**UNIVERSITÀ DI BRESCIA**
FACOLTÀ DI INGEGNERIA
Dipartimento di Elettronica per l'Automazione

**Laboratorio di Robotica Avanzata**
**Advanced Robotics Laboratory**

Corso di Robotica
(Prof. Riccardo Cassinis)

# Dimostratore funzionamento sonar con Arduino

**Elaborato di esame di:**

**Elisa Lazzari, Massimo Pellini.**

Consegnato il: **25 Gennaio 2013**

# Sommario

*Il lavoro svolto consiste nel pilotaggio di un sonar tramite computer, utilizzando il dispositivo Arduino Uno. Nello specifico sono stati collegati ad una scheda Shield, collegata ad Arduino Uno, una scheda sonar, un servomotore, un sensore di temperatura, uno speaker e sette led; successivamente sono stati scritti i software di controllo per Arduino Uno e per il PC. Arduino Uno acquisisce una serie di parametri per il pilotaggio del sonar e del servomotore, che andranno quindi a rilevare la presenza e la posizione di ostacoli.*

# 1. Introduzione

Una delle tipologie di sensori più utili nella robotica sono senz'altro i sensori di prossimità, ovvero tutti quei dispositivi in grado di rilevare un ostacolo posto sulla traiettoria di un robot.

L'obiettivo di questo progetto consiste nel collegare uno di questi sensori, ossia un modulo sonar Devantech SRF04, ad un computer, per poterlo successivamente controllare tramite quest'ultimo. Per realizzare il circuito di pilotaggio ci si è serviti di un microcontrollore, integrato nella periferica Arduino Uno, utilizzato per pilotare direttamente il modulo sonar e ricevere i dati rilevati, il quale a sua volta viene controllato dall'utente tramite il software presente sul computer.

Schematicamente la catena di pilotaggio realizzata è dunque la seguente:


**Fig. 1 - Schema generale**

Il pilotaggio della scheda sonar tramite Arduino Uno ci pone di fronte a diversi problemi da risolvere:

- individuare i compiti richiesti al sonar;
- identificare i segnali di controllo richiesti dalla scheda sonar per funzionare e far generare opportune combinazioni di questi ultimi al PC per pilotarla;
- intercettare i segnali di output della scheda sonar e del sensore di temperatura ed elaborarli tramite il PC;
- collegare e comandare il motore;
- produrre un output a video delle rilevazioni effettuate.

# 2. I componenti utilizzati

## 2.1. Il sonar

Prima dello sviluppo del sonar SRF04, in commercio era presente il modulo ad ultrasuoni Polaroid Ranging, il quale tuttavia presenta un certo numero di svantaggi per l'utilizzo in piccoli robot e simili applicazioni:

1. un consumo di corrente di 2,5 Ampere durante il picco sonoro è eccessivo;
2. anche una corrente a riposo di 150mA è troppo elevata;
3. una distanza minima di rilevazione di 26 centimetri è inadeguata;
4. il modulo è troppo grande per entrare in sistemi di piccole dimensioni.

L'SRF04 è stato progettato per essere altrettanto facile da utilizzare come il sonar Polaroid: anch'esso richiede un breve impulso di trigger e fornisce un impulso di echo. Il controller deve solo cronometrare la durata di questo impulso per calcolare la distanza a cui si trova l'ostacolo rilevato. La piedinatura del modulo SRF04 è riportata nella figura qui di seguito:

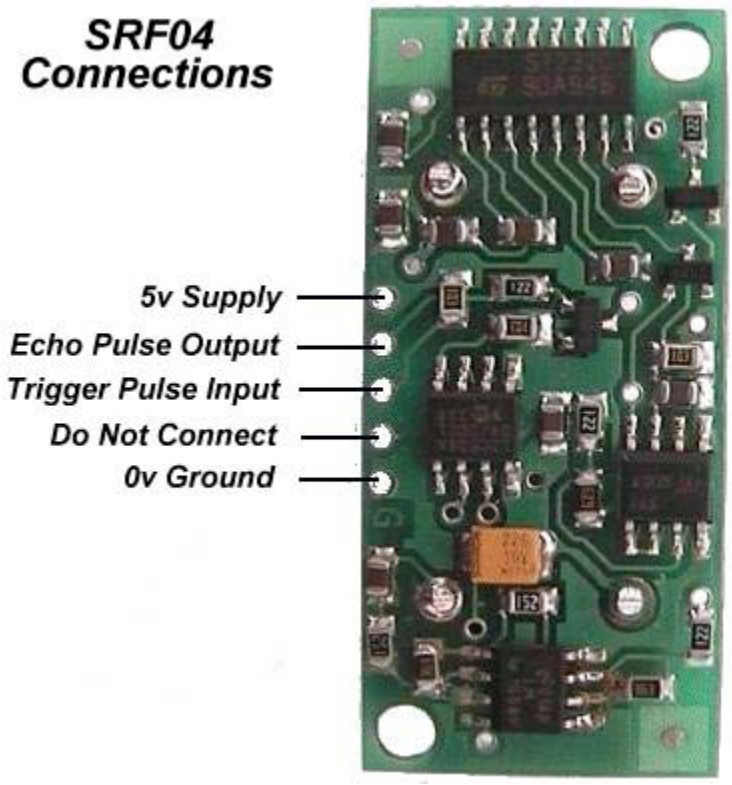



**Fig. 2 - Circuito stampato del SRF04**

Il diagramma dei segnali con relative temporizzazioni è mostrato nella figura 3. Basta fornire un impulso di 10µS al pin "Trigger Pulse Input" per avviare la rilevazione. Il sonar invierà una sequenza di 8 impulsi a 40 kHz (quindi ultrasuoni) e successivamente alzerà il pin di uscita "Echo Pulse Output". Quindi

attende un'eco, e non appena la rileva abbassa il livello del pin di uscita. L'impulso che si andrà a rilevare sul pin di uscita è quindi di durata proporzionale alla distanza dall'oggetto rilevato. Cronometrando la durata dell'impulso è possibile calcolare la distanza dell'oggetto rilevato in pollici, centimetri o qualsiasi altra unità di misura.
Se non viene rilevato alcun oggetto, il sonar abbasserà il pin "Echo Pulse Output" in ogni caso dopo circa 36 ms.

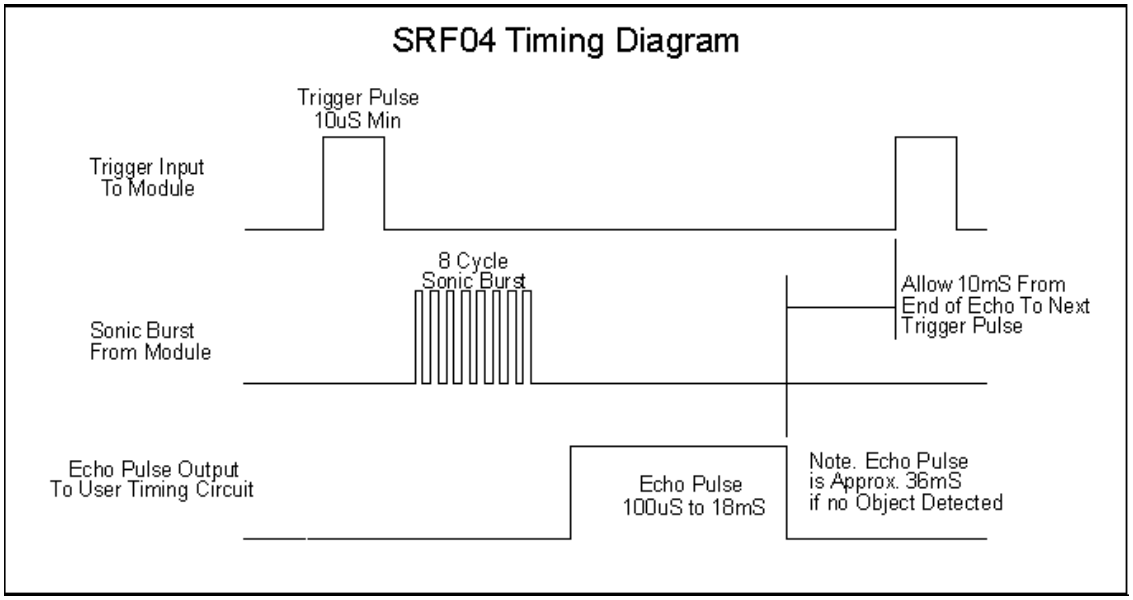



**Fig. 3 - SRF04 Timing diagram**

In appendice è possibile vedere lo schema elettrico della scheda SRF04.

Il circuito è progettato per avere un basso costo. Esso utilizza un PIC12C508 per gestire le funzioni di controllo ed utilizza trasduttori piezoelettrici standard a frequenze di 40 kHz. Il pilotaggio del trasduttore trasmittente potrebbe essere gestito direttamente dal PIC. Un pilotaggio a 5Vpuò essere utile per oggetti grandi, ma può dare problemi nel rilevamento di oggetti più piccoli. Il trasduttore è in grado di gestire tensioni di pilotaggio dell'ordine di 20V, così nel sonar SRF04 è stato deciso di utilizzare una tensione vicina a tale livello. Nella fattispecie si utilizza una segnale di pilotaggio a 16V, forniti dal componente MAX232.

Il ricevitore è un classico circuito amplificatore operazionale a due stadi. Il condensatore C8 in ingresso blocca alcuni residui di corrente continua che sembrano sempre essere presenti. Ogni stadio di guadagno è impostato su 24 per un guadagno totale di 576. L'uscita del circuito amplificatore viene mandata in un comparatore LM311.

Per ottenere il funzionamento fino a 1-2cm bisogna evitare l'accoppiamento diretto fra il trasmettitore e il ricevitore, che è proprio accanto ad esso. A peggiorare le cose, il trasduttore piezoelettrico è un oggetto meccanico che continua ad oscillare per un certo tempo (fino a 1mS) dopo che il segnale di pilotaggio è stato rimosso. E 'molto più difficile capire la differenza tra questo accoppiamento fra i trasduttori e un eco di ritorno: per questo motivo molti progetti, tra cui il modulo Polaroid, semplicemente azzerano, trascurano, questo periodo. Guardando il segnale dell'eco di ritorno su un oscilloscopio si nota che esso è molto più grande in ampiezza rispetto all'ampiezza del segnale dato dall'accoppiamento dei due trasduttori. È sufficiente quindi regolare la soglia di rilevamento durante questo tempo in modo che solo l'eco sia rilevabile. Il condensatore C10 da 100nF viene caricato con -6V durante la sequenza di 8 impulsi a 40KHz. Questo poi si scarica abbastanza rapidamente attraverso la resistenza R6 da 10kΩ, ripristinando la sensibilità originaria per poter rilevare gli echi più lontani.

Una tensione negativa conveniente per gli amplificatori operazionali e per il comparatore viene generata dal componente MAX232; purtroppo però, quest'ultimo genera anche del rumore ad alta frequenza. Per ovviare a questo problema tale componente viene spento mentre si rimane in ascolto di eventuali segnali di eco. Il condensatore C9 da 10uF serve appunto per questo scopo: disattivare il componente MAX232 per il tempo necessario ad ascoltare gli eventuali echi di ritorno.

Durante il funzionamento, il processore attende un impulso in ingresso sul pin "Trigger Pulse Input" simile a quello rappresentato nel grafico in figura 3. Quindi emette dal trasmettitore otto impulsi a 40 kHz. Quando la linea "Echo Pulse Output" viene poi posta a livello alto, questo è il segnale per il processore di avviare il cronometro. L'innalzamento di tale linea comporta inoltre lo spegnimento del dispositivo MAX232. Dopo un certo lasso di tempo - non più di 10 - 12ms normalmente - l'eco di ritorno viene rilevato e il PIC abbasserà la linea "Echo Pulse Output". La durata di questo impulso positivo rappresenta il tempo di volo del segnale sonoro. Se non viene rilevato alcun segnale di eco, allora sarà automaticamente decretato un time-out dopo circa 36mS.

Inoltre a causa dello spegnimento del MAX232 durante il rilevamento di eventuali echi, è necessario attendere almeno 10mS tra due cicli di misura successivi.
Il raggio d'azione massimo di tale sonar è un po' più di 3m, mentre l'assorbimento medio di corrente è meno di 50mA e tipicamente di circa 30mA.

La scheda sonar misura l'intervallo di tempo impiegato da ogni singolo impulso a percorrere la doppia distanza tra l'oggetto rilevato ed il sonar. Conoscendo il tempo dopo cui giunge questo eco e la velocità del suono nell'aria, è possibile calcolare con precisione la distanza dell'oggetto rilevato, come si vedrà nei prossimi paragrafi.

Va sottolineato, che non vi è alcun modo semplice per ridurre o modificare la larghezza del fascio di impulsi sonar. Il fascio di impulsi del sonar è di forma conica; la larghezza del fascio è in funzione della superficie dei trasduttori ed è fissa. Nel grafico sottostante (figura 4) si può vedere un dettaglio del fascio dei trasduttori impiegati in questo modulo, preso dal datasheet del sonar fornito dal fabbricante.

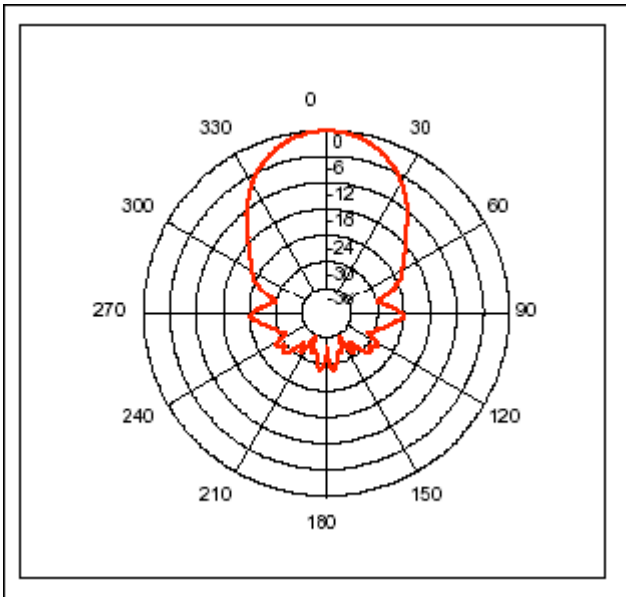



**Fig. 4 - Fascio dei trasduttori**

Il pin "Do Not Connect" generalmente non è collegato. Tuttavia, qualora lo si collegasse a massa, l'effetto di tale connessione è quello di inserire un ritardo di 300µS tra la fine dell'impulso ricevuto sul

pin “Trigger Pulse Input” e l’invio degli 8 impulsi utilizzati per la scansione dello spazio antistante il sonar.

## 2.2. Arduino Uno

Arduino Uno è una scheda microcontrollore basata su ATmega328. È dotata di 14 pin I/O digitali (sei dei quali utilizzabili come uscite PWM), sei entrate analogiche, oscillatore a cristallo da 16 MHz, una porta USB, una presa di alimentazione, una connessione ICSP e un pulsante di reset.


**Fig. 5 - Arduino Uno**

Dispone di tutto il necessario per gestire il microcontrollore a bordo; per utilizzarla basta connettere la scheda al computer con un cavo USB o alimentarla con un alimentatore (max 12V) o delle batterie esterne.

In Appendice A si trova lo schema elettrico del dispositivo, per informazioni più dettagliate si consiglia di consultare il relativo datasheet scaricabile direttamente dal sito http://www.arduino.cc.

### 2.2.1. Specifiche tecniche

| | |
|---|---|
| Microcontrollore | ATmega328 |
| Tensione di funzionamento | 5V |
| Tensione di Alimentazione (raccomandata) | 7-12V |
| Massima Tensione supportata (non raccomandata) | 20V |
| I/O digitali | 14 (6 dei quali con uscita PWM) |
| ingressi analogici | 6 |
| Corrente in uscita per I/O Pin | 40 mA |
| Corrente in uscita per 3.3V Pin | 50 mA |
| Memoria Flash | 32 KB (ATmega328) di cui 0.5 KB usata bootloader |
| SRAM | 2 KB (ATmega328) |
| EEPROM | 1 KB (ATmega328) |
| Velocità di clock | 16 MHz |

## 2.3. Il servomotore

Il servomotore è un motore con delle caratteristiche particolari. È molto usato nel modellismo. Esso è costituito da un motore completo di riduzione meccanica, un sistema di feedback per la posizione dell'asse di uscita e tutta l'elettronica di controllo racchiusi in un unico contenitore.

**Fig. 6 - Servomotore**

Tramite un sistema di comando opportuno, che sfrutta un segnale PWM (si vedano i capitoli 3 e 4 a tal proposito) è possibile far ruotare l'asse di uscita del servo, posizionarlo e mantenerlo nella posizione voluta.

I diversi tipi di servomotori si differenziano in funzione della forza sviluppata e dalla qualità dei componenti meccanici utilizzati nella costruzione, nonché per la capacità di ruotare su un arco di 360° o solo di 180°.

La parte meccanica del servomotore altro non è che una serie di ingranaggi che ha lo scopo di ridurre il numero dei giri del motore ad un valore utilizzabile dal dispositivo che dobbiamo comandare.

Il servomotore utilizzato in questo progetto è in grado di operare su un fronte di 180°.

## 2.4. Il sensore di temperatura LM35DZ

Il componente LM35DZ è un sensore di temperatura di precisione, con uscita lineare, in voltaggio, proporzionale alla temperatura rilevata in scala Celsius.

**Fig. 7 - Il sensore LM35DZ**

Con questo integrato non è necessario affrontare dei calcoli per avere la temperatura effettiva in gradi centigradi dalla scala Kelvin. La precisione è di 1/4 di grado, il range di funzionamento varia da -55 a +150 gradi centigradi.

Il consumo di tale componente si attesta al massimo sui 60 mA, ed opera da 4 a 30 volt.

La bassa impedenza di uscita e l'uscita lineare rendono questo sensore particolarmente semplice da utilizzare, sia con microcontrollori che con strumenti di lettura.

### 2.4.1. Specifiche tecniche

- Calibrato direttamente in ° Celsius (Centigradi)
- Scala lineare + 10.0 mV/°C
- Accuratezza garantibile al 0.5°C (a +25°C)
- Range misurabile 0° to +100°C
- Adatto ad applicazioni remote
- Basso costo
- Opera da 4 a 30 Volt
- Meno di 60 μA di consumo
- Non linearità tipica soltanto±0.25°C

Per ulteriori informazioni, si consiglia di consultare il datasheet scaricabile dalla rete.

## 2.5. Speaker

È stato montato sul dispositivo uno speaker, il cui scopo è di emettere un segnale sonoro ogni qualvolta viene emesso un impulso sonar. Per fornire all'utilizzatore tale riscontro l'altoparlante emetterà un segnale udibile a 3,5 KHz per 50 ms.

Per ottenere la frequenza ottimale dello speaker, questo è stato collegato ad un impedenzimetro (4194A Impedence /Gain-Phase Analizer della Hewlett Packard); dal grafico ottenuto, si è osservata la frequenza di funzionamento ottimale, pari appunto a 3,5 KHz.

# 3. Problematiche affrontate

Al fine di analizzare le problematiche affrontate in questo progetto, si comincia riprendendo brevemente la situazione iniziale: si utilizza una scheda sonar SRF04 da pilotare attraverso un Personal Computer. Per far ciò verrà utilizzata una periferica Arduino e un servomotore, quest'ultimo al fine di realizzare una periferica che sia in grado di scansionare un orizzonte di 180° . Quindi in sostanza si utilizzeranno 4 dispositivi principali da interfacciare fra loro: il sonar, Arduino, il servomotore e un PC.

Per quanto riguarda la scheda sonar, questa presenta 5 pin, di cui due di alimentazione e tre utilizzati per pilotarla. Come si vede dalla figura 2 nel paragrafo 2.1, questi tre pin sono:

- Echo Pulse Output
- Trigger Pulse Input
- Do Not Connect

I pin "Trigger Pulse Input" e "Do Not Connect" richiedono in ingresso, mentre il pin "Echo Pulse Output" fornisce in uscita, dei segnali elettrici digitali. Quindi per pilotare la scheda sonar occorrerà sostanzialmente operare una gestione di tali segnali (si veda paragramo 2.1 per le specifiche di tali segnali).

Discorso simile si può fare per la gestione del servomotore, in particolare della sua posizione. La posizione in cui deve portarsi è comunicata al motore tramite un segnale PWM (Pulse Width Modulation); il segnale di comando è costituito infatti da un'onda rettangolare inviata ripetutamente: l'impulso deve avere una durata compresa tra 1 e 2 millisecondi (tra 0,5 ms e 2,5 ms per i servomotori che ruotano di 180°, come quello usato) e il periodo dell'onda deve essere di circa 20 ms (corrispondenti ad una frequenza di 50 Hz). Il segnale fatto in questo modo (un segnale PWM appunto) deve essere inviato di continuo se si vuole che il servocomando, sotto sforzo, mantenga la posizione desiderata. Ad esempio, se si vuole che il motore mantenga la posizione centrale si dovrà inviare un segnale PWM il cui impulso abbia ampiezza di 1,5 ms.

Assunta la gestione delle comunicazioni fra scheda sonar e Arduino e fra motore e Arduino tramite i segnali digitali o PWM sopra menzionati, si è passati alla gestione della comunicazione fra Arduino e il PC. Il framework Arduino si basa sul microcontroller AtMega, il quale è progettato per comunicare con altre periferiche (o con il PC in questo caso) mediante l'invio di segnali seriali. Di conseguenza la comunicazione fra Arduino e il PC dovrà essere una comunicazione di tipo seriale.

Si torni ora al motore. Un punto molto importante da considerare è quanto consuma il motore quando è sotto sforzo, in quanto se il motore sotto sforzo richiede più corrente di quella che le porte di Arduino sono in grado di fornire, si rischia di bruciarle. Per controllare i consumi del motore sotto sforzo, quest'ultimo è stato collegato in serie fra un generatore di tensione (impostato a 5V, lo stesso voltaggio fornito dalle porte 5V di Arduino) e un amperometro, quest'ultimo collegato a massa. Al motore è stato collegato un carico che simulasse il carico di lavoro dato dalla scheda sonar e dal cavo di collegamento con Arduino. Dai test successivi eseguiti sono stati riscontrati dei picchi di consumo di 900 mA; dato che le porte 5V di Arduino forniscono in uscita solo 40 mA (si veda il paragrafo 2.2.1), si è compresa la necessità di un'alimentazione esterna per il motore.

Un'altra situazione meritevole di interesse si presenta quando si instaura una nuova comunicazione seriale fra Arduino e il PC. Infatti, secondo le sue specifiche di progettazione, Arduino esegue un reset software ogni volta che rileva l'instaurazione di una nuova comunicazione seriale. Per ovviare a questo comportamento indesiderato, come si vedrà nel prossimo paragrafo, si utilizzerà un condensatore al tantalio da 22 μF.

Altro aspetto considerato è la differente velocità di propagazione del suono nell'aria. Come noto dalla fisica, ad una temperatura di 20°C il suono viaggia ad una velocità di circa 340 m/s. Più in generale la velocità del suono nell'aria varia secondo la seguente relazione:

$$(331,5 + 0,6 * T)$$

ove T è la temperatura dell'ambiente in °C, 331,5 m/s è la velocità del suono a 0°C e 0,6 rappresenta il fattore di incremento della velocità per ogni grado di temperatura. Ci occorre quindi un sensore di temperatura per rilevare la temperatura dell'ambiente in cui opera la periferica, al fine di operare tale compensazione.

Infine, si è pensato ad un sistema per fornire una visualizzazione immediata dello stato della periferica all'utente, in modo tale che l'utente sappia riconoscere a colpo d'occhio semplicemente guardando la periferica alcune informazioni di base su di essa: queste informazioni riguardano l'alimentazione della periferica e del motore, lo stato della linea (ossia se si stanno trasmettendo o ricevendo dati), lo stato della periferica (ossia se è in attesa di comandi oppure se sta eseguendo una scansione) e infine viene visualizzato se la modalità programmabile della periferica è abilitata o meno.

# 4. La soluzione adottata

Verrà ora spiegato come sono state risolte le problematiche elencate al punto precedente.

Essendo la scheda sonar alimentata direttamente da Arduino attraverso le sue porte 5V e GND e pilotata attraverso segnali elettrici digitali, tutti i 5 pin della scheda SRF04 sono collegati mediante un cavo flessibile alle opportune porte dello Shield di Arduino. Nella fattispecie alimentazione e massa, come detto, sono collegate alle porte 5V e GND, mente i pin "Echo Pulse Output", "Trigger Pulse Input" e "Do Not Connect" sono collegati ai digital pin 11, 10 e 9 di Arduino (passando come detto attraverso lo Shield). In realtà, al fine di rendere più flessibile la struttura e poter agevolmente scollegare la scheda sonar da Arduino, il cavo flessibile in uscita dalla scheda sonar è stato collegato allo Shield mediante dei connettori.

Stessa soluzione è stata adottata anche per il motore. Richiedendo il segnale di controllo del motore un'onda PWM, il suo pin di controllo è stato collegato al digital pin 5 di Arduino (uno dei pin progettati per supportare un'onda PWM). Inoltre, come detto nel paragrafo precedente, si è appurata la necessità di un'alimentazione esterna per sostenere le richieste di amperaggio del motore sotto sforzo. in questo caso i cavetti di alimentazione e massa provenienti dall'alimentatore esterno sono stati saldati direttamente ad un connettore sullo Shield, assieme al cavetto proveniente dal digital pin 5 per il controllo del motore; a questo connettore poi è stato collegato direttamente il connettore femmina di cui è provvisto di fabbrica il cavo del motore.

Al fine di incrementare il livello di protezione dei componenti del circuito e di fornire all'utente la possibilità di aprire o chiudere a piacimento l'alimentazione del motore con un interruttore, quest'ultimo è stato posto fra il cavetto Vcc in uscita dall'alimentazione esterna e il connettore sullo Shield.

Si torni per un momento alla gestione del motore tramite segnale PWM. Come detto nel paragrafo precedente, la posizione che deve assumere viene comunicata al motore modulando la durata dell'impulso del segnale PWM. Per la gestione del motore abbiamo utilizzato una libreria fornita direttamente da Arduino, Servo.h, Essa contiene una serie di metodi scritti appositamente per gestire i servomotori in maniera semplice ed intuitiva: ad esempio tramite il metodo Servo.write(<posizione>) si può impostare la direzione in cui posizionare il motore direttamente in gradi. Saranno poi i metodi ad occuparsi in automatico della conversione del comando richiesto in un segnale PWM opportunamente modulato.

In merito alla comunicazione fra Arduino e PC, a livello concettuale tutte le schede vengono programmate attraverso una porta seriale RS-232, ma il modo in cui questa funzionalità è implementata nell'hardware varia da versione a versione. Le versioni più recenti di Arduino vengono gestite via USB, grazie a un'implementazione che usa un chip adattatore USB-seriale come l'FT232 della FTDI. In questo caso il problema riscontrato è stata l'assenza di librerie in Java per la gestione della comunicazione seriale: versioni precedenti di Java prevedevano le librerie javax.comm, librerie che sono state deprecate nelle versioni più recenti. Sono quindi state scaricate ed installate delle librerie specifiche (librerie RxTx versione 2.1) per la gestione di questo tipo di comunicazione.

Successivamente è stato affrontato il discorso del reset automatico del software di Arduino, reset che avviene ogni qualvolta si instaura una nuova comunicazione seriale. Questo comportamento indesiderato è stato risolto collegando fra loro i pin 5V e RESET di Arduino mediante un condensatore al tantalio da 22 μF. Ogni volta che si avvia una nuova comunicazione seriale infatti, sul pin RESET si può rilevare un calo di tensione che, in base alle specifiche di progettazione di Arduino, causa il reset software del dispositivo. Tramite questo condensatore si evita questo calo di tensione e si impedisce il reset della periferica. Tuttavia bloccando in questo modo il reset automatico di Arduino si impedisce di fatto di poter caricare del nuovo software su di esso. Quindi nei momenti in cui si vuole caricare del nuovo software sul dispositivo, bisogna scollegare questo condensatore: a tal fine esso è stato collegato in serie con un interruttore che permette di collegarlo o scollegarlo con una semplice azione.

Si è inoltre accennato alla differente velocità di propagazione del suono nell'aria a seconda della temperatura. Per risolvere questo punto, appena dietro la scheda sonar è stato montato un trasduttore di temperatura LM35, collegato anch'esso ad Arduino tramite il medesimo cavo flessibile e connettori usati per la scheda sonar. In particolare il sensore di temperatura ha tre pin: i due laterali sono quelli di alimentazio-

ne e massa (collegati come si può intuire a 5V e GND di Arduino, sempre passando per lo Shield, il connettore e il cavo flessibile), mentre quello centrale è il pin di output, che fornisce una tensione proporzionale alla temperatura ambiente (si vedano il paragrafo 2.4 o il datasheet del componente per maggiori informazioni a riguardo). Questo segnale è stato collegato all'analog pin 0 (A0) di Arduino.

Resta da discutere della soluzione adottata per mostrare al'utente lo stato della periferica. Come si può intuire tale funzionalità è stata realizzata mediante l'impiego di 7 LED di differente colore. Osservando la parte superiore della periferica (v. Appendice 11.6) si possono notare tali diodi, disposti su 4 file. La prima fila presenta due LED, uno verde e uno giallo, che mostrano lo stato della periferica: se è acceso il LED giallo significa che la periferica è in attesa di un comando e non sta svolgendo alcun lavoro, se invece è acceso quello verde significa che la periferica sta svolgendo una qualche scansione. Sulla seconda fila vi sono due LED blu, che si accendono rispettivamente quando la periferica sta inviando o ricevendo dei dati dal PC sul cavo USB. La terza fila di LED presenta i LED di alimentazione: quello più a sinistra è acceso quando Arduino (e di conseguenza il sonar e il trasduttore di temperatura sono alimentati), quello di destra si accende invece quando l'interruttore del circuito di alimentazione del motore è chiuso (cioè quando il motore è alimentato). Infine vi è la quarta fila. Essa in realtà presenta due interruttori: quello più a destra è quello del circuito di alimentazione del motore. Quello a sinistra invece è l'interruttore di esclusione del condensatore. Quando il condensatore è escluso il LED rosso al suo fianco si accende e Arduino si trova in modalità programmabile. Per disattivare la modalità programmabile basta azionare nuovamente l'interruttore, ristabilendo il collegamento col condensatore. Come indicato nel paragrafo 2.5, i LED devono essere attraversati da correnti dell'ordine dei 10 – 20 mA. Per garantire il rispetto di tali limiti, a ciascun diodo è stata anteposta una resistenza di 300Ω. Si veda il prossimo paragrafo per una visione d'insieme di tutti i componenti elettrici usati nella periferica e del corrispondente circuito elettrico completo.

# 5. Progettazione del circuito

In questo paragrafo si descrive il circuito elettrico costituente il cuore dell'intera periferica sonar, i cui schemi si trovano in appendice A.

Per prima cosa si osservi che i 7 diodi, gli interruttori e lo speaker sono componenti montati sul coperchio della scatola (o su un lato, nel caso dello speaker) in cui alloggia Arduino, mentre tutti gli altri componenti (le resistenze, Arduino, il condensatore e i cavetti di collegamento) sono tutti componenti che risiedono all'interno della scatola. Il motore, che non è fisicamente rappresentato in questo schema (ci si è limitati a rappresentare solo il suo cavo di collegamento), è appeso al di sotto del coperchio della scatola, in modo tale da consentire l'uscita del perno rotante, su cui poi sono montate la scheda sonar e il sensore di temperatura.

Procedendo nell'analisi dello schema dall'alto verso il basso, si osservino per prima cosa i due spezzoni di cavo nero, etichettati "cavo motore" e "cavo sonar". Essi rappresentano i due cavi flessibili che partono dallo Shield di Arduino e portano alimentazione e segnali di controllo/output rispettivamente al motore e al complesso sonar/trasduttore di temperatura montato sul motore stesso. I colori con cui sono stati rappresentati gli isolanti dei singoli cavetti interni ai due cavi dello schema rispecchiano esattamente i colori degli isolanti nei due cavi utilizzati nel progetto; in tal modo si vuole fornire all'utente una rappresentazione immediata e a prova d'errore dei collegamenti effettuati sul circuito reale, cosicchè quest'ultimo possa comprendere immediatamente la funzione alla quale è stato associato ciascuno cavetto. La stessa rappresentazione è riportata nel secondo schema, che mostra il collegamento di ciascun cavetto al rispettivo pin sulla scheda sonar o sul sensore di temperatura. Ad esempio, dagli schemi si evince come il cavetto giallo collegato al pin "Trigger Pulse Input" del modulo sonar sia collegato all'altra estremità del cavo al digital pin 10 di Arduino. Lo stesso collegamento, con cavetto del medesimo colore, si vedrà sul circuito reale: in tal modo l'utente sa che il segnale di trigger del sonar parte dal pin 10 di Arduino e viaggia sul cavetto giallo fino al sonar.

L'interruttore I1 è quello responsabile di abilitare e disabilitare la modalità programmazione, mentre l'interruttore I2 è l'interruttore di alimentazione del motore. Come si nota, I1 è un interruttore a due vie: se si attiva una via, si accende il LED rosso e si scollega il condensatore, abilitando di fatto la modalità programmazione, mentre attivando l'altra via si ottiene il risultato opposto (condensatore collegato, Led

spento e reset automatico bloccato, impedendo in tal modo successive riprogrammazioni di Arduino). I2 invece è un semplice interruttore a una via, che apre o chiude il circuito di alimentazione del motore.

Si osservi inoltre che, come spiegato nel paragrafo precedente, ogni LED collegato in serie ad una resistenza da 300Ω, al fine di limitare ad una decina di mA la corrente che transita nei diodi.

Si noti infine che è stato realizzato un riferimento di massa comune, collegando fra loro i vari pin GND forniti da Arduino e collegandoli con la massa dell'alimentazione esterna.

# 6. Protocollo di comunicazione

La scheda sonar come detto si pilota tramite segnali elettrici digitali, generati da Arduino con proprie funzioni innescate da dei comandi dati dall'utente.

Per controllare Arduino (e quindi il pilotaggio del sonar) si utilizza un computer che, mediante un applicativo creato ad hoc, comunica via trasmissione seriale con Arduino.

Al fine di poter controllare via software Arduino con il calcolatore, è stato definito il seguente protocollo di comunicazione, che stabilisce le regole per l'invio dei comandi e relativi parametri di controllo ad Arduino e per la ricezione dei dati rilevati.

Per quanto riguarda l'invio dei segnali di pilotaggio da PC ad Arduino è stato adottato il seguente standard:

- come separatore fra i vari parametri viene utilizzato un singolo carattere #;
- ogni stringa di comando termina sempre con il carattere #;
- il primo parametro in una stringa di comando è sempre costituito da 5 caratteri che identificano il nome del comando stesso.

I comandi adottati sono i seguenti:

1. "dirfi#<millisecondi>#<direzione>#";
2. "arcsc#<millisecondi>#<estremoinferiore>#<estremosuperiore>#<intervallo>#<impostavanzate>#";
3. "siimp#<direzione>#";
4. "false#"

Come è possibile vedere dai comandi sopra elencati, a seconda del comando la stringa avrà un numero differente di parametri, ognuno in una posizione ben definita.

Il primo comando, "dirfi" (che sta per *direzione fissa*) viene utilizzato quando l'utente opta per una serie di scansioni tutte nella medesima direzione. In questo caso il secondo parametro contiene il numero di millisecondi stabiliti dall'utente che dovranno trascorrere fra una scansione e quella successiva, mentre il terzo parametro memorizza la direzione (in gradi) scelta dall'utente nella quale fare le scansioni.

Nel secondo comando, "arcsc" (*arco scansione*, ossia la scansione su un arco di tot gradi) rientrano tutte le situazioni in cui l'utente ha scelto di effettuare una scansione su un fronte di n gradi. In tal caso il secondo parametro, come nel comando precedente, contiene i millisecondi di attesa fra una scansione e la successiva, il terzo parametro contiene l'estremo inferiore dell'intervallo di gradi in cui fare la scansione, il quarto contiene l'estremo superiore di tale intervallo, il quinto parametro memorizza ogni quanti gradi fare la scansione (ad esempio se l'utente vuol fare una scansione ogni 5 gradi anziché ogni grado, o ogni due e così via); infine l'ultimo parametro contiene il numero di scansioni da effettuare per direzione. È come una sorta di scansione a direzione fissa ripetuta per le varie direzioni. Se l'utente decide che per ogni direzione dell'arco il sistema deve effettuare 4 scansioni prima di passare alla direzione successiva, tale parametro memorizzerà il valore 4. Di default tale parametro vale 1.

Il terzo comando viene utilizzato qualora l'utente voglia mandare un singolo impulso in una direzione specifica. In questo caso non deve far altro che specificare in quale direzione mandarlo, e tale direzione sarà contenuta nel secondo e unico altro parametro di questo comando.

L'ultimo comando è il comando di arresto scansione: come si nota non presenta alcun parametro.

Per quanto riguarda la ricezione dei dati da Arduino, la stringa di dati inviati da quest'ultimo al calcolatore non contiene alcun comando ma semplicemente una sequenza dei dati rilevati in un ordine posizionale ben definito:

1. millisecondi (i millisecondi di attesa prima di ricevere un eco da un bersaglio rilevato, ossia i millisecondi di durata dell'impulso cronometrati da Arduino sul pin "Echo Pulse Output" della scheda sonar)
2. conversione digitale dei millivolt restituiti dal sensore di temperatura
3. direzione nella quale è stata effettuata la scansione di cui si stanno trasmettendo i risultati

separati tra loro da ##. In poche parole il messaggio che restituisce i dati rilevati al calcolatore ha la seguente struttura: <millisecondi>##<millivolt>##<direzione>. Ogni stringa di dati termina quindi sempre con il dato sul rilevamento e non con il simbolo ##. Questo perché in Arduino il simbolo # viene usato per identificare la fine di ogni comando, mentre in java non è necessario in quanto usiamo le funzionalità offerte dalla classe String Tokenizer. Poiché Arduino invia come secondo parametro i millivolt, convertiti in un formato digitale a 10 bit, sarà il programma Java a convertirli nel corrispondente valore di temperatura espresso in gradi Celsius.

# 7. Il software di pilotaggio di Arduino

Per quanto riguarda la programmazione di Arduino, per prima cosa sono stati associati i pin di Arduino alle relative costanti, che prendono il nome dai pin o dalle funzioni a cui i pin stessi sono deputati:

1. ECHO_PULSE_OUT = 11 rappresenta il segnale di uscita;
2. TRIGGER_PULSE_IN = 10 rappresenta il pilotaggio;
3. RESET = 9;
4. IDLE_STATUS_PIN = 2 rappresenta il led giallo indicante l'attesa;
5. SCAN_STATUS_PIN = 7 rappresenta il led verde indicante la scansione in corso;
6. TX_PIN = 4 è il led blu della trasmissione dati;
7. RX_PIN = 3 è il led blu della ricezione dati;
8. MOTOR_CONTROL = 5 memorizza il pin attraverso il quale avverrà il controllo del motore;
9. LM35DZ_OUTPUT = 0, indica il pin analogico che è stato collegato al pin di output del sensore di temperatura.

Ciò è stato fatto per rendere il codice manipolabile in maniera più efficiente. Se ad esempio si cambia il pin di Arduino a cui è collegato l'output del sensore di temperatura, nel codice sarà sufficiente variare il valore della relativa costante e automaticamente tutte le istruzioni che utilizzato tal valore saranno aggiornate.

Successivamente, tramite la funzione *setup()*, che viene eseguita una sola volta all'avvio di Arduino, viene preparata la periferica, predisponendola in modalità attesa. Viene quindi fatto quanto segue:

- si specifica quali pin di Arduino sono utilizzati come OUTPUT e quali come INPUT, ossia quali pin di Arduino butteranno fuori un segnale in uscita e quali riceveranno dei segnali in ingresso;

- si settano a livello basso i segnali TRIGGER_PULSE_IN e RESET;
- viene avviata la comunicazione tra Arduino ed il calcolatore;
- i pin SCAN_STATUS_PIN, RX_PIN e TX_PIN vengono settati a livello basso per spegnere i relativi led, mentre il pin IDLE_STATUS_PIN viene settato a livello alto per accendere il relativo led indicante la modalità di attesa.

Quindi Arduino si pone in stato di attesa di comandi dal canale seriale. Quando il dispositivo si accorge di avere dei dati disponibili, cioè un nuovo comando sostanzialmente, lo legge, estrae il comando e i parametri dalla stringa e li salva nelle opportune variabili, dopodiché richiama la modalità di funzionamento relativa al comando inviato:

- il comando *dirfi* è associato alla funzione *modeOne()*. Tramite tale metodo il sonar esegue ripetute scansioni in una direzione fissa impostata dall'utente;
- il comando *arcsc*, tramite la funzione *modeTwo()*, esegue una scansione su un arco definito dall'utente;
- il comando *siimp* esegue invece una scansione singola in una direzione specifica attraverso la funzione *modeThree()*.
- il comando *false* invece resetta la periferica, la riporta in stato di attesa (tramite la funzione *modeIdle()*) e invia al PC la stringa particolare "-1##-1##-1", che comunica all'applicativo Java che è stato ricevuto il comando di arresto e che il sonar è quindi in stato di attesa.

Tutte queste quattro funzioni continueranno ad essere eseguite fintanto che Arduino non riceve un nuovo comando via comunicazione seriale, nel qual caso esse vengono interrotte per effettuare la lettura del nuovo comando per poi eseguirlo.

Si fornisce ora una spiegazione più in dettaglio sulla modalità di scansione di un fronte. Nell'applicativo Java, è stato utilizzato un riferimento da -90° a + 90° (0° la posizione centrale), mentre la libreria utilizzata su Arduino utilizza un sistema di riferimento da +180° a 0° (90° la posizione centrale). Si supponga che l'utente voglia effettuare una scansione da -45° a +45° e che voglia che le scansioni non siano effettuate ogni grado bensì ogni 5 gradi, 2 scansioni per ogni direzione prima di passare a quella successiva. In questo caso Arduino riceverà un comando in cui l'estremo inferiore (si veda il capitolo 6 per la definizione delle stringhe di comando) dell'intervallo conterrà il valore 135, l'estremo superire conterrà il valore 45, il parametro "intervallo" conterrà il valore 5 e infine il parametro "impostavanzate"conterrà il valore 2. Il perché di questa differenza si intuisce osservando le differenze fra i due sistemi di riferimento adottati. Quindi è necessario che il programma Java operi una conversione automatica fra i due sistemi di riferimento, in modo da passare poi ad Arduino il sistema di riferimento corretto per il motore, ma fornendo all'utente un più logico sistema di riferimento da -90° a +90°.

Si fa anche un breve cenno ai seguenti metodi:

- *scan()*, ossia il metodo vero e proprio che acquisisce i dati rilevati da sonar (cronometrando la durata dell'impulso del segnale in uscita dal pin Echo Pulse Output) e sensore di temperatura (effettuando direttamente una conversione digitale a 10 bit del valore di tensione rilevato sul pin analogico specificato e ritornando il corrispondente valore intero) e prepara la stringa dati da inviare al PC a mezzo canale seriale.
- *sendPilotImpulse()*, chiamata all'interno di *scan()*, si occupa di inviare il segnale di trigger alla scheda sonar;
- *nextscan(time)*, che è responsabile dell'attesa di *time* millisecondi prima della scansione successiva e che si preoccupa di trasmettere un segnale sonoro attraverso lo speaker nel momento in cui il sonar stesso invia il suo impulso sonoro.

Nel prossimo paragrafo sarà invece illustrato l'applicativo creato per il PC.

# 8. L'applicativo java

Infine si passa ad analizzare l'applicativo Java sviluppato per il PC, ossia l'interfaccia con la quale l'utente è in grado di pilotare la periferica.

L'applicativo è stato realizzato tramite l'utilizzo di NetBeans e delle librerie grafiche di Java. Tale software ha semplificato la realizzazione dell'interfaccia grafica, in quanto consente di costruirla semplicemente trascinando i componenti grafici desiderati all'interno dell'area di lavoro, posizionandoli e poi settando opportunamente le relative proprietà di interesse. In particolare, delle cinque classi di cui si compone il programma, le classi PilotaggioSonarView e PilotaggioSonarAboutBox (della quale non tratteremo in questo capitolo) sono state realizzate tramite la relativa GUI (Graphical User Interface).

PilotaggioSonarApp è la classe principale, ed estende la classe SingleFrameApplication, necessaria per realizzare l'interfaccia grafica.

I compiti svolti da questa classe sostanzialmente sono:

- connessione alla porta USB e instaurazione della comunicazione seriale con Arduino. A tal proposito, durante l'installazione del driver che si deve fare al primo utilizzo (in questa fase Arduino deve essere collegato ad una porta USB), quest'ultimo assegna il dispositivo ad una specifica porta COM (ad esempio la COM4). Il nome di questa porta ("COM4" nel nostro esempio) dovrà essere specificato nel codice di questa classe, altrimenti le librerie RxTx non saranno in grado di instaurare una comunicazione seriale con Arduino e il nostro dimostratore non potrà funzionare;
- prelevamento dei parametri impostati dall'utente, creazione della stringa di comando opportuna, invio di quest'ultima ad Arduino secondo il protocollo specificato in precedenza ed avvio della scansione;
- terminazione della scansione in corso, sempre creando l'opportuna stringa di comando e inviandola ad Arduino;
- ricezione della stringa di dati inviata da Arduino, elaborazione dei dati ricevuti (ad esempio: calcolo della temperatura rilevata, calcolo della velocità effettiva del suono in base alla temperatura stessa e successivo calcolo della distanza a cui si trova l'oggetto rilevato);
- aggiornamento dell'interfaccia grafica, caricando i dati testuali calcolati al punto precedente;
- chiamata alla classe PilotaggioSonarRadarMonitor, che si occupa di realizzare la rappresentazione grafica dei rilevamenti effettuati.

La classe PilotaggioSonarView è la classe che definisce l'interfaccia grafica dell'applicazione, realizzata come detto precedentemente tramite GUI. In questa classe, ai metodi e al codice generato direttamente da NetBeans, sono stati affiancati ulteriori metodi per il controllo degli errori di inserimento. In particolare si notino due situazioni:

1. nella maggior parte dei casi, l'inserimento dei parametri avviene tramite RadioButton, Slider e Spinner. Questi sono stati programmati per escludere automaticamente i componenti che andrebbero a causare conflitto con quelli selezionati nel caso dei RadioButton; per aggiornare automaticamente le proprietà degli altri Slider (utilizzati per specificare la direzione di scansione o gli estremi e l'intervallo della scansione di un arco) quando si modifica il valore di uno di essi, in modo tale da non avere combinazioni di valori senza senso; infine impostando lo Spinner affinché consenta l'inserimento di soli valori numerici interi positivi;
2. nel caso delle due Textfield per l'inserimento delle temporizzazioni (in millisecondi, secondi o frequenza) sono stati creati due metodi di controllo che verificano la correttezza dell'input inserito non appena si perde l'OnFocus su tali Textfield ed eventualmente segnalano l''errore all'utente, riportando la sua attenzione sulla Textfield opportuna.

La classe PilotaggioSonarRadarMonitor contiene una serie di funzioni e strutture dati ausiliarie per la gestione della visualizzazione grafica dei rilevamenti effettuati. In particolare questa classe si occupa di disegnare il monitor (con le curve per rappresentare la scala di distanze), il pennello (che ci fornisce un'indicazione visiva della direzione in cui si sta effettuando la scansione) e gli ultimi (massimo 10) rilevamenti effettuati dal sonar.

Infine la classe PilotaggioSonarRilevamento si occupa unicamente di definire una struttura dati specifica, adatta a memorizzare tutti i dati di ogni singolo rilevamento.

Si veda la documentazione java realizzata mediante javaDoc per ogni ulteriore informazione sui singoli metodi e classi.

# 9. Utilizzo del software

In questo capitolo saranno descritte brevemente le procedure per il collegamento, l'avvio e lo spegnimento della periferica.

## 9.1. Procedure di avvio e di arresto

L'utilizzo del sonar richiede che vengano eseguite una serie di procedure di avvio in ordine ben specifico per attivare la periferica:

1. Collegare l'alimentazione esterna del motore al dispositivo e poi alla rete elettrica;
2. L'interruttore del passo 1 deve essere commutato su ON;
3. Il cavo dati USB deve essere collegato ad Arduino e al calcolatore;
4. A questo punto l'applicazione può essere correttamente avviata;
5. Ora l'applicazione e il dispositivo sono operativi, pronti per ricevere comandi ed effettuare scansioni.

Per quanto riguarda le procedure di arresto, invece:

1. Scollegare il cavo dati USB: in tal modo si toglierà alimentazione ad Arduino, al sonar e al sensore di temperatura.;
2. L'interruttore del circuito di alimentazione del motore deve essere posizionato su OFF;
3. Scollegare l'alimentazione esterna del motore prima dal dispositivo e poi dalla rete elettrica.

## 9.2 Interfaccia utente

L'interfaccia dell'applicazione realizzata si mostra come in figura 8.

Nella parte superiore sinistra si osserva il monitor che visualizza graficamente gli ultimi dieci rilevamenti effettuati e il pennello che indica la direzione attuale nella quale sta puntando il sonar, qualora la periferica sia in modalità funzionamento. Se la periferica è in modalità di attesa, non vengono visualizzati né il pennello né alcun rilevamento precedentemente effettuato. Nella parte a destra i dati dell'ultimo rilevamento effettuato vengono mostrati in formato testuale (oppure i campi sono vuoti quando la periferica è in modalità attesa).

Nella parte inferiore sono presenti tre macrosezioni, nelle quali l'utente può impostare i parametri relativi alla scansione che vuole effettuare. In particolare nella sezione "controlli intervallo scansioni" è possibile selezionare uno tra i tre metodi di inserimento dell'intervallo di attesa:

1. Tempo: quando si vuole specificare l'intervallo di attesa fra due scansioni in secondi o millisecondi;
2. Frequenza, per specificare con quale frequenza effettuare le scansioni;

3. Singolo impulso, per inviare un singolo impulso nella direzione specifica. Se si seleziona quest'opzione, automaticamente viene disattivata la possibilità di selezionare la seconda opzione della prossima sezione.

Nella sezione "Controlli motore" si può scegliere fra:

1. Direzione fissa, se si vuole far eseguire tutte le scansioni in una direzione ben specifica;
2. Specifica fronte di scansione, se si vogliono impostare i parametri opportuni affinché il sonar spazi su un determinato intervallo di direzioni.

Infine l'ultima sezione in basso "Impostazioni avanzate" contiene un parametro selezionabile solo quando è stata scelta l'opzione "Specifica fronte di scansione" della sezione precedente. In tal caso questo parametro permette all'utente di impostare il numero di scansioni che la periferica deve effettuare in ogni direzione prima di passare alla successiva.

Nella figura 8 è possibile vedere uno screen shot dell'applicazione in funzione:

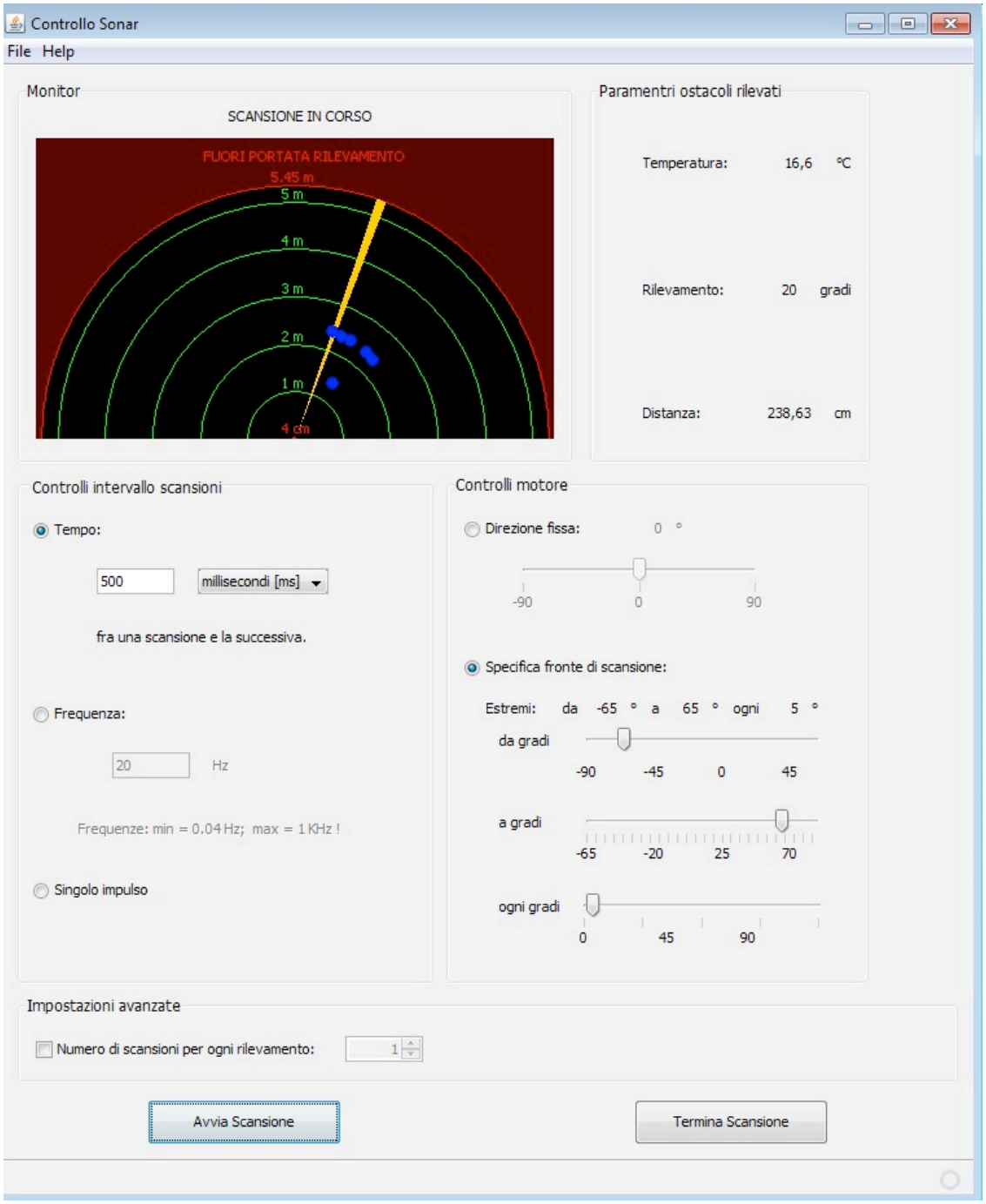



**Fig. 8 - Applicazione in esecuzione**

# 10. Conclusioni e sviluppi futuri

In robotica i sonar sono essenzialmente usati come dei sensori di prossimità, ossia come dei sensori atti a rilevare se un qualche ostacolo si sta avvicinando al robot o alla periferica che li monta (o viceversa). In un contesto come questo non serve la precisione assoluta che fornirebbe ad esempio un sistema di visione mediante telecamere, non è necessario sapere che l'ostacolo si trova esattamente a 5° sulla destra della direzione in cui il robot si sta muovendo. Se anche il sonar sbagliasse il rilevamento di qualche

grado, il discorso non cambia: il robot sta andando ad urtare qualcosa, quindi se la distanza dall'ostacolo rilevato scende al di sotto di una certa soglia di guardia, esso dovrà cambiare direzione per evitare la collisione. In cui contesto come questo il sonar va benissimo, ed infatti è ampiamente utilizzato.

Con questa periferica si è dimostrata la scarsa precisione nella rilevazione di oggetti mediante il sonar. La scheda utilizzata ha una portata massima di poco oltre i 5 metri, inoltre il fascio di impulsi sonar si diffonde nello spazio antistante i trasduttori disperdendosi di circa 15° in ogni direzione rispetto alla direzione di puntamento. Cosa significa ciò? Per brevi distanze di rilevamento, sostanzialmente inferiori al mezzo metro, questo discorso è trascurabile, poiché porterebbe ad assumere dei rilevamenti in direzioni che al massimo sono sbagliate di una manciata di centimetri (ad esempio il sonar rileva un ostacolo a 15° sulla sinistra quando magari in realtà è di 3 – 4 cm più a sinistra). Per lunghe distanze tuttavia questo costituisce un problema, perché l'errore di rilevamento è in questo caso dell'ordine di qualche decina di centimetri: come si può intuire, sarebbe impensabile in un caso come questo ad esempio utilizzare un sonar per guidare un braccio che deve afferrare un oggetto per spostarlo, poiché mancare l'oggetto di qualche decina di centimetri può significare mancarlo completamente, o addirittura afferrarlo male col rischio di danneggiare l'oggetto o addirittura danneggiare anche l'attuatore e il braccio stessi.

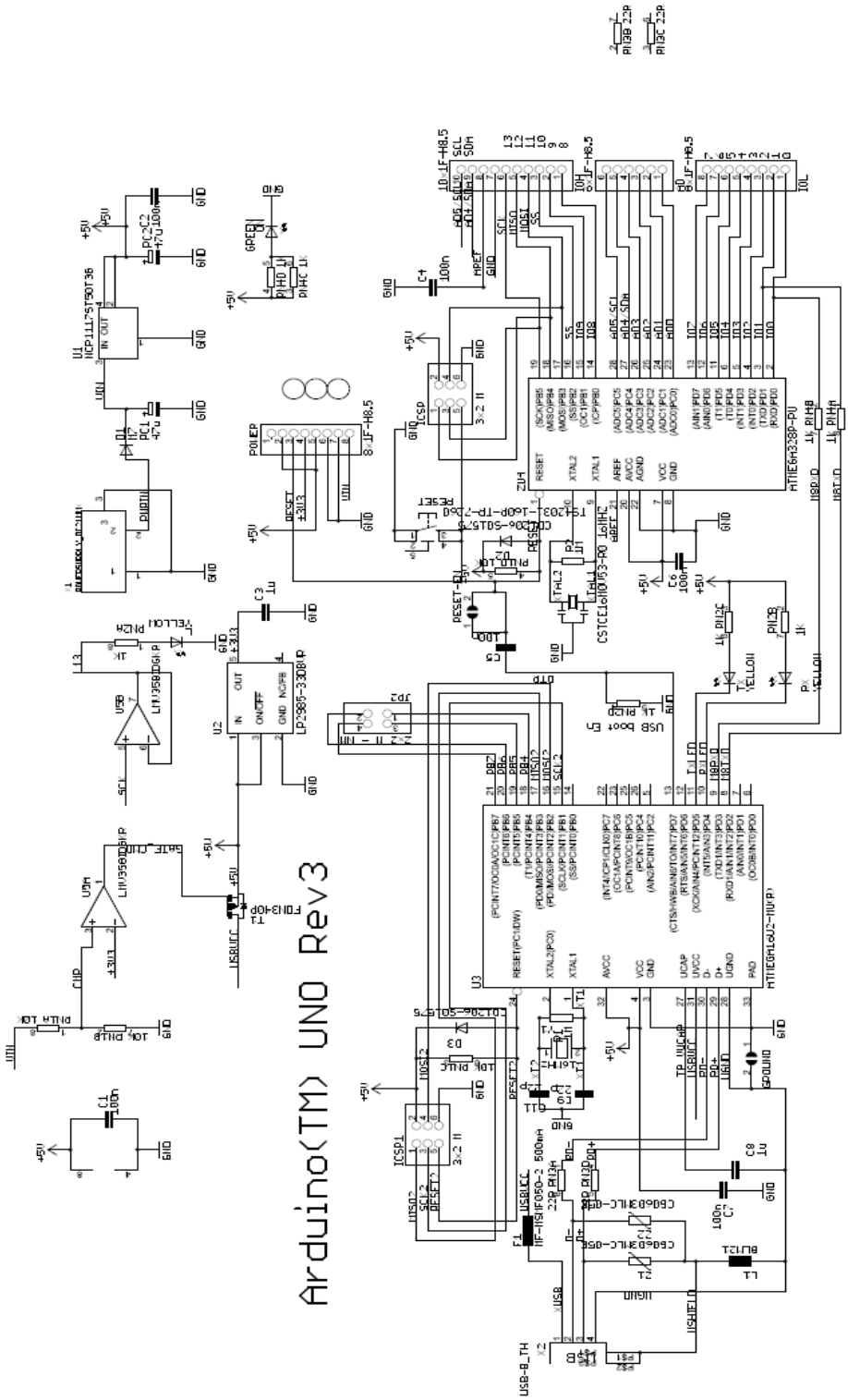
Per contro invece si può dimostrare l'estrema precisione nella rilevazione della distanza alla quale si trova l'oggetto rilevato, precisione dovuta al fatto che si conoscono esattamente le leggi fisiche che governano la velocità del suono nell'aria e possiamo misurare con precisione il tempo di viaggio dell'impulso sonar.

In conclusione quindi si può osservare con questa applicazione come il sonar sia completamente inutilizzabile per rilevamenti di precisione, mentre sia un sistema molto più economico e semplice da gestire per la semplice rilevazione di possibili collisioni con oggetti posti ad esempio sul cammino di un robot, dove non è richiesta la precisione al grado nel rilevamento della direzione in cui si trova l'oggetto, ma è sufficiente una precisione del tipo "guarda che se prosegui in questa direzione vai a sbattere".

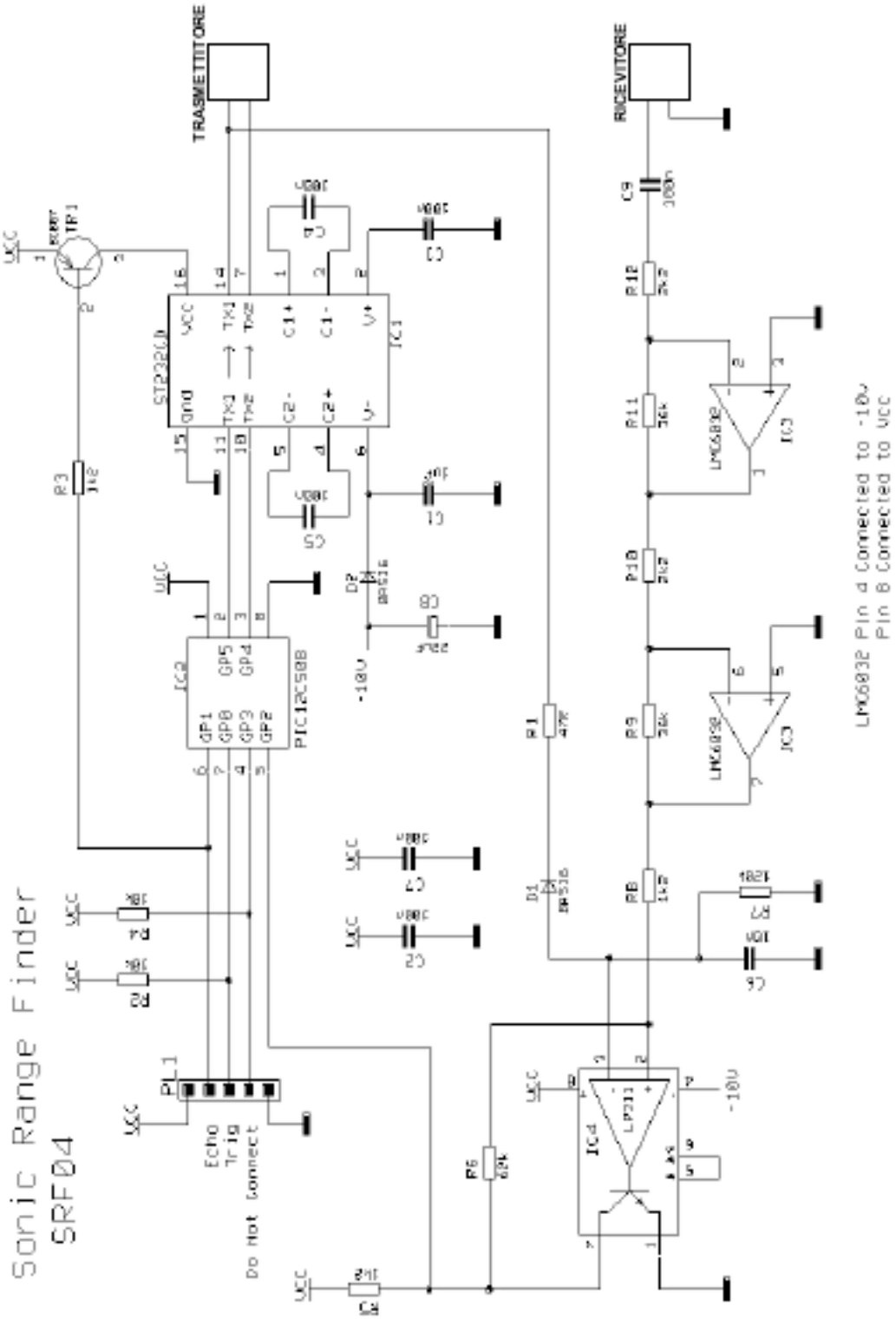
Per quanto riguarda gli sviluppi futuri, uno di questi potrebbe essere tentare di sviluppare un sistema di triangolazione mediante l'uso di diversi sonar che vadano a scandagliare l'area di interesse da direzioni diverse, unendo poi tutte le rilevazioni effettuate e cercando di ricavare la posizione effettiva dell'ostacolo rilevato.

# 11. Appendice

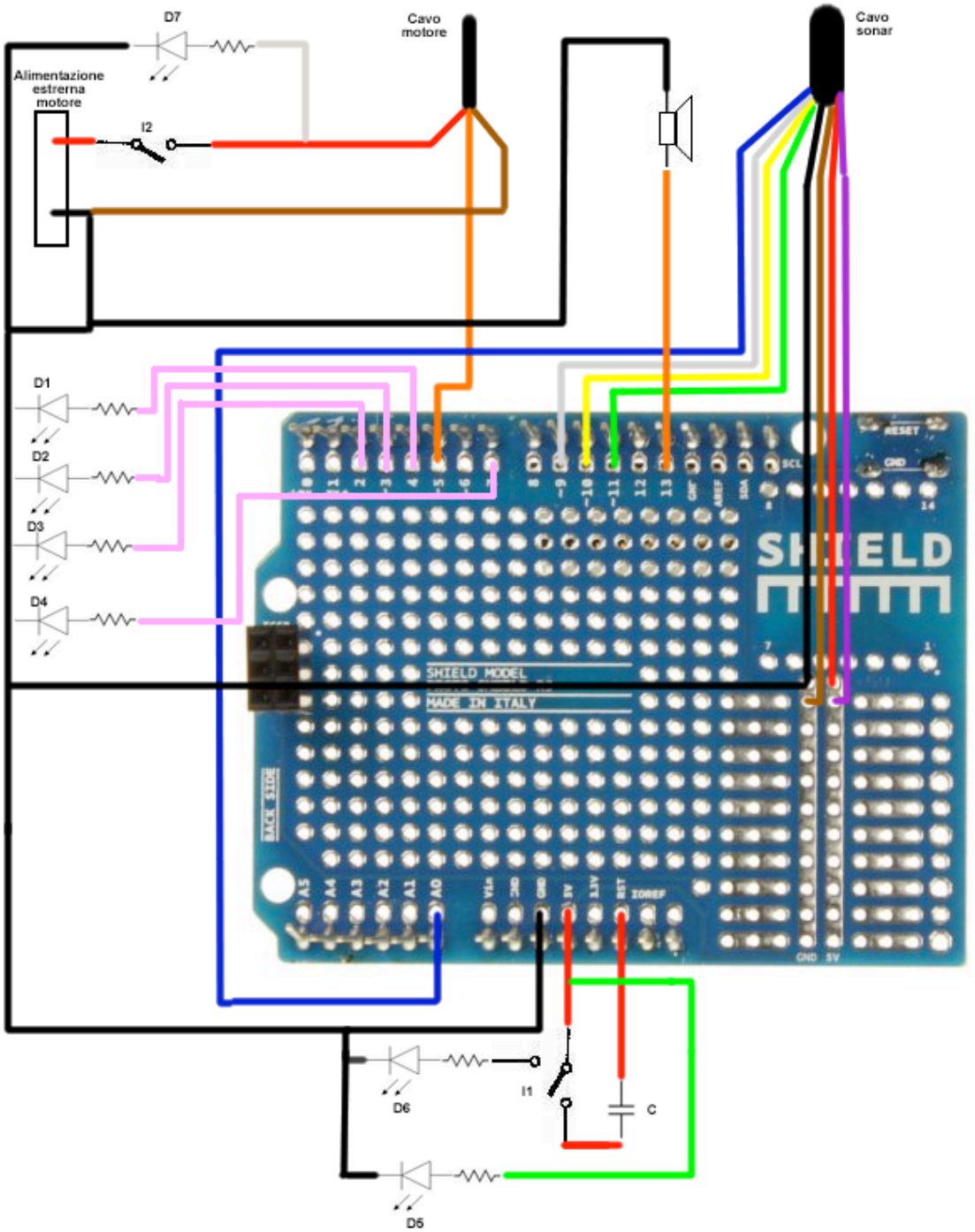
## 11.1. Schema elettrico arduino



## 11.2. Schema elettrico Sonar

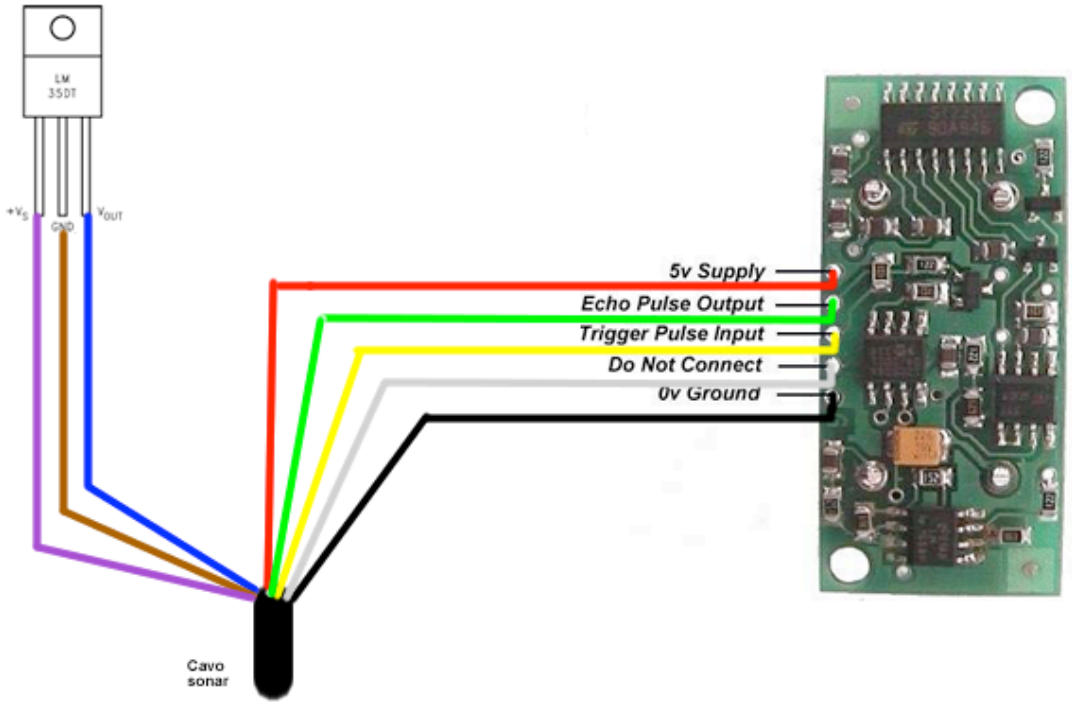


## 11.3. Schema di cablaggio circuito principale

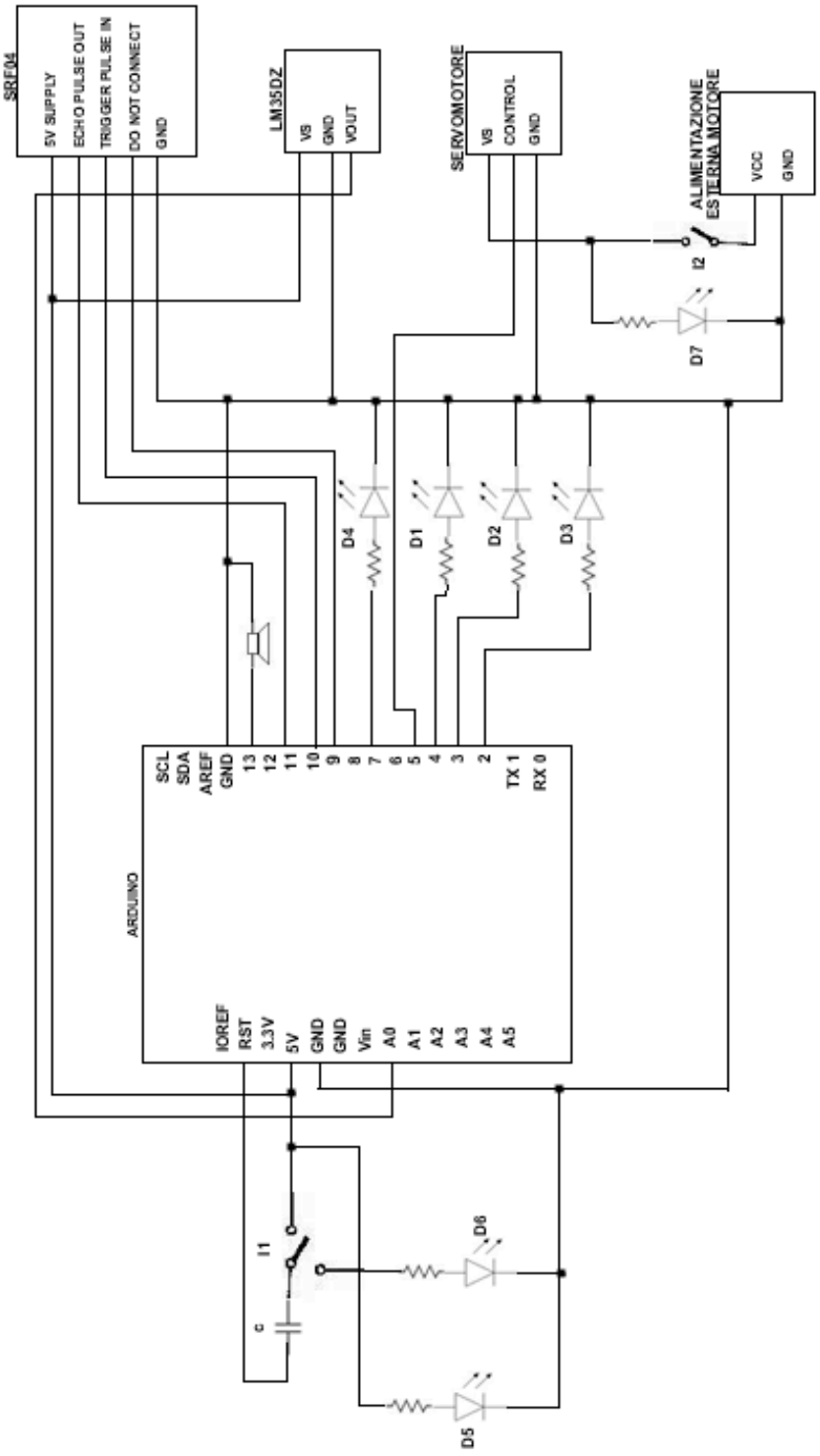


## 11.4. Schema dei collegamenti della piedinatura del circuito sonar e del sensore lm35dz

Questo schema illustra la corrispondenza fra i colori dei cavi e la piedinatura del sonar e del sensore di temperatura.



## 11.5. Schema elettrico del circuito



## 11.6. Il progetto finito

# Bibliografia

[1] Datasheet LM35DZ

[2] Specifiche Arduino

[3] Datasheet sonar

[4] http://ww.arduino.cc

[5] http://docs.oracle.com/javase/6/docs/api/

[6] http://it.wikipedia.org

[7] http://www.rxtx.org

# Indice

**SOMMARIO** ..... **1**

**1. INTRODUZIONE** ..... **1**

**2. I COMPONENTI UTILIZZATI** ..... **2**
**2.1. Il sonar** **2**
**2.2. Arduino Uno** **5**
2.2.1. Specifiche tecniche ..... 5
**2.3. Il servomotore** **6**
**2.4. Il sensore di temperatura LM35DZ** **6**
2.4.1. Specifiche tecniche ..... 7
**2.5. Speaker** **7**

**3. PROBLEMATICHE AFFRONTATE** ..... **7**

**4. LA SOLUZIONE ADOTTATA** ..... **9**

**5. PROGETTAZIONE DEL CIRCUITO** ..... **10**

**6. PROTOCOLLO DI COMUNICAZIONE** ..... **11**

**7. IL SOFTWARE DI PILOTAGGIO DI ARDUINO** ..... **12**

**8. L'APPLICATIVO JAVA** ..... **14**

**9. UTILIZZO DEL SOFTWARE** ..... **15**
**9.1. Procedure di avvio e di arresto** **15**
**9.2 Interfaccia utente** **15**

**10. CONCLUSIONI E SVILUPPI FUTURI** ..... **17**

**11. APPENDICE** ..... **19**
**11.1. Schema elettrico arduino** **19**
**11.2. Schema elettrico Sonar** **20**
**11.3. Schema di cablaggio circuito principale** **21**
**11.4. Schema dei collegamenti della piedinatura del circuito sonar e del sensore lm35dz** **22**
**11.5. Schema elettrico del circuito** **23**
**11.6. Il progetto finito** **24**

**BIBLIOGRAFIA** ..... **25**

**INDICE** ..... **26**