Nel definire i due contatti dei pulsanti come ingressi (INPUT) ho usato un nuovo termine: ***INPUT_PULLUP*,** questo indica al processore di utilizzare la resistenza interna per mantenere il contatto allo stato alto (***HIGH*** = +5V), nel funzionamento normale tale resistenza è disabilitata per tutti i pin digitali. Quando noi premiamo il pulsante visto che l'altra estremità è collegata a massa, porterà il pin allo stato basso (***LOW*** = 0V) per cui nel loop controlleremo lo stato dei due pulsanti con l'istruzione: ***digitalRead***(pulsante) == ***LOW***. Il doppio segno di uguaglianza non è un errore, è necessario per effettuare un confronto tra i due termini, nel caso se ne usi uno solo diventa un'assegnazione. Questa istruzione è abbinata alla funzione ***if*** (se) in questo modo
if (***digitalRead***(pulsante) == ***LOW***) che significa se la lettura digitale del contatto X è allo stato basso allora esegui le istruzioni seguenti, in questo caso se lo stato del contatto non è basso non verrà eseguita nessuna istruzione e si procederà con le istruzioni del prossimo blocco.
L'istruzione ***if*** può essere seguita da un secondo blocco chiamato ***else*** (altrimenti) in questo modo

```
if ( X == 10 ){
  //serie di istruzioni da eseguire se il test è vero
}
else {
  // serie di istruzioni da eseguire se il test è falso
}
```

se vogliamo eseguire più test i blocchi ***if*** possono essere nidificati o innestati uno all'interno dell'altro in questo modo:

```
if ( X == 10 ){
  //serie di istruzioni da eseguire se il test è vero
}
else if ( X == 9 ){
  // serie di istruzioni da eseguire se il primo test è falso
  // e il secondo test è vero
}
else {
  // istruzioni da eseguire se entrambi i test sono falsi
}
```

In questo modo possiamo verificare situazioni anche molto complesse nidificando molti blocchi
in questi casi, bisogna fare particolare attenzione alle parentesi graffe in quanto è facile dimenticarne qualcuna. Fortunatamente se nelle opzioni abbiamo scelto la chiusura automatica dei blocchi, la scrittura sarà più semplice.
Compiliamo e trasferiamo il programma e vediamo come funziona.
Qui accendiamo e spegniamo un led ma possiamo pensare più in grande, tramite opportuni circuiti alzare e abbassare le tapparelle o la porta del garage, accendere e spegnere un ventilatore etc. …

Ora quello che facciamo con due interruttori, potremmo farlo anche con uno solo, alla prima pressione accendiamo se premiamo ancora spegniamo e così via. Proviamo dunque usando lo stesso circuito ed usando solo il primo pulsante (pin 9):

```
/* 1 led 2 pulsanti versione 1
 *  il led è collegato al pin 12
 *  i due pulsanti al pin 9 e pin 8
 *  l'altro contatto è verso massa
 */
int ledPin = 12;
int buttonApin = 9;

void setup(){
  pinMode(ledPin, OUTPUT);              // il pin del led è OUTPUT
  pinMode(buttonApin, INPUT_PULLUP);    // i pin del pulsante è INPUT e usando
                                        // la resistenza interna va a +5V
}

void loop(){
  if (digitalRead(buttonApin) == LOW){    // se il pulsante A è premuto porto a 0V il pin, il pin sarà LOW
    if (digitalRead(buttonApin) == LOW){  // se il led è spento
      digitalWrite(ledPin, HIGH);         // accendo il led
    }
    else {                                // altrimenti
      digitalWrite(ledPin, LOW);          // spengo il led
    }
  }
}
```

Proviamo ad usare questo programma e vediamo come funziona.
Notiamo che non sempre il led si comporta come dovrebbe in risposta alla pressione del tasto, questo succede perché il contatto del nostro tasto è formato da una lamina metallica che è elastica e quando la premiamo comincia a vibrare, queste oscillazioni continuano per molti millisecondi provocando una serie di falsi contatti chiamati in inglese bouncing (rimbalzi), il nostro processore esegue l'intero loop migliaia di volte al secondo e quindi intercetta anche questi rimbalzi che vengono interpretati come pressioni successive del pulsante. Per ovviare a questo dobbiamo applicare una delle tecniche definite di debouncing (anti-rimbalzo), la più semplice consiste nel verificare qual'era lo stato del pulsante l'istante precedente e, a seconda del risultato, procedere o meno con il programma.
Vediamo quali sono le modifiche al codice che dobbiamo apportare:
essenzialmente ci servono due variabili in cui memorizzare ad ogni ciclo lo stato del pulsante attuale e lo stato del pulsante al ciclo precedente e una variabile per cambiare lo stato del led da acceso a spento e viceversa. Ecco come si presentalo sketch:

```
1  /* 1 led 1 pulsanti versione 2
2   * versione antirimbalzo
3   * il led è collegato al pin 12
4   * il pulsante al pin 9
5   * l'altro contatto è verso massa
6   */
7  int ledPin = 12;
8  int buttonApin = 9;
9  int val = 0;                // si userà val per conservare lo stato del pin di input
10 int vecchio_val = 0;         // si userà vecchio_val per conservare lo stato del pin di input al passo precedente
11 int stato = 0;               // ricorda lo stato in cui si trova il led, stato = 0 led spento, stato = 1 led acceso
12
13 void setup() {
14   pinMode(ledPin, OUTPUT);             // il pin del led è OUTPUT
15   pinMode(buttonApin, INPUT_PULLUP);   // i pin del pulsante è INPUT e usando
16                                        // la resistenza interna va a +5V
17 }
18
19 void loop() {
20   val = digitalRead(buttonApin);  // legge il valore dell'input e lo conserva
21
22   if ((val == LOW) && (vecchio_val == HIGH)){   // controlla se è accaduto qualcosa
23     stato = 1 - stato;
24   }
25
26   vecchio_val = val;           // ricordiamo il valore precedente di val
27
28   if (stato == 1) {
29     digitalWrite(ledPin, HIGH);   // accende il led
30   }
31   else {
32     digitalWrite(ledPin, LOW);    //spegne il led
33   }
34 }
```

Alla riga 22 notiamo l'istruzione:

if ((val == LOW) && (vecchio_val == HIGH)) {

Questo blocco ***if*** controlla due condizioni (val == ***LOW***) e (vecchio_val == ***HIGH***) osservate la doppia e commerciale (***&***) che separa le due condizioni ***&&*** equivale all'operatore booleano ***and*** (e in italiano) cioè ***val*** deve essere ***LOW E vecchio_val*** deve essere ***HIGH*** quindi le condizioni devono essere vere entrambe allo stesso istante per eseguire le istruzioni successive.

Gli operatori Booleani principali sono:

- **&&** (doppia & commerciale) = **and** (e) la condizione scritta prima del segno e quella successiva devono essere entrambe vere
- || (doppia linea verticale) = **or** (oppure) una delle due condizioni scritte prima e dopo deve essere vera (anche entrambe ovviamente)
- **!** (punto esclamativo) = **not** (negazione) vero se la condizione in esame è falsa

alla riga 23 l'istruzione stato = 1 – stato prevede di eseguire l'operazione (1- stato) e di assegnare il nuovo valore alla stessa variabile (stato =)
quindi al primo ciclo stato vale 0 e quindi l'operazione 1-0 = 1 per cui ora stato varrà 1
al prossimo passaggio sarà 1 – 1 = 0 per cui ora stato varrà 0 … e così via.

In questo modo otteniamo un funzionamento abbastanza regolare anche se a volte può ancora succedere qualche errore.
Per risolvere completamente il problema dobbiamo tener conto anche del tempo durante il quale possono verificarsi i rimbalzi ed apportare ancora una modifica al programma, introduciamo quindi una nuova variabile contenente il tempo in millisecondi che devono trascorrere prima di considerare valido un nuovo contatto del pulsante, tipicamente si usa un valore da 50 a 200 millisecondi. Inoltre ho semplificato la gestione della variabile stato che è usata per accendere e spegnere il led usando l'operatore booleano **not** (**!**) e qui la riga 27 significa che il valore da assegnare alla variabile stato è l'opposto del valore corrente.
Ed ecco qui lo sketch finale

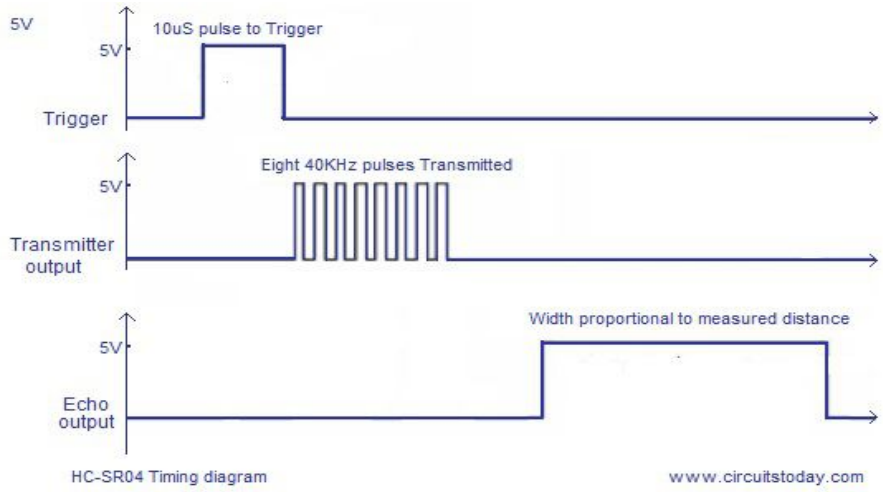
```
1  /* 1 led 1 pulsanti versione 3
2   * versione antirimbalzo
3   * il led è collegato al pin 12
4   * il pulsante al pin 9
5   * l'altro contatto è verso massa
6   */
7  int ledPin = 12;
8  int buttonApin = 9;
9  int val = 0;                  // si userà val per conservare lo stato del pin di input
10 int vecchio_val = 0;          // si userà vecchio_val per conservare lo stato del pin di input al passo precedente
11 int stato = LOW;              // ricorda lo stato in cui si trova il led, stato = low led spento, stato = high led acceso
12 int rimbalzo = 150;           // tempo di rimbalzo in millisecondi
13 unsigned long tempo = 0;      // memoria tempo dell'ultima attivazione del pulsante
14
15 void setup() {
16   pinMode(ledPin, OUTPUT);              // il pin del led è OUTPUT
17   pinMode(buttonApin, INPUT_PULLUP);    // i pin del pulsante è INPUT e usando
18                                         // la resistenza interna va a +5V
19 }
20
21 void loop(){
22   val = digitalRead(buttonApin);       // Leggo il valore in ingresso dato dal pulsante
23
24   if (val == LOW && vecchio_val == HIGH && millis() - tempo > rimbalzo) { //Se abbiamo premuto il pulsante (LOW)
25                                                                           //e la volta prima il suo stato era HIGH
26                                                                           //ed è trascorso il tempo necessario 150mill.
27     stato = !stato;                          // inverto lo stato
28     tempo = millis();                        // Ricorda quando l'ultima volta è stato premuto il pulsante
29   }
30   digitalWrite(ledPin, stato);               // Scrivo lo stato sul LED
31   vecchio_val = val;                         // salvo il valore attuale per il prossimo ciclo
32 }
```

Abbandoniamo per il momento i led e vediamo qualcuno dei moduli presenti nel kit. Cominciamo con il sensore ad ultrasuoni, nello specifico è il modulo hc-sr04, se digitate questo codice e fate una ricerca su internet, troverete migliaia di pagine che vi raccontano come usarlo. Questa è l'immagine del sensore, che è composto da due parti, un trasmettitore (T) e un ricevitore (R) a ultrasuoni e sono presenti quattro pin di collegamento che sono nell'ordine:

Vcc +5V
Trig attivatore dell'invio del segnale ultrasonico
Echo riceve il segnale di eco
Gnd 0V/massa

Le specifiche dicono che lavora in un range che va dai 2 cm ai 4 metri, la precisione della rilevazione non è soddisfacente anche se non può essere impiegato per progetti critici.

Esaminiamo nel dettaglio il funzionamento del sensore: Per iniziare la misurazione, bisogna portare allo stato ***LOW*** i pin ***Trig*** ed ***Echo*** quindi portare ad ***HIGH*** (+5V) il pin ***Trig*** per 10 microsecondi e poi riportarlo alla stato ***LOW*** (0V), in questo modo il trasmettitore invia un treno di 8 impulsi ultrasonici e fa partire un timer interno, se il segnale incontra un ostacolo, viene riflesso e ricevuto dal pin ***Echo*** che immediatamente va allo stato ***HIGH*** e ci rimane per un tempo proporzionale alla distanza percorsa dall'impulso



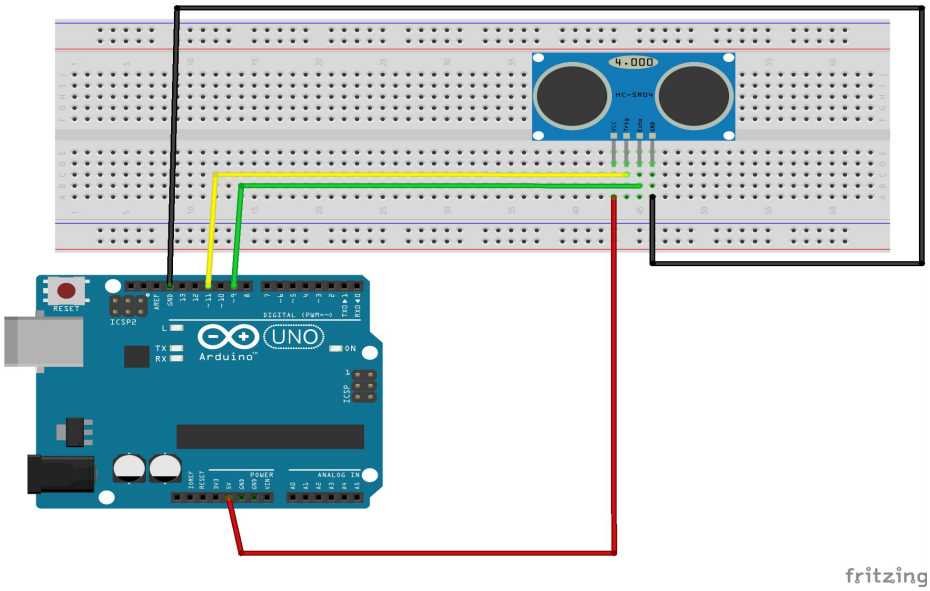
Ecco un esempio di diagramma dei tempi del HC-SR04, che trovate anche sul datasheet.

Quindi con una semplice formula matematica possiamo ricavare la distanza dell'ostacolo dal sensore:
Distanza ostacolo = (Tempo Echo HIGH x velocità del suono) / 2
la divisione per 2 in quanto il tempo misurato comprende l'andata e il ritorno del segnale
dove velocità del suono = 331,45 m/s a 0 °C e aumenta a seconda della temperatura in questo modo
a(T)= 331,45 + (062* T) m/s con T misurata in °C

T °C	a m/s
0	331,45
10	337,65
15	340,75
20	343,85
25	346,95
30	350,05

Noi abbiamo un segnale che ha una durata misurabile in microsecondi (milionesimi di secondo)
e distanze che è più pratico misurare in centimetri quindi dividiamo la cifra ricavata dalla formula per 20.000 e la moltiplichiamo per il tempo di Echo per avere direttamente la misura in cm.

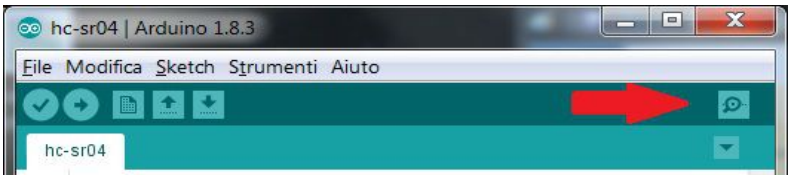
Realizziamo quindi il nostro circuito che è molto semplice in quanto il sensore non ha bisogno di nessun altro componente per funzionare



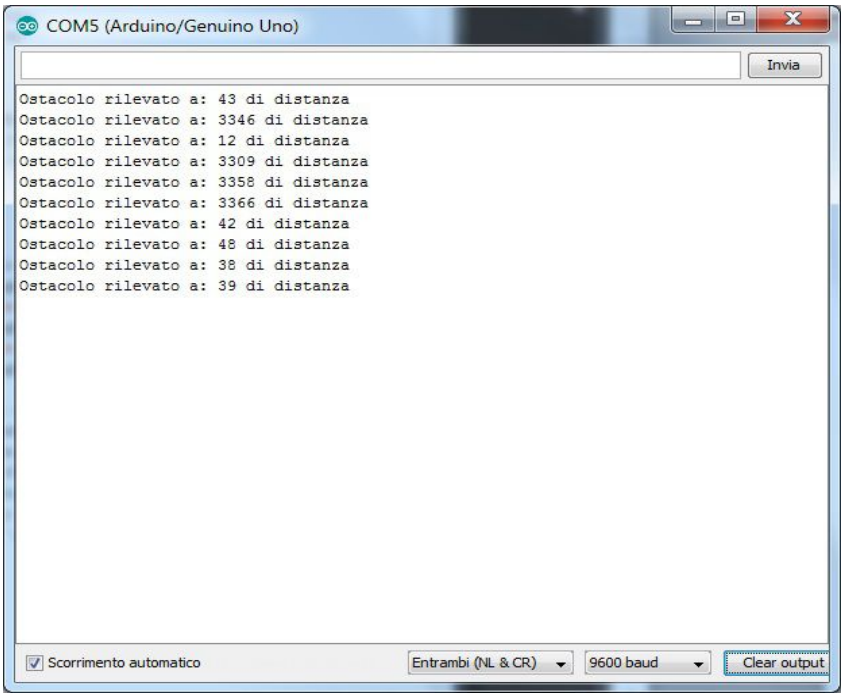
e scriviamo il nostro sketch

```
/* sensore ultrasuoni HC-SR04
 *  programma di test funzionamento
 *  distanza= (tempo segnale echo high/2) *velocità del suono (345,09m/s a 22 °C)
 *  echo collegato al pin 9
 *  trig collegato al pin 11
 *
*/

//definiamo i pin a cui siamo collegati
const int trigPin = 11;
const int echoPin = 9;
float durata, cm;

void setup() {
    Serial.begin(9600);
    pinMode(trigPin,OUTPUT);
    pinMode(echoPin,INPUT);
}

void loop() {
    //richiesta invio impulsi al sensore
    digitalWrite(trigPin,LOW);
    delayMicroseconds(2);
    digitalWrite(trigPin,HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin,LOW);

    //restituisce il tempo di echo
    durata=pulseIn(echoPin,HIGH);

    //trasformo in cm
    cm=durata/20000.0*345.09;

    //scrivo sul monitor seriale
    Serial.print("Ostacolo rilevato a: ");
    Serial.print(cm);
    Serial.println(" di distanza");
    delay(1000);
}
```

Qui abbiamo l'occasione di utilizzare una nuova funzionalità dell'ambiente di sviluppo Arduino, quella che si chiama “interfaccia monitor seriale”, il monitor seriale ci consente di comunicare con le periferiche collegate appunto alle porte seriali, nel nostro caso la porta è la Comx alla quale è connessa la nostra scheda. Per attivarla dobbiamo inserire il comando ***Serial.begin(velocità_di_connessione)***, il parametro velocità di connessione deve essere uguale a quello impostato nella finestra del monitor seriale che si apre cliccando sul pulsante con la lente di ingrandimento in alto a destra nella finestra di Arduino



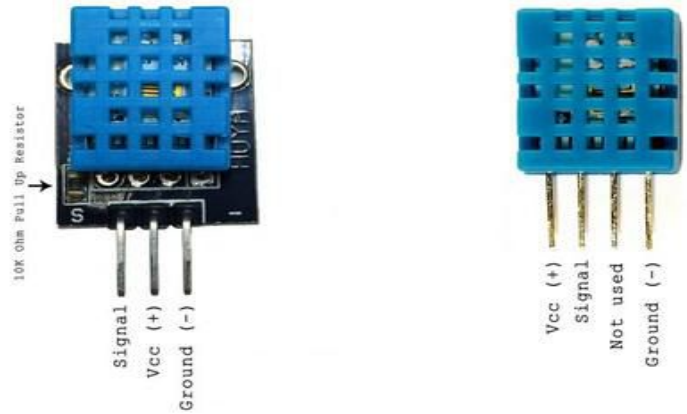
si aprirà la finestra relativa alla porta connessa:



in basso a destra abbiamo la possibilità di impostare la velocità di comunicazione, al momento non ci interessa usare velocità alte per cui la impostiamo a 9600 baud e sarà appunto questo numero che metteremo come velocità di connessione.
Ho inserito le variabili durata e cm come ***float*** cioè numeri contenenti una parte decimale per poter effettuare dei calcoli più precisi.
Dopo aver impostato i pin di ***Trig*** come uscita ed ***Echo*** come ingresso, il setup è concluso. Nel loop inseriamo i comandi come abbiamo visto precedentemente per effettuare la richiesta di emissione del segnale ad ultrasuoni e cioè: partiamo con il ***Trig*** a livello basso poi lo portiamo a +5V per 10 microsecondi e poi di nuovo a livello basso. A questo punto usiamo l'istruzione ***pulseIn(pin, STATO)***, pulseIn si mette in attesa che sul pin indicato ci sia una transizione di livello come indicato da ***STATO***, se ***HIGH*** da basso ad alto e se ***LOW*** da alto a basso e quando avviene la transizione fa partite un timer interno e misura, in microsecondi, il tempo che passa prima che lo stato ritorni quello precedente.
A questo punto effettuiamo il calcolo matematico che ci da i centimetri di distanza del bersaglio e attraverso il comando ***Serial.print(“testo da scrivere”)*** della porta seriale, scriviamo il dato sulla finestra

del monitor. Il comando ***Serial.println("")*** aggiunge alla stampa un carattere di ritorno a capo cioè qualsiasi cosa verrà stampata successivamente, comincerà su una nuova linea. Attendiamo un secondo e poi ricominciamo il ciclo.
Un'altra istruzione nuova che usiamo qui è: ***delayMicroseconds(tempo)*** come si intuisce dal nome è un comando di attesa che però come parametro usa i microsecondi (secondi $*10^{-6}$).

Aggiungiamo un altro sensore in modo da aumentare la precisione delle nostre misurazioni, abbiamo detto che la velocità del suono dipende dalla temperatura ed allora proviamo a controllarla con il sensore DHT11, che ci permette di rilevare temperatura ed umidità dell'ambiente, si tratta di un sensore resistivo di umidità e da un sensore NTC (negative temperature coefficient), cioè un termistore dove la resistenza cala con l'aumentare della temperatura



Questa è l'immagine di due versioni dello stesso sensore, a destra la versione base per usare la quale dovremo aggiungere una resistenza da 4,7K Ohm tra il piedino Signal e Vcc, e la versione a sinistra con un piccolo circuitino con la resistenza già inclusa e quindi pronto all'uso, notare che i contatti sono stati ridotti a tre in quanto il quarto non risulta collegato a nulla e i contatti Signal e Vcc sono stati invertiti cosicché anche invertendolo non lo mandiamo in cortocircuito. Realizziamo il seguente circuito collegando il sensore temperatura all'ingresso 2 di Arduino