Realizzazione di un anemometro a coppe, portatile, basato su Arduino

Federico Davanteri
2014

Materiali e componenti utilizzati

Di seguito l'elenco del materiale utilizzato per il progetto:

- Scheda Arduino UNO R3
- 2 resistori da 10 KOhm
- 11 resistori da 330 Ohm
- Sensore di vento FuturaNet 6710-WIND02
- Display LED 4 digit LuckyLight KW4-56NCLB-P (a catodo comune)
- 1 interruttore
- 3 LED verdi
- 1 pulsante NA (normalmente aperto)
- 1 portabatterie per alimentazione
- 1 presa da 5 mm e relativo jack
- Cavi, connettori e basetta millefori
- Materiali vari per la realizzazione dell'involucro e dell'impugnatura del sensore

Sensore di vento

Il sensore è l'articolo 6710-WIND02 prodotto da http://www.futuranet.it/ (prezzo di circa 27 Euro).

Il funzionamento è basato su un interruttore magnetico "reed" che viene chiuso al passaggio di un piccolo magnete montato sulla parte rotante dello strumento.
Il sensore ha quindi un cavo bipolare, da alimentare a 5V per ottenere un impulso ogni volta che il magnete compie un giro.

La meccanica risulta di buona qualità, mentre ho dovuto correggere un problema legato all'interruttore, che era stato montato in posizione errata. In pratica veniva generato un doppio impulso ravvicinato, che cambiava anche in funzione del senso di rotazione. Ciò rendeva impossibile qualsiasi tipo di lettura affidabile.
Quindi ho riposizionato il reed ottenendo un unico impulso per giro del tutto privo di ulteriori interferenze.

Le specifiche dichiarate sono: 4 impulsi al secondo = 10 Km/h di vento

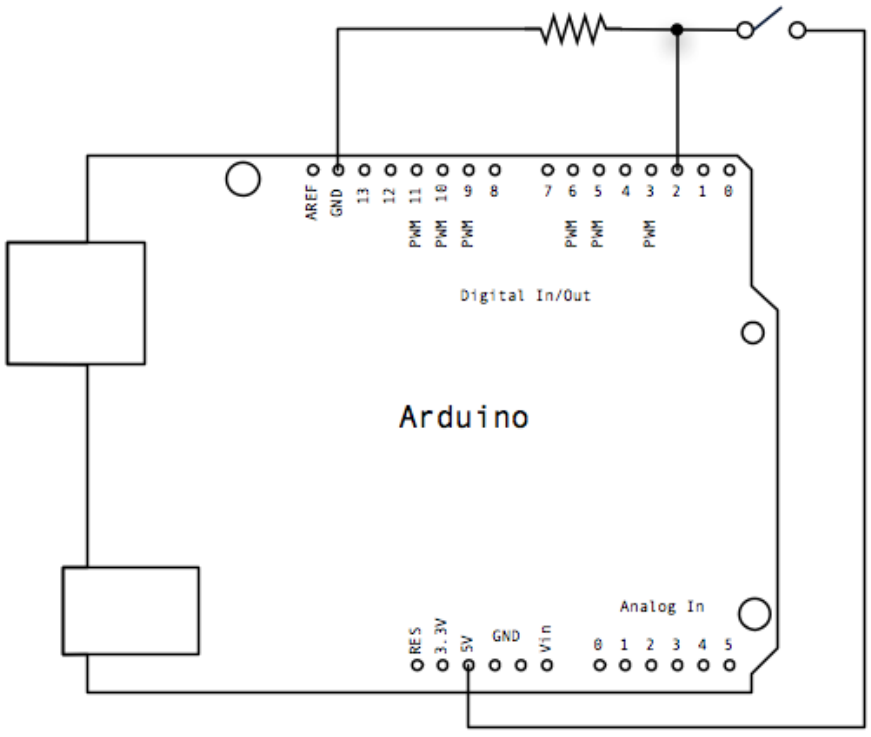
NOTA: Gli anemometri a coppe, come quello da me utilizzato, vengono calibrati in galleria del vento. La funzione che permette di calcolare la velocità è: V = A * f + B dove A e B sono due parametri, determinati sperimentalmente, che dipendono dalla geometria dello strumento.

Circuito per il sensore di vento

L'anemometro è assimilabile (e di fatto lo è) ad un semplice interruttore che si chiude ad ogni giro. Pertanto collegando un conduttore all'uscita +5V di Arduino, riceveremo un impulso di +5V / giro che leggeremo su un pin digitale settato in modalità INPUT.
L'altro conduttore proveniente dall'anemometro andrà collegato al GND di Arduino.

Bisogna però notare che quando l'interruttore è aperto, il pin digitale è collegato direttamente al GND, e se andiamo a leggere il segnale osserviamo che lo stato varia da HIGH a LOW e viceversa in modo del tutto imprevedibile, rendendo impossibile qualsiasi lettura attendibile.
Per evitare questo inconveniente si interpone una resistenza da 10 KOhm, detta di pull-down, che forza a zero il livello del pin quando non arrivano impulsi dall'anemometro.

Schema di collegamento del sensore
(il sensore è schematizzato con l'interruttore in alto a destra)



Software

Determinazione della velocità del vento

Le specifiche dello strumento dicono che 4 impulsi/s corrispondono a 10 Km/h di vento. Dobbiamo perciò calcolare il numero di impulsi al secondo, da rapportare ai 4 di riferimento, per ottenere la velocità che stiamo cercando.
Il programma deve quindi determinare il tempo che intercorre tra due impulsi successivi, che chiameremo "durata". Ottenuta la durata calcoliamo il numero di impulsi al secondo.

$$(1)\ impulsi = \frac{1000}{durata\ (s)}$$

Da questi calcoliamo facilmente la velocità usando una proporzione:

$$(2)\ Velocità\ \left(\frac{Km}{h}\right) = impulsi * \frac{10}{4}$$

Quindi sostituendo la (1) nella (2) otteniamo:

$$(3)\ Velocità\ \left(\frac{Km}{h}\right) = \frac{1000}{durata(s)} * \frac{10}{4} = \frac{2500}{durata(s)}$$

Per scrivere il programma ho dovuto prima comprendere bene il comportamento del sensore facendo alcune prove di lettura preliminari. Essendo dotato di un interruttore reed magnetico, questo rimane chiuso finché si trova sotto l'influenza del magnete. Pertanto l'impulso prodotto rimane in stato HIGH mentre il sensore copre un certo arco di cerchio.

In pratica, visualizzando il valore del pin su tutto il giro si ottiene un output di questo tipo:

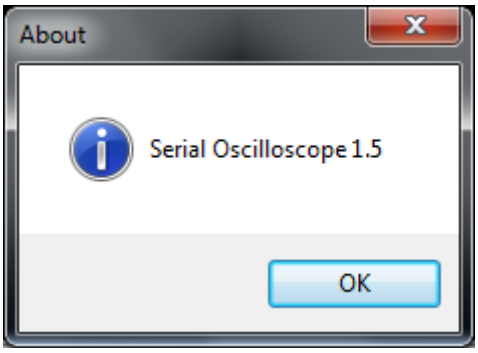
00000000000000000001111111111111111111000000000000000000000000000000

Quindi per la lettura dell'impulso ho scritto il software in modo da individuare il punto indicato con "**+**", ossia ho stabilito di prendere come punto di riferimento la transizione da LOW a HIGH (fronte di salita dell'onda).

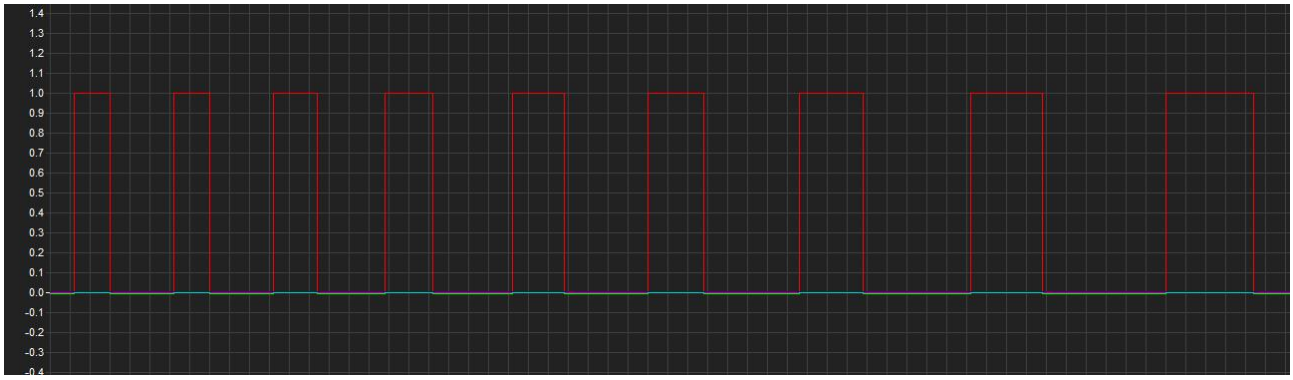
000000000000000000**+**11111111111111111110000000000000000000000000000

L'immagine sotto mostra graficamente le pulsazioni provenienti dal sensore visualizzate tramite un oscilloscopio software: Serial Oscilloscope 1.5.

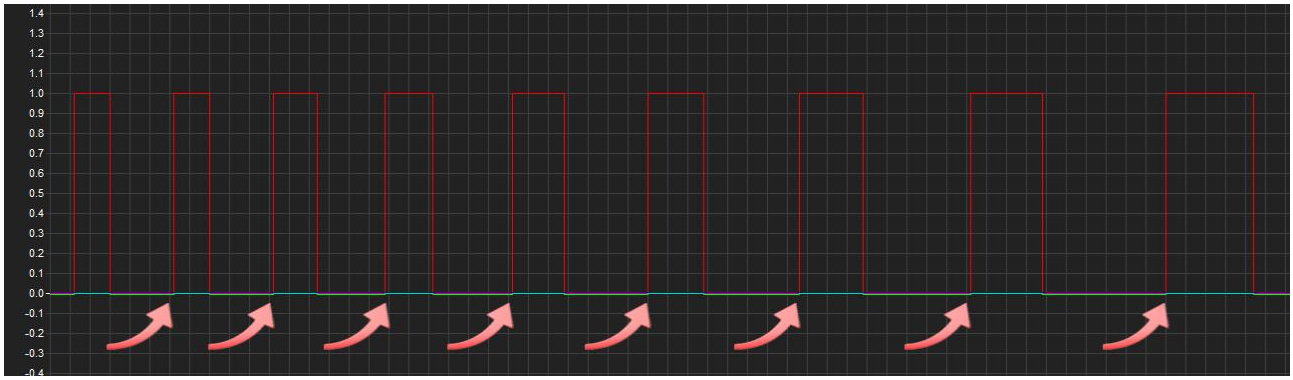
Il programma è scaricabile gratuitamente dal sito http://www.x-io.co.uk/serial-oscilloscope/ e sfrutta le potenzialità derivanti dall'utilizzo di Processing come ambiente di sviluppo integrabile con Arduino, dal quale riceve un flusso seriale di dati che rappresenta graficamente.



Si vede bene l'allungamento della durata delle pulsazioni mentre il sensore rallenta la sua rotazione.



I punti indicati dalle frecce sono quelli di riferimento per il calcolo della velocità, come spiegato in precedenza.



L'algoritmo per il calcolo è piuttosto semplice. Esso si basa essenzialmente su due variabili:

"stato" memorizza lo stato corrente in ogni istante
"pinval" memorizza il valore letto dal sensore in ogni istante

In pratica ad ogni loop del programma "stato" viene posto unguale al valore letto "pinval", ma solo dopo aver verificato la condizione posta dall'istruzione "if".
In questo modo è possibile rilevare il punto esatto in cui avviene il cambiamento della lettura (punto di transizione).
Nel punto di transizione ("stato" uguale a zero e "pinval" uguale a uno), inizia il calcolo del tempo trascorso dalla pulsazione precedente e il conseguente calcolo della velocità.

Il codice che segue fa quello che ho appena descritto

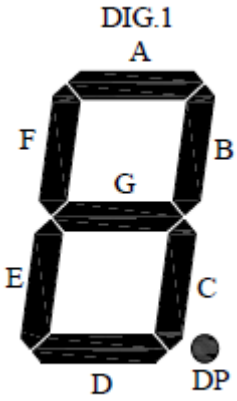
```
void loop ()
{
   pinval = digitalRead(getpulsepin);   // legge il pin del sensore

   if ((stato == 0) & (pinval == 1)) {    // punto di transizione
   durata = millis() - startime;          // calcola la durata della pulsazione
   startime = millis();                   // setta il nuovo tempo di partenza
   windspeed = 2500.0/durata;             // calcola la velocità in Km/h
   }

   stato = pinval;                        // imposta lo stato uguale alla lettura del pin
}
```

Gestione del display

Premetto che esistono vari modi di gestire un display di questo tipo. In commercio si trovamo moduli a 7 segmenti già dotati dell'elettronica necessaria per gestirli in modo semplice e veloce. Inoltre si possono utilizzare circuti intergati denominati "shift register" che permettono di utilizzare un numero contenuto di ping digitali di Arduino.
Io ho preso un display senza elettronica di gestione ed ho implementato tutta la circuiteria e il software necessari per farlo funzionare.
E' stato molto istruttivo, ma ho dovuto utilizzare praticamente tutti i ping digitali e analogici della scheda e scrivere il software di "multiplexing" per la visualizzazione delle cifre.

Il display utilizzato è del tipo detto a "sette segmenti", dove ogni cifra è composta da una combinazione diversa di segmenti accesi o spenti. Ciascun segmento é un LED, quindi l'intero display è formato da 4x7 = 28 LED + 4 punti decimali e un doppio punto centrale. In totale 34 LED.

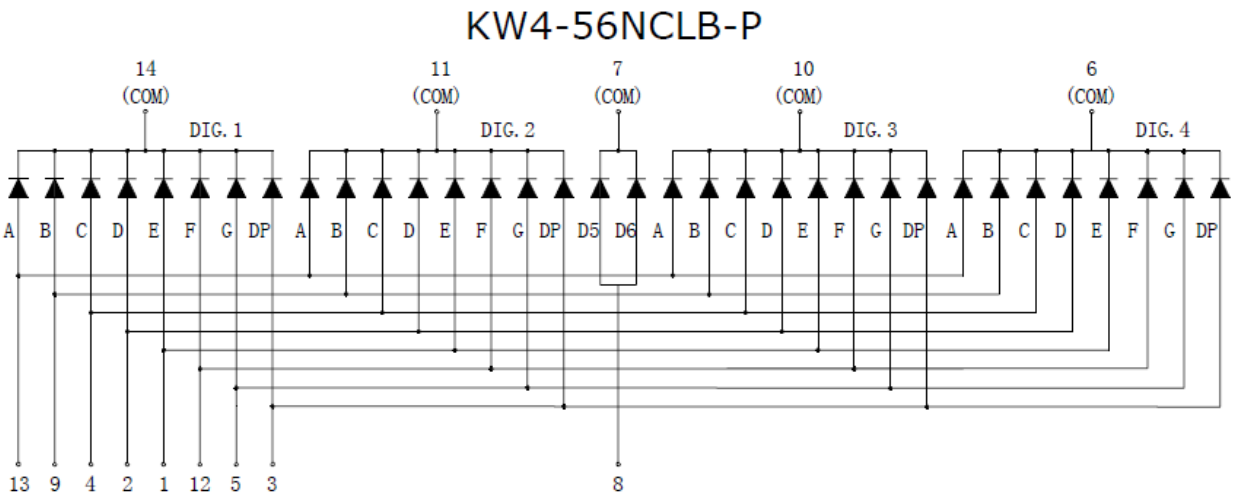
I LED (segmenti) che costituiscono la cifra "DIG.x" sono identificati da una lettera da "A" a "G", come mostrato qui sotto a sinistra. IL punto è "DP", mentre i due punti centrali sono "D5" e "D6".



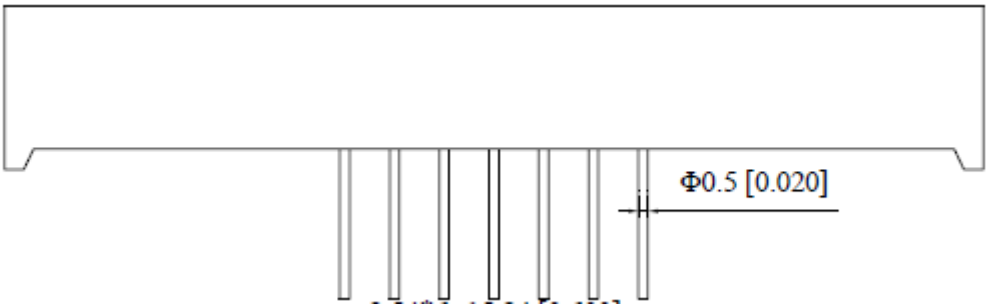
Nel caso specifico del display utilizzato da me, gli anodi dei LED (poli positivi) sono separati e raggruppati per segmento (es: tutti i segmenti "F" condividono lo stesso anodo), mentre i catodi (poli negativi) sono raggruppati per cifra. Da qui la definizione di display a catodo comune.
Per poter accendere e spegnere i led abbiamo quindi bisogno di 14 pin, ma io ne ho utilizzati 12 in quanto il doppio punto non mi serviva.

Nota: i pin digitali 0 e 1 sono utilizzati per la comunicazione seriale, pertanto se vengono impegnati per la gestione di altri dispositivi o sensori, essi vengono inibiti dall'attivazione della comunicazione seriale con il computer (Serial.begin…).

Nella figura seguente si vede lo schema completo preso da datasheet del display.



Anodi e catodi corrispondono ai piedini del display (immagine sotto), e sono numerati da 1 a 14 (da 1 a 7 sul lato anteriore e da 8 a 14 su quello posteriore). Quindi vediamo (immagine sopra) che gli anodi corrispondono ai piedini 13, 9, 4, 2, 1, 12, 5, 3 e 8 (la posizione disordinata deriva dalla geometria interna dei collegamenti), mentre i catodi sono i piedini 6, 10, 7, 11, e 14.
Il piedino 13 collega quindi tutti i segmenti “A” delle quattro cifre, il 9 i segmenti “B” e così via.



Dallo schema appare chiaro che collegando il display e inviando gli opportuni segnali a ciascun pin, tutte le cifre mostreranno lo stesso numero, dato che ogni pin alimenta lo stesso segmento di ogni cifra. Come fare quindi per mostrare cifre diverse ?
Qui entra in gioco il software. Con il meccanismo di multiplexing, cui ho già accennato, si riesce a mostrare cifre diverse pur inviando lo stesso segnale a tutte le cifre.

Vediamo come funziona.
La funzione Loop() del programma richiama continuativamente una funzione, che ho chiamato WriteDig() la quale riceve come parametri la cifra da rappresentare, e su quale delle quattro indirizzarla. Facciamo un esempio pratico. Supponiamo di voler rappresentare le cifre 5, 6, 7 e 8.

```
void loop ()
{
WriteDig(5,1);  // cifra 5 sulla prima posizione
WriteDig(6,2);  // cifra 6 sulla seconda posizione
WriteDig(7,3);  // cifra 7 sulla terza posizione
WriteDig(8,4);  // cifra 8 sulla quarta posizione
}
```

La chiamata della funzione avviene quattro volte per ogni loop, passando di volta in volta le quattro cifre indirizzate nelle quattro posizioni diverse del display.

La funzione WriteDig() quindi, con quattro chiamate successive esegue quanto segue:

1) accende la prima cifra (le altre spente) e scrive 5. Attende 1/50 sec e spegne tutto.
2) accende la seconda cifra (le altre spente) e scrive 6. Attende 1/50 sec e spegne tutto.
3) accende la terza cifra (le altre spente) e scrive 7. Attende 1/50 sec e spegne tutto.
4) accende la quarta cifra (le altre spente) e scrive 8. Attende 1/50 sec e spegne tutto.

Dato che l'occhio umano non è in grado di percepire cambiamenti così repentini dell'immagine (fenomeno detto della "persistenza della visione"), esso crede di vedere quattro cifre diverse, mentre in realtà ogni cifra viene inviata sempre in tutte le quattro posizioni, ma solo quella corrispondente alla posizione corretta viene di volta in volta accesa. La velocità di ripetizione dell'intero processo dà l'impressione che le diverse cifre siano tutte accese allo stesso istante mentre in realtà sono accese nelle rispettive posizioni ma in istanti diversi. E' un inganno per l'occhio, ma un inganno fatto bene !

Abbiamo detto che ogni segmento del display è un LED, quindi deve essere protetto da un resistore che ne regoli la corrente, come espressamente specificato nel datasheet del display.
Sempre dal Datasheet si legge che la tensione di "forward" dei LED è pari a 2V. Quindi, alimentando il tutto con 5V calcoliamo facilmente il valore della resistenza che ci serve per avere una corrente di 10 mA, più che sufficiente per alimentare i LED e ampiamente al di sotto dei 40 mA di corrente massimi raccomandati per i pin della scheda Arduino UNO.

$$\frac{5V - 2V}{0{,}010\ A} = 300\ Ohm$$

Ho preso quindi il valore standard più vicino per eccesso, che è 330 Ohm, ed ho collegato i resistori agli 8 pin (anodi) che mi serviva alimentare.
Da un punto di vista circuitale, ogni resistenza serve i quattro led corrispondenti ad un determinato segmento. Non si tratta però di LED in parallelo, avendo tutti catodi distinti.

Gestione del pulsante per cambio modalità di calcolo

All'accensione lo strumento visualizza la velocità del vento in m/s, che è l'unità di misura comunemente utilizzata. Ho voluto però aggiungere un pulsante NA (Normalmente Aperto) che permetta di cambiare la modalità ed esprimere la velocità anche in Km/h e in nodi.

Per far ciò ho utilizzato alcuni pin analogici, che all'occorrenza possono essere utilizzati come digitali (come da documentazione ufficiale di Arduino).

Lo schema elettrico è il seguente. Il pulsante è collegato al pin digitale 13 e al ground tramite un resistore da 10 Kohm (resistore di pull-down, come per la lettura del sensore di vento). Il programma legge lo stato del pulsante ed in base ad esso setta una variabile che condiziona la modalità di calcolo della velocità.

Ho aggiunto tre LED verdi collegati ai pin analogici (in modalità digitale) A0, A1 e A2, per indicare quale delle tre modalità di calcolo è attiva. I LED vengono accesi alternativamente ad ogni pressione del pulsante.

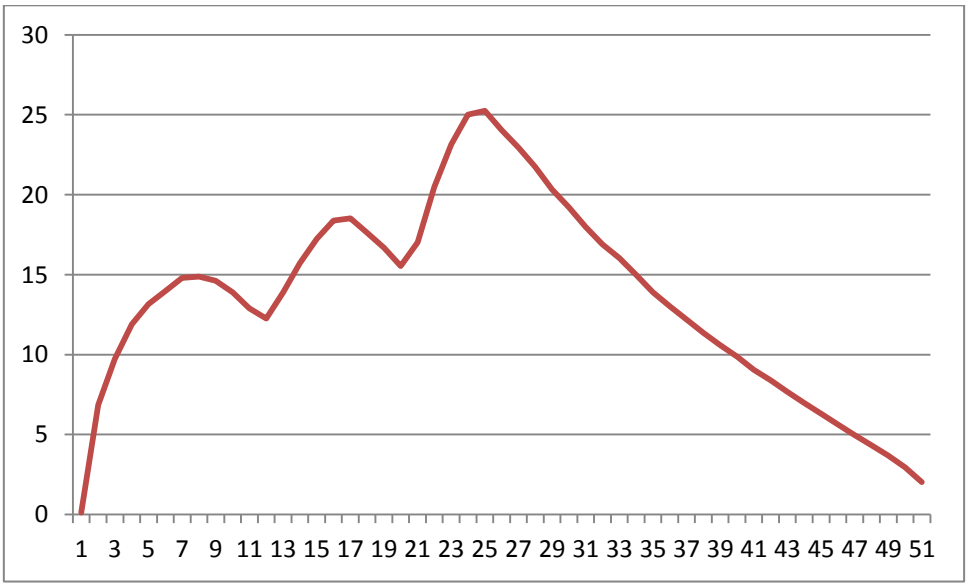
Va detto che per una buona gestione del pulsante è necessario aggiungere, alla semplice lettura dello stato del pulsante, anche alcune istruzioni di "debouncing" ossia antirimbalzo.

In pratica, il programma controlla lo stato del pulsante all'interno di un periodo di 50 millisecondi, per evitare che la pressione accidentale prolungata o altri tipi di interferenze nel segnale possano dare una falsa lettura. Se, trascorsi questi 50 ms, lo stato del pulsante non è più cambiato, allora la variabile di stato viene impostata al suo valore definitivo.

Senza questo accorgimento, infatti, risulta difficile premere il pulsante e variare lo stato dei led in modo preciso e netto. Con l'introduzione dell'antirimbalzo il sistema diviene invece affidabile e ogni pressione del pulsante causa lo switch tra un led e l'altro senza alcun problema.

Sperimentazione

Il grafico sotto rappresenta una semplice prova effettuata soffiando ripetutamente sull'anemometro. I valori sono stati inviati al monitor seriale di Arduino e poi copiati in Execl per generare il grafico.
Si osserva la linea regolare di smorzamento dello strumento, quando non più soggetto al flusso d'aria, rallenta fino a fermarsi.



Per verificare la correttezza della velocità rilevata ho fatto poi una prova utilizzando l'automobile come "generatore di vento". Ho montato il sensore su un'asta che ho poi esposto dal finestrino ad una velocità di 36 Km/h (10 m/s), verificata dal navigatore satellitare (non dal tachimetro dell'auto !). L'indicazione dell'anemometro ha confermato i dati di calibrazione con una buone precisione, tuttavia la prova così effettuata è facilmente soggetta ad errori. Mi riprometto, in futuro, di eseguire ulteriori test per convalidare la precisione dello strumento. Sarebbe molto utile il confronto diretto con uno strumento professionale o con un modello di serie, che putroppo non ho a disposizione.

Durante la prova ho riscontrato un problema nella visualizzazione dei dati sul display. La valocità, nella prima versione del software, veniva visualizzata ad ogni ciclo del programma, ma già a 36 Km/h l'eccessiva cadenza degli impulsi disturbava la visualizzazione rendendola poco stabile (sfarfallio accentuato).
Ho quindi introdotto una modifica al software, limitando la visualizzazione della velocità ad una volta al secondo. In questo modo la rappresentazione risulta stabile senza influire sull'algoritmo di lettura che procede alla velocità normale di esecuzione.

Per far ciò ho fatto ricorso di nuovo alla funzione millis(), che restituisce il numero di millisecondi trascorsi dall'accensione di Arduino.

Tale funzione è utilissima nella programmazione di Arduino, infatti permette di controllare gli intervalli di tempo senza utilizzare l'istruzione "delay()", la quale ferma l'esecuzione del programma per il tempo specificato, andando ad inibire anche la lettura dei sensori.

Sfruttando la funzione **millis()** Ho realizzato un timer dinamico (nel senso che misura intervalli di tempo parziali sequenziali), utilissimo in molte occasioni ed estremamente facile da implementare.

Si setta una variabile **t_start = millis()** ad un certo punto del programma. Dopodichè si va a verificare il tempo trascorso semplicemente testando il valore **mills() – t_start**
Se questo soddisfa determinati requisiti si azzera il timer imponendo **t_start = 0** il che permette di iniziare la misurazione di un nuovo intervallo di tempo.

t_start deve essere una variabile di tipo **usigned long** (valore restituito dalla funzione millis()), che può memorizzare valori da 0 + 4,294,967,295.
Ciò significa che va in overflow (assume un valore troppo grande e quindi non rappresentabile) dopo circa 50 giorni di funzionamento continuativo di Arduino, dopodichè si riazzera automaticamente.
Una variabile di tipo int andrebbe invece in overflow (con Arduino UNO) dopo 32 secondi, causando con molta probabilità errori nell'esecuzione del programma.

Realizzazione del circuito definitivo

Fino ad ora abbiamo visto ciò che ha riguardato la sperimentazione sul prototipo. Dopo la messa a punto del circuito, e del software, ho disegnato il circuito in modo da poterlo trasportare su una schedina "millefori" da inserire poi in un contenitore appositamente costruito.
Questa fase è piuttosto impegnativa. La posizione di tutti i componenti deve essere decisa a priori, e lo stesso vale per i fili di collegamento. A differenza di una piastra stampata, infatti, sulla piastra millefori le piste di collegamento tra i vari componenti sono realizzate con fili saldati da punto a punto.

Naturalmente ho dovuto tenere conto della posizione di tutti i componenti che devono essere accessibili dall'esterno dello strumento:

Display
3 LED
Pulsante toggle
Interruttore di accensione
Presa di connessione del sensore
Ingresso USB
Ingresso di alimentazione di Arduino

Il contenitore dello strumento è stato realizzato con compensato da 4 mm di spessore. All'interno sono alloggiate la scheda UNO, la millefori con il circuito e il portabatterie per l'alimentazione.
Il coperchio, che è anche il pannello frontale, reca le aperture per il display, i LED, il pulsante e l'interruttore. Sul lato superiore ci sono la presa per connettere il sensore e l'interruttore, mentre sul lato sinistro la presa USB e l'alimentazione esterna di Arduino.
Il sensore è montato su una impugnatura, per essere retto con una mano durante le letture.

Considerazioni finali

Gennaio 2015.
Ho sviluppato alcune considerazioni sulla struttura del software e come questa può influenzare la lettura del sensore.
Innanzitutto ho analizzato la durata del loop(), scoprendo che il ritardo in microsecondi, utilizzato per la visualizzazione del display era errato. delayMicroseconds() infatti funziona correttamente solo fino a circa 16000 microsecondi. Oltre questo valore produce ritardi di lunghezza imprevedibile. Questa informazione è documentata sul sito ufficiale di Arduino. Ho quindi sostituito l'istruzione con un delay(5); che funziona correttamente (la funzione delayMicroseconds() con un valore di 20000 generava del tutto casualmente un delay di circa 5/1000, motivo per cui il programma funzionava ugualmente anche nella prima versione).

La durata del loop() risulta influenzata quasi esclusivamente dal ritardo legato al display. In pratica, ogni cifra accesa genera un ritardo di 5/1000. Quindi, con due cifre accese il loop dura circa 10/1000 e quando le cifre sono tre, 15/1000. Il tempo richiesto per le altre operazioni svolte durante il loop() è del tutto trascurabile rispetto al ritardo del display (verifica fatta sperimentalmente).

La lettura del sensore viene quindi effettuata ad intervalli di tempo variabili che vanno da 10 a 15 millesimi di secondo, in funzione del numero di cifre accese in quel momento (che dipende a sua volta dalla velocità rappresentata).

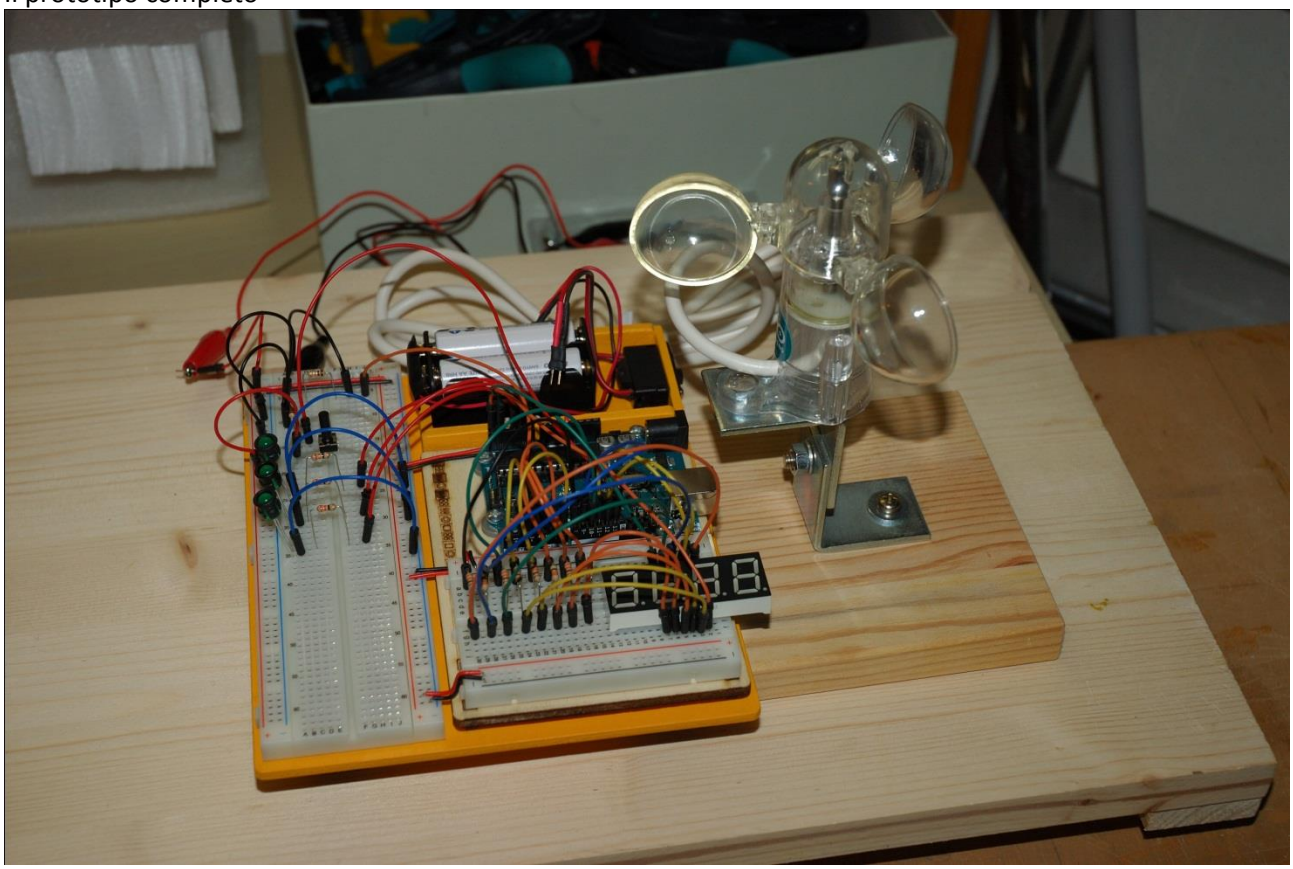
Con questa premessa ho cercato di capire se, e come, questo ritardo può influenzare le letture, e di conseguenza i valori di velocità calcolati.
Analizzando ciò che accade ad ogni loop del programma, con un elapsed time diciamo di 10 ms, si osserva che anche se il ritardo del display avviene in corrispondenza di una salita del segnale (che così non viene rilevata subito), la gestione dello stato del pin permette di rilevare comunque il cambio di stato, e quindi la salita, al loop successivo. Questo è vero perché il segnale rimane HIGH per una certa frazione (circa 30%) di giro del sensore (a causa del comportamento del reed magnetico).

In considerazione di quanto detto, prendiamo in esame un intervallo di un secondo, durante il quale consideriamo costante la velocità del vento. Diciamo 20 Km/h (che equivale a 8 pulsazioni al secondo). Assumiamo inoltre che il ritardo del display sia di 10 ms per ogni loop, che si traduce nel fatto che ogni secondo verranno effettuate circa 100 letture del sensore. Infine assumiamo che il pin del sensore rimanga in HIGH per 1/3 di ogni giro dell'anemometro. Questo significa che ogni secondo circa 300 ms danno una lettura HIGH. Considerando le 8 pulsazioni/sec prese come riferimento, della durata di 125 ms ciascuno, durante ogni pulsazione vengono effettuate circa 12 letture del sensore, più che sufficienti a determinare i punti di salita del segnale e calcolare correttamente la velocità.

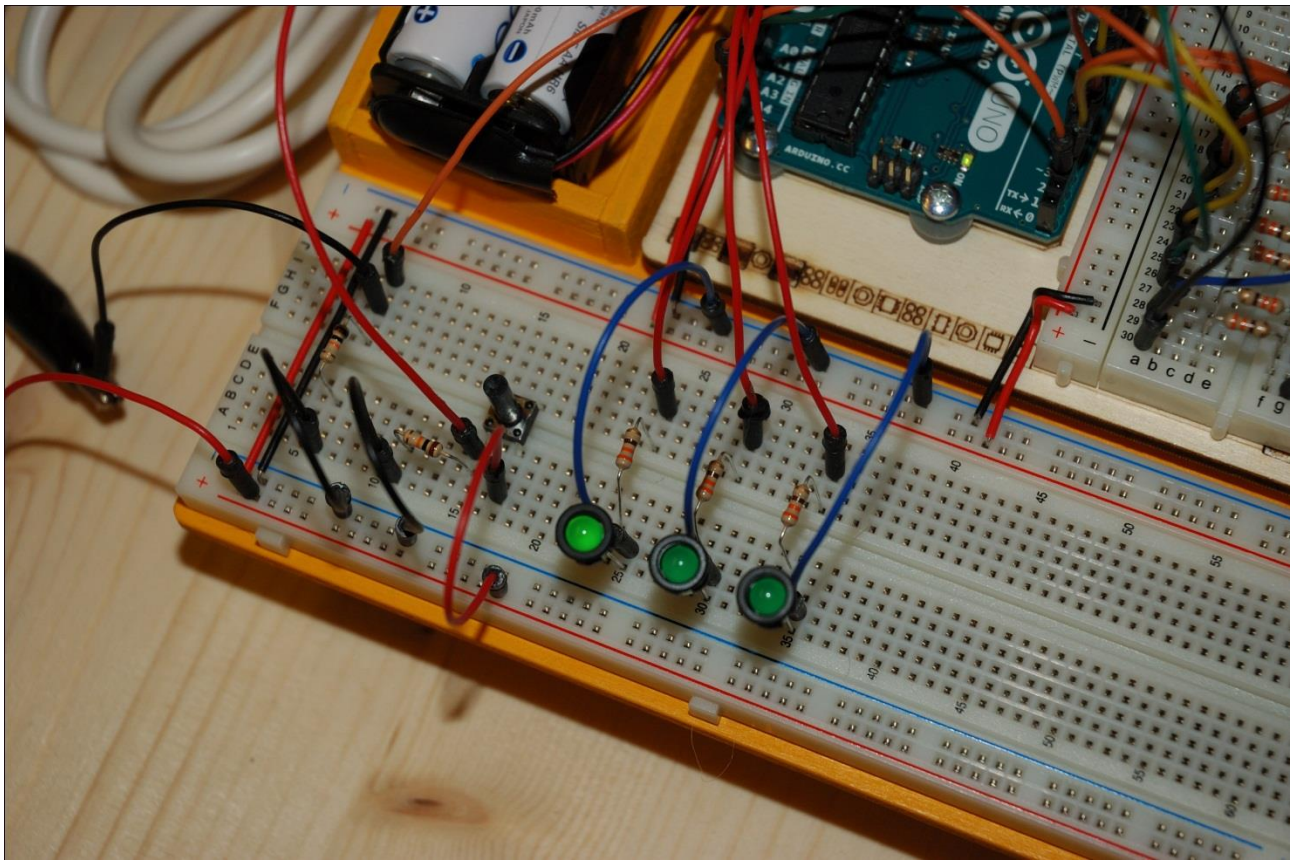
L'alternativa all'approccio "sequenziale" che ho utilizzato per sviluppare il programma sarebbe l'uso dell'interrupt 0 sul pin digitale 2, al quale è collageto il sensore. La gestione dell'interrupt permette di rilevare le salite del segnale esattamente quando queste avvengono e indipendentemente dal ritardo dovuto al display. Tuttavia, la documentazione di Arduino dice che nelle routine di servizio dell'interrupt è da evitare l'uso di delay() e millis(), perché non funzionano correttamente in quel contesto. Ciò impedisce quindi di calcolare la durata della pulsazione all'interno di una routine ISR (Interrupt Service Routine), obbligando così a farlo durante il normale loop e quindi incorrendo nuovamente nel "problema" del ritardo dovuto al display. In ogni caso è possibile modificare il software eliminando l'istruzione di lettura del sensore e attivando l'interrupt 0 sul pin 2. La funzione ISR associata all'interrupt dovrà quindi settare la variabile "pinval" al valore 1 quando rileva la salita del segnale. Il resto del programma rimane invariato. E' una soluzione che proverò.

Immagini

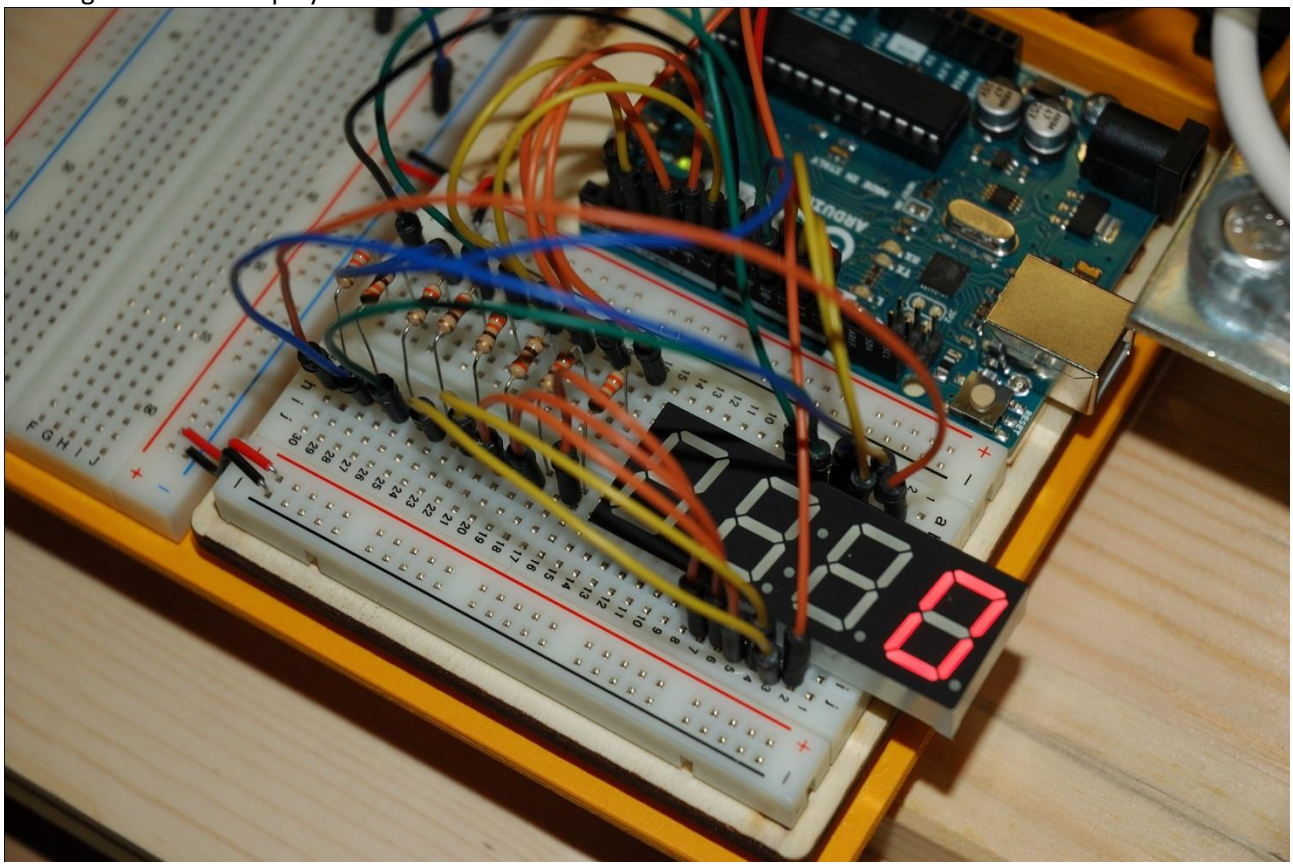
Il prototipo completo



Dettaglio dei led con il pulsante



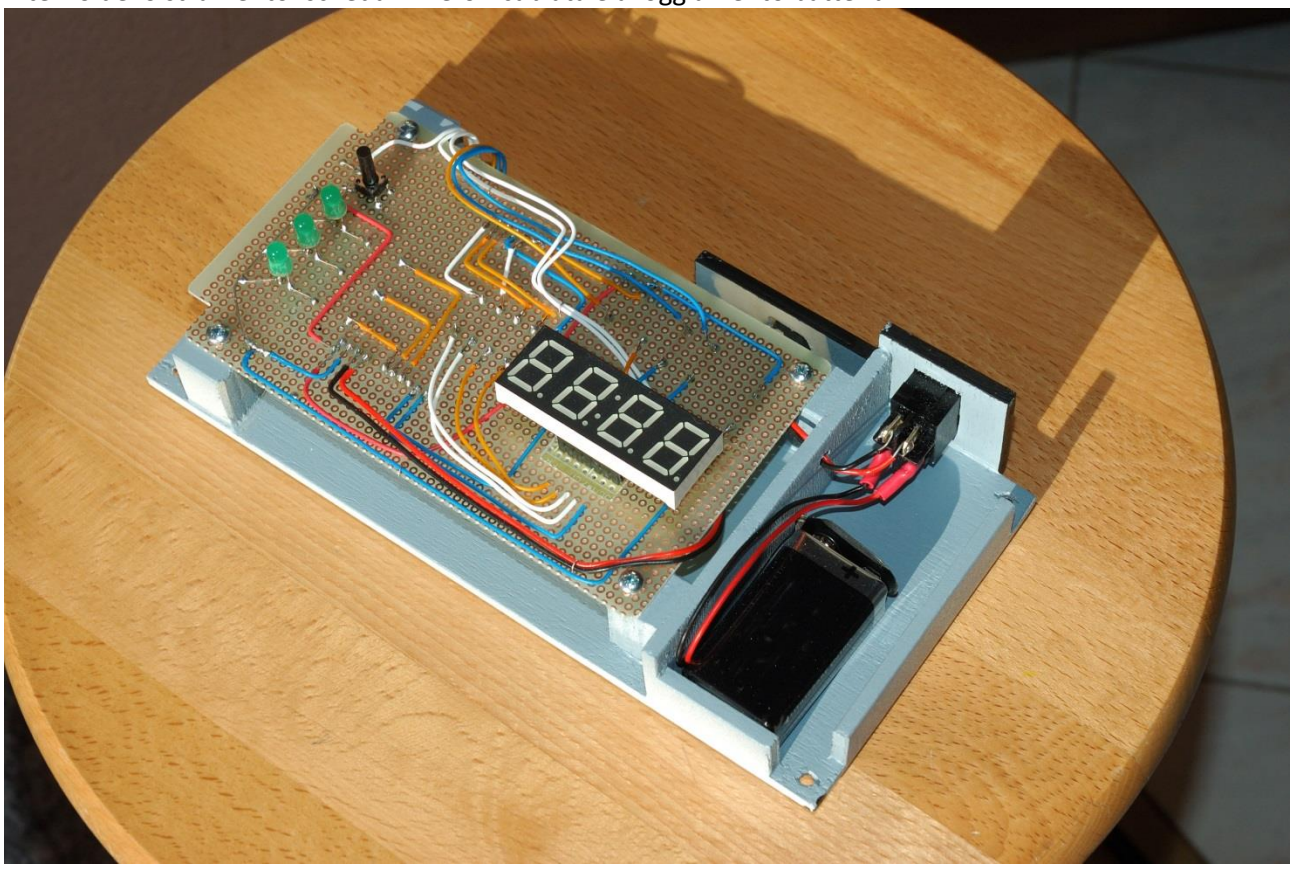
Il collegamento del display



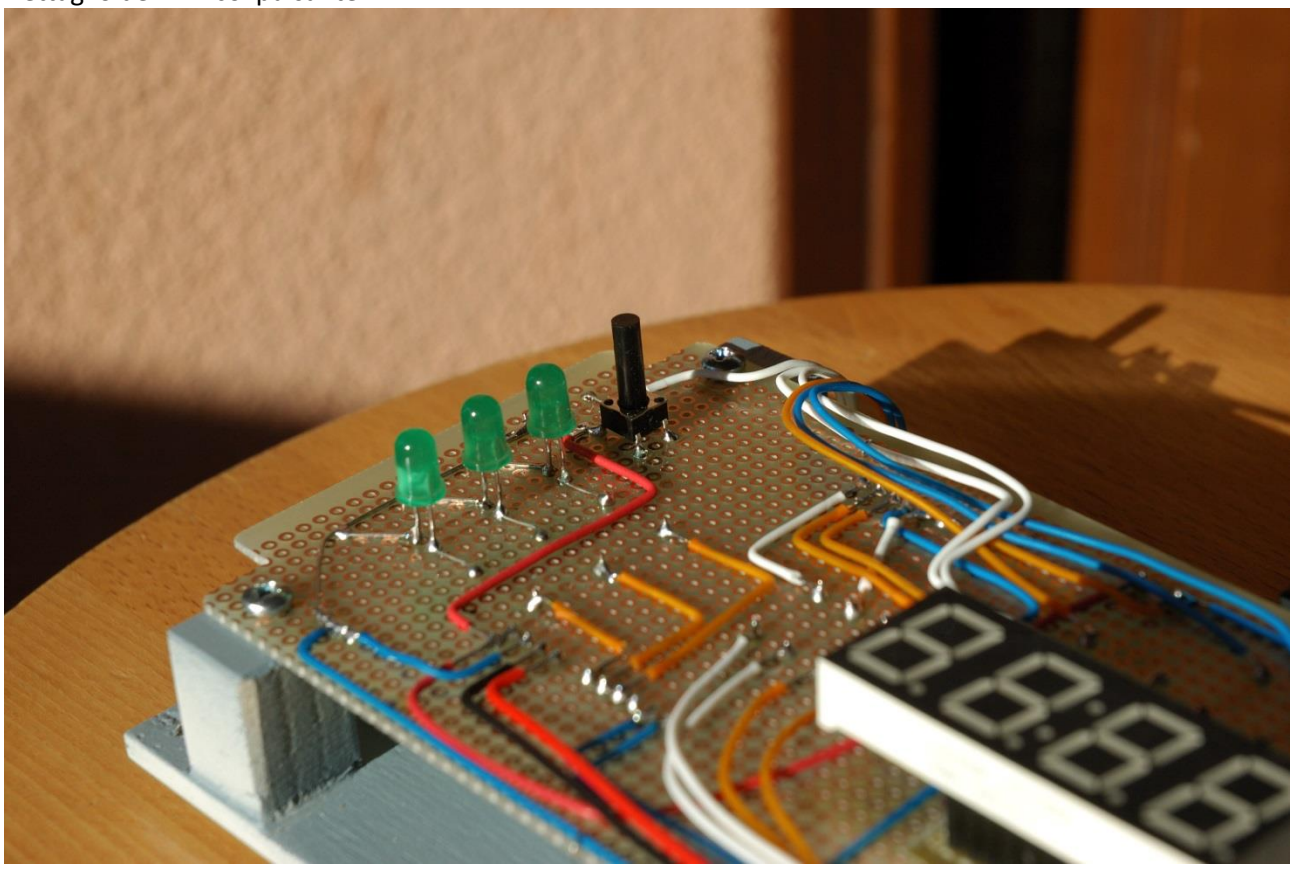
Il sensore di vento

Involucro esterno (legno verniciato)

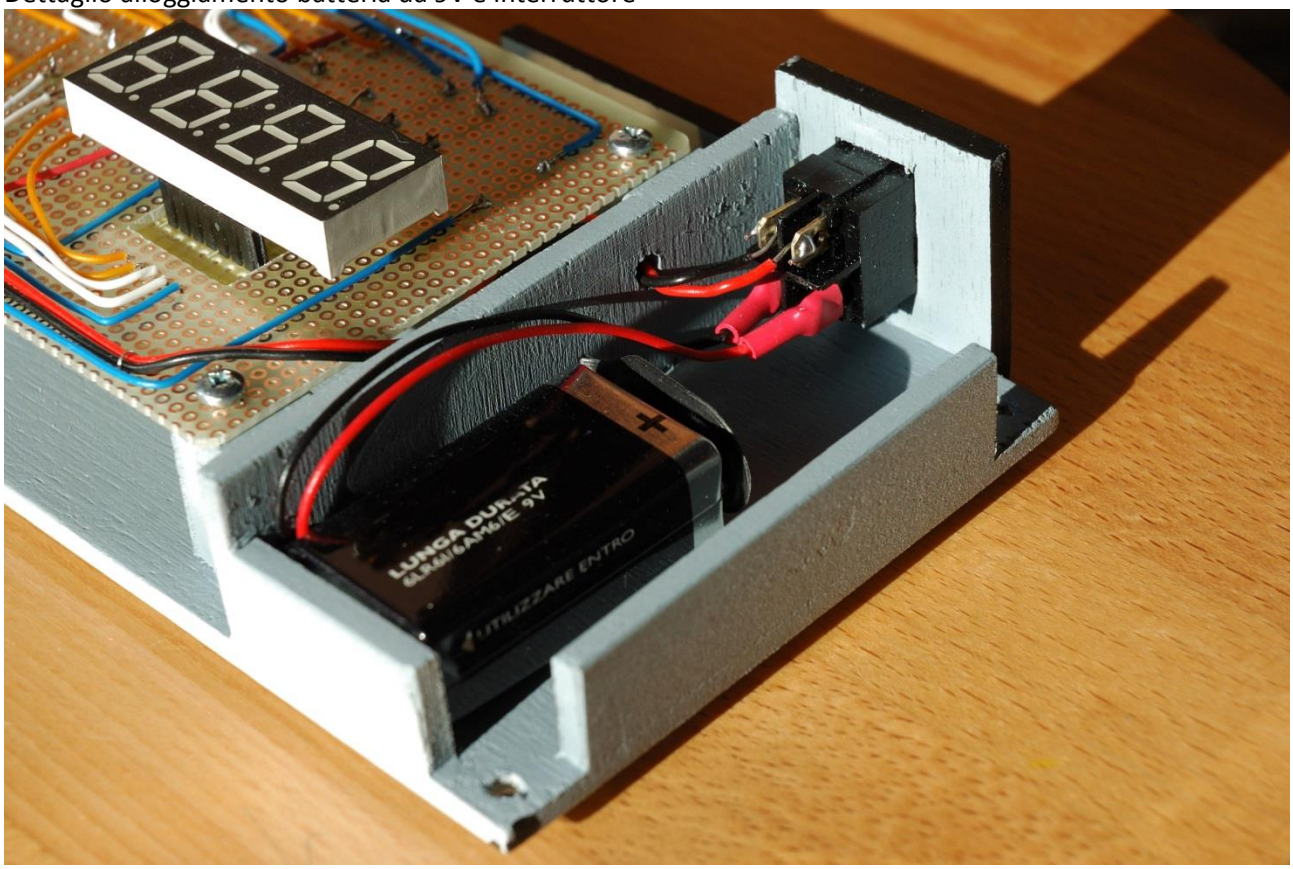
Interno dello strumento: scheda millefori cablata e alloggiamento batteria



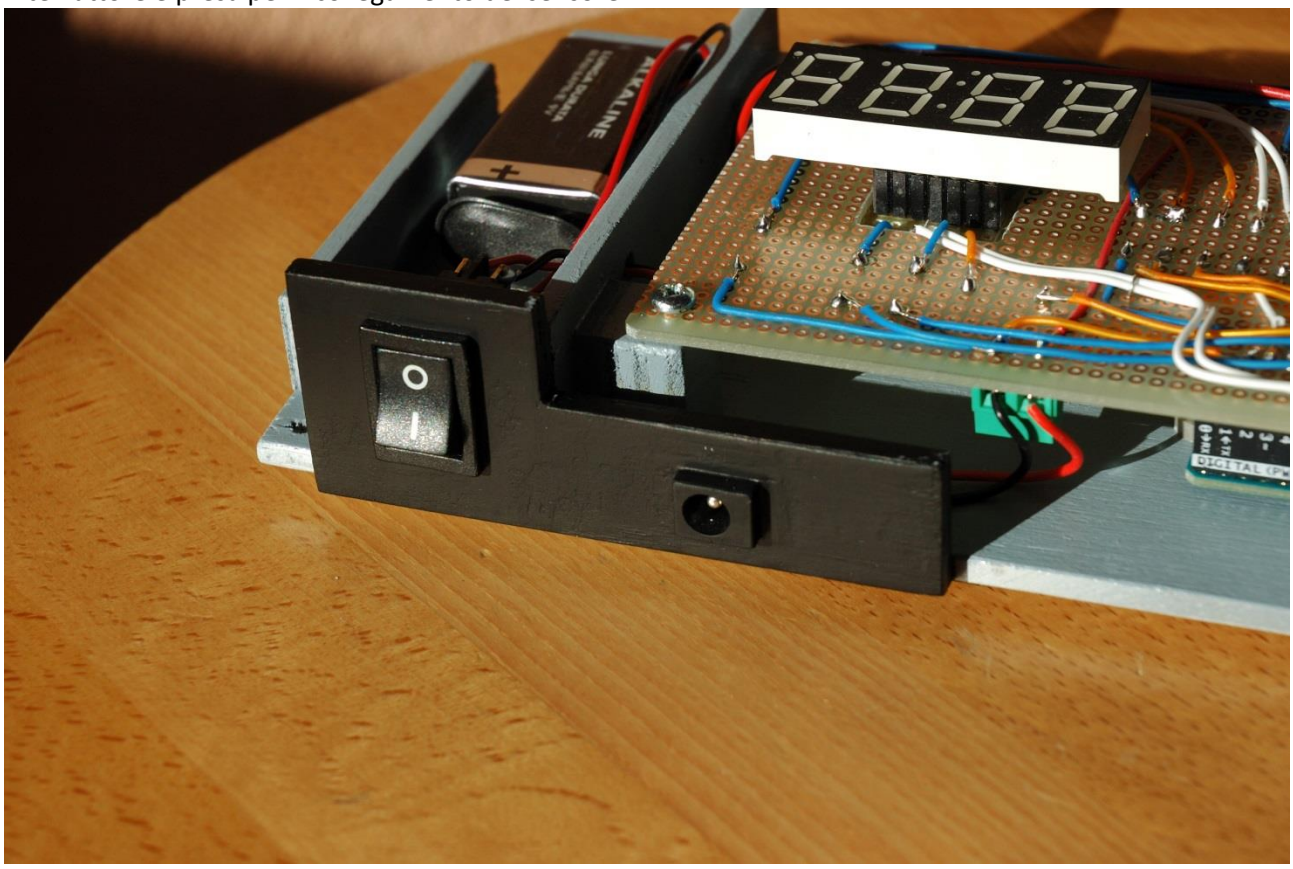
Dettaglio dei LED col pulsante



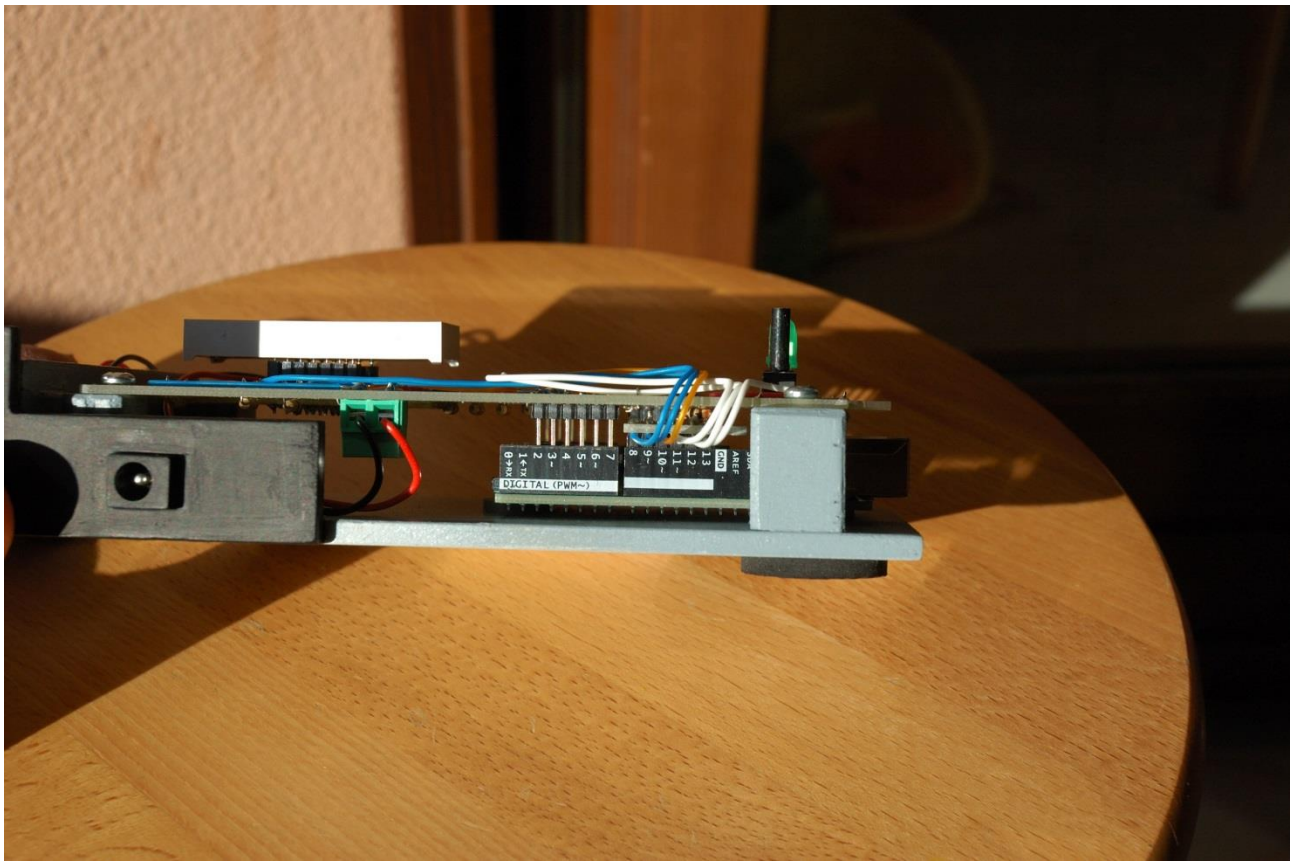
Dettaglio alloggiamento batteria da 9V e interruttore

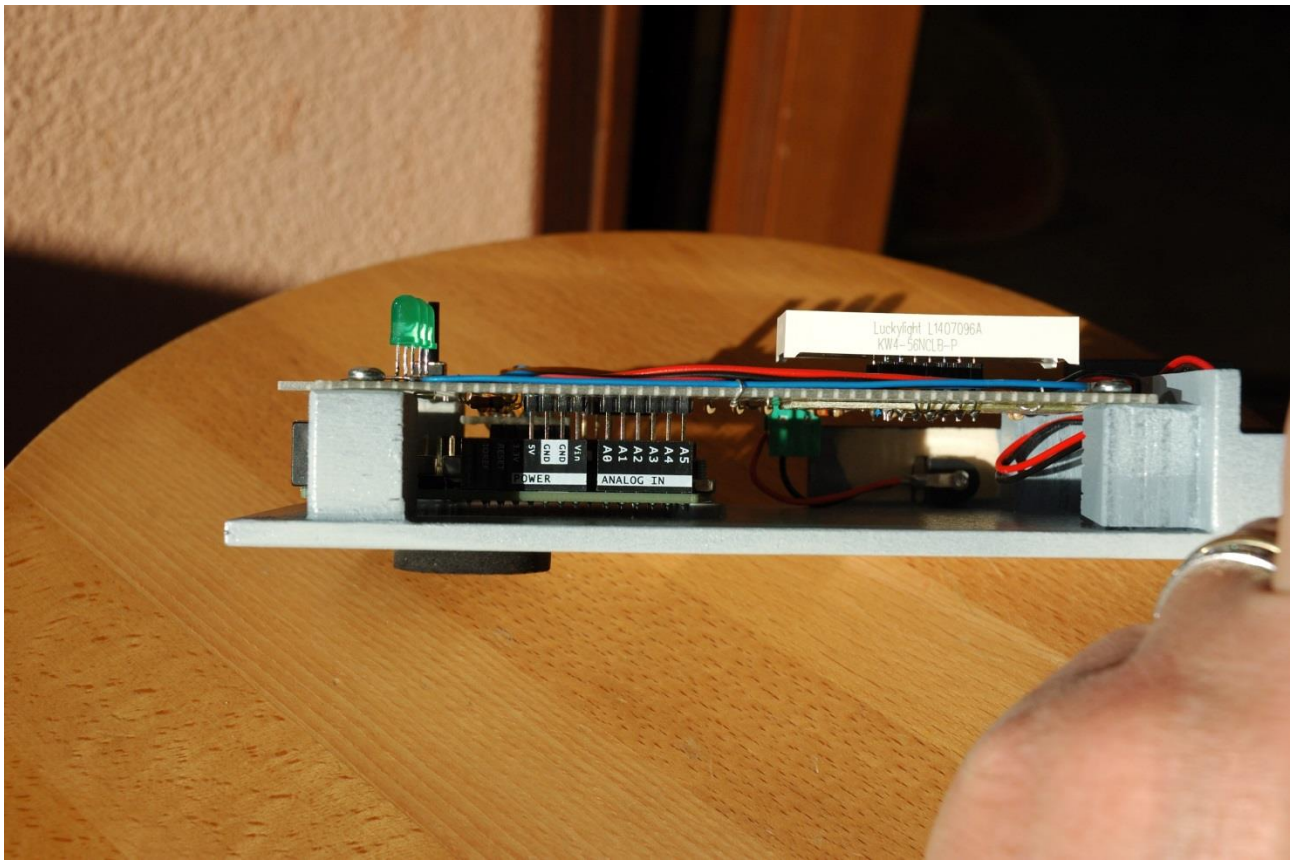


Interruttore e presa per il collegamento del sensore

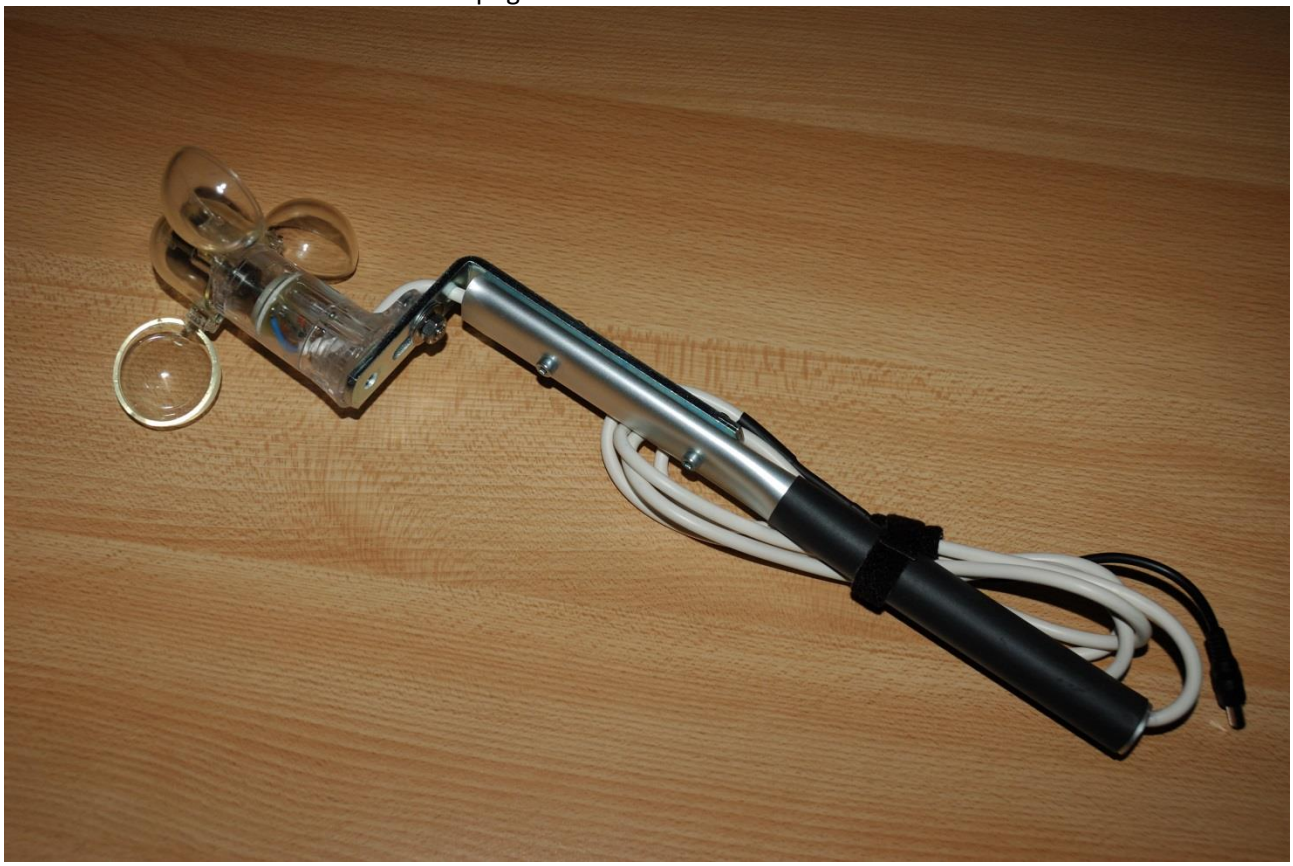


La scheda Arduino si trova sotto la scheda del circuito





Il sensore di vento è montato su una impugnatura di alluminio



Il cavo passa all’interno dell’impugnatura

Tabelle, schemi e disegni

A titolo di esempio riporto una tabella che mostra la corrispondenza tra i vari parametri che entrano in gioco nel calcolo della velocità:

pulse/s Numero di pulsazioni al secondo
pulse Durata della pulsazione il millisecondi
nodi Velocità equivalente in nodi
Km/h Velocità equivalente in nodi
m/s Velocità equivalente in nodi

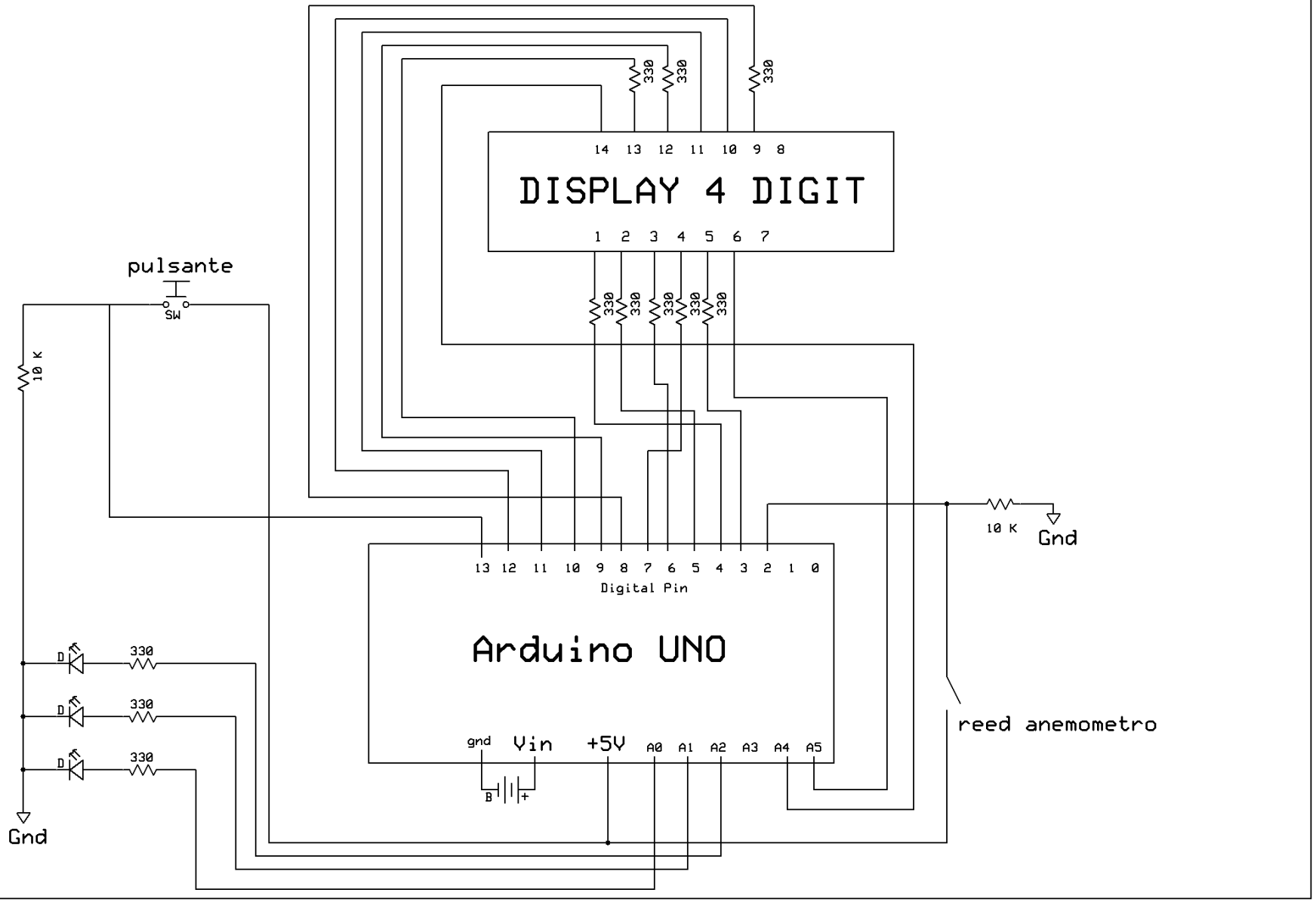
nodi	Km/h	m/s	pulse/s	pulse
2,70	5	1,4	2,00	500
3,24	6	1,7	2,40	417
3,78	7	1,9	2,80	357
4,32	8	2,2	3,20	313
4,86	9	2,5	3,60	278
5,40	10	2,8	4,00	250
5,94	11	3,1	4,40	227
6,48	12	3,3	4,80	208
7,02	13	3,6	5,20	192
7,56	14	3,9	5,60	179
8,10	15	4,2	6,00	167
8,64	16	4,4	6,40	156
9,18	17	4,7	6,80	147
9,72	18	5,0	7,20	139
10,26	19	5,3	7,60	132
10,80	20	5,6	8,00	125
11,34	21	5,8	8,40	119
11,88	22	6,1	8,80	114
12,42	23	6,4	9,20	109
12,96	24	6,7	9,60	104
13,50	25	6,9	10,00	100
14,04	26	7,2	10,40	96
14,58	27	7,5	10,80	93
15,12	28	7,8	11,20	89
15,66	29	8,1	11,60	86
16,20	30	8,3	12,00	83

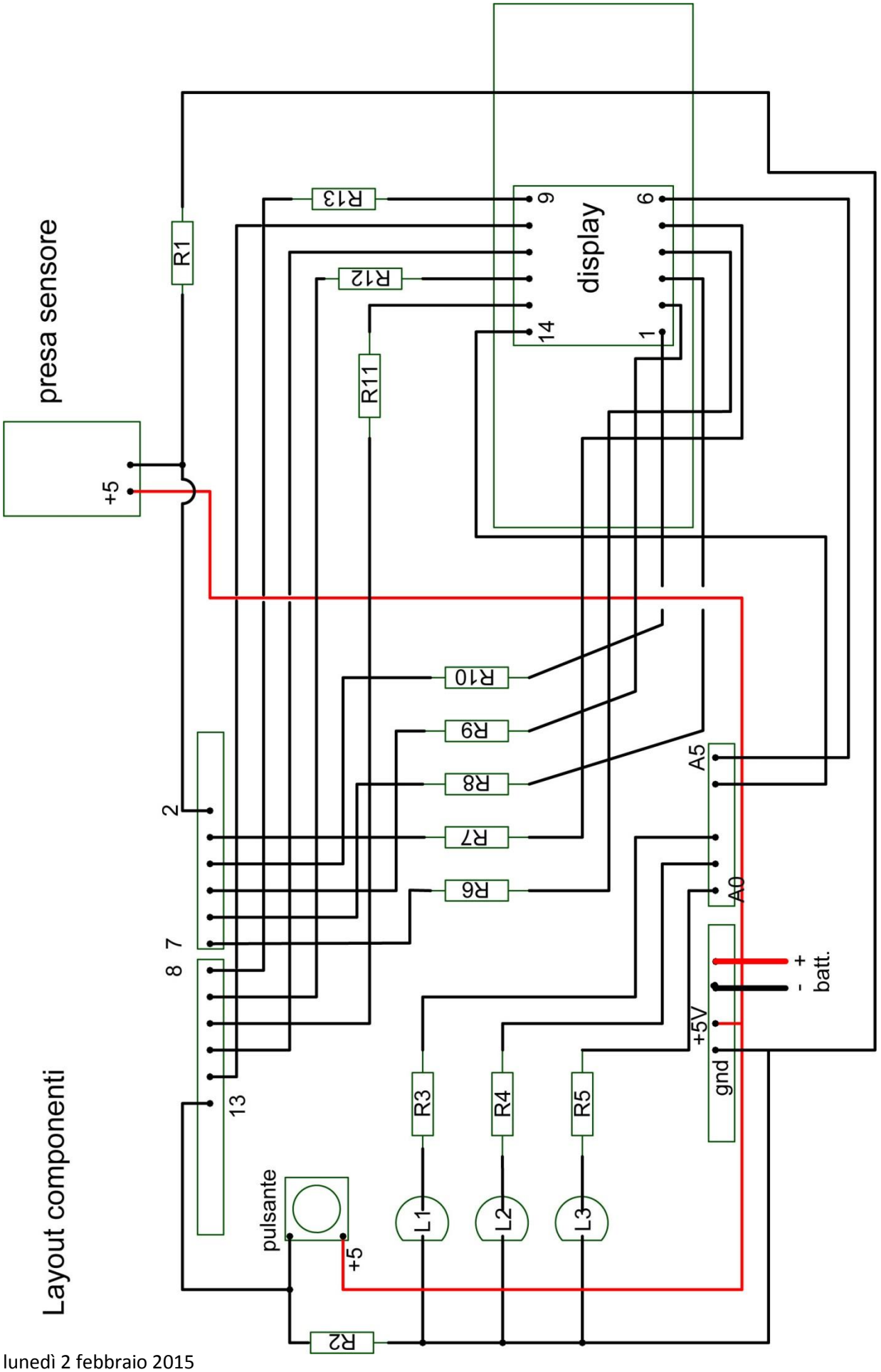
Il vento viene classificato in base all' intensità con la quale spira (leggero, forte,teso, ecc.).
L'intensità è determinata dalla velocità, espressa in m/sec, Km/h o nodi e misurata con gli anemometri. In base alla velocità, i venti vengono classificati in dodici gradi di intensità, secondo una scala di misura detta di Beaufort.
La tabella seguente riporta la classificazione dei venti e le velocità espressa in Km/h, m/s e nodi.

Grado Beaufort	Velocità (Km/h)	Tipo di vento	Velocità (nodi)	Condizioni ambientali e del mare	Velocità (m/s)
0	0 – 1	calma	0 – 1	Il fumo ascende verticalmente; il mare è uno specchio. **Mare forza zero.**	< 0.3
1	1 – 5	bava di vento	1 – 3	Il vento devia il fumo; increspature dell'acqua. **Mare forza uno.**	0.3 – 1.5
2	6 – 11	brezza leggere	4 – 6	Le foglie si muovono: onde piccole ma evidenti. **Mare forza due.**	1.6 – 3.3
3	12-19	brezza	7 – 10	Foglie e rametti costantemente agitati; piccole onde, creste che cominciano ad infrangersi. **Mare forza due.**	3.4 – 5.4
4	20 - 28	brezza vivace	11 – 16	Il vento solleva polvere, foglie secche, i rami sono agitati: piccole onde che diventano più lunghe. **Mare forza tre.**	5.5 – 7.9
5	29 – 38	brezza tesa	17 – 21	Oscillano gli arbusti con foglie; si formano piccole onde nelle acque interne; onde moderate allungate. **Mare forza quattro.**	8 – 10.7
6	39 – 49	vento fresco	22 – 27	Grandi rami agitati, sibili tra i fili telegrafici; si formano marosi con creste di schiuma bianca e spruzzi. **Mare forza cinque.**	10.8 – 13.8
7	50 – 61	vento forte	28 – 33	Interi alberi agitati, difficoltà a camminare contro vento; il mare è grosso, la schiuma comincia ad essere sfilacciata in scie. **Mare forza sei.**	13.9 - 17.1
8	62 – 74	burrasca moderata	34 – 40	Rami spezzati, camminare contro vento è impossibile: marosi di altezza media e più allungati, dalle creste si distaccano turbini di spruzzi. **Mare forza sette.**	17.2 – 20.7
9	75 – 88	burrasca forte	41 – 47	Camini e tegole asportati; grosse ondate,spesse scie di schiuma e spruzzi, sollevate dal vento,riducono la visibilità. **Mara forza otto.**	20.8 – 24.4
10	89 – 102	tempesta	48 – 55	Rara in terraferma, alberi sradicati, gravi danni alle abitazioni: enormi ondate con lunghe creste a pennacchio. **Mare forza nove.**	24.5 – 28.4
11	103 – 117	fortunale	56 – 63	Raro, gravissime devastazioni: onde enormi ed alte, che possono nascondere navi di media stazza; ridotta visibilità. **Mare forza dieci.**	28.5 – 32.6
12	oltre 118	uragano	64 +	Distruzione di edifici, manufatti, ecc.;in mare la schiuma e gli spruzzi riducono assai la visibilità. **Mare forza dieci.**	32.7 +

Riporto di seguito i fattori di conversione per le unità di misura comunemente utilizzate
1 nodo = 1,852 Km/h **1 m/s** = 3,6 Km/h **1 m/s** = 1,944 nodi

ANEMOMETRO: schema elettrico




Layout componenti
presa sensore
+5
R1
R13
R12
R11
display
9
6
14
1
R10
R9
R8
R7
R6
2
7
8
13
A5
A0
+5V
gnd
+ batt. -
R3
R4
R5
L1
L2
L3
pulsante
+5
R2

Software

Di seguito è riportata la versione definitiva del software

```
// ============================================================
// PROJECT:     ANEMOMETRO
// DESCRIPTION:  Realizzazione di un anemometro gestito da Arduino
// AUTHOR:      FEDERICO DAVANTERI
// DATE:        29/10/2014
// REV:         DEFINITIVO_2015_01_00 - 1.0
// REV:         DEFINITIVO_2015_01_01 - 1.1 - 22/01/2015
//              Sostituito delayMicroseconds(ritardo); con delay(5); nella routine di multiplexing del display
//              Il precedente valore di 20000 microsec. generava in realtà un ritardo di soli 4 ms circa.
//              Problema documentato su arduino.cc : delayMicroseconds funziona correttamente solo fino a
16383.
// ============================================================

// Pin dedicato alla lettura del sensore di vento
const int getpulsepin = 2;

// Associazione dei pin ai segmenti delle cifre (anodi)
const int a = 10;
const int b = 8;
const int c = 7;
const int d = 5;
const int e = 4;
const int f = 9;
const int g = 3;
const int p = 6;       // punto decimale
const int toggle = 13;  // pin del pulsante di modo calcolo

// Associazione dei pin alle cifre (catodi)
// cc1 e cc4 sono collegati ai pin A4 e A5 utilizzati come digitali
// const int cc1 = A4;
// const int cc4 = A5;
const int cc2 = 11;
const int cc3 = 12;

// Variabili relative alla gestione del display
unsigned long counter = 0;      // Contatore per azzeramento cifre dopo 2 sec inattività
unsigned long disp_start = 0;   // Contatore per visualizzazione asincrona sul display
unsigned long disp_time = 1000;  // Durata (ms) visualizzazione asincrona display

// Ritardo spegnimento cifre per multiplexing display
// 1 microsecondo = 1/1.000.000 sec  -->  20.000 microsecondi = 1/50 sec
// int ritardo = 20000;    modifica del 22/01/2015

// Variabili per rappresentazione delle cifre sul display
int n1 = 0;     // valore calcolato in tempo reale
int n2 = 0;     // valore calcolato in tempo reale
int dec = 0;    // valore calcolato in tempo reale
int digit1 = 0;  // valore n1  assegnato in visualizzazione ritardata
```

```
int digit2 = 0;  // valore n2  assegnato in visualizzazione ritardata
int digit3 = 0;  // valore dec assegnato in visualizzazione ritardata

// variabili per calcoli anemometro
float windspeed = 0.0;
float m_s_conv = 1000.0/3600.0; // Fattore conversione Km/h --> m/s
float nodi_conv = m_s_conv * 1.944;
unsigned long startime = 0;
unsigned long durata = 0;
int stato = 0;
int pinval = 0;

// variabili per cambio modalità calcolo
int mode;              // variabile di lettura del pulsante
int toggle_status = 0;  // toggle_status  0 --> m/s     1 --> Km/h   2 --> Nodi

// antirimbalzo per il pulsante
unsigned long lastDebounceTime = 0;     // istante della precedente pressione del pulsante
unsigned long debounceDelay = 50;       // tempo di debounce
int  buttonState = 0;                // stato corrente del pulsante
int  lastButtonState = LOW;            // ultimo stato del pulsante

// Funzioni di gestione per il display

void WriteDig(int n, int dig)
{
 // Funzione che gestisce il display 4 digits a CATODO COMUNE
 // n   = valore da rappresentare
 // dig = posizione cifra
 // spegnere  un digit = HIGH (perchè ha catodo comune)
 // accendere un digit = LOW  (perchè ha catodo comune)
 // spegnere  un segmento = LOW
 // accendere un segmento = HIGH

 switch(dig) // gestione cifre
 {

 case 1: // la prima cifra è sempre spenta (non utilizzata)
   digitalWrite(A4, HIGH);
   digitalWrite(cc2, HIGH);
   digitalWrite(cc3, HIGH);
   digitalWrite(A5, HIGH);
   digitalWrite(p, LOW);
   break;
 case 2:
   digitalWrite(A4, HIGH);
   digitalWrite(cc2, LOW);
   digitalWrite(cc3, HIGH);
   digitalWrite(A5, HIGH);
   digitalWrite(p, LOW);
   break;
 case 3:
```

```
  digitalWrite(A4, HIGH);
  digitalWrite(cc2, HIGH);
  digitalWrite(cc3, LOW);
  digitalWrite(A5, HIGH);
  // accende il punto decimale solo tra la terza e quarta cifra
  digitalWrite(p, HIGH);
  break;
 case 4:
  digitalWrite(A4, HIGH);
  digitalWrite(cc2, HIGH);
  digitalWrite(cc3, HIGH);
  digitalWrite(A5, LOW);
  digitalWrite(p, LOW);
  break;
 }

 switch(n)  // gestione segmenti
 {
 case 0:
  digitalWrite(a, HIGH);
  digitalWrite(b, HIGH);
  digitalWrite(c, HIGH);
  digitalWrite(d, HIGH);
  digitalWrite(e, HIGH);
  digitalWrite(f, HIGH);
  digitalWrite(g, LOW);
  break;
 case 1:
  digitalWrite(a, LOW);
  digitalWrite(b, HIGH);
  digitalWrite(c, HIGH);
  digitalWrite(d, LOW);
  digitalWrite(e, LOW);
  digitalWrite(f, LOW);
  digitalWrite(g, LOW);
  break;
 case 2:
  digitalWrite(a, HIGH);
  digitalWrite(b, HIGH);
  digitalWrite(c, LOW);
  digitalWrite(d, HIGH);
  digitalWrite(e, HIGH);
  digitalWrite(f, LOW);
  digitalWrite(g, HIGH);
  break;
 case 3:
  digitalWrite(a, HIGH);
  digitalWrite(b, HIGH);
  digitalWrite(c, HIGH);
  digitalWrite(d, HIGH);
  digitalWrite(e, LOW);
  digitalWrite(f, LOW);
```

```
    digitalWrite(g, HIGH);
    break;
  case 4:
    digitalWrite(a, LOW);
    digitalWrite(b, HIGH);
    digitalWrite(c, HIGH);
    digitalWrite(d, LOW);
    digitalWrite(e, LOW);
    digitalWrite(f, HIGH);
    digitalWrite(g, HIGH);
    break;
  case 5:
    digitalWrite(a, HIGH);
    digitalWrite(b, LOW);
    digitalWrite(c, HIGH);
    digitalWrite(d, HIGH);
    digitalWrite(e, LOW);
    digitalWrite(f, HIGH);
    digitalWrite(g, HIGH);
    break;
 case 6:
    digitalWrite(a, HIGH);
    digitalWrite(b, LOW);
    digitalWrite(c, HIGH);
    digitalWrite(d, HIGH);
    digitalWrite(e, HIGH);
    digitalWrite(f, HIGH);
    digitalWrite(g, HIGH);
    break;
  case 7:
    digitalWrite(a, HIGH);
    digitalWrite(b, HIGH);
    digitalWrite(c, HIGH);
    digitalWrite(d, LOW);
    digitalWrite(e, LOW);
    digitalWrite(f, LOW);
    digitalWrite(g, LOW);
    break;
 case 8:
    digitalWrite(a, HIGH);
    digitalWrite(b, HIGH);
    digitalWrite(c, HIGH);
    digitalWrite(d, HIGH);
    digitalWrite(e, HIGH);
    digitalWrite(f, HIGH);
    digitalWrite(g, HIGH);
    break;
  case 9:
    digitalWrite(a, HIGH);
    digitalWrite(b, HIGH);
    digitalWrite(c, HIGH);
    digitalWrite(d, HIGH);
```

```
   digitalWrite(e, LOW);
   digitalWrite(f, HIGH);
   digitalWrite(g, HIGH);
   break;
  }

// ritardo tra un refresh e il successivo
// attende il ritardo e spegne tutte le cifre

   // delayMicroseconds(ritardo);   modifica del 22/01/2015

   delay(5);

   digitalWrite(A4, HIGH);
   digitalWrite(cc2, HIGH);
   digitalWrite(cc3, HIGH);
   digitalWrite(A5, HIGH);
}

void setup() {

 //Serial.begin(9600);

// pin digitali segmenti display
  pinMode(a, OUTPUT);
  pinMode(b, OUTPUT);
  pinMode(c, OUTPUT);
  pinMode(d, OUTPUT);
  pinMode(e, OUTPUT);
  pinMode(f, OUTPUT);
  pinMode(g, OUTPUT);
  pinMode(p, OUTPUT);

// pin digitali cifre display
  pinMode(A4, OUTPUT);
  pinMode(cc2, OUTPUT);
  pinMode(cc3, OUTPUT);
  pinMode(A5, OUTPUT);

// pin digitali pulsante e led di modalità calcolo
  pinMode(toggle, INPUT);
  pinMode(A0, OUTPUT);
  pinMode(A1, OUTPUT);
  pinMode(A2, OUTPUT);
  digitalWrite(A0, HIGH);
  digitalWrite(A1, LOW);
  digitalWrite(A2, LOW);

// pin dell'anemometro
  pinMode(getpulsepin, INPUT);
}
```

```
void loop ()
{
   pinval = digitalRead(getpulsepin);

// quando rileva una pulsazione avvia il calcolo della velocità
   //Serial.println(pinval);
   if ((stato == 0) & (pinval == 1)) {

   counter = millis();               // contatore per azzeramento display dopo timeout
   durata = millis() - startime;        // intervallo tra due pulsazioni
   startime = millis();              // setta nuovo startime per pulsazione successiva

   switch(toggle_status)
   {
   case 0:
   windspeed = (2500.0/durata)*m_s_conv;  // calcola velocità in m/s
   //Serial.println("m/s");
   //Serial.println(windspeed);
   break;
   case 1:
   windspeed = 2500.0/durata;           // calcola velocità in Km/h
   //Serial.println("Km/h");
   //Serial.println(windspeed);
   break;
   case 2:
   windspeed = (2500.0/durata)*nodi_conv;    // calcola velocità in Nodi
   //Serial.println("nodi");
   //Serial.println(windspeed);
   break;
   }

// decodifica le cifre per la visualizzazione

   n1  = int(windspeed/10);
   n2  = int(windspeed-(n1*10));
   dec = int((windspeed-int(windspeed))*10);
}

// visualizzazione asincrona sul display
// visualizza le stesse cifre per la durata definita da disp_time (1000 ms)
// poi aggiorna le cifre con il nuovo valore di velocità e resetta il contatore

   if ((millis() - disp_start) < disp_time) {
   // WriteDig(0,1);  modifica del 22/01/2015
   if (digit1 != 0) { WriteDig(digit1,2); };  // se le decine sono a zero non accende la cifra
   WriteDig(digit2,3);
   WriteDig(digit3,4);
   } else {
   disp_start = millis();
   digit1 = n1;
   digit2 = n2;
   digit3 = dec;
```

```
   }

// azzera le cifre "0.0" dopo 2 sec di inattività del sensore

   if ((millis() - counter) > 2000.0) {
   n1 = 0;
   n2 = 0;
   dec = 0;
   }

 // gestione del pulsante di modalità calcolo (antirimbalzo)

  mode = digitalRead(toggle);       // legge lo stato del pulsante
  //Serial.println(mode);

  if (mode != lastButtonState) {     // se lo stato è cambiato resetta il contatore del tempo
  lastDebounceTime = millis();
  }

// se è trascorso l'intervallo di controllo setta il valore definitivo di toggle_status
  if ((millis() - lastDebounceTime) > debounceDelay) {
    if (mode != buttonState) {
      buttonState = mode;

       if(buttonState == 1) {
         if(toggle_status == 0) {
         digitalWrite(A0, LOW);
         digitalWrite(A1, HIGH);
         digitalWrite(A2, LOW);
         toggle_status = 1;
         } else if (toggle_status == 1){
         digitalWrite(A0, LOW);
         digitalWrite(A1, LOW);
         digitalWrite(A2, HIGH);
         toggle_status = 2;
         } else {  // toggle_status == 2
         digitalWrite(A0, HIGH);
         digitalWrite(A1, LOW);
         digitalWrite(A2, LOW);
         toggle_status = 0;
         }
       }
    }
  }

   // Serial.println(toggle_status);
   stato = pinval;
   lastButtonState = mode;
}
```