Ministero dell'Istruzione
dell'Università e Ricerca

ITIS
LEONARDO DA VINCI
Via Toscana,10 43122 PARMA Tel.0521266511 fax 0521266550 e-mail itis@itis.pr.itc.f.80007330345 Cod.PRTF010006

AGV differenziata

Bruschi, Rocchi, Simonazzi

Classe 5ªA LEN

A.S. 2013/2014

1. INTRODUZIONE

Oggi la riduzione dell'impatto ambientale è un tema che si trova al centro dell'attenzione in qualunque ambito in cui l'uomo agisce. Al primo posto abbiamo l'impatto che un'industria può causare con la sua inevitabile interazione con l'ambiente. Per ridurre l'impatto ambientale la soluzione più semplice e, se attuata con continuità, anche la più efficace è la raccolta differenziata dei rifiuti in vista di un futuro riciclaggio e riutilizzo dei materiali di scarto. È facile però comprendere che una soluzione di tale tipo richieda un ingente investimento a causa degli elevati costi dovuti soprattutto alla manodopera, all'organizzazione e alla gestione di quest'ultima. Per un'azienda risulterebbe eccessivamente oneroso e pertanto inattuabile un completo e corretto processo di raccolta differenziata dei rifiuti.

Il nostro progetto nasce per portare una soluzione innovativa che consentirebbe di risolvere in gran parte il problema facendo uso dell'automazione.

AGV differenziata è un prototipo di veicolo a guida autonoma (AutomatedGuidedVehicle) in grado di automatizzare il processo di raccolta differenziata dei rifiuti.

Mediante un progetto di questo tipo è possibile abbattere notevolmente gli oneri organizzativi e di mano d'opera che potrà essere impiegata nella fase di riciclaggio.

2. DESCRIZIONEGENERALE

Il progetto consiste in un prototipo di veicolo a guida autonoma (AutomatedGuidedVehicle) equipaggiato di strumentazione necessaria alla raccolta differenziata dei rifiuti, utilizzabile in ambito industriale o per infrastrutture con ambienti spaziosi; ciò non esclude un impiego nella raccolta differenziata urbana.
L'unico intervento umano richiesto è di supervisione. Questo controllo viene effettuato con un computer remoto collegato al camion tramite Wi-Fi.
Nel nostro progetto, il camion, segue un percorso (chiuso) contrassegnato da una pista nera posta su un pavimento bianco. Con opportune modifiche al software si possono invertire i colori nel caso di pavimento di colore opposto. Tramite un apposito segno sulla pista (una striscia bianca perpendicolare alla pista) il camion è in grado di fermarsi davanti al bidone. Successivamente attiverà il sistema di riconoscimento della tipologia del bidone effettuando una discriminazione del colore dello stesso. Se si tratta del materiale desiderato il bidone sarà caricato per svuotarne il contenuto. In caso contrario si passa ad analizzare il bidone successivo.
Al termine del giro il veicolo ritornerà alla posizione di partenza.
Istante per istante il camion comunica al computer remoto il suo stato attuale. Inoltre l'utente addetto allasupervisione sarà in grado di selezionarequale ciclo di raccolta intraprendere.
È stato predisposto un sistema di sicurezza nella parte anteriore del camion per evitare danni al mezzo e per garantire l'incolumità delle persone. In tal modo il veicolo, rileva la presenza di ostacoli ed è così in grado di fermarsi prima di urtarli. Se dovesse incontrare qualche ostacolo, lo comunicherà al computer remoto. Nel caso non venisse intrapresa nessuna azione da parte dell'operatore in un tempo utile stabilito e l'ostacolo non venisse più rilevato il veicolo proseguirà la sua fase di lavoro.

AGV differenziata è un sistema automatizzato, telecontrollato e dotato d'interfaccia grafica per la supervisione.

3. PARTE HARDWARE

3.1 SCHEMA LOGICO DI FUNZIONAMENTO

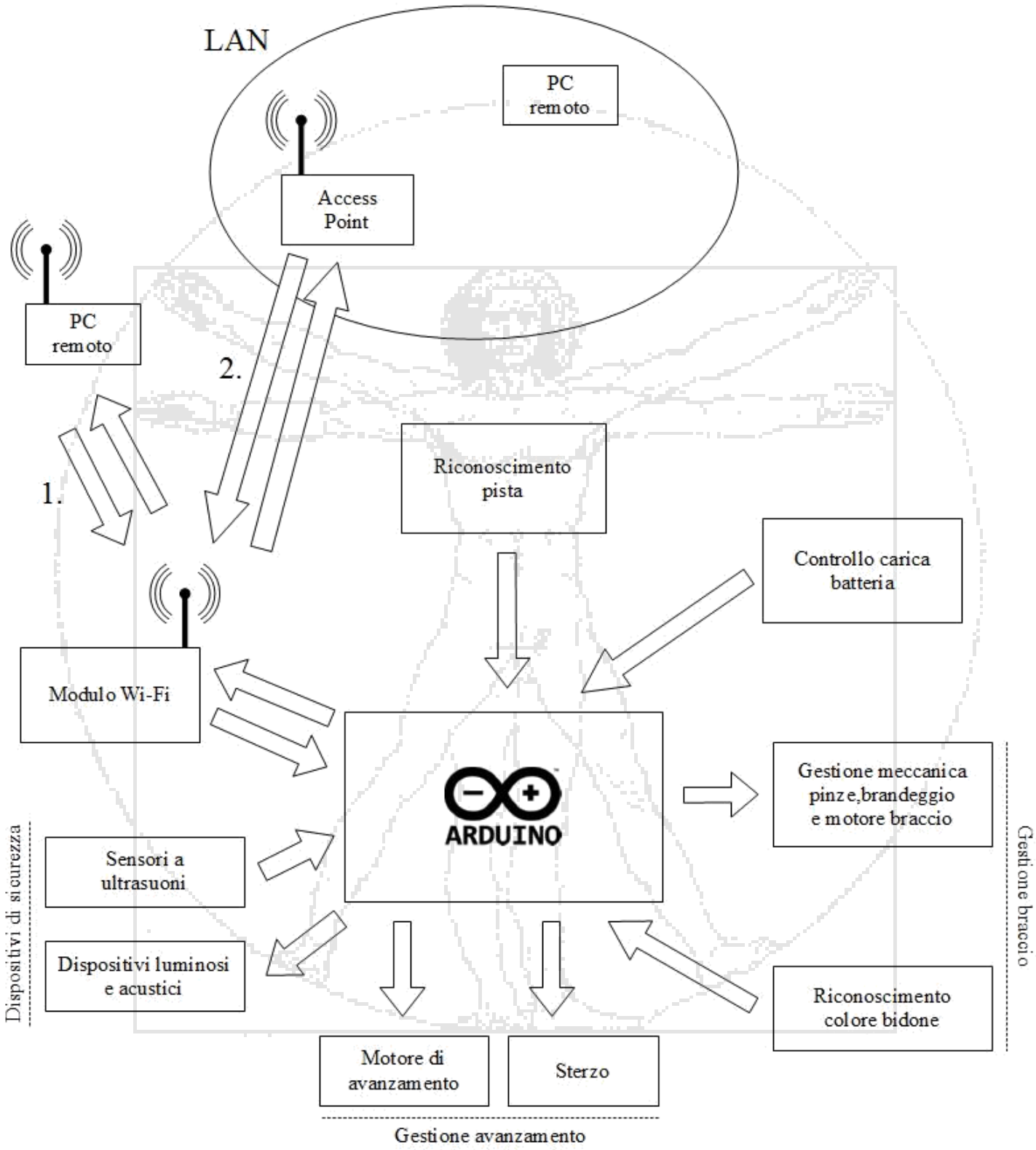


Fig. 1

3.2 DESCRIZIONE BLOCCHI

3.2.1 Sistema di controllo centrale

Il sistema di controllo centrale utilizzato è la scheda Arduino Mega 2560 basata sul microcontrollore Atmel ATmega2560.

Caratteristiche:

- 54 pin di input/output digitale di cui:
  - 15 pin configurabili in funzionamento PWM
  - 8 pin (4 TX e 4 RX) per la comunicazione su 4 porte seriali TTL
- corrente massima erogabile da ogni pin pari a 40mA
- 16 input analogici (convertitore A/D con 10 bit di risoluzione)
- 256 KB di memoria flash
- 4 KB di memoria EEPROM
- processore con velocità di clock di 16 MHz
- dotato di un modulo USB to TTL serial per la comunicazione seriale
- stabilizzatore di tensione interno a 5V e 3,3V
- alimentazione:
  - da 7V a 12V
  - 5V se alimentato tramite USB

3.2.2 Riconoscimento pista

Questo modulo permette di riconoscere la pista che il veicolo dovrà seguire. Esso è realizzato tramite 4 fotoresistenze e un LED bianco ad alta intensità posto al centro delle due centrali. Le due fotoresistenze interne sono posizionate a una distanza tale per cui in condizioni di spostamento rettilineo esse si trovino entrambe sulla traccia nera. Quelle esterne invece si trovano sul pavimento. Le fotoresistenze sono state opportunamente schermate per evitare che venissero direttamente influenzate dal led e da fonti di luceesterne in modo da recepire solo la luce riflessa dalla pista.
I segnali analogici prodotti dalle fotoresistenze, proporzionali alle intensità luminose rilevate, sono digitalizzati tramite Arduino.

Schema:

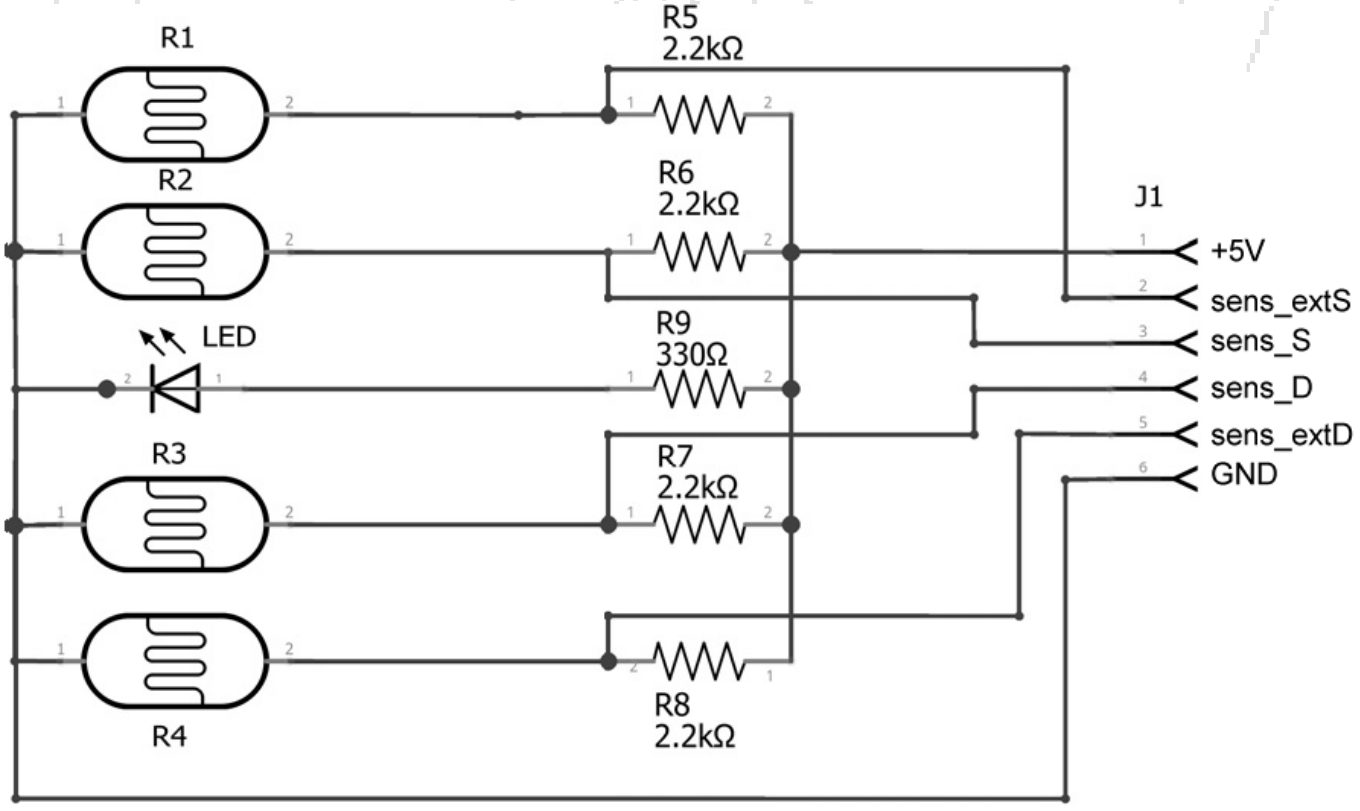


Fig.2

Funzionamento:
Le fotoresistenze sono resistori la cui resistenza varia in proporzione all'intensità del fascio luminoso che la investe.In particolare ad un aumento d'intensità luminosa corrisponde una diminuzione del valore ohmico della resistenza.
Per convertire la variazione di resistenza in variazione di tensione si utilizza un partitore di tensione. Poiché l'uscita del partitore corrisponde alla caduta di tensione sulla fotoresistenza, si ottieneche ad un aumento di luminosità si ha una diminuzione di tensione; viceversa ad una diminuzione di luminosità si ha un aumento di tensione.
La lettura del valore di tensione d'uscita del partitore viene fatto tramite gli ingressi analogici del microcontrollore Arduino.La chiamata di un'apposita funzione via software si occuperà di ottenere dal convertitore A/D integrato, un valore decimale compreso tra 0 e 1023, corrispondente al valore binario che si ottiene sui 10bit del convertitore.
La gestione di avanzamento e di correzione di direzione è effettuata tramite una lettura di tensione ad intervalli regolari molto brevi dei valori d'uscita dei due partitori delle fotoresistenze esterne. Se si leggono due valori di tensioni bassi (corrispondenti ad una presenza di luce e quindi al bianco) le due fotoresistenze si trovano ai lati della pista e che quindi il veicolo sta andando nella direzione corretta. Se invece viene rilevato un solo valore di tensione alto (corrispondente ad una luminosità minore e quindi al nero) bisogna correggere la direzione nel verso opposto perché il veicolo sta uscendo dalla pista.
Nel caso si rilevino due tensioni basse dai partitori delle fotoresistenze interne il veicolo si ferma poiché si trova in prossimità di un bidone.
Bisogna progettare il percorso in modo che l'angolo delle curve non superi l'angolo massimo di sterzo altrimenti il veicolo finirebbe fuori pista.

Collaudo:
Per la prima fase di collaudo il circuito è stato montato su una basetta millefori. Dopodiché sono stati letti e digitalizzati i valori di luminosità rilevati dalle fotoresistente ed è stato verificato che:

- nel caso in cui le fotoresistenze si trovassero a leggere la stessa intensità luminosa, lo scostamento tra i valori letti fosse minimo e soprattutto sufficientemente stabile
- nel caso ciascuna si trovasse a leggere un colore differente, lo scostamento tra i valori letti fosse sufficientemente elevato per permettere una corretta discriminazione.

Durante questa fase si è constatato che isolando le fotoresistenze dalla luce esterna e posizionando un led al centro di esse la precisione dei valori rilevati era maggiore e costante.
Al termine la scheda è stata testata insieme ai moduli per la gestione dell'avanzamento per verificare che la traccia venisse seguita in modo corretto.

3.2.3 Gestione avanzamento

Il blocco di gestione dell'avanzamento del veicolo è costituito da due parti:

- motore di avanzamento
- sterzo

Motore di avanzamento

L'avanzamento del veicolo è reso possibile da un motore in corrente continua collegato al differenziale che provvederà a distribuire la coppia alle ruote motrici (posteriori).
Per il pilotaggio di potenza è stato utilizzato il circuito integrato L298 che ha consentito di pilotare il motore tramite la logica TTL del microcontrollore, permettendo di gestire il verso di rotazione, l'azionamento e l'arresto rapido del motore. Per la regolazione della velocità angolare del motore si è invece fatto uso dei moduli PWM messi a disposizione dal microcontrollore.

Schema:

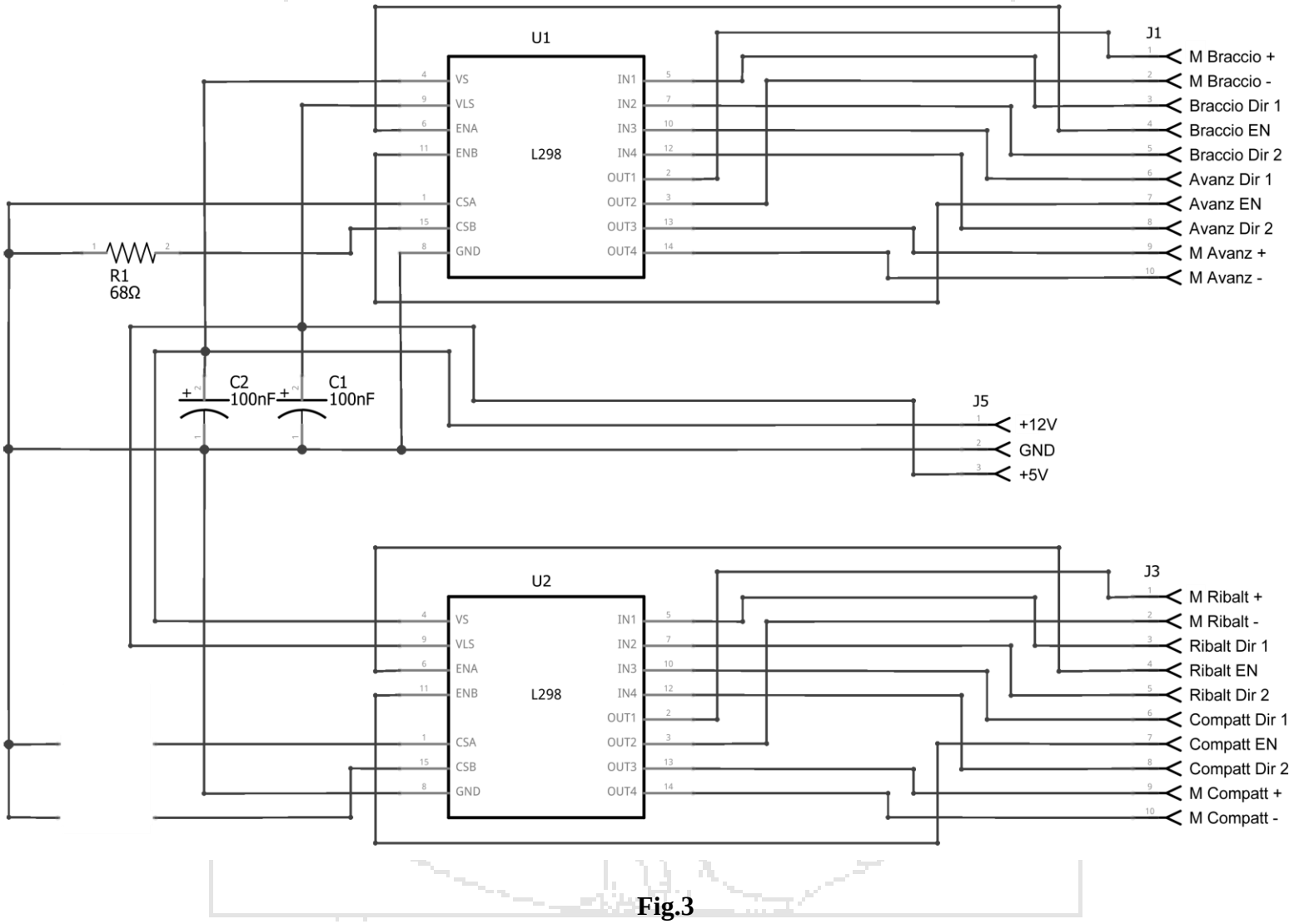


Fig.3

Funzionamento:
Il pilotaggio di potenza del motore è stato utilizzato con il circuito integrato L298 che è un ponte ad H per il pilotaggio di motori in DC.
In questo circuito integrato sono presenti due ponti ad H formati ciascuno da 4 transistor npn. Sulle basi di questi transistor sono presenti delle porte logiche AND.

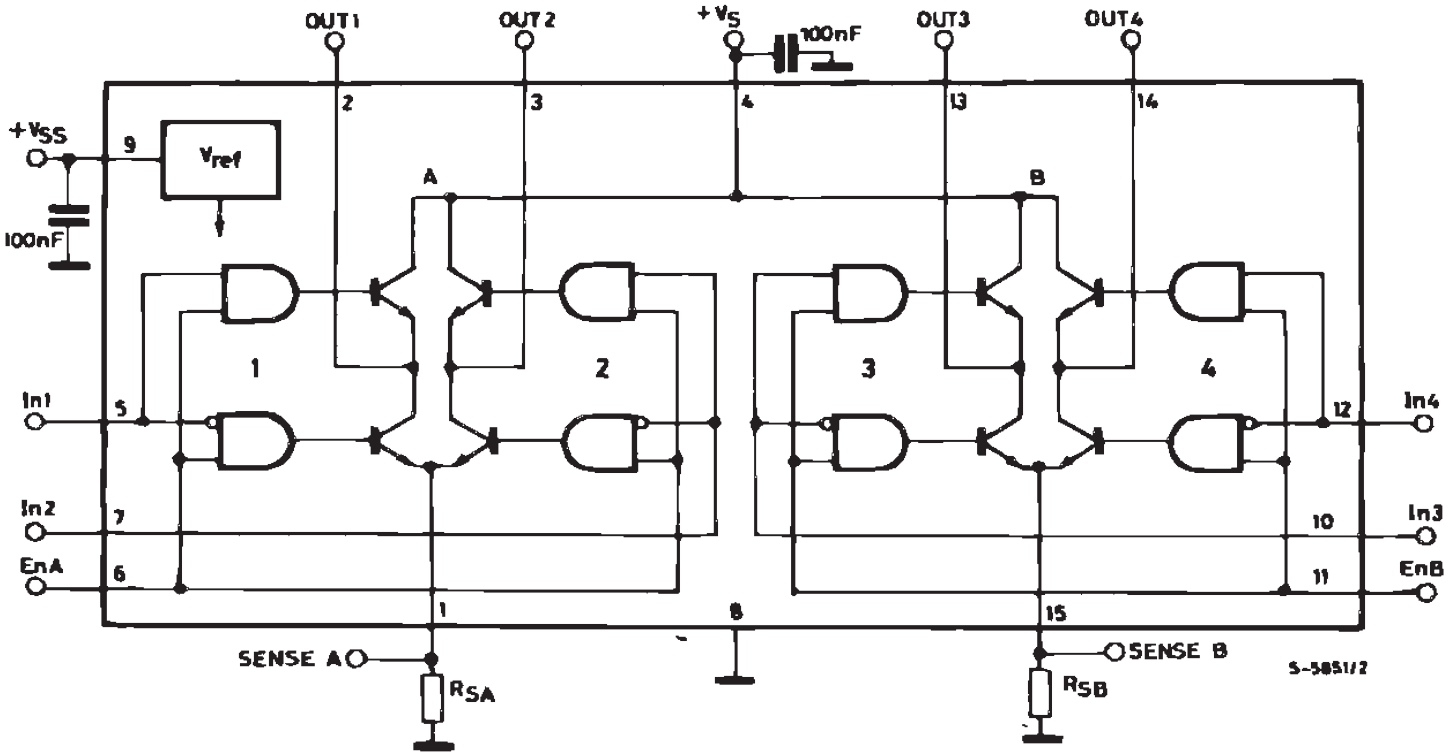


Fig.4

I due pin EnAeEnb servono per abilitare ciascun ponte, mentre i pin In1, In2, In3, In4 servono per la direzione del motore. In particolare:

- quando In1 (In4) è alto mentre In2 (In3) è basso il motore ruoterà in un senso
- quando In1 (In4) è basso mentre In2 (In3) è alto il motore ruoterà nell'altro senso
- quando In1 (In4) e In2 (In 3) hanno lo stesso valore il motore si fermerà.

Poiché nel nostro caso il pilotaggio dei motori avviene con tecnica PWM si è deciso di utilizzare i due pin (Ena/Enb) come pin di pilotaggio del motore. Su questi due pin verrà inviata dal microcontrollore l'onda rettangolare necessaria per pilotare il motore: quando l'uscita del microcontrollore connessa al pin di enable si trova a valore logico alto (Ton), il ponte è abilitato, e il motore ruoterà nella direzione precedentemente impostata; quando invece il microcontrollore da un'uscita bassa (Toff=T-Ton), il ponte è interdetto e il motore si ferma.
Sulla scheda sono presenti due L298. Questa scelta è stata fatta per predisporre il progetto ad un eventuale inserimento di altri due motori: uno per il sollevamento del cassone e l'altro per il sistema di compattamento dei rifiuti.

Collaudo:
Per prima cosa è stato testato il motore a pieno carico, in condizioni di carico normale e a vuoto per vedere l'assorbimento di corrente. Durante questa fase è stato rilevato che l'assorbimento massimo di corrente (2,5A) era inferiore alla corrente massima erogata dal circuito integrato (3A per canale fino ad un massimo di 4A totali) e che quindi non erano necessari altri dispositivi di potenza per pilotare il motore. Successivamente il circuito è stato realizzato su basetta millefori e dopo aver verificato che il motore rispondesse correttamente ai comandi è stato realizzato il circuito stampato.

Sterzo:

Il blocco dello sterzo è costituito da un servomotore il cui albero va ad imprimere la necessaria rotazione agli organi di sterzo del veicolo.

Schema:

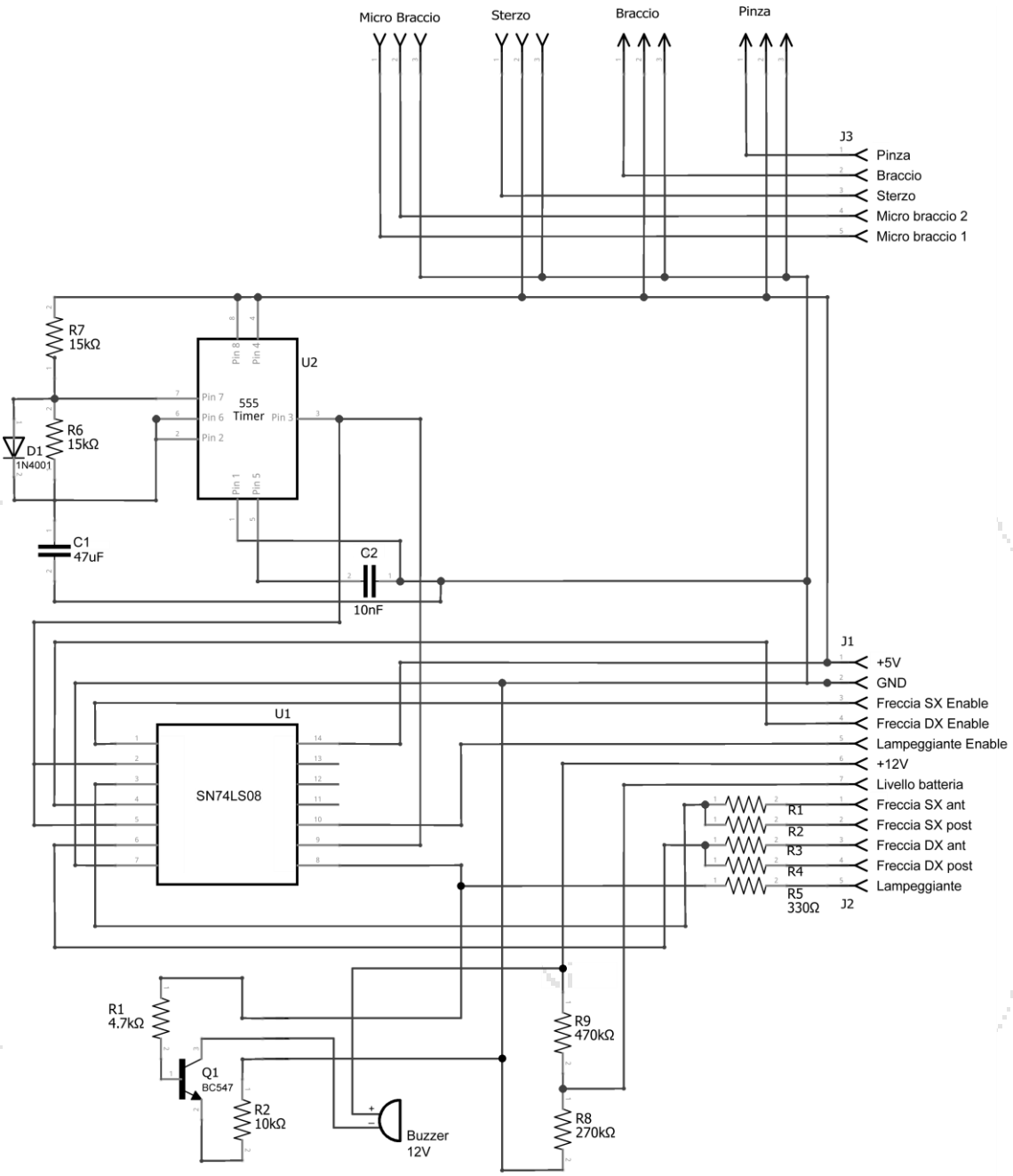


Fig. 5

Funzionamento:

Il servomotoreè costituito da un motore elettrico,da un meccanismo di demoltiplica che consente di aumentare la coppia in fase di rotazione e dall'elettronica di controllo. La rotazione del motore è effettuata tramite un circuito di controllo interno a retroazione in grado di rilevare l'angolo di rotazione raggiunto dal perno tramite un potenziometro resistivo e bloccare il motore sul punto desiderato.

Il servomotore viene gestito completamente da Arduino. Su questa scheda sono presenti solo dei connettori che consentono di collegare il servomotore al microcontrollore.Questo motore dispone di 3 pin: uno per l'alimentazione (5V), uno per la massa e l'altro per il controllo.I servomotori sono pilotati tramite un'onda rettangolare con periodo di 20ms applicata al pin di controllo. La durata degli impulsi varia la posizione dell'albero del servomotore. In particolare:

- impulso di 0,5ms: posizionedell'albero 0°
- impulso di 1,5ms: posizione dell'albero90°
- impulso di 2,5ms: posizione dell'albero180°

Collaudo:
Arduino semplifica la gestione dei servomotori fornendo una libreria dedicata, perciò non sono necessari particolari circuiti di controllo da testare. Per il collaudo di questo blocco è stato utilizzato un semplice softwaretramite il quale sono stati rilevati per via sperimentale gli angolo relativi alle massime posizioni che gli organi di sterzo sono in grado di raggiungere senza sforzi meccanici.

3.2.4 Dispositivi di sicurezza

Il blocco dei dispositivi di sicurezza è diviso in due parti:

- Sensore a ultrasuoni
- Dispositivi luminosi e acustici

Sensore a ultrasuoni:
Questo blocco è formato da una scheda industriale fornita di un trasmettitore e un ricevitore ad ultrasuoni. Il compito di questo blocco è rilevare la presenza di eventuali ostacoli sulla pista e garantire così una maggiore sicurezza nell'ambiente di lavoro.

Schema:

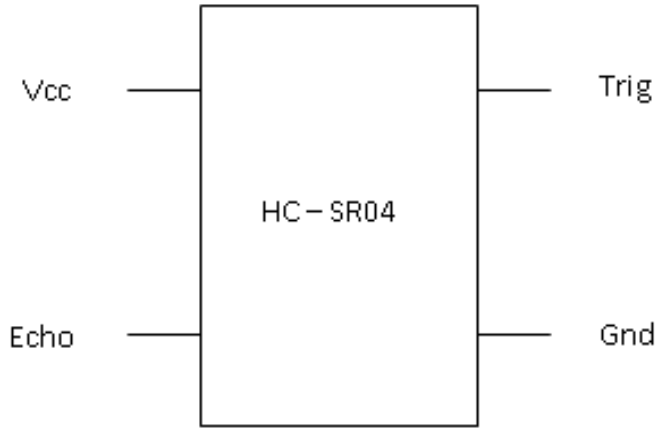


Fig. 6

Funzionamento:
La scheda da noi utilizzata è la HC-SR04. Questa scheda ha 2 pin per il controllo e 2 per l'alimentazione:

- Vcc: pin di alimentazione a 5V.
- Trig: pin di trigger. Su questo pin bisogna fornire un impulso di 10µs. Dopo aver fornito questo segnale verranno trasmessi 8 impulsi con frequenza di 40KHz.
- Echo: su questo pin verrà mantenuto un impulso di durataproporzionale al tempo di ritorno del segnale a 40KHz.
- Gnd: pin di massa.

Dopo aver fornito l'impulso di 10µs e trasmesso il segnale a 40KHz un timer inizia a contare il tempo che impiega il segnale per essere ricevuto. Quando il ricevitore rileva il segnale di ritorno il timer viene fermato e sul pin di echo viene mantenuto un impulso pari al tempo del conteggio del timer. Tramite software bisogna poi convertire questo tempo in una distanza tramite la seguente formula:

$$\text{distanza[cm]} = \text{tempo[µs]} / 58$$

Collaudo:
Per il collaudo di questo blocco è stato realizzato un semplice software che fosse in grado di misurare a quale distanza un oggetto si trovasse rispetto al sensore a ultrasuoni. Una volta confermato che la distanza misurata fosse corretta si è montata la scheda sul veicolo e si è integrato il software principale.

Dispositivi luminosi e acustici:
Questo blocco è costituito da LED lampeggianti posti sulla cabina e ai lati del veicolo e da un dispositivo di segnalazione acustica. Durante la marcia il LED sulla cabina continuerà a lampeggiare mentre gli altri dispositivi si attiveranno solo durante la fase di carico e scarico dei bidoni. Durante questa fase sarà attivo anche il dispositivo acustico posto all'interno della cabina del veicolo.

Schema:
Per lo schema si faccia riferimento alla **Fig. 5**.

Funzionamento:
I dispositivi lampeggianti sono costituiti da 5 LED ad alta intensità luminosa mentre il dispositivo acustico è costituito da un buzzer a 12V. Essi sono pilotati tramite un'onda quadra con frequenza di 1Hz. Quest'onda è generata grazie al circuito integrato NE555. Esso è un timer che può essere usato in diverse configurazioni. Nel nostro caso verrà utilizzato in configurazione di oscillatore astabile che consente di creare un'onda quadra con periodo 1s e Ton 500ms.
Per controllare l'accensione dei vari dispositivi luminosi e del dispositivo acustico si è fatto uso di porte logiche AND (integrato 74LS08) impiegate come buffer digitali. Un ingresso di ogni porta è utilizzato come ingresso del buffer mentre l'altro come enable. Sugli ingressi dei buffer è presente il segnale proveniente dall'NE555.
Quando l'enableha valore alto (1 logico) l'onda quadra verrà copiata sull'uscita mentre se ha valore basso (0 logico) l'uscita sarà nulla.

Collaudo:
Per prima cosa il circuito è stato realizzato e testato su una breadboard. Dopo aver verificato il periodo e il Duty Cycle dell'onda quadra è stato controllato che cambiando il valore d'ingresso del pin di controllo della porta AND l'uscita assumesse il valore desiderato, ovvero che l'onda quadra fosse copiata sull'uscita. Una volta verificato ciò, è stato realizzato il circuito stampato.

3.2.5 Gestione braccio
Il braccio è stata la parte di hardware più complessa da costruire. Il primo problema è stato quello di realizzare un progetto di sollevamento che fosse attuabile con la minor spesa possibile e che:

- Includesse un sistema di aggancio/sgancio bidone di facile controllo
- Garantisse la possibilità di portare il bidone in una posizione consona allo svuotamento (garantire il ribaltamento del bidone e l'apertura del coperchio)
- Sviluppasse abbastanza forza da riuscire a sollevare gli organi di carico più il peso del bidone e del suo contenuto.

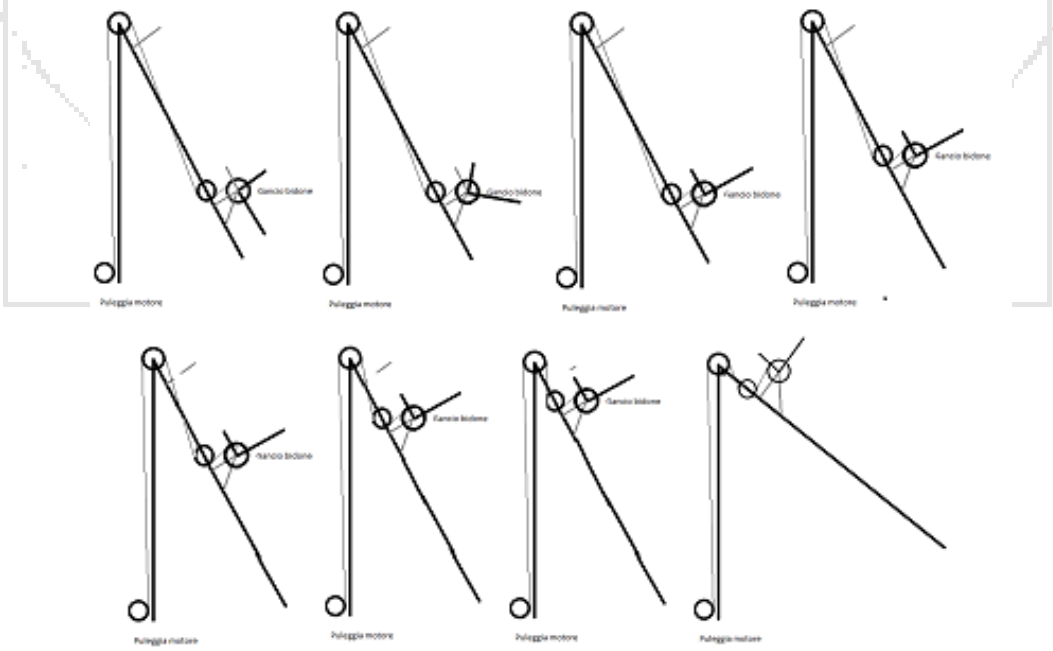


Fig.7

La prima ipotesi è stata quella di figura 7. Il sistema era costituito da un insieme di pulegge e due slitte. Il tutto era azionato da un sistema di finecorsa meccanici. L'azionamento era delegato ad un unico motore. Tale ipotesi ha però evidenziato i seguenti problemi:

- Il motore doveva applicare una coppia molto elevata per azionare gli organi di carico
- Non era possibile riportare il bidone nell'apposito stallo
- La struttura presentava un sistema di leve poco vantaggioso

Si è optato allora per un altra soluzione con un braccio rigido collegato direttamente all'albero di un motoriduttore in corrente continua. Per riuscire a svuotare il bidone all'interno del cassone, la parte terminale del braccio è stata inserita una pinza con sistema di brandeggio. Di questa soluzione è stato realizzato un prototipo che soddisfacesse tutti i requisiti ipotizzati a livello teorico. Riportiamo di seguito il disegno meccanico:

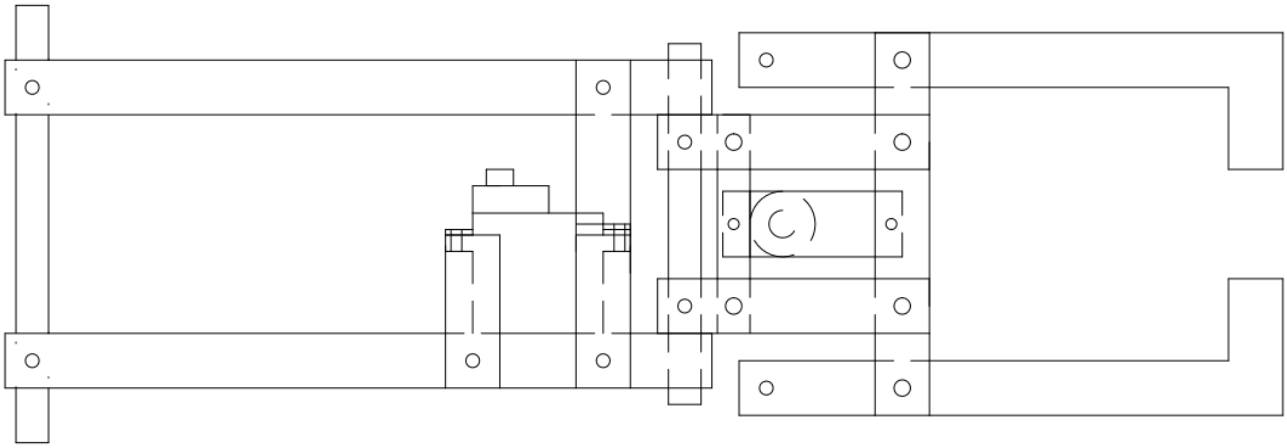


Fig. 8

Nella parte sottostante la pinza è stata fissata una piastra di alluminio necessaria per il supporto della scheda per il sistema di riconoscimentodei bidoni:

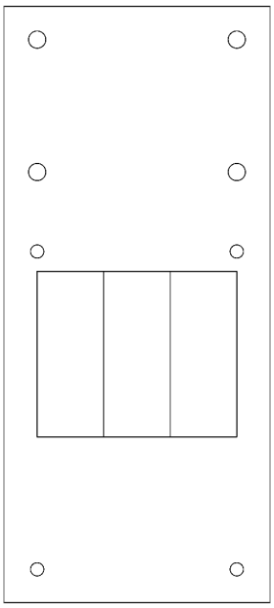


Fig. 9

L'elettronica che gestisce tutto il braccio è divisa in due parti: la prima si occupa della gestione meccanica delle varie parti che costituiscono il braccio mentre la seconda si occupa del riconoscimento del colore del bidone.

Gestione meccanica pinze, brandeggio e motore braccio:

Questo blocco è composto da un motore in continua**,** il cui albero è solidale ad un riduttore meccanico di giri**,** e da due servomotori. Il motore in continua serve per il sollevamento del braccio. Poiché questo motore deve sollevarepesidi elevata entità, necessita di un'elevata coppia. Quest'ultima gli è fornita dal blocco di riduzione che diminuendo il numero di giri consente al motore di avere una coppia maggiore.

Questo motore è controllato tramite il secondo ponte ad H contenuto nell'integrato L298 di cui si è fatto uso nel blocco del motore di avanzamento. La posizione del braccio (riposo: alto, lavoro: basso) è rilevata tramite due micron finecorsa direttamente interfacciati col microcontrollore, il quale realizza automaticamente un pull-up interno, impostato via software.

Il brandeggio e la pinza sono invece comandati da due servomotori. La gestione di questi due motori è realizzata tramite software.

Schema:

Per lo schema si faccia riferimento alla **Fig. 3.**

Funzionamento:

Durante la marcia il braccio e brandeggio saranno alzati e le pinze chiuse. Questa posizione è rilevata da un micron finecorsa il quale, grazie ad un pull up interno ad Arduino, manderà un segnale basso quando il braccio si trova a finecorsa mentre un segnale alto (5V) quando si trova in un'altra posizione. Il micron di finecorsa inferiore funziona allo stesso modo. Quando il veicolo arriva alla postazione dei bidoni si ferma, apre le pinze e abbassa il braccio(lasciando attivo il motorefinché non arriva un segnale basso dal micron inferiore) e il brandeggio. Il motore del braccio viene gestito allo stesso modo di quello di avanzamento. Si attiva poi il sensore di riconoscimento del colore e se il bidone è quello corretto si chiudono le pinze. A questo punto si inizia il sollevamento del bidone. Si aziona il motore del braccio fino a che il micron superiore non da un segnale basso. Si alza il brandeggio e questo permette al bidone di rilasciare il suo contenuto all'interno del cassone del veicolo. Bisogna poi abbassare il brandeggio e il braccio e quando quest'ultimo arriva a finecorsa aprire le pinze per rilasciare il bidone. Eseguiti questi passaggi si riporta il braccio in posizione iniziale e si prosegue fino alla stazione successiva.

Nel caso in cui il primo bidone analizzato non fosse quello desiderato occorre alzare il braccio e passare a quello successivo. Se questo bidone viene riconosciuto il veicolo lo caricherà altrimenti passerà al bidone successivo.

Collaudo:

Dopo aver realizzato il braccio, averlo montato sul veicolo e realizzato i collegamenti con la scheda della gestione dei motori e con Arduino, si passa al collaudo. Come primo collaudo si è realizzato un software che aziona il motore finché non rileva un segnale basso dal micron superiore o da quello inferiore. Una volta appurato che questo movimento veniva eseguito correttamente si è realizzato un altro software per verificare le posizioni estreme dei servomotori di pinze e brandeggio. Dopo aver memorizzato gli angoli corrispondenti alle posizioni estreme dei servomotori si è passato alla prova di sollevamento di un bidone.

Riconoscimento colore bidone:

Questo blocco permette di riconoscere il colore del bidone analizzato. Per la discriminazione dei bidoni si utilizza un circuito di riconoscimento del colore realizzato mediante 3 diodi LED ad alta intensità (red, green, blue) e da 3 fototransistor.

Schema:

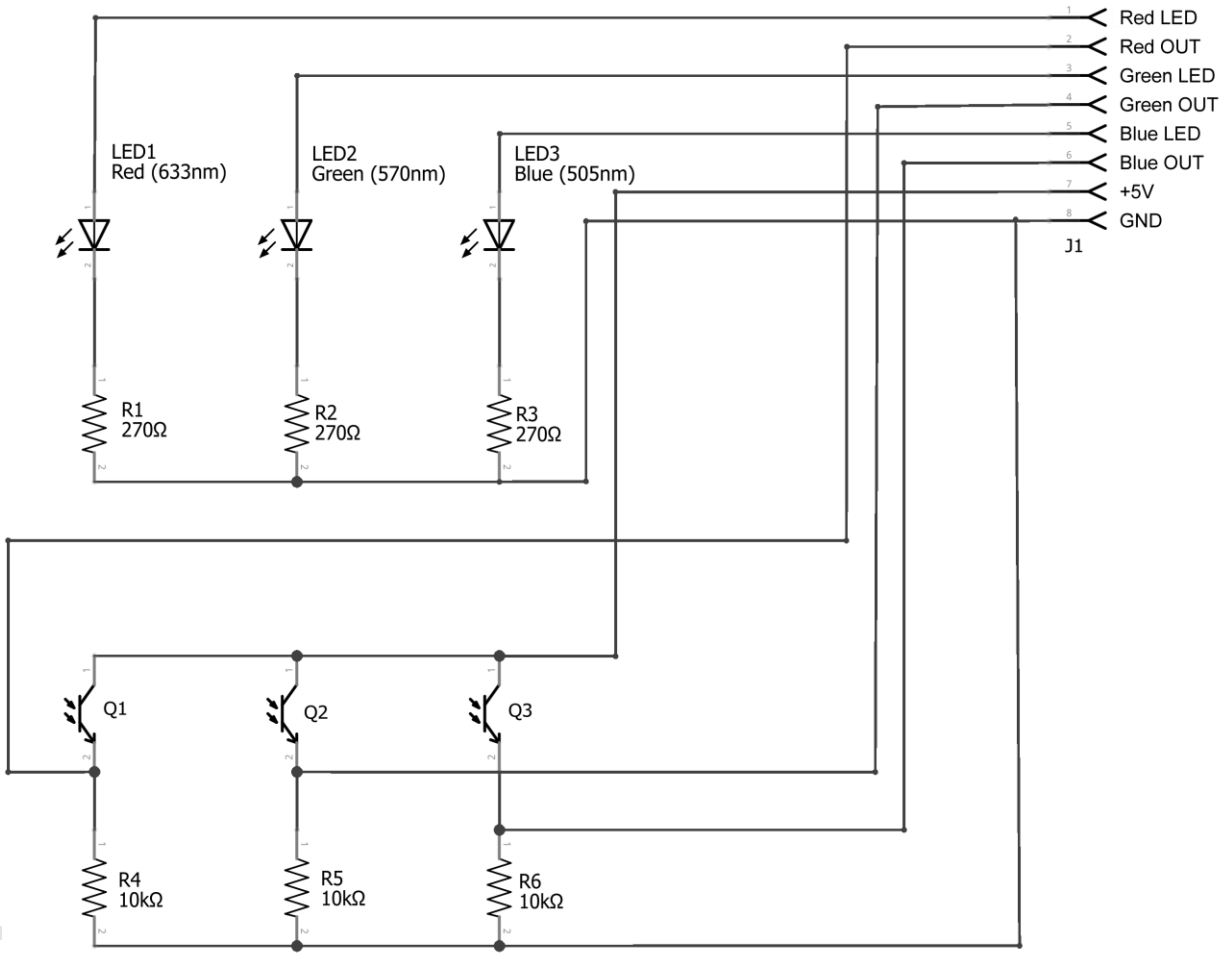


Fig. 10

Funzionamento:
I fototransistor e i LED sono disposti in file parallele a coppie. A ogni LED corrisponde un fototransistor che rileverà la luce riflessa dal bidone. La scheda è opportunamente schermata da luce esterna. I LED vengono accesi uno alla volta da Arduino. Dopo aver acceso il LED si passa alla lettura del valore di luminosità riflessa. Lalettura viene fatta attraverso un partitore di tensione fra una resistenza e il fototransistor. Quest'ultimo varia la sua conducibilità elettrica proporzionalmente all'intensità luminosa che lo colpisce e**,** di conseguenza**,** la tensione ai suoi capi è proporzionale all'intensità luminosa. La letturaavviene avviene grazie agli input analogici di Arduino che, come nel caso delle fotoresistenze, andrà a convertire il valore della tensione d'ingresso in un valore che va da 0 a 1023 trami te il convertitore A/D integrato. Dopo aver effettuato la prima lettura si spegne il primo LED e si ripetono le stesse operazioni per gli altri due colori.
Il sistema di discriminazione del colore del bidone è effettuatotramite il modello coloriRGB. Tramite software si analizzano i valori corrispondenti alla luce riflessa da ogni colore e grazie al confronto incrociato dei 3 valori si è in grado di discriminare un colore con precisione.

Collaudo:
Per prima cosa si è verificato che la conducibilità dei fototransostor variasse in modo significativo quando sottoposti alla luce riflessa dei vari colori. Appurato ciò si è passati alla realizzazione del circuito stampato.
Per il collaudo di questa scheda abbiamo realizzato due software di prova. Il primo permetteva la lettura di diversi valori per ogni colore. Grazie a questo software si è riusciti a memorizzare le soglie di colore utilizzate poi nel software definitivo per la discriminazione. Il secondo software realizzato è stato utilizzato come collaudo per il riconoscimento dei colori. Esso esegue le letture dei 3 colori e stampa via seriale il colore riconosciuto. Dopo aver appurato il funzionamento di questo software lo abbiamo integrato nel software principale.

3.2.6 Controllo carica batteria

Questo blocco rileva il livello di carica della batteria tramite un partitore resistivo.

Schema:

Per lo schema si faccia riferimento alla **Fig. 5**.

Funzionamento:

La tensione ai capi del partitore è quella della batteria (12V). Le resistenze del partitore sono state dimensionate in modo che non assorbissero troppa corrente, ma che quest'ultima fosse sufficiente percausare una caduta di tensione sulle resistenze apprezzabile dal microcontrollore. Inoltre poiché il microcontrolloreha una tensione di riferimento pari a 5V (il range dei pin analogici di ingresso è limitato da 0 a 5V) si è dovuto dimensionare le resistenze in modo che sull'uscita del partitore si avessero al massimo 5V in caso di batteria completamente carica.Il microcontrollore esegue una lettura della tensione d'uscita e poiché il valore di questa lettura va da 0 a 1023 lo converte in uno che va da 0 a 100 e lo invia al PC se si è scostato dal valore rilevato in precedenza.

Collaudo:

Si è realizzato il partitore su una breadboard e lo si è alimentato con una tensione massima di 13V proveniente da un generatore di tensione in modo da simulare il valore di tensione presente sui poli della batteria quando essa si trova al massimo valore di carica. Si è poi verificato che l'uscita del partitore variasse in modo significativo alle variazioni della tensione di alimentazione. Una volta appurato ciò si è passati alla realizzazione del circuito stampato.

3.2.7 Modulo wi-fi

Le comunicazioni a distanza necessarie al telecontrollo del veicolo sono state realizzate tramite un modulo USART(Universal Synchronous-AsynchronousReceiver/Transmitter) to Wi-Fi Converter, ovvero un dispositivo che si interfaccia tramite porta seriale con Arduino e permette una conversione dei dati ricevuti tramite socket via wifi in byte da inviaretramite interfaccia seriale e, viceversa, di inviare via socket ad un hosti dati ricevuti sull'interfaccia seriale, attraversola tecnologia Wi-Fi. Il dispositivo da noi impiegato è l'USR-WIFI232-B.

Per la configurazione il dispositivo mette a disposizione un'interfaccia web raggiungibile tramite l'indirizzo IP ad esso assegnato. Con tale interfaccia è possibile settare:

a. La modalità operativa:
    - *Access Point Mode*: il dispositivo genera una rete Wi-Fi a cui qualunque host dotato di interfaccia Wi-Fi può collegarsi.
    - *Station Mode*: il dispositivo verrà configurato per connettersi ad una rete Wi-Fi già esistente, ovvero ad un accesspoint esterno. Grazie a questa configurazione potrà essere raggiunto da ciascun host collegato alla rete di appartenenza.
b. Impostazioni per la modalitàaccesspoint:
    - Modalità Wi-Fi: tipo b, g, n o mixed mode
    - Impostazione dell'SSID
    - Impostazione indirizzo IP del dispositivo
    - Impostazione SubnetMask
    - Abilitazione del servizio DHCP
c. Impostazioni per la modalità station mode:
    - SSID della rete a cui connettersi
    - Impostazioni di sicurezza (nel caso la rete a cui ci si deve connettere sia protetta da chiavi WEP o WPA)
    - Uso diindirizzo IP dinamico o statico
    - Impostazione dell'host name

d. Impostazioni di comunicazione seriale e di conversione WiFi - Seriale:
   - Baudrate(velocità di modulazione
   - Numero di bit trasmessi (default 8)
   - Abilitazione del bit di parità
   - Numero di bit di stop
   - Ruolo del dispositivo: client o server
   - Protocollo di trasporto impiegato: TCP o UDP
   - Porta di livello applicazione(default 8899)
   - Indirizzo IPhost di destinazione a cui inviare le richieste (in caso di funzionamento client)

Schema:

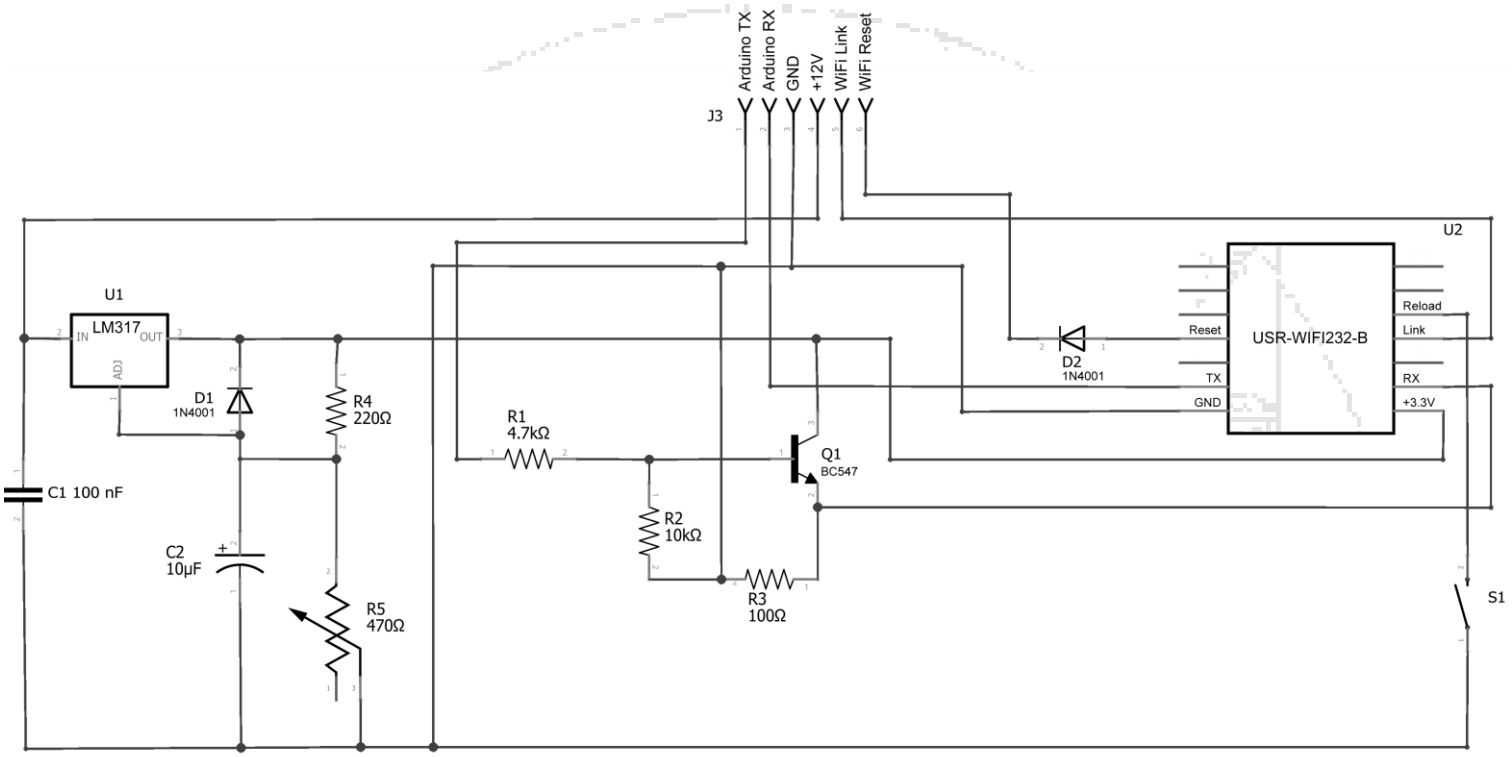


Fig. 11

Funzionamento:

Il dispositivo Wi-Fi è alimentato con una tensione continua a 3,3V. Per ottenere questa tensione da quella della batteria si è utilizzato l'integrato LM317 che è uno stabilizzatore di tensione. Tramite un potenziometro è possibile tarare il valore di tensione che si desidera ottenere in uscita.

Poiché alimentatoa 3,3V, il modulo Wi-Fi comunicherà con il microcontrollore con un segnale binario il cui valore massimo corrisponde alla tensione di alimentazione. La porta seriale di Arduino funziona però con tecnologia TTL: l'uno logico corrispondequindi ad una tensione di 5V. Il modulo Wi-Fi utilizzato però non può ricevere questa tensione.Per ovviarea questo problema si è realizzato un circuito per il pilotaggio di un transistor a cui è delegato il compito di convertire i due diversi livelli di tensione: 5V con 3,3V. Il collettore è collegato ai 3,3V mentre l'emettitore è collegato al ricevitore del dispositivo Wi-Fi. Sulla base del transistor è presente il segnale proveniente dalla porta seriale di Arduino il quale andrà a polarizzareil transistor. Per il reset, ovvero il reboot del sistema operativo Linux(installatosul modulo WiFi), è necessario fornire uno 0 logico sul pin di reset del dispositivo per almeno 300ms. Per far ciò è stato predisposto un diodo sul pin di reset del dispositivo Wi-Fi. Quando sul catodo è presente un valore alto (proveniente da Arduino) il diodo è interdetto eil pin di reset è flottante e si trova a livello logico alto, poiché in logica TTL, quindi il dispositivo si trova in condizione di normale funzionamento. Quando invece sul catodo è presente un valore basso il diodo entra in conduzioneportando il pin di reset del modulo WiFi a massa, e viene dato il reset. Il reset è necessario nel caso il dispositivo non rispondesse ai comandi correttamente.

Nel caso si volesse cambiare modalità operativa e il dispositivo non fosse raggiungibile tramite Wi-Fi si è predisposto uno switch per il reload, cioè per riportare il dispositivo alla sua configurazione di default. Per questo comando è necessario mantenere un valore basso per almeno 1s.

Collaudo:
Il collaudo della scheda è stato effettuato per testare il corretto funzionamento della circuiteria di interfaccia tra il modulo Wi-Fi, compatibile con logica TTL a 3.3V, e Arduino compatibile con logica TTL a 5V, dopo aver dimensionato gli opportuni componenti.
Per tale test si è fatto uso di un software appositamente realizzato per inviare dati su un socket TCP/IPevisualizzare a monitor la relativa risposta. Su Arduino è stato caricato invece un software tale che ad ogni byte ricevuto sulla porta seriale lo ripeteva identico in trasmissione.
Una volta verificata la compatibilità tra modulo Wi-Fi e Arduino abbiamo realizzato il circuito di alimentazione a 3.3V per il modulo USR-WiFi tramite uno stabilizzatore di tensione regolabile. Attraverso un potenziometro abbiamo provveduto alla calibrazione della tensione d'uscita in modo da stabilizzarla a 3.3V.

4. DESCRIZIONE SOFTWARE

Vi sono due principali blocchi software uno per la programmazione del microcontrollore e uno per la realizzazione dell'interfaccia di controllo remoto.

4.1 PROGRAMMAZIONE DEL MICROCONTROLLORE

Per quanto riguarda la programmazione di Arduino si possono distinguere diverse parti legate ai blocchi hardware già analizzati nei paragrafi precedenti.
La programmazione del microcontrollore è stata effettuata tramite l'IDE fornito dalla casa costruttrice che prevede l'utilizzo del linguaggio Wiring. Esso è basato sul linguaggio C a cui sono aggiunte alcune strutture tipiche del C++ e alcune istruzioni specifiche legate alle funzionalità del microcontrollore stesso.
Nel programma del microcontrollore si distinguono innanzitutto 3 macro blocchi: la fase di **dichiarazione** ed inizializzazione delle variabili e istanza delle classi, la fase di **setup** (istruzioni eseguite una volta per tutte nella fase iniziale) e la fase di **loop** (istruzioni eseguite ciclicamente all'infinito).

4.1.1 Dichiarazione e inizializzazione

In questa fasevengono dichiarate e inizializzate:

- Costanti identificative dei pin: consentono di fare riferimento ad un pin all'interno del programma tramite un unico identificatore. Nel caso in cui si dovessero apportare modifiche al cablaggio sarà sufficiente aggiornare il valore della costante.
- Costanti relative ad alcuni parametri di configurazione.
- Variabili globali utilizzate per l'interscambio di dati tra le diverse funzioni.
- Classi utilizzate per la gestione dei servomotori.

4.1.2 Setup

Nella fase di **setup** si eseguono le seguenti operazioni:

- Inizializzazione delle interfacce seriali.
  Medianteil metodo `begin(baudrate)` delle classi `Serial` e `Serial1` sono state inizializzate le due interfacce seriali al baudrate specificato in argomento. La classe `Serial`controlla l'interfaccia seriale che permette la comunicazione tramite modulo USB con il PC per scopi di debug. La classe `Serial1` controlla invece l'interfaccia di comunicazione con il modulo Wi-Fi.
- Configurazione dei pin in input o output.
  Tale operazione è eseguita tramite la funzione `pinMode(pin, mode)` specificando come parametri in argomento il numero del pin digitale da configurare e la modalità, ovvero, `INPUT`, `OUTPUT` o `INPUT_PULLUP` (in quest'ultima modalità sul pin viene realizzato un circuito di pull-up interno al microcontrollore. In tal modo quando il pin è flottante il valore di tensione su di essoè imposto a 5V).
- Configurazione delle classi per il pilotaggio dei servomotori.
  In fase di dichiarazione sono state create 3 istanze alla classe `Servo` messa a disposizione dall'ambiente di sviluppo nella libreria `Servo.h`.
  Le 3 istanze create sono:
  - sterzo: preposta al controllo del servomotore per l'azionamento degli organi di sterzo
  - brandeggio: preposta al controllo del servomotore per il brandeggio della pinza d'aggancio dei bidoni
  - pinze: preposta al controllo del servomotore d'azionamento dei bracci della pinza

  In fase di setup si è invece proceduto alla configurazione iniziale dei parametri privati di ciascuna classe. In particolare si è andati a creare una corrispondenza logica tra il pin a cui è collegato un servomotore e la rispettiva classe di pilotaggio. Ciò è reso possibile dal metodo `attach(pin)` della classe `Servo`, specificando in argomento il numero del pin cui è collegato il servomotore da controllare.
- Raddrizzamento dello sterzo.
  Si porta l'albero del servomotore ad un angolo pari a 90 gradi (ovvero la metà dell'angolo massimo di rotazione). Questo è realizzato tramite il metodo `write(angolo)` della classe Servo, specificando come argomento l'angolo (in gradi sessagesimali) a cui si desidera che l'albero del servomotore si posizioni, ovvero 90 gradi.

- Posizionamento in punto di riposo degli organi meccanici del braccio.
  È realizzato richiamando la funzione `voidalza_braccio()` (che si analizzerà al paragrafo 4.1.8) che si occupa di: alzare il braccio e il brandeggio. Successivamente si chiudono le pinze agendo sull'apposito servomotore tramite il metodo `write(angolo)` della libreria `Servo`, specificando questa volta un angolo di 80 gradi.
- Spegnimento dei dispositivi luminosi e acustici.
  L'operazione è eseguita tramite l'istruzione `digitalWrite(pin, value)` specificando in argomento il numero del pin digitale su cui agire e il valore logico a cui andrà settato, in questo caso `LOW`.
- Calibrazione delle soglie per il rilevamento della pista.
  La routine di calibrazione è delegata alla funzione `voidcalcola_soglia()` la quale dopo aver determinato tramite appositi calcoli statistici le soglie di rilevamento di bianco e di nero, provvederà a salvarne i risultati nelle apposite variabili globali`soglia_bianco` e `soglia_nero`. Tale valore rappresentano rispettivamente i valori, digitalizzati, di caduta di tensione che si hannorispettivamente sulle fotoresistenze esterne, quando si trovano sul pavimento bianco, e su quelle interne, quando si trovano sulla pista nera.
  I risultati vengono ottenuti calcolando la media dei valori di 10 diverse letture a cui si sottrae un valore di tolleranza che permette di evitare interferenze dovute a fonti di luce esterne o a lievi variazioni della tonalità di colore della pista o del pavimento.

4.1.3Loop

Terminato il setup si entra nella fase di **loop**che è formata da un gruppo di istruzioni che vengono eseguite in modo ciclico fino a che il microcontrollore non viene disalimentato (o resettato).
Dopo aver verificato che il modulo Wi-Fi sia connesso il microcontrollore richiama la funzione `voidpolling_wifi()` (paragrafo 4.1.13) che si occupa di gestire le richieste che arrivano dal PC remoto (che sono state inviate sulla seriale del microcontrollore tramite il modulo Wi-Fi).Dopodiché si stabilisce se ci si trova in modalità manuale o automatica. Nel primo caso si resta in attesa di comandi dall'utente. Nel secondo si esegue un test per verificare che il sensore ad ultrasuoni non stia rilevando ostacoli e che sia stato ricevutodal PC remoto il comando di start. Se il test risulta positivo si richiama la funzione `void gestione_avanzamento()` che si occuperà di gestire la fase di lavoro. In caso contrario si ferma il motore d'avanzamento del veicolo (o ci si assicura che sia fermo),richiamando la funzione `void pwm_marcia(0)`, e si controlla per quale motivo (ostacolo o comando remoto) il veicolo si sia fermato in modo da comunicarlo al PC remoto.
Come ultima cosa si aggiorna una variabile, precedentemente creata (`time`), a cui si assegna il valore restituito dall'istruzione `millis()`, ovvero il tempo trascorso dall'accensione del microcontrollore (espresso in millisecondi). Ad ogni loop viene confrontato che siano trascorsi 100ms. In caso positivo viene richiamata la funzione `void ultrasuoni()` e viene aggiornato il valore della variabile `time`.
Nel caso in cui invece si rilevi che la connessione Wi-Fi non è attiva il veicolo si ferma in attesa che il collegamento sia ripristinato.

4.1.4 Gestione avanzamento

Per la gestione della fase di lavoro è stata creata un'apposita funzione`voidgestione_avanzamento()` la quale si occupa di:

- leggere e memorizzare i valori digitalizzati relativi alle tensioni prodotte dalle fotoresistenze tramite la funzione `voidlettura_foto()` che si occupa di salvare in 4 variabili globali i valori digitalizzati relativi alle cadute sulle 4 fotoresistenze calcolati a meno di un valore di isteresi presente tra le coppie di fotoresistenze interne ed esterne;
- verificare lo scostamento dei valori letti e in base a questo:
  - decidere se è necessario sterzare, verificare la direzione di sterzo, calcolare l'angolo di sterzo
  - decidere se procedere diritto
  - verificare la presenza del segnale di presenza di stallo dei bidoni (striscia bianca perpendicolare)

Per la lettura digitalizzata delle tensioni analogiche prodotte dalle fotoresistenze si è fatto uso della funzione `intanalogRead(`*`analogue_pin`*`)` che rientra nell'instruction-set messo a disposizione dall'IDE di

Arduino. Tale funzione restituisce un numero intero variabile da 0 a 1023 proporzionale alla variazione da 0 a 5V della tensione analogica sul pin analogico specificato in argomento.
La valutazione dello scostamento dei valori rilevati è eseguita con un test condizionale multiplo:

- Se entrambi i valori rilevati dalle fotoresistenze interne sono inferiori alla soglia del colore nero significa che entrambe si trovano a leggere il colore bianco della pavimentazione. Ciò significa che si è raggiunta una zona di raccolta. Si arresta quindi l'avanzamento del veicolo e si procede con il processo d'individuazione del bidone e una volta individuato all'aggancio e sollevamento, svuotamento e rilascio ed, infine, alla ripresa della fase di avanzamento.
- Se la differenza in valore assoluto tra i valori rilevati dalle fotoresistenze esterne è superioread un valore di tolleranza massimo (che tiene conto dei difetti di fabbricazione delle fotoresistenze) si va a valutare quale delle due fotoresistenze è sotto l'influenza del colore nero della pista e si provvede a correggere lo sterzo per ritornare sulla traccia. In particolare se il valore letto sulla fotoresistenza di sinistra è superiore a quello letto su quella di destra ad ogni loop si diminuirà il grado di curvatura dello sterzo di 10 gradi. Questo provoca una progressiva sterzatura verso sinistra. Viceversa se il valore letto sulla fotoresistenza di destra è maggiore di quello letto sull'altra si andrà ad aumentare di 10 gradi,ad ogni loop, l'angolo di curvatura dello sterzo. Ciò provoca una progressiva sterzatura verso destra.
  In entrambi i casi quando l'angolo di curvatura supera un certo angolo di ampiezza predefinita (`sterzo_max`) si provvede ad aumentare sensibilmente la velocità del motore di avanzamento per contrastare l'attrito dovuto alla posizione di sterzatura delle ruote anteriori. La velocità sarà riportata al suo normale valore non appena l'angolo di curvatura sarà ritornato al di sotto di`sterzo_max`.
  Per quanto riguarda il pilotaggio del servomotore preposto all'azionamento degli organi di sterzo si è fatto uso del metodo `write(angolo)` messo a disposizione dalla classe `Servo` di cui si è istanziata una copia, denominata `sterzo`, in fase di setup. Come argomento si specifica l'angolo in gradi a cui si vuole portare l'albero del servomotore.
- Se, infine,lo scostamento tra i valori rilevati tramite le fotoresistenze esterneè inferiore al massimo valore di tolleranza significa che si deve procedere diritto, infatti entrambe le fotoresistenze stanno rilevando colore bianco.
  Si procede perciò ad assicurarsi che lo sterzo si trovi in posizione diritta (angolo di sterzatura pari a 90 gradi), si setta il motore con direzione di rotazione tale che permetta al veicolo di avanzare in avanti e ci si assicura che la velocità di avanzamento sia al suo normale valore (consono alla marcia rettilinea).
  La selezione della direzione di marcia è fatta agendo sul valore logico TTL dei pin collegati all'integrato L298 tramite la funzione `digitalWrite(n_pin, valore_logico)` (messa a disposizione dall'IDE) specificando in argomento il numero del pin digitale sul quale agire e il valore logico `LOW` (0V) o `HIGH` (5V) da assegnarvi.
  Per settare la velocità di avanzamento si richiama la funzione `voidpwm_marcia(vel)` appositamente creata di cui si tratterà al successivo paragrafo, passando come argomento un valore da 0 a 255 proporzionale alla velocità.

4.1.5 Gestione raccolta bidoni

Come descritto al paragrafo precedente quando le fotoresistenze interne rilevano colore bianco ci si aspetta che sia presente una stazione di raccolta del bidone. Per verificare con maggior certezza che ci si trovi in tale situazione si attendono tre cicli di lettura e se tutti hanno dato come esito il rilevamento del colore bianco allora si procede alla fase di raccolta.
Per prima cosa si avvisa l'interfaccia di controllo remoto dell'avvenuto rilevamento della stazione e si arresta il veicolo. Dopodiché si procede all'apertura delle pinze e all'abbassamento del braccio.
È perciò ora possibile procedere al riconoscimento del bidone poiché la pinza, sulla quale è fissata la scheda di riconoscimento dei colori (paragrafo 4.1.11), si trova di fronte al bidone stesso. Si richiama perciò la funzione `booleancheck_colore()` la quale restituirà valore vero solo se viene rilevato il colore del bidone corrispondente al ciclo di raccolta in corso. In caso positivo si procede alla chiusura della pinza, si alzano braccio e brandeggio (provocando lo svuotamento del bidone, dopodiché si riabbassa il braccio e si apre la pinza per rilasciare il bidone. In ogni caso si procede poi ad alzare il braccio riportandolo così alla sua normale posizione di riposo.
Si procede quindi a raddrizzare lo sterzo e a riprendere l'avanzamento del veicolo.

4.1.6 Gestione del motore di avanzamento

Per il pilotaggio dell'avanzamento del veicolo si è realizzata un'apposita funzione (`voidpwm_marcia(vel)`) che consente sia di variare la velocità del motore, sia di rilevare se il motore è fermo o in movimento. In tal modo:

- se alla funzione è richiesto di variare la velocità e il motore è già in movimento, si procede semplicemente a variare l'ampiezza dell'impulso PWM sul pin collegato all'enable dell'integrato L298;
- se è richiesto di fermare il motore si imposta il PWM con impulso di ampiezza massima e si settano i pin di pilotaggio dell'L298 in modo da entrare in modalità di arresto rapido (fast motor stop);
- se è richiesto di variare la velocità e il motore è fermo si procede a dare uno spunto mediante impulso PWM di massima ampiezza mantenuto per 200ms, facilitando in tal modo la partenza. Si porta poi il PWM alla velocità richiesta in argomento.

La sintassi della funzione è `pwm_marcia(intpwmdc)`. Come argomento si specifica un numero da 0 a 255 proporzionale alla velocità di avanzamento desiderata. (0 significa arresto rapido).

4.1.7 Gestione sensore a ultrasuoni

Questo blocco si basa sull'utilizzo della funzione `void ultrasuoni()` che si occupa di calcolare la distanza rilevata in cm e scrivere nella variabile booleana `ultrasonic_en`un valore vero quando si rileva un ostacolo ad una distanza maggiore di 25cm, mentre scrive un valore falso se l'ostacolo è in prossimità del veicolo.

Il calcolo della distanza viene realizzato rilevando la durata dell'impulso generato dalla scheda ultrasuoni sul pin di echo dopo che sul pin di trigger è stato generato una transizione di livello logico con fronte di salita. Il valore ottenuto viene poi diviso per una costante (pari circa a 55) che consente di ottenere la distanza in centimetri dall'ostacolo.

4.1.8 Sollevamento braccio

La funzione che si occupa del sollevamento del braccio è stata denominata `voidalza_braccio()`. Esegue i seguenti passaggi:

- Settaggio della direzione di rotazione del motore agendo sui pin di controllo dell'integrato L298.
- Azionamento del motore tramite tecnica PWM. Si è fatto uso della funzione `analogWrite(pin,t_on)` messa a diaposizione dall'IDE stesso. Essa accetta come argomento il parametro relativo al pin su cui effettuare la modulazione PWM e il parametro relativo alla durata dell'impulso del segnale digitale PWM.
- Attesa del raggiungimento del finecorsa alto del braccio verificando l'avvenuta transizione di livello logico del pin cui è collegato il micron relativo al finecorsa superiore. Il test relativo al valore logico del pin è effettuato tramite l'istruzione `digitalRead(pin)` la quale accetta come argomento il numero del pin di cui leggere il valore digitale. Verrà restituito valore `HIGH` o `LOW` a seconda che il pin si trovi a valore logico alto o basso.
- Al termine si solleva il brandeggio andando a portare il relativo servomotore adun angolo pari a 40 gradi. Ciò è effettuato tramite il metodo `write(angolo)` della libreria `Servo`.
- Si richiama la funzione `voidchiudi_pinze()` (descritta nel paragrafo x.x).
- Come ultima cosa si spengono i dispositivi luminosi e acustici.

4.1.9 Abbassamento braccio

La funzione `voidabbassa_braccio()` esegue i seguenti passaggi:

- Attivazione dei dispositivi luminosi e acustici.
- Si abbassa il brandeggio portando l'albero del relativo servomotore ad un angolo di 80 gradi.
- Settaggio della direzione di rotazione del motore preposto alla movimentazione del braccio agendo sui pin di controllo dell'integrato L298.
- Azionamento del motore tramite tecnica PWM.
- Attesa del raggiungimento del finecorsa basso del braccio verificando l'avvenuta transizione di livello logico del pin cui è collegato il micron relativo al finecorsa inferiore.

4.1.10 Gestione organi di brandeggio e pinze

Questi due organi sono gestiti da due servomotori. Per rallentare i movimenti di questi ultimi sono state create 4 funzioni:

- `voidapri_pinze()`
- `voidchiudi_pinze()`
- `voidbrandeggio_su()`
- `voidbranmdeggio_giu()`

Tutte e 4 si occupano di incrementare o decrementare lentamente (mediante l'aggiunta di ritardi via software) l'angolo di posizione dell'albero del servomotore.

4.1.11 Riconoscimento colore

La funzione `booleancheck_colore()` si occupa di leggere il colore del bidone e di verificare se esso è compatibile con il ciclo di raccolta attualmente in corso. In caso positivo restituirà valore `vero` altrimenti `falso`.

Il riconoscimento del colore viene effettuato tramite il seguente algoritmo:

a) Si effettua la lettura di ciascuno dei tre colori del modello RGB andando a rilevare la caduta di tensione sui fototransistor mediante l'istruzione `analogRead(analog_pin)`.
b) Si determina il minimo valore tra i trevalori di colore letti.
c) Si sottrae il minimo a ciascuno di essi consentendo di eliminare così il disturbo dato dalle fonti di illuminazione esterna come, ad esempio, la luce ambientale.
d) Si vanno a confrontare i valori ottenutidi rosso, verde e blu con dei parametri definiti in precedenza.
e) Si determina di quale colore si tratta.
f) Si incrementa all'interno di un array (`intserie_test_colore[4]`) un contatore relativo al colore rilevato.
g) Se nessuno dei colori dei bidoni (blu, giallo o verde) è stato rilevato si incrementa invece il contatore relativo alle letture nulle.
h) Si ripetono tali operazioni per 10 volte dal punto (a).
i) Al termine si ottiene che nell'array `serie_test_colore`si avrà in corrispondenza di ciascun indice (0 → blu, 1 → giallo, 2 → verde, 3 → lettura nulla) il numero di volte in cui è stato rilevato un certo colore, o una lettura nulla.
j) Si calcola il massimo tra i valori di ciascun indice
k) Il dato che è stato rilevato con maggiore frequenza indica il bidone rilevato.

Terminata la routine di riconoscimento del bidone si esegue un confronto per verificare se il bidone rilevato corrisponde al ciclo di raccolta in corso di svolgimento.

4.1.12 Livello batteria

Il monitoraggio del livello di carica della batteria è effettuato mediante la funzione `booleantest_livello_batteria()` la quale verifica se il nuovo valore di carica si discosta da quello precedente. In caso positivo aggiornail valore della variabile globale `percentuale_batteria` e restituisce un valore vero. Altrimenti restituisce semplicemente un valore falso.

All'interno di tale funzione si procede anche all'adeguamento dei valori del duty-cycle impiegati per il controllo con modulazione PWM del motore di avanzamento. Grazie a ciò è possibile mantenere una velocità pressoché costante al variare della corrente erogata dalla batteria.

4.1.13 Routine di comunicazione Wi-Fi

La comunicazione Wi-Fi con il PC remoto viene gestita dalla funzione `voidpolling_wifi()` la quale viene richiamata ad ogni loop e all'interno delle varie routine presenti all'interno del programma. Tale funzione si occupa di rispondere alle richieste iniziate dal software di gestione remota il quale necessita di una risposta con tempi di latenza massimi di 250ms per consentire un monitoraggio dettagliato e in tempo reale del veicolo.

Ogni volta che la funzione viene richiamata viene verificato se nel buffer di ricezione della porta seriale sono presenti dei dati sotto forma di caratteri (byte).

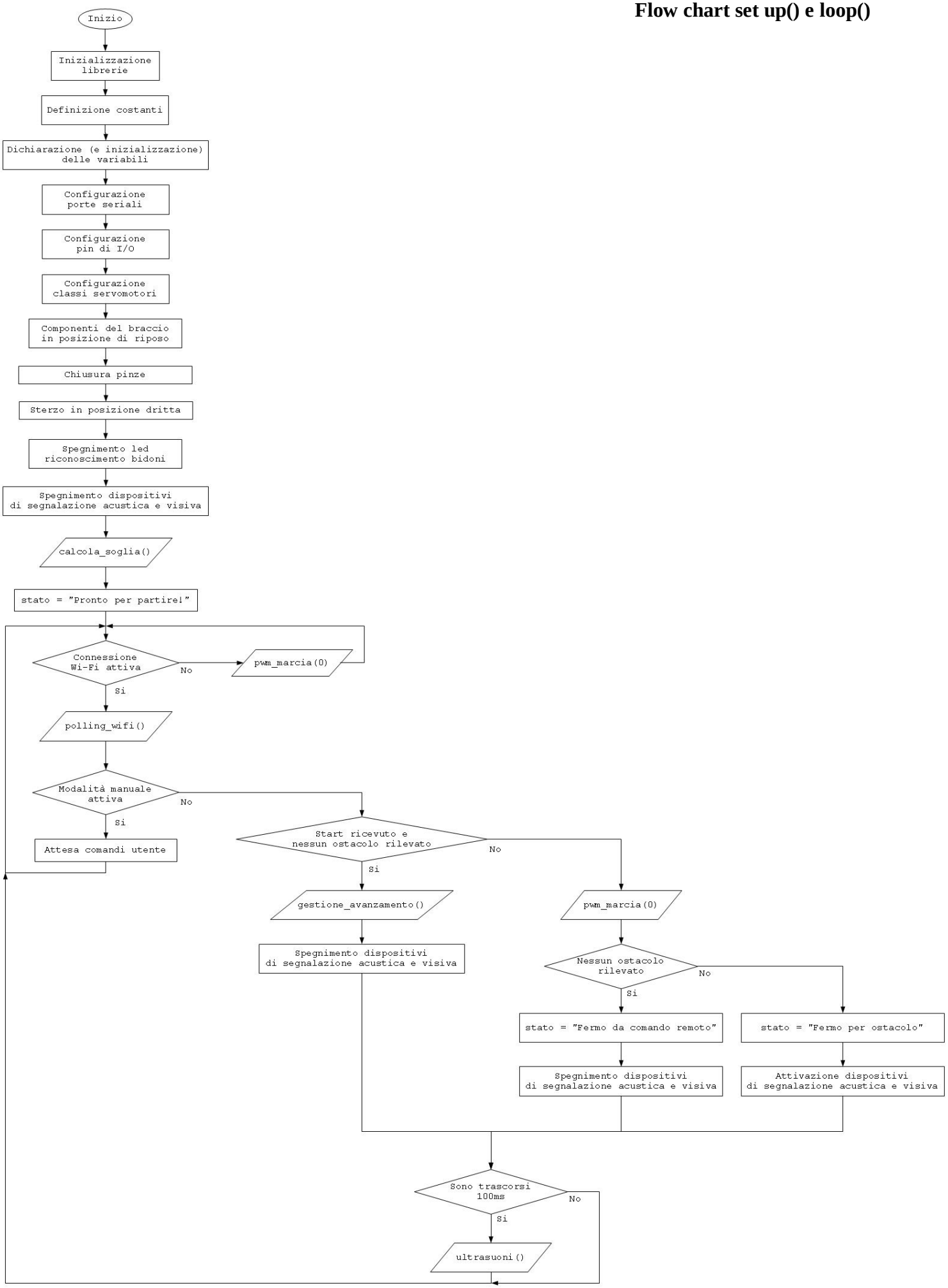
In caso positivo si legge il primo carattere presente nel buffer e tramite una selezione (switch) si esegue il blocco di codice da esso individuato. Ciascun carattere ricevuto rappresenta perciò un diverso comando che il software di gestione remoto richiede di essere eseguito.

Tabella di analisi dei comandi:

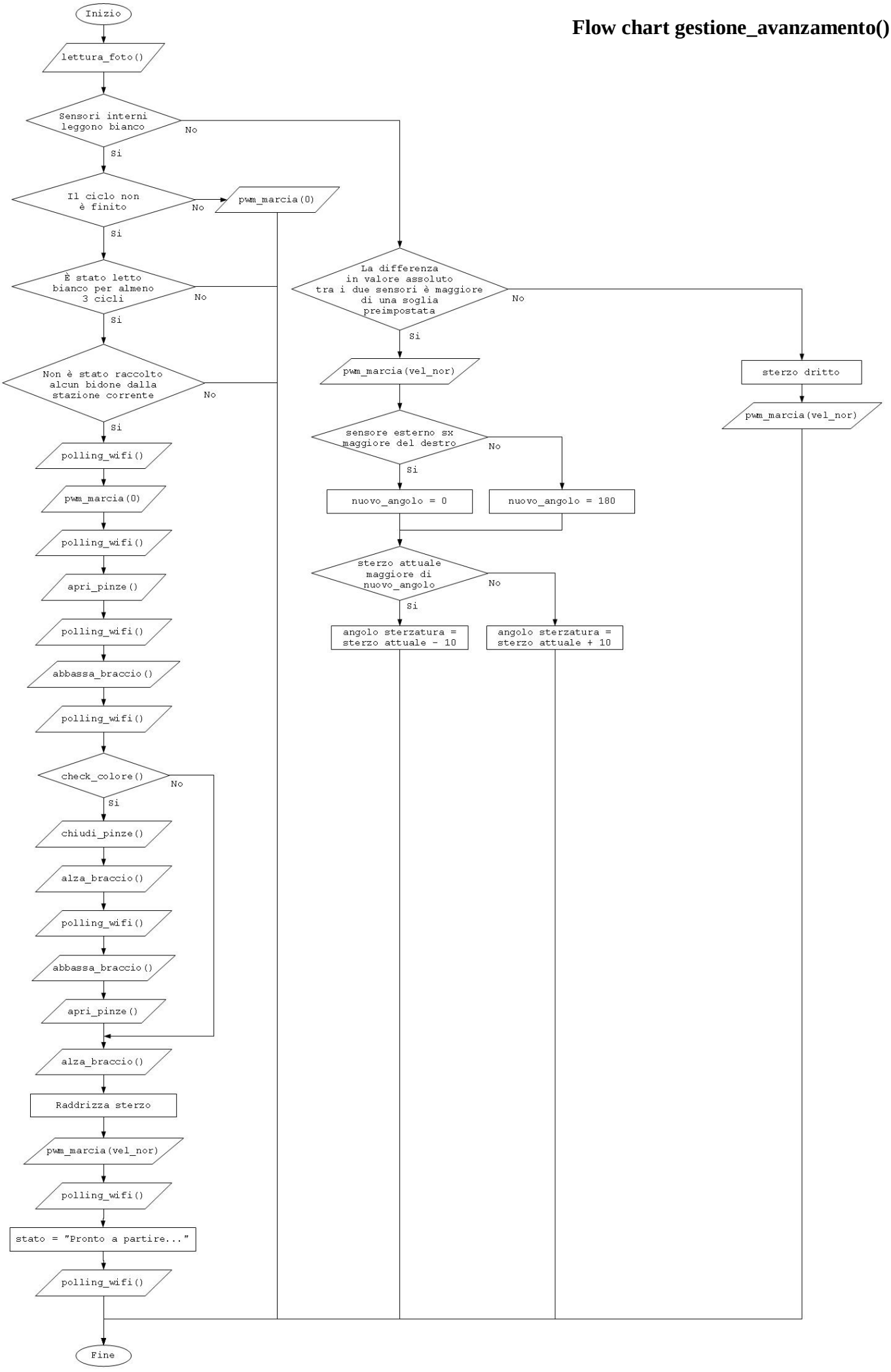
Carattere identificativo	*Descrizione*
`*`	*Il software remoto richiede un aggiornamento di stato del veicolo.* Ogni cinque richieste si verifica se ci sono aggiornamenti riguardanti il livello di carica della batteria richiamando la funzione `booleantest_livello_batteria()` descritta al paragrafo precedente. In caso positivo si invia al PC remoto una stringa contenente la dicitura `"Batteria:P"` dove P è un numero indicante il livello percentuale di carica. In tutti gli altri casi si invia all'host remoto il contenuto della variabile globale `stringstato`, la quale contiene le informazioni riguardanti lo stato corrente del veicolo. Tale variabile viene aggiornata all'interno di ciascuna routine eseguita dal software.
`m`	*Si è ricevuto un comando relativo alla modalità manuale.* Si attendono 30ms per motivi di pulitura del buffer di ricezione e si legge il successivo carattere ricevuto. Tale carattere identifica il comando relativo alla modalità manuale che dovrà essere eseguita: - `1` → la modalità manuale viene attivata - `0` → la modalità manuale viene disattivata - `s` → si è ricevuto un comando relativo al pilotaggio dello sterzo: il successivo byte ricevuto rappresenta un numero proporzionale all'angolo di sterzatura - `a` → si è ricevuto un comando relativo alla velocità di avanzamento: il successivo byte ricevuto rappresenta un numero proporzionale alla velocità - `i` → si è ricevuto un comando relativo alla velocità di retromarcia: il successivo byte ricevuto rappresenta un numero proporzionale alla velocità - `p` → si è ricevuto un comando relativo al pilotaggio della pinza del braccio: se il successivo carattere ricevuto è `a` allora si procede all'apertura della pinza, se il carattere è `c` allora si procede alla chiusura; - `b` → si è ricevuto un comando relativo al sollevamento o abbassamento del braccio: se il successivo carattere ricevuto è `s` si procede ad alzare il braccio, se si riceve invece `g` il braccio viene abbassato.
`b`	*Richiesta di lettura del livello di carica della batteria.* Viene inviato all'host remoto il valore percentuale relativo al livello di carica della batteria.
`s`	*Conferma avvio fase di lavoro.* La variabile globale `ON`viene settata al valore booleano `true`. La fase di raccolta viene avviata.
`f`	*Stop del veicolo.* La variabile globale `ON`viene settata al valore booleano `false`. L'avanzamento del veicolo e l'attuale fase di lavoro vengono arrestati.
`r`	*Reset generale.* Il veicolo viene riavviato. Tutti i dati relativi alla fase di lavoro in corso vanno perduti.
`c`	*Selezione del ciclo di raccolta della carta.* La variabile globale `intbidone_ciclo_attuale`viene settata a 0. Vengono resettate le variabili di controllo del ciclo di raccolta.
`p`	*Selezione del ciclo di raccolta della plastica.* La variabile globale `intbidone_ciclo_attuale`viene settata a 1. Vengono resettate le variabili di controllo del ciclo di raccolta.
`v`	*Selezione del ciclo di raccolta del vetro.* La variabile globale `intbidone_ciclo_attuale`viene settata a 2. Vengono resettate le variabili di controllo del ciclo di raccolta.

4.2 FLOW CHART

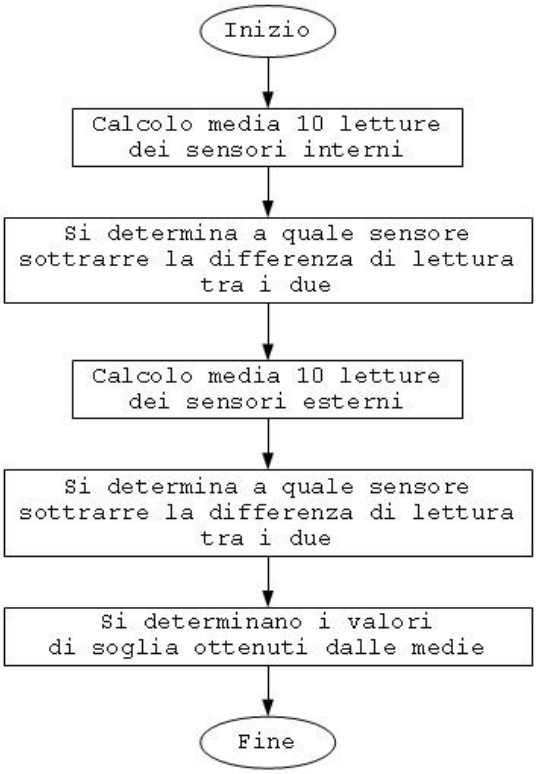
Flow chart set up() e loop()



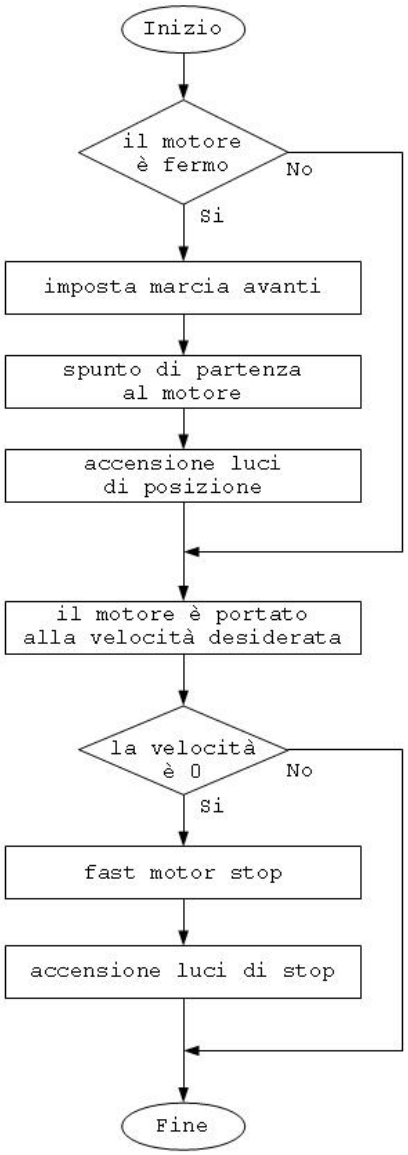
Flow chart gestione_avanzamento()



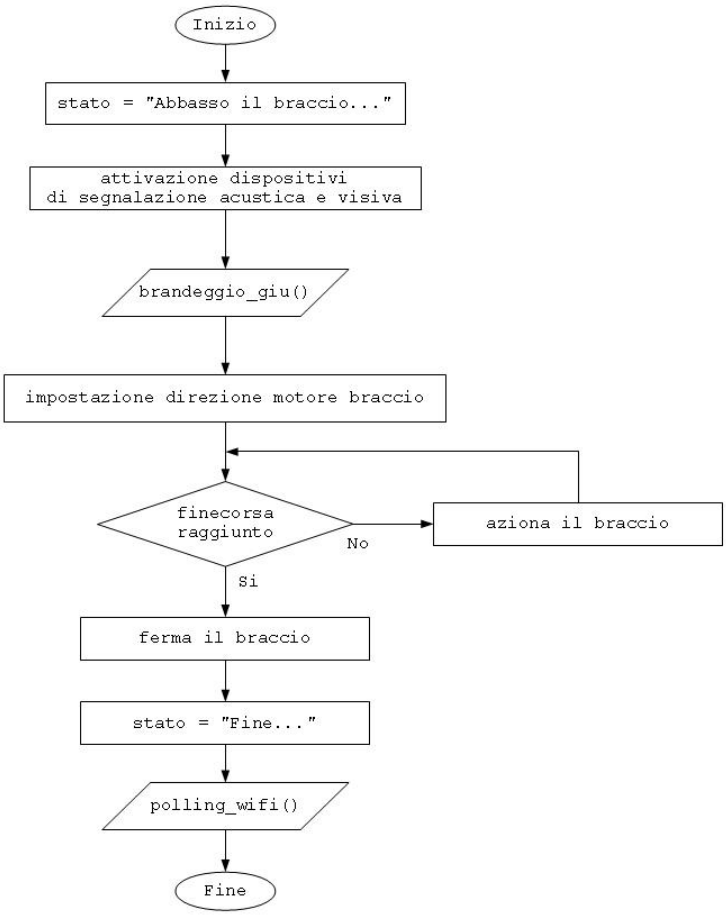
Flow chart calcolo_soglia()



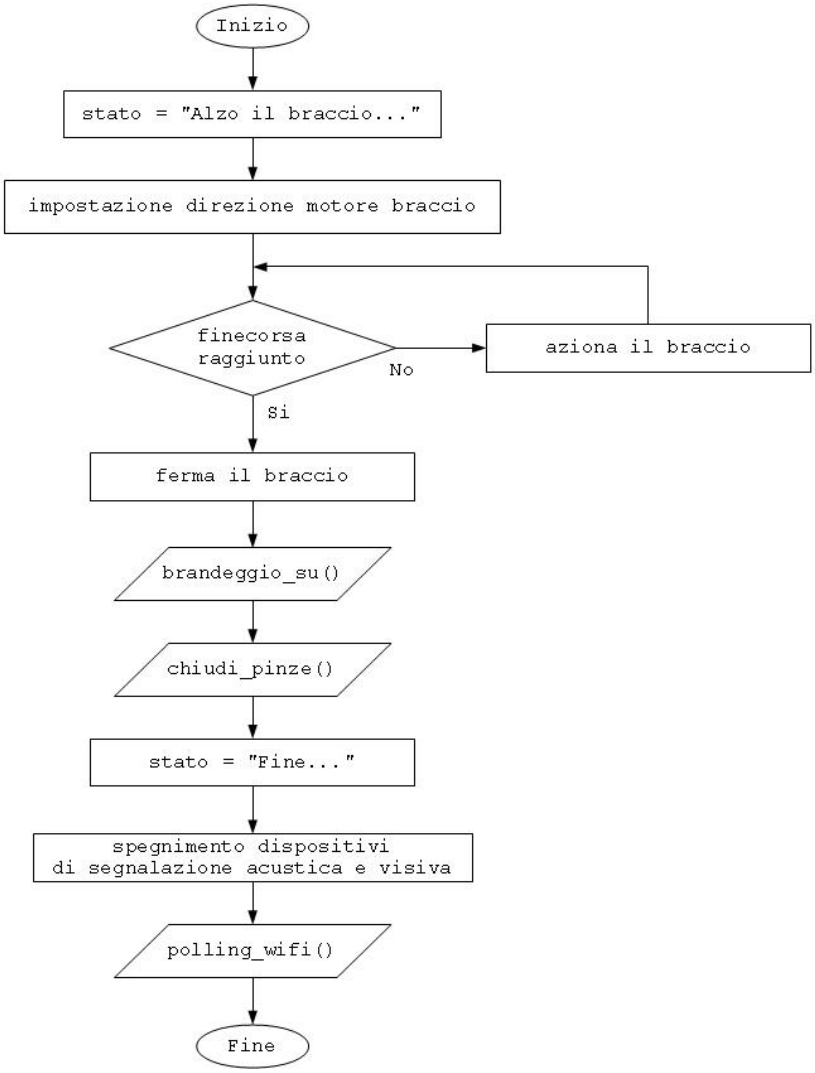
Flow chart pwm_marcia()



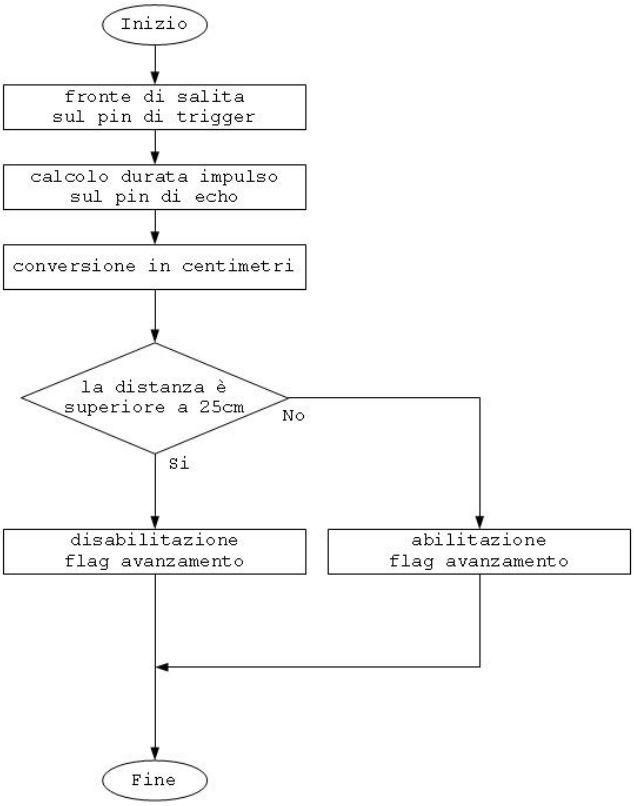
Flow chart abbassa_braccio()



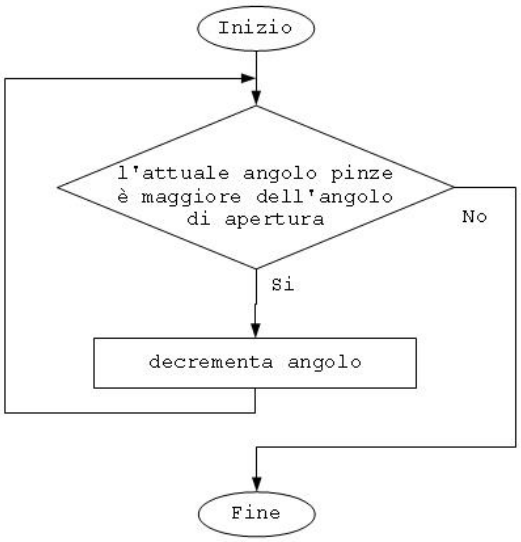
Flow chart alza_braccio()



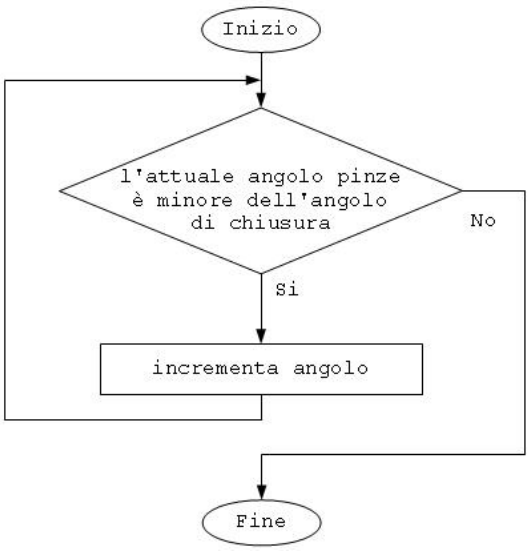
Flow chart ultrasuoni()



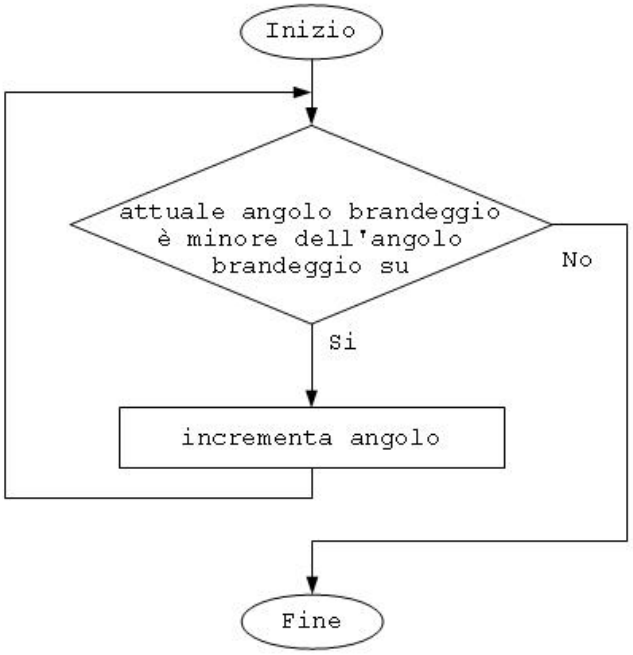
Flow chart apri_pinze()



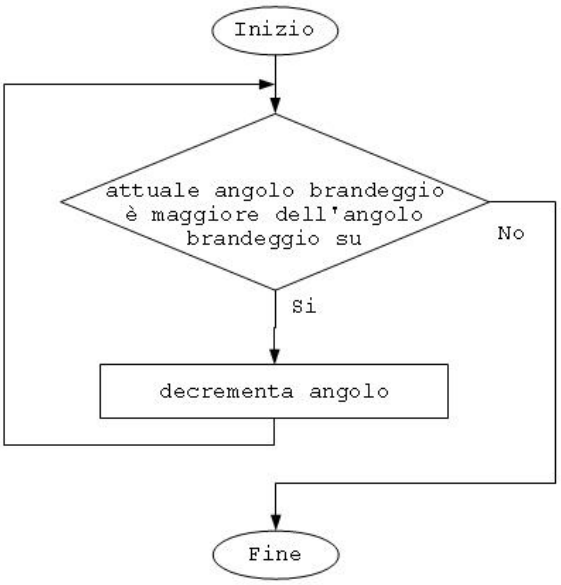
Flow chart chiudi_pinze()



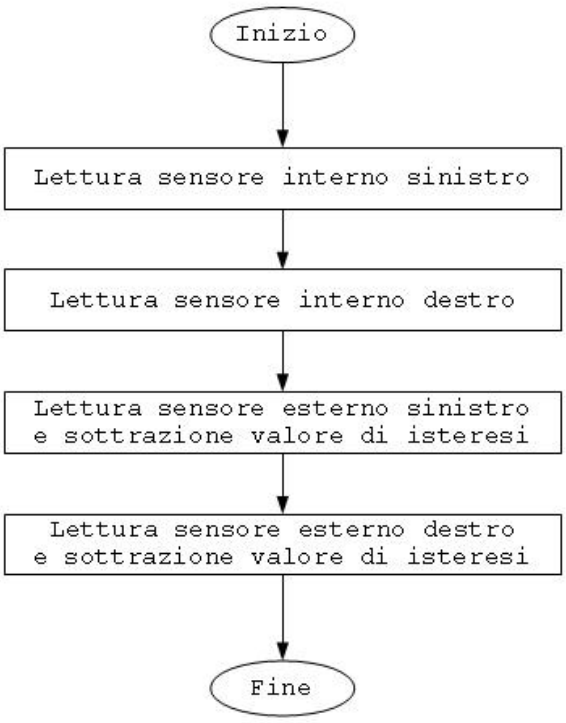
Flow chart brandeggio_giu()



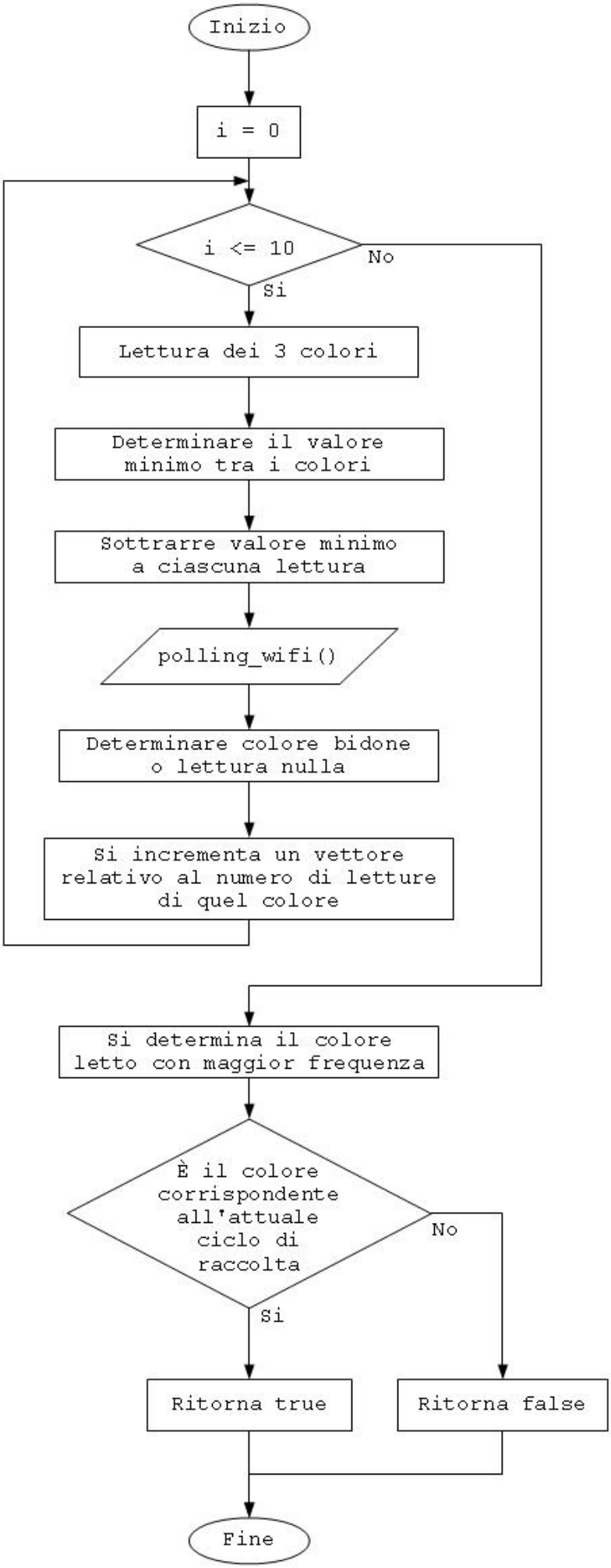
Flow chart brandeggio_su()



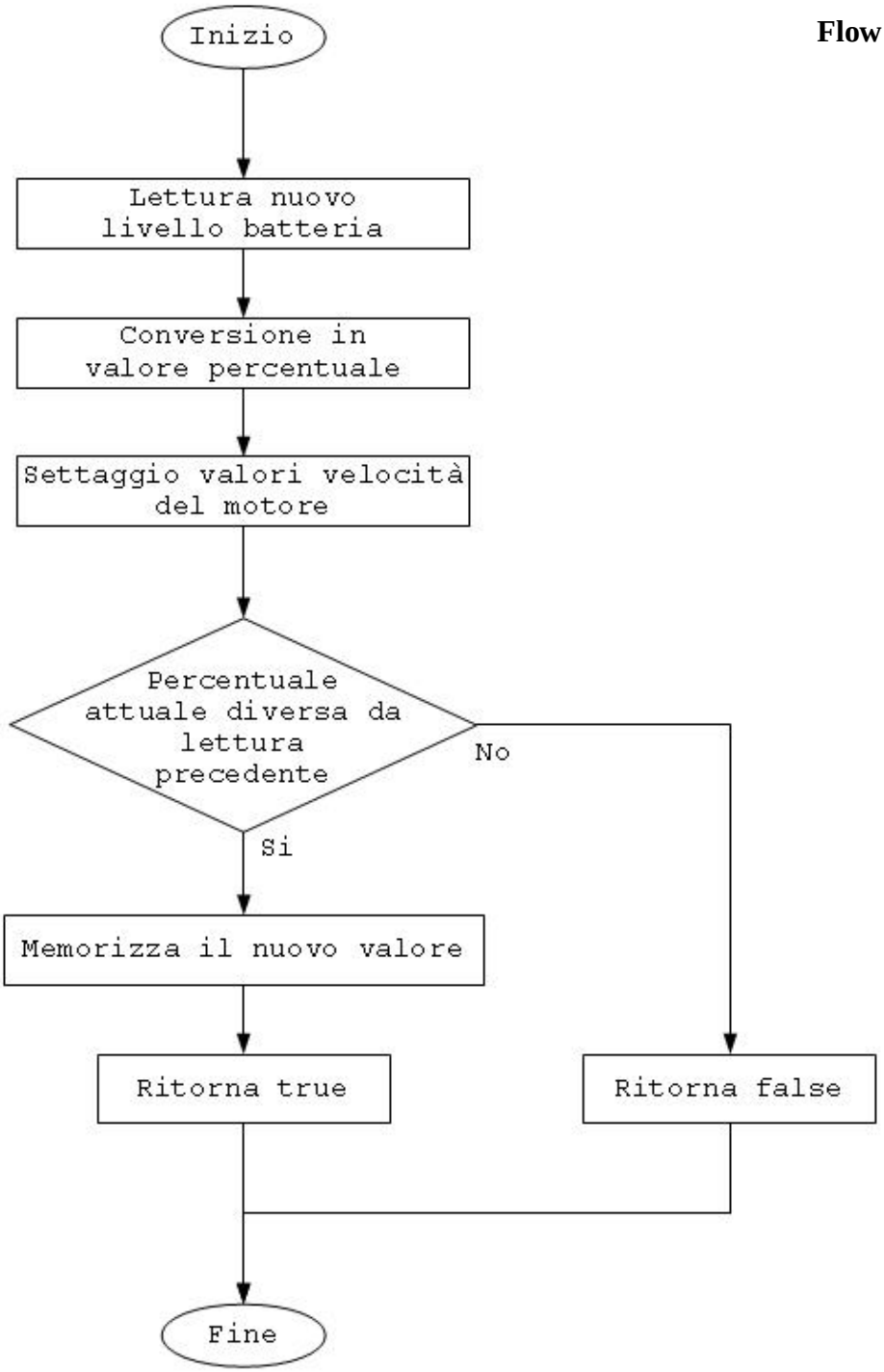
Flow chart lettura_foto()



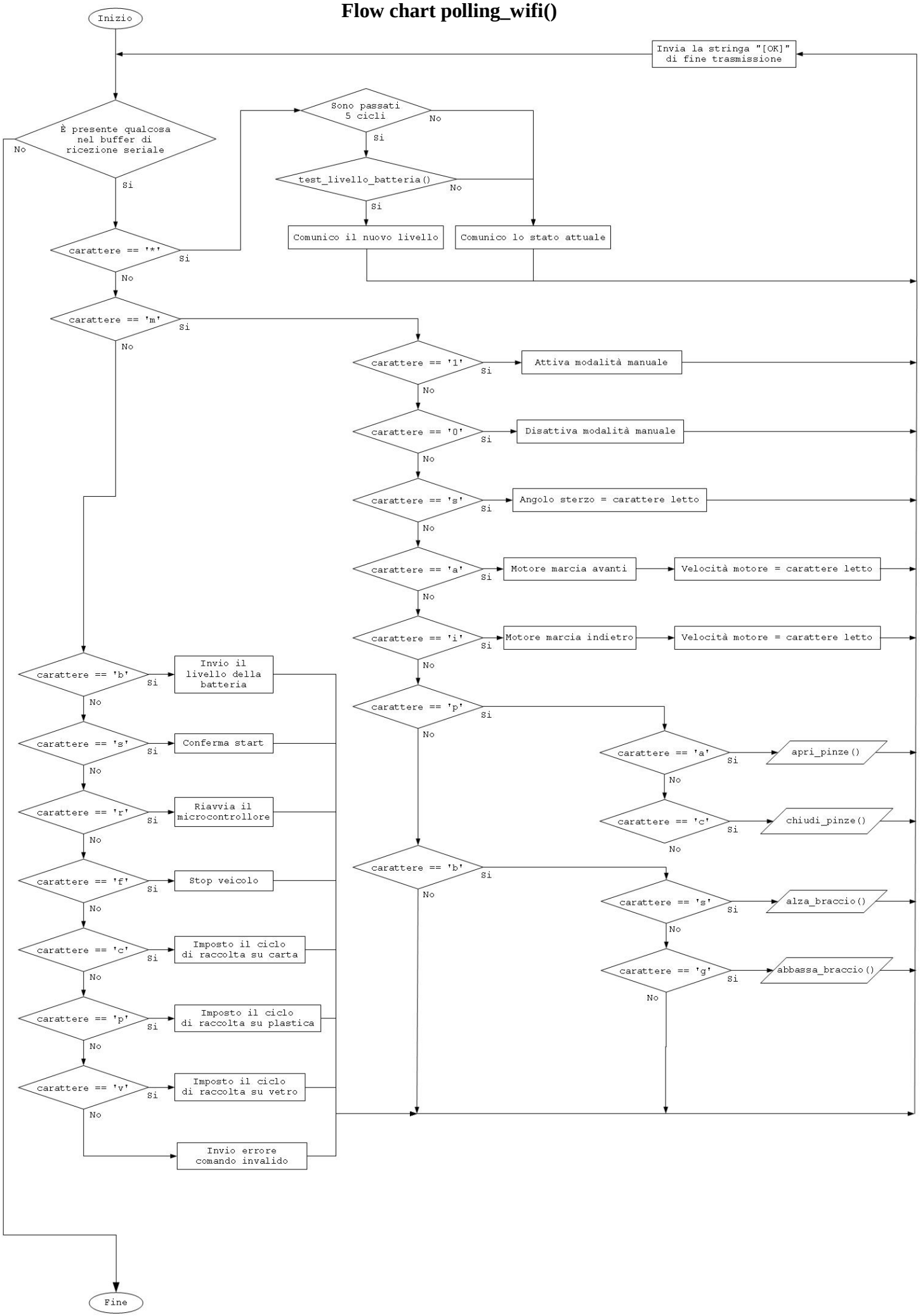
Flow chart check_colore()



Flow chart test_livello_batteria()



Flow chart polling_wifi()



4.3 LISTATO SOFTWARE MICROCONTROLLORE

```
#include <Servo.h>
#define DEBUG 0

Servo sterzo;
Servo brandeggio;
Servo pinze;

const int pin_motore =              2;
const int pin_braccio =             3;
const int pin_posAD =               4;
const int pin_posAS =               5;
const int pin_posPD =               6;
const int pin_posPS =               7;

const int pin_sterzo =             22;
const int pin_brandeggio =         23;
const int pin_pinze =              24;

const int pin_finecorsa_basso =    25;
const int pin_finecorsa_alto =     26;

const int pin_lamp_avv =           27;
const int pin_frecce_dx =          28;
const int pin_frecce_sx =          29;

const int pin_wifi_connected =     30; //pin connessione wifi attiva
const int pin_wifi_reset =         31;

const int trigpin =                32; //pin trigger ultrasuoni
const int echopin =                33; //echo pin ultrasuoni

const int pin_led_rosso =          47;
const int pin_led_verde =          48;
const int pin_led_blu =            49;

const int pin_braccio_dir2 =       50;
const int pin_braccio_dir1 =       51;

const int pin_motore_dir2 =        52;
const int pin_motore_dir1 =        53;

const int pin_sens_extD =          A0;
const int pin_sens_D =             A1;
const int pin_sens_S =             A2;
const int pin_sens_extS =          A3;
const int pin_livello_batteria =   A4;
const int pin_lettura_blu =        A5;
const int pin_lettura_verde =      A6;
const int pin_lettura_rosso =      A7;

int sensore_S;
int sensore_D;
int sensore_extS;
int sensore_extD;
int soglia_nero;
int soglia_bianco;
unsigned long time;
```

```
int percentuale_batteria =         -1;
int diff_sensS =                    0;
int diff_sensD =                    0;
int diff_sens_extS =                0;
int diff_sens_extD =                0;
int n_batt_check =                  0;
int vedo_bianco =                   0;
int vel_nor =                      85; //valore PWM velocità normale
int vel_sterzata =                100; //valore PWM velocità sterzata
const int sterzo_max =                 55;//valore sterzata massima a
                                        //velocità normale
const int vel_sollevamento =      100; //velocità sollevamento braccio
const int angolo_brandeggio_su =   55;
const int angolo_brandeggio_giu = 110;
const int angolo_pinze_aperte =    50;
const int angolo_pinze_chiuse =    80;
const int diff_sens = 50; //soglia differenza fra i due
                           //sensori

String stato = "Preparazione avvio in corso...";
boolean MANUALE = false;              //flag modalità manuale
boolean ON = false;                   //flag di start
boolean ultrasonic_en = false;        //flag ultrasuoni
boolean avanzamento = false;      //flag ultima memoria movimento motore
int bidone_ciclo_attuale = -1;       //ciclo selezionato
int conta_cicli = 0;                 //contatore cicli per ultrasuoni
int conta_bidoni = 0;//contatore di bidoni per stazione
boolean flag_fine_ciclo = false;
boolean bidone_flag = false;     //flag raccolta completata per stazione


void setup()
{
  #if DEBUG
  Serial.begin(9600);                 // Usb PC
Serial.write("Seriale inizializzata...\n\n");
#endif

  Serial1.begin(9600);                // Serial WiFi

  #if DEBUG
Serial.write("Configurazione in corso...\n");
#endif

  pinMode(pin_motore_dir1,OUTPUT);
  pinMode(pin_motore_dir2,OUTPUT);
  pinMode(pin_motore,OUTPUT);
  pinMode(pin_braccio_dir1,OUTPUT);
pinMode(pin_braccio_dir2,OUTPUT);
pinMode(pin_braccio,OUTPUT);

  pinMode(pin_led_rosso,OUTPUT);
  pinMode(pin_led_verde,OUTPUT);
  pinMode(pin_led_blu,OUTPUT);

pinMode(pin_finecorsa_basso,INPUT_PULLUP);
  pinMode(pin_finecorsa_alto,INPUT_PULLUP);
```

```
pinMode(pin_frecce_dx,OUTPUT);
  pinMode(pin_frecce_sx,OUTPUT);
  pinMode(pin_lamp_avv,OUTPUT);
  pinMode(pin_posAD,OUTPUT);
  pinMode(pin_posAS,OUTPUT);
  pinMode(pin_posPD,OUTPUT);
  pinMode(pin_posPS,OUTPUT);
  analogWrite(pin_posAD,65);
  analogWrite(pin_posAS,65);

  pinMode(pin_wifi_reset,OUTPUT);
  pinMode(pin_wifi_connected,INPUT);
  digitalWrite(pin_wifi_reset,HIGH);

  pinMode(trigpin,OUTPUT);
  pinMode(echopin,INPUT);

sterzo.attach(pin_sterzo);
  brandeggio.attach(pin_brandeggio);
  pinze.attach(pin_pinze);

  alza_braccio();
sterzo.write(90);
  chiudi_pinze();

  digitalWrite(pin_led_rosso,LOW);
  digitalWrite(pin_led_verde,LOW);
  digitalWrite(pin_led_blu,LOW);

  digitalWrite(pin_frecce_sx,LOW);
  digitalWrite(pin_frecce_dx,LOW);
  digitalWrite(pin_lamp_avv,LOW);

time=millis();

  #if DEBUG
  Serial.write("Calcolo valore di soglia...\n");
  #endif

  stato = "Calcolo valore di soglia...";
  calcola_soglia();      //imposto soglie nero e bianco

  #if DEBUG
  Serial.write("Soglia nero: ");
  Serial.print(soglia_nero,DEC);
  Serial.write("\n");
  Serial.write("Pronto per la connessione \n");
  #endif

  stato = "Pronto per partire!";
}

void loop()
{
  if(digitalRead(pin_wifi_connected) == LOW)      //verifico connessione
{
    polling_wifi();
```

```
    if(MANUALE)           //modaltà manuale
    {
      stato = "Modalita manuale attiva";
}
    else
    {
      if(ON && ultrasonic_en)
{
        gestione_avanzamento();
        digitalWrite(pin_frecce_dx,LOW);
digitalWrite(pin_frecce_sx,LOW);
        digitalWrite(pin_lamp_avv,LOW);
      }
      else
      {
        pwm_marcia(0);
        if(ultrasonic_en)
{
          stato = "Fermo da comando remoto";
digitalWrite(pin_frecce_dx,LOW);
          digitalWrite(pin_frecce_sx,LOW);
          digitalWrite(pin_lamp_avv,LOW);
}
        else
        {
          stato = "Fermo per ostacolo";
digitalWrite(pin_frecce_dx,HIGH);
          digitalWrite(pin_frecce_sx,HIGH);
          digitalWrite(pin_lamp_avv,HIGH);
        }
      }

      if((millis()-100)>time)
      {
        ultrasuoni();
        time=millis();
}
    }
  }
  else
  {
    pwm_marcia(0);         // se la connessione è caduta ferma il veicolo
  }
}


void gestione_avanzamento()
{
  lettura_foto();

int rapp;

  #if DEBUG
  Serial.write("Interno sx: ");
  Serial.print(sensore_S,DEC);
Serial.write("\n");
  Serial.write("Interno dx: ");
  Serial.print(sensore_D,DEC);
```

```
  Serial.write("Esterno sx: ");
Serial.print(sensore_extS,DEC);
Serial.write("\n");
  Serial.write("Esterno dx: ");
Serial.print(sensore_extD,DEC);
Serial.write("\n\n");
  #endif

  if(sensore_S < soglia_nero   && sensore_D < soglia_nero)   //entrambi
                                                             //leggono bianco
  {
    if(vedo_bianco>=3)
    {
      vedo_bianco = 0;
      #if DEBUG
      Serial.write("Fermo per il bianco\n\n");
#endif
      if(!flag_fine_ciclo)
      {
        if(!bidone_flag)
        {
          polling_wifi();
stato = "Fermo per il bianco";
delay(350);
          polling_wifi();

          pwm_marcia(0);

polling_wifi();

          apri_pinze();

          polling_wifi();

          abbassa_braccio();

polling_wifi();

          if(check_colore())
{
            chiudi_pinze();
            delay(100);
            alza_braccio();
            polling_wifi();
            delay(100);
            abbassa_braccio();
            apri_pinze();
            bidone_flag = true;
          }

          alza_braccio();
          sterzo.write(90);
          pwm_marcia(vel_nor);

          polling_wifi();

          stato = "Pronto a partire...";
delay(350);
```

```
          polling_wifi();
          delay(350);
        }
        conta_bidoni++;
if(conta_bidoni>=3)
        {
          conta_bidoni = 0;
bidone_flag = false;
        }
        delay (7);
      }
      else
      {
flag_fine_ciclo = false;
        pwm_marcia(0);
      }
    }
    else vedo_bianco++;
  }
  else if(abs(sensore_extS-sensore_extD)>diff_sens)     //gestione destra
                                                        //sinistra esterna
  {
      pwm_marcia(vel_nor);
      vedo_bianco = 0;
      if(sensore_extS > sensore_extD)         // dx bianco -> volto a sx
      {
          rapp = 0;
      }
      else                                  // sx bianco -> volto a dx
{
          rapp = 180;
      }
      if(sterzo.read()>rapp)
{
        if((90-sterzo.read())>sterzo_max) pwm_marcia (vel_sterzata);
        sterzo.write(sterzo.read()-10);
      }
      else
      {
        if((sterzo.read()-90)>sterzo_max) pwm_marcia (vel_sterzata);
        sterzo.write(sterzo.read()+10);
      }
  }
  else
  {
     sterzo.write(90);
     pwm_marcia(vel_nor);
     vedo_bianco = 0;
  }
}


void calcola_soglia()
{
  int media_bianco = 0;
  int media_nero = 0;
  int diff;
```

```
  for(int i=0;i<10;i++)                          // media foto interne
{
    sensore_S = analogRead(pin_sens_S);
    sensore_D = analogRead(pin_sens_D);
media_nero += (sensore_S + sensore_D)/2;
    delay(100);
  }
  media_nero = (media_nero/10)-100;
diff = abs(sensore_S-sensore_D);
  if(sensore_S>sensore_D)
  {
    diff_sensS = diff/2;
    diff_sensD =- diff/2;
  }
  else
  {
    diff_sensD = diff/2;
    diff_sensS =- diff/2;
}
  soglia_nero = media_nero;

  for(int i=0;i<10;i++)                          // media foto esterne
{
    sensore_extS = analogRead(pin_sens_extS);
    sensore_extD = analogRead(pin_sens_extD);
media_bianco += (sensore_extS + sensore_extD)/2;
    delay(100);
  }
  media_bianco = (media_bianco/10)+100;
diff = abs(sensore_extS-sensore_extD);

  if(sensore_extS>sensore_extD)
  {
    diff_sens_extS = diff/2;
    diff_sens_extD =- diff/2;
  }
  else
  {
    diff_sens_extD = diff/2;
diff_sens_extS =- diff/2;
  }
  soglia_bianco = media_bianco;
}


void pwm_marcia(byte pwmdc)
{
  if(avanzamento == false && pwmdc != 0)
  {
    digitalWrite(pin_motore_dir1,LOW);
    digitalWrite(pin_motore_dir2,HIGH);          // marcia avanti
    analogWrite(pin_motore,200);
    analogWrite(pin_posPD,65);
analogWrite(pin_posPS,65);
    delay(100);
  }
  analogWrite(pin_motore,pwmdc);
  avanzamento = true;
```

```
  if(pwmdc == 0)
  {
    digitalWrite(pin_motore_dir1,LOW);
    digitalWrite(pin_motore_dir2,LOW);          //fast stop
analogWrite(pin_motore,255);                  //enable alto
    avanzamento = false;
    analogWrite(pin_posPD,255);
    analogWrite(pin_posPS,255);
  }
}


void ultrasuoni()
{
  int distance;
  digitalWrite(trigpin,LOW);
  delayMicroseconds (5);
  digitalWrite(trigpin,HIGH);
  delayMicroseconds (10);
  digitalWrite(trigpin,LOW);

  distance = pulseIn(echopin,HIGH);
  distance = distance/55,2466;

  #if DEBUG
  Serial.print("Distanza ultrasuoni: ");
  Serial.println(distance);
  #endif

  if(distance < 25)
  {
    ultrasonic_en = false;
  }
  else
  {
    ultrasonic_en = true;
}
}


void alza_braccio()
{
  #if DEBUG
  Serial.write("Alzo il braccio...\n");
  #endif

  stato = "Alzo il braccio...";

  digitalWrite(pin_braccio_dir1,HIGH);
  digitalWrite(pin_braccio_dir2,LOW);                  //il braccio sale
while(digitalRead(pin_finecorsa_alto)==HIGH)
{
    analogWrite(pin_braccio,vel_sollevamento+154);
    polling_wifi();
  }
  analogWrite(pin_braccio,0);

  brandeggio_su();
```

```
  chiudi_pinze();

  #if DEBUG
  Serial.write("Fine...\n");
  #endif

  stato = "Fine...";

  digitalWrite(pin_frecce_sx,LOW);
  digitalWrite(pin_frecce_dx,LOW);
digitalWrite(pin_lamp_avv,LOW);

polling_wifi();
}


void abbassa_braccio()
{
  #if DEBUG
  Serial.write("Abbasso il braccio...\n");
  #endif

  stato = "Abbasso il braccio...";
digitalWrite(pin_frecce_sx,HIGH);
  digitalWrite(pin_frecce_dx,HIGH);
  digitalWrite(pin_lamp_avv,HIGH);

brandeggio_giu();

  digitalWrite(pin_braccio_dir1,LOW);
  digitalWrite(pin_braccio_dir2,HIGH);

   while(digitalRead(pin_finecorsa_basso) == HIGH)   //il braccio scende
   {
     analogWrite(pin_braccio,vel_sollevamento);
polling_wifi();
   }
   analogWrite(pin_braccio,0);

   #if DEBUG
Serial.write("Fine...\n");
   #endif

   stato = "Fine...";

polling_wifi();
}


void apri_pinze()
{
  while(pinze.read() > angolo_pinze_aperte)
{
    polling_wifi();
    pinze.write(pinze.read()-1);
delay(75);
  }
```

```
}


void chiudi_pinze()
{
  while(pinze.read() < angolo_pinze_chiuse)
{
    polling_wifi();
    pinze.write(pinze.read()+1);
delay(75);
  }
}


void brandeggio_su()
{
  while(brandeggio.read() > angolo_brandeggio_su)
{
    polling_wifi();
    brandeggio.write(brandeggio.read()-1);
    delay(50);
}
}


void brandeggio_giu()
{
  while(brandeggio.read() < angolo_brandeggio_giu)
{
    polling_wifi();
    brandeggio.write(brandeggio.read()+1);
    delay(50);
  }
}


boolean check_colore()
{
  int rosso;
int blu;
  int verde;
  int minimo;
  int massimo;
  int colore_riconosciuto;
int serie_test_colore[4] = {0, 0, 0, 0};

  for(int a = 0; a <= 10; a++)
  {
    digitalWrite(pin_led_rosso,HIGH);
delay(150);
    rosso = analogRead(pin_lettura_rosso);
digitalWrite(pin_led_rosso,LOW);

    digitalWrite(pin_led_verde,HIGH);
delay(150);
    verde = analogRead(pin_lettura_verde);
digitalWrite(pin_led_verde,LOW);
```

```
    digitalWrite(pin_led_blu,HIGH);
    delay(150);
    blu = analogRead(pin_lettura_blu);
    digitalWrite(pin_led_blu,LOW);

minimo = min(min(rosso,blu), min(blu,verde));

    rosso = rosso-minimo;
    blu = blu-minimo;
    verde = verde-minimo;

    polling_wifi();

#if DEBUG
    Serial.println("------------------");
    Serial.write("Blu: ");
Serial.println(blu,DEC);
    Serial.write("Verde: ");
    Serial.println(verde,DEC);
    Serial.write("Rosso: ");
    Serial.println(rosso,DEC);
    #endif

    if(rosso>15 && blu<17 && verde<10)                    // giallo
{
      serie_test_colore[1] = serie_test_colore[1]+1;
      #if DEBUG
      Serial.println("--> Giallo");
      #endif
    }
    else if(blu>40 && rosso<25)                            // blu
    {
      serie_test_colore[0] = serie_test_colore[0]+1;
      #if DEBUG
      Serial.println("--> Blu");
      #endif
    }
    else if (blu>18 && rosso>10)                          // verde
    {
      serie_test_colore[2] = serie_test_colore[2]+1;
      #if DEBUG
Serial.println("--> Verde");
      #endif
    }
    else                                                  //nulla
{
      serie_test_colore[3] = serie_test_colore[3]+1;
      #if DEBUG
      Serial.println("--> NULLA");
      #endif
    }

    #if DEBUG
    Serial.println("------------------");
    #endif
    delay(1);
  }
```

```
  massimo      =      max(max(serie_test_colore[0],serie_test_colore[1]),
                          max(serie_test_colore[2],serie_test_colore[3]));

  #if DEBUG
  Serial.write("Massimo: ");
  Serial.println(massimo,DEC);
  for(int a = 0; a <= 3; a++)
  {
    Serial.write("Colore: ");
    Serial.println(serie_test_colore[a],DEC);
  }
  #endif

  for(int a=0; a<=3; a++)
{
    if(massimo == serie_test_colore[a])
    {
       colore_riconosciuto = a;
    }
  }

  #if DEBUG
  Serial.write("colore riconosciuto: ");
  Serial.println(colore_riconosciuto,DEC);
  Serial.write("Bidone ciclo attuale: ");
  Serial.println(bidone_ciclo_attuale,DEC);
  #endif

  if(colore_riconosciuto == bidone_ciclo_attuale)
return true;
  else
    return false;
}


boolean test_livello_batteria()
{
  int valore;
  valore = analogRead(pin_livello_batteria);

  #if DEBUG
  Serial.write("Batteria:");
  Serial.print(valore,DEC);
  Serial.write("\n");
  #endif

  if(valore > 875)
  {
    valore = 100;
  }
  else if(valore > 850)
  {
    valore = 75;
    vel_nor = 90;
    vel_sterzata = 105;
  }
  else if(valore > 825)
  {
```

```
    valore = 50;
    vel_nor = 95;
    vel_sterzata = 110;
  }
  else if(valore > 750)
{
    valore = 25;
    vel_nor = 100;
    vel_sterzata = 115;
  }
  else
    valore = 0;

  if(percentuale_batteria != valore)
  {
    percentuale_batteria = valore;
    return true;
}
  else
    return false;
}


void lettura_foto()
{
  sensore_S = analogRead(pin_sens_S);
  sensore_D = analogRead(pin_sens_D);
  sensore_extS = analogRead(pin_sens_extS)-diff_sensS;
  sensore_extD = analogRead(pin_sens_extD)-diff_sensD;
}


void polling_wifi()                    // GESTIONE COMANDI DA SERIAL WIFI
{
  while(Serial1.available() > 0) {
    char cmd = Serial1.read();
    char tmp_stato[1024];
    stato.toCharArray(tmp_stato,1024);
    char tmp2m;
    char tmp3m;
    switch(cmd)
    {
      case '*':
        if(n_batt_check > 5)
        {
          if (test_livello_batteria())
{
              Serial1.write("Batteria:");
              Serial1.print(percentuale_batteria,DEC);
} else Serial1.write(tmp_stato);
          n_batt_check = 0;
        }
        else
        {
          n_batt_check++;
Serial1.write(tmp_stato);           // invia lo stato attuale
}
      break;
```

```
      case 'm':
        delay(30);
        tmp2m = Serial1.read();
        switch(tmp2m)
        {
          case '1':
            MANUALE = true;
          break;
          case '0':
            MANUALE = false;
          break;
          case 's':                                        //gestione sterzo
            delay(30);
            tmp3m = Serial1.read();
            sterzo.write(map(tmp3m,0,127,0,180));
            #if DEBUG
Serial.write("Angolo sterzo: ");
            Serial.println(tmp3m,DEC);
#endif
          break;
          case 'a':                                         // marcia avanti
            delay(30);
            tmp3m = Serial1.read();
            digitalWrite(pin_motore_dir1,LOW);
            digitalWrite(pin_motore_dir2,HIGH);
            analogWrite(pin_motore, map(tmp3m,0,127,0,255));
            #if DEBUG
Serial.write("Avanti: ");
            Serial.println(tmp3m,DEC);
#endif
          break;
          case 'i':                                       // marcia indietro
            delay(30);
            tmp3m = Serial1.read();
            digitalWrite(pin_motore_dir1,HIGH);
            digitalWrite(pin_motore_dir2,LOW);
            analogWrite(pin_motore, map(tmp3m,0,127,0,255));
            #if DEBUG
Serial.write("Indietro: ");
            Serial.println(tmp3m,DEC);
#endif
          break;
          case 'p':
            delay(30);
            tmp3m = Serial1.read();
            if(tmp3m=='a') apri_pinze();
            else if (tmp3m=='c') chiudi_pinze();
          break;
          case 'b':
            delay(30);
            tmp3m = Serial1.read();
if(tmp3m=='s') alza_braccio();
else if (tmp3m=='g') abbassa_braccio();
          break;
        }
      break;
case 'b':
        Serial1.print(percentuale_batteria,DEC);
```

```
      break;
      case 's':                                          // conferma partenza
ON = true;
      break;
case 'f':                                          // ferma il camion
        ON = false;
break;
      case 'r':                                          // reset
        Serial1.print("[OK]");
setup();
      break;
      case 'c':            // ciclo carta = blu
        bidone_ciclo_attuale = 0;
        conta_bidoni = 0;
        bidone_flag = false;
      break;
      case 'p':            // ciclo plastica = giallo
        bidone_ciclo_attuale = 1;
        conta_bidoni = 0;
        bidone_flag = false;
      break;
      case 'v':            // ciclo vetro = verde
        bidone_ciclo_attuale = 2;
        conta_bidoni = 0;
bidone_flag = false;
      break;
      case 'e':
        flag_fine_ciclo = true;
      break;
      default:
        Serial1.write("INVALID");
      break;
}
    Serial1.print("[OK]");
  }
}
```

4.4INTERFACCIA GRAFICA DI GESTIONE REMOTA

L'interfaccia di tele gestione è stata sviluppata in ambiente visuale mediante l'IDE Microsoft Visual Studio facendo uso del Framework .Net versione 4. Il linguaggio di programmazione impiegato è C#.
L'applicazione realizzata è costituita principalmente da 3form (finestre):

- form principale per il controllo di processo,
- form per il pilotaggio in modalità manuale del veicolo
- form per la configurazione remota dell'interfaccia Wi-Fi

4.4.1 Form principale

Quando il software viene lanciato, nel suo punto di ingresso principale, carica il form di controllo di processo. Tale form consente innanzitutto di effettuare la connessione con il veicolo. La comunicazione avviene tramite l'apertura di un socket con l'indirizzo IP e il numero di porta di livello applicazione specificati negli appositi text box del form stesso.
Una volta effettuata la connessione vengono abilitati i comandi che permettono di selezionare la fase di lavoro da intraprendere e il pulsante di attivazione della modalità di controllo manuale. Allo stesso tempo ha inizio il processo di aggiornamento della barra di stato che da questo momento in poi (fino alla disconnessione) visualizzerà in modo dettagliato, istante per istante, lo stato in cui si trova il veicolo e le operazioni in corso di svolgimento.
Tramite pulsanti grafici rappresentanti le diverse tipologie di bidone l'operatore è in grado di selezionare il ciclo di raccolta che desidera sia intrapreso e, dopo aver specificato tramite apositocount box il numero di postazioni di bidoni presenti sul percorso, potrà dare inizio alla fase di lavoro premendo il tasto S*tart*.
In qualunque momento è possibile sospendere la fase di lavoro in corso premendo il pulsante S*top*. Dopodiché la fase potrà essere ripresa premendo nuovamente su *Start*.
Nell'angolo inferiore sinistro del form è presente un riquadro bianco all'interno del quale verranno visualizzate delle animazioni riguardanti alcuni avvisi provenienti dal veicolo. Ad esempio un'animazione relativa alla movimentazione del braccio e una relativa al rilevamento di ostacoli sul percorso.
Nell'angolo superiore destro è invece presente un indicatore del livello della batteria e un pulsante che consente di accedere all'interfaccia di configurazione remota del modulo Wi-Fi presente sul veicolo.
Durante l'esecuzione del programma viene mantenuto un file di log utile all'analisi di eventuali problemi riscontrati. Inoltre alla chiusura dell'applicazione vengono salvati per poi essere ripristinati al successivo avvio:

- l'indirizzo IP del veicolo;
- il numero di porta di livello applicazione;
- il numero di stazioni di raccolta.

4.4.1.1 Il pulsante Connetti/Disconnetti

Quando la comunicazione tra il veicolo e il PC è inattiva il pulsante presenterà l'etichetta *Connetti*. Premendolo verrà richiamata la funzione `void connetti()` la quale si occupa di iniziare la comunicazione tra i due host mediante l'apertura di un socket con indirizzo destinazione l'indirizzo IP e porta specificati dall'utente nell'apposito box. Tramite socket si spedisce il carattere `*` attendendo un riscontro da parte del dispositivo remoto. Se il dispositivo risponde la connessione è stabilita. La comunicazione via socket col dispositivo remoto è delegata alla funzione `stringWifiCmd(stringtxtxt)` (si veda il paragrafo 4.3.1.5).
Stabilita la connessione vengono abilitati i pulsanti di selezione dei bidoni e il box per impostare il numero di stazioni presenti sul percorso, i pulsanti di reset e stop e quello di apertura del form per la modalità manuale.
Viene poi attivato un timer, chiamato `PollingTmr` (paragrafo 4.3.1.4), con interrupt, che è stato impostato avvenire ogni 250ms, che si occupa di mantenere un costante dialogo di aggiornamento tra PC e veicolo.
Si legge infine il livello della batteria e sia aggiorna l'apposito indicatore.
Quando, invece, la comunicazione tra il dispositivo e l'host è attiva il pulsante avrà etichetta Disconnetti. Alla pressione si richiama quindi la funzione `void disconnetti()`. Tale funzione si occupa di:

- arrestare il timer `PollingTmr` richiamando il metodo `Stop()` della classe `Timer`;
- arrestare il veicolo inviando il carattere `s` tramite la funzione `WiFiCmd`;
- disabilitare i comandi dell'interfaccia grafica;

4.4.1.2 Selezione del ciclo di raccolta

Le impostazioni relative al ciclo di raccolta vengono settate tramite quattro controlli: tre pulsanti corrispondenti alle tre tipologie di bidone selezionabili e un box numerico per il settaggio del numero di stazioni di raccolta. Alla pressione di uno dei pulsanti di selezione bidone si invia al veicolo tramite `WiFiCmd` il carattere ad esso corrispondente (`c` → carta, `p` → plastica, `v` → vetro). Dopodiché si scrive nell'etichetta ciclo attuale il nome dell'attuale raccolta di bidoni (Carta, Plastica, Vetro), i pulsanti vengono disabilitati assieme al box di impostazione del numero di stazioni e si abilita il pulsante di *Start*.

4.4.1.3 Il pulsante reset

Alla pressione di tale pulsante viene spedito un carattere che comunica al veicolo di intraprendere le operazioni di reset. Dopdiché si effettua la disconnessione.

4.4.1.4 PollingTmr

Ad ogni interrupt di tale timer impostato per raggiungere l'overflow ogni 250ms si richiede al veicolo di inviare lo stato in cui si trova. Se lo stato è cambiato si procede ad intraprendere una particolare azione ad esso legata:

- se lo stato equivale a `"Fermo per il bianco"` si attiva l'animazione relativa alla movimentazione del braccio;
- se lo stato equivale a `"Pronto a partire... "` l'animazione del braccio viene arrestata e si incrementa il computo dei bidoni svuotati;
- se lo stato equivale `"Fermo per ostacolo"` si attiva l'animazione relativa alla rilevazione di un ostacolo.
- se all'interno della stringa di stato è presente la dicitura `"Batteria:"` significa che dopo i due punti sarà presente un numero indicante in percentuale il livello della batteria. Si procede perciò ad aggiornare l'apposito indicatore.
-

In tutti gli altri casi si procede all'aggiornamento della barra di stato.

4.4.1.5 WiFiCmd

La funzione `stringWiFiCmd(stringtxtxt)` si occupa di aprire un socket TCP/IP utilizzando come indirizzo e numero di porta sorgenti quelli specificati negli appositi box del form principale. L'aperura e l'utilizzo del socketè resa possibile dai metodi del namespace`System.Net` in particolare la classe `System.Net.Sockets.Socket`. Dopo aver aperto il socket si trasforma la stringa ricevuta in argomento in un array di caratteri (bytes) e si procede all'invio. Dopodiché si resta in attesa di una risposta, il cui contenuto dopo essere stato convertito in stringa verrà utilizzato come valore di ritorno della funzione.

4.4.2 Form di modalità manuale

L'apertura di tale form è causata dalla pressione nella finestra principale del pulsante *Modalità manuale*.
Quando il formviene caricato si procede innanzitutto a sospendere la fase di lavoro in corso, dopodiché il veicolo reta in attesa di comandi dall'utente.
Il corpo della finestra è costituito da 2 linee perpendicolari rappresentanti gli assi di riferimento cartesiano e da una pallina rossa posta al centro di essi. Quando l'utente trascina quest'ultima se ne rilevano le coordinate riferite agli assi. Se l'ascissa è positiva significa che il veicolo dovrà avanzare in avanti, se negativa dovrà muoversi in retromarcia. Dopodiché si ottiene la velocità mediante il valore assoluto raggiunto dall'ordinata della pallina. Tramite l'ordinata si calcolano invece i valori dell'angolo di sterzatura. Un'ordinata negativa corrisponde allo sterzo a sinistra, positiva allo sterzo verso destra. Anche in questo caso il valore assoluto raggiunto dall'ordinata identifica l'ampiezza dell'angolo di sterzatura.
Operate le dovute conversioni dei valori si procede all'aggiornamento di tre variabili: `charactSterzo`, `charactDir`, `charactVel` contenenti rispettivamente l'angolo di sterzatura, la direzione di avanzamento (avanti o retromarcia) e la velocita di spostamento.
Quando la pallina viene rilasciata essa ritorna al centro degli assi causando un reset ai valori iniziali delle sopracitate variabili. In particolare `actSterzo` viene portata ad valore relativo allo sterzo in posizione centrale e `actVel` viene portata a zero causando l'arresto del veicolo.
Attraverso un timer, il cui interrupt si verifica ogni 100ms, vengono comunicati i valori assunti dalle variabili al microcontrollore tramite la funzione WiFiCmd (paragrafo 4.3.1.5) passando come argomento una

stringa gli opportuni caratteri (per la tabella dei caratteri di comando si faccia riferimento al paragrafo 4.1.13).
Assieme a tale form se ne apre uno aggiuntivo contenente alcuni pulsanti relativi al pilotaggio manuale del braccio. In particolare:

- *Chiudi pinze*,
- *Apri pinze*,
- *Alza braccio*,
- *Abbassa braccio*.

Quando uno del pulsanti viene premuto si procede ad inviare tramite la funzione WiFiCmd al microcontrollore il relativo comando identificato da una particolare stringa di caratteri.

4.4.3 Form di configurazione remota dell'interfaccia Wi-Fi

Tale form è richiamato dalla pressione, all'interno del menu impostazioni del form principale, del pulsante *Configura accesso Wi-Fi*.
Al caricamento del formviene avviata una routine che si occupa di individuare il modulo Wi-Fi del veicolo all'interno della rete a cui è connesso il PC. Se il dispositivo viene rilevato si richiama l'interfaccia webdi configurazione dello stesso. Per la visualizzazione di tale interfaccia si fa uso del controllo `WebBrowser` messo a disposizione da C# all'interno del namespace`System.Windows.Forms`. Per facilitare l'intervento dell'utente l'autenticazione all'interno dell'interfaccia web viene gestita in background tramite Http Basic Authenticationspecificando le credenziali di accesso all'interno dello stesso URL con cui si richiama la pagina.

4.5LISTATO SOFTWARE INTERFACCIA GRAFICA

4.5.1 Form principale

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Net;
using System.Net.Sockets;
using System.IO;

namespace camion_pc_gui
{
publicpartialclassForm1 : Form
    {
constint CONNTIMEOUT = 5000; // timeout in millisecondi per le richieste
publicstring IP = "10.10.100.254";
publicint PORT = 8899;
string ERRBUFFER = "";
string STATUSBAK = "Non connesso!";
publicint POLLING_TIME_SHORT = 250;
publicint POLLING_TIME_LONG = 1000;
bool CONNECTED = false;

publicstring EXTSERVERPTH = "http://marcosim.homepc.it:8080/sites/AGV_Differenziata/r_control.php";
string OLDWEBSEND = "";

string SETTINGS_INI_PATH = Application.StartupPath + "\\camionPCguiSettings.ini";
string LOG_PATH = Application.StartupPath + "\\camionPCguiLog.log";
Dictionary<string, string> SETTINGS = newDictionary<string,string>();

int stazioniCompletate = 0;

string[] braccioAniList = {"0","1","2","3","4","5","4","3","2","0"};
int braccioAniIndex = 0;
string[] obstacleAniList = { "1", "2", "3", "4", "5", "5", "5", "5" };
int obstacleAniIndex = 0;

public Manual_mode Manual_mode_win;

public Form1()
        {
            InitializeComponent();
        }

privatevoid Form1_Load(object sender, EventArgs e)
        {
            chgPollingTime(POLLING_TIME_SHORT);
            readSettings();
statusbar.Text = "Pronto per la connessione";
WebSendTmr.Start();
            progressbar.Hide();
            writeToLog("Program started... Ready!");
        }

publicvoid chgPollingTime(int interval)
        {
            PollingTmr.Interval = interval;
        }

void readSettings()
        {
if (File.Exists(SETTINGS_INI_PATH))
            {
String[] lines = File.ReadAllLines(SETTINGS_INI_PATH);
```

```
for (int i = 0; i < lines.Length; i++)
                {
string line = lines[i].Replace(" = ", "=").Replace(" =", "=").Replace("= ", "=");
string[] tmp = line.Split('=');
if (tmp.Length == 2)
                        SETTINGS[tmp[0]] = tmp[1];
                }
            }

string temp = "";
if (SETTINGS.TryGetValue("NPostazioni", out temp))
numeroPostazioni.Value = Convert.ToDecimal(SETTINGS["NPostazioni"]);
if (SETTINGS.TryGetValue("Conn_IP", out temp))
                IPbox.Text = SETTINGS["Conn_IP"];
if (SETTINGS.TryGetValue("Conn_Port", out temp))
                PORTbox.Text = SETTINGS["Conn_Port"];
}

void saveSettings()
        {
            SETTINGS["NPostazioni"] = numeroPostazioni.Value.ToString();
SETTINGS["Conn_IP"] = IPbox.Text;
            SETTINGS["Conn_Port"] = PORTbox.Text;

String text = "[Camion PC gui]\r\n\r\n";
foreach (string key in SETTINGS.Keys)
            {
                text += key +" = " + SETTINGS[key] + "\r\n";
            }
File.WriteAllText(SETTINGS_INI_PATH, text);
        }

void writeToLog(string logaddtxt) //LOG_PATH DateTime.UtcNow
        {
StreamWriter log = File.AppendText(LOG_PATH);
            log.WriteLine(DateTime.UtcNow + "\t" + logaddtxt);
            log.Close();
        }

publicstring WifiCmd(string txtxt)
        {
// Buffer di ricezione
byte[] bytes = newbyte[1024];
string result = "";

// Connect to a remote device.
try
            {
                IPEndPoint remoteEP = new IPEndPoint(IPAddress.Parse(IP), PORT);

// Crea il socket
                Socket sender = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);
                sender.SendTimeout = CONNTIMEOUT;
                sender.ReceiveTimeout = CONNTIMEOUT;

try
                {
// Tenta apertura connessione
                    sender.Connect(remoteEP);

// Trasforma la stringa in un array di byte (di caratteri)
byte[] msg = Encoding.ASCII.GetBytes(txtxt);

// Invia i dati col socket
int bytesSent = sender.Send(msg);

// Gestisce la risposta
int bytesRec = sender.Receive(bytes);
                    result += Encoding.ASCII.GetString(bytes, 0, bytesRec);
```

```
// Release the socket.
                    sender.Shutdown(SocketShutdown.Both);
                    sender.Close();

return result.Replace("[OK]", "");
                }
catch (ArgumentNullException ane)
                {
                    ERRBUFFER = "Boh";
}
catch (SocketException se)
                {
                    ERRBUFFER = "Il dispositivo remoto non è connesso o non risponde.\r\nVerifica di
avere inserito correttamente indirizzo IP e porta.";
                }
catch (Exception e)
                {
                    ERRBUFFER = "Errore sconosciuto nel tentativo di stabilire la connessione col
dispositivo remoto.";
                }

            }
catch (Exception e)
            {
                ERRBUFFER = "Impossibile creare il socket per la connessione.\r\nVerifica di avere
inserito un IP e porta validi.";
}

return"//ERROR//";
        }

privatevoid PollingTmr_Tick(object sender, EventArgs e)
{
// Polling automatico al dispositivo
string tmp = WifiCmd("*");
if (tmp == "//ERROR//")
            {

                writeToLog("Il dispositivo ha smesso di rispondere!");
                disconnetti();
                errorShow("Il dispositivo ha smesso di rispondere!", "Connessione fallita!");
}
else
            {
if(tmp.IndexOf("Batteria:")>=0) {
string tmpDat = tmp.Replace("Batteria:","");
                    set_batt(tmpDat);
                }
elseif (STATUSBAK != tmp)
                {
                    statusbar.Text = tmp;
                    STATUSBAK = tmp;
writeToLog("Status changed: [" + tmp + "]");
if (STATUSBAK == "Fermo per il bianco")
                    {
                        startBraccioAni();
                    }
if (STATUSBAK == "Pronto a partire...")
                    {
                        stopBraccioAni();
                        stazioniCompletate++;
if (stazioniCompletate < numeroPostazioni.Value)
                        {
                            cicloProgressText.Text = "" + stazioniCompletate + " di " +
numeroPostazioni.Value + " stazioni completate";
                        }
elseif (stazioniCompletate >= numeroPostazioni.Value)
                        {
                            endCicloRaccolta();
                        }
```

```
else
                        {
                            WifiCmd("e");
                        }
                    }
if (STATUSBAK == "Fermo per ostacolo")
{
if(obstacleAni.Enabled == false)
                            obstacleAni.Start();
                    }
                }
            }
        }

void set_batt(string tmpDat)
        {
if (tmpDat == "100")
            {
                battery_indicator.Image = new Bitmap(camion_pc_gui.Properties.Resources.battery_5);
toolTip.SetToolTip(battery_indicator, "Carica completa!");
}
elseif (tmpDat == "75")
            {
                battery_indicator.Image = new Bitmap(camion_pc_gui.Properties.Resources.battery_4);
                toolTip.SetToolTip(battery_indicator, "Livello carica 75%");
            }
elseif (tmpDat == "50")
            {
                battery_indicator.Image = new Bitmap(camion_pc_gui.Properties.Resources.battery_3);
                toolTip.SetToolTip(battery_indicator, "Livello carica 50%");
            }
elseif (tmpDat == "25")
            {
                battery_indicator.Image = new Bitmap(camion_pc_gui.Properties.Resources.battery_2);
                toolTip.SetToolTip(battery_indicator, "Livello carica 25%");
            }
elseif (tmpDat == "0")
            {
                battery_indicator.Image = new Bitmap(camion_pc_gui.Properties.Resources.battery_1);
toolTip.SetToolTip(battery_indicator, "Livello batteria basso!");
}
            writeToLog("Battery level: [" + tmpDat + "%]");
        }

privatevoid CONNECTbtn_Click(object sender, EventArgs e)
        {
            progressbar.Show();
if (CONNECTbtn.Text == "Connetti")
connetti();
else
                disconnetti();
            progressbar.Hide();
        }

void connetti()
        {
            statusbar.Text = "Tentativo di connessione...";
IP = IPbox.Text;
            PORT = Convert.ToInt32(PORTbox.Text);
if (WifiCmd("*") != "//ERROR//")
            {
                CONNECTED = true;

statusbar.Text = "Tentativo di connessione... POSITIVO!";
                CONNECTbtn.Text = "Disconnetti";
                CONNECTbtn.ToolTipText = "Termina connessione col dispositivo";
IPbox.Enabled = false;
                PORTbox.Enabled = false;

                CARTAbtn.Enabled = true;
                VETRObtn.Enabled = true;
```

```
                PLASTICAbtn.Enabled = true;

                numeroPostazioni.Enabled = true;

//STARTbtn.Enabled = true;
                STOPbtn.Enabled = true;
                RESETbtn.Enabled = true;
                manualModeBtn.Enabled = true;

PollingTmr.Start();

                statusbar.Text = "Attesa selezione bidone...";

                writeToLog("Connetti... Tentativo di connessione... RIUSCITO!");

string batt_tmp = WifiCmd("b");
                set_batt(batt_tmp);
}
else
            {
                errorShow(ERRBUFFER, "Connessione fallita!");
                ERRBUFFER = "";
                statusbar.Text = "Tentativo di connessione... FALLITO!";

                writeToLog("Connetti... Tentativo di connessione... FALLITO!");
            }
        }
void disconnetti()
        {
            statusbar.Text = "Disconnessione...";

            PollingTmr.Stop();

            WifiCmd("f");

            CONNECTbtn.Text = "Connetti";
            CONNECTbtn.ToolTipText = "Effettua connessione col dispositivo";
IPbox.Enabled = true;
            PORTbox.Enabled = true;

            STARTbtn.Enabled = false;
            STOPbtn.Enabled = false;
            RESETbtn.Enabled = false;
            manualModeBtn.Enabled = false;

try { Manual_mode_win.Close(); } catch(Exception e) {}

            CONNECTED = false;

            statusbar.Text = "Disconnesso!";

            writeToLog("Disconnetti... DISCONNESSO!");
        }

void errorShow(string txt, string title = "")
        {
            trayIcon.BalloonTipText = txt;
            trayIcon.BalloonTipIcon = ToolTipIcon.Error;
            trayIcon.ShowBalloonTip(2000);
            MessageBox.Show(txt, title, MessageBoxButtons.OK, MessageBoxIcon.Error);
        }

privatevoid RESETbtn_Click(object sender, EventArgs e)
        {
            reset();
        }

void reset()
        {
            progressbar.Show();
statusbar.Text = "Tentativo di reset iniziato...";
```

```
if (WifiCmd("r") != "//ERROR//")
            {
                statusbar.Text = "Tentativo di reset RIUSCITO!";
                disconnetti();

                CICLOlabel.Text = "Nessuno";
                cicloProgressText.Text = "-";

PLASTICAbtn.Enabled = false;
                CARTAbtn.Enabled = false;
VETRObtn.Enabled = false;

                numeroPostazioni.Enabled = false;

                stopBraccioAni();

                STATUSBAK = "Reset RIUSCITO!";
            }
else
            {
                statusbar.Text = "Tentativo di reset FALLITO!";
}
            progressbar.Hide();
        }

privatevoid STARTbtn_Click(object sender, EventArgs e)
        {
            WifiCmd("s");
        }

privatevoid STOPbtn_Click(object sender, EventArgs e)
        {
            WifiCmd("f");
        }

privatevoid CARTAbtn_Click(object sender, EventArgs e)
        {
            WifiCmd("c");
            CICLOlabel.Text = "Carta";
setCicloRaccolta();
        }

privatevoid PLASTICAbtn_Click(object sender, EventArgs e)
        {
            WifiCmd("p");
            CICLOlabel.Text = "Plastica";
            setCicloRaccolta();
}

privatevoid VETRObtn_Click(object sender, EventArgs e)
        {
            WifiCmd("v");
            CICLOlabel.Text = "Vetro";
setCicloRaccolta();
        }

void setCicloRaccolta()
        {
            PLASTICAbtn.Enabled = false;
CARTAbtn.Enabled = false;
            VETRObtn.Enabled = false;

            numeroPostazioni.Enabled = false;

            STARTbtn.Enabled = true;

stazioniCompletate = 0;

            cicloProgressText.Text = "" + stazioniCompletate + " di " + numeroPostazioni.Value + "
stazioni completate";
```

```
            statusbar.Text = "Pronto per lo start...";
}

void endCicloRaccolta()
        {
            PLASTICAbtn.Enabled = true;
            CARTAbtn.Enabled = true;
            VETRObtn.Enabled = true;

            numeroPostazioni.Enabled = true;

            STARTbtn.Enabled = false;

            cicloProgressText.Text = "Raccolta " + CICLOlabel.Text.ToLower() +" completata!";

CICLOlabel.Text = "Nessuno";

            statusbar.Text = "Attesa selezione bidone...";
}

privatevoid statusbar_TextChanged(object sender, EventArgs e)
{
            trayIcon.Text = "Camion spazzatura GUI\r\n" + statusbar.Text;
}

void showHide()
        {
if (this.WindowState == FormWindowState.Minimized)
            {
this.Show();
this.WindowState = FormWindowState.Normal;
this.TopMost = true;
this.TopMost = false;
            }
else
            {
this.Hide();
this.WindowState = FormWindowState.Minimized;
            }
        }

privatevoid trayIcon_BalloonTipClicked(object sender, EventArgs e)
        {
this.Show();
this.WindowState = FormWindowState.Normal;
this.TopMost = true;
this.TopMost = false;
        }

privatevoid chiudiToolStripMenuItem_Click(object sender, EventArgs e)
        {
this.Close();
        }

privatevoid mostraNascondiToolStripMenuItem_Click(object sender, EventArgs e)
        {
            showHide();
        }

privatevoid trayIcon_MouseDoubleClick(object sender, MouseEventArgs e)
        {
            showHide();
        }

privatevoid Form1_FormClosing(object sender, FormClosingEventArgs e)
{
if (CONNECTbtn.Text == "Disconnetti")
            {
                DialogResult confirm = MessageBox.Show("Sicuro di voler uscire durante la fase di
lavoro?\r\nQuesto comporterà l'arresto della fase e la successiva disconnessione.", "Conferma
chiusura", MessageBoxButtons.YesNo, MessageBoxIcon.Question);
```

```
if (confirm == DialogResult.Yes)
                {
                    disconnetti();
                    saveSettings();
                    writeToLog("Chiusura applicazione!");
}
else
                {
                    e.Cancel = true;
                }
            }
else
            {
                saveSettings();
                writeToLog("Chiusura applicazione!");
            }
        }

privatevoid INFObtn_Click(object sender, EventArgs e)
        {

System.Diagnostics.Process.Start("http://marcosim.homepc.it:8080/sites/AGV_Differenziata/");
        }

privatevoid WebSendTmr_Tick(object sender, EventArgs e)
        {
if (WebSendEnable.Checked)
            {
if (STATUSBAK != OLDWEBSEND)
                {
string tmp = WebSend("status", "val=" + STATUSBAK);
                    ServerResponseBox.Text = tmp;
                    OLDWEBSEND = STATUSBAK;
                }
            }
        }

string WebSend(string cmd, string param = "")
        {
try
            {
                WebRequest wrGETURL = WebRequest.Create(EXTSERVERPTH + "?cmd=" + cmd + "&" + param);
Stream objStream = wrGETURL.GetResponse().GetResponseStream();
StreamReader objReader = newStreamReader(objStream);
string response = objReader.ReadToEnd();
return response;
            }
catch (Exception e)
            {
                e.ToString();
            }
return"//ERROR//";
        }

privatevoid WebConnTestbtn_Click(object sender, EventArgs e)
        {
string tmp = WebSend("testConnection");
if (tmp == "OK")
{
                MessageBox.Show("Il server remoto è disponibile!", "Test connessione riuscito",
MessageBoxButtons.OK, MessageBoxIcon.Information);
            }
else
            {
                MessageBox.Show("Il server remoto NON è disponibile!", "Test connessione fallito",
MessageBoxButtons.OK, MessageBoxIcon.Error);
}
        }

privatevoid configuraAccessoWiFiToolStripMenuItem_Click(object sender, EventArgs e)
        {
```

```
            Form WiFi_Config = new camion_pc_gui.WiFi_Config();
WiFi_Config.ShowDialog();
        }

void startBraccioAni()
        {
            braccioAni.Start();
            braccioAniIndex = 0;
        }

void stopBraccioAni()
        {
            braccioAni.Stop();
            braccioPicture.Image = null;
        }

privatevoid braccioAni_Tick(object sender, EventArgs e)
        {
Object image_tmp = camion_pc_gui.Properties.Resources.ResourceManager.GetObject("braccio_" +
braccioAniList[braccioAniIndex]);
braccioPicture.Image = (Bitmap)image_tmp;

if (braccioAniIndex < braccioAniList.Length - 1) braccioAniIndex++;
else braccioAniIndex = 0;
        }

privatevoid manualModeBtn_Click(object sender, EventArgs e)
{
            DialogResult conf = MessageBox.Show("Entrando in modalità manuale l'eventuale fase di
lavoro in corso sarà interrotta. Procedere?", "Avviso interruzione fase di lavoro",
MessageBoxButtons.YesNo, MessageBoxIcon.Question);
if (conf == System.Windows.Forms.DialogResult.Yes)
            {
                Manual_mode_win = new camion_pc_gui.Manual_mode(this);
                Manual_mode_win.ShowDialog(this);
            }
        }

privatevoid obstacleAni_Tick(object sender, EventArgs e)
        {
Object image_tmp = camion_pc_gui.Properties.Resources.ResourceManager.GetObject("obstacle_" +
obstacleAniList[obstacleAniIndex]);
            braccioPicture.Image = (Bitmap)image_tmp;

if (obstacleAniIndex < obstacleAniList.Length - 1) obstacleAniIndex++;
else
            {
                obstacleAniIndex = 0;
                braccioPicture.Image = null;
if (STATUSBAK != "Fermo per ostacolo") obstacleAni.Stop();
}
        }

    }
}
```

4.5.2 Form modalità manuale

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace camion_pc_gui
{
publicpartialclassManual_mode : Form
    {
        Form1 Form1;

bool EN = false;
char actSterzo = Convert.ToChar(90);
char actDir = 'a';
char actVel = Convert.ToChar(0);

public Manual_mode_toolbar Toolbar_win;

public Manual_mode(Form1 owner)
        {
this.Form1 = owner; // riferimento al form principale
            InitializeComponent();

this.Form1.chgPollingTime(this.Form1.POLLING_TIME_LONG); // rallento il polling principale

sendPolling.Interval = 100; // velocità aggiornamento comandi (velocità, direzione, sterzo)
            sendPolling.Start();

this.Form1.WifiCmd("f"); // fermo il veicolo
this.Form1.WifiCmd("m1"); // entro in modalità manuale

rePos();
        }

privatevoid Manual_mode_Load(object sender, EventArgs e)
        {
            Toolbar_win = new Manual_mode_toolbar(this.Form1);
            Toolbar_win.Top = this.Top;
            Toolbar_win.Left = this.Left + this.Width + 10;
Toolbar_win.Show(); // apro la toolbar per i comandi del braccio
}

void calPos()
        {
if (EN)
            {
int winBordersX = this.Width - this.ClientSize.Width;
int winBordersY = this.Height - this.ClientSize.Height;

int X = Cursor.Position.X - this.Left - winBordersX;
int Y = Cursor.Position.Y - this.Top - winBordersY;

                redBall.Left = X - redBall.Width / 2;
                redBall.Top = Y - redBall.Height / 2;


int Xmid = (X - (this.ClientSize.Width / 2));
int Ymid = -(Y - (this.ClientSize.Height / 2));

string infoText = "";

if (Ymid > 0)
                {
                    infoText = "AVANTI";
```

```
                    actDir = 'a';

if (Ymid < 200) actVel = Convert.ToChar(Ymid / 2);
                }
else
                {
                    infoText = "INDIETRO";

                    actDir = 'i';

if (Ymid > -200) actVel = Convert.ToChar((-Ymid) / 2);
                }

if (Xmid > 0)
                {
                    infoText += " -> destra";

if (Xmid < 200) actSterzo = Convert.ToChar(64 + Xmid / 4);
                }
else
                {
                    infoText += " -> sinistra";

if(Xmid>-200) actSterzo = Convert.ToChar(64 - (-Xmid) / 4);
                }

                infoLabel.Text = infoText;
                coordsLabel.Text = "X = " + Xmid + "     Y = " + Ymid;
            }
        }

void rePos()
        {
            redBall.Left = (this.ClientSize.Width / 2) - (redBall.Width / 2);
            redBall.Top = (this.ClientSize.Height / 2) - (redBall.Height / 2);

actSterzo = Convert.ToChar(64);
            actVel = Convert.ToChar(0);

            infoLabel.Text = "STOP : Muovi il joystick per iniziare...";
coordsLabel.Text = "X = 0     Y = 0";
        }

privatevoid redBall_MouseDown(object sender, MouseEventArgs e)
{
            EN = true; // se il mouse è premuto abilita il movimento
}

privatevoid redBall_MouseUp(object sender, MouseEventArgs e)
{
            EN = false; // il mouse è rilasciato: disabilita il movimento e porta tutto in posizione di
partenza
rePos();
        }

privatevoid redBall_MouseMove(object sender, MouseEventArgs e)
{
            calPos(); // il mouse si sta muovendo: calcolo delle posizioni
}

privatevoid Manual_mode_FormClosing(object sender, FormClosingEventArgs e)
{
            sendPolling.Stop(); // stoppo il polling di aggiornamento velocità e sterzo
this.Form1.WifiCmd("m0"); // esco dalla modalità manuale
            Toolbar_win.Close(); // chiudo la toolbar del braccio
this.Form1.chgPollingTime(this.Form1.POLLING_TIME_SHORT); // ripristino il polling principale ad alta
velocità
        }
```

```
privatevoid sendPolling_Tick(object sender, EventArgs e)
{
// invio valori di sterzo, direzione di marcia e velocità
this.Form1.WifiCmd("ms"+actSterzo);
this.Form1.WifiCmd("m"+ actDir + actVel);
        }
    }
}
```

4.5.3 Toolbar modalità manule

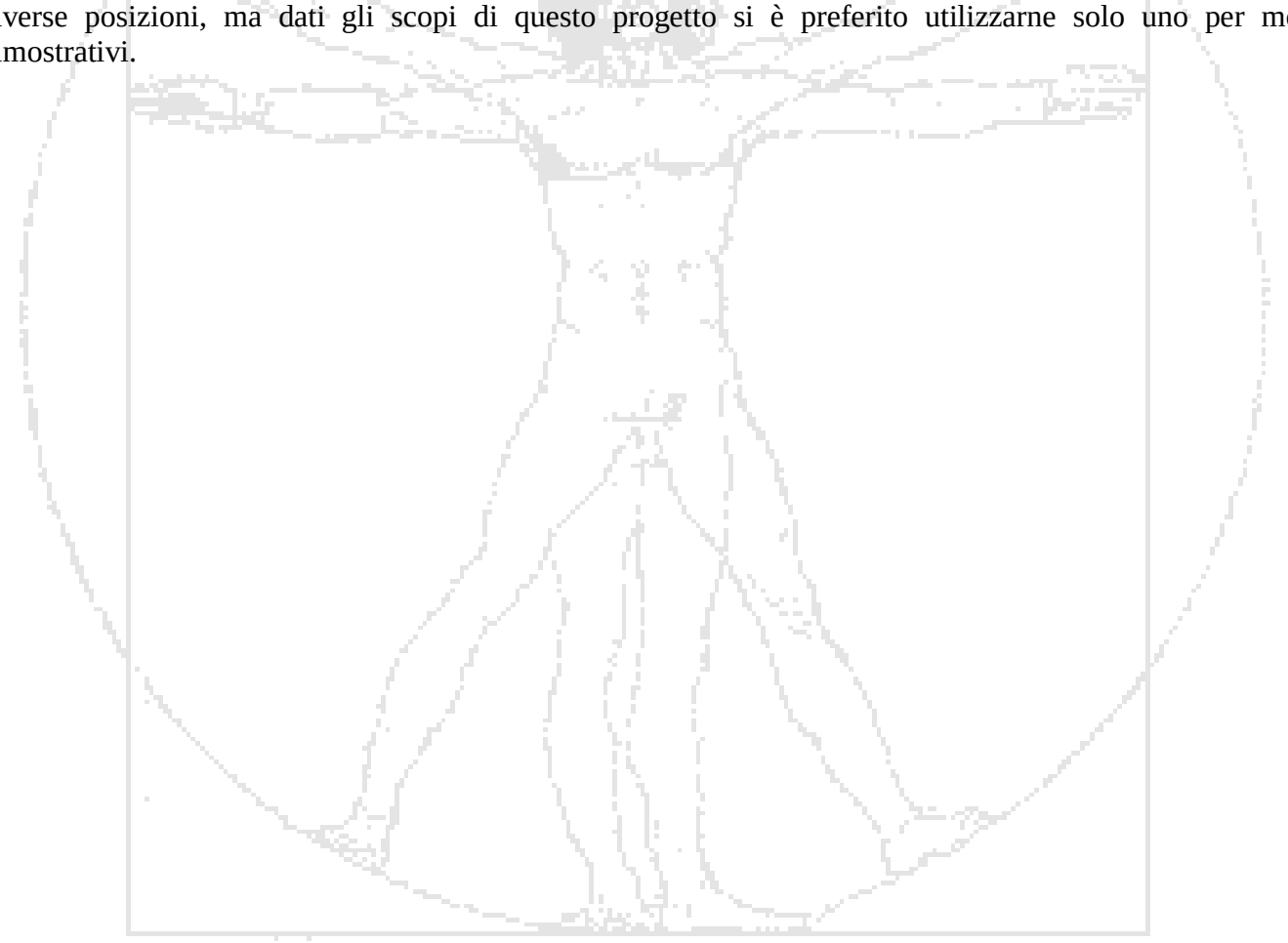
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace camion_pc_gui
{
publicpartialclassManual_mode_toolbar : Form
    {
        Form1 Form1;

public Manual_mode_toolbar(Form1 owner)
        {
this.Form1 = owner; // riferimento al form principale

            InitializeComponent();
        }

privatevoid Manual_mode_toolbar_Load(object sender, EventArgs e)
        {
this.ControlBox = false;
        }

privatevoid pinze_close_btn_Click(object sender, EventArgs e)
        {
this.Form1.WifiCmd("mpc");
        }

privatevoid pinze_open_btn_Click(object sender, EventArgs e)
        {
this.Form1.WifiCmd("mpa");
        }

privatevoid braccio_up_btn_Click(object sender, EventArgs e)
        {
this.Form1.WifiCmd("mbs");
        }

privatevoid braccio_down_btn_Click(object sender, EventArgs e)
        {
this.Form1.WifiCmd("mbg");
        }

privatevoid close_man_mode_Click(object sender, EventArgs e)
        {
            Form1.Manual_mode_win.Close();
}
    }
}
```

4.5.5 Interfaccia remota di configurazione modulo Wi-Fi

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Net;
using System.Net.NetworkInformation;
using System.Net.Sockets;
using System.IO;

namespace camion_pc_gui
{
publicpartialclassWiFi_Config : Form
    {
public WiFi_Config()
        {
            InitializeComponent();
        }

privatevoid WiFi_Config_Load(object sender, EventArgs e)
        {
string user = "admin";
string pass = "admin";
string hostname = "HF-A11";

            Ping pingSender = new Ping();
string result = "notFound";
try
            {
                PingReply reply = pingSender.Send(hostname, 120,
Encoding.ASCII.GetBytes("aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"), new PingOptions());
if (reply.Status == IPStatus.Success) result = "station";
            }
catch (Exception exc) { }


if (result == "station")
            {
                wbr1.Navigate("http://" + user + ":" + pass + "@" + hostname +"/");
            }
else
            {
try
                {
                    PingReply reply = pingSender.Send("10.10.100.254", 120,
Encoding.ASCII.GetBytes("aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"), new PingOptions());
if (reply.Status == IPStatus.Success) result = "apmode";
                }
catch (Exception exc) { }
if (result == "apmode")
                {
                    wbr1.Navigate("http://" + user + ":" + pass + "@" + "10.10.100.254/");
}
            }

if (result == "notFound")
            {
                MessageBox.Show("Verificare che il PC sia connesso alla rete del dispositivo.",
"Impossibile trovare il dispositivo!", MessageBoxButtons.OK, MessageBoxIcon.Error);
this.Close();
            }
        }
    }
}
```

5. COLLAUDO GENERALE E PROBLEMI RISCONTRATI

Dopo aver testato ogni blocco separatamente e aver realizzato i circuiti stampati si è passati al montaggio sul telaio del veicolo e al collaudo generale.
Come prima prova si è verificato che il veicolo seguisse la pista costruita. Durante questa prova si è notato che durante i rettilinei il veicolo tendeva a oscillare da una parte all'altra della pista. Per ovviare a questo problema si sono attuate modifiche al software in modo da diminuire l'effetto di oscillazione. Per motivi di tempo e materiale non si è riusciti però a risolvere completamente questo difetto.
Durante la seconda fase si è passati alla prova del braccio meccanico. Si è notato che a causa del primo problema il veicolo non riusciva a fermarsi sempre nella stessa posizione. Bisognava quindi in caso di necessità aggiustare la posizione del bidone in relazione a quella del veicolo. Per migliorare la fermata si sono predisposti ai lati della pista coppie di lamierini dentro ai quali le ruote anteriori si andranno a bloccare.
A fermata avvenuta si è testata la movimentazione del braccio meccanico con ripetute prove di sollevamento di bidoni. Durante questa fase si è convenuto che i bidoni dovessero essere leggermente rialzati grazie ad alcuni sostegni in modo da facilitare la presa da parte delle pinze. Questi sostegni dovevano inoltre presentare un incavo per l'alloggiamento del bidone. Non è stato rilevato nessun problema dal blocco del riconoscimento del colore.
Durante l'ultima fase del collaudo generale si è verificato il funzionamento dei sensori ad ultrasuoni e si è constatato che il veicolo riesce a fermarsi prima di urtare un ostacolo purché questo si trovi ad una minima altezza di 10 cmda terra. Per migliorare questo rilevamento si possono disporre altri sensori ad ultrasuoni in diverse posizioni, ma dati gli scopi di questo progetto si è preferito utilizzarne solo uno per motivi dimostrativi.



6. MANUALE D'USO

Si presenta in questa sezione il manuale d'uso di AGV differenziata.

6.1 PRIMO UTILIZZO

Al primo utilizzo è necessario impostare il modulo Wi-Fi seguendo i seguenti step:

- Assicurarsi della carica della batteria. In caso contrario occorre ricaricarla tramite apposito jack di alimentazione.
- Accendere il veicolo tramite pulsante di accensione posto a fianco del jack di alimentazione.
- Agire sul pulsante di reset presente nella cabina, tenendolo premuto per almeno 1 secondo. In questo modo il modulo viene riportato alle configurazioni di fabbrica ovvero in funzionamento accesspoint.
- Connettere il computer al modulo Wi-Fi (SSID: HF-A11_AP).
- Avviare l'interfaccia di controllo remoto.
- Premere sul pulsante *Impostazioni* nella toolbar e poi su *Configura accesso Wi-Fi*. Si aprirà una finestra dalla quale si può:
    - Impostare la modalità di lavoro (accesspoint o station mode).
    - Impostare i parametri relativi alla modalità accesspoint (in particolare l'indirizzo IP).
    - Impostare i parametri relativi alla modalità station mode (in particolare l'SSID della rete a cui connettersi e il tipo di metodo di autenticazione utilizzato con eventuali parametri).
    - Impostare il numero di porta di livello applicazione
- Uscire da questa finestra e nel form principale impostare negli appositi box l'indirizzo IP e il numero di porta da utilizzare per la comunicazione col veicolo. Di default in modalitàaccesspoint l'indirizzo IP è 10.10.100.254 e il numero di porta è 8899.
- Si procede poi al normale utilizzo del veicolo.

6.2 NORMALE UTILIZZO

1. Verificare il livello di carica di batteria (se questo non è stato fatto precedentemente) e in caso di necessità ricaricarla tramite l'apposito jack di alimentazione.
2. Prima di procedere con l'accensione tramite l'apposito pulsante è necessario assicurarsi che il veicolo sia posizionato sulla pista in modo tale che in fase di accensione il microcontrollore possa procedere alla calibrazione del sistema di lettura della pista.
3. Una volta posizionata si può accendere la macchina.Terminata la calibrazione il veicolo si porrà in attesa della connessione del software di gestione remoto.
4. L'utente dovrà poi aprire il software di controllo sul pc e premere il pulsante *connetti*.
5. Occorre poiimpostare il numero di postazioni presenti sul circuito di raccolta e successivamente quale ciclo di raccolta si vuole iniziare. Il numero di postazioni viene salvato perciò lo si dovrà modificare solo in caso di variazioni apportate al circuito di raccolta.
6. Dopo aver premuto il pulsante s*tart* il veicolo provvederà ad iniziare il ciclo di raccolta dei bidoni.
7. Nel caso si volesse fermare il veicolo occorre premere il pulsante *stop*. Si può riprendere la fase di lavoro premendo ancora il pulsante *start*. Se si vuole annullare la fase di lavoro attualeoccorre premere il pulsate *reset*. In questo caso il veicolo dovrà essere riportato alla posizione iniziale tramite il pulsante *modalità manuale*. Occorre premere questo pulsante ogni qualvolta si voglia utilizzare il veicolo in questa modalità.
8. Terminato il ciclo di raccolta la macchina ritornerà di nuovo al punto di partenza. Si dovrà ora controllare lo stato di carica tramite software e se la carica risulta insufficiente per un nuovo ciclo si dovrà:
    a. Disconnettere la macchina dal pc tramite il pulsante *disconnetti* del form principale.
    b. Spegnere la macchina premendo il pulsante generale.
    c. Collegare il jack di alimentazione.
    d. Riprendere a leggere dal punto 2.

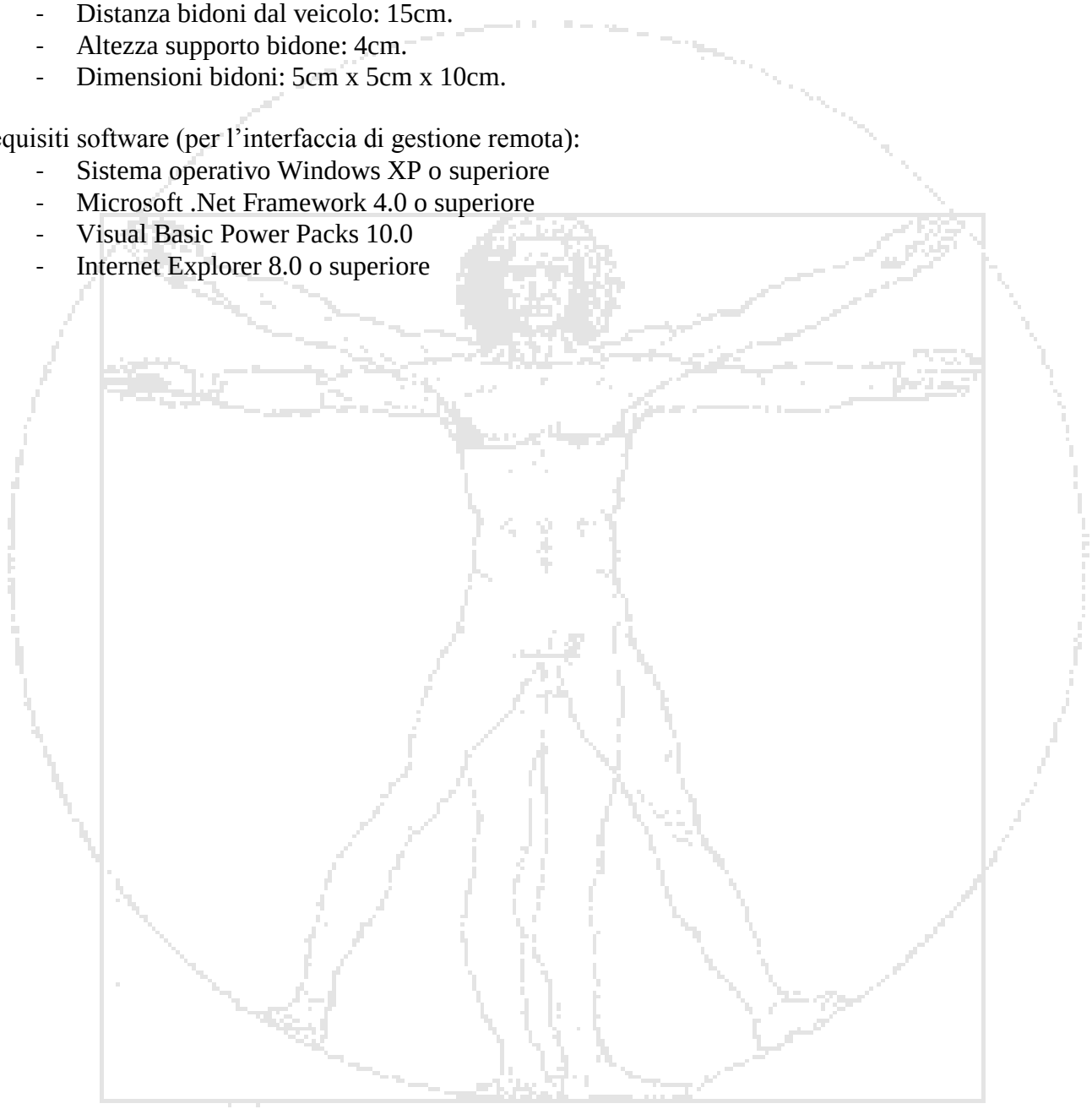
7. CONDIZIONI DI UTILIZZO

Prima di utilizzare questo veicolo occorre verificare che si rispettino alcune condizioni che ne garantiranno il corretto funzionamento:

- Larghezza pista: 4cm.
- Massima curva: 170cm di diametro.
- Colore pista (modificabile tramite software): nera o bianca.
- Colore bidoni: giallo, verde e blu.
- Altezza minima di rilevamento ostacoli: 10cm.
- Distanza dal centro della pista minimo 25 cm per lato.
- Distanza bidoni dal veicolo: 15cm.
- Altezza supporto bidone: 4cm.
- Dimensioni bidoni: 5cm x 5cm x 10cm.

Requisiti software (per l'interfaccia di gestione remota):

- Sistema operativo Windows XP o superiore
- Microsoft .Net Framework 4.0 o superiore
- Visual Basic Power Packs 10.0
- Internet Explorer 8.0 o superiore

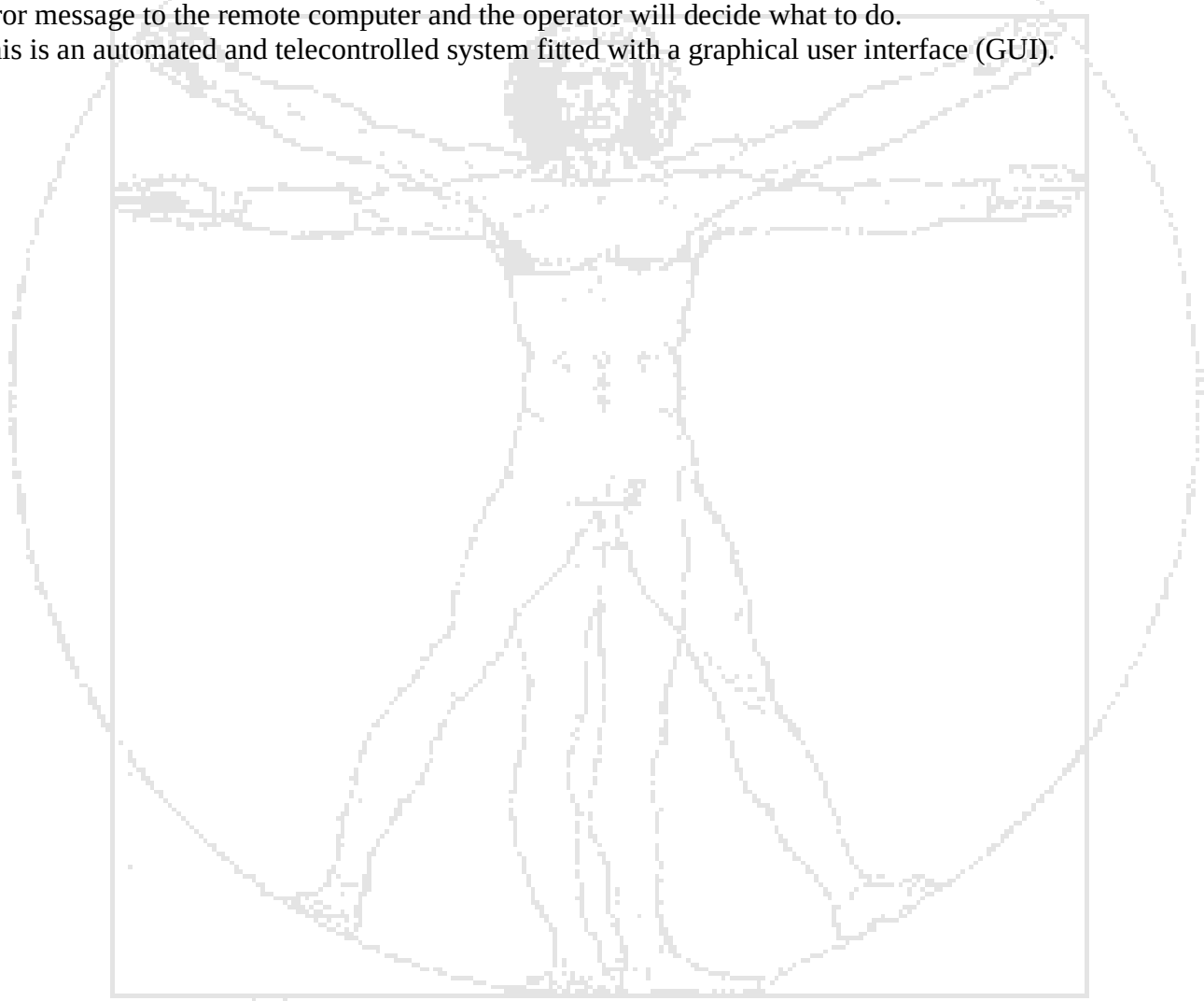


AGV differenziata

English version

E1. GENERAL DESCRIPTION

The project consists in a prototype of an automated guided vehicle (AGV) equipped with the necessary instruments for the collection of waste. It can be used in factories or infrastructures with wide environments but it can also be used in urban waste collection.
A person is required only for the supervision once it has been programmed. This supervision is done through a remote computer connected to the vehicle via Wi-Fi link.
In our project the vehicle follows a set route, marked with a black track, on the white floor. By the proper changes we can invert the colors in case we have a black (or a dark) floor.
By an opportune sign on the route (a perpendicular white strip) the vehicle is able to stop in front of the bin. After that it lowers its arm down and recognizes the color of the bin. If the color is the desired one the vehicle raises its arm and releases the trash into the box. When it finishes the lap it returns to the start position.
Instant by instant the vehicle communicates its present state to the remote computer. Moreover by this connection the user will order the vehicle what material it have to pick up every lap.
To avoid the vehicle to be damaged when it encounters an obstacle it has been equipped with an ultrasonic sensor in its front part. By this sensor the vehicle is able to stop avoiding collisions. After the stop it sends an error message to the remote computer and the operator will decide what to do.
This is an automated and telecontrolled system fitted with a graphical user interface (GUI).



E2. DETAILED DESCRIPTION

The project consists of many macro blocks:

- a central control system;
- track recognition system;
- motion block (which consists in a motor with a differential gear and a steering system);
- arm control system (which includes a color recognition circuit);
- safety devices system
- battery level control system;
- Wi-Fi module
- a remote GUI (Graphical User Interface) to control the vehicle from a remote computer.

E2.1 The central control system

The central unit of the vehicle is a microcontroller: Arduino Mega 2560.
Its main features are:

- 54 digital input/output pins including:
  - 15 pins compatible with PWM modulation
  - 8 pins (4 TX, 4 RX) able to communicate over 4 TTL serial ports
- 40mA max current support on each pin
- 16 analogue inputs (with integrated 10 bit A/D converter)
- 256 KB flash memory
- 16 MHz clock processor
- USB to TTL serial module for programming and communicating with a PC
- Supply voltage range:
  - from 7V to 12V
  - 5V with USB power supply

E2.2 Track recognition system

This module is made to read the black track on the floor that the vehicle has to follow to determine its route. It is made of two photoresistors with a white high intensity led positioned between them. When the vehicle is moving forward the photoresistors are both on the black track. When the track has a bend, one of the photoresistors gets out of the track and the vehicle acts on the steering to rectify its direction according to the track.
Photoresistors are characterized by a variation in their resistance according to external light intensity. By a voltage divider, the resistance variation is converted into a voltage one. This is read by the microcontroller and it is converted into a digital number thanks to the integrated A/D converter.
The corrective actions are made by the software which reads the digitalized voltage values at a relatively high frequency and according to the differences between the voltage values of the two photoresistor outputs, it acts on the steering servomotor.
If both the photoresistors read white color the software knows that the vehicle is close to a collection station. So the vehicle is stopped and the bin color recognition procedure is started.

E2.3 Motion block

The first part consists in a DC motor connected to the rear-wheels drive by a differential gear that permits to distribute the torque. This motor is driven by the integrated circuit L298. Thanks to this integrated circuit Arduino can determine both the direction and the speed of the motor. This last one is controlled by Arduino's PWM module.
The second part consists in a servomotor directly controlled by Arduino. The servomotor is attached to the front wheels. By changing the angle of the servomotor it changes the steering angle.

E2.4 Arm control system

The arm is made of three parts: the lever, the tilt and the pliers. The lever is driven by a DC gear motor while the tilt and the pliers are driven by two servomotors. Arduino is able to raise and to lower the arm by controlling the direction and the speed of the DC gear motor. The integrated circuit L298 permits Arduino to interface with the motor. The two servomotors are directly controlled by Arduino.

In the bottom front part of the arm there is the color recognition circuit which consists in 3 LEDs (red, blue and green) and 3 phototransistors. By switching on every LED and measuring the voltage across the phototransistor, corresponding to every LED, the system is able to determine if the color is the desired one.

E2.5 Safety devices system

This block consists in an ultrasonic sensor, a buzzer and some LEDs. In the front part of the vehicle there is an ultrasonic transmitter and an ultrasonic sensor. The presence of an obstacle on the road causes the reflection of the ultrasonic waves so Arduino, through the ultrasonic sensor, can notice it and stop the vehicle. Buzzer and LEDs are activated when the arm is moving.

E2.6 Battery level control system

This system consists in a voltage divider. Through the integrated A/D converter the microcontroller is able to check the battery level and send it to the remote computer via Wi-Fi.

E2.7 Wi-Fi module

For the communication between the microcontroller and the remote computer is used a serial to Wi-Fi module USR-WIFI232-B. When the computer sends data by opening a socket to the Wi-Fi module it sends the received bytes to the serial interface. When Arduino responds the received serial data are sent to the remote computer via Wi-Fi.
This module can work in two modes:

- *Access Point Mode*: the device creates a Wi-Fi net which host fitted with Wi-Fi interface can access.
- *Station Mode*: the devices connect to an existing Wi-Fi net, therefore it can be reached by every host connected to this net.

E2.8Remote GUI

The graphical interface is written in C#. It permits to:

- Manage the Wi-Fi module settings
- Open and close the communication with the vehicle
- Select the number of stations
- Select the desired collection cycle
- Start and stop the vehicle
- Monitor the current state of the system

E3. POSSIBLE APPLICATIONS

This vehicle could be used in industries or in wide space environments to simplify the collection of waste. In fact this vehicle needs only one person for programming and supervising. All the other processes necessary to carry out the work phase are managed automatically.

E4. PROBLEMS FACED

The main problems we met are:

- On the straights the vehicle oscillates because its steering is controlled by a servo motor that has a long response time. Because of time issues we cannot solve this problem completely but we improved the steer by software.
- The discrimination of the track to follow: this is caused by external light and the non-linearity of the photoresistors. This is mainly solved by software.
- The vehicle has a maximum steering angle and so the bends must not be too close.
- The building of the arm.
- The precision of the stop of the vehicle. This is solved thanks to some metal sheets fixed to the floor on the track sides.
- The discrimination of the color of the bins: this is caused by the external light.
- The management of the ultrasonic transmitter.

E5. INDIVIDUAL TASKS

E5.1 Bruschi Davide

My tasks were:

- Creating the schematic and the PCB for every module that we have. It wasn’ttoo difficult to design the schematic. The first problem occurred when we had all the schematics projects done and we have to physically made the PCB. The PCB maker machine at school broke two times his cutter. To solve this problem the PCBs were homemade with another process called photogravure. In this way all the PCBs have been done.
- Designing and building the robotic arm that picks up the binsand empties them into the box of the truck. This was another big problem. We had to design all the arm because there wasn't anything of precasthaving the dimensions that we would. I started to project it and to create some prototypes. I think that Ibuiltabout 4 or 5 different prototypes but all the prototypes had the same problem: Ihad a big difficult to do parts that must have a tolerance of 0.5mm or at least 1mm. This was a very big problem because we didn't know anyone that could do it with the laser technique. I tried another time to do it but this time I used a table that had some guide for the jigsaw and so I cut the aluminum, that I used to make the arm. I done all the parts and I finally assembled them together. After that I added to the arm the color recognition circuit.
- Makingall the wiring of the truck and test it to control that it works fine. This has been a long process but not too difficult because I used wires of different colors so I couldn't make mistakes.
- Building the cabin of the truck. The cabin is made of wood. Here are fixed the microcontroller, the circuit of the motors, the circuit for the control of the safety devices, the Wi-Fi module with his circuits, the buzzer and the ultrasonic sensor.
- Assembling all the parts together and control if there were some problems. Luckily when I finished to do all the wirings and I switched on the system allworked with any problem.

E5.2 Rocchi Nicholas

In this project I mainly occupied of:

- Designing the circuits: I designed myself the circuit of the photoresistors, the one for the color recognition and the one of the safety devices. I also designed with Simonazzi the Wi-Fi module interface system and with Bruschi the one to control the motors. During this task I met a bit difficult sizing the components like resistors and transistors. After the designing and the realization of the printed circuit boards we tested them. We didn’t use complicated circuits so the tests weren’t very difficult too.
- Writing the Arduino software. I wrote the first Arduino software myself but in the other versions the entire group worked on this software because it was very difficult to write and everyone could improve it. This software is the heart of the project and it’s composed of functions that control every part of the vehicle.
- Realizing the track. We used black cardboard to realize the track. After some tests at home we decided to make the track at school, in a electronics laboratory. So I designed the track to adapt it to the environment. It was not difficult to design but it was difficult to choose some specifics like the dimensions of the track and the bending angles.
- Realizing the bins. I made this task with Simonazzi. We realized 9 plastic bins. We cut the plastic material and realized the bins with the use of hot glue. After that three bins were painted of green, three of blue and three of yellow. We made 9 bins because we decided to do three collection stations. Each station has a green, a blue and a yellow bin.
- Realizing the box. I cut the wooden planks for the construction of the box. After that I painted them. Once they were painted Bruschi assembled them to build the box.

E5.3 Simonazzi Marco

In the development of the project I principally handled two aspects: the Wi-Fi communication software and the remote management interface realization.

The Wi-Fi communication is realized by opening a TCP/IP socket from the remote interface to the vehicle’s Wi-Fi module through its IP address and the set application protocol port.

The remote management software is written in Visual C# code. It consists of three forms (windows):

- The main control form
- The manual mode form (with its toolbar)
- The Wi-Fi configuration form

When software is launched in its main entry point it is loaded the **main window**. Through this form the user can open the connection with the remote vehicle by inserting the vehicle's Wi-Fi module IP address and application port and by pressing the *Connect* button.
After the connection has established the application starts polling (every 250 milliseconds) the vehicle by asking it its current state. If the state has changed the application displays it in the main form status bar.
Thanks to this polling the operator can also monitor in real time the battery level.
The user is now able to set up all the parameters required to carry out the collection cycle:

- the kind of the bin (paper, plastic or glass),
- the number of collection station on the route.

Then he can hit the S*tart* button and the vehicle starts working. During the collection cycle the user can stop the vehicle by pressing the *Stop* button and, if something goes wrong, it can reset it by pressing the *Reset* button. While the machine is working detailed information are displayed in the status bar of the main window and some additional one are displayed via graphical animation reproduced in a white rectangle in the left bottom corner of the window. Here are displayed animations related to the arm movements and the presence of obstacles on the road.
If during its work the vehicle goes out of the black route the operator can manually control it by pressing the *Manual mode* button. When the button is activated the vehicle is automatically stopped and the **manual mode window** is shown.
This window is equipped with a red ball drawn at the center of the form. The user can control the steering, the speed and the direction of the vehicle by dragging it, such as a joystick. When the mouse is released the ball returns in the middle and the vehicle is stopped.
When the manual mode form is open it is also shown a tool bar with some buttons that allows the user to manually control the arm operation such as:

- opening tongs,
- closing tongs,
- rising the arm,
- lowering the arm.

When the manual mode form is closed the vehicle is still stopping. The user has to manually press the start button to resume the collection cycle.
From the main form the operator can also access the Wi-Fi configuration interface by pressing the *Settings* button and then the *Wi-Fi configuration* one.
Through this last window the operator can set:

- the Wi-Fi operation mode (Access Point or Station mode)
- set the Wi-Fi module IP address setting (static IP or dynamic IP)
- set the TCP/IP application port (default 8899)

SOMMARIO

Ultrasonic Ranging Module HC - SR04

Product features:

Ultrasonic ranging module HC - SR04 provides 2cm - 400cm non-contact measurement function, the ranging accuracy can reach to 3mm. The modules includes ultrasonic transmitters, receiver and control circuit. The basic principle of work:
(1) Using IO trigger for at least 10us high level signal,
(2) The Module automatically sends eight 40 kHz and detect whether there is a pulse signal back.
(3) IF the signal back, through high level , time of high output IO duration is the time from sending ultrasonic to returning.
Test distance = (high level time×velocity of sound (340M/S) / 2,

Wire connecting direct as following:

- 5V Supply
- Trigger Pulse Input
- Echo Pulse Output
- 0V Ground

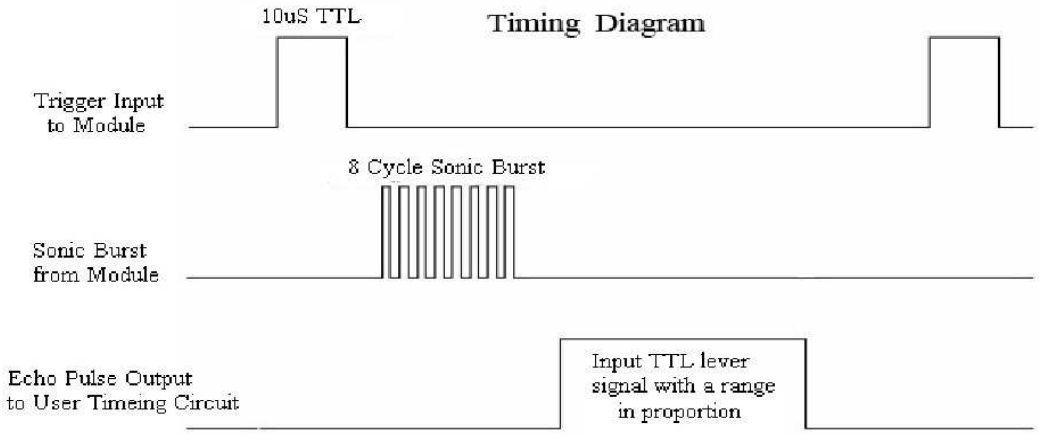
Electric Parameter

Working Voltage	**DC 5 V**
Working Current	**15mA**
Working Frequency	**40Hz**
Max Range	**4m**
Min Range	**2cm**
MeasuringAngle	**15 degree**
Trigger Input Signal	**10uS TTL pulse**
Echo Output Signal	**Input TTL lever signal and the range in proportion**
Dimension	**45*20*15mm**

Vcc Trig Echo GND

Timing diagram

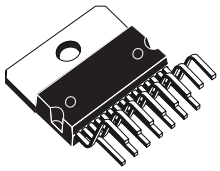
The Timing diagram is shown below. You only need to supply a short 10uS pulse to the trigger input to start the ranging, and then the module will send out an 8 cycle burst of ultrasound at 40 kHz and raise its echo. The Echo is a distance object that is pulse width and the range in proportion .You can calculate the range through the time interval between sending trigger signal and receiving echo signal. Formula: uS / 58 = centimeters or uS / 148 =inch; or: the range = high level time * velocity (340M/S) / 2; we suggest to use over 60ms measurement cycle, in order to prevent trigger signal to the echo signal.



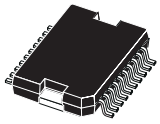
L298

DUAL FULL-BRIDGE DRIVER

- OPERATING SUPPLY VOLTAGE UP TO 46 V
- TOTAL DC CURRENT UP TO 4 A
- LOW SATURATION VOLTAGE
- OVERTEMPERATURE PROTECTION
- LOGICAL "0" INPUT VOLTAGE UP TO 1.5 V (HIGH NOISE IMMUNITY)



Multiwatt15



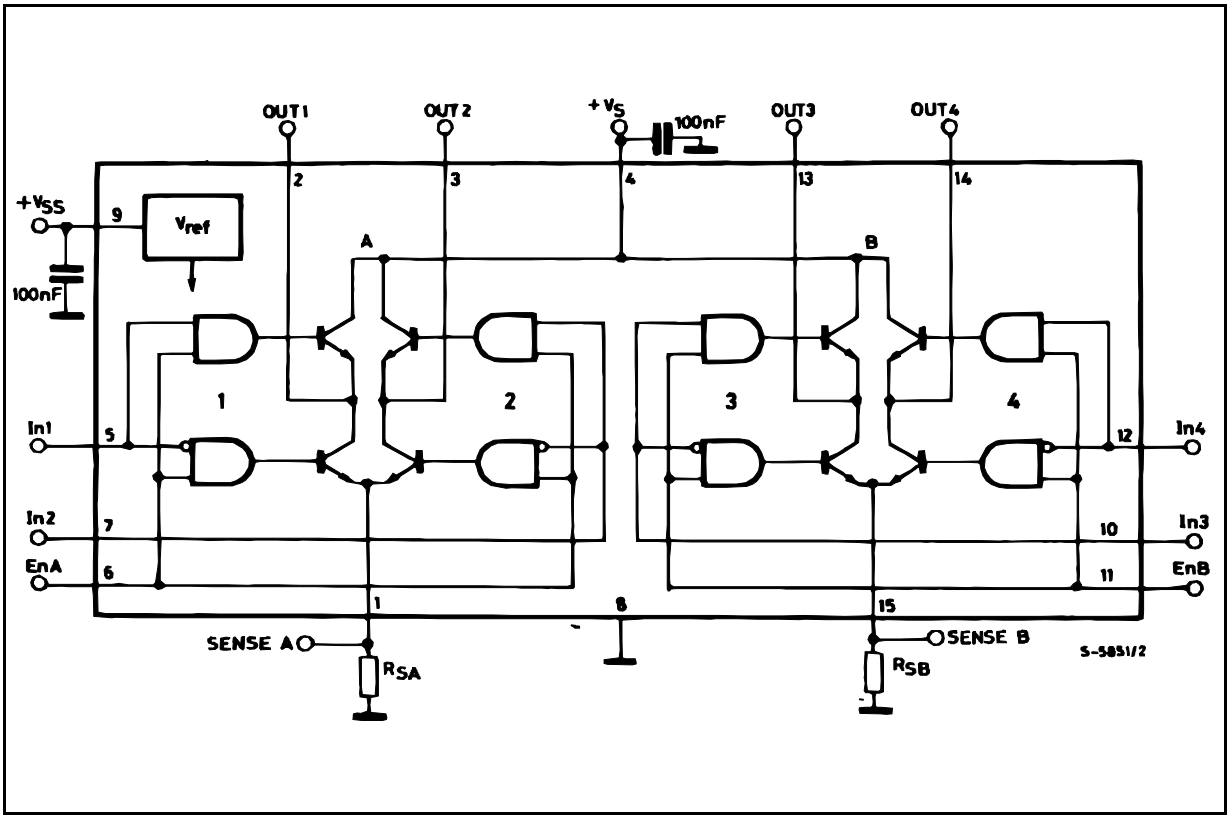
PowerSO20

ORDERING NUMBERS : L298N (Multiwatt Vert.)
L298HN (Multiwatt Horiz.)
L298P (PowerSO20)

DESCRIPTION

The L298 is an integrated monolithic circuit in a 15-lead Multiwatt and PowerSO20 packages. It is a high voltage, high current dual full-bridge driver designed to accept standard TTL logic levels and drive inductive loads such as relays, solenoids, DC and stepping motors. Two enable inputs are provided to enable or disable the device independently of the input signals. The emitters of the lower transistors of each bridge are connected together and the corresponding external terminal can be used for the connection of an external sensing resistor. An additional supply input is provided so that the logic works at a lower voltage.

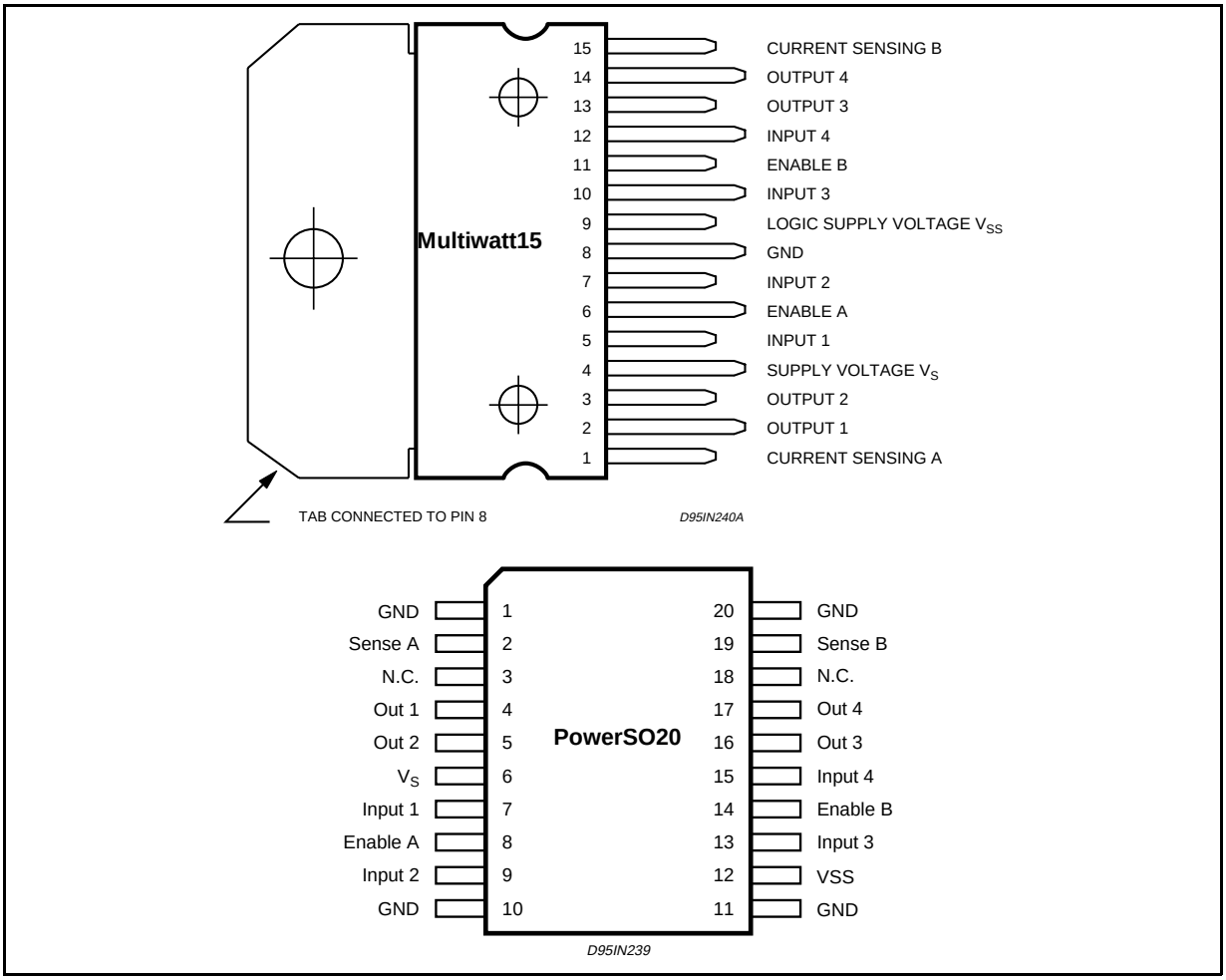
BLOCK DIAGRAM



ABSOLUTE MAXIMUM RATINGS

Symbol	Parameter	Value	Unit
V_S	Power Supply	50	V
V_{SS}	Logic Supply Voltage	7	V
V_I,V_{en}	Input and Enable Voltage	–0.3 to 7	V
I_O	Peak Output Current (each Channel) – Non Repetitive (t = 100µs) –Repetitive (80% on –20% off; t_{on} = 10ms) –DC Operation	 3 2.5 2	 A A A
V_{sens}	Sensing Voltage	–1 to 2.3	V
P_{tot}	Total Power Dissipation (T_{case} = 75°C)	25	W
T_{op}	Junction Operating Temperature	–25 to 130	°C
T_{stg}, T_j	Storage and Junction Temperature	–40 to 150	°C

PIN CONNECTIONS (top view)



THERMAL DATA

Symbol	Parameter		PowerSO20	Multiwatt15	Unit
$R_{th\ j\text{-}case}$	Thermal Resistance Junction-case	Max.	–	3	°C/W
$R_{th\ j\text{-}amb}$	Thermal Resistance Junction-ambient	Max.	13 (*)	35	°C/W

(*) Mounted on aluminum substrate

PIN FUNCTIONS (refer to the block diagram)

MW.15	PowerSO	Name	Function
1;15	2;19	Sense A; Sense B	Between this pin and ground is connected the sense resistor to control the current of the load.
2;3	4;5	Out 1; Out 2	Outputs of the Bridge A; the current that flows through the load connected between these two pins is monitored at pin 1.
4	6	V_S	Supply Voltage for the Power Output Stages. A non-inductive 100nF capacitor must be connected between this pin and ground.
5;7	7;9	Input 1; Input 2	TTL Compatible Inputs of the Bridge A.
6;11	8;14	Enable A; Enable B	TTL Compatible Enable Input: the L state disables the bridge A (enable A) and/or the bridge B (enable B).
8	1,10,11,20	GND	Ground.
9	12	VSS	Supply Voltage for the Logic Blocks. A100nF capacitor must be connected between this pin and ground.
10; 12	13;15	Input 3; Input 4	TTL Compatible Inputs of the Bridge B.
13; 14	16;17	Out 3; Out 4	Outputs of the Bridge B. The current that flows through the load connected between these two pins is monitored at pin 15.
–	3;18	N.C.	Not Connected

ELECTRICAL CHARACTERISTICS (V_S = 42V; V_{SS} = 5V, T_j = 25°C; unless otherwise specified)

Symbol	Parameter	Test Conditions	Min.	Typ.	Max.	Unit
V_S	Supply Voltage (pin 4)	Operative Condition	V_{IH} +2.5		46	V
V_{SS}	Logic Supply Voltage (pin 9)		4.5	5	7	V
I_S	Quiescent Supply Current (pin 4)	V_{en} = H; I_L = 0 V_i = L V_i = H		13 50	22 70	mA mA
		V_{en} = L V_i = X			4	mA
I_{SS}	Quiescent Current from V_{SS} (pin 9)	V_{en} = H; I_L = 0 V_i = L V_i = H		24 7	36 12	mA mA
		V_{en} = L V_i = X			6	mA
V_{iL}	Input Low Voltage (pins 5, 7, 10, 12)		–0.3		1.5	V
V_{iH}	Input High Voltage (pins 5, 7, 10, 12)		2.3		VSS	V
I_{iL}	Low Voltage Input Current (pins 5, 7, 10, 12)	V_i = L			–10	µA
I_{iH}	High Voltage Input Current (pins 5, 7, 10, 12)	Vi = H ≤ V_{SS} –0.6V		30	100	µA
V_{en} = L	Enable Low Voltage (pins 6, 11)		–0.3		1.5	V
V_{en} = H	Enable High Voltage (pins 6, 11)		2.3		V_{SS}	V
I_{en} = L	Low Voltage Enable Current (pins 6, 11)	V_{en} = L			–10	µA
I_{en} = H	High Voltage Enable Current (pins 6, 11)	V_{en} = H ≤ V_{SS} –0.6V		30	100	µA
$V_{CEsat\ (H)}$	Source Saturation Voltage	I_L = 1A I_L = 2A	0.95	1.35 2	1.7 2.7	V V
$V_{CEsat\ (L)}$	Sink Saturation Voltage	I_L = 1A (5) I_L = 2A (5)	0.85	1.2 1.7	1.6 2.3	V V
V_{CEsat}	Total Drop	I_L = 1A (5) I_L = 2A (5)	1.80		3.2 4.9	V V
V_{sens}	Sensing Voltage (pins 1, 15)		–1 (1)		2	V

Figure 5 : Sink Current Delay Times vs. Input 0 V Enable Switching.

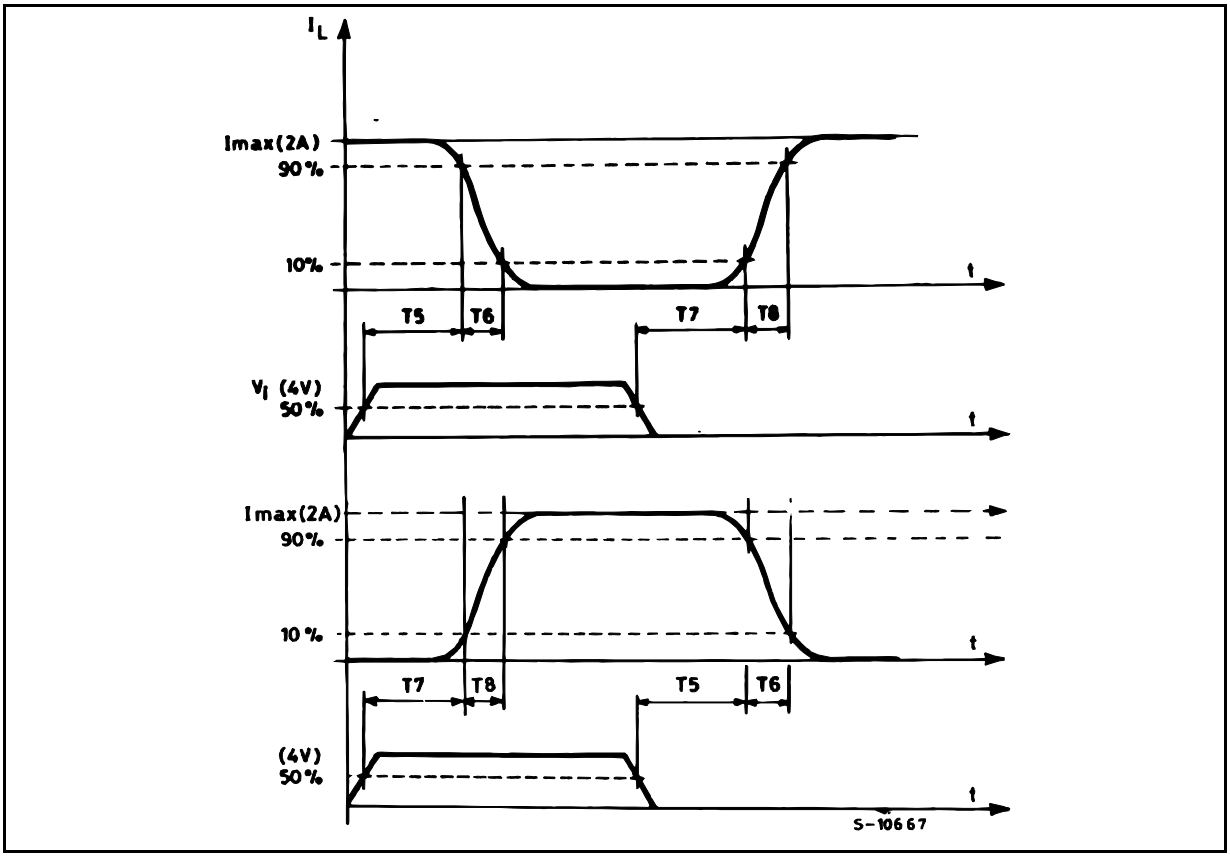
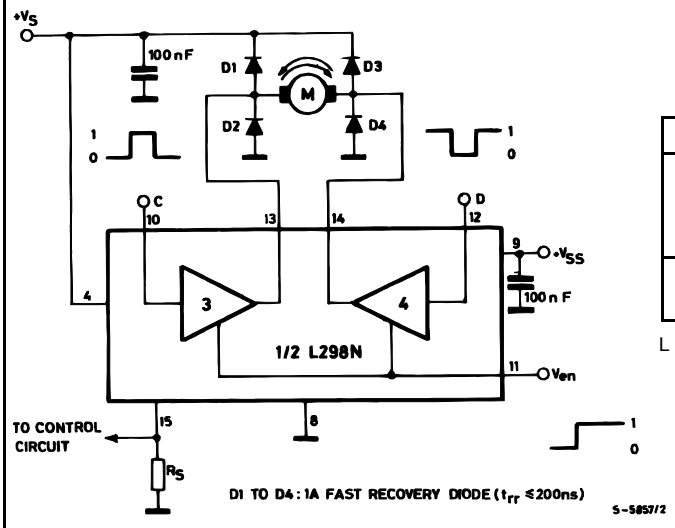


Figure 6 : Bidirectional DC Motor Control.



Inputs		Function
V_{en} = H	C = H ; D = L	Forward
	C = L ; D = H	Reverse
	C = D	Fast Motor Stop
V_{en} = L	C = X ; D = X	Free Running Motor Stop

L = Low H = High X = Don't care

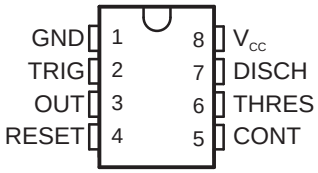
PRECISION TIMERS

Check for Samples: NA555, NE555, SA555, SE555

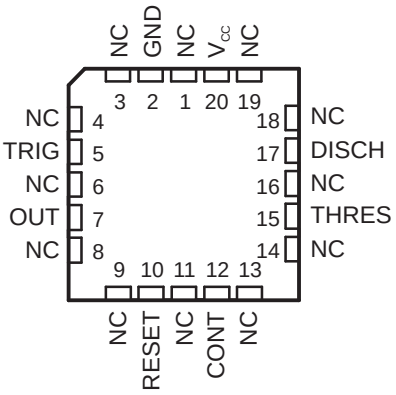
FEATURES

- **Timing From Microseconds to Hours**
- **Astable or Monostable Operation**
- **Adjustable Duty Cycle**
- **TTL-Compatible Output Can Sink or Source up to 200 mA**

NA555...D OR P PACKAGE
NE555...D, P, PS, OR PW PACKAGE
SA555...D OR P PACKAGE
SE555...D, JG, OR P PACKAGE
(TOP VIEW)



SE555...FK PACKAGE
(TOP VIEW)



NC – No internal connection

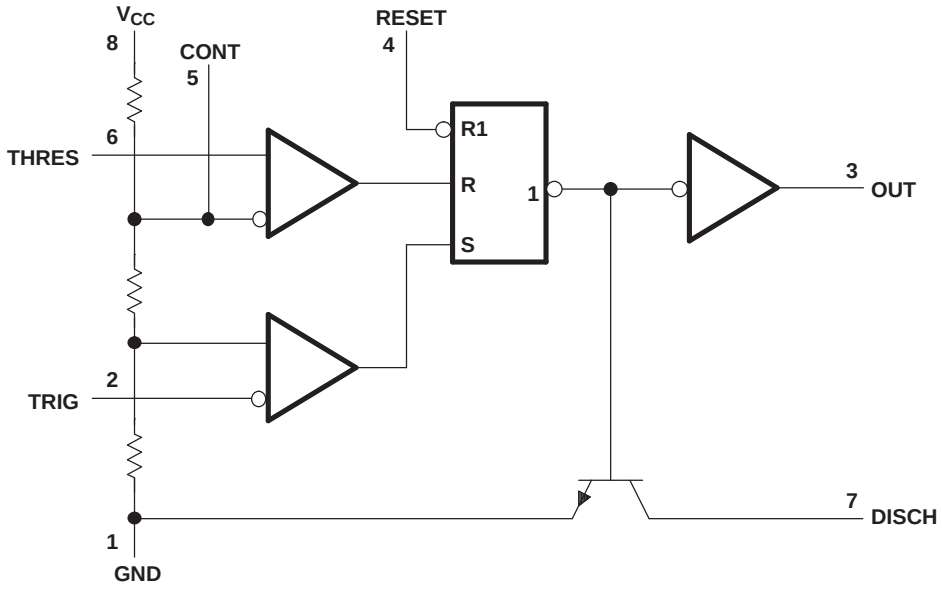
DESCRIPTION/ORDERING INFORMATION

These devices are precision timing circuits capable of producing accurate time delays or oscillation. In the time-delay or monostable mode of operation, the timed interval is controlled by a single external resistor and capacitor network. In the astable mode of operation, the frequency and duty cycle can be controlled independently with two external resistors and a single external capacitor.

The threshold and trigger levels normally are two-thirds and one-third, respectively, of V_{CC}. These levels can be altered by use of the control-voltage terminal. When the trigger input falls below the trigger level, the flip-flop is set, and the output goes high. If the trigger input is above the trigger level and the threshold input is above the threshold level, the flip-flop is reset and the output is low. The reset (RESET) input can override all other inputs and can be used to initiate a new timing cycle. When RESET goes low, the flip-flop is reset, and the output goes low. When the output is low, a low-impedance path is provided between discharge (DISCH) and ground.

The output circuit is capable of sinking or sourcing current up to 200 mA. Operation is specified for supplies of 5 V to 15 V. With a 5-V supply, output levels are compatible with TTL inputs.

FUNCTIONAL BLOCK DIAGRAM



A. Pin numbers shown are for the D, JG, P, PS, and PW packages.

B. RESET can override TRIG, which can override THRES.

Absolute Maximum Ratings(1)

over operating free-air temperature range (unless otherwise noted)

			MIN	MAX	UNIT
V_{CC}	Supply voltage(2)			18	V
V_I	Input voltage	CONT, RESET, THRES, TRIG		V_{CC}	V
I_O	Output current			±225	mA
θ_{JA}	Package thermal impedance(3) (4)	D package		97	°C/W
		P package		85	
		PS package		95	
		PW package		149	
θ_{JC}	Package thermal impedance(5) (6)	FK package		5.61	°C/W
		JG package		14.5	
T_J	Operating virtual junction temperature			150	°C
	Case temperature for 60 s	FK package		260	°C
	Lead temperature 1, 6 mm (1/16 in) from case for 60 s	JG package		300	°C
T_{stg}	Storage temperature range		–65	150	°C

(1) Stresses beyond those listed under "absolute maximum ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated under "recommended operating conditions" is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.
(2) All voltage values are with respect to GND.
(3) Maximum power dissipation is a function of $T_J(max)$, θ_{JA}, and T_A. The maximum allowable power dissipation at any allowable ambient temperature is $P_D = (T_J(max) - T_A)/\theta_{JA}$. Operating at the absolute maximum T_J of 150°C can affect reliability.
(4) The package thermal impedance is calculated in accordance with JESD 51-7.
(5) Maximum power dissipation is a function of $T_J(max)$, θ_{JC}, and T_C. The maximum allowable power dissipation at any allowable case temperature is $P_D = (T_J(max) - T_C)/\theta_{JC}$. Operating at the absolute maximum T_J of 150°C can affect reliability.
(6) The package thermal impedance is calculated in accordance with MIL-STD-883.

Recommended Operating Conditions

over operating free-air temperature range (unless otherwise noted)

			MIN	MAX	UNIT
V_{CC}	Supply voltage	NA555, NE555, SA555	4.5	16	V
		SE555	4.5	18	
V_I	Input voltage	CONT, RESET, THRES, and TRIG		V_{CC}	V
I_O	Output current			±200	mA
T_A	Operating free-air temperature	NA555	–40	105	°C
		NE555	0	70	
		SA555	–40	85	
		SE555	–55	125	

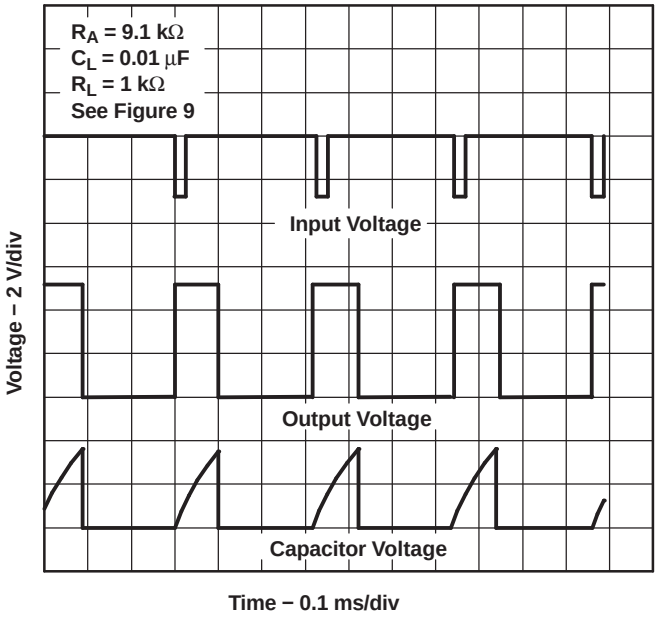


Figure 10. Typical Monostable Waveforms

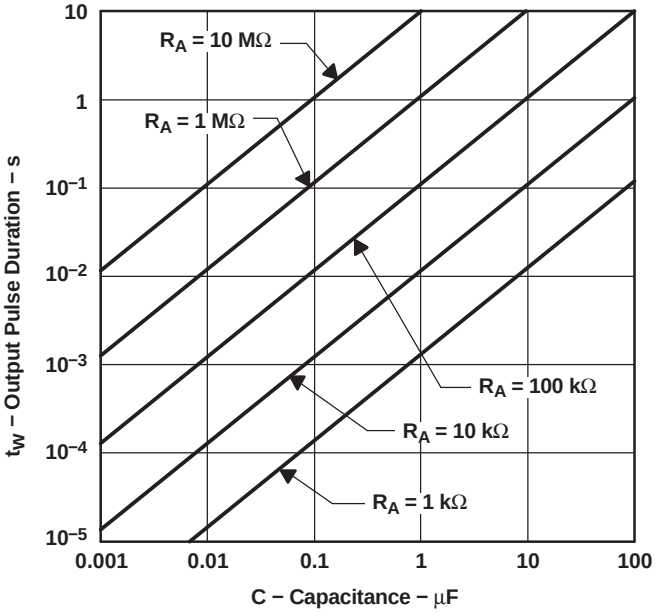
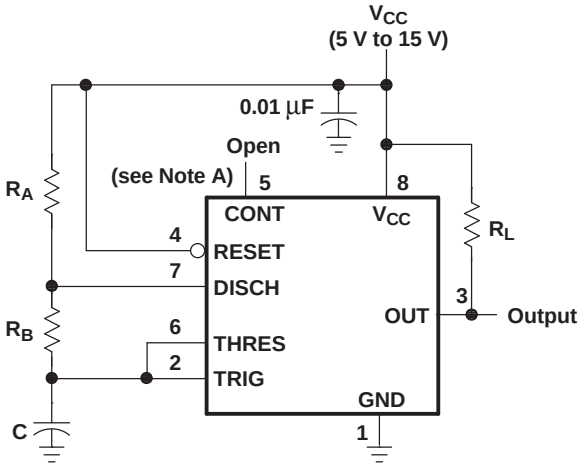


Figure 11. Output Pulse Duration vs Capacitance

Astable Operation

As shown in Figure 12, adding a second resistor, R_B, to the circuit of Figure 9 and connecting the trigger input to the threshold input causes the timer to self-trigger and run as a multivibrator. The capacitor C charges through R_A and R_B and then discharges through R_B only. Therefore, the duty cycle is controlled by the values of R_A and R_B.

This astable connection results in capacitor C charging and discharging between the threshold-voltage level (≠0.67 × V_{CC}) and the trigger-voltage level (≠0.33 × V_{CC}). As in the monostable circuit, charge and discharge times (and, therefore, the frequency and duty cycle) are independent of the supply voltage.



Pin numbers shown are for the D, JG, P, PS, and PW packages.

NOTE A: Decoupling CONT voltage to ground with a capacitor can improve operation. This should be evaluated for individual applications.

Figure 12. Circuit for Astable Operation

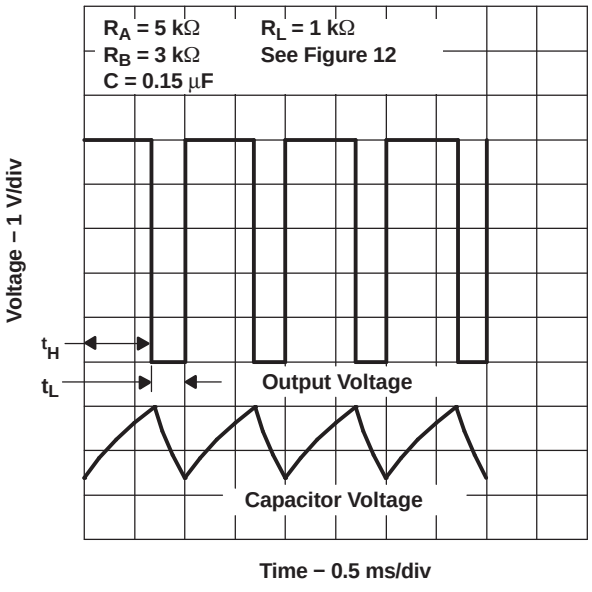


Figure 13. Typical Astable Waveforms

Figure 12 shows typical waveforms generated during astable operation. The output high-level duration t_H and low-level duration t_L can be calculated as follows:

$$t_H = 0.693 (R_A + R_B) C$$

$$t_L = 0.693 (R_B) C$$

Other useful relationships are shown below.

$$\text{period} = t_H + t_L = 0.693 (R_A + 2R_B) C$$

$$\text{frequency} \approx \frac{1.44}{(R_A + 2R_B) C}$$

$$\text{Output driver duty cycle} = \frac{t_L}{t_H + t_L} = \frac{R_B}{R_A + 2R_B}$$

Output waveform duty cycle

$$= \frac{t_H}{t_H + t_L} = 1 - \frac{R_B}{R_A + 2R_B}$$

$$\text{Low-to-high ratio} = \frac{t_L}{t_H} = \frac{R_B}{R_A + R_B}$$

Figure .

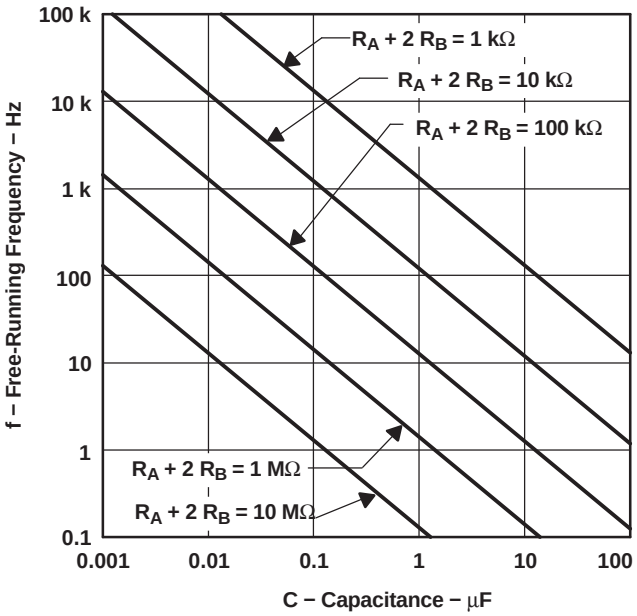
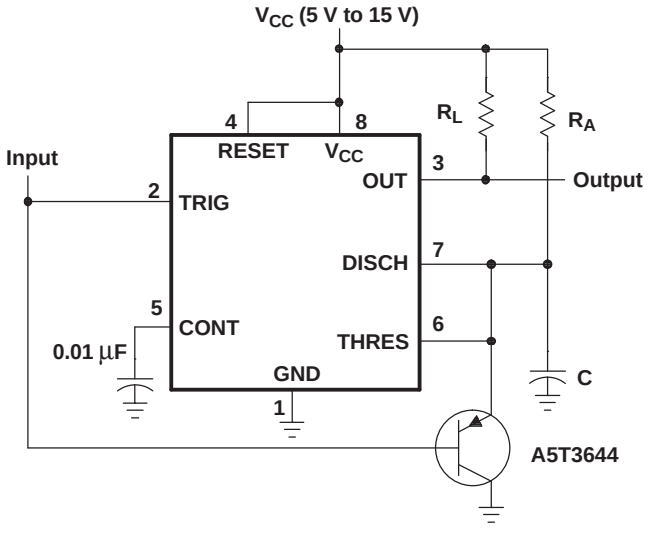


Figure 14. Free-Running Frequency

Missing-Pulse Detector

The circuit shown in Figure 15 can be used to detect a missing pulse or abnormally long spacing between consecutive pulses in a train of pulses. The timing interval of the monostable circuit is retriggered continuously by the input pulse train as long as the pulse spacing is less than the timing interval. A longer pulse spacing, missing pulse, or terminated pulse train permits the timing interval to be completed, thereby generating an output pulse as shown in Figure 16.



Pin numbers shown are shown for the D, JG, P, PS, and PW packages.

Figure 15. Circuit for Missing-Pulse Detector

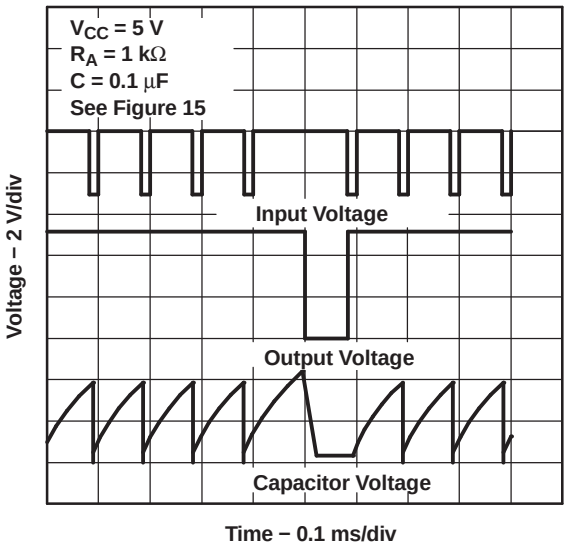


Figure 16. Completed Timing Waveforms for Missing-Pulse Detector

- **Package Options Include Plastic "Small Outline" Packages, Ceramic Chip Carriers and Flat Packages, and Plastic and Ceramic DIPs**
- **Dependable Texas Instruments Quality and Reliability**

description

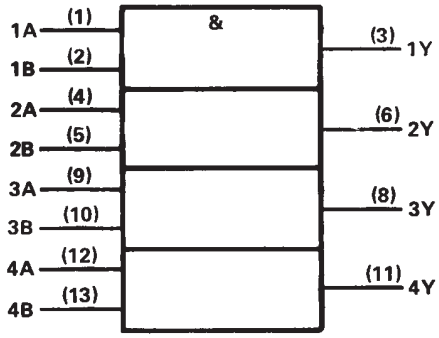
These devices contain four independent 2-input AND gates.

The SN5408, SN54LS08, and SN54S08 are characterized for operation over the full military temperature range of −55°C to 125°C. The SN7408, SN74LS08 and SN74S08 are characterized for operation from 0° to 70°C.

FUNCTION TABLE (each gate)

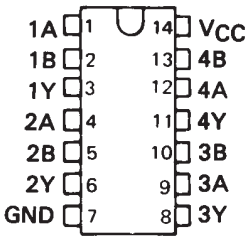
INPUTS		OUTPUT
A	B	Y
H	H	H
L	X	L
X	L	L

logic symbol†

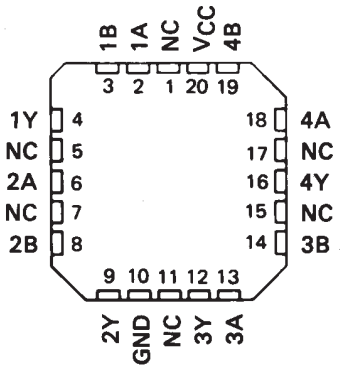


† This symbol is in accordance with ANSI/IEEE Std 91-1984 and IEC Publication 617-12.
Pin numbers shown are for D, J, N, and W packages.

SN5408, SN54LS08, SN54S08 . . . J OR W PACKAGE
SN7408 . . . J OR N PACKAGE
SN74LS08, SN74S08 . . . D, J OR N PACKAGE
(TOP VIEW)

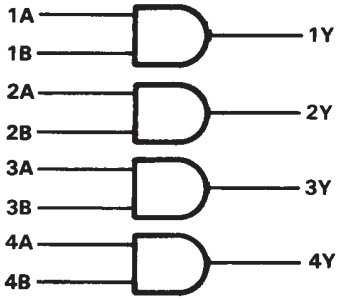


SN54LS08, SN54S08 . . . FK PACKAGE
(TOP VIEW)



NC—No internal connection

logic diagram (positive logic)



$Y = A \cdot B$ or $Y = \overline{\overline{A} + \overline{B}}$

recommended operating conditions

	SN5408			SN7408			UNIT
	MIN	NOM	MAX	MIN	NOM	MAX	
V_{CC} Supply voltage	4.5	5	5.5	4.75	5	5.25	V
V_{IH} High-level input voltage	2			2			V
V_{IL} Low-level input voltage			0.8			0.8	V
I_{OH} High-level output current			−0.8			−0.8	mA
I_{OL} Low-level output current			16			16	mA
T_A Operating free-air temperature	−55		125	0		70	°C

electrical characteristics over recommended operating free-air temperature range (unless otherwise noted)

PARAMETER	TEST CONDITIONS †	SN5408			SN7408			UNIT
		MIN	TYP‡	MAX	MIN	TYP‡	MAX	
V_{IK}	V_{CC} = MIN, I_I = −12 mA			−1.5			−1.5	V
V_{OH}	V_{CC} = MIN, V_{IH} = 2 V, I_{OH} = −0.8 mA	2.4	3.4		2.4	3.4		V
V_{OL}	V_{CC} = MIN, V_{IL} = 0.8 V, I_{OL} = 16 mA		0.2	0.4		0.2	0.4	V
I_I	V_{CC} = MAX, V_I = 5.5 V			1			1	mA
I_{IH}	V_{CC} = MAX, V_I = 2.4 V			40			40	μA
I_{IL}	V_{CC} = MAX, V_I = 0.4 V			−1.6			−1.6	mA
I_{OS}§	V_{CC} = MAX	−20		−55	−18		−55	mA
I_{CCH}	V_{CC} = MAX, V_I = 4.5 V		11	21		11	21	mA
I_{CCL}	V_{CC} = MAX, V_I = 0 V		20	33		20	33	mA

† For conditions shown as MIN or MAX, use the appropriate value specified under recommended operating conditions.
‡ All typical values are at V_{CC} = 5 V, T_A = 25°C.
§ Not more than one output should be shorted at a time.

switching characteristics, V_{CC} = 5 V, T_A = 25°C (see note 2)

PARAMETER	FROM (INPUT)	TO (OUTPUT)	TEST CONDITIONS	MIN	TYP	MAX	UNIT
t_{PLH}	A or B	Y	R_L = 400 Ω, C_L = 15 pF		17.5	27	ns
t_{PHL}					12	19	ns

NOTE 2: Load circuits and voltage waveforms are shown in Section 1.

MOTOROLA

LM317

Three-Terminal Adjustable Output Positive Voltage Regulator

The LM317 is an adjustable 3–terminal positive voltage regulator capable of supplying in excess of 1.5 A over an output voltage range of 1.2 V to 37 V. This voltage regulator is exceptionally easy to use and requires only two external resistors to set the output voltage. Further, it employs internal current limiting, thermal shutdown and safe area compensation, making it essentially blow–out proof.

The LM317 serves a wide variety of applications including local, on card regulation. This device can also be used to make a programmable output regulator, or by connecting a fixed resistor between the adjustment and output, the LM317 can be used as a precision current regulator.

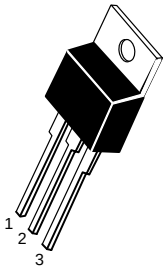
- Output Current in Excess of 1.5 A
- Output Adjustable between 1.2 V and 37 V
- Internal Thermal Overload Protection
- Internal Short Circuit Current Limiting Constant with Temperature
- Output Transistor Safe–Area Compensation
- Floating Operation for High Voltage Applications
- Available in Surface Mount D^2PAK, and Standard 3–Lead Transistor Package
- Eliminates Stocking many Fixed Voltages

THREE–TERMINAL ADJUSTABLE POSITIVE VOLTAGE REGULATOR

SEMICONDUCTOR TECHNICAL DATA

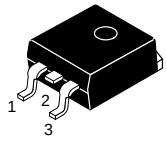
T SUFFIX
PLASTIC PACKAGE
CASE 221A

Heatsink surface connected to Pin 2.



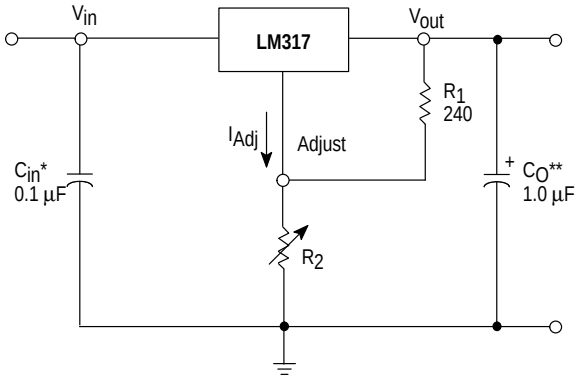
Pin 1. Adjust
2. V_{out}
3. V_{in}

D2T SUFFIX
PLASTIC PACKAGE
CASE 936
(D^2PAK)



Heatsink surface (shown as terminal 4 in case outline drawing) is connected to Pin 2.

Standard Application



* C_{in} is required if regulator is located an appreciable distance from power supply filter.
** C_O is not needed for stability, however, it does improve transient response.

$$V_{out} = 1.25\ V \left(1 + \frac{R_2}{R_1}\right) + I_{Adj}\ R_2$$

Since I_{Adj} is controlled to less than 100 µA, the error associated with this term is negligible in most applications.

ORDERING INFORMATION

Device	Operating Temperature Range	Package
LM317BD2T	T_J = –40° to +125°C	Surface Mount
LM317BT		Insertion Mount
LM317D2T	T_J = 0° to +125°C	Surface Mount
LM317T		Insertion Mount

MAXIMUM RATINGS

Rating	Symbol	Value	Unit
Input–Output Voltage Differential	$V_I–V_O$	40	Vdc
Power Dissipation			
Case 221A			
T_A = +25°C	P_D	Internally Limited	W
Thermal Resistance, Junction–to–Ambient	θ_{JA}	65	°C/W
Thermal Resistance, Junction–to–Case	θ_{JC}	5.0	°C/W
Case 936 (D^2PAK)			
T_A = +25°C	P_D	Internally Limited	W
Thermal Resistance, Junction–to–Ambient	θ_{JA}	70	°C/W
Thermal Resistance, Junction–to–Case	θ_{JC}	5.0	°C/W
Operating Junction Temperature Range	T_J	–40 to +125	°C
Storage Temperature Range	T_{stg}	–65 to +150	°C

ELECTRICAL CHARACTERISTICS ($V_I–V_O$ = 5.0 V; I_O = 0.5 A for D2T and T packages; $T_J = T_{low}$ to T_{high} [Note 1]; I_{max} and P_{max} [Note 2]; unless otherwise noted.)

Characteristics	Figure	Symbol	Min	Typ	Max	Unit
Line Regulation (Note 3), T_A = +25°C, 3.0 V ≤ $V_I–V_O$ ≤ 40 V	1	Reg_{line}	–	0.01	0.04	%/V
Load Regulation (Note 3), T_A = +25°C, 10 mA ≤ I_O ≤ I_{max}	2	Reg_{load}				
V_O ≤ 5.0 V			–	5.0	25	mV
V_O ≥ 5.0 V			–	0.1	0.5	% V_O
Thermal Regulation, T_A = +25°C (Note 6), 20 ms Pulse		Reg_{therm}	–	0.03	0.07	% V_O/W
Adjustment Pin Current	3	I_{Adj}	–	50	100	μA
Adjustment Pin Current Change, 2.5 V ≤ $V_I–V_O$ ≤ 40 V, 10 mA ≤ I_L ≤ I_{max}, P_D ≤ P_{max}	1, 2	ΔI_{Adj}	–	0.2	5.0	μA
Reference Voltage, 3.0 V ≤ $V_I–V_O$ ≤ 40 V, 10 mA ≤ I_O ≤ I_{max}, P_D ≤ P_{max}	3	V_{ref}	1.2	1.25	1.3	V
Line Regulation (Note 3), 3.0 V ≤ $V_I–V_O$ ≤ 40 V	1	Reg_{line}	–	0.02	0.07	% V
Load Regulation (Note 3), 10 mA ≤ I_O ≤ I_{max}	2	Reg_{load}				
V_O ≤ 5.0 V			–	20	70	mV
V_O ≥ 5.0 V			–	0.3	1.5	% V_O
Temperature Stability (T_{low} ≤ T_J ≤ T_{high})	3	T_S	–	0.7	–	% V_O
Minimum Load Current to Maintain Regulation ($V_I–V_O$ = 40 V)	3	I_{Lmin}	–	3.5	10	mA
Maximum Output Current	3	I_{max}				A
$V_I–V_O$ ≤ 15 V, P_D ≤ P_{max}, T Package			1.5	2.2	–	
$V_I–V_O$ = 40 V, P_D ≤ P_{max}, T_A = +25°C, T Package			0.15	0.4	–	
RMS Noise, % of V_O, T_A = +25°C, 10 Hz ≤ f ≤ 10 kHz		N	–	0.003	–	% V_O
Ripple Rejection, V_O = 10 V, f = 120 Hz (Note 4)	4	RR				dB
Without C_{Adj}			–	65	–	
C_{Adj} = 10 μF			66	80	–	
Long–Term Stability, $T_J = T_{high}$ (Note 5), T_A = +25°C for Endpoint Measurements	3	S	–	0.3	1.0	%/1.0 k Hrs.
Thermal Resistance Junction to Case, T Package		$R_{\theta JC}$	–	5.0	–	°C/W

NOTES: 1. T_{low} to T_{high} = 0° to +125°C, for LM317T, D2T. T_{low} to T_{high} = –40° to +125°C, for LM317BT, BD2T.
2. I_{max} = 1.5 A, P_{max} = 20 W
3. Load and line regulation are specified at constant junction temperature. Changes in V_O due to heating effects must be taken into account separately. Pulse testing with low duty cycle is used.
4. C_{Adj}, when used, is connected between the adjustment pin and ground.
5. Since Long–Term Stability cannot be measured on each device before shipment, this specification is an engineering estimate of average stability from lot to lot.
6. Power dissipation within an IC voltage regulator produces a temperature gradient on the die, affecting individual IC components on the die. These effects can be minimized by proper integrated circuit design and layout techniques. Thermal Regulation is the effect of these temperature gradients on the output voltage and is expressed in percentage of output change per watt of power change in a specified time.

APPLICATIONS INFORMATION

Basic Circuit Operation

The LM317 is a 3–terminal floating regulator. In operation, the LM317 develops and maintains a nominal 1.25 V reference (V_{ref}) between its output and adjustment terminals. This reference voltage is converted to a programming current (I_{PROG}) by R_1 (see Figure 17), and this constant current flows through R_2 to ground.

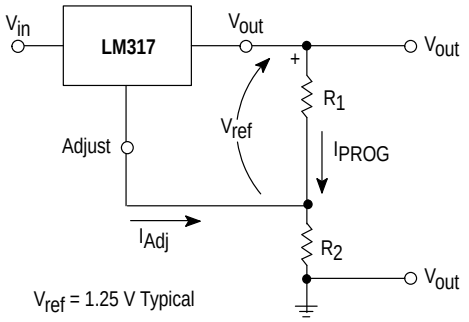
The regulated output voltage is given by:

$$V_{out} = V_{ref}\left(1 + \frac{R_2}{R_1}\right) + I_{Adj}\,R_2$$

Since the current from the adjustment terminal (I_{Adj}) represents an error term in the equation, the LM317 was designed to control I_{Adj} to less than 100 μA and keep it constant. To do this, all quiescent operating current is returned to the output terminal. This imposes the requirement for a minimum load current. If the load current is less than this minimum, the output voltage will rise.

Since the LM317 is a floating regulator, it is only the voltage differential across the circuit which is important to performance, and operation at high voltages with respect to ground is possible.

Figure 17. Basic Circuit Configuration



Load Regulation

The LM317 is capable of providing extremely good load regulation, but a few precautions are needed to obtain maximum performance. For best performance, the programming resistor (R_1) should be connected as close to the regulator as possible to minimize line drops which effectively appear in series with the reference, thereby degrading regulation. The ground end of R_2 can be returned near the load ground to provide remote ground sensing and improve load regulation.

External Capacitors

A 0.1 μF disc or 1.0 μF tantalum input bypass capacitor (C_{in}) is recommended to reduce the sensitivity to input line impedance.

The adjustment terminal may be bypassed to ground to improve ripple rejection. This capacitor (C_{Adj}) prevents ripple from being amplified as the output voltage is increased. A 10 μF capacitor should improve ripple rejection about 15 dB at 120 Hz in a 10 V application.

Although the LM317 is stable with no output capacitance, like any feedback circuit, certain values of external capacitance can cause excessive ringing. An output capacitance (C_O) in the form of a 1.0 μF tantalum or 25 μF aluminum electrolytic capacitor on the output swamps this effect and insures stability.

Protection Diodes

When external capacitors are used with any IC regulator it is sometimes necessary to add protection diodes to prevent the capacitors from discharging through low current points into the regulator.

Figure 18 shows the LM317 with the recommended protection diodes for output voltages in excess of 25 V or high capacitance values (C_O > 25 μF, C_{Adj} > 10 μF). Diode D_1 prevents C_O from discharging thru the IC during an input short circuit. Diode D_2 protects against capacitor C_{Adj} discharging through the IC during an output short circuit. The combination of diodes D_1 and D_2 prevents C_{Adj} from discharging through the IC during an input short circuit.

Figure 18. Voltage Regulator with Protection Diodes

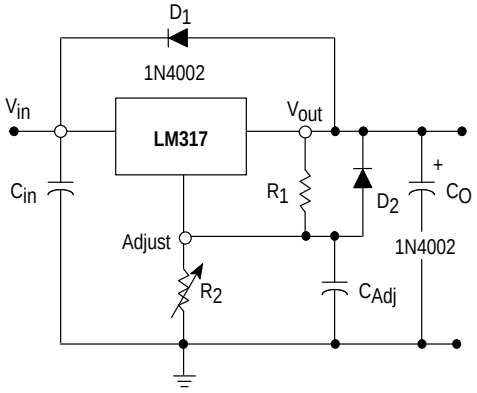
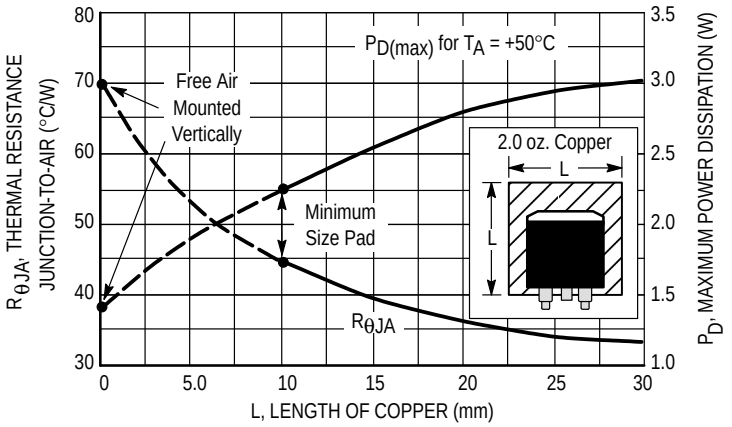


Figure 19. D²PAK Thermal Resistance and Maximum Power Dissipation versus P.C.B. Copper Length



FAIRCHILD SEMICONDUCTOR®

August 2013

LM78XX / LM78XXA
3-Terminal 1 A Positive Voltage Regulator

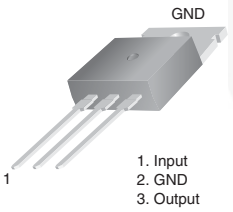
Features

- Output Current up to 1 A
- Output Voltages: 5, 6, 8, 9, 10, 12, 15, 18, 24 V
- Thermal Overload Protection
- Short-Circuit Protection
- Output Transistor Safe Operating Area Protection

Description

The LM78XX series of three-terminal positive regulators is available in the TO-220 package and with several fixed output voltages, making them useful in a wide range of applications. Each type employs internal current limiting, thermal shut-down, and safe operating area protection. If adequate heat sinking is provided, they can deliver over 1 A output current. Although designed primarily as fixed-voltage regulators, these devices can be used with external components for adjustable voltages and currents.

TO-220 (Single Gauge)



Ordering Information(1)

Product Number	Output Voltage Tolerance	Package	Operating Temperature	Packing Method
LM7805CT	±4%	TO-220 (Single Gauge)	-40°C to +125°C	Rail
LM7806CT				
LM7808CT				
LM7809CT				
LM7810CT				
LM7812CT				
LM7815CT				
LM7818CT				
LM7824CT				
LM7805ACT	±2%		0°C to +125°C	
LM7809ACT				
LM7810ACT				
LM7812ACT				
LM7815ACT				

Note:

1. Above output voltage tolerance is available at 25°C.

Electrical Characteristics (LM7805)

Refer to the test circuit, -40°C < T_J < 125°C, I_O = 500 mA, V_I = 10 V, C_I = 0.1 µF, unless otherwise specified.

Symbol	Parameter	Conditions		Min.	Typ.	Max.	Unit
V_O	Output Voltage	T_J = +25°C		4.80	5.00	5.20	V
		I_O = 5 mA to 1 A, $P_O \leq$ 15 W, V_I = 7 V to 20 V		4.75	5.00	5.25	
Regline	Line Regulation(2)	T_J = +25°C	V_I = 7 V to 25 V		4.0	100.0	mV
			V_I = 8 V to 12 V		1.6	50.0	
Regload	Load Regulation(2)	T_J = +25°C	I_O = 5 mA to 1.5 A		9.0	100.0	mV
			I_O = 250 mA to 750 mA		4.0	50.0	
I_Q	Quiescent Current	T_J =+25°C			5.0	8.0	mA
ΔI_Q	Quiescent Current Change	I_O = 5 mA to 1 A			0.03	0.50	mA
		V_I = 7 V to 25 V			0.30	1.30	
$\Delta V_O/\Delta T$	Output Voltage Drift(3)	I_O = 5 mA			-0.8		mV/°C
V_N	Output Noise Voltage	f = 10 Hz to 100 kHz, T_A = +25°C			42.0		µV/V_O
RR	Ripple Rejection(3)	f = 120 Hz, V_I = 8 V to 18 V		62.0	73.0		dB
V_{DROP}	Dropout Voltage	T_J = +25°C, I_O = 1 A			2.0		V
R_O	Output Resistance(3)	f = 1 kHz			15.0		mΩ
I_{SC}	Short-Circuit Current	T_J = +25°C, V_I = 35 V			230		mA
I_{PK}	Peak Current(3)	T_J = +25°C			2.2		A

Notes:

2. Load and line regulation are specified at constant junction temperature. Changes in V_O due to heating effects must be taken into account separately. Pulse testing with low duty is used.
3. These parameters, although guaranteed, are not 100% tested in production.

1. PRODUCT OVERVIEW

1.1. General Specification

Table 1 USR-WIFI232-X Module Technical Specifications

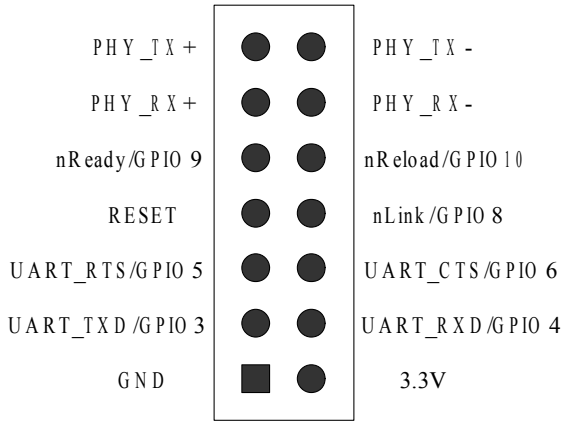
Class	Item	Parameters
Wireless Parameters	Certification	FCC/CE
	Wireless standard	802.11 b/g/n
	Frequency range	2.412GHz-2.484GHz
	Transmit Power	802.11b: +20 dBm (Max.)
		802.11g: +18 dBm (Max.)
		802.11n: +15 dBm (Max.)
		Configurable
	Receiver Sensitivity	802.11b: -89 dBm
		802.11g: -81dBm
		802.11n: -71dBm
	Antenna Option	External:I-PEX Connector
		Internal:On-board chip antenna
Hardware Parameters	Data Interface	UART: 1200bps - 230400bps
		GPIOs
		Ethernet: 100Mpbs
	Operating Voltage	3.3V (+/-5%)
	Operating Current	170mA~300mA
	Operating Temperature	-25℃- 85℃
	Storage Temperature	-40℃- 85℃
	Dimensions and Size	25×40×8mm
Software Parameters	Network Type	Station /AP mode
	Security Mechanisms	WEP/WAP-PSK/WAP2-PSK/WAPI
	Encryption	WEP64/WEP128/TKIP/AES
	Work Mode	Transparent Transmission and Agreement Transmission mode
	Serial command	AT+instruction set
	Network Protocol	TCP/UDP/ARP/ICMP/DHCP/DNS/HTTP
	Max. TCP Connection	32
	User Configuration	Web Server＋AT command config.
	User Application SW	Support customized application SW with Linux system.

1.2. Hardware Introduction

Figure 1.USR-WIFI232-X

1.2.1. Pins Definition

Pin type module Pins Map:



SMD type module Pins Map:

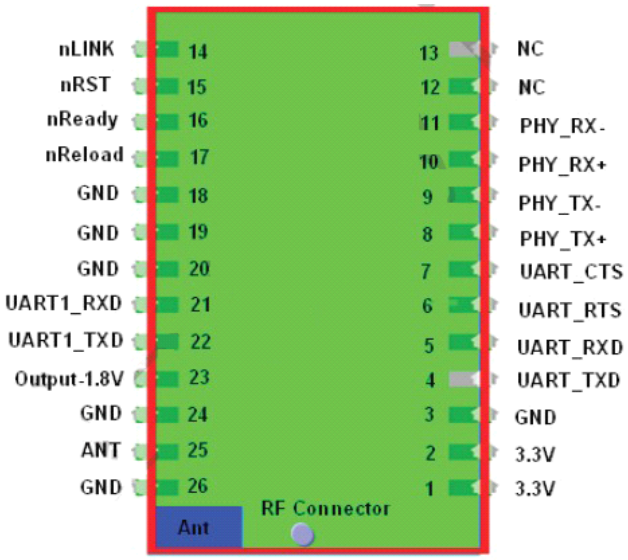


Figure 2 USR-WIFI232-X Pins Map

Table 2 USR-WIFI232-X Pins Definition

Pin type module Definition:

Pin	Description	Name	Direction	Note
1	Ground	GND	Power	
2	VCC	3.3V	Power	3.3V @ 350mA power input
3	UART Data Transmit	UART_TXD	O	If not use UART function，this 4 pins can be configured as GPIO pins, and can change GPIO pin status through AT command
	GPIO	GPIO3	I/O	
4	UART Data Receive	UART_RXD	I	
	GPIO	GPIO4	I/O	
5	UART sends request of data transmission	UART_RTS	O	
	GPIO	GPIO5	I/O	
6	UART receives data transmission permission	UART_CTS	I	
	GPIO	GPIO6	I/O	
7	Module reset signal	RESET	I	“Low (0)” effective reset input. The reset duration should be kept more than 300ms
8	WiFi status Indication	nLink	O	“1”- WIFI connection available, “0”- No WIFI connection Can be configured as GPIO.
	GPIO	GPIO8	I/O	
9	Indicate the module status of power on process	nReady	O	“0” or “Palmodic Signal” - Finish module boot up process; “1” - Module boot up not finish. Can be configured as GPIO.
	GPIO	GPIO9	I/O	
10	Restore configuration	nReload	I	Module will Restore factory default configuration after set this pin “0” more than 1s, then set “1”.
	GPIO	GPIO10	I/O	
11	Ethernet Interface	PHY_RX+	I	+1.8V Ethernet Data Interface Support transformer and direct connection (AC couple) mode.
12	Ethernet Interface	PHY_RX-	I	
13	Ethernet Interface	PHY_TX+	O	
14	Ethernet Interface	PHY_TX-	O	