Costruire un robot mobile con Raspberry Pi e Arduino

Emanuele Paiano

27 aprile 2017

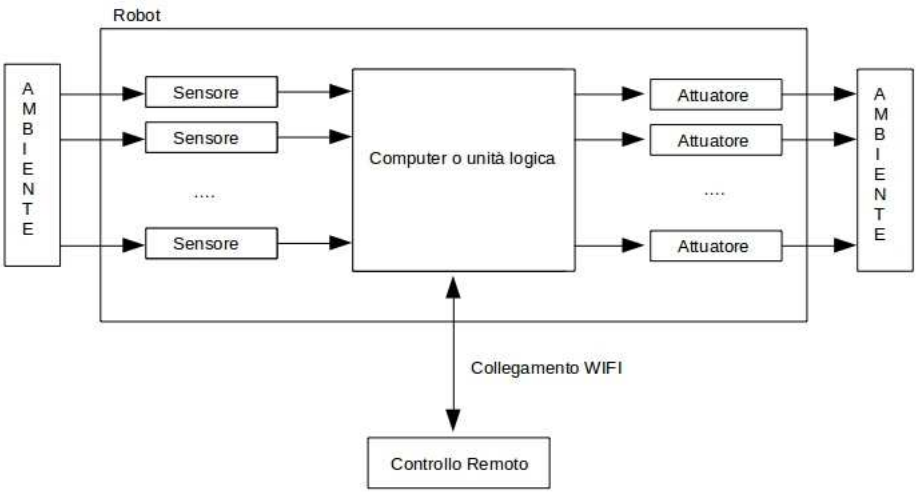
1 Licenza

Per eventuali chiarimenti potete contattare l'autore all'indirizzo nixw0rm@gmail.com.

2 Introduzione

Con l'evoluzione delle varie board di sviluppo e dei sistemi embedded, oggi è estremamente facile assemblare piccoli robot programmabili. E' possibile tirar su delle piattaforme hardware che ci consentono di sperimentare diversi tipi di algoritmi per la robotica, come la navigazione o il semplice controllo remoto. Se consideriamo i costi contenuti dei vari componenti in commercio, ci rendiamo conto che il progetto diventa ancora più realizzabile e alla portata di molte tasche.

Intuitivamente, possiamo paragonare un semplice robot mobile ad un piccolo computer dotato di sensori e attuatori: i primi sono dispositivi di input in grado di percepire lo stato dell'ambiente, mentre i secondi possiamo considerarli dei dispositivi di output che modificano tale stato (compresa la posizione del robot), in base a delle istruzioni contenute in memoria.



Prendendo spunto da una mia esperienza personale, in questa serie di articoli vedremo come realizzarne uno, equipaggiato con:

- sensori ultrasuoni per riconoscere gli ostacoli,
- webcam USB dotata di servomotori,
- altoparlante per permettergli di parlare pronunciando delle frasi preimpostate.

Il telaio può essere recuperato da una vecchia macchina radiocomandata oppure è possibile acquistarne uno già pronto. Il nostro robot avrà in tutto 4 ruote e due motorini: due posteriori per la trazione e due anteriori per lo sterzo, il quale viene gestito da un singolo motorino. La board di riferimento è Raspberry Pi accompagnata da Arduino Nano per una gestione più efficiente dei motori.

Il controllo remoto, consiste in un server TCP che gestisce i singoli comandi ricevuti da uno smartphone o pc (su cui gira il client), mentre per il flusso della webcam si farà uso di un server HTTP per lo streaming video.

All'eventuale domanda sul perchè non ci si limita ad Arduino, la risposta metterebbe in evidenza tutti i vantaggi di avere installato Linux in un robot: Raspbian lo rende scalabile e flessibile, permettendo di aggiungere o implementare diverse funzionalità aggiuntive, tra cui:

- far parlare il robot installando espeak, noto sintetizzatore vocale per linux;
- configurare un server vpn e fare in modo che il robot venga gestito fuori dalla nostra rete locale;
- implementare un piccolo sistema di sorveglianza remota direttamente nel robot, con notifica email per eventuali movimenti sospetti (rendendolo un «robot da guardia»).

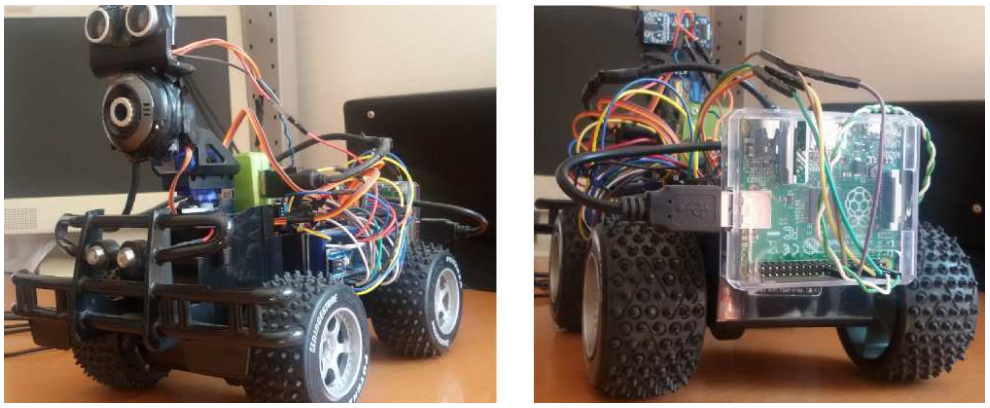


Figura 1: Un robot mobile realizzato con Raspberry

- possiamo scegliere, tra i linguaggi supportati da raspberry, quello che ci fa più comodo per scrivere il server;
- fare quasi tutto quello che si può fare con un raspberry (immaginate di dare un comando vocale al robot per riprodurre un mp3 in memoria).

Implementare tali caratteristiche con un microcontrollore, appare molto più complesso. Il prezzo da pagare si riversa sui consumi energetici che saranno decisamente maggiori.

2.1 Una prima versione: un Raspberry con le ruote

Il primo passo è quello di assemblare un «Raspberry dotato di ruote e webcam». Per ora ignoriamo servomotori e sensori, che saranno aggiunti successivamente.

La lista del materiale occorrente prevede:

- Un Raspberry A+, ma andrebbe bene qualunque versione, tenendo conto però, che le versioni B e B+ hanno consumi energetici più elevati (600mAh contro 1500-3000mAh);
- Un dongle Wifi per connettere il raspi ad una rete wireless o per impostarlo come access point. Tale componente non è necessario se usiamo Raspberry Pi 3 (ha la wifi integrata);
- Un hub usb con almeno 3 porte: una per connettere la cam, un'altra per l'adattatore wifi e una di riserva. Tale componente è indispensabile se usiamo il Raspberry A+, in quanto quest'ultimo dispone di una sola porta usb;
- Un Arduino Nano, comandato da Raspberry, si occuperà di gestire il modulo motori ed eventuali attuatori aggiuntivi (luci e servomotori per la webcam).

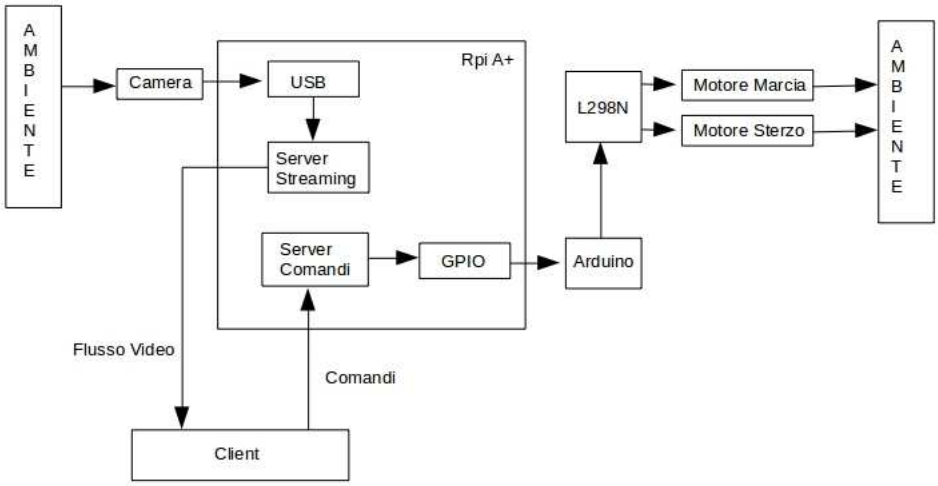


Figura 2: Schema di funzionamento della prima versione del robot

- Un modulo L298N che permette di gestire i motori di marcia e sterzo, oltre che ad alimentare Arduino;
- Un convertitore di livello per far comunicare Raspberry e Arduino. Raspberry lavora a 3.3 V, mentre Arduino a 5. Se non si converte tale voltaggio, si rischia di danneggiare seriamente il Raspi. Come alternativa, i più esperti potranno far uso di un partitore di tensione;
- Batteria ricaricabile da 9-12 Volt e 2500mAh. Per i test iniziali, è possibile utilizzare un alimentatore universale per notebook impostato a 12 Volt;
- Un DC Step-Down per ridurre i 12 Volt in ingresso in 5 Volt per Raspberry. Come alternativa, i più esperti potranno realizzare un regolatore di tensione apposito, utilizzando un LM7805. Ovviamente tutte le masse (GND) andranno collegate tra loro altrimenti non funzionerà nulla.

ALTERNATIVA

- Una batteria lipo da 7,4V per Arduino / Motori e un Powerbank da 5V per Raspberry, ricordando di collegare la massa (GND) del Raspberry a quella di Arduino.

Per avere un'idea del funzionamento generale del del robot, basta fare riferimento allo schema in Figura 2

Tale schema è intuitivo: la webcam è collegata ad una porta USB di Raspberry, Arduino al GPIO del raspi e i motori al modulo L298N che a sua volta sarà collegato ad Arduino. Il motivo per cui non si collega il modulo direttamente al GPIO, è dovuto al fatto che Raspberry non è un sistema realtime (a meno

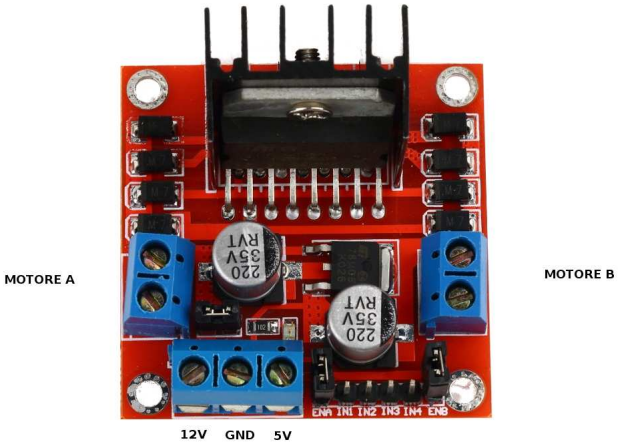


Figura 3: Il modulo L298N

che non si faccia uso di schede aggiuntive): ciò lo rende instabile di fronte a task che devono essere eseguiti e terminati in intervalli di tempo limitati. Infatti, può succedere che il sistema operativo dia maggiore priorità a processi che non riguardano la gestione dei motori, causando una risposta ritardata ai comandi ricevuti (per esempio, il robot potrebbe sterzare in ritardo). Arduino fa da tramite riducendo le operazioni da eseguire, quindi raspberry dovrà limitarsi ad inviare i comandi sulla porta seriale, al resto ci pensa il microcontrollore. Delegando la gestione degli attuatori, si risolve il problema dell'instabilità, dovuta all'impiego di un sistema che non sia realtime. Altre alternative (già pronte) sono le schede HAT, fatte appositamente per permettere a Raspberry di supportare Servo e Motori CC senza problemi, purtroppo tali schede non sono a disposizione dell'autore. Andiamo a vedere nel dettaglio come saranno effettuati i collegamenti di Arduino al modulo L298N.

2.2 Collegamento motori

La prima componente da assemblare è la parte relativa ad Arduino e i Motori CC (la parte più a destra dello schema in Figura 2), seguito dalla realizzazione di un semplice sketch. Prima di procedere, diamo un'occhiata alle caratteristiche del modulo motori.

2.2.1 Il modulo L298N

Il modulo L298N (figura 3), detto anche H-Bridge, è un modulo utilizzabile per la gestione dei motori a corrente continua. Esso è composto da un ingresso a 12 Volt, due uscite a 3 V per i motori, e un'uscita a 5 Volt per alimentare la scheda di controllo. Purtroppo, quest'ultima, non può essere utilizzata per alimentare

AZIONE MOTORE A	IN1	IN2
FERMO	LOW	LOW
RUOTA IN SENSO ORARIO	HIGH	LOW
RUOTA IN SENSO ANTIORARIO	LOW	HIGH

Tabella 1: Valori degli ingressi per pilotare il motore A

PIN ARDUINO	PIN H-BRIDGE
GND	GND
D9	ENA
D3	ENB
D7	IN1
D6	IN2
D5	IN3
D4	IN4

Tabella 2: Collegamento pin tra H-Bridge e Arduino

Raspberry (l'amperaggio fornito è insufficiente) ma va benissimo per Arduino. Tuttavia, non sarà utilizzata per le fasi di test, infatti ad Arduino collegheremo solo la massa (GND) ma non l'alimentazione, in quanto il microcontrollore sarà alimentato dalla stessa porta USB del PC. Gli ingressi per la gestione dei motorini sono:

- Enable Motor A (ENA): E' l'ingresso utilizzato per impostare il PWM del motore A, quindi anche la sua velocità. Se messo in corto con il pin superiore (5V+), la velocità rimarrà constante (al valore massimo). Diversamente, possiamo regolare gli impulsi da Arduino e farla variare (sfruttando il Duty Cycle);
- Enable Motor B (ENB): vale lo stesso discorso di ENA, ma per il motore B.
- Gli ingressi IN1 e IN2 funzionano come mostrato in Tabella 1. Lo stesso discorso vale per IN3 e IN4, solo che viene gestito il motore B.

2.2.2 Montaggio e sketch

I collegamenti da effettuare tra i pin di Arduino e quelli del H-Bridge sono descritti in Tabella 2

In Figura 4 possiamo vedere lo schema completo. Il motore cc utilizzato è il Mabuchi FA130 a 3V (spesso presente nei giocattoli elettrici, come trenini e macchine da pista polistil), ma il modulo ne supporta anche altri tipi. Montato il circuito, possiamo pensare all'implementazione dello sketch di Arduino.

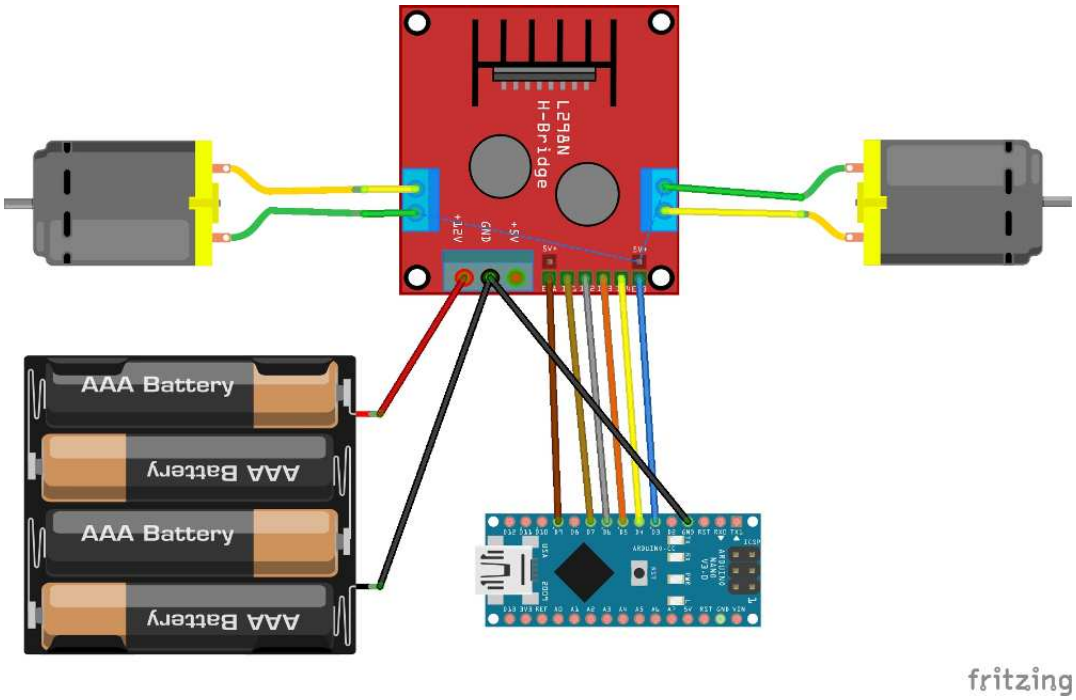


Figura 4: Collegamento di Arduino al modulo L298N.

Le funzioni che utilizzeremo sono *analogWrite()* per impostare il valore di ENA e quello di ENB, mentre *digitalWrite()* lo utilizzeremo per inviare i valori *HIGH* e *LOW* agli altri pin.

Il codice seguente è lo sketch da caricare in Arduino.

```
1  /*
2   * Car Arduino simple sketch
3   * 2015 - Emanuele Paiano
4   */
5   /* Primo motore */
6   const int enA = 9;
7   const int in1 = 7;
8   const int in2 = 6;
9
10  /* Secondo motore */
11  const int enB = 5;
12  const int in3 = 7;
13  const int in4 = 4;
14
15  /* Massimo valore PWM motorino di Marcia*/
16  const int kMaxPwm=200;
17
18 /* valore PWM Sterzo */
19 const int kWheelPwm=150;
20
21 /* SpeedVal Max. Massimo valore delle "marce" */
22 const int kMaxSpeed=5;
23
24 /* SpeedVal attuale (in gergo marcia) */
25 int currentSpeed=4;
26
27 void setup() {
28   /* Uscite Modulo L298N */
29   pinMode(enA, OUTPUT);
30   pinMode(enB, OUTPUT);
31   pinMode(in1, OUTPUT);
32   pinMode(in2, OUTPUT);
33   pinMode(in3, OUTPUT);
34   pinMode(in4, OUTPUT);
35
36  /* Inizializzo la comunicazione seriale usb a 9600 bit/
       sec */
37   Serial.begin(9600);
38
39  /* Svuoto il buffer della seriale */
40   Serial.flush();
```

```
41 }
42
43
44 void loop() {
45   if (Serial.available())
46    {
47     /* Leggi byte porta seriale */
48     char c = Serial.read();
49     switch(c)
50     {
51               /* se ho letto una 'A' vado avanti */
52       case 'A':
53                 forward();
54                 Serial.println("FORWARD OK");
55                 break;
56           /* se ho letto una 'a' vado in retromarcia */
57      case 'a':
58                 reverse();
59                 Serial.println("REVERSE OK");
60                 break;
61          /* se ho letto una 's' mi fermo */
62      case 's':
63                 stopMotorA();
64                 Serial.println("STOP OK");
65                 break;
66
67         /* se ho letto una 'L' giro a Destra */
68      case 'L':
69                 turnRight();
70                 Serial.println("RIGHT OK");
71                 break;
72
73           /* se ho letto una 'l' giro a sinistra */
74      case 'l':
75                 turnLeft();
76                 Serial.println("LEFT OK");
77                 break;
78
79          /* se ho letto una 'S' raddrizzo lo sterzo */
80      case 'S':
81                 straight();
82                 Serial.println("STRAIGHT OK");
83                 break;
84
85      /* se ho letto un '+' aumento velocità */
86      case '+':
```

```
87                  currentSpeed++;
88                  currentSpeed=constrain(currentSpeed, 1,
                        kMaxSpeed);
89                  Serial.println(currentSpeed);
90                  break;
91        /* se ho letto un '-' riduco velocità */
92        case '-':
93                  currentSpeed--;
94                  currentSpeed=constrain(currentSpeed, 1,
                        kMaxSpeed);
95                  Serial.println(currentSpeed);
96                  break;
97                }
98          }
99 }
100
101 /* Avanti */
102 void forward() {
103    stopMotorA();
104    digitalWrite(in1, HIGH);
105    digitalWrite(in2, LOW);
106    analogWrite(enA, getSpeedVal(currentSpeed));
107   }
108
109 /* recupera corretto valore di velocità */
110 int getSpeedVal(int value)
111 {
112   int val=0, mult=kMaxPwm/kMaxSpeed;
113   value=constrain(value, 0, kMaxSpeed);
114   val=value*mult;
115   val=constrain(val, 0, kMaxPwm);
116   return val;
117  }
118
119 /* inverti marcia */
120 void reverse() {
121     stopMotorA();
122     digitalWrite(in2, HIGH);
123     digitalWrite(in1, LOW);
124     analogWrite(enA, getSpeedVal(currentSpeed));
125 }
126
127 /* Ferma motore di marcia */
128 void stopMotorA() {
129    digitalWrite(in1, LOW);
130    digitalWrite(in2, LOW);
```

```
131     analogWrite(enA, 0);
132 }
133
134 /* Sterzo a destra */
135 void turnRight() {
136       straight();
137       delay(100);
138       analogWrite(enB, kWheelPwm);
139       digitalWrite(in3, HIGH);
140       digitalWrite(in4, LOW);
141 }
142
143 /* sterzo a sinistra */
144 void turnLeft() {
145   straight();
146   delay(100);
147   analogWrite(enB, kWheelPwm);
148   digitalWrite(in3, LOW);
149   digitalWrite(in4, HIGH);
150 }
151
152 /* raddrizza sterzo */
153 void straight() {
154     digitalWrite(in3, LOW);
155     digitalWrite(in4, LOW);
156     analogWrite(enB, 0);
157 }
```

Il funzionamento del programma è relativamente semplice: i comandi sono semplici caratteri ricevuti sulla porta seriale e, in base al byte ricevuto, viene richiamata una funzione corrispondente ad un'azione da eseguire (*forward, reverse, turnLeft, turnRight, stop e straight*). I comandi possono essere riassunti in Tabella 3.

I possibili livelli di velocità variano da 1 a 5. Si è preferito usare questi valori fittizzi (anzicchè quelli compresi tra 0 a 255), per semplificare l'aumento e la riduzione della velocità utilizzando i caratteri '+' e '-'. Il livello iniziale è 4, in quanto se c'è scarsa trazione, il robot non si muove. Inoltre, per allungarne la vita, si è preferito impostare a 200 il limite massimo del pwm di marcia e a 150 quello dello sterzo: valori più alti sono inutili, data la struttura della nostra macchina (dare troppa potenza al motorino dello sterzo potrebbe essere dannoso).

Per verificare se abbiamo fatto tutto correttamente, possiamo collegare Arduino alla porta usb del PC, alimentiamo il modulo (tramite alimentatore o

COMANDO	AZIONE
A	MARCIA AVANTI
a	MARCIA INDIETRO
L	STERZO DESTRA
l	STERZO SINISTRA
s	MARCIA STOP
S	RADDRIZZA STERZO
+	AUMENTA VELOCITÀ
-	RIDUCI VELOCITÀ

Tabella 3: Comandi da inviare ad Arduino

PIN ARDUINO	CONVERTER HV	CONVERTER LV	PIN RASPI
TX	RXI	RXO	UART_RXD
RX	TXO	TXI	UART_TXD
GND	GND	GND	GND
5V	HV	LV	3,3V

Tabella 4: Collegamento UART arduino-convertitore-raspberry.

batterie) e proviamo ad inviare i caratteri con il monitor seriale dell'ide di Arduino, impostando la velocità a 9600bps. Se i motorini girano, e il modulo risponde correttamente ai comandi, allora possiamo passare alla tappa successiva: il collegamento del Raspberry.

2.3 Collegare il Raspberry

Per collegare il Raspberry, abbiamo bisogno di un convertitore dei livelli di tensione (Figura 5, in alto a destra), in particolare sul pin di ricezione del raspi (UART_RXD). L'elenco dei Pin è visibile in Figura 5.

Il collegamento da realizzare è di tipo UART: consiste nel collegare il Pin TX di Arduino al Pin UART_RXD di Raspberry, e il pin UART_TXD al pin RX di Arduino come mostrato in figura 6, facendo riferimento ai pin in Tabella 4. Il metodo è molto simile a quello della costruzione di un cavo null modem, ma le controparti hanno livelli di tensione differenti (Arduino lavora a 5V, mentre Raspberry a 3,3).

Collegato il raspberry, non resta che collegare anche il pin VIN di Arduino al pin 5V del modulo L298N (staccate arduino dalla usb del pc), come mostrato in figura 7. Per alimentare raspberry, possiamo usare un alimentatore da smartphone (purchè sia a 5V), almeno per le fasi di test. Successivamente, si farà uso di un modulo dc step down: si prenderà parte della tensione globale del sistema, e la si ridurrà a 5 V per Raspberry. In questo modo, avremo due diramazioni a partire dalla sorgente a 12V: una a 5V per Raspberry e l'altra a 12 per alimentare il modulo motori (insieme al microcontrollore). L'importante è che fate arrivare almeno 9V (meglio 12) e 2400mAh in ingresso.

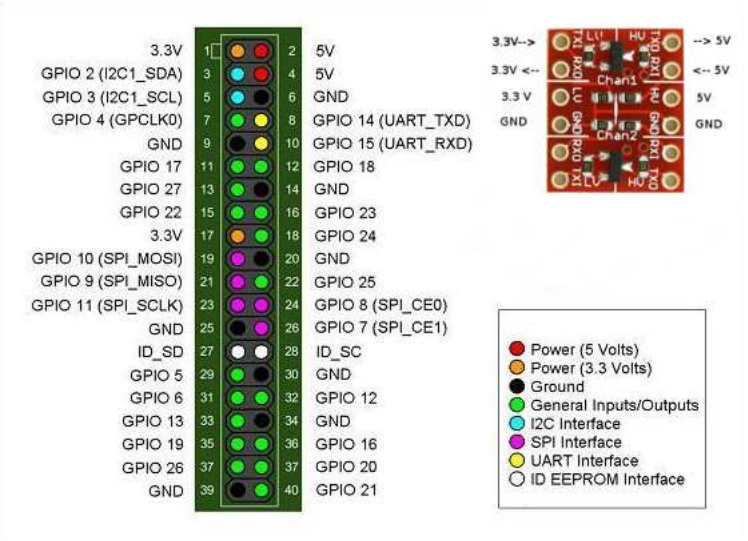


Figura 5: Mappatura pin di Raspberry (a sinistra) e un convertitore di livello (in alto a destra)

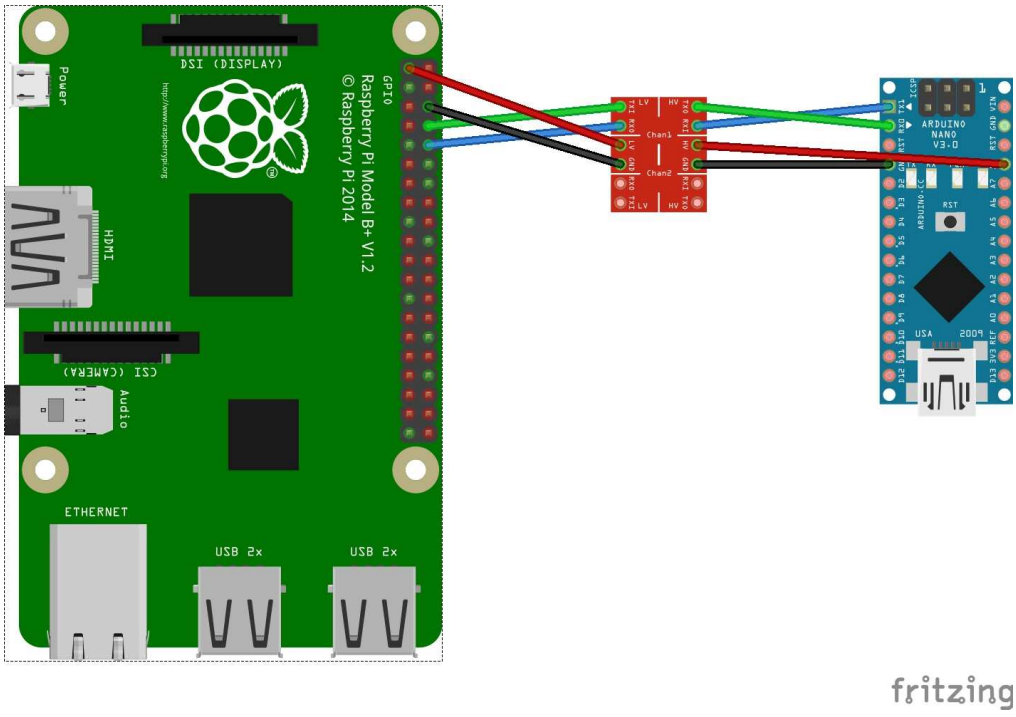


Figura 6: Collegamento seriale Raspberry - Arduino con convertitore di livello

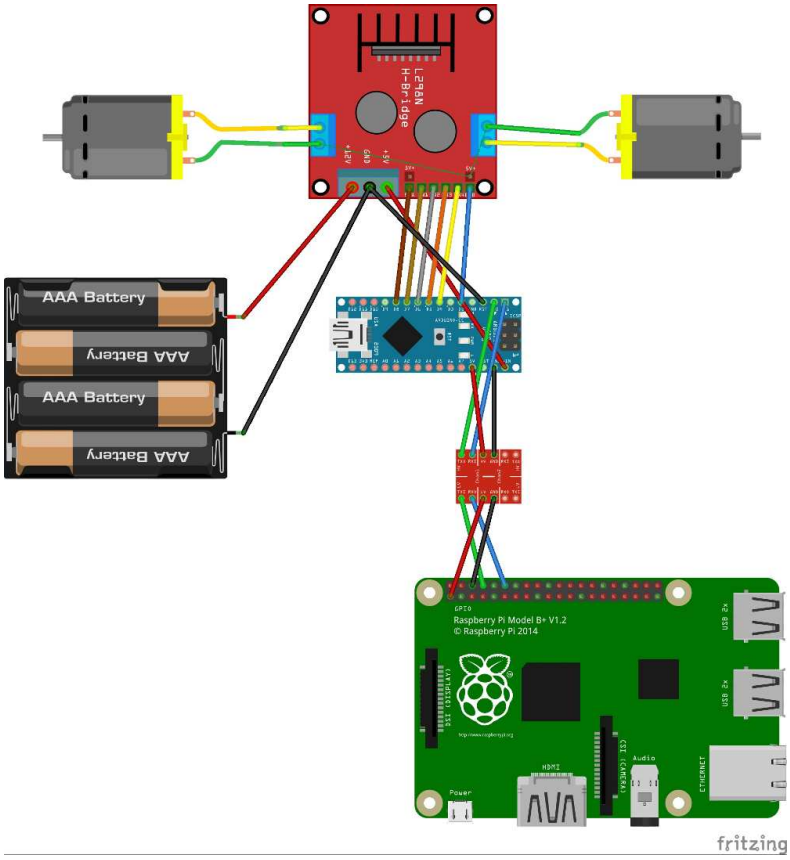


Figura 7: Il circuito nel complesso: Arduino alimentato direttamente dall'H-bridge, mentre Raspberry sarà alimentato a parte. Da non dimenticare il collegamento GND di raspberry a quello di tutto il circuito.

2.4 Prima configurazione e test del collegamento

Dopo aver configurato Raspbian, ci limiteremo a testare i motorini da Raspbian. Forse, non tutti sanno che sulla linea seriale *UART*, di default c'è un attivo un processo getty che permette il login: dobbiamo disattivarlo o il nostro circuito avrà dei malfunzionamenti. Per fare ciò apriamo il file /etc/inittab e troviamo la stringa

```
T0:23:respawn:/sbin/getty -L ttyAMA0 115200 vt100
```

possiamo disattivarla aggiungendo il carattere # per commentarla

```
#T0:23:respawn:/sbin/getty -L ttyAMA0 115200 vt100
```

e salviamo il file. Inoltre, quando raspbian si avvia, tutte le informazioni vengono inviate sulla porta seriale, in quanto tale parametro viene passato al boot. Per disattivarlo, apriamo il file /boot/cmdline.txt e troviamo la riga simile a questa

```
dwc_otg.lpm_enable=0 console=ttyAMA0,115200 kgdboc=
    ttyAMA0,115200 console=tty1 root=/dev/mmcblk0p2
    rootfstype=ext4 elevator=deadline rootwait
```

e rimuoviamo tutti i riferimenti a ttyAMA0, nel nostro caso la riga diventa

```
dwc_otg.lpm_enable=0 console=tty1 root=/dev/mmcblk0p2
    rootfstype=ext4 elevator=deadline rootwait
```

salviamo il file e riavviamo raspberry. Ovviamente tali impostazioni possono variare da una versione all'altra di raspbian, ma spesso sono simili. Ora si lancia minicom sulla porta */dev/ttyAMA0* con una velocità a 9600bps:

```
minicom -b 9600 -o -D /dev/ttyAMA0
```

digitando i comandi, possiamo inviare i caratteri e vedere i motorini funzionare.

2.5 Riepilogo e conclusioni

Siamo partiti dal collegare i motori al modulo L298N. Abbiamo realizzato uno sketch per Arduino che riceve caratteri ascii e genera segnali HIGH e LOW da mandare al modulo motori, per poi interfacciare tutto con raspberry in modo da ricevere i comandi da minicom.

A questo punto, possiamo pensare alla parte applicativa: l'idea è quella di scrivere un insieme di librerie e classi in un linguaggio che, agendo sul file /dev/ttyAMA0, gestisca i servo della cam e i motori di marcia. Si potrebbe pensare di scrivere una classe per ogni sensore e attuatore, in modo tale da arrivare a programmare il robot in stile oop, arrivando anche ad implementare un server TCP in qualunque linguaggio. L'obiettivo finale di questo progetto, è quello di arrivare a ricevere i caratteri ascii direttamente da un client remoto.

I comandi saranno inviati da un'apposita app Android, configurata per pilotare il robot via wlan. Su Google Play alcune sono gratuite e con supporto streaming da cam: una app che fa a caso nostro, può essere WIFI TCP/UDP Controller.