizzare servirà un'alimentazione a parte, visto che richiedono una quantità di Ampere maggiori rispetto a quelli che puo fornire l'Atmega.

La comunicazione attraverso la porta seriale è facile quando è Arduino a dover parlare con il pc, dato che la scheda invia il suo messaggio carattere per carattere, lettera per lettera. Quindi altra circuiteria sul computer ed il giusto software si preoccuperanno di memorizzare temporaneamente quanto inviato e poi di riassemblarlo per l'utente. Il contrario, un messagio che dal computer arriva alla scheda, richiede un po' più di impegno, visto che il codice per l'assemblaggio del messaggio è nelle nostre mani. Così vedremo come si leggono i messaggi sulla porta seriale e come si possono riutilizzare.

Questo progetto, come detto, presenta una parte aggiuntiva: un server web in linguaggio Python che serve una sola pagina affinché l'utente possa controllare i motori. Una volta che l'utente interagisce con le "ali" il server web scrive in un file temporaneo l'operazione scelta. Questo viene letto ogni secondo da un altro programma in Python che funge da ponte vero e proprio tra il server web e la scheda. Sarà esso, infatti, a leggere questo file temporaneo per decidere cosa inviare alla scheda, ovvero una sequenza di coppie di gradi sessagesimali.

## Lista della spesa

- 2 servomotori;
- un sistema per alimentarli a parte perché non basta la corrente dell'Arduino.

## Il codice

```
#include <Servo.h>

const int NUMBER_OF_SERVOS = 2;
const int SERIAL_BUFFER_SIZE = 9;
```

```
int horizServoPin = 2;
int vertServoPin = 3;
Servo horizServo;
Servo vertServo;

int destination[] = {90, 90};
double actualPosition[] = {90, 90};

void setup()  {
  horizServo.attach(horizServoPin);
  vertServo.attach(vertServoPin);
  setServosPosition(
    destination[0], destination[1]
  );
  resetDestination();
  Serial.begin(9600);
}

void loop() {
  if ( Serial.available() > 0 ) readDegrees();
  if ( !resettedDestination() ) {
    double diffHoriz = (double) destination[0]-actualPosition[0];
    double diffVert = (double) destination[1]-actualPosition[1];
    double radius = sqrt(pow(diffHoriz, 2)+pow(diffVert, 2));
    double incrHoriz = diffHoriz/radius;
    double incrVert = diffVert/radius;
    actualPosition[0] += incrHoriz;
    actualPosition[1] += incrVert;
    setServosPosition(
      (int) actualPosition[0], (int) actualPosition[1]
    );
    delay(5);
    if ( destinationReached() )
      resetDestination();
  }
}

void readDegrees() {
  initializeDestination();
  int selectedServo = 0;
  char buffer[SERIAL_BUFFER_SIZE];
  Serial.readBytes(buffer, SERIAL_BUFFER_SIZE);
  int val;
  int state = 0;
  for ( int i = 0 ; i < SERIAL_BUFFER_SIZE ; i++ ) {
    val = buffer[i]-48;
    if ( ( state == 0 || state == 1 ) && val >= 0 && val <= 9 ) {
      destination[selectedServo] = destination[selectedServo]*10+val;
      state = 1;
    } else if ( state == 1 && val == -4 ) {
      selectedServo++;
      state = 2;
    } else if ( ( state == 2 || state == 3 ) && val >= 0 && val <= 9 ) {
      destination[selectedServo] = destination[selectedServo]*10+val;
      state = 3;
    } else if ( state == 3 && val == -35 ) {
      state = 4;
      Serial.println("Received");
      break;
    } else {
```

```
      Serial.print("Error (");
      Serial.print(state);
      Serial.print(", ");
      Serial.print(buffer);
      Serial.println(")");
      resetDestination();
      break;
    }
  }
  if ( state != 4 ) resetDestination();
}

void resetDestination() {
  for ( int i = 0 ; i < NUMBER_OF_SERVOS ; i++ )
    destination[i] = -1;
}

void initializeDestination() {
  for ( int i = 0 ; i < NUMBER_OF_SERVOS ; i++ )
    destination[i] = 0;
}

boolean resettedDestination() {
  return destination[0] == -1 && destination[1] == -1;
}

void setServosPosition(int degHoriz, int degVert) {
  horizServo.write(degHoriz);
  vertServo.write(degVert);
}

boolean destinationReached() {
  double destHoriz = (double) destination[0];
  double destVert = (double) destination[1];
  return destHoriz-0.5 <= actualPosition[0] &&
         destHoriz+0.5 >= actualPosition[0] &&
         destVert-0.5 <= actualPosition[1] &&
         destVert+0.5 >= actualPosition[1];
}

// Il resto del codice è in un archivio allegato a queste dispense.

// Si tratta del piccolo server web in Python che fornisce all'utente le
// pagine con i comandi necessari a muovere i servomotori.
wings_commander_http.py

// E si tratta del "demone" da mantenere in esecuzion in background e che
// riceve i comandi dall'utente.
wings_commander_device.py
```

### Qualche link utile

- non me ne sono venuti in mente

# Note sparse

Potrebbe essere che non tutte le variabili abbiano il tipo giusto, tipo delle `int` al posto di `unsigned long`. Sarebbero da controllare e correggere.

Sì, mancano gli schemi: ci sarà da ringraziare se ci saranno le foto delle breadboard!

Dato che generalmente ho fatto in modo di avere dei progetti che non necessitano del computer dal quale li si programma, li si potrebbe rendere indipendenti anche energeticamente con una pila da 9 volt e un qualche adattatore che si attacchi o al jack od ai pin GND e Vin.