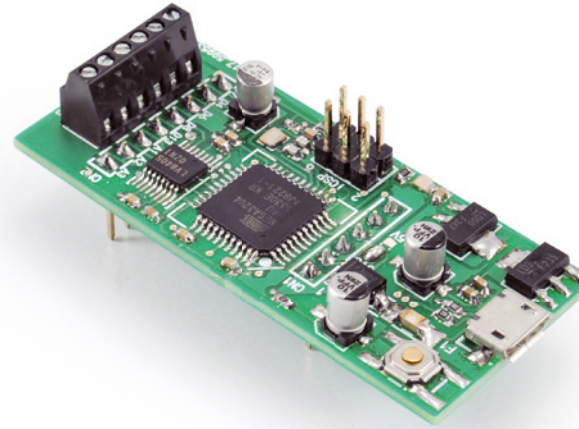
Open Motor Control

(cod. FT1173M)



Driver ideale per gestire piccoli motori con un'elevata flessibilità operativa, al fine di permettere a chiunque di poterlo adattare alle proprie esigenze.
OpenMotorControl, abbreviato in OMC21, dove 21 indica il numero di canali (due in questo caso) e la corrente gestita per ciascun canale (1A).
Un semplice ed economico circuito integrato contenente un ponte ad H permette di contenere costi ed ingombri. Il circuito integrato adottato è siglato LV8405 ed è un completo ponte ad H a MOSFET ad elevate prestazioni che implementa anche un controllo sulla tensione di alimentazione e dispone di protezioni sia in temperatura che in corrente. Per realizzare un controllo motori facile da interfacciare e programmare è stato abbinato a questo IC un microcontrollore debitamente programmato (ATmega32U4) con caratteristiche interessanti:

- configurazione a doppio ponte H per pilotare due motori in cc o uno stepper motor bipolare;
- tensione della sezione di potenza: 3 ÷ 15V;
- tensione di alimentazione della logica: 3 ÷ 5V;
- corrente di uscita: 1,4A continua (2,5A picco) per motore;
- interfacciamento USB, seriale, I²C-Bus;
- ingressi compatibili 3,3V e 5V;
- alimentazione selezionabile tra interna ed esterna a 3,3V o 5V.

Il controller è molto flessibile, specialmente nella sezione di alimentazione; il driver è compatibile con logiche a 3,3V e 5V, può essere alimentato dalla stessa tensione della logica di comando, e consente di alimentare una logica esterna, così da lavorare con gli stessi livelli di alimentazione.
Per selezionare la necessaria tensione di alimentazione è necessario unire con una goccia di stagno le piazzole corrispondenti del jumper JP1. Saldando la piazzola centrale con la 5V si imposta la tensione di alimentazione a 5V, mentre collegandola a 3V3 si fissa la tensione a 3,3 volt. Grazie al bootloader, non appena OMC21 viene inserito nella USB esso verrà riconosciuto come periferica e creata

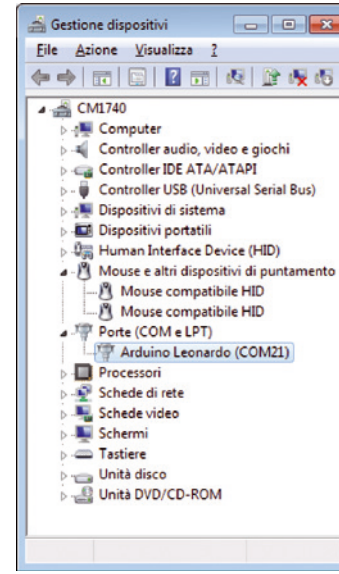
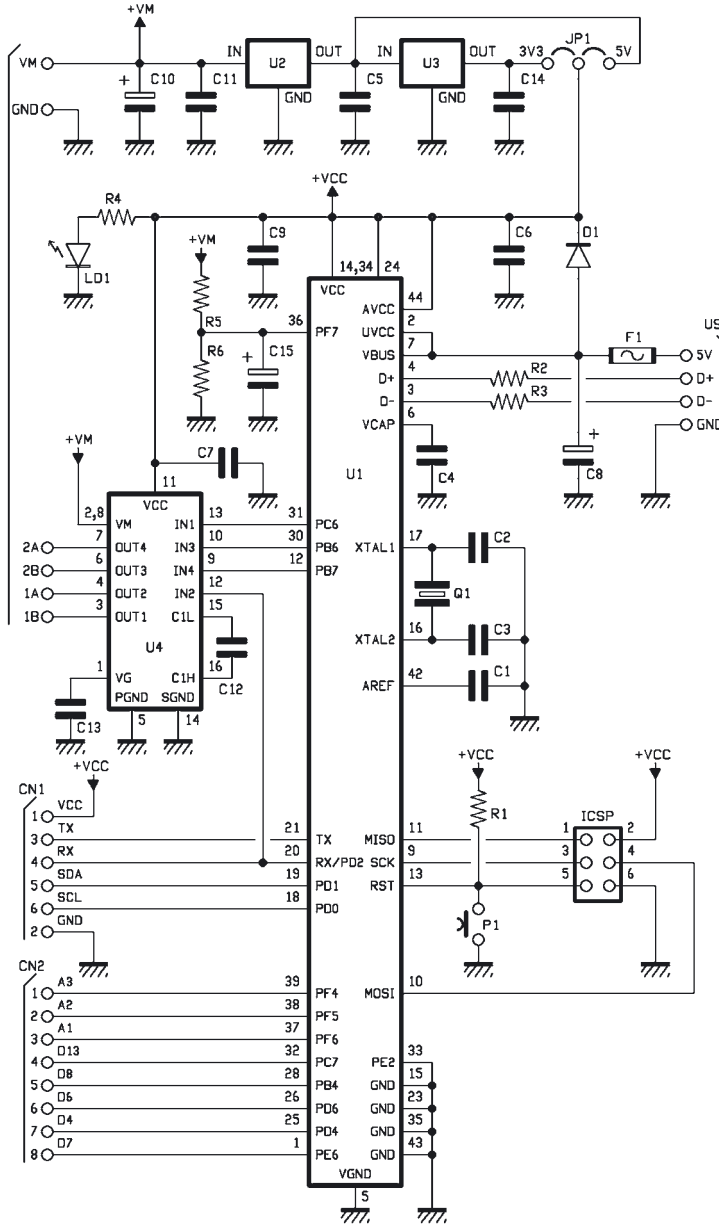


***Fig. 1** - L'OMC21 viene rilevato come periferica seriale Arduino Leonardo.*

Schema elettrico



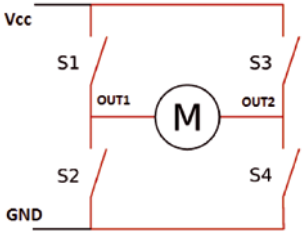


***Fig. 2** - Schematizzazione del ponte ad H.*

una seriale virtuale etichettata come Arduino Leonardo alla quale viene assegnato un nome: nel caso di **Fig. 1**, COM21.
A questo punto l'OMC21 può essere gestito completamente via USB. Leggendo attentamente il data-sheet del driver LV8405V-D e ricavando la tabella di verità (**Tabella 1**) relativa allo stato degli interruttori (S1-S4) e delle uscite (OUT1 e OUT2) per ogni possibile combinazione degli ingressi; la lettera L indica un livello logico basso e la lettera H un livello logico alto. Analizzando la **Tabella 1** è possibile capire più in dettaglio le diverse modalità di funzionamento a seconda di come vengono pilotati i due ingressi IN1 e IN2. Nel primo caso (IN1=H e IN2=H) entrambe le uscite si trovano in alta impedenza; in questa situazione è come se il motore fosse scollegato dal driver e libero di ruotare. Se invece ci poniamo nel quarto caso (IN1=L e IN2=L) è come se il motore si trovasse in cortocircuito; cosa cambia all'atto pratico tra i due casi? Se si utilizzano i motori per far avanzare un robot che sta procedendo a velocità sostenuta, nel primo caso, privando i motori dell'alimentazione, otterremo un arresto graduale del robot, che si fermerà in uno spazio dipendente dalla propria inerzia. L'arresto deriverà dai vari attriti negli organi meccanici e da quelli tra il robot e la superficie su cui si muoverà. Per comprendere cosa succede nella seconda ipotesi, occorre considerare che un motore in corrente continua è una macchina reversibile, ovvero se gli forniamo energia elettrica il suo motore ruota, ma se siamo noi a ruotarlo il motore fornirà energia elettrica, comportandosi, cioè, da generatore elettrico; in questo secondo caso, per il fenomeno della reazione d'indotto, lo sforzo richiesto per ruotare l'albero dipenderà da quanta corrente verrà prelevata dagli estremi degli avvolgimenti del motore.
Trasferendo questo concetto al nostro caso, diciamo che se quando il robot è in movimento togliamo tensione al motore, questo continuerà a girare per inerzia, ma se il ponte mette in cortocircuito i suoi terminali, la corrente che ne deriva nell'avvolgimento causa a sua volta un campo elettromagnetico tale da opporsi alla causa che l'ha generato; essendo tale causa la rotazione dell'albero, quest'ultimo verrà frenato fino a fermarsi, allorché mancando la corrente smetterà anche l'azione frenante. In molti shield commerciali queste due funzioni non sono selezionabili, mentre in questo circuito è stata prevista la possibilità di selezionare (via software) in quale modalità utilizzare le uscite.
I rimanenti due casi descritti nella **Tabella 1** sono invece più facili da comprendere: se poniamo l'ingresso IN1 a livello basso e comandiamo l'ingresso IN2 con un segnale PWM, abbiamo la possibilità di comandare il motore in direzione forward con una potenza direttamente proporzionale al valore del duty-cycle, il che permette di regolare la velocità di avanzamento del robot (quando il duty-cycle sarà zero fermeremo il robot con l'effetto freno precedentemente descritto). Se invece applichiamo il segnale PWM all'ingresso IN2, ma poniamo

***Tabella 1** - Tabella di verità del driver LV8405.*

IN1	IN2	OUT1	OUT2	S1	S2	S3	S4	funzione
H	H	Z	Z	open	open	open	open	standby
H	L	L	H	open	close	close	open	reverse
L	H	H	L	close	open	open	close	forward
L	L	L	L	open	close	open	close	brake

l'ingresso IN1 a livello alto, quando il duty-cycle ha valore massimo siamo nel caso delle uscite ad alta impedenza ed il robot si fermerà senza alcun effetto frenante.
Invertendo IN1 con IN2 avremmo le stesse funzioni, ma con il motore che ruoterà in senso inverso. Quindi giocando sugli ingressi, abbiamo la possibilità di scegliere direzione e velocità del motore e di attivare o meno l'effetto frenante.
Gli ingressi del driver IN1 e IN2 per il primo motore e IN3 e IN4 per il secondo motore, sono connessi a quattro uscite PWM dell'integrato ATmega32U4, facilmente gestibili con l'istruzione *analogWrite* disponibile nell'IDE di Arduino. Per il collegamento dei motori e dell'alimentazione di potenza è stata morsettiera a 6 poli. Abbiamo previsto anche la possibilità di leggere la tensione di alimentazione VM utilizzando un semplice partitore di tensione; per rendere compatibile il livello della tensione di alimentazione dei motori con quella di lettura degli ingressi analogici del microprocessore, la tensione viene ridotta di un fattore undici e quindi resa disponibile all'ingresso PF7 del microcontrollore. Conoscere la tensione permette, nel caso di alimentazione a batteria, di evitare la scarica completa, ovvero, in carica, di impedire la sovra-scarica (sempre dannosa). Al connettore CN2 fanno capo ulteriori pin del microprocessore, ovvero i rimanenti pin digitali non usati dal driver e tre pin analogici. In pratica è come se disponessimo di una scheda Arduino Leonardo con a bordo un driver per motori.

Il firmware / protocollo

Lo sketch prevede quindi una routine di ascolto sulle tre porte di comunicazione, in attesa di dati in arrivo; l'ATmega32U4 dispone di due moduli UART ed una porta I²C hardware, ideali per il nostro progetto. La seriale principale (Serial) fa capo al modulo USB e dal lato software si comporta esattamente come la seriale di Arduino Uno. In questa modalità l'OMC21 è controllabile tramite un PC o una qualsiasi periferica che disponga di una USB host. Abbiamo usato 9600 come velocità predefinita perché è quella più usata. Il protocollo di comunicazione che abbiamo pensato di utilizzare prevede l'utilizzo di quattro byte, di cui il primo è solo il carattere '$', necessario per riconoscere l'inizio della sequenza; questo, assieme

***Tabella 2** - Sequenza di byte per il comando del controller.*

byte1	byte2	byte3	byte4
start byte = '$'	modo e direzione	velocità del motore 1	velocità del motore 2

***Tabella 3** - Funzioni relative al byte2 della sequenza di comando.*

bit	funzione	condizione dei motori
0	motor 1 direction	0=foward, 1=reverse
1	motor 2 direction	0=foward, 1=reverse
2	motor 1 brake	0=brake, 1=standby
3	motor 2 brake	0=brake, 1=standby
4	read VM	0=null, 1=request motor volage
5	unused	-
6	unused	-
7	unused	-

al numero di caratteri inviati, rappresenta il controllo di correttezza nella lettura del comando. Terminata la lettura dei quattro byte, si verifica se ve ne siano altri; in ogni caso il buffer viene svuotato. Può capitare che inviando i comandi si aggiungano inavvertitamente dei caratteri non necessari, come ad esempio un fine riga o un ritorno a capo; nella nostra applicazione questi vengono ignorati e cancellati dal buffer. I rimanenti tre byte dopo il carattere '$' rappresentano i comandi per il controller: il secondo byte contiene informazioni sulla modalità di funzionamento (brake o standby) e sulla direzione di marcia dei motori, mentre i byte tre e quattro contengono l'informazione della velocità, rispettivamente, per il motore 1 e 2. In questo modo, inviando una sequenza di quattro byte si comandano contemporaneamente entrambi i motori (**Tabella 2**).
Il byte2 contiene informazioni sia sulla modalità di funzionamento che sulla direzione dei motori (**Tabella 3**). Il quinto bit del byte2, se posto a uno consente di richiedere la lettura della tensione di alimentazione; è l'unico caso in cui è necessaria la comunicazione seriale bidirezionale, altrimenti la sola linea di invio dati al controller è sufficiente.
Se si desidera comandare il controller tramite la seriale di un microcontrollore, è necessario utilizzare la seconda porta seriale (Serial1) del microcontrollore ATmega32U4 i cui pin TX e RX sono disponibili nel connettore CN1. La connessione prevede che la linea TX del microcontrollore sia connessa alla linea RX di OMC21 e se richiesta la lettura della tensione di alimentazione, anche la linea RX del microcontrollore connessa alla linea TX di OMC21. Per il resto la gestione a livello software è la stessa della seriale principale. Per quanto riguarda la porta di comunicazione I²C. Ad ogni arrivo di dati su questa porta viene richiamata la procedura che si preoccupa di verificare se la sequenza è compatibile con lo standard OMC21 e quindi ne estrapola i valori. Lo sketch per OMC21 è quindi predisposto per monitorare continuamente l'invio dei byte di comando su tutte e tre le porte di comunicazione: USB, seriale e I²C.

Gestione tramite USB

A titolo di esempio vediamo come costruire i tre byte dati per il controllo dei motori nel caso volessimo il motore 1 con direzione forward con potenza del 50% e funzione frenatura attivata ed il motore 2 con direzione reverse con potenza del 25% senza frenatura (standby). Essendo la velocità espressa come valore numerico compreso da 0 (0% della potenza) a 255 (100% della potenza), ai due valori 50% e 25% corrispondono rispettivamente i valori decimali 128 e 64. Il byte2 dovrà essere costruito bit per bit come nella **Tabella 4**.
I tre byte di comando saranno quindi quelli riportati in **Tabella 5**. Se il nostro controller è connesso alla seriale difficilmente potremmo controllarlo utilizzando la funzione SerialMonitor di Arduino, per il semplice motivo che i primi 31 caratteri del codice ASCII non sono stampabili, in quanto usati come comandi relativamente alla comunicazione con le stampanti. Se vogliamo inviare i quattro byte così costruiti è necessario utilizzare un monitor seriale che permetta di gestire anche i caratteri non stampabili; nel nostro caso abbiamo usato un software gratuito, distribuito in versione *portable* di nome SSCOM (**Fig. 3**), ma tanti altri sono disponibili in rete. Nell'esempio sono stati inviati (cerchio rosso) i byte 0x24 0x10 0x8F 0x00 corrispondenti a motore 1 fermo e motore 2 forward al 56% e richiesta tensione di alimentazione VM.
Evidenziati con un cerchio blu, i dati di risposta relativi ad una tensione di alimentazione di circa 6,8 volt. Selezionare COM nel menu a tendina ComNum; se, come spesso accade, non disponete di COM hardware ed al PC è connesso solo OMC21, nella lista trovate una sola COM ad esso relativa.

Gestione via Seriale

Se si vuole comandare l'OMC21 tramite una scheda Arduino, conviene utilizzare una seriale software, utilizzando una sola linea digitale da connettere alla linea RX del connettore CN1; in questo modo la seriale hardware rimarrà libera di comunicare con il PC.

***Tabella 4** - Costruzione del byte2 per l'esempio indicato.*

bit	funzione	valore
0	motor 1 direction forward	0
1	motor 2 direction reverse	1
2	motor 1 brake ON	0
3	motor 2 brake OFF	1
4	read VM	0
5	unused	0
6	unused	0
7	unused	0

Gestione tramite I²C

Se si intende controllare l'OMC21 tramite la porta TWI di Arduino, è necessario semplicemente connettere le due linee SDA e SCL di entrambe le porte tra loro e collegare i GND di entrambe le schede. Non sono neces-

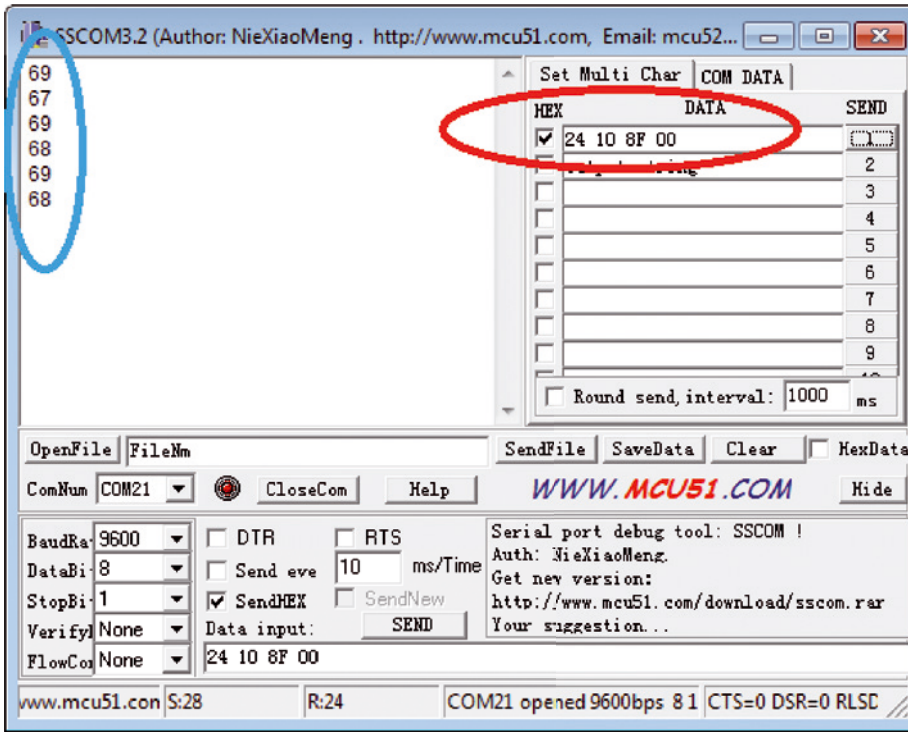


***Fig. 3** - Monitor Seriale SSCOM per inviare i comandi ad OMC21.*

***Tabella 5** - Sequenza di byte per l'esempio indicato.*

formato	byte1	byte2	byte3	byte4
binario	B00011000	B00001010	B10000000	B01000000
decimale	D36	D10	D128	D64
esadecimale	0x24	0x0A	0x80	0x40

sarie le resistenze di pull-UP per il semplice motivo che sono già presenti internamente al microcontrollore.
C'è però un'importante differenza con il modo di comunicare tramite seriale che riguarda il sistema di indirizzamento delle periferiche. Mentre in una comunicazione seriale lo scambio di dati avviene punto a punto, ovvero solo due periferiche dialogano tra loro, nel Bus I²C possono coesistere sino a 127 periferiche di tipo Slave, tutte in comunicazione con l'unico master. Il master in questo caso è Arduino Uno (o la logica di comando) mentre gli slave sono le schede OMC21; il vantaggio evidente è quello di poter utilizzare più controller e quindi più motori, tutti gestiti da un'unica scheda Arduino utilizzando una sola porta TWI.
Ovviamente sul BUS TWI potranno coesistere anche altri slave non necessariamente controller motori OMC21. Nello sketch *OMC21.ino* (già caricato) è stato specificato come indirizzo predefinito di periferica il valore 4, ma questo può essere modificato a piacimento se intendete usare sul bus una periferica che ha già questo indirizzo.
Se vengono utilizzati più controller OMC21 è necessario che ciascuno abbia un indirizzo univoco; sarà necessario quindi caricare su ciascun controller uno sketch in cui sarà stato specificato un indirizzo differente. Nella stesura del programma di controllo bisognerà poi tenere conto dei vari indirizzi assegnati, per assicurare il corretto transito dei dati.

Utilizzo del controller

Sono possibili numerose combinazioni per l'alimentazione della logica di controllo e della sezione di potenza dell'integrato driver LV8405V-D; la **Fig. 4** mostra il collegamento generale dei due motori in cc (per lo stepper-motor bipolare ogni avvolgimento va al posto di un motore) e dell'alimentazione della scheda.
Invece, nella **Fig. 5** viene illustrato come impostare il ponticello 3V3/5V (in questo caso va lasciato completamente aperto) di selezione dell'alimentazione del microcontrollore e della logica del driver LV8405V-D affinché venga prelevata direttamente dalla connessione USB; nella stessa figura i motori vengono alimentati da VM.
La **Fig. 6** mostra come impostare il ponticello se si desidera alimentare il circuito di potenza con VM e il microcontrollore e la logica del driver con i 3,3 volt ricavati dalla stessa VM tramite il regolatore U3 (il ponticello va realizzato chiudendolo con una goccia di stagno).
Nella **Fig. 7** è invece proposta l'impostazione del ponticello ammettendo di alimentare il microcontrollore e la logica del driver con i 5 volt che il regolatore U2 ricava da VM (tensione d'ingresso del circuito e di potenza).
Vengono esaminate ora le connessioni con Arduino Uno per il controllo seriale: le **Fig. 8** e **Fig. 9** riguardano rispettivamente la con-

***Tabella 6** - Funzioni del connettore CN1.*

	byte4
1	+5V
2	GND
3	TX serial
4	RX serial
5	SDA
6	SCL

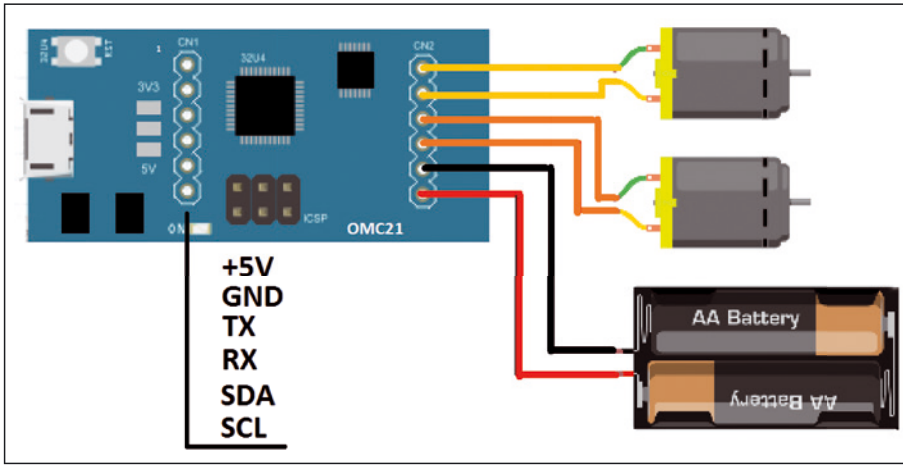


***Fig. 4** - Collegamento di motori e alimentazione al controller OMC21.*

nessione via seriale TTL e quella tramite I²C-Bus. Aggiungiamo ora alcune raccomandazioni per il corretto utilizzo di questo controller, tenendo conto di tutte le problematiche che si possano incontrare all'atto pratico nell'utilizzo dei motori a spazzole.
È buona norma prima di collegare un motore verificare se la massima corrente assorbita (di solito chiamata corrente di stallo) non eccede il valore massimo consentito. Un'altra buona norma è di saldare direttamente sui contatti del motore tre condensatori ceramici del valore compreso tra 10 nF e 100 nF, con lo scopo di sopprimere i disturbi derivati dallo strisciamento delle spazzole sul collettore. I condensatori vanno saldati tra i due terminali e la carcassa ed eventualmente anche uno tra i due terminali di alimentazione

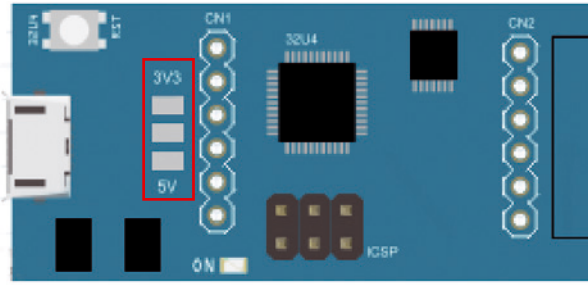


***Fig. 5** - Motori alimentati da VM e ATmega32U4 alimentato esternamente dalla logica di controllo.*

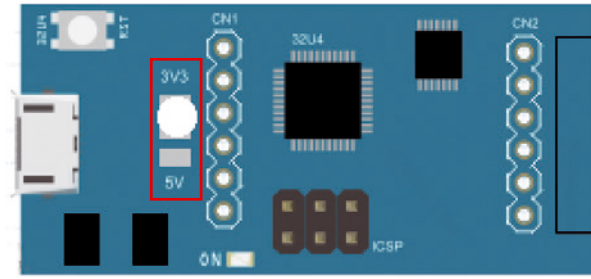


***Fig. 6** - Motori alimentati da VM e ATmega32U4 alimentato a 3,3V da VM.*

(vedere **Fig. 10**).
Riportiamo anche un esempio pratico di utilizzo del nostro controller utilizzandolo in sostituzione della circuiteria elettronica guasta in una piccola automobilina RC. In questo esempio utilizziamo l'OMC21 per leggere i segnali provenienti da un ricevitore RC, di quelli usati in ambito modellistico, e co mandare il motore dell'automobilina. La ricevente è alimentata dallo stesso OMC grazie allo stabilizzatore di tensione interno impostato sui 5V. Il servo per lo sterzo è controllato direttamente dalla ricevente RC. L'alimentazione giunge da una piccola batteria LiPo connessa direttamente ad OMC21. Il programma da inserire sul controller si chiama *esc_rc_car2.ino* scaricabile dalla scheda on-line del prodotto e prevede anche una funzione di diagnostica grazie alla quale su serial monitor è

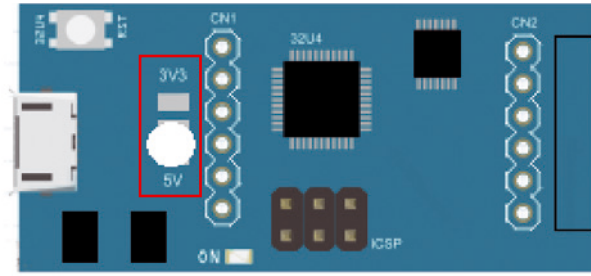


***Fig. 7** - Motori alimentati da VM e ATmega32U4 alimentato a 5V da VM.*

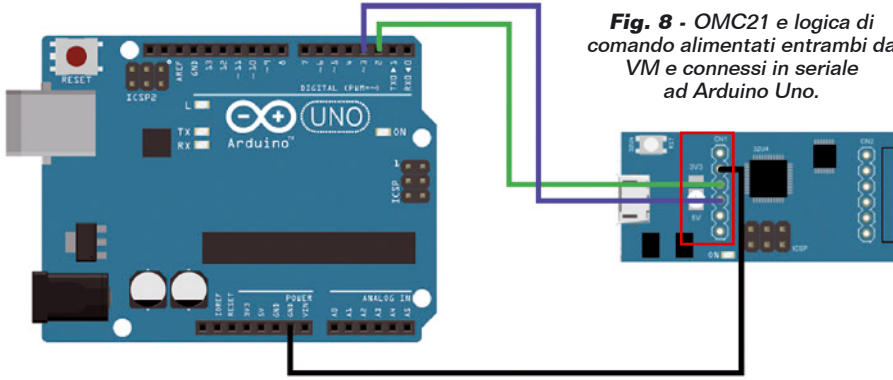


***Fig. 8** - OMC21 e logica di comando alimentati entrambi da VM e connessi in seriale ad Arduino Uno.*

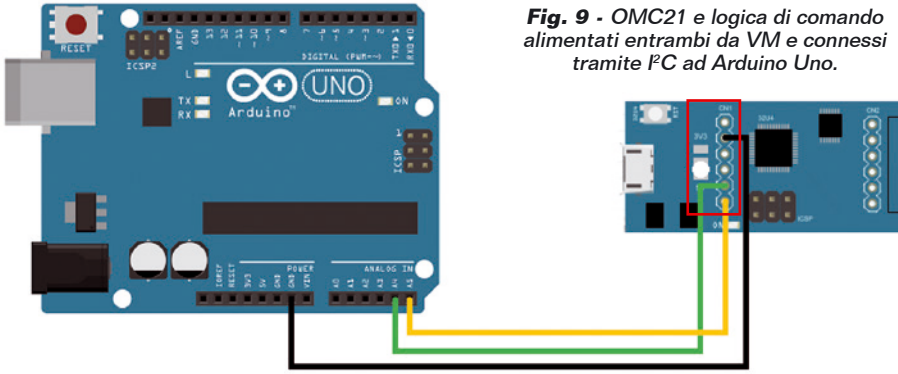


***Fig. 9** - OMC21 e logica di comando alimentati entrambi da VM e connessi tramite I²C ad Arduino Uno.*

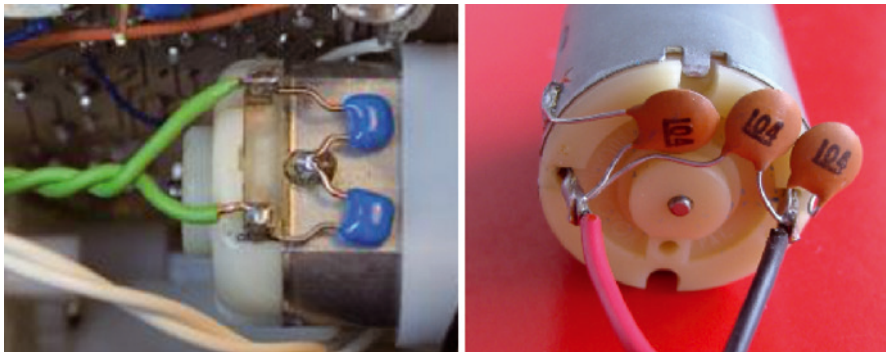


Fig. 10 *- Utilizzo dei condensatori come soppressori dei disturbi su un motore a spazzole.*

possibile visualizzare i dati ricevuti dalla ricevente RC (**Fig. 11**).
Il programma è molto semplice e si basa sulla lettura della durata dell'impulso ricevuto dalla ricevente RC grazie alla funzione *pulseIn* di Arduino; da questa lettura è poi sufficiente ricavare il giusto valore del duty-cycle per il comando dei motori. Per il resto sono state usate le funzioni già precedentemente descritte, con in aggiunta la lettura della tensione della batteria, impostando lo spegnimento motore (cut-off), non appena questa scenda al di sotto dei 6V, al fine di salvaguardare la batteria di alimentazione.

L'articolo completo del progetto è stato pubblicato su: Elettronica In n. 195

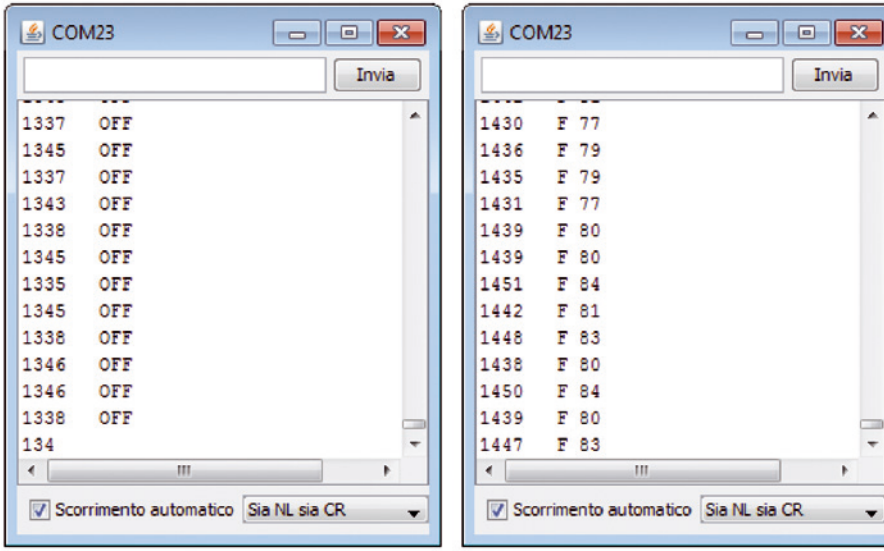


Fig. 11 *- Dati ricevuti dalla ricevente RC.*

A tutti i residenti nell'Unione Europea
Importanti informazioni ambientali relative a questo prodotto

Questo simbolo riportato sul prodotto o sull'imballaggio, indica che è vietato smaltire il prodotto nell'ambiente al termine del suo ciclo vitale in quanto può essere nocivo per l'ambiente stesso. Non smaltire il prodotto (o le pile, se utilizzate) come rifiuto urbano indifferenziato; dovrebbe essere smaltito da un'impresa specializzata nel riciclaggio.
Per informazioni più dettagliate circa il riciclaggio di questo prodotto, contattare l'ufficio comunale, il servizio locale di smaltimento rifiuti oppure il negozio presso il quale è stato effettuato l'acquisto.

Distribuito da:
FUTURA GROUP SRL
Via Adige, 11 - 21013 Gallarate (VA) Tel. 0331-799775 Fax. 0331-792287
web site: www.futurashop.it
supporto tecnico: www.futurashop.it/Assistenza-Tecnica