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

Laboratorio di Robotica Avanzata
Advanced Robotics Laboratory

Corso di Robotica
(Prof. Riccardo Cassinis)

MARMOT-Mover 2

Elaborato di esame di: **Arrighini Stefano, Ferrari Thomas**

Consegnato il: **04 luglio 2002**

Sommario

Questo elaborato ha lo scopo di potenziare e migliorare le possibilità di movimento e di controllo del robot MARMOT.

Dal punto di vista dell'utente il robot viene controllato tramite la spedizione di semplici pacchetti lungo una linea seriale EIA/TIA-232; tali pacchetti vengono elaborati "a bordo" tramite una scheda a microcontrollore (della famiglia Motorola 68HC11), la quale provvede a generare i segnali di controllo per le schede driver dei motori.

I miglioramenti da noi introdotti consentiranno al robot di effettuare movimenti di tipo diverso nonché di aumentare la precisione del suo posizionamento. Sarà possibile effettuare movimenti sia a velocità costante che con rampe di accelerazione e di decelerazione; per quanto riguarda il posizionamento sarà possibile in ogni momento determinare lo spostamento effettuato dalle ruote del robot e sarà possibile richiedere un avviso dopo il compimento di un tragitto prefissato. L'utente avrà inoltre la possibilità di monitorare più semplicemente lo stato del robot.

1. Introduzione

Il nostro lavoro è partito dall'analisi della documentazione del team di sviluppo del robot MARMOT, attualmente in fase di sviluppo e integrazione presso il Laboratorio di Robotica Avanzata, per comprendere il sistema e capire quanto le restanti risorse del microcontrollore adottato fossero adatte alle nostre esigenze.

Durante l'elaborato ci siamo occupati di tutti gli aspetti progettuali e realizzativi coinvolti. In questo siamo stati coadiuvati dall'ing. Paolo Meriggi. Abbiamo inoltre collaborato con il tesista Alessandro Savalli per l'integrazione delle nostre funzionalità nel suo lavoro.

2. Il problema affrontato

La prima versione del programma per il robot [ROS_SIG01] permette di raggiungere una velocità impostata mediante rampe di accelerazione e decelerazione. Tali rampe vengono generate internamente al robot, esternamente risulta quindi difficile conoscerle con esattezza.

Questo, unito al non determinismo dei tempi di ricezione e gestione dei pacchetti da parte del robot, genera non pochi problemi per la stima della posizione del robot stesso. Infatti, non essendo per ora il robot dotato di sensori che permettano la misura della posizione rispetto a punti di riferimento, è necessario stimarla in base al cammino fatto (*dead reckoning*). La difficoltà di conoscere le rampe si traduce in una difficoltà di determinare lo spostamento fatto.

Il nostro compito è quindi quello di dotare il robot di funzionalità per posizionare il robot in modo esatto e per semplificare le operazioni di dead reckoning.

Attualmente il robot non è in grado di dare alcun'informazione sulle condizioni in cui si trova. Queste informazioni dovrebbero comprendere il tipo di operazione attualmente in esecuzione ed eventuali malfunzionamenti verificatisi. Inoltre la creazione di una seconda versione introduce il problema di sapere qual è quella correntemente installata.

Sarà nostro compito implementare anche funzionalità che permetteranno di sopperire a queste mancanze.

3. La soluzione adottata

Per posizionare il robot in modo preciso abbiamo, come già detto, due problemi: uno legato alla difficoltà di ricostruire le rampe e uno legato al non determinismo dei tempi di gestione delle routine.

Per eliminare il primo problema abbiamo dotato il robot della possibilità di muoversi a velocità costante, con partenza e fermata immediate. Ciò significa sostanzialmente non eseguire l'algoritmo che genera le rampe di accelerazione e decelerazione. Visto che il primo controllo fatto dall'algoritmo è un confronto tra la velocità attuale e quella da raggiungere, è bastato impostare le due velocità allo stesso valore.

Per risolvere i problemi legati ai tempi di ricezione e gestione dei pacchetti, abbiamo deciso di permettere l'impostazione del tragitto da percorrere, completato il quale il robot si ferma.

Per semplificare le operazioni di dead reckoning abbiamo introdotto due nuove funzionalità: una che permette di conoscere in qualunque momento il tragitto effettuato dal robot dall'ultima variazione della velocità e una seconda che permette di essere avvisati dopo il compimento di un tragitto prefissato.

Utilizzando il robot degli motori passo-passo per muoversi, abbiamo deciso di utilizzare i passi dei motori come unità di misura per l'indicazione del tragitto. Per poter fare questo, abbiamo introdotto un conteggio nella routine di generazione delle forme d'onda di pilotaggio dei motori; in questo modo abbiamo un contatore con valore direttamente proporzionale al numero di passi effettuati dal motore.

Per quanto riguarda la possibilità di conoscere lo stato del robot (con eventuali malfunzionamenti verificatisi) e la versione del software installata, abbiamo introdotto funzionalità apposite che restituiscono le informazioni richieste.

Tutte queste scelte sono state validate da prove sia a vuoto che a terra che ci hanno aiutato a capire le velocità massime (quindi i periodi minimi) di avvio sopportabili dal robot senza che perda passi.

Come richiesto, tutte le funzionalità implementate sono applicabili con comandi trasmessi attraverso la porta seriale utilizzando il protocollo a pacchetti definito nella prima versione.

4. Note al lettore

Per rendere più chiara la lettura di questo documento, esponiamo alcune definizioni e convenzioni che abbiamo utilizzato nello stesso:

- Il termine *CCCU* è l'acronimo usato per indicare la *Central Communication and Control Unit*, vale a dire il mini PC installato a bordo del MARMOT con lo scopo di controllarne il funzionamento ad alto livello.
- La sigla *MCU*, *Micro Controller Unit*, è spesso usata nella documentazione Motorola per alludere al 68HC11; noi la prenderemo in prestito. A volte ci riferiremo al medesimo dispositivo utilizzando anche i termini processore e microcontrollore. Nel nostro contesto, la sigla MCU può anche essere vantaggiosamente interpretata come Motor Controller Unit.
- Con il termine GOAL (o OBIETTIVO) intendiamo il numero di passi che l'utente vuole fare compiere al robot. Il raggiungimento del goal (o dell'obiettivo) indica il percorrimento di quel numero di passi.
- Le sigle LSB, MiddleSB e MSB sono gli acronimi di *Last, Middle* e *Most Significant Byte*. Nelle variabili composte da più byte la parte bassa è contraddistinta dalla sigla LSB (byte meno significativo), il byte centrale (necessario per le variabili da tre byte) è il MiddleSB, mentre la parte alta è il MSB (byte più significativo).

Per concludere elenchiamo alcuni prerequisiti che è necessario soddisfare per proseguire con profitto la lettura:

- è necessario che si conosca un minimo dell'architettura della famiglia di processori 68HC11;
- serve qualche conoscenza di carattere generale sui motori a passo;
- serve avere una conoscenza discreta della logica binaria e una certa dimestichezza con le operazioni con i byte;
- è consigliabile inoltre leggere la documentazione relativa alla prima versione del programma.[ROS_SIG01].

Per maggiori approfondimenti legati soprattutto all'implementazione dei programmi si rimanda ai listati degli stessi.

5. Valutazione spostamento

5.1. Il raggiungimento dell'obiettivo (goal)

Il fatto che il robot sia dotato di tre motori crea qualche problema per la valutazione del raggiungimento di un obiettivo (numero di passi impostati): quando consideriamo raggiunto l'obiettivo? Quando almeno un motore ha raggiunto il numero di passi prefissato, quando tutti e tre i motori hanno raggiunto il numero indicato…?

5.1.1. Diversi modi di identificazione del goal

E possibile identificare il raggiungimento del goal in vari modi:

- Mandare una segnalazione per ogni motore. In questo ci sarebbero tre pacchetti di conferma di raggiungimento del goal (uno per ogni motore).
- Considerare raggiunto l'obiettivo quando almeno uno dei motori ha effettuato un numero di passi uguale (o maggiore) di quello prefissato.
- Considerare raggiunto l'obiettivo quando tutti e tre i motori hanno percorso un numero di passi maggiore o uguale a quello prefissato.
- Considerare raggiunto l'obiettivo quando tutti e tre i motori sono in un *intorno* della posizione da raggiungere. Per *intorno* intendiamo un range di valori più o meno ampio intorno al punto desiderato. Non appena risultasse evidente l'impossibilità che i tre motori arrivino nell'*intorno* bisognerebbe prevedere un'opportuna segnalazione d'errore oltre all'immediata fermata dei motori.

5.1.2. Implementazioni diverse in base alle funzionalità

Le soluzioni implementate sono diversificate in base al tipo di funzionalità utilizzata.

Quando il robot viene utilizzato a velocità costante la partenza avviene da fermo e, al raggiungimento del goal, esso si ferma; quindi abbiamo pensato che ogni ruota si debba fermare quando ha raggiunto il numero di passi definito e il pacchetto di segnalazione del goal debba essere inviato quando tutti e tre i motori si sono fermati. In questo modo effettuiamo esattamente i numeri di passi impostati e la segnalazione di goal rappresenta il raggiungimento di una posizione e non semplicemente di tre distanze. Visto che l'intenzione è quella di muoversi a velocità costante, è necessario che il tragitto impostato e le velocità relative determinino una fermata contemporanea dei tre motori. Il non verificarsi di questa condizione potrebbe essere dannoso per i tre motori oltre che portare a movimenti imprevedibili del robot.

La situazione è diversa quando si vuole essere avvisati dopo che il robot ha percorso un determinato tragitto (senza modificare il moto al raggiungimento di tale obiettivo). In questo caso potremmo non essere interessati al raggiungimento di una posizione, ma semplicemente a tre distanze indipendenti l'una dall'altra sui tre motori; abbiamo quindi preferito segnalare il raggiungimento in modo separato per ogni motore.

5.2. Conteggio dei passi

Le schede driver dei motori ricevono in ingresso un'onda quadra generata dal microcontrollore. La velocità dei motori è determinata dalla frequenza di tale forma d'onda. Il motore compie un passo ad ogni fronte di salita di tale forma d'onda. Quindi è sufficiente contare i fronti di salita per poter ottenere il numero di passi effettuati.

Analizzando come viene generata quest'onda, vediamo che è utilizzata la funzionalità output compare in configurazione "Toggle OCx pin on successful compare". Per ricaricare il registro relativo (TOCx) viene utilizzata una routine invocata mediante apposito interrupt quando il contatore free running raggiunge il valore contenuto nel registro (per maggiori dettagli vedere la documentazione relativa alla prima versione del programma di movimentazione [ROS_SIG01] e i relativi datasheet dei driver [TA8435H]). Quindi tale routine viene richiamata ad ogni fronte di salita e discesa della forma d'onda che pilota il driver del motore; perciò viene eseguita due volte ad ogni passo del motore.

Inserendo un conteggio all'interno di questa routine, il contatore conterrà un numero doppio rispetto ai passi effettuati dal motore. Avremmo potuto gestire l'incremento solo sui fronti di salita, ma non è stato fatto per problemi riguardanti il tempo di servizio della routine stessa (maggiori dettagli nel paragrafo 5.2.2).

Il contatore viene resettato ad ogni variazione di velocità, così è sempre possibile conoscere il numero di passi effettuati dall'ultima modifica del movimento.

Naturalmente è stato implementato un contatore per ogni motore.

Il controllo del raggiungimento degli obiettivi viene fatto all'esterno delle routine sopraccitate, sempre per problemi relativi al tempo di servizio delle stesse. Tale controllo viene eseguito ciclicamente così da verificare continuamente il compimento o meno del numero prefissato di passi.

5.2.1. Dimensionamento del contatore

Per il dimensionamento del contatore è necessario fare alcune considerazioni:

- dobbiamo essere in grado di gestire spostamenti nell'ordine dei 10 m;
- il driver del motore funziona in modalità "W1-2 Phase Excitation" quindi ogni giro è composto da 1600 *micropassi*;
- le ruote sono collegate al rotore attraverso una cinghia con rapporto di riduzione 1:4;
- le ruote hanno un diametro di 15 cm;
- il sistema a ruote svedesi ha una cinematica tale che per ogni giro di ruota, il baricentro del robot può spostarsi di una distanza pari a 1.41 volte la circonferenza della stessa;

Adottando contatori da due byte, lo spostamento che il robot sarebbe in grado di fare prima che il contatore vada in overflow è al massimo circa 3,4 metri, misura non accettabile; anche ipotizzando di incrementare i contatori ad ogni fronte di salita, non

arriverebbe nemmeno a 7 metri. Abbiamo perciò scelto di usare contatori a tre byte, che permettono di soddisfare largamente le specifiche, infatti si possono superare gli 800 metri senza fare modifiche di velocità prima che i contatori vadano in overflow.

5.2.2. Valutazioni riguardanti il tempo di servizio delle routine di interrupt

Le routine di gestione degli interrupt devono essere il più veloci possibile, per evitare di dedicare troppo tempo alle gestioni delle stesse, il che porterebbe a distorsione delle onde generate e possibile perdita di controllo del microcontrollore e quindi del robot.

L'introduzione dell'incremento di un contatore allunga il tempo di servizio, ma è indispensabile; bisogna però cercare di ottimizzare le operazioni necessarie.

Questo è il motivo per cui l'incremento avviene sia sui fronti di salita che di discesa e non solo sui primi.

Il fatto di utilizzare un contatore da tre byte allunga i tempi in quanto non ci sono istruzioni dedicate al trattamento di locazioni di tre byte, ma per quanto visto precedentemente questa scelta è indispensabile.

La verifica del raggiungimento dell'obiettivo invece può essere fatta all'esterno delle routine , anche se questo può ritardare il riconoscimento del goal (soprattutto ad alta velocità). Per evitare problemi derivanti dal controllare il raggiungimento dell'obiettivo quando esso è stato superato, il goal viene considerato raggiunto quando sono stati percorsi un numero di passi uguale o maggiore di quelli richiesti.

I test eseguiti ci hanno confermato che il nostro intervento (con le opportune modifiche delle velocità massime) non introduce né perdite di passi né distorsioni della forma d'onda.

5.2.3. Gestione degli overflow

Se il robot compie un numero di passi maggiore di quello esprimibile con 24 bit (diviso per due per la questione del doppio conteggio) senza che venga modificata la sua velocità, il contatore va in overflow. Non gestire tale situazione pregiudicherebbe il corretto controllo del raggiungimento dell'obiettivo.

La gestione degli overflow avviene quasi esclusivamente all'esterno delle routine di servizio degli interrupt.

Dalla CCCU una situazione di overflow di un contatore è rilevabile in quanto il numero di passi spediti risulta pari a 2^{24}=16777216. Tale condizione non è verificabile altrimenti, visto che l'overflow avviene dopo $\frac{2^{24}}{2}$ passi.

6. Posizionamento a velocità costante

6.1. Introduzione

Per effettuare con semplicità alcune operazioni mentre si sta effettuando uno spostamento a velocità costante, oltre al pacchetto di impostazione di tale velocità per un numero prefissato di passi, ne abbiamo creati altri. In particolare abbiamo considerato l'eventualità di voler interrompere il movimento del robot (*sleep*) e di riprenderlo successivamente (*wakeup*) verso la posizione precedentemente impostata.

L'operazione di *sleep* può essere utilizzata anche semplicemente per fermare il robot, in quanto poi sarà possibile riprendere con qualsiasi tipo di movimento (sia a velocità costante che non).

Un'operazione di *wakeup* potrà avvenire solo se il robot si trova in uno stato di *sleep* (dopo l'operazione di *sleep* non è stata eseguita nessun'operazione di variazione della velocità); un'operazione di *sleep* potrà essere effettuata solo se ci stiamo spostando a velocità costante verso un obiettivo prefissato.

6.2. Pacchetto SETCONSTSP

Pacchetto trasmesso dalla CCCU alla MCU quando si vuole fissare una velocità costante dei tre motori per un numero fissato di passi. E' composto da diciannove byte, il cui significato è illustrato nella Figura 1. Le sigle M1, M2 e M3 si riferiscono al primo, al secondo e al terzo motore, rispettivamente.

Lunghezza: *19*	
ID_µ: *0*	ID_pkt: ***ID_SETCONSTSP***
MSB periodo M1	
LSB periodo M1	
MSB passi M1	
MiddleSB passi M1	
LSB passi M1	
MSB periodo M2	
LSB periodo M2	
MSB passi M2	
MiddleSB passi M2	
LSB passi M2	

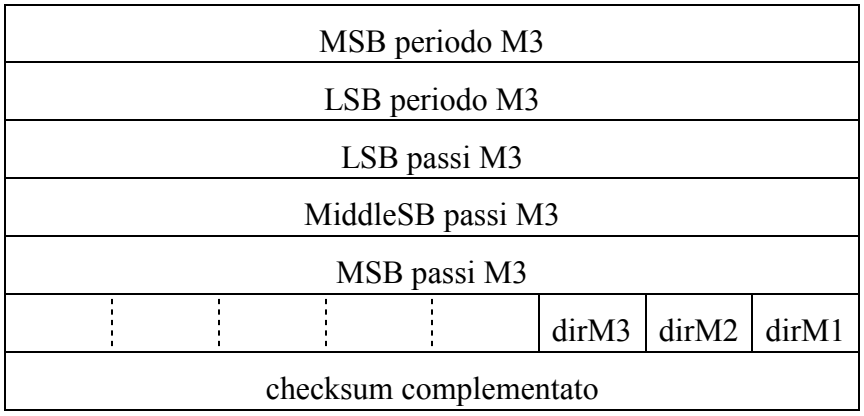


Figura 1: struttura del pacchetto SETCONSTSP

Utilizzando 24 bit per il numero di passi di ogni motore e ricordando che internamente al microcontrollore il numero deve poter essere raddoppiato, ne possiamo impostare fino a $\frac{2^{24}}{2} = 2^{23} = 8388608$, equivalenti a circa 860 metri. Anche impostando una quantità maggiore, il numero verrà fissato internamente al robot a 8388608.

Non utilizzando rampe di modifica della velocità per la partenza e la fermata, il rischio di perdita di passi è maggiore, quindi è stata diminuita la velocità massima, cioè aumentato il periodo minimo. Tale periodo minimo è stato portato a 1000 (comunque solo per il movimento a velocità costante).

In caso di successo, la MCU risponde alla CCCU mandandole un pacchetto di tipo SETCONSTSP, ma senza dati, come mostrato nella Figura 2.

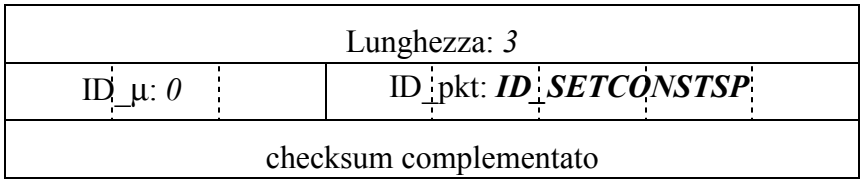


Figura 2 risposta al pacchetto SETCONSTSP

NOTA: questa risposta è facilmente eliminabile trasformando in commento alcune righe del sorgente del programma che gira sulla MCU. In questo caso una mancata risposta significa che tutto procede bene.

In caso di problemi, dovuti al checksum errato oppure a timeout del pacchetto ricevuto, la MCU risponde rispettivamente con un pacchetto di tipo BADCHKSUM oppure TIMEOUT, come illustrato nella documentazione relativa alla prima versione del programma.

6.2.1. Pacchetto GOALSETSPCONST

Pacchetto trasmesso dalla MCU alla CCCU al raggiungimento dell'obiettivo prefissato che, come precedentemente detto, implica il percorrimento dei passi impostati di tutti e tre i motori. E' composto da tre byte, il cui significato è illustrato nella Figura 3.

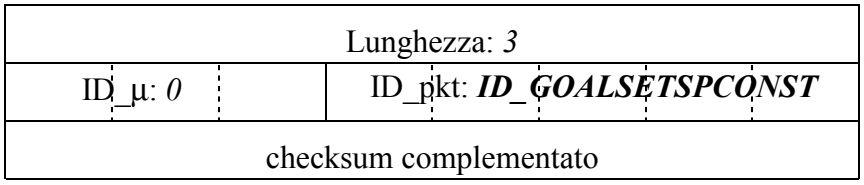


Figura 3 raggiungimento obiettivo impostato con SETSPCONST

6.3. Pacchetto SLEEPSPCONST

Pacchetto trasmesso dalla CCCU alla MCU quando si intende fermare il robot mantenendo in memoria l'obiettivo impostato (il numero di passi mancante). La struttura del pacchetto è mostrata in Figura 4.

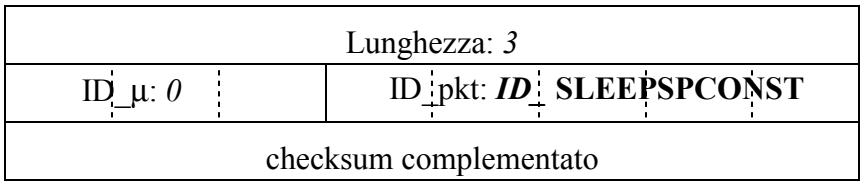


Figura 4 struttura del pacchetto SLEEPSPCONST

In caso di successo, la MCU risponde mandando un pacchetto identico come illustrato in Figura 5.

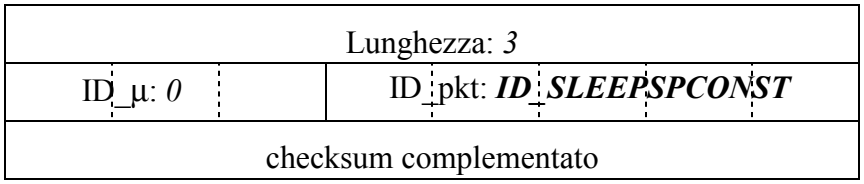


Figura 5 struttura del pacchetto di risposta SLEEPSPCONST

NOTA: questa risposta è eliminabile trasformando in commento alcune righe del sorgente del programma che gira sulla MCU. In questo caso una mancata risposta significa che tutto procede bene.

In caso di problemi, dovuti al checksum errato oppure a timeout del pacchetto ricevuto, la MCU risponde rispettivamente con un pacchetto di tipo BADCHKSUM oppure TIMEOUT, come illustrato nella documentazione relativa alla prima versione del programma.

In caso il pacchetto di SLEEPSPCONST arrivi mentre il robot non si sta muovendo a velocità costante verso un obiettivo prefissato, il pacchetto non può essere accettato e la MCU risponde alla CCCU con un pacchetto di tipo BADSLEEPSPCONST.

6.3.1. Pacchetto BADSLEEP

Pacchetto trasmesso dalla MCU alla CCCU per segnalare la ricezione di un pacchetto SLEEPSPCONST mentre il robot non si trova in modalità *velocità costante*. La struttura del pacchetto è mostrata in Figura 6.

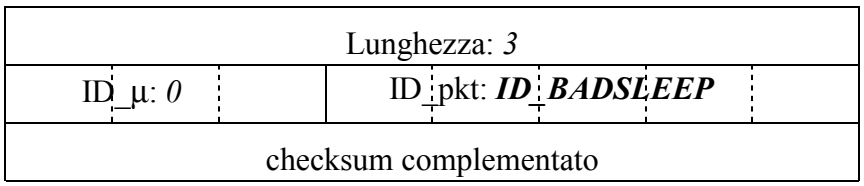


Figura 6 struttura del pacchetto BADSLEEP

6.4. Pacchetto WAKEUPSPCONST

Pacchetto trasmesso dalla CCCU alla MCU quando s'intende riprendere il movimento, interrotto da un'operazione di *sleep*, verso la posizione precedentemente impostata. La struttura del pacchetto è mostrata in Figura 7.

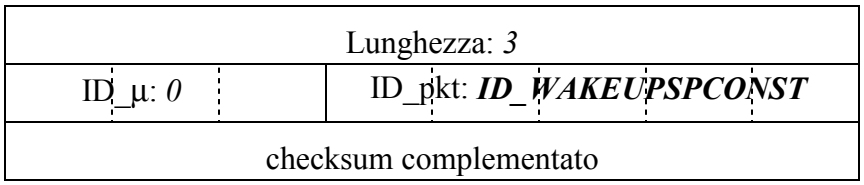


Figura 7 struttura del pacchetto WAKEUPSPCONST

In caso di successo la MCU risponde alla CCCU con un pacchetto dello stesso tipo di WAKEUPSPCONST mostrato in Figura 8.

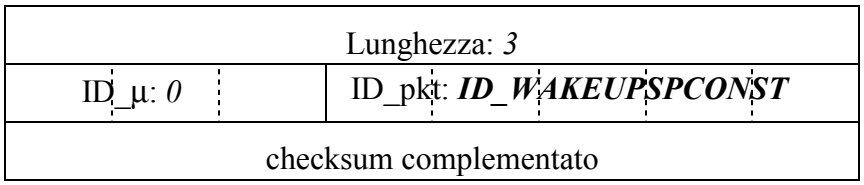


Figura 8 struttura del pacchetto di risposta WAKEUPSPCONST

NOTA: questa risposta è eliminabile trasformando in commento alcune righe del sorgente del programma che gira sulla MCU. In questo caso una mancata risposta significa che tutto procede bene.

In caso di problemi, dovuti al checksum errato oppure a timeout del pacchetto ricevuto, la MCU risponde rispettivamente con un pacchetto di tipo BADCHKSUM

oppure TIMEOUT, come illustrato nella documentazione relativa alla prima versione del programma.

In caso il pacchetto di WAKEUPSPCONST arrivi mentre il robot non è fermo in seguito ad un'operazione di *sleep*, il pacchetto non può essere accettato e la MCU risponde alla CCCU con un pacchetto di tipo BADWAKEUP.

6.4.1. Pacchetto BADWAKEUP

Pacchetto trasmesso dalla MCU alla CCCU per segnalare la ricezione di un pacchetto WAKEUPSPCONST mentre il robot non si trova in uno stato di *sleep*. La struttura del pacchetto è mostrata in Figura 9.

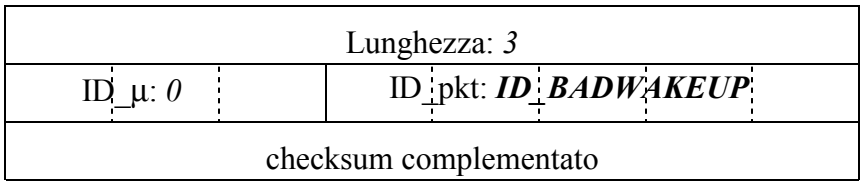


Figura 9 struttura del pacchetto BADWAKEUP

7. Informazioni sullo spostamento effettuato

7.1. Introduzione

Come già detto, avere informazioni sulla posizione del robot è utile per conoscere e controllare il suo spostamento.

È possibile chiedere al robot quanti passi ha percorso dall'ultima ricezione di un pacchetto di impostazione delle velocità, oppure impostare un numero di passi, compiuti i quali il robot spedisce un'opportuna segnalazione.

7.2. Pacchetto GETSTEP

Pacchetto trasmesso dalla CCCU alla MCU quando si vogliono conoscere i passi effettuati dai tre motori dall'ultimo invio di un pacchetto di settaggio della velocità. E' composto da tre byte, il cui significato è illustrato nella Figura 10.

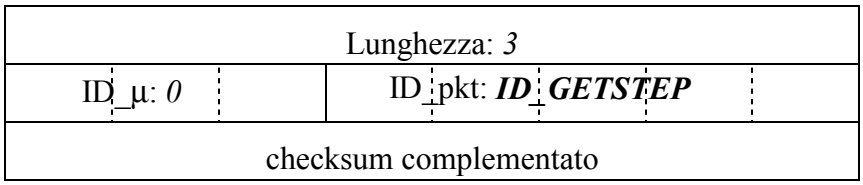
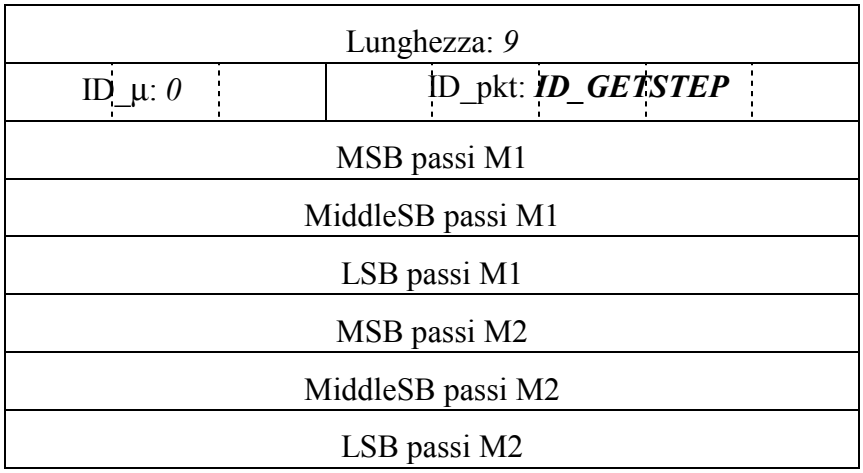


Figura 10 struttura del pacchetto GETSTEP

La MCU risponde mandando un pacchetto dello stesso tipo GETSTEP, ma con i dati illustrati nella Figura 11. Il pacchetto di risposta è composto da 12 byte; le sigle M1, M2 e M3 si riferiscono al primo, al secondo ed al terzo motore, rispettivamente.



LSB passi M3
MiddleSB passi M2
MSB passi M3
Checksum complementato

Figura 11 risposta al pacchetto GETSTEP

Se un motore effettua un numero di passi maggiore o uguale a $\frac{2^{24}}{2} = 8388608$ (e quindi il relativo contatore è andato in overflow), i tre byte del motore corrispondente vengono fissati a FF. Vengono considerati pacchetti di settaggio della velocità SETSPEED e SETCONSTSP. In caso di problemi, dovuti a checksum errato oppure a timeout del pacchetto ricevuto, la MCU risponde rispettivamente con un pacchetto di tipo BADCHKSUM oppure TIMEOUT, come illustrato nella documentazione relativa alla prima versione del programma.

7.3. Pacchetto AFTERSTEP

Pacchetto trasmesso dalla CCCU alla MCU quando si vuole essere avvisati del compimento di un prefissato numero di passi, senza modificare il movimento al goal. Come già spiegato, la valutazione del goal è indipendente per ogni motore: infatti al raggiungimento dell'intervallo di passi fissato, viene spedito il pacchetto che indica il compimento dell'obiettivo su quel motore. E' composto da nove byte, il cui significato è illustrato nella Figura 12.

Lunghezza: *9*	
ID_µ: *0*	ID_pkt: ***ID_AFTERSTEP***
MSB passi M1	
MiddleSB passi M1	
LSB passi M1	
MSB passi M2	
MiddleSB passi M2	
LSB passi M2	
LSB passi M3	
MiddleSB passi M2	
MSB passi M3	
Checksum complementato	

Figura 12 Pacchetto AFTERSTEP

Analogamente a quanto accade per SETCONSTSP possiamo impostare fino a 8388608 passi. Anche in questo caso impostando una quantità maggiore, il numero verrà fissato internamente al robot a 8388608.

In caso di successo la MCU risponde mandando un pacchetto dello stesso tipo di AFTERSTEP, ma senza dati; il pacchetto è formato da tre byte illustrati nella Figura 13.

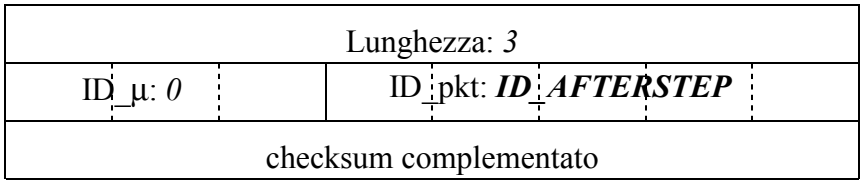


Figura 13 struttura del pacchetto di risposta AFTERSTEP

NOTA: questa risposta è eliminabile trasformando in commento alcune righe del sorgente del programma che gira sulla MCU. In questo caso una mancata risposta significa che tutto procede bene.

In caso di problemi, dovuti al checksum errato oppure a timeout del pacchetto ricevuto, la MCU risponde rispettivamente con un pacchetto di tipo BADCHKSUM oppure TIMEOUT, come illustrato nella documentazione relativa alla prima versione del programma.

7.3.1. Pacchetto BADAFTERSTEP

Per evitare un'eccessiva complessità del programma, viene utilizzata un'unica variabile per memorizzare i riferimenti sia di AFTERSTEP che di SETCONSTSP. Questo porta all'impossibilità di utilizzare le due funzioni contemporaneamente. Abbiamo deciso di dare priorità al pacchetto SETCONSTSP, quindi se viene impostato un riferimento per l'AFTERSTEP e arriva un pacchetto SETCONSTSP, il primo viene annullato; mentre se l'ordine di arrivo è inverso, il pacchetto AFTERSTEP viene rifiutato.

Il pacchetto BADAFTERSTEP viene trasmesso dalla MCU alla CCCU per segnalare l'annullamento o l'impossibilità di elaborare un'operazione di afterstep per i motivi precedentemente esposti. La struttura del pacchetto è mostrata in Figura 14.

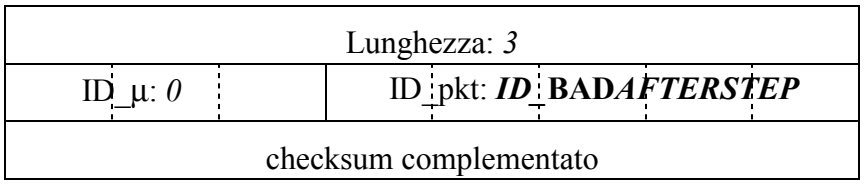


Figura 14 struttura del pacchetto BADAFTERSTEP

7.4. Pacchetti M1GOAL, M2GOAL, M3GOAL

Pacchetti trasmessi dalla MCU alla CCCU quando il rispettivo motore ha raggiunto il goal in seguito ad un'operazione di *afterstep*; i pacchetti sono formati da tre byte illustrati nella Figura 15 per il motore M1, nella Figura 16 per il motore M2 e nella Figura 17 per il motore M3.

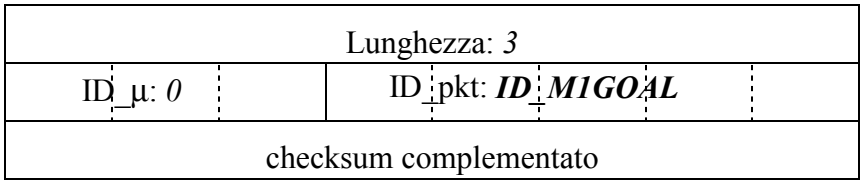


Figura 15 struttura del pacchetto M1GOAL

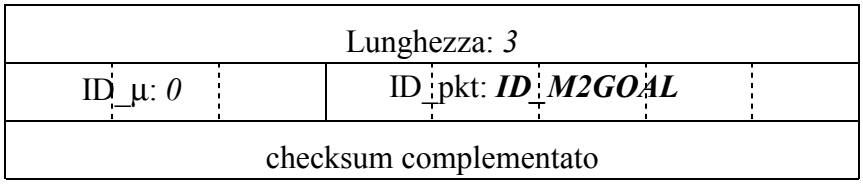


Figura 16 struttura del pacchetto M2GOAL

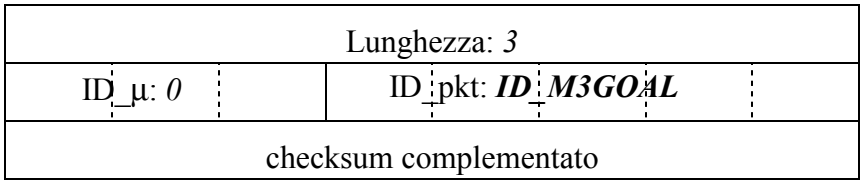


Figura 17 struttura del pacchetto M3GOAL

8. Controllo dello stato

Il robot spedisce sempre un pacchetto di segnalazione alla CCCU quando varia il suo modo di funzionamento, ma tale pacchetto in qualche caso potrebbe andare perso; la CCCU potrebbe quindi avere l'esigenza di conoscere il comportamento del robot in un preciso istante.

Questo è significativo soprattutto nel caso venga interrotto il collegamento tra i due sistemi e venga successivamente ripristinato. Dopo il ripristino la CCCU potrebbe non essere in grado di sapere con certezza cosa stia facendo il robot.

8.1. Diagramma a stati finiti

Il diagramma a stati finiti permette di comprendere come il robot modifica il proprio stato in risposta ai vari eventi (ricezione pacchetti, raggiungimento goal o perdita della connessione).

I cinque stati in cui può trovarsi il MARMOT sono:

- *SpeedConst*: il robot si sta muovendo a velocità costante verso un obiettivo in seguito alla ricezione di un pacchetto SETCONSTSP, quando lo raggiungerà si fermerà e manderà un opportuno pacchetto di segnalazione.
- *Sleep*: il robot è stato fermato da un pacchetto SLEEP mentre era in stato *SpeedConst;* viene mantenuta l'informazione dell'obiettivo da raggiungere.
- *Deadconnection*: da più di 5 sec. non ricevo pacchetti: considero persa la connessione.
- *Fermo*: tutti e tre i motori sono fermi.
- *Rampa*: mi sto muovendo utilizzando rampe di accelerazione e decelerazione in seguito alla ricezione di un pacchetto SETSPEED.

Sugli archi sono rappresentati i vari eventi che possono accadere. Per non appesantire troppo la notazione sono stati rappresentati solo quelli che potrebbero far cambiare stato; sono stati tralasciati tutti i pacchetti che implicano solo la spedizione di dati e i pacchetti che vengono rifiutati.

Gli eventi considerati sono:

- setconstsp: è stato ricevuto un pacchetto SETCONSTSP con velocità non nulla
- setspeed: è stato ricevuto un pacchetto SETSPEED con velocità non nulla
- sleep: è stato ricevuto un pacchetto SLEEP
- wakeup: è stato ricevuto un pacchetto WAKEUP
- 5s: sono passati più di 5 secondi dall'ultima ricezione di un pacchetto
- goal: è stato raggiunto il goal

- stop: è stato spedito un pacchetto di variazione della velocità (SETSPEED o SETCONSTSP) con velocità nulla
- nospeed: è stato spedito un qualsiasi tipo di pacchetto tranne setspeed o setconstsp (con velocità diverse da zero).

Il diagramma a stati finiti è illustrato in Figura 18.

NOTA: tale diagramma si riferisce al caso in cui tutti e tre i motori siano alimentati

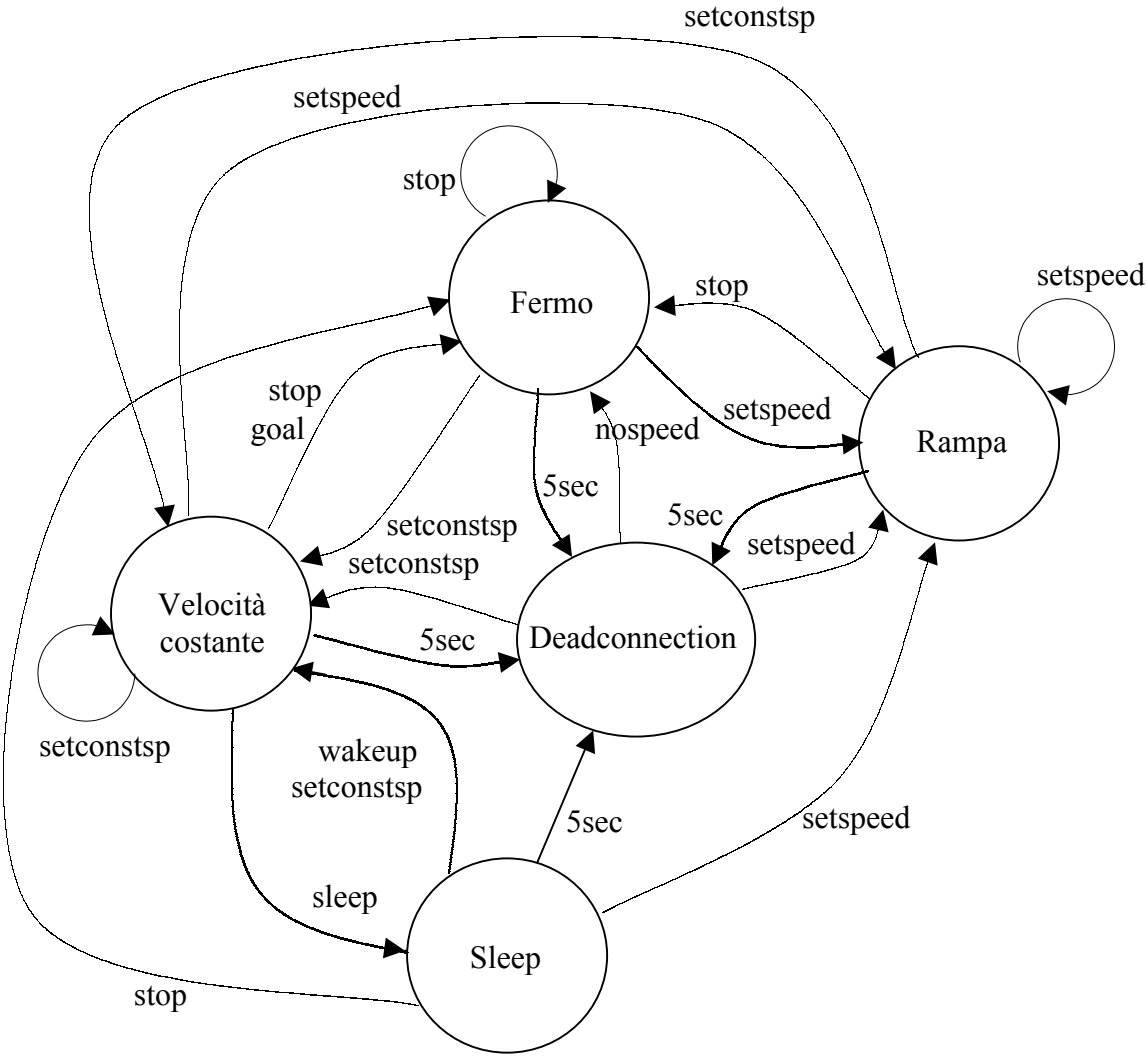


Figura 18 Diagramma stati finiti

8.2. Pacchetto GETSTATUS

Viene inviato dalla CCCU alla MCU per richiedere lo stato in cui si trova il robot. E' composto da tre byte, il cui significato è illustrato in Figura 19

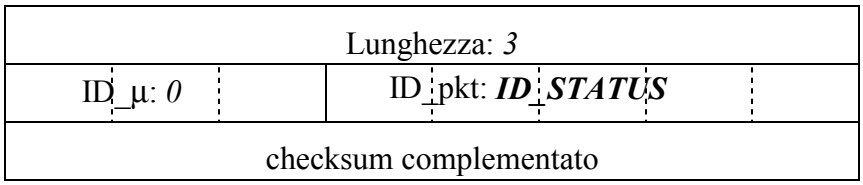


Figura 19 Struttura pacchetto GETSTATUS

La MCU risponde mandando un pacchetto dello stesso tipo GETSTATUS, ma con i dati illustrati nella Figura 20. Il pacchetto di risposta è composto da 4 byte.

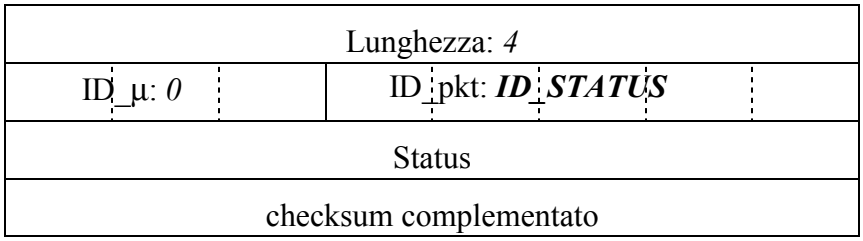


Figura 20 Struttura del pacchetto di risposta a GETSTATUS

I bit del byte *status* hanno il significato indicato in Figura 21. I cinque bit meno significativi segnalano in che stato è il robot, il bit *NoTuttiAlim* viene settato se non tutti i motori sono alimentati, mentre quello *Afterstep* indica l'attivazione o meno dell'opzione *afterstep*.

NoTuttiAlim	Afterstep	-	Rampa	Fermo	Deadconn	Constspeed	Sleep

Figura 21 Significato dei bit del byte *Status*

In caso di problemi, dovuti al checksum errato oppure timeout del pacchetto ricevuto, la MCU risponde rispettivamente con un pacchetto di tipo BADCHKSUM oppure TIMEOUT, come illustrato nella documentazione relativa alla prima versione del programma.

9. Selftest

9.1. Introduzione

L'operazione di autotest dell'MCU consiste semplicemente nell'invio di una stinga di caratteri contenente il nome, la versione e la data di creazione del software correntemente installato. Non essendo al momento disponibili informazioni sul funzionamento effettivo dei motori, delle schede driver o di altri componenti, non è possibile fornire ulteriori dettagli sul corretto funzionamento del robot.

9.2. Pacchetto SELFTEST

Viene inviato dalla CCCU alla MCU per richiedere un test del funzionamento del robot. E' composto da tre byte, il cui significato è illustrato in Figura 22.

Lunghezza: *3*	
ID_µ: *0*	ID_pkt: ***ID_SELFTEST***
checksum complementato	

Figura 22 struttura del pacchetto SELFTEST

In caso di successo, la MCU risponde alla CCCU inviandole un pacchetto di tipo SELFTEST, ma con i dati illustrati in Figura 23.

Lunghezza: *30*	
ID_µ: *0*	ID_pkt: ***ID_SELFTEST***
M	
A	
R	
M	
O	
T	
-	
M	
o	

v
e
r
“ ”
2
.
X
“ “
g
g
-
m
m
-
a
a
a
a
checksum complementato

Figura 23 risposta al pacchetto SELFTEST

I caratteri da inviare sono codificati in codice ASCII.

NOTA: Il carattere X e la data saranno sostituiti dai corretti valori nelle varie release.

In caso di problemi, dovuti al checksum errato oppure a timeout del pacchetto ricevuto, la MCU risponde rispettivamente con un pacchetto di tipo BADCHKSUM oppure TIMEOUT, come illustrato nella documentazione relativa alla prima versione del programma.

10. Programma MCU

10.1. Struttura del programma

Da un punto di vista ad alto livello il programma è un semplice ciclo infinito che svolge queste tre attività come mostrato Figura 24

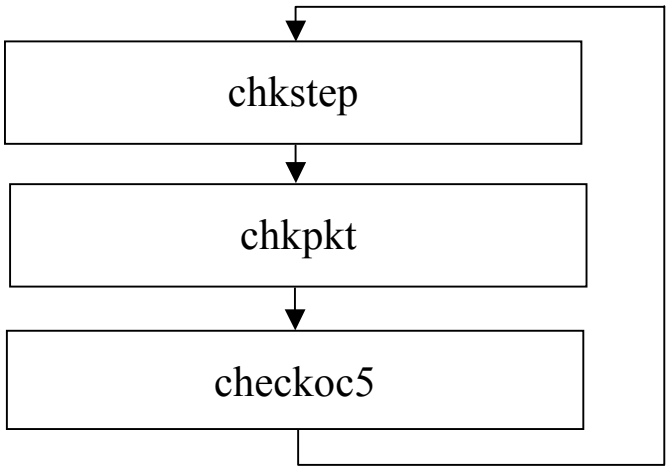


Figura 24 Ciclo del programma

1. Controllare, tramite una chiamata alla routine **chkstep,** se è stato raggiunto qualche obiettivo. In caso positivo, in base al tipo di goal raggiunto, vengono modificate le variabili che memorizzano di quali e quanti goal sono stati raggiunti e, in alcuni casi, viene spedito un pacchetto di notifica del goal.
2. Controllare, tramite una chiamata alla routine **chkpkt,** se sono stati ricevuti dati dal canale seriale. In caso positivo questi verranno accumulati in memoria fino a completare un pacchetto. Alla conclusione di un pacchetto, questo sarà esaminato per verificarne la validità. Se il test viene superato le richieste che contiene saranno esaudite, con eventuale ritrasmissione alla CCCU di un pacchetto di notifica.
3. Verificare, tramite una chiamata alla routine **checkoc5**, se il quarto timer ha esaurito il tempo impostato. In caso affermativo saranno trascorsi almeno **5ms** dal precedente caricamento del timer ed è necessario eseguire questi tre compiti:
   3.1. Decrementare il contatore utilizzato per rilevare eventuali timeout durante la ricezione di pacchetti dalla seriale. In caso il contatore raggiunga lo zero, la routine **chkpkt** si incaricherà di gestire l'eventuale timeout invalidando il pacchetto e informando la CCCU della circostanza.

3.2. Aggiornare, tramite tre chiamate consecutive alla routine parametrica **aggiornarampa**, la velocità attuale di rotazione di ciascuno dei tre motori. Questa routine è stata implementata appositamente per la gestione delle rampe di accelerazione e decelerazione, infatti viene saltata quando il robot è in modalità *constspeed*. Le frequenze dei segnali di clock vengono quindi aggiornate ogni 5ms. La routine **checkperiodi** si incarica di rendere effettive le modifiche, trasmettendole ai vari dispositivi di I/O coinvolti e alterando i registri dei timer.

3.3. Controllare lo stato della connessione con la CCCU, dichiarandola persa se non si ricevono dati da troppo tempo. Questo intervallo è stato fissato da noi in **5s** attraverso la costante **DEADTIME**. Per ragioni di sicurezza i motori vengono fermati se questa situazione si verifica.

Organizzando il programma in questo modo siamo riusciti a soddisfare le specifiche del problema senza avere necessità di introdurre alcun ciclo di ritardo. Le routine sono state scritte per la massima velocità operativa e non per minimizzare l'occupazione di memoria.

10.2. Identificatori pacchetti

In Tabella 1 sono riportati i nuovi pacchetti definiti in questa versione, caratterizzati da nome univoco e identificatore (associato al rispettivo valore).

Nome	*Identificatore*	*Valore*
M1GOAL	**ID_M1GOAL**	10
M2GOAL	**ID_M2GOAL**	11
M3GOAL	**ID_M3GOAL**	12
GOALCONSTSP	**ID_GOALCONSTSP**	13
SLEEP	**ID_SLEEP**	14
BADSLEEP	**ID_BADSLEEP**	15
SETCONSTSP	**ID_SETCONSTSP**	16
WAKEUP	**ID_WAKEUP**	17
BADWAKEUP	**ID_BADWAKEUP**	18
AFTERSTEP	**ID_AFTERSTEP**	19
BADAFTERSTEP	**ID_BADAFTERSTEP**	20
GETSTEP	**ID_GETSTEP**	21
STATUS	**ID_STATUS**	22
SELFTEST	**ID_SELFTEST**	23
NOTTHISVER	**ID_NOTTHISVER**	24

Tabella 1 Identificatori pacchetti

NOTA: il pacchetto NOTTHISVER verrà presentato nel paragrafo 11.2.1.

I pacchetti GOALCONSTSP, SLEEP, SETCONSTSP, WAKEUP, AFTERSTEP, GETSTEP, STATUS, SELFTEST appartengono alla categoria dei *pacchetti di comando*. Questi pacchetti provengono dalla CCCU e sono diretti alla MCU, che deve risolverli in un cambiamento delle sue uscite o nella trasmissione verso la CCCU di alcuni valori. Durante la trasmissione verso la CCCU, la MCU può utilizzare gli stessi tipi di pacchetto per comunicare dati, oppure semplicemente per indicare alla CCCU che il comando relativo è stato ricevuto correttamente ed eseguito con successo.

I rimanenti pacchetti M1GOAL, M2GOAL, M3GOAL, BADSLEEP, BADWAKEUP, BADAFTERSTEP, NOTTHISVER, sono di pura segnalazione: vengono inviati soltanto dalla MCU alla CCCU per notificarle particolari situazioni.

10.3. Uso di variabili di stato

Per una gestione più semplice e intuitiva dei vari modi di funzionamento del robot e dei vari goal raggiunti, abbiamo introdotto quattro variabili che permettono di ottenere facilmente informazioni su:

- stato in cui si trova il robot
- overflow dei contatori e dei riferimenti
- goal raggiunti
- disabilitazione incremento variabili contenenti il numero di passi percorsi

Ognuna di queste variabili ha dimensione di un byte di cui ogni bit ha un particolare significato. Le variabili sono chiamate rispettivamente state, stepbits, goalstep e no_inc. La loro struttura è mostrata in Figura 25, Figura 26, Figura 27 e Figura 28

-	-	-	-	Deadconn	Afterstep	Sleep	Constspeed

Figura 25 Significato dei bit della variabile *state*

I significati dei bit della variabile state sono i seguenti (da notare la leggera diversità rispetto al byte status del pacchetto GETSTEP):

bit 7 6 5 4 3 2 1 0 significato del valore 1

- E' stato impostato un obiettivo da raggiungere a vel. costante
- Il robot è in stato di *sleep*
- Opzione *afterstep* attiva
- Si è interrotta la connessione
- non usato
- non usato
- non usato
- non usato

-	-	Ovrref3	Ovrref2	Ovrref1	Step3ovr	Step2ovr	Step1ovr

Figura 26 Significato dei bit della variabile stepbits

I significati dei bit della variabile stepbits sono i seguenti:

bit 7 6 5 4 3 2 1 0 significato del valore 1

- Il contatore dei passi del primo motore è andato in overflow
- Il contatore dei passi del secondo motore è andato in overflow
- Il contatore dei passi del terzo motore è andato in overflow
- La variabile dell'obiettivo del primo motore è in overflow
- La variabile dell'obiettivo del secondo motore è in overflow
- La variabile dell'obiettivo del terzo motore è in overflow
- non usato
- non usato

-	-	-	-	-	Goalstep3	Goalstep2	Goalstep1

Figura 27 Significato dei bit della variabile goalstep

I significati dei bit della variabile goalstep sono i seguenti:

bit 7 6 5 4 3 2 1 0 significato del valore 1

- Il primo motore ha percorso il numero di passi fissato
- Il secondo motore ha percorso il numero di passi fissato
- Il terzo motore ha percorso il numero di passi fissato
- non usato
- non usato
- non usato
- non usato
- non usato

-	-	-	-	-	Noincstep3	Noincstep2	Noincstep1

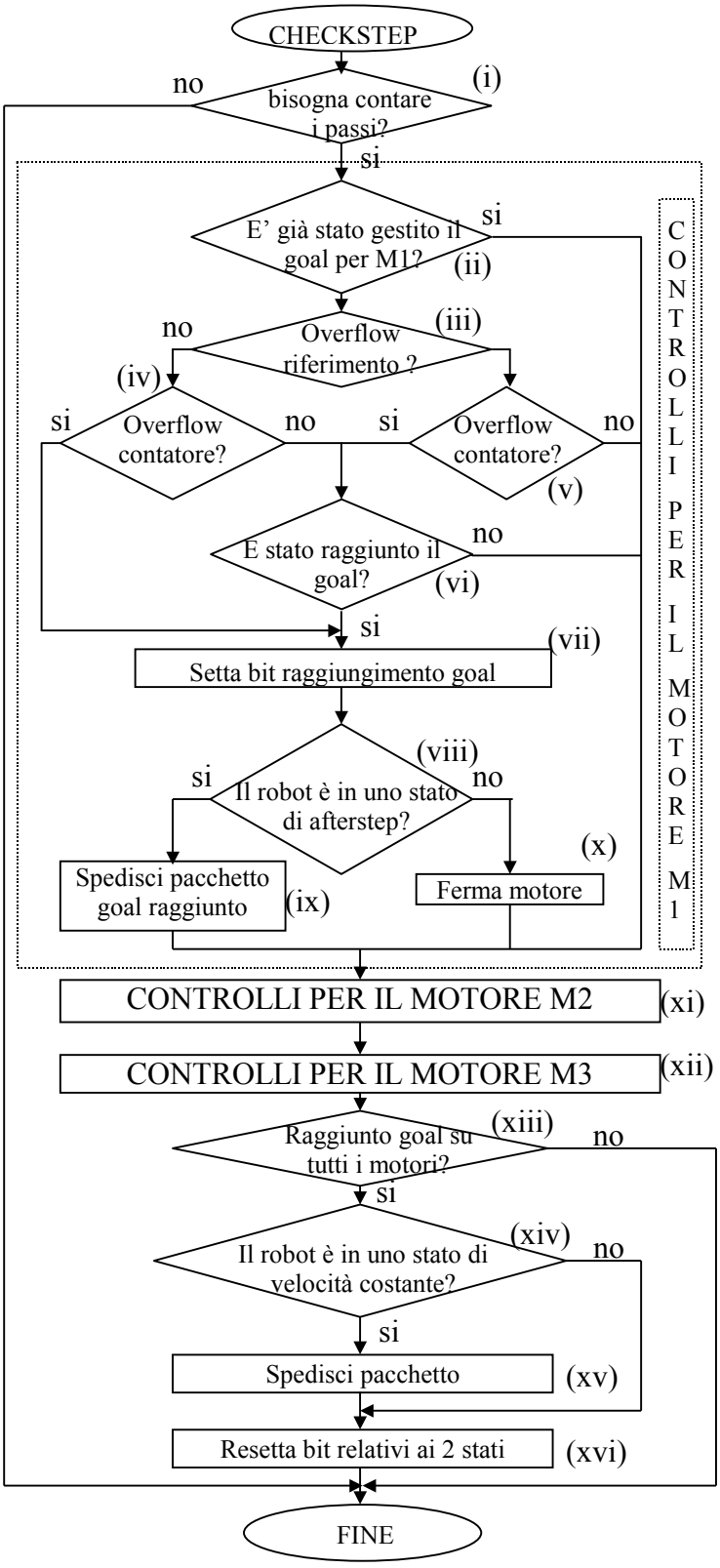
Figura 28 Significato dei bit della variabile no_inc

I significati dei bit della variabile no_inc sono i seguenti:

bit 7 6 5 4 3 2 1 0 significato del valore 1

- Step1 non deve essere incrementato dalle routine di interrupt
- Step2 non deve essere incrementato dalle routine di interrupt
- Step3 non deve essere incrementato dalle routine di interrupt
- non usato
- non usato
- non usato
- non usato
- non usato

10.4. Routine checkstep



(i) Il requisito principale per poter entrare nella routine è di dover controllare i passi (per verificare l'eventuale raggiungimento di un obiettivo), il che implica essere in uno stato *speedconst* o avere attiva l'opzione *afterstep* (variabile *state*). Se questo non è verificato si passa direttamente alla fine della routine.

Gestione del motore 1 (M1):

(ii) Se è già stato gestito il raggiungimento dell'obiettivo per il primo motore (variabile goalstep) si può passare al controllo del secondo.

Altrimenti:

(iii) Viene controllato l'overflow del riferimento dell'obiettivo per poi decidere come comportarsi con l'overflow del contatore (variabile *stepbits*).

(iv) Nel caso in cui il riferimento non sia in overflow, il raggiungimento del goal viene verificato solo se il contatore non è andato in overflow, se invece ciò accade significa che il riferimento è stato superato quindi il goal è sicuramente stato raggiunto (variabile *stepbits*).

(v) Viceversa se il riferimento è in overflow, il raggiungimento del goal viene verificato solo se anche il contatore è in questa situazione, infatti in questa situazione il mancato overflow del contatore implica il non raggiungimento del goal (variabile *stepbits*).

(vi) Se il contatore ha superato il riferimento del goal significa che il goal è stato raggiunto.

(vii) Il raggiungimento del goal implica il settaggio del bit associato a questo motore nella variabile *goalstep.*

(viii) In base allo stato in cui il robot si trova (variabile *state*) o alle eventuali opzioni attive, bisogna decidere cosa fare dato che è stato raggiunto l'obiettivo:

(ix) Se il robot è in stato di *speedconst* viene fermato immediatamente il motore.

(x) Se l'opzione *afterstep* è settata, viene spedito il pacchetto che avvisa del raggiungimento del goal di M1.

(xi) e (xii) Lo stesso flusso di controllo appena illustrato viene ripetuto per i motori M2 e M3.

(xiii) Se i tre motori hanno superato l'obiettivo (variabile *goalstep*), viene controllato se il robot è in stato *speedconst.*

(xiv) e (xv) Se lo stato del robot è *speedconst* (variabile *state*) viene segnalato alla CCCU il raggiungimento del goal attraverso la spedizione del pacchetto relativo.

(xvi) Dato che sono stati raggiunti i goal si devono resettare i bit della variabile di stato *state* relativi a *afterstep e speedconst.*

Figura 29 flowchart routine *checkstep*

La routine checkstep, mostrata in Figura 29, è la modifica principale all'architettura generale; viene infatti modificato il ciclo principale del programma. Permette di controllare l'eventuale raggiungimento degli obiettivi.

10.4.1. Controllo dell'overflow

Il controllo del raggiungimento di un obiettivo è basato sul confronto tra le due variabili di tre byte *stepX* (contenente il numero di passi percorsi) e *refstepX* (contenente il numero di passi da compiere per raggiungere l'obiettivo). Nel momento in cui il valore di *stepX* è maggiore o uguale di quello di *refstepX,* si considera raggiunto l'obiettivo. L'eventuale overflow di una delle due variabili potrebbe pregiudicare il corretto raggiungimento del goal. Per evitare questo abbiamo implementato particolari gestioni delle condizioni di overflow.

Nella Tabella 2 vengono riportate le quattro possibili combinazioni degli overflow con il relativo metodo di controllo del raggiungimento del goal:
Control: viene eseguito il controllo normale (stepX ≥ refstepX?);
NoGoal: sicuramente non è stato raggiunto l'obiettivo;
Goal: sicuramente è stato raggiunto l'obiettivo;

	stepX	**refstepX**	**Goal?**
A	No_ovr	No_ovr	Control
B	No_ovr	Ovr	NoGoal
C	Ovr	No_ovr	Goal
D	Ovr	Ovr	Control

Tabella 2 Casi di overflow

In Figura 30 è rappresentato un esempio del caso A (il più frequente), in cui non si sono verificati overflow e quindi non è necessario adottare particolari strategie per il controllo del raggiungimento del goal: verranno semplicemente confrontate le variabili refstepX e stepX (3E 00 00 ≥ BC 78 00?).

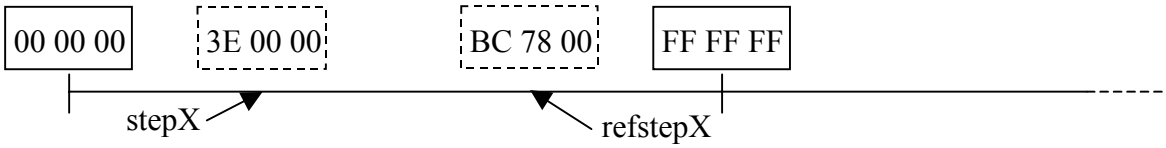


Figura 30 Esempio caso A overflow

Nel caso B, di cui è mostrato un esempio in Figura 31, bisogna saltare il controllo normale perché non è stato raggiunto il goal (stepX precede refstepX), ma tale controllo darebbe esito opposto (C2 00 00>40 00 00!).

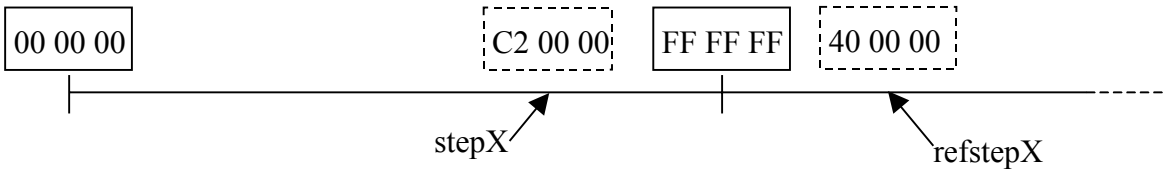


Figura 31 Esempio caso B overflow

Nel caso C (esempio in Figura 32) bisogna dichiarare raggiunto l'obiettivo senza fare nessun ulteriore confronto perché stepX ha superato refstepX, ma il confronto darebbe risultato opposto (00 00 0C<FF FF E2!). Questo è particolarmente importante quando il riferimento si trova molto vicino all'overflow. Può accadere infatti che il contatore vada in overflow (e quindi contenga 00 00 00) prima che vengano confrontate le due variabili, a questo punto il confronto darebbe risultato negativo.

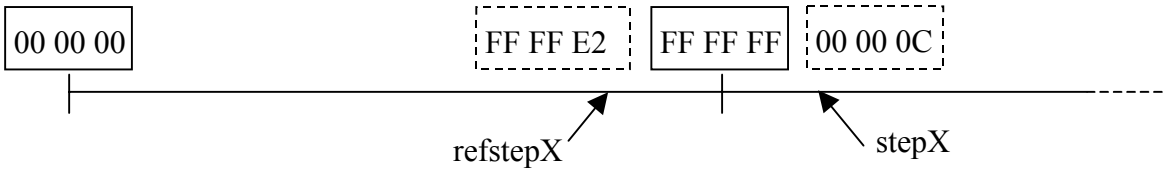


Figura 32 Esempio caso C overflow

Nell'ultimo caso (D) il controllo del raggiungimento del goal viene effettuato con un normale controllo come nel caso A (10 00 0C≥30 00 01?). Esempio in Figura 33.

Figura 33 Esempio caso D overflow

I quattro casi elencati sono gestiti nella routine checkstep, la quale è stata programmata ottimizzando (rispetto al tempo di calcolo) il caso più frequente. Per stimare la frequenza con cui si presenta ogni caso basta pensare all'ambiente di lavoro del Robot: ha un'estensione nell'ordine dei 10 metri, mentre l'overflow del contatore si verifica dopo aver percorso circa 800 metri senza variare la velocità del robot. E' abbastanza chiaro come l'overflow del contatore sia un evento piuttosto raro. Per quanto riguarda l'overflow del riferimento invece la probabilità è leggermente maggiore. Infatti potrebbe venire impostato un obiettivo distante 800 metri mediante un operazione *afterstep* (che quindi non azzera il contatore dei passi); a questo punto il riferimento dovrà contenere il valore dei passi fatti più quelli rimanenti e quindi potrebbe presentarsi una situazione di overflow. E' interessante notare come tale obiettivo può essere raggiunto senza che anche il contatore dei passi vada in overflow (con variazioni di velocità intermedie che lo azzerano). La probabilità del verificarsi di un overflow del riferimento è quindi leggermente più alta di quella del contatore, ma resta comunque molto bassa. Risulta chiaro quindi come il caso più frequente sia quello A.

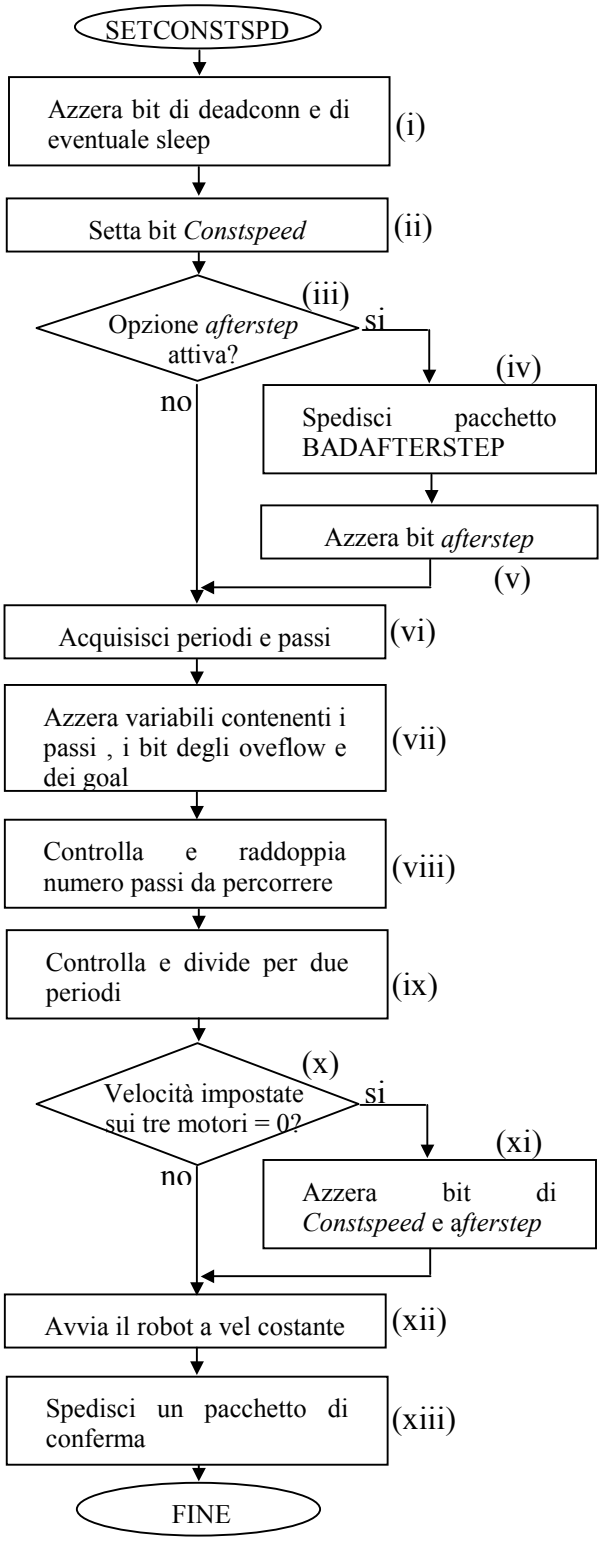
10.5. Altre routine

Vengono presentate ora altre routine che, pur non essendo complicate quanto la routine checkstep, necessitano di una breve spiegazione. Queste routine non introducono modifiche alla struttura (ciclo principale) ma gestiscono la ricezione di particolari pacchetti.

Per le restanti si ritengono esaurienti i commenti nel codice sorgente [File_Marmot].

10.5.1. Setconstspd

Questa routine elabora un pacchetto di tipo SETCONSTSP. In Figura 34 possibile vedere il flow chart della routine.



(i) Vengono resettati i bit di D*eadconnection* e dell'eventuale stato di *sleep*.

(ii) Viene settato il bit dello stato C*onstspeed.*

(iii) Viene controllato se l'opzione *afterstep* è attiva, in caso affermativo viene spedito un pacchetto di BADSLEEP

(iv) per segnalare l'annullamento di tale opzione e viene azzerato il bit relativo (v).

(vi) Vengono salvati i dati ricevuti dei periodi e dei passi nelle relative variabili.

(vii) Vengono azzerate le variabili contenenti i passi effettuati, i bit degli overflow (dei contatori e dei riferimenti) e i bit riguardanti i goal raggiunti.

(viii) Forzo i numeri di passi ricevuti ad assumere un valore compreso nel range consentito quindi raddoppio i valori ottenuti.

(ix) Forzo anche i periodi ricevuti a rientrare nel range consentito e divido per due i valori ottenuti.

(x) Se le velocità impostate per i tre motori sono tutte a zero vengono azzerati i bit relativi a *Constspeed* e *afterstep.*

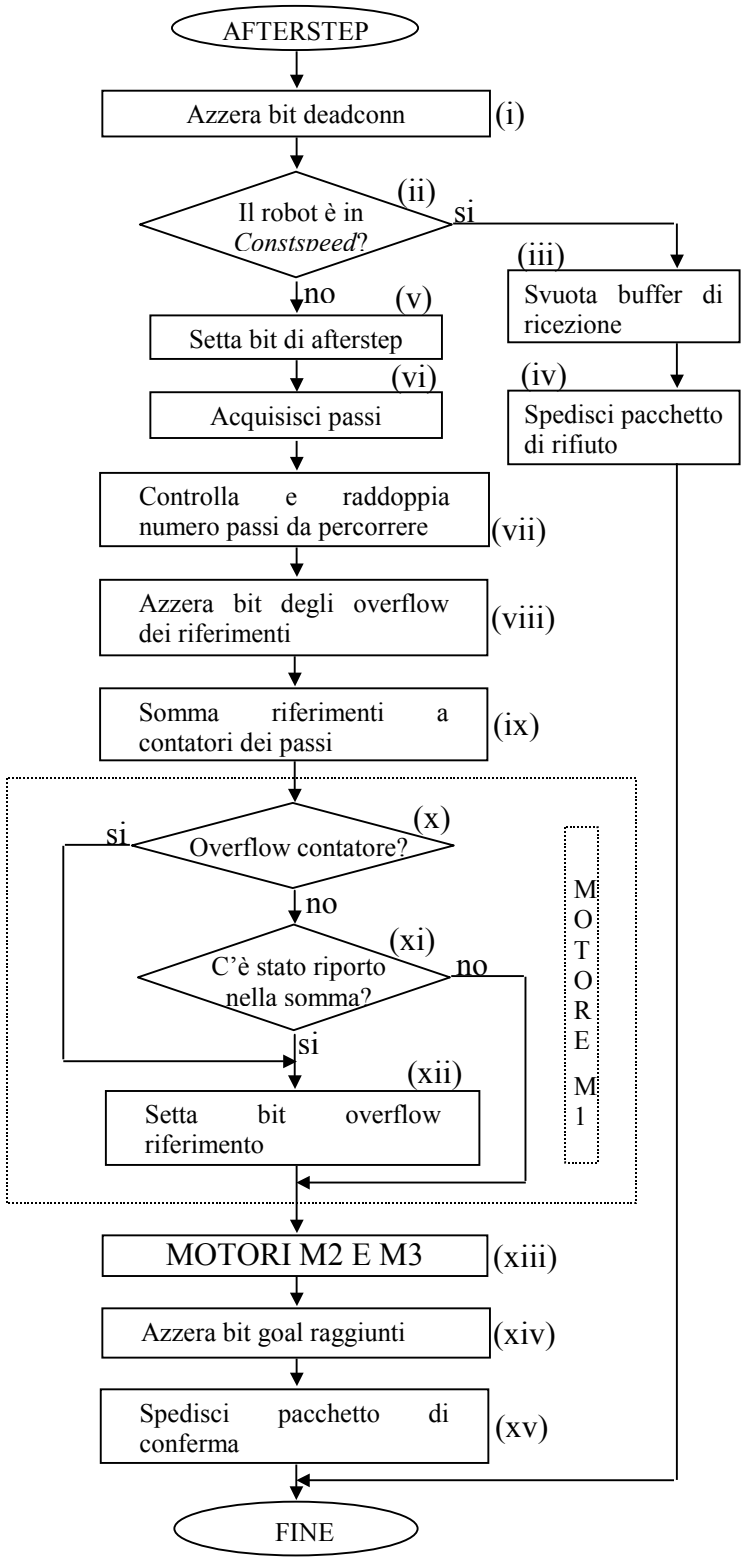
(xii) Fisso le variabili contenenti i periodi (ossia le velocità) correnti delle ruote al nuovo valore, così da saltare le rampe di accelerazione in modo che il robot possa partire a velocità costante.

(xiii) Spedisco un pacchetto di conferma alla CCCU di tipo SETCONSTSP.

Figura 34 Flow chart routine *setconstspd*

10.5.2. Afterstep

Questa routine elabora un pacchetto di tipo AFTERSTEP. In Figura 35 è possibile vedere il flow chart della routine.



(i) Viene azzerato il bit di deadconnection.

(ii) Se il robot si sta muovendo a velocità costante il comando AFTERSTEP non può essere accettato, viene così svuotato il buffer di ricezione (iii) e viene spedito un pacchetto di rifiuto (iv), dopodiché la routine termina.

Se il robot non si sta muovendo a velocità costante, il flusso prosegue normalmente.

(v) Viene settato il bit dell'opzione *afterstep* e vengono acquisiti i passi da percorrere (vi).

(vii) Vengono forzati i numeri di passi ricevuti ad assumere un valore compreso nel range consentito, quindi raddoppio i valori ottenuti.

(viii) Vengono azzerati i bit relativi agli overflow dei riferimenti.

(ix) Vengono sommati il contenuto dei contatori dei passi con quelli dei riferimenti, il risultato viene salvato nelle variabili de riferimenti.

Ora per ogni motore vengono controllati opportunamente i bit degli overflow dei riferimenti:

(x) Se il contatore dei passi è andato in overflow, bisogna settare il bit dell'overflow del riferimento (xii).

(xi) Se invece il contatore non è andato in overflow, viene controllato se nella precedente somma si è verificato un riporto. In caso affermativo viene settato il bit di overflow del riferimento (xii).

(xiii) Lo stesso controllo viene fatto per i motor M1 e M2.

(xiv) Viene azzerata la variabile contenente le indicazioni dei goal raggiunti.

(xv) Viene spedito il pacchetto di conferma di tipo AFTERSTEP alla CCCU.

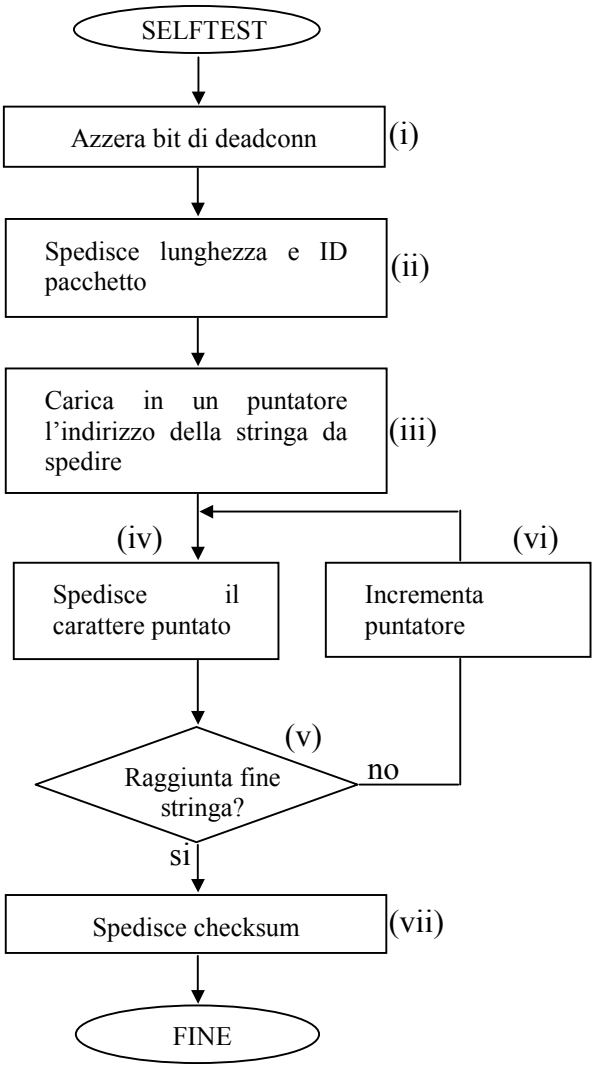
Figura 35 Flow chart routine *afterstep*

10.5.3. Selftest

Questa routine elabora un pacchetto di tipo SELFTEST.

Per poter modificare facilmente la stringa di caratteri ASCII spedita in risposta, la lunghezza del pacchetto è memorizzata in una costante (SELFTESTLENGHT), mentre la stringa è memorizzata come successione di caratteri ASCII a partire dalla locazione con etichetta VERSIONE (per comodità memorizzata nelle ultime locazioni di memoria disponibili). L'ultimo carattere della stringa deve essere NULL (00).

In Figura 36 è possibile vedere il flow chart della routine.



(i) Viene resettato il bit di *deadconnection.*

(ii) Viene spedita la lunghezza e l'ID del pacchetto.

(iii) Viene caricato in un puntatore l'indirizzo del primo carattere della stringa da spedire.

(iv) Viene inviato il carattere puntato.

(v) Si controlla se è stata raggiunta la fine della stringa (indicata da un carattere NULL).

In caso negativo incremento il puntatore (vi) e spedisco il nuovo carattere puntato (iv).

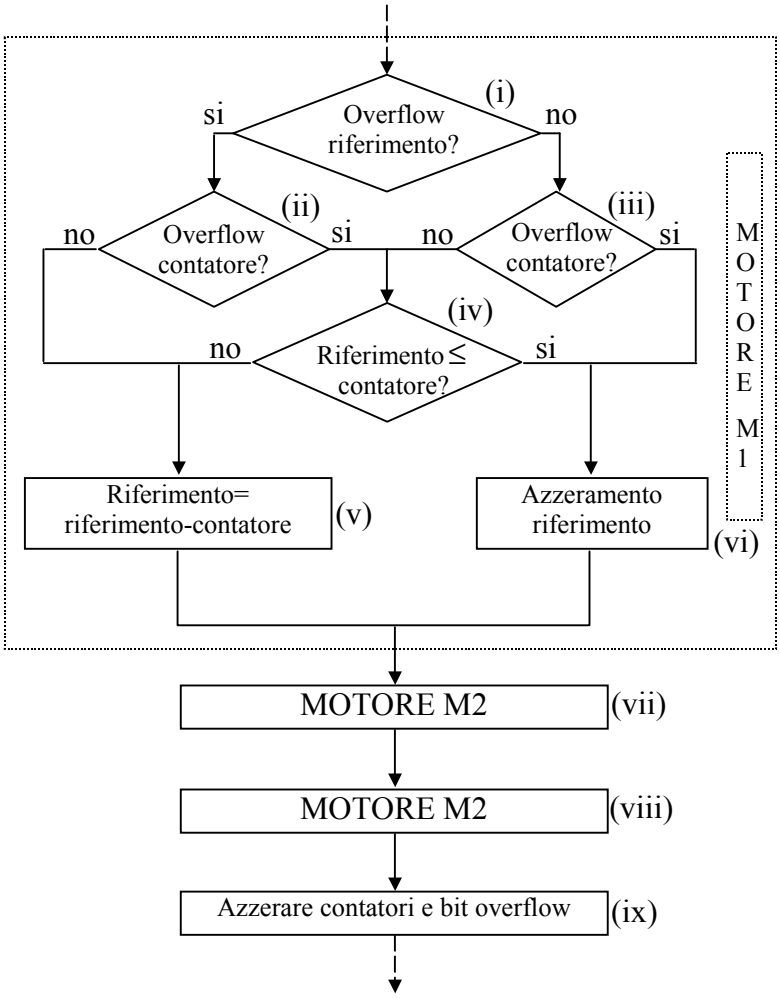
Se invece è stata raggiunta la fine della stringa, spedisco il checksum (vii).

Figura 36 flow chart routine s*elftest*

10.5.4. Setspddir

Questa routine elabora un pacchetto di tipo SETSPEED.

Essa era già compresa nella prima versione, ma è stata modificata per adattarla alle nuove esigenze. La modifica fondamentale è l'aggiornamento delle variabili refstepX per poter completare correttamente l'eventuale *afterstep*. Infatti, visto che il pacchetto SETSPEED comporta una modifica della velocità, la routine dovrà azzerare le variabili stepX; per raggiungere correttamente l'eventuale obiettivo di tipo *afterstep* sarà necessario sottrarre alle variabili refstepX il contenuto di stepX, naturalmente tenendo conto delle varie situazioni di overflow. In Figura 37 viene riportato il flow chart della parte modificata della routine.



In base al verificarsi o meno dell'overflow del contatore e del riferimento dobbiamo applicare strategie diverse (i), (ii) e (iii).

Se il riferimento lo ha raggiunto mentre il contatore no, sicuramente non è stato raggiunto il goal, quindi devo sottrarre dal riferimento il contenuto del contatore (v).

Se invece il riferimento non è andato in overflow ma il contatore sì, ho sicuramente raggiunto l'obiettivo, devo quindi azzerare il riferimento (vi) per poter raggiungere correttamente l'obiettivo (infatti anche il contatore verrà azzerato).

Se sia il contatore che il riferimento (o nessuno dei due) hanno raggiunto l'overflow, allora è necessario controllare se il primo ha superato il secondo (iv). Infatti potrei aver superato l'obiettivo ma non averlo ancora gestito; quindi devo azzerare il riferimento (vi) o sottrargli il contatore (v) secondo i casi.

Lo stesso flusso viene ripetuto per i motori M2 e M3 (vii) e (viii).

Infine vengono azzerate i contatori e i bit relativi agli overflow (ix);i contatori non saranno in overflow perché li ho appena resettati, mentre i riferimenti non lo saranno, visto che la distanza tra stepX e refstepX non potrà mai essere superiore a FF FF FF.

Figura 37 Modifiche alla routine *setspddir*

10.6. Modifica delle variabili stepX

Le variabili stepX vengono incrementate all'interno di alcune routine di servizio degli interrupt (in particolare quelle dell'*output compare*). Questo può causare alcuni problemi quando cerchiamo di modificare o leggere il contenuto di quelle variabili all'esterno delle routine di interrupt.

Il problema deriva dal fatto che (come già accennato in precedenza) le variabili stepX sono a tre byte e non esistono istruzioni in grado di operare direttamente su tre byte. Il fatto di dover effettuare letture e/o modifiche in più passi separati può generare errori nel caso tra le diverse fasi avvenga un incremento della variabile interessata.

Vediamo ad esempio in Figura 38 cosa succede se durante la lettura di una variabile si verifica un interrupt che ne determina l'incremento.

PASSO 0	Valore iniziale stepX	00	FF	FF
PASSO 1	Lettura 2 byte meno significativi		**FF**	**FF**
PASSO 2	*Chiamata interrupt*			
PASSO 3	*Incremento stepX*	01	00	00
PASSO 4	*Fine gestione interrupt*			
PASSO 5	Lettura byte più significativo	**01**		
PASSO 6	Risultato lettura	**01**	**FF**	**FF**

Figura 38 Esempio problema lettura variabile StepX

Tale problema è sicuramente raro, ma un suo verificarsi porterebbe a gravi conseguenze in quanto si commetterebbe un errore di oltre 65000 passi.

Problemi analoghi si hanno quando si modifica il contenuto della variabile.

Per ovviare a questi problemi vengono adottate due tecniche diverse.

Se si deve semplicemente azzerare una variabile, è sufficiente azzerare per primo il byte meno significativo, in modo che un eventuale incremento intermedio non interferisca con i byte più significativi.

Per tutti gli altri casi di modifica e lettura delle variabili viene temporaneamente disabilitato l'incremento all'interno delle routine di gestione degli interrupt. Questo viene fatto mediante l'uso di una variabile ausiliaria *no_inc* come già anticipato nel Paragrafo 10.3.

10.7. Osservazioni sul testing

Il monitor debugger PCBUG11 necessita di molte risorse per poter eseguire programmi on-line; tra le risorse richieste vi sono la porta di comunicazione seriale, alcuni interrupt e parecchia memoria RAM. A causa di queste forti limitazioni il suo impiego è stato circoscritto al testing di piccoli programmi scritti appositamente per controllare l'esattezza di alcune parti di routine.

Per il testing delle routine complete e dell'interazione delle varie routine abbiamo utilizzato frequentemente la comunicazione seriale. Abbiamo modificato alcune routine, in modo da spedire pacchetti alla CCCU contenenti dati tali da permetterci di rilevare eventuali errori o malfunzionamenti.

11. Divisione in più versioni.

11.1. Problemi di spazio

Il programma completo (2.0) [File_Marmot] occupa circa 3 KByte, purtroppo la EEPROM a bordo del microcontrollore è di soli 2 KByte. Abbiamo quindi diviso il programma in tre versioni (2.1, 2.2 e 2.3) [File_MarmotC], [File_MarmotR], [File_MarmotO], cercando di distribuire uniformemente le funzionalità.

In tutte le versioni è stata mantenuta la stessa architettura generale, modificando solo la routine di gestione dei pacchetti elaborati, le relative routine e naturalmente le locazioni di memoria contenenti la versione installata.

Se viene ricevuto un pacchetto non gestito dalla versione correntemente installata, viene mandata una relativa segnalazione di errore come spiegato successivamente (Paragrafo 11.2.1).

Vediamo nella Tabella 3 le caratteristiche delle varie versioni

Caratteristiche	**Versione completa**	**Versione vel costante**	**Versione rampa**	**Versione old pkt**
Nome File	Marmot.ASC	MarmotC.ASC	MarmotR.ASC	MarmotO.ASC
Nome Versione	MARMOT MOVER 2.0	MARMOT MOVER 2.1	MARMOT MOVER2.2	MARMOT MOVER 2.3
Breve spiegazione pacchetti elaborati	Tutti	Riguardanti gli spostamenti a velocità costante	Riguardanti gli spostamenti mediate rampe	Tutti quelli contenuti nella prima versione del programma

Tabella 3 Informazioni sulle versioni

11.2. Pacchetti gestiti nelle varie versioni

Vediamo in Tabella 4 quali sono i *pacchetti di comando* accettati dalle varie versioni.

Pacchetti	Versione completa	Versione vel costante	Versione rampa	Versione old pkt
SETSPEED	SI	NO	SI	SI
GETSPEED	SI	SI	NO	SI
GETADC	SI	NO	NO	SI
SETMOTORS	SI	NO	NO	SI
GETMOTORS	SI	SI	NO	SI
SLEEP	SI	SI	NO	NO
SETCONSTSP	SI	SI	NO	NO
WAKEUP	SI	SI	NO	NO
AFTERSTEP	SI	NO	SI	NO
GETSTEP	SI	SI	NO	NO
STATUS	SI	SI	SI	SI
SELFTEST	SI	SI	SI	SI

Tabella 4 Dettaglio pacchetti gestiti nelle versioni

11.2.1. Pacchetto NOTTHISVER

Pacchetto trasmesso dalla MCU alla CCCU per segnalare la ricezione di un pacchetto non gestibile dalla versione attualmente installa sul robot. La struttura del pacchetto è mostrata in Figura 39

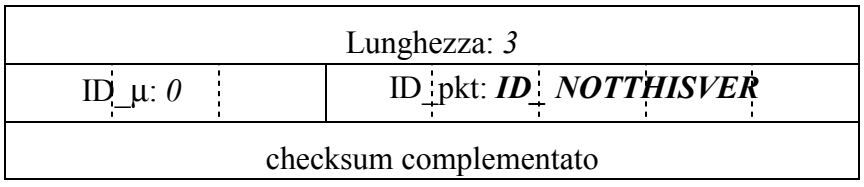


Figura 39 Struttura del pacchetto NOTTHISVER

11.2.2. Modifiche locali delle routine

La mancanza di alcune routine nelle versioni non complete (2.1, 2.2 e 2.3), ha portato a doverne modificare altre rispetto alle originali inserite nella versione completa.

Ad esempio nella versione velocità costante è stata modificata la routine *checkconn*, infatti l'arresto delle ruote non può avvenire con una decelerazione dato che manca la routine *aggiornarampa* che gestisce appunto le rampe di decelerazione.

Tutte le differenze rispetto alla versione completa sono segnalate nel listato del codice sorgente [File_MarmotC], [File_MarmotR], [File_MarmotO].

12. Programma CCCU

La gestione ad alto livello del robot è affidata ad un pc che colloquia con la MCU attraverso la porta seriale. La comunicazione avviene, come già spiegato, grazie a un protocollo a pacchetti.

Per testare le funzionalità implementate sulla MCU è stato creato un programma da compilare ed eseguire su una macchina *linux,* per gestire la comunicazione tra CCCU e MCU. Tale programma deve occuparsi della corretta spedizione e ricezione dei pacchetti [File_CCCU].

L'interfaccia utente è un semplice dialogo testuale, che permette di inserire l'identificativo del pacchetto che si desidera inviare con i relativi dati e consente inoltre di ricevere dati dalla MCU e presentarli all'utente. L'interfaccia testuale presenta sicuramente problemi sia estetici che di utilizzo, ma il nostro unico scopo era quello di avere la possibilità di dialogare con la MCU e poter così testare i nostri programmi installati sul robot. Non abbiamo quindi dato troppa importanza (e quindi dedicato troppo tempo) alla "forma" con cui abbiamo raggiunto questo scopo.

12.1. Problemi per pacchetti "asincroni"

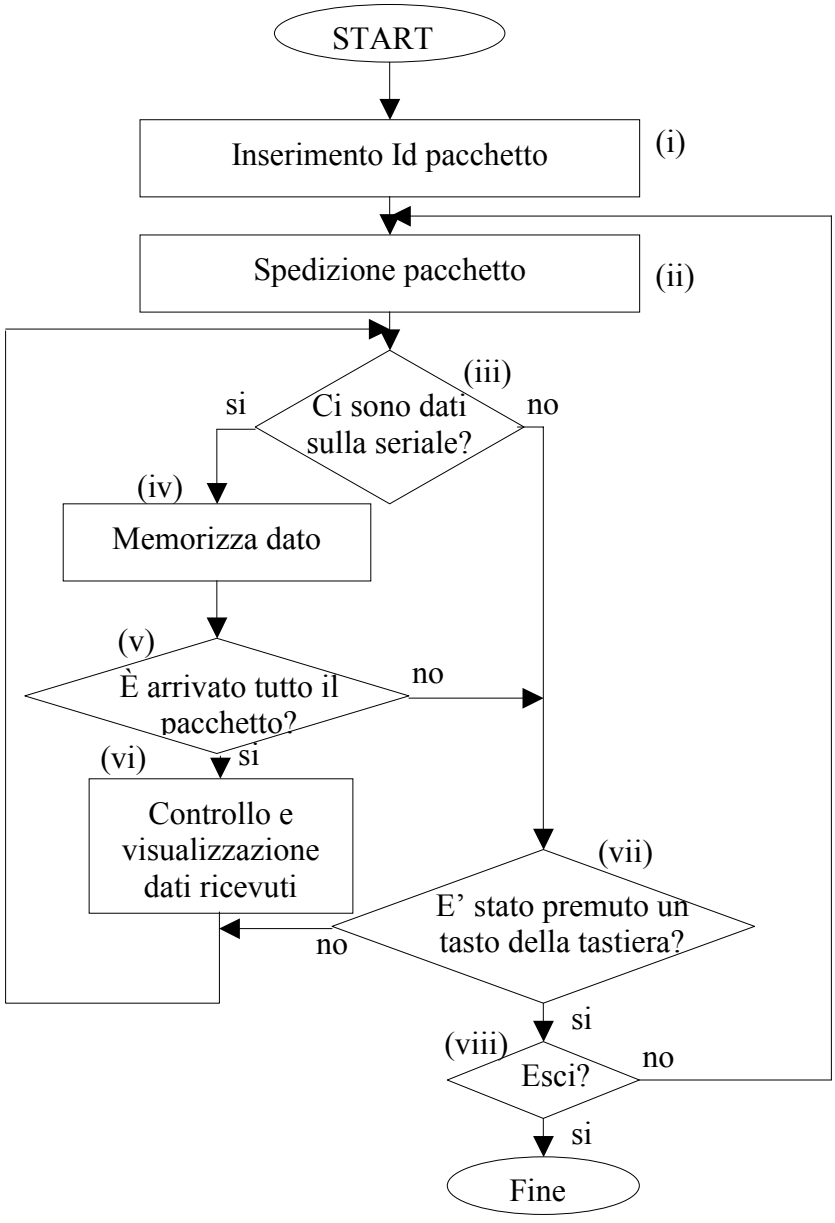
Il nostro lavoro ha preso spunto da quello svolto dai ragazzi che hanno implementato la prima versione del programma di movimentazione, abbiamo cioè sfruttato parte del codice che loro avevano scritto per poter testare le loro routine [File_CCCU_R_S].

La differenza fondamentale tra il nostro lavoro e il loro sta nella possibilità di ricevere pacchetti in qualunque momento. Nel loro programma, dopo aver spedito un pacchetto, si restava in attesa di uno di conferma della ricezione o di uno contenente i dati eventualmente richiesti. Noi non possiamo adottare una strategia di questo tipo, infatti i pacchetti di raggiungimento del goal e di caduta della connessione possono arrivare in qualunque istante (da qui la definizione di asincroni). Questo impone che il programma sia praticamente sempre "in ascolto" sulla porta seriale.

Ciò viene risolto, come si può notare dal flowchart illustrato in Figura 40, continuando a eseguire la routine di controllo della presenza di dati sulla seriale fino a quando l'utente non preme un tasto.

12.2. Struttura programma

Presentiamo in Figura 40 il flow chart del programma eseguito sulla CCCU.



(i) Viene richiesto all'utente di inserire il numero corrispondente al pacchetto che si vuole spedire.

(ii) Viene spedito il pacchetto tramite porta seriale.

(iii) Viene controllata l'eventuale presenza di dati sulla porta seriale. Se sono giunti dei byte, devono essere memorizzati (iv); se invece non è arrivato nulla, viene controllato se c'è stata una nuova richiesta di spedizione di un pacchetto da parte dell'utente (vii).

(v) Viene controllato se è arrivato tutto il pacchetto, nel caso ciò non sia accaduto si controlla se c'è stata una nuova richiesta di spedizione di pacchetto da parte dell'utente (vii).

(vi) Se invece il pacchetto è arrivato tutto, viene controllato il checksum e vengono visualizzati i dati contenuti.

(vii) Viene controllato se è stato premuto un tasto dall'utente per eseguire una nuova azione, in caso negativo si torna al controllo dello stato della seriale (iii).

(viii) Se invece è stato premuto un tasto, si verifica se l'utente desidera terminare la comunicazione (è stato inserito un numero superiore a 12).

Se l'identificatore del pacchetto inserito è valido, si passa alla spedizione del pacchetto in questione (ii); altrimenti l'esecuzione termina.

Figura 40 flowchart programma CCCU

12.3. ID comandi

Le istruzioni che l'utente può eseguire (corrispondenti all'invio di altrettanti pacchetti) sono codificate da un numero (ID), come illustrato in Tabella 5. Quando l'utente desidera inviare al robot un comando, deve inserire da tastiera l'ID corrispondente e successivamente, se necessario, i dati (passi, periodi, alimentazione dei motori…).

Pacchetto da spedire	ID da inserire	Dati da inserire		
		periodi	passi	settaggio alimentazione
SETSPEED	1	p1 p2 p3	-	-
SETCONSTSP	2	p1 p2 p3	s1 s2 s3	-
SLEEP	3	-	-	-
WAKEUP	4	-	-	-
GETSTEP	5	-	-	-
AFTERSTEP	6	-	s1 s2 s3	-
SELFTEST	7	-	-	-
GETSPEED	8	-	-	-
GETADC	9	-	-	-
GETMOTORS	10	-	-	-
SETMOTORS	11	-	-	Alim
STATUS	12	-	-	-

Tabella 5 ID comandi del programma su CCCU

13. Errata Corrige versione 1.0

Di seguito segnaliamo alcuni errori o imprecisioni che abbiamo rilevato dall'analisi della documentazione della prima versione.

13.1. Modalità funzionamento schede driver dei motori

Pag. 13, Figura 7: lo schema che rappresenta le schede driver contiene una tabella che illustra le posizioni degli switch che permettono di selezionare la modalità di funzionamento del motore a passo; tale tabella, riportata nella Tabella 6, è errata.

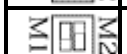
	M1	M2
	1	1
	0	1
	1	0
	0	0

Tabella 6 Posizioni errate switch schede driver

Riportiamo nella Tabella 7 le posizioni corrette.

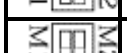
	M1	M2
	0	0
	1	0
	0	1
	1	1

Tabella 7 Posizioni corrette switch schede driver

13.2. Inversione direzione pacchetti di segnalazione

Pag. 33, II paragrafo: i pacchetti di segnalazione vengono inviati soltanto dalla MCU alla CCCU e non viceversa come indicato.

13.3. Utilizzo SETMOTORS

Non viene dato nessun suggerimento in merito all'utilizzo del pacchetto SETMOTORS, ma tale pacchetto potrebbe influire negativamente sull'esecuzione di altre funzionalità, è consigliabile quindi utilizzarlo solo a robot fermo.

13.4. Spedizione semiperiodi

Sia per quanto riguarda il pacchetto SETSPEED che quello GETSPEED, si fa riferimento ai *periodi* spediti o ricevuti. In realtà ciò che viene spedito o ricevuto sono i *semiperiodi.*

In merito alla conversione da dato inviato/ricevuto a frequenza del segnale, notiamo che i risultati sono corretti; questo perché è stata considerata una risoluzione minima del timer di 1μs, mentre in realtà tale risoluzione minima è di 0.5μs. Così mancando sia una moltiplicazione che una divisione per due, il risultato finale è corretto.

Vediamo un esempio: per generare un segnale a 2.5KHz è necessario richiedere un periodo di 1/2.5KHz = 400 μs ossia un semiperiodo di 200 μs, perciò tenendo presente la risoluzione di 0.5 μs il valore da impostare risulta essere 200 μs/ 0.5 μs = 400. Considerando di spedire i periodi e usare una risoluzione di 1μs, il risultato è il medesimo.

Per coerenza abbiamo deciso di modificare internamente il programma installato sulla MCU in modo da ricevere realmente i periodi e non i semiperiodi. I valori da spedire vanno quindi raddoppiati per avere una velocità identica a quella che si otteneva con la vecchia versione. Conseguentemente sono stati modificati i periodi minimi e massimi impostabili, chiaramente sono stati raddoppiati. Il valore che convenzionalmente segnala il robot fermo non è stato semplicemente raddoppiato, ma è stato portato a "periodo massimo + 1" (come accadeva anche nella versione precedente), cioè a 32001. Per i motivi già esposti, a fronte di variazioni dei valori massimi e minimi impostabili, le frequenze massime e minime dei segnali che vengono generati rimangono le stesse.

Visto che internamente al microcontrollore i periodi ricevuti vengono immediatamente divisi per due, è consigliabile spedire solo valori pari, in quanto un valore dispari verrebbe automaticamente approssimato al valore pari più vicino per difetto.

14. Possibili sviluppi futuri

Sicuramente il software per la CCCU da noi prodotto si presta, come già detto, a notevoli miglioramenti. Tali miglioramenti possono riguardare sia l'interfaccia grafica sia la possibilità di elaborare operazioni più complesse delle semplici richieste di spedizione dei vari pacchetti.

Per quanto riguarda il "lato MCU", sarebbe utile cercare di migliorare le operazioni di testing. Per fare questo però è necessario avere dispositivi hardware che permettano una migliore identificazione dell'effettivo stato del robot e dei vari malfunzionamenti.

Ad esempio encoder o finecorsa montati sulle ruote permetterebbero di effettuare un migliore (e più realistico) controllo del movimento delle ruote. Infatti attualmente se per qualche motivo i motori "perdono passi", non ce ne accorgiamo.

Inoltre l'utilizzo di una scheda con una EEPROM con circa 3 Kbyte di memoria permetterebbe l'istallazione della versione completa (2.0), il che consentirebbe di sfruttare tutti i pacchetti e non solo quelli gestiti dalle versione correntemente caricata.

Allegati

[File_Marmot] File *Marmot.ASC*

[File_MarmotC] File *MarmotC.ASC*

[File_MarmotR] File *MarmotR.ASC*

[File_MarmotO] File *MarmotO.ASC*

[File_CCCU] File *CCCU.C* contiene il listato del programma CCCU

Bibliografia

[ROS_SIG01] S. Rosa, S. Sigala *Movimentazione del robot Marmot* versione 1

[TA8435H] Data sheet of pwm chopper type bipolar stepping motor driver

[AFMICRO00] Prof.ssa Alessandra Flammini, *Dispensa per il corso di Elettronica Dei Sistemi Digitali*, CLUB Brescia

[M68HC11RM/D] M68HC11 Reference Manual Rev. 4.0

[M68HC11E/D] M68HC11E Family Technical Data Rev. 3.0

[M68PCBUG11] M68HC11 PCbug11 User's Manual

[File_CCCU_R_S] S. Rosa, S. Sigala Listato *programma CCCU* relativo alla prima versione

Tabelle

Tabella 1 Identificatori pacchetti 23

Tabella 2 Casi di overflow 27

Tabella 3 Informazioni sulle versioni 35

Tabella 4 Dettaglio pacchetti gestiti nelle versioni 36

Tabella 5 ID comandi del programma su CCCU 40

Tabella 6 Posizioni errate switch schede driver 41

Tabella 7 Posizioni corrette switch schede driver 41

Figure

Figura 1: struttura del pacchetto SETCONSTSP 9
Figura 2 risposta al pacchetto SETCONSTSP 9
Figura 3 raggiungimento obiettivo impostato con SETSPCONST 10
Figura 4 struttura del pacchetto SLEEPSPCONST 10
Figura 5 struttura del pacchetto di risposta SLEEPSPCONST 10
Figura 6 struttura del pacchetto BADSLEEP 11
Figura 7 struttura del pacchetto WAKEUPSPCONST 11
Figura 8 struttura del pacchetto di risposta WAKEUPSPCONST 11
Figura 9 struttura del pacchetto BADWAKEUP 12
Figura 10 struttura del pacchetto GETSTEP 13
Figura 11 risposta al pacchetto GETSTEP 14
Figura 12 Pacchetto AFTERSTEP 14
Figura 13 struttura del pacchetto di risposta AFTERSTEP 15
Figura 14 struttura del pacchetto BADAFTERSTEP 15
Figura 15 struttura del pacchetto M1GOAL 16
Figura 16 struttura del pacchetto M2GOAL 16
Figura 17 struttura del pacchetto M3GOAL 16
Figura 18 Diagramma stati finiti 18
Figura 19 Struttura pacchetto GETSTATUS 19
Figura 20 Struttura del pacchetto di risposta a GETSTATUS 19
Figura 21 Significato dei bit del byte *Status* 19
Figura 22 struttura del pacchetto SELFTEST 20
Figura 23 risposta al pacchetto SELFTEST 21
Figura 24 Ciclo del programma 22
Figura 25 Significato dei bit della variabile *state* 24
Figura 26 Significato dei bit della variabile stepbits 25
Figura 27 Significato dei bit della variabile goalstep 25
Figura 28 Significato dei bit della variabile no_inc 25
Figura 29 flowchart routine *checkstep* 26
Figura 30 Esempio caso A overflow 27
Figura 31 Esempio caso B overflow 27
Figura 32 Esempio caso C overflow 28

Figura 33 Esempio caso D overflow 28
Figura 34 Flow chart routine *setconstspd* 29
Figura 35 Flow chart routine *afterstep* 30
Figura 36 flow chart routine s*elftest* 31
Figura 37 Modifiche alla routine *setspddir* 32
Figura 38 Esempio problema lettura variabile StepX 33
Figura 39 Struttura del pacchetto NOTTHISVER 36
Figura 40 flowchart programma CCCU 39

Indice

SOMMARIO **1**

1. INTRODUZIONE **1**

2. IL PROBLEMA AFFRONTATO **2**

3. LA SOLUZIONE ADOTTATA **3**

4. NOTE AL LETTORE **4**

5. VALUTAZIONE SPOSTAMENTO **5**

5.1. IL RAGGIUNGIMENTO DELL'OBIETTIVO (GOAL) 5
5.1.1. Diversi modi di identificazione del goal *5*
5.1.2. Implementazioni diverse in base alle funzionalità *5*
5.2. CONTEGGIO DEI PASSI 6
5.2.1. Dimensionamento del contatore *6*
5.2.2. Valutazioni riguardanti il tempo di servizio delle routine di interrupt7
5.2.3. Gestione degli overflow *7*

6. POSIZIONAMENTO A VELOCITÀ COSTANTE **8**

6.1. INTRODUZIONE 8
6.2. PACCHETTO SETCONSTSP 8
6.2.1. Pacchetto GOALSETSPCONST *10*
6.3. PACCHETTO SLEEPSPCONST 10
6.3.1. Pacchetto BADSLEEP *11*
6.4. PACCHETTO WAKEUPSPCONST 11
6.4.1. Pacchetto BADWAKEUP *12*

7. INFORMAZIONI SULLO SPOSTAMENTO EFFETTUATO **13**

7.1. INTRODUZIONE 13
7.2. PACCHETTO GETSTEP 13
7.3. PACCHETTO AFTERSTEP 14
7.3.1. Pacchetto BADAFTERSTEP *15*
7.4. PACCHETTI M1GOAL, M2GOAL, M3GOAL 16

8. CONTROLLO DELLO STATO **17**

8.1. DIAGRAMMA A STATI FINITI 17
8.2. PACCHETTO GETSTATUS 19

9. SELFTEST **20**

9.1. INTRODUZIONE 20
9.2. PACCHETTO SELFTEST 20

10. PROGRAMMA MCU **22**

10.1. STRUTTURA DEL PROGRAMMA 22
10.2. IDENTIFICATORI PACCHETTI 23
10.3. USO DI VARIABILI DI STATO 24
10.4. ROUTINE CHECKSTEP 26
10.4.1. Controllo dell'overflow 27
10.5. ALTRE ROUTINE 28
10.5.1. Setconstspd 29
10.5.2. Afterstep 30
10.5.3. Selftest 31
10.5.4. Setspddir 32
10.6. MODIFICA DELLE VARIABILI STEPX 33
10.7. OSSERVAZIONI SUL TESTING 34

11. DIVISIONE IN PIÙ VERSIONI. 35

11.1. PROBLEMI DI SPAZIO 35
11.2. PACCHETTI GESTITI NELLE VARIE VERSIONI 36
11.2.1. Pacchetto NOTTHISVER 36
11.2.2. Modifiche locali delle routine 37

12. PROGRAMMA CCCU 38

12.1. PROBLEMI PER PACCHETTI "ASINCRONI" 38
12.2. STRUTTURA PROGRAMMA 39
12.3. ID COMANDI 40

13. ERRATA CORRIGE VERSIONE 1.0 41

13.1. MODALITÀ FUNZIONAMENTO SCHEDE DRIVER DEI MOTORI 41
13.2. INVERSIONE DIREZIONE PACCHETTI DI SEGNALAZIONE 41
13.3. UTILIZZO SETMOTORS 42
13.4. SPEDIZIONE SEMIPERIODI 42

14. POSSIBILI SVILUPPI FUTURI 43

ALLEGATI 44

BIBLIOGRAFIA 44

TABELLE 44

FIGURE 45

INDICE 47