DOTTORATO DI RICERCA IN
INGEGNERIA DELL'INFORMAZIONE
XX Ciclo

Sede Amministrativa:
UNIVERSITÀ DEGLI STUDI
DI FERRARA

Tesi per il conseguimento del titolo di
Dottore di Ricerca:

Progetto e realizzazione di una piattaforma scalabile per la robotica mobile

Candidato: ENRICO PICCININI

Tutore: Prof. SERGIO BEGHELLI

Coordinatore del Corso: Prof. STEFANO TRILLO

Grazie Sergio per il tuo impegno di docente; il tuo esempio mi è servito a crescere non solo come professionista ma anche come uomo.

Grazie Marcello per avermi dedicato parte del tuo tempo libero in nome della passione, dell'impegno professionale e dell'amicizia

Grazie a tutti voi del LIRA che mi avete accolto e fatto sentire un po' a casa anche durante le ore di lavoro

Mamma e papà grazie sempre e di tutto

Questa bellissima avventura è per mia moglie Stefania che ha creduto in me più di quanto abbia fatto io stesso e mi accompagna in una vita che probabilmente non è sempre così tranquilla. Ai miei figli Giovanni e Gabriele dedico le mie ore di "non gioco" con loro, spese nell'intento che questo mio impegno sia servito almeno un po' a "lasciare il mondo un po' migliore di come l'ho trovato" [Baden Powell].

Sommario

La presente tesi di dottorato analizza alcune nuove tecnologie della scienza dell'informazione come possibili scelte al fine di realizzare una piattaforma robotica mobile. Viene considerata la possibilità di estendere i concetti applicati alla piattaforma robotica mobile, a uno scenario di controllo dove le unità di elaborazione sono dislocate su una rete di telecomunicazione locale, distribuita, wired e wifi.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

1.1 Obiettivi di questa tesi

L'obiettivo principale dell'attività svolta nel corso del dottorato di ricerca, è stato quello di realizzare una piattaforma mobile con particolare attenzione a due aspetti: **didattica** e **ambito applicativo**. Per quanto riguarda la didattica, fondamentale è stata la scelta di sviluppare internamente la maggior parte dei componenti piuttosto che acquisirli da terze parti; questa scelta ha favorito la completa conoscenza di ogni singola parte del mobot, sia essa meccanica, hw o sw, lasciando aperta ogni strada di sviluppo futuro e di replica del sistema. Il risultato è un dispositivo su cui è possibile condurre attività di ricerca nell'ambito della robotica mobile e dell'intelligenza artificiale, permettendo agli studenti di fare esperienze nel campo dei controlli automatici a svariati livelli di sviluppo: sw,fw,hw e meccanico. Per quanto riguarda l'ambito applicativo si è prestata particolare attenzione a due settori di impiego di questi dispositivi: i mobot di servizio (i.e carrelli porta oggetti, agenti sorveglianti,...) e sistemi orientati all'assistenza alle persone diversamente abili. Più in dettaglio vediamo i componenti che maggiormente hanno caratterizzato lo sviluppo del dispositivo SabotOne.

- **Il nome SabotOne** Sabot è l'acronimo di Scalable mobot (i.e mobile robot). Il principio guida del design di ogni singolo componente è stato quello di realizzare un sistema modulare che consentisse l'espandibilità del mobot.

- **Meccanica** È stato eseguito il progetto delle parti meccaniche del SabotOne modellando tutto il sistema con CAD 3D che è poi stato realizzato in alluminio presso una azienda di lavorazione meccanica. Le piattaforme sono modulari, prevedono l'alloggiamento dell'elettronica nei livelli inferiori destinando i livelli superiori ad impieghi specifici dipendenti dal settore di utilizzo del mobot. La cinematica del modello realizzato, rispecchia quella dell'uniciclo che favorisce la mobilità del dispositivo anche in spazi ridotti e risulta molto simile a quella delle carrozzine per disabili.

- **Elettronica** È stata realizzata una scheda assi per il controllo in loop di posizione dei due motori in corrente continua impiegati nella trazione del mobot. È stato realizzato il firmware di controllo e progettati i protocolli di comunicazione al fine di rendere tale scheda idonea all'applicazione di robotica in cui veniva utilizzata. Si sono estese le funzionalità della board al fine di dotarla di un sistema acquisizione di segnali sensoriali (A/D converter, GPIO). La si è equipaggiata con tre protocolli di comunicazione: CAN, TCP/IP (su

802.11b/g, RJ45) e modbus su RS485. Sull'interfaccia ethernet la scheda espone un web server per la configurazione dei parametri di bordo, mentre mediante la seriale può scambiare informazioni con una unità intelligente quale ad esempio un embedded-pc montante un sistema operativo.

- **Software**
  - **Sviluppo dello SCADA Sabot Workbench** È un tool che utilizzato per il monitoraggio remoto dei mobot connessi ad una rete. È stato sviluppato utilizzando il moderno framework Java Eclipse-RCP. Tale framework ha permesso la modellazione di tutte le funzionalità dello SCADA attraverso il paradigma del *plugin*. Questo consente di aggiungere delle funzionalità all'applicativo senza doverne modificare il core. Dato che attualmente questo risulta essere il primo applicativo di controllo di mobot sviluppato con questa tecnologia (orientata alla scalabilità), si intende proporre il lavoro svolto alla prossima conferenza internazionale della Eclipse Foundation.
  - **Design del protocollo di comunicazione tra Workbench e i Sabot** Si basa sul protocollo Jabber/XMPP e la specifica SensML/SWE. Si è progettata l'integrazione tra XMPP (usato come information carrier) e SensML/SWE che definisce le ontologie per la descrizione di fenomeni misurabili e i relativi sensori per il rilevamento. È stato definito il *concetto* di identità di un sistema controllato e integrato all'interno del protocollo XMPP. Questa attività è stata segnalata alla OGC (Open Geospatial Consortium) e alla University of Alabama (creatori del SensML) in quanto risulta essere la prima implementazione di SensML/SWE su XMPP.
  - **Sviluppo degli algoritmi di trajectory control, path planning e autolocalization** È stato sviluppato il path planning implementando in C++ l'algoritmo LPN. È stato costruito un simulatore per verificare l'efficienza e la completezza dell'algoritmo. Si è visto che l'algoritmo è completo (riesce sempre a trovare una traiettoria verso un target qual'ora questa esista) ed è efficiente (pianificazione di una traiettoria in meno di 100ms su uno spazio partizionato in 10.000 celle). Il trajectory control è stato realizzato implementando due algoritmi differenti; il primo basato sulla pseudoinversa del sistema a vincoli non olonomi, il secondo realizzato mediante la *dynamic feedback linearization*. È stato costruito un simulatore per verificare l'errore di inseguimento dei controllori e si è visto che entrambe i controllori riescono ad annullare l'errore di inseguimento. Per l'autolocalizzazione è stata acquisita una piattaforma inerziale sulla quale è stato implementato un filtro di kalman per la "fusione" dei vari dati sensoriali (accelerometri, giroscopi e magnetometri). Si è visto che i dati inerziali dovranno essere ulteriormente "fusi" con riferimenti di posizione assoluti (feature estratte da immagini o tag RFID) a causa delle derive intrinseche dei sensori inerziali.
  - **Architettura sw sull'embedded PC a bordo del Sabot** È stato realizzato un abstaction layer object oriented per svincolare il livello applicativo dal S.O adottato (linux). Si è sviluppata una architettura in grado di realizzare il concetto di run-time plugin mediante il lazy-loading delle librerie in linux. In questo framework si stanno integrando gli algoritmi di pianificazione delle traiettorie, di autolocalizzazione e di obsatcle avoidance. Si stanno inoltre integrando il protocollo di comunicazione XMPP, il protocollo di comunicazione verso la scheda assi e tutta la logica di bordo del mobot.

Il framework è stato progettato in linguaggio UML applicando strategie orientate agli oggetti e i criteri di “Design Pattern” e “Real time design pattern”

Parte I

Architettura di sistema

Capitolo 2

Architettura di sistema

2.1 Scelte progettuali e relative giustificazioni

Il software è stato sviluppato in due linguaggi: C++ e Java. In entrambe i casi si è scelto di adottare linguaggi object oriented per disporre degli strumenti che potessero modellare al meglio il mondo reale. Il C++ è stato utilizzato per lo sviluppo di tutti quei componenti "faceless" (i.e senza una controparte grafica) che hanno come come requisiti principali il tempo di esecuzione e la compattezza in memoria. D'altro canto Java offre un'ottima integrazione tra API di sistema (i.e I/O, thread handling,...) e UI-API; è stato pertanto utilizzato nello sviluppo di tutti quei componenti che hanno come requisito principale la necessità di integrare strutture dati con una controparte grafica. Come si vedrà nel seguito le unità software dell'uno e dell'altro linguaggio condividono parte dei loro dati, ma normalmente risiedono su elaboratori separati. Si è pertanto scelto che la condivisione dei dati dovesse avvenire mediante un protocollo di comunicazione piuttosto che utilizzando delle sovrastrutture sw come *CORBA*. L'uso di quest'ultima tecnologia è stato ulteriormente disincentivato dalla complessità architetturale; il suo utilizzo avrebbe infatti richiesto la messa in esecuzione di servizi di mediazione (i.e ORB) su tutti gli apparati che devono comunicare nella rete e la generazione di stub di interfaccia costruite con semantiche ad-hoc. Queste inefficienze strutturali del sistema CORBA rendono di difficile portabilità il codice che si interfaccia con esso e avrebbe richiesto una strutturazione del sw basata sulla separazione dei processi che mantenga separati il layer CORBA da tutto il resto del software.

2.2 Architettura hardware di bordo

A bordo di ogni mobot è installata una unità di elaborazione centrale denominata SSB (*Sabot System manager Board*) che ha il compito di eseguire gli algoritmi di path-planning, trajectory tracking, obstacle avoidance oltre che quello di gestire tutta la comunicazione verso il server di controllo e la diagnostica di bordo. Per una descrizione dettagliata di questa unità si rimanda al capitolo 3. Ogni sensore (o gruppo di sensori) viene *virtualizzato* mediante la propria elettronica di condizionamento che svolge una preelaborazione al fine di fornire ad SSB una informazione di più alto livello. In particolare attualmente si contraddistinguono i seguenti *sensori virtuali*:

1. **Obstacle Detection Sensor** Gruppo di 3 sensori ad infrasuoni montati su motori servo che mediante un movimento rotatorio rilevano la presenza di ostacoli nelle vicinanze del mobot. Tali sensori forniscono alla scheda SSB le coordinate rispetto al centro di massa

del mobot di eventuali ostacoli. Questi sensori sono controllati da una scheda di controllo dedicata che esegue l'acquisizione e il condizionamento dei dati dai sensori, esegue l'attuazione sui motori servo ed esegue le elaborazioni dei dati al fine di ottenere la posizione dell'ostacolo rispetto al centro di massa del mobot.

2. **Inertial Navigation Sensor** Gruppo di 3 accelerometri, 3 giroscopi, 3 magnetometri e un GPS che forniscono i dati di navigazione inerziale alla scheda SSB. L'uso di queste informazioni verrà spiegato più in dettaglio nel capitolo 6 (pag.49). Anche questi sensori posseggono una elettronica dedicata costituita da:

   - Scheda a microcontrollore (ATMega128) per l'acquisizione e il condizionamento dei segnali di tutti i sensori inerziali
   - Scheda a microprocessore montante un PXA255 con Linux a bordo su cui viene effettuato un algoritmo di filtraggio (*Extended Kalman Filter*) per la fusione di tutti i dati dei sensori acquisiti dalla board di acquisizione.

   L'uscita del *Kalman Filter* è la stima dell'assetto e della posizione del mobot. Tali dati vengono resi disponibili all'algoritmo di autolocalizzazione in esecuzione sulla scheda SSB con lo scopo di correggere la deviazione sulla stima della posizione assoluta che si avrebbe utilizzando esclusivamente una procedura odometrica.

Categoria affine a quella dei sensori virtuali, è quella degli attuatori virtuali. Questi, analogamente ai sensori virtuali, posseggono una elettronica di virtualizzazione che è chiamata SACT (*Sabot Actuators board*) e verrà descritta nel dettaglio nel capitolo 4 (pag.23). Ognuna di queste unità *intelligenti* dialoga con la scheda SSB attraverso il protocollo modbus su seriale punto-multi punto[1]. In figura 2.1 (pag.8) è riportato un deployment diagram della struttura di **SabotOne** .

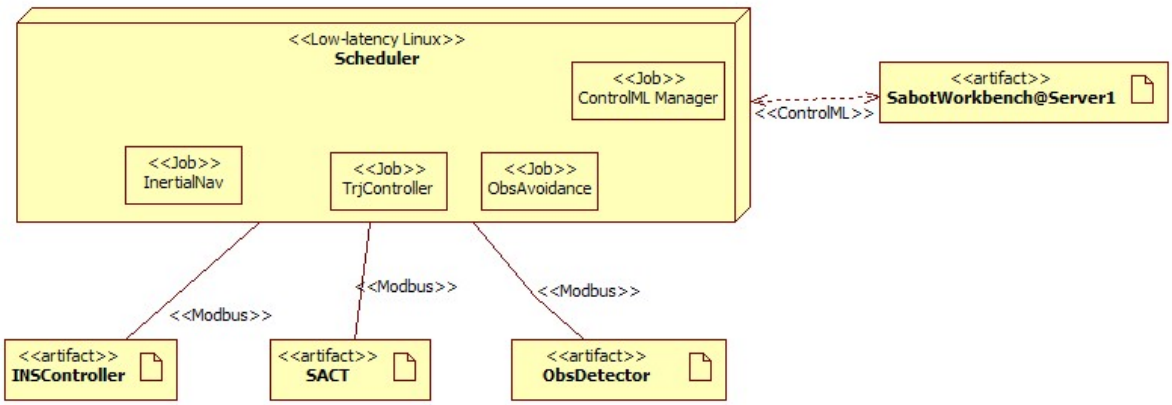

Figura 2.1: *Deployment diagram dell'architettura di* ***SabotOne*** *e di alcuni dei processi principali*

[1] Attualmente i line driver montati per il modbus sono per delle seriali RS232 per semplicità di debug con dei normali PC senza dover ricorrere a convertitori. Ogni scheda dialoga infatti su una porta seriale differente della scheda SSB

2.3 Protocolli di comunicazione

Il sistema è strutturato in due categorie principali di componenti:

- uno o più manager (i.e SCADA)
- uno o più **SabotOne**

Ogni **SabotOne** espone agli altri **SabotOne** e allo SCADA una interfaccia *ControlML* che verrà descritta nel dettaglio nei Capitoli 7 e 8 mediante la quale vengono inviati i comandi al mobot e ne viene rilevato lo stato. Lo SCADA è realizzato in Java e normalmente risiede sul PC di una postazione di controllo. In figura 2.2 a pagina 9 è riportata l'architettura della rete di interconnessione dei **SabotOne** e degli SCADA; il grafo dei link tra i dispositivi (per ragioni di chiarezza nella figura sono stati riportati solamente i principali), è strettamente connesso con relazioni bidirezionali realizzate mediante uno o più server *Jabber/XMPP* di dominio.

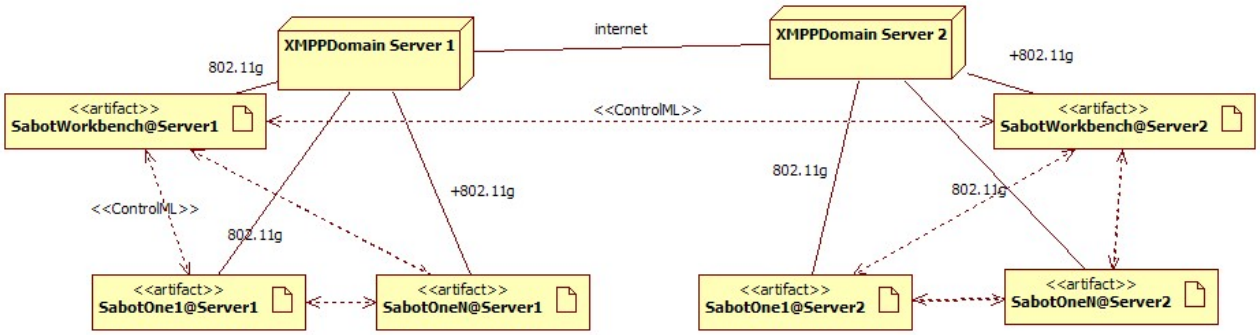

Figura 2.2: *Deployment diagram dell'architettura di rete. In tratto continuo sono visualizzati i link di comunicazione reale: tutti i dispositivi inviano i propri messaggi al proprio server di dominio (centro stella) il quale esegue il routing di tali messaggi verso il dispositivo a cui il messaggio stesso è destinato. Con linea tratteggiata è invece riportata la visibilità dei dispositivi dal punto di vista del protocollo Jabber/XMPP Ogni dispositivo può dialogare con ogni altro dispositivo della propria sottorete. Non riportati link che evidenziano la possibilità di ogni dispositivo di dialogare con ogni componente della sottorete con server di dominio Server2.*

In riferimento al deployment diagram riportato in figura 2.1 a pagina 8, ogni **SabotOne** è dotato di una unità intelligente di coordinamento dei vari sistemi di bordo su cui è installato un S.O Linux con kernel a bassa latenza. L'unità intelligente dialoga con i sottosistemi di bordo mediante un protocollo locale di campo punto-multi punto. Nel caso del **SabotOne** si è scelto di utilizzare un protocollo di comunicazione seriale di tipo Modbus-RS485 [gro02]. Le ragioni di questa scelta sono le seguenti

- Modbus: in quanto è un protocollo largamente diffuso e semplice da implementare. Non richiede controller hw dedicati (come il CAN), la specifica di tale protocollo è stata estesa per supportare connessioni di nuova generazione (rispetto alla data di nascita del protocollo che risale al 1979) come il TCP/IP e il SOAP.

- RS485: in quanto è una linea multipunto, il line driver è un componente di basso costo integrabile con semplicità anche sulle elettroniche più semplici (è infatti sufficiente che il microcontrollore disponga di una periferica UART).[2]

- Open: il protocollo è royalty-free ed è possibile reperire svariato materiale in rete come la specifica e svariate implementazioni in vari linguaggi.

In particolare sono stati implementati i due comandi Modbus 0x14 (Read file) e 0x15 (Write File) e le macchine a stati sono riportate nelle figure 2.3 a pagina 10 e 2.4 a pagina 11. La scelta di eseguire questa implementazione minimale incontra la necessità di mantenere il protocollo il più leggero possibile adatto ad essere eseguito anche su microcontrollori con scarse risorse.

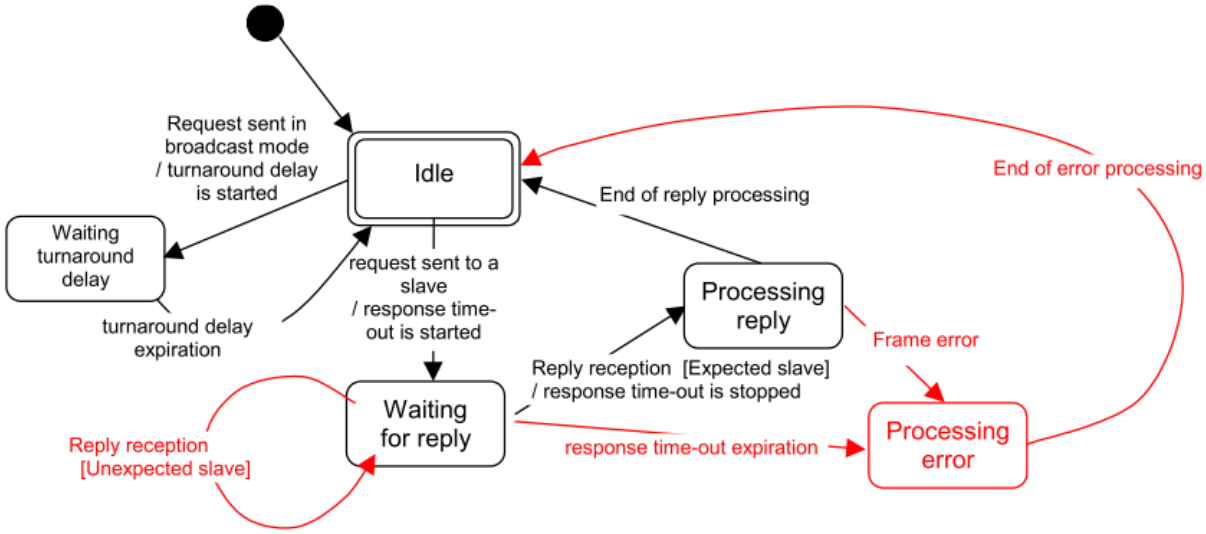

Figura 2.3: *Diagramma a stati del Modbus master in esecuzione sulla scheda SSB*

2.4 SabotOne Workbench

2.4.1 Tecnologia Eclipse Rich Client Platform

Come anticipato lo SCADA, denominato Sabot Workbench, è realizzato con tecnologia Java ed è pensato come strumento di interfaccia verso il/i **SabotOne** presenti nella sottorete. Per soddisfare al requisito di scalabilità del sistema si è pensato di utilizzare un framework di sviluppo chiamato Eclipse-RCP (Eclipse Rich Client Platform) che realizza in modo adeguato il concetto di run-time plugin. La tecnologia Eclipse-RCP si fonda infatti su una implementazione del framework OSGi R4[Jef06]. Questo definisce un framework di specifica, codifica, esecuzione e aggiornamento dei *componenti* di una applicazione. Tali *componenti* sono in realtà modellati

[2] Attualmente i line driver montati per il modbus sono per delle seriali RS232 per semplicità di debug con dei normali PC senza dover ricorrere a convertitori.

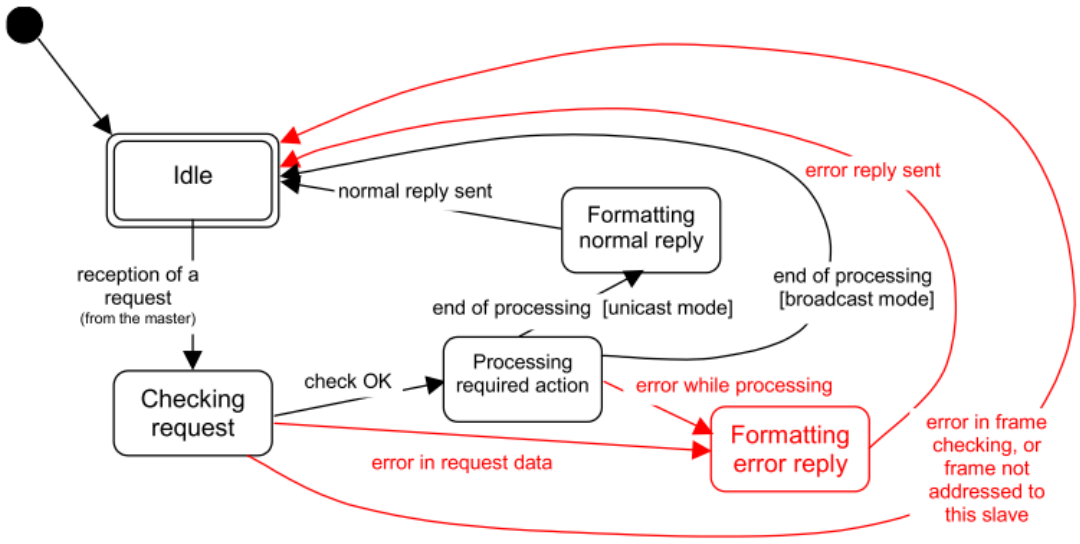

Figura 2.4: *Diagrammi a stati del Modbus slave in esecuzione sulle schede dei virtual sensor*

dal prototipo del *plugin* all'interno del framework RCP. Eclipse-RCP è costruito "sopra" due framework java noti con il nome di *SWT* e *JFace*[ROB04]. Il primo, a sua volta costruito sulle librerie grafiche standard di java AWT e SWING, fornisce il supporto per i così detti *rich controls*. Ossia fornisce una serie di controlli grafici come bottoni, viste, layout manager,... per la gestione delle interfacce grafiche "rich" di applicazioni di tipo "client". JFace fornisce invece un framework di sviluppo che implementa il modello MVC che fornisce un "modo d'uso" dei controlli di SWT utilizzati all'interno di una applicazione. In altri termini, in JFace l'uso dei controlli grafici viene quasi sempre mutuato da un pattern di progetto di tipo MVC implementato dal framework.

In Eclipse-RCP "*tutto è un plugin e il tutto può essere reso un plugin*"[Jef06]. Questa struttura è rappresentata in fig.2.5 (pag.12) in cui tutte le ellissi sono dei plugin e ognuno di questi plugin può concorrere a diventare parte di altri plugin o di una applicazione. Eclipse-RCP unisce a tutte le funzionalità di JFace e SWT un runtime che gestisce il ciclo di vita dell'applicazione. Si ha infatti che il run-time è una sorta di launcher per ogni applicazione rcp; questo esegue il caricamento dinamico di tutte le classi dell'applicazione, il caricamento dinamico di tutti i plugin di cui l'applicazione stessa è composta ed esegue tutti i metodi di inizializzazione dell'applicazione stessa prima di lanciarla. Ovviamente l'applicazione in RCP è costruita secondo un paradigma rigido sancito da una serie di classi di interfaccia di cui si deve eseguire un override ogni volta che si costruisce un nuovo plugin o una nuova applicazione.

Lo UI-plugin è il responsabile della gestione dei *punti di contribuzione* e dei *segnali* inviati dal'interfaccia grafica dell'applicazione al framework. I punti di contribuzione sono il *metodo strutturato* mediante il quale un plugin definisce il proprio legame con altri plugin e l'applicazione che lo contiene. Un esempio di contribuzione di un plugin potrebbe essere l'aggiunta di un particolare insieme di Tool ad un editor. I possibili punti di contribuzione in questo caso potrebbero essere: una nuova entry nella menu-bar della main application, un bottone nella tool-bar,.... I vantaggi dei punti di contribuzione sono:

- Modularità delle varie parti (plugin) dell'applicazione
- Non necessità di ricompilare tutta l'applicazione ogni volta che si desidera estendere le sue funzionalità
- Non necessità di conoscere il funzionamento dell'applicazione per poter contribuire ad essa; sarà necessario solamente stabilire come contribuire all'applicazione stessa ed implementare tutta la logica del plugin dentro al plugin stesso (ovviamente bisognerà conoscere le parti dell'applicazione che possono fornire o ricevere i dati dal plugin)

L'OSGi fornisce tutta la specifica al framework tra cui il concetto di plugin stesso, di *manifesto* e di metodo di caricamento del plugin. In modo particolare esso definisce anche il protocollo di aggiornamento dei plugin. Infatti la specifica definisce anche un sistema di aggiornamento di tutti i plugin mediante un sistema di repository che possono essere distribuiti su una rete http/ftp. Il creatore del plugin non dovrà fare altro che fornire in un file XML l'indirizzo del repository e un *manifesto* del repository e sarà cura del framework offrire una sorta di wizard per la gestione degli aggiornamenti dei plugin.

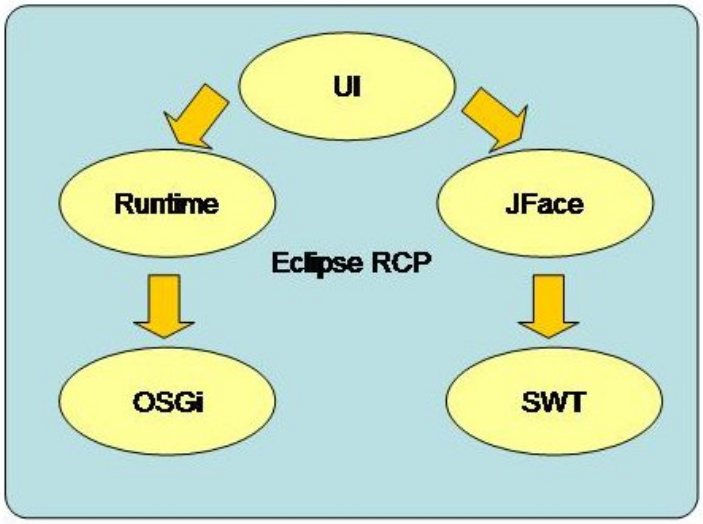

Figura 2.5: *Struttua del framework Eclipse-rcp*

L'Eclipse-RCP possiamo dire in breve che mette a disposizione le seguenti feaures

- Configurazione dell'applicativo client mediante un sistema di file XML: viene cioè deciso a tempo di esecuzione l'aspetto e le funzionalità che avrà l'applicativo.
- Gestione dei plugin mediante dei file XML: viene cioè decisa a tempo di esecuzione quali sono i plugin da caricare e il modo in cui questi si devono integrare all'interno dell'applicativo principale.
- Sistema di condivisione dei dati tra i vari plugin mediante un modello MVC (Model View Controller)
- Sistema di gestione degli eventi di sistema

- Sistema di aggiornamento dei plugin mediante una architettura client server distribuita su protocollo http/ftp

2.4.2 Sabot Workbench

Attualmente Sabot Workbench è strutturato con un core attorno al quale vengono montati i vari plugin funzionali. Nella fattispecie il core ha la capacità di visualizzare i dati relativi al workspace come posizione e orientamento dei mobot e la posizione degli ostacoli. I plugin aggiungono al core funzionalità extra; quello principale mette a disposizione del core stesso i dati di posizione da visualizzare. L'interazione tra plugin e core avviene mediante una interfaccia di tipo Subject-Observer (a.k.a Publish-Subscribers)[Eri98] dove il plugin gioca il ruolo di publisher. In figura 2.6 (pag.13) è rappresentato il modello Subject-Observer, dove la parte Subject risiede all'interno del plugin e la parte Observer risiede all'interno del core.

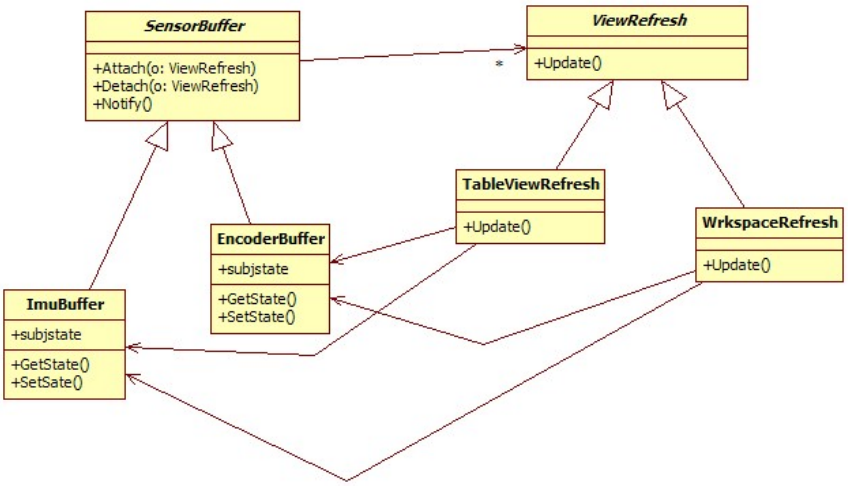

Figura 2.6: *Interfaccia pubilsh-subscribers tra plugin e workspace view del core*

Stabiliamo che i plugin possono essere di due tipi differenti: **UI plugin** e **Faceless plugin**. Uno UI plugin aggiunge una nuova funzionalità al *Sabot Workbench* utilizzando uno qualsiasi dei metodi di *contribuzione* definiti dal framework RCP aggiungendo anche una controparte *visuale* alle proprie funzionalità. Un Faceless Plugin aggiunge funzionalità al *Sabot Workbench* senza aggiungere alcuna controparte grafica di interfaccia. Per garantire la scalabilità del *Sabot Workbench* si da la seguente regola per la gestione dei Faceless Plugin.

Regola 1. Si stabilisce che lo scambio di dati tra un plugin importato e il *Sabot Workbench* core, debba avvenire mediante una interfaccia di tipo Object-Subscribers dove il core è almeno uno dei subscribers.

Lo stato attuale di sviluppo del *Sabot Workbench* prevede come già detto una view di visualizzazione del workspace. È stato poi aggiunto un faceless plugin di gestione del protocollo *ControlML* (vd. cap. 7 e 8). Il core è un subscribers di questo plugin in quanto i dati di posizione

e orientamento dei **SabotOne** e degli ostacoli arrivano mediante questo protocollo. Altri plugin che sono stati aggiunti sono:

- Gestione dei parametri PID dei motori: subscriber del *ControlML* plugin
- Gestione di un software joystick per il controllo manuale del robot
- Interfaccia per la gestione remota del file system della scheda intelligente a bordo del **SabotOne**

In figura 2.7 a pagina 14 è riportato uno screenshot dell'applicativo durante lo startup.

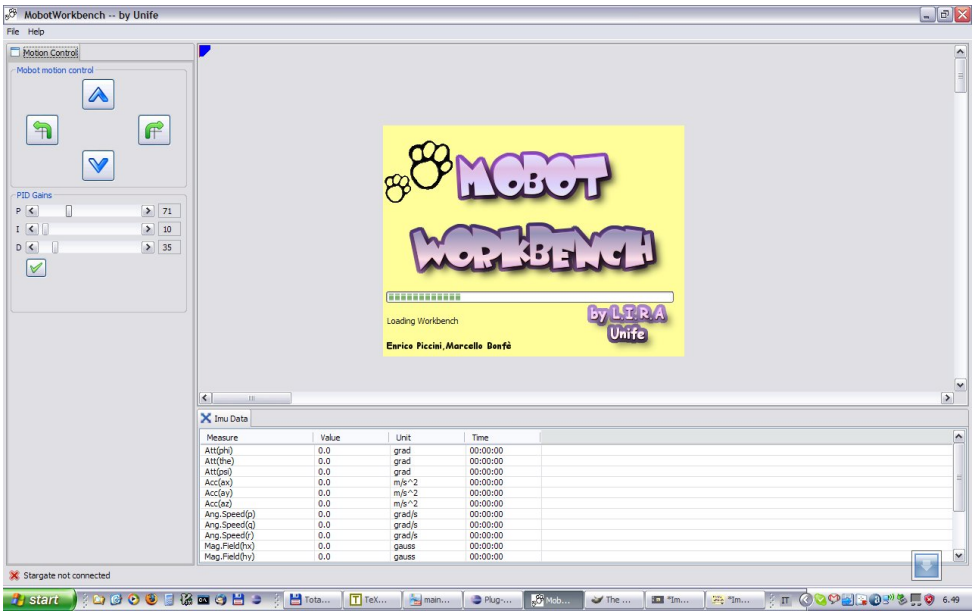

Figura 2.7: *Screenshot di Sabot Workbench durante il caricamento dei plugin*

Capitolo 3

Software SSB: Sabot System-manager Board

La scheda SSB è l'unità di gestione di tutti i sottosistemi presenti sul **SabotOne** . Tale unità di elaborazione è deputata allo svolgimento dei seguenti compiti:

- gestione del protocollo *Jabber/XMPP* di comunicazione verso gli altri **SabotOne** e verso *Sabot Workbench*
- realizzazione di tutti gli algoritmi di alto livello di: pianificazione delle traiettorie verso il target, strategie di obstacle avoidance, autolocalization
- strategie di comportamento e gestione dei task
- configurazione di sistema

3.1 Hardware e sistema operativo

Si è adottata una scheda formato EPIC modello: NANO-LUKE-1G-1G-R11. La board è stata così configurata:

- RAM: 1GB DDR
- Storage: 8GB compact flash (nessun hd per evitare parti meccaniche in movimento danneggiabili con le vibrazioni del sistema)
- Fanless: il processore, opportunamente accoppiato con un box metallico, può funzionare senza ventola di dissipazione
- Bus di espansione: PC104+
- Possibilità di installare un display con interfaccia LVDS

Sulla scheda è stato installato un sistema operativo Linux con kernel 2.6.24. Questo kernel supporta una patch che permette di utilizzare uno schedulatore real time e timer ad alta risoluzione (i.e usleep). Per risparmiare spazio su C.F (compact flash) il sistema attualmente non comprende: l'X-server (300MB c.ca) e il desktop manager; se in futuro si desiderasse installare un

display grafico per la visualizzazione delle informazioni di stato (condizioni di fault, carica della batteria,...), sarà possibile utilizzare framework tipo QTopia che lavorano direttamente utilizzando il frame-buffer. Attualmente il sistema espone anche un server-web per la configurazione dei parametri di bordo.

3.2 Architettura di SSB software

L'architettura sw è stata progettata ponendosi i seguenti obiettivi:

- **modularità**: per favorire l'espandibilità delle funzionalità del sw e la testabilità dei singoli componenti
- **astrazione del sistema operativo**: per rendere il sw indipendente dal S.O

Per raggiungere questi obiettivi, ancora una volta si è fatto ricorso al concetto di plugin sw che è stato realizzato mediante una opportuna strutturazione della gerarchia di librerie e di task; a questa strutturazione è stato dato il nome di **SSBFramework**.

3.2.1 Organizzazione di SSBFramework

Il framework è organizzato secondo una struttura denominata 5-layer pattern [Dou06] riportata in figura 3.1 (pag.17).

Regola 2. In SSBFramework lo *stereotipo* "Domain" è realizzato mediante la strutturazione in un folder principale di dominio (root) e nei subfolder dei *componenti* del dominio. Ogni *componente* viene compilato come *libreria* dal proprio makefile. La compilazione del Domain provoca la compilazione incrementale di tutti i suoi componenti. Ogni *Domain* esporta una interfaccia di tipo *Abstract Factory*.

Il concetto di *Domain* stabilisce in altri termini una regola di organizzazione del file system al fine di fissare una gerarchia di compilazione ben determinata e stabilire il formato dei binari prodotti dalla compilazione, stabilendo inoltre una interfaccia di creazione degli oggetti presenti nel dominio stesso. In SSBFramework tutti i layer sono realizzati con lo stereotipo *Domain*. Unica eccezione è il layer *SSB* che ha uno stereotipo di tipo "Application". All'interno del *Domain* troviamo anche un folder xxFactory all'interno del quale viene implementata l'interfaccia di utilizzo del dominio.

Esempio 1. Strutturazione del *Driver Domain* sul **SabotOne**

```
+ Driver
    |
    + DriverFactory
    + SACTDriver
    + INSDriver
      Makefile
libDriver.so
```

Regola 3. In SSBFramework lo *stereotipo* "Application" è realizzato mediante la strutturazione in una *Aplication root* nella quale si trovano i file standard dell'applicazione e il subfolder contenente i *Task* dell'applicazione. Ogni *Task* viene compilato come *libreria* dal proprio makefile.

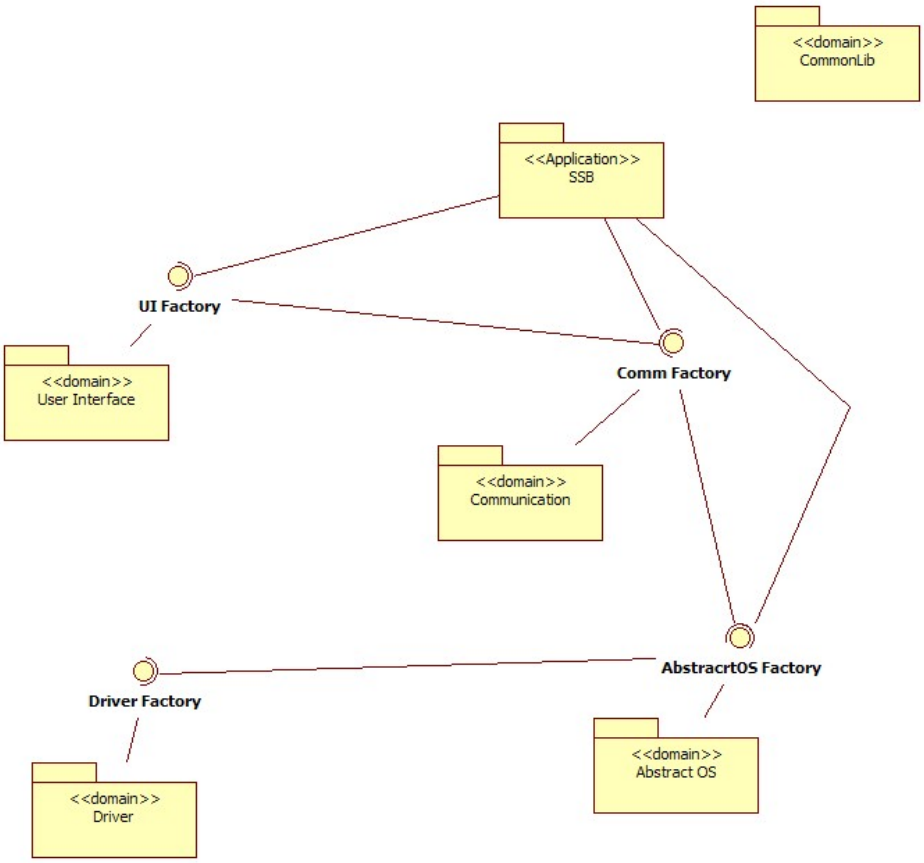

Figura 3.1: *Architettura 5-layer pattern di SSBFramework*

La struttura della *Aplication root* è prefissata. Qui infatti troviamo sempre il file **startup.cpp** all'interno del quale viene fatta l'inizializzazione principale dell'applicativo. Ci sono poi i file *Nome_ applicazione.cpp/.h* dove vengono create le definizioni alla classe *Application* di struttura dell'applicazione che verrà utilizzata da startup per lanciare tutto l'applicativo.

Esempio 2. Strutturazione dell'*Application Domain* sul **SabotOne**

```
+ SSB
    |
    + Tasks
    |    |
    |    + Localizer
    |    |      |
    |    |      Localizer.cpp
    |    |      Localizer.h
    |    |      LocalizerTask.cpp
    |    |      LocalizerTask.so
    |    |      Makefile
    |    + TrajPlanner
    |    + TrajTrakker
    |    + ...
    |    Makefile
    + SSBApp
    |    |
    |    startup.cpp
    |    SSB.cpp
    |    SSB.h
    |  SSB.bin
    |    Makefile
    Makefile
```

Regola 4. Ogni *Domain* espone una interfaccia di tipo *Abstract Factory* [Eri98]

Questa regola stabilisce implicitamente che ogni oggetto rappresentato all'interno del dominio debba avere una controparte astratta che dissocia l'uso dell'oggetto stesso da una sua particolare implementazione. Di quanto detto ne è un esempio l'"abstractOS" Domain che astrae tutte le funzionalità del sistema operativo svincolando il layer applicativo dal particolare sistema operativo su cui gira. In questo senso la definizione di layer (cfr. 5-layer) mediante il concetto di *Domain* e la corrispettiva interfaccia è consistente; ogni layer può infatti essere sostituito con un altro layer che implementi una interfaccia analoga.

3.2.2 Driver Layer

È il layer che sta alla base dell'SSBFramework. Non usa le interfacce esposte dagli altri layer ed è utilizzato solamente dal layer di astrazione del sistema operativo mediante il DriverTask. Ogni dispositivo viene utilizzato attraverso un handler rilasciato da uno specifico metodo *Create* come previsto dalla struttura *Abstract Factory* che viene rappresentato nella figura 3.2 a pag.19

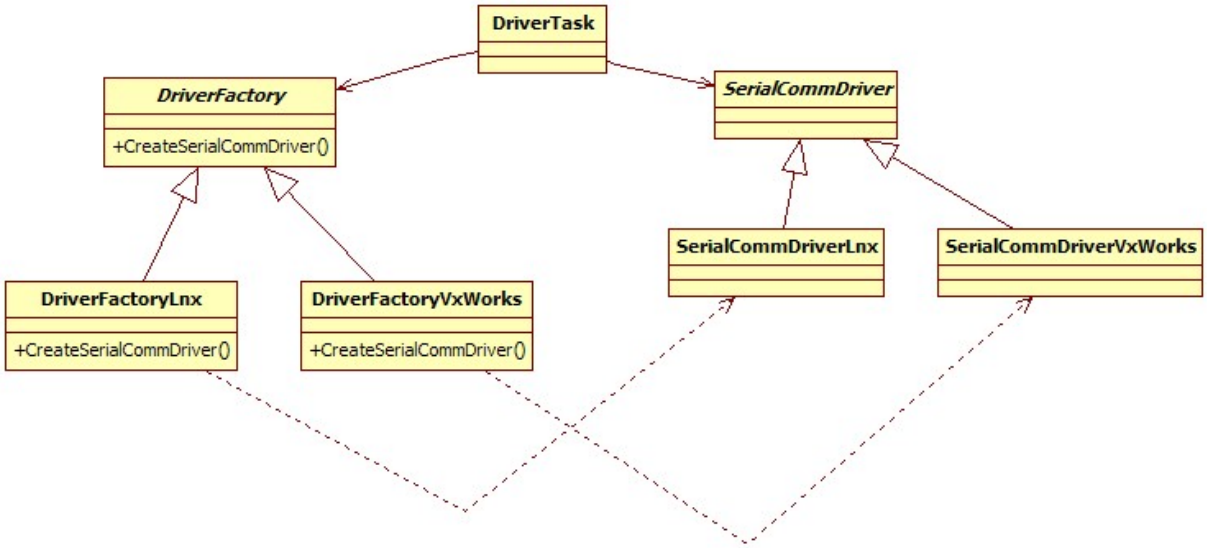

Figura 3.2: *Class diagram dell'interfaccia DriverFactory*

3.2.3 AbstractOS Layer

Anche il layer di astrazione del sistema operativo è strutturato con il pattern *Abstract Factory*[Eri98] (fig.3.3, pag.20). La classe *OSAbstractFactory* è una classe astratta dove vengono concentrati i metodi creatori per tutti gli oggetti del sistema operativo. Tutti i metodi della classe *OSAbstractFactory* sono virtuali puri e vengono implementati nelle due classi derivate *OSvxWorks* e *OSLinux*. Il client utilizza solamente la classe *OSAbstractFactory* specificando la tipologia di sistema operativo (Linux o vsWorks) sul quale l'applicativo deve essere messo in esecuzione. L'istanza al sistema operativo è unica e viene garantita dal pattern *Singletone* realizzato mediante il membro *instance* delle due classi *OSvxWorks* e *OSLnux*. I *prodotti* del pattern sono le classi rappresentanti i vari elementi del sistema operativo che si desidera astrarre. Ogni *prodotto* ha una controparte astratta che viene esposta al client e una controparte concreta che realizza le reali funzionalità in relazione al sistema operativo. Ogni prodotto viene implementato sia per il sistema operativo Linux che per vxWorks.

3.2.4 Communication Layer

Il communication layer realizza tutte le funzionalità relative ai protocolli di comunicazione. Anch'esso espone una interfaccia di tipo *Abstract Factory* per la creazione degli handler di gestione delle interfacce dei vari protocolli. Attualmente nel comunication layer sono implementati due protocolli, il Modbus per la comunicazione verso i *Sensori virtuali* di bordo e il *Jabber/XMPP* per la comunicazione verso altri Sabot e verso il Sabot-Workbench.

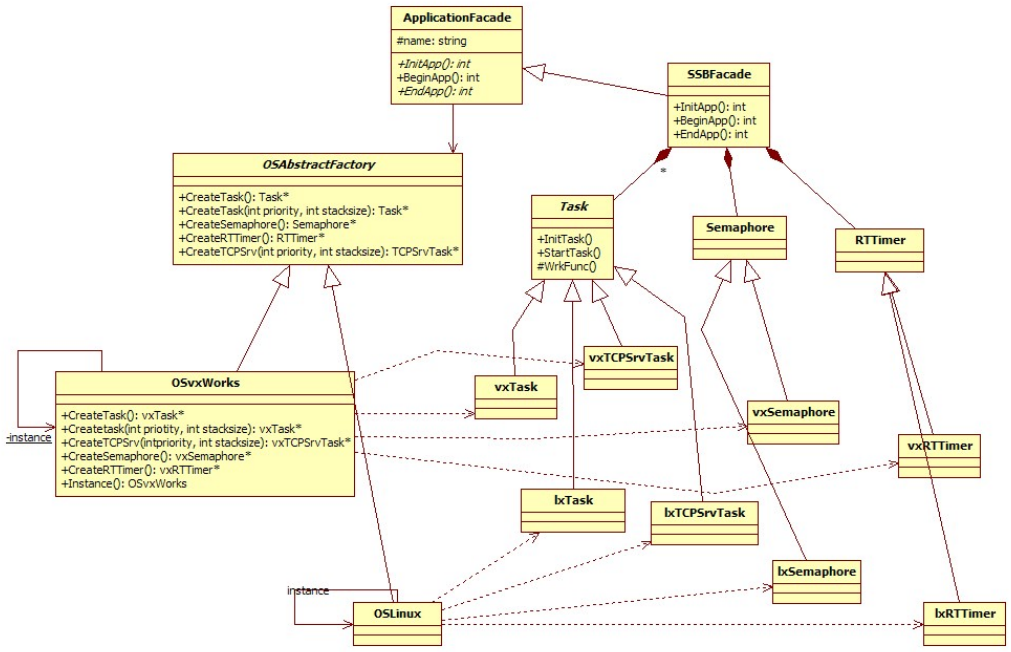

Figura 3.3: *Class diagram del layer di astrazione AbstractOS*

3.2.5 UI Layer

Gestisce tutte le funzinalità di pilotaggio di un eventuale display grafico di bordo per la visualizzazione dello stato del mobot. Attualmente comunque non è presente alcun display per prolungare la vita delle batterie il più possibile in fase di sviluppo

3.2.6 Application Layer

L'application layer è l'armonizzatore di tutti i componenti software fino ad ora citati e di tutti i task che vengono messi in esecuzione sul mobile robot. Nell'application layer vengono svolte le seguenti macro attività:

- parserizzato un file XML di inizializzazione di tutti i parametri ed eseguita l'inizializzazione di tutti i componenti hw/sw del mobot
- creata l'istanza al sistema operativo su cui l'applicazione viene eseguita (nel nostro caso Linux)
- creata l'istanza alla classe principale dell'applicazione (che segue sempre un pattern *Facade*[Eri98]) e avviata l'applicazione stessa
- è compito della classe principale dell'applicativo creare l'istanza, inizializzare e avviare tutti i task dell'applicativo

La classe principale dell'applicativo segue la *politica* del pattern facade nel senso che ha il compito di concentrare tutte le funzioni dell'applicativo stesso in una interfaccia semplificata che

ne nasconde la complessità e che viene esposta ad un generico launcher che mette in esecuzione l'applicazione (fig. 3.4 a pag. 21) seguendo una procedura standard indipendente dall'applicativo.

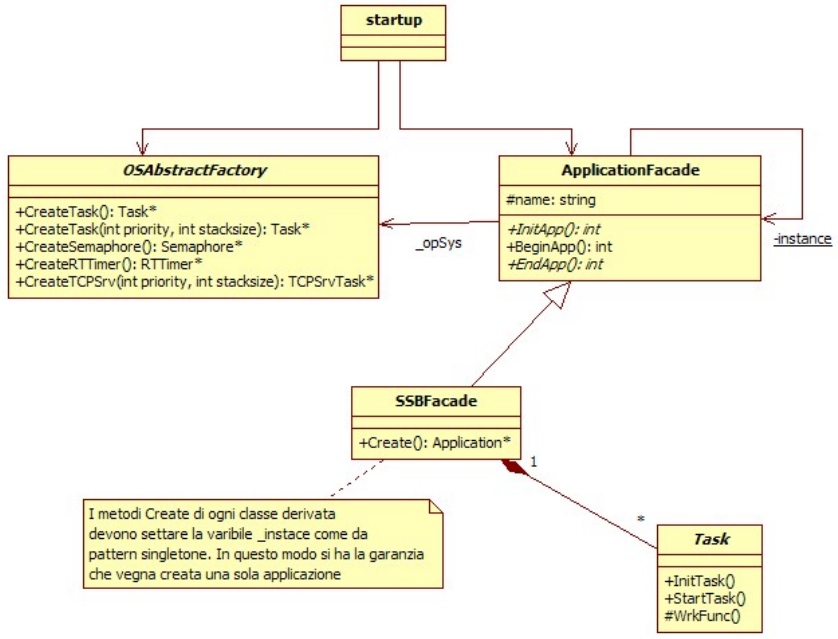

Figura 3.4: *Struttura facade della classe principale dell'applicativo e interazioni con il launcher*

La classe **startup** realizza il generico application launcher; questo dopo aver creato le corrette istanze al sistema operativo e alla generica classe **ApplicationFacade** esegue le chiamate alle funzioni **InitiApp e BeginApp**. Terminata l'esecuzione della funzione BeginApp il controllo del programma viene restituito alla classe **startup** che chiama la funzione **EndApp** di *finalizzazione* del programma (i.e deallocazione di tutta la memoria allocata dinamicamente e terminazione di tutti i task). Le funzioni della classe astratta **ApplicationFacade** sono implementate nella classe concreta **SSBFacade**. In particolare la funzione **BeginApp** è responsabile della istanza di tutte le classi e della creazione di tutti i Task che realizzeranno i vari algoritmi di gestione del Sabot. La regola 3 (pag.16) stabilisce la struttura dell'*Application Domain*. Ogni Task viene compilato come libreria (i.e shared object in Linux e vxWorks) in quanto su questi viene poi eseguito un *lazy-loading* a runtime. Desiderando infatti configurare i vari task come *plugin* selezionabili a runtime con l'uso di un file di configurazione, si è reso necessario definire una interfaccia comune per ogni task (i.e classe Task fig.3.3 a pag.20) e predisporre un meccanismo di caricamento dinamico del Task. Questa funzionalità è importante dal punto di vista dello sviluppatore che potrà decidere, ad esempio, se caricare il task che implementa un determinato algoritmo piuttosto che un altro

Esempio 3. Per la pianificazione delle traiettorie è possibile implementare svariati algoritmi come: roadmap, potenziali, LPN...Nel nostro caso è stato implementato il metodo LPN-DE che risulta essere molto veloce pertanto adatto alla pianificazione in ambiente fortemente dinamico. Tale metodo è implementato in un *Task*, compilato in un shared object che viene caricato a tempo

di esecuzione. Qual'ora venisse implementato un algoritmo di pianificazione quale ad esempio il metodo dei campi vorticosi, questo potrebbe sostituire il precedente semplicemente modificando il file di configurazione del'applicativo.

La possibilità di caricare algoritmi e task run-time si sposa bene con una certa attività di ricerca che richiede la sostituzione di svariati moduli software operativi e la collaborazione di gruppi di lavoro con competenze che spesso sono compartimentate.

3.2.7 CommonLib

Il dominio CommonLib non rientra direttamente nella struttura 5-layer in quanto realizza un namespace di funzioni, macro e costanti di uso generico quali ad esempio: algoritmi numerici, algoritmi di ordinamento, funzioni di gestione dei file e di altre entità del sistema operativo. In questo dominio vengono altresì incluse tutte le librerie di terze parti che, dotate di un opportuno makefile, vengono fatte rientrare nel meccanismo di compilazione descritto in precedenza. Alcuni dei principali elementi di CommonLib sono:

- **log4cxx** fornita dalla Apache Foundation per la gestione di tutto il log di sistema. Questo logger può essere configurato mediante un file di configurazione, è thread-safe, permette una raffinata formattazione delle righe di log
- **Spliner** è l'algoritmo implementato per il calcolo delle spline
- **Inifile Parser** è il parser dei file di inizializzazione del Sabot che sono scritti in formato ini
- **Jabber** è la libreria utilizzata per la gestione del protocollo jabber/XMPP su cui è stato costruito il *ControlML* protocol
- **Numerical** è la libreria in cui sono presenti alcuni algoritmi numerici di uso generico

Nei prossimi capitoli verranno esaminati nel dettaglio gli algoritmi di controllo delle traiettorie, di path planning e di autolocalizzazione.

Capitolo 4

Unità SACT: Sabot Actuator Board

4.1 Dimensionamento dei motori

Come si è già detto il modello di guida che si è adottato è il *differential drive*. Nella tabella 4.1 sono messi a confronto le caratteristiche dei vari modelli di guida.

Il dimensionamento dei motori passa attraverso il calcolo delle caratteristiche di coppia e di velocità angolare necessarie per movimentare la piattaforma mobile al fine di scegliere dai cataloghi il motore che meglio si adatta a queste caratteristiche. Tuttavia, prima della costruzione completa del mobot, non è possibile conoscere con precisione tutti i parametri del sistema. Si hanno infatti numerosi fattori d'incertezza, tra i quali attriti reali ed il peso del veicolo: quest'ultimo potrebbe essere inoltre variabile; possiamo comunque prevedere i seguenti parametri per il mobot:

- Massa complessiva massima: $15Kg$;
- Velocità massima: $0,5m/s$;
- Accelerazione massima: $0,5m/s^2$;
- Raggio delle ruote: $7,5cm$;

Consideriamo quindi il veicolo in moto rettilineo su una superficie scabra obliqua. Il veicolo ha una massa complessiva m ed ha il centro di massa posto poco oltre l'asse delle ruote motrici, verso il castor, in un punto equidistante dalle ruote. In questo modo possiamo assumere che il peso sia distribuito in misura uguale sulle due ruote motrici. Assumiamo che la massa delle ruote sia trascurabile rispetto al carico della piattaforma e trascuriamo inoltre il disturbo dovuto al caster.

In riferimento alla figura 4.1 si ha che:

$$m\overline{a} = \overline{F}_t + \overline{R} + \overline{F}_a + \overline{F}_g \tag{4.1}$$

$$\tag{4.2}$$

con:
$\overline{F}_g = m\overline{g}$ forza peso del veicolo
$\overline{F}_a$ forza di resistenza con l'aria
$\overline{F}_t$ forza di trazione

Struttura	Vantaggi	Svanaggi
Differential Drive	- Facile programmare semplice costruzione - Facile da costruire - Sterzata e movimento disaccoppiati	- Non adatto a terreni sconnessi - Difficile a mantenere in linea retta - Può essere instabile
Ackerman Steering	- Compatibile con terreni sconnessi - Buon controllo direzionale	- Sterzata accoppiata al movimento - Manovre difficili - Programmazione complessa
Tricycle Drive	- Compromesso tra stabilità e semplicità - Bassi raggi di sterzata	- Sterzata accoppiata al movimento - Non adatto a terreni sconnessi
Tank Treads	- Stabile su terreni sconnessi - Sterzata e movimento disaccoppiati	- Elevate potenze di attuazione - Sterzata non accurata
Synchro Drive	- Stabile e facile da programmare - Sterzata e movimento disaccoppiati	- Non adatto a terreni scoscesi - Meccanica complessa
Omnidirectional Drive	- Stabile e facile da programmare - Sterzata e movimento disaccoppiata	- Non adatto a tereni scoscesi - Ruote complesse da realizzare - Bassi payload

Tabella 4.1: *Tabella comparativa tra i vari modelli di guida di un veicolo mobile*

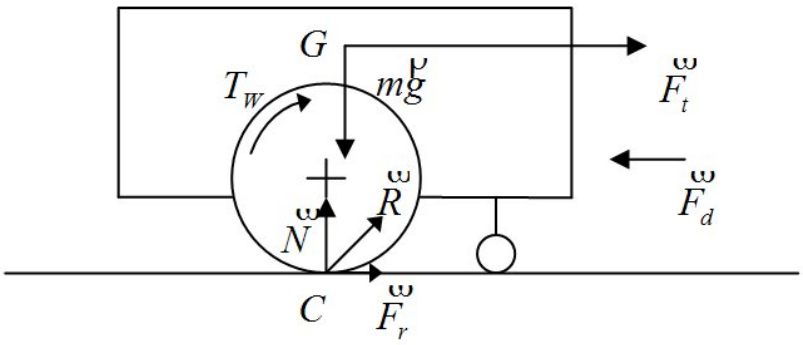

Figura 4.1: *Modello delle forze in gioco sul mobot*

$\overline{F}_r$ forza di attrito evolvente
$\overline{R}$ forza di reazione del piano alla forza peso

Considerando il moto su un piano orizzontale si avrà che:

$$m\overline{a} = \overline{F}_t + \overline{F}_a$$

Per il coefficiente di attrito dinamico si ha che:

$$\overline{F}_r = \mu_d m\overline{g}$$

dove μ_d è il coefficiente di attrito dinamico.

Per il coefficiente di attrito aereodinamico abbiamo che:

$$\overline{F}_d = S_g C_d A v^2$$

dove:
$S_g = 0,00071 Kgm^{-3}$ densità dell'aria
A massima area della sezione trasversale dei veicolo nella direzione del moto
C_d coefficiente di trascinamento dell'aria

Forma del veicolo	C_d
Solido rettangolare	$1,4$
Solido cilindrico	$0,9$
Semisfera	$0,4$
Sfera	$0,3$
Automobile	$0,35 - 0,6$

Tabella 4.2: *Alcuni coefficienti di trascinamento dell'aria*

Per movimentare il veicolo con la massima accelerazione, nelle condizioni più sfavorevoli al moto, sino alla massima velocità lineare v_{max} occorre rendere disponibile la potenza:

$$P_{tmax} = F_t v_{max}$$

Considerando $\mu_D = 0,2$, $C_d = 1$, $A = 0,4m^2$ e $v_d = 10m/s$ (v_d velocità relativa rispetto all'aria) si avrà che $F_t \approx 33.2N$. Detta T_W la coppia del motore, la potenza P_W che deve essere erogata per movimentare il mobot sarà $P_W = T_W \omega_W = P_t$ dove ω_W è la velocità di rotazione della ruota. Abbiamo quindi:

$$P_{t(max)} = F_t v_{max} = 33,2 * 0,5 = 16,6W$$

La corrispondente coppia motrice sarà:

$$T_{W(max)} = rF_t = 0,075 * 33,2 = 2,5Nm$$

Questa potenza deve essere erogata in corrispondenza alla velocità massima dell'albero motore che stando alle ipotesi di progetto deve valere:

$$\omega_{Wmax} = \frac{v_{max}}{r} = \frac{0,5}{0,075} = 6,67rad/s = 63,72rpm$$

Per soddisfare i requisiti di potenza e velocità è necessario utilizzare un motoriduttore per il quale (a meno di un fattore di efficienza) vale la seguente relazione:

$$n = \frac{\omega_{M(max)r}}{v_{max}}$$

dove $\omega_{M(max)}$ è la velocità massima a vuoto del motore dichiarata sul data-sheet. Scegliendo un motore che abbia $\omega_{M(max)} = 4000rpm \approx 419rad/s$ otteniamo un rapporto di riduzione pari a $n \approx 63$. In riferimento ai valori commerciali dei motoriduttori planetari maxon (rif. tab. 4.1 a pag. 26), ne viene scelto uno con rapporto di riduzione pari a $66:1$ che è quello più vicino al valore teorico calcolato.

Rapporto di riduzione	**Efficienza ς(%)**	**Stadi di riduzione**	**Coppia massima continua**
4:1	80	1	2,25
18:1	75	2	2,25
66:1	70	3	4,50
318:1	60	4	4,50
1526:1	50	5	4,50

Tabella 4.3: *Efficienza dei motoriduttori planetari a ingranaggi metallici Maxon Serie GP32A*

La potenza e la coppia meccanica massima che dovrà sviluppare il motore saranno:

$$P_{M(max)} = \frac{P_{t(max)}}{\varsigma} = \frac{16.6}{0,7} = 23,7W$$

$$T_{W(max)} = \frac{T_W}{\varsigma n} = \frac{2,85}{0,7 * 66} = 62 * 10^{-3} Nm$$

in corrispondenza della velocità massima del motore:

$$\omega_{M(max)} = n\frac{v_{max}}{r} = 440 rad/s = 4200 rpm$$

Questi valori di coppia e velocità appartengono al normale campo di lavoro di dei motori c.c. di piccola potenza. Una scelta basata esclusivamente sui questi valori massimi porterebbe a sovradimensionare il motore. Occorre pertanto calcolare il valore medio quadratico della coppia richiesta ai motori. D'altra parte non è possibile determinare a priori il tipo di moto che condurrà il veicolo. La frequenza e la durata delle accelerazioni dipendono dagli ostacoli presenti nell'ambiente e dagli algoritmi di controllo del cammino. Di conseguenza non è possibile stabilire il valore efficace della coppia richiesta al motore. Nella nostra applicazione prevediamo di utilizzare il veicolo prevalentemente in ambiente chiuso e su piano orizzontale; abbiamo quindi supposto che il moto del veicolo sia alternato da frequenti fasi con massima accelerazione (decelerazione) a fasi con velocità lineare costante. I valori ricavati sono riportati in Tabella 4.1 (pag.27). La colonna più a destra della tabella riporta una stima percentuale del tempo di utilizzo per ciascuna condizione operativa, sull'intervallo di tempo di riferimento.

Tipo di moto	**Coppia sull'asse della ruota T_W**	**Percentuale di utilizzo**
Moto su piano orizz. con acc. max.	2,0	80
Moto su pinao orizz. a velocità cost.	1,7	20

Tabella 4.4: *Coppie massime richieste sull'albero della ruota in funzione del tipo di utilizzo*

Il valore efficace della coppia richiesta sull'asse delle ruote sarà pertanto:

$$T_{W(rms)} = \sqrt{\frac{\sum T_{Wi}^2 t_i}{\sum t_i}} = 1,9 Nm$$

dalla quale ricaviamo la coppia efficace richiesta ai motori

$$T_{M(rms)} = \frac{T_{W(rms)}}{\varsigma n} = 41 * 10^{-3} Nm$$

Un modello di motore idoneo alla nostra applicazione è stato selezionato dal catalogo Maxon (mod. A-MAX32).

Tipo di moto	**Coppia sull'asse della ruota T_W**	**Percentuale di utilizzo**	**U.M**
Potenza nominale	P_N	11	W
Tensione nominale	E	24	V
Velocità a vuoto	ω_{M0}	8400	RPM
Coppia di stallo	T_{PK}	$85,1$	mNm
Corrente a vuoto	I_{NL}	$29 * 10^{-3}$	A
Corrente di avvio	I_{PK}	$3,04$	A
Resistenza ai morsetti	R	$7,90$	Ω
Massima corrente continuativa	I_C	$0,732$	A
Coppia continuativa massima	T_C	$19,6 * 10^{-3}$	Nm
Potenza massima alla tensione nominale		$17,4$	W
Efficienza massima	η	80	%
Costante di coppia	K_T	$29,6 * 10^{-3}$	N/A
Costante di velocità	K_E	350	rpm/V
Costante di tempo meccanica	τ_m	$14 * 10^{-3}$	s
Inerzia rotore	J_M	$1,24 * 10^{-6}$	Kgm^2
Induttanza	L	$0,77$	mH

Tabella 4.5: *Parametri del motore A-MAX32*

4.2 Elettronica di controllo

4.2.1 Dimensionamento driver

Convertitore di potenza

Trascurando l'induttanza degli avvolgimenti di armatura, la massima tensione richiesta sui morsetti dei motori è

$$E_{max} = R\frac{T_{WM(max)} + T_{NL}}{K_T} + K_E\omega_{M(max)} = 19,9V$$

dove con T_{NL} si è indicato la coppia dovuta alle perdite interne motore, ottenuta moltiplicando la corrente a vuoto per la costante di coppia del motore.

La corrente efficace richiesta da ciascun motore è:

$$I = \frac{T_{WMrms} + T_{NL}}{K_T} = 0,77A$$

In commercio sono disponibili diversi integrati per il controllo di motori a collettore per correnti sino a $4Arms$ e tensioni massime di $50V$. Questi dispositivi rendono disponibile un'interfaccia a livelli logici TTL in grado di comunicare direttamente con la logica di controllo. Con questi dispositivi risulta molto semplice ed economico il controllo dei motori in quanto implementano al loro interno le temporizzazioni (dead time) necessarie per la corretta gestione dello spegnimento e dell'accensione dei transistor sullo stesso ramo del ponte ad H.

Per il progetto della scheda SACT, è stato adottato il dispositivo **LMD18200T** della *National Semiconductor*. Questo dispositivo è un ponte ad H integrato che contiene quattro DMOS di potenza con i rispettivi diodi intrinseci di ricircolo, collegati nella configurazione "ponte ad H". Sullo stesso chip sono integrati dei circuiti convertitori di livello che permettono di controllare l'intera struttura del ponte tramite dei segnali logici d'ingresso aventi livelli elettrici tipici della logica digitale. Sono inoltre presenti circuiti di protezione da cortocircuito delle uscite, dal sovrariscaldamento, e di blocco nel caso la tensione di alimentazione scenda sotto $10V$. Con questo ponte integrato è possibile controllare il motore in due modi distinti:

- Controllo a semionda (Sign/Magnitude Controll)
- Controllo a onda intera (Locked Antiphase Control)

La scheda realizzata utilizza la modalità di controllo ad onda intera. Il pilotaggio PWM ad onda intera è ottenuto con una continua inversione di polarità della tensione applicata al motore alternativamente positiva e negativa pari a $\pm V_S - 2V_{DSon}$, dove V_S indica la tensione di alimentazione del ponte e V_{DSon} è la caduta di tensione sui transistori nello stato di conduzione. La frequenza di inversione scelta è di $20KHz$, è più elevata dell'inverso della costante di tempo del circuito. A causa di questa condizione la corrente non può cambiare verso nell'induttanza ma oscilla attorno ad un valor medio. In corrispondenza ad un valore numerico V_{pid} nullo all'uscita dal regolatore PID digitale, è inviato un segnale PWM di controllo al ponte con duty cycle del 50%. In questa situazione la corrente media nel motore è nulla, il motore non sviluppa coppia e quindi è fermo. Modificando il duty cycle è possibile regolare il valor medio della tensione sul motore e quindi controllare la velocità. Il verso di rotazione dipende dal fatto che il duty cycle è minore o maggiore del 50%. Esiste un limite al range del duty cycle imposto dalparametro t_{pw} *(Minimum Input Pulse Width)* dell'integrato LMD18200. Questo esprime il valore minimo della durata dei segnali d'ingresso al ponte per avere garanzia di corretto funzionamento. Nella nostra applicazione il segnale PWM ha un periodo $T_{PWM} = 50\mu s$ e una risoluzione di 10 bit. Il valore numerico minimo $N_{PWM(min)}$ da assegnare al modulo PWM del dsPIC per rispettare il tempo minimo t_{pw} si ricava imponendo la condizione:

$$\frac{T_{PWM}}{1024} N_{PWM(min)} > t_{pw}$$

da cui si ricava

$$N_{PWM(min)} > 1024 \frac{t_{pw}}{T_{PWM}} = 21$$

Per simmetria non è possibile applicare al ponte un segnale con duty cycle prossimo al 100%. Il valore numerico massimo da assegnare al modulo PWM è pertanto $N_{PWM(max)} = 1024 - 21 = 1003$.

Sensing della corrente

La corrente in uscita sul pin di sensing è tipicamente di $377\mu A$ per ogni ampère di corrente complessiva sul dispositivo. Abbiamo collegato questo pin ad un resistore del valore pari a $1,87K\Omega$; in questo modo in corrispondenza della corrente massima di $3A_{rms}$ sul ponte, si ottiene una tensione di $2,1V$. In questo modo il convertitore A/D del microcontrollore, che utilizza le tensioni di alimentazione come riferimenti interni, rileva la corrente massima sul ponte misurando un valore numerico (a 8 bit) pari a: $adc = 255\frac{2,1}{5} \cong 107$. Per ridurre l'errore casuale il programma esegue la media di 16 misure per ciascun canale prima di elaborarne il risultato.

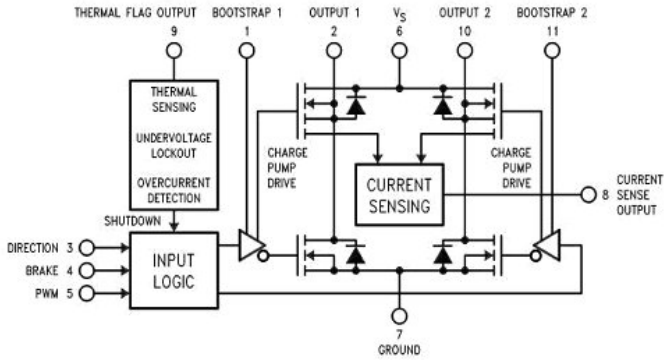

Figura 4.2: *Schema dell'integrato LMD18200*

4.2.2 CPU

I requisiti di sistema che deve soddisfare la scheda sono:

- Due driver per il pilotaggio dei motori
- Una porta seriale per comunicare con altre schede nell'applicazione **SabotOne** (Modbus)
- Una porta CAN per applicazioni future e attività didattica
- Un connettore per programmare il controllore in-circuit
- Due ingressi encoder (uno per ogni motore)
- Input/Output digitali (pilotaggio sensori sensori)
- Ingressi analogici (acquisizione sensori)
- Interfaccia ethernet

Per pilotare i due driver è necessario generare due segnali PWM indipendenti che sono già disponibili sul processore scelto (PWM Generator Units). Il dsPic scelto dispone anche di un ingresso apposito per il conteggio dei passi dell'encoder, detto Quadrature Encoder Interface (QEI). Dato che si ha l'esigenza di gestire due motori con i relativi encoder è necessario realizzare l'elttronica di condizionamento per l'acquisizione del secondo encoder. Si è utilizzato il chip LS7183 della LSI che è un quadature clock convert e perette di converterire le due fasi di un encoder in un segnale di clock e un segnale di direzione applicando già il prefiltraggio sulle transizioni spurie. Siamo in questo modo in grado di utilizzare due ingressi GPIO del microcontrollore risolvendo il problema della gestione del secondo encoder con un solo contatore.

La scheda di controllo SACT comunica con la scheda SSB mediante un protocollo seriale punto multipunto (Modubus). Tuttavia per scopi e applicazioni future, si desidera dotare la scheda di una interfaccia ethernet con i relativi stack TCP e UDP. Poichè il dsPIC30 non dispone di una componentistica dedicata a questi scopi, si sceglie di utilizzare un convertitore ethernet/seriale "*smart*" evitando così di caricare il micro con le funzioni di gestione degli stack di comunicazione che porterebbero via molte delle risorse necessarie agli algoritmi di controllo. Si tratta di un

componente prodotto dalla Lantronix e denominato XPort. È disponibile in svariati modelli (tutti con footprint compatibili) per supportare sia la comunicazione su RJ45 che in 802.11b/g. Montanto a bordo un microntrollore con sistema operativo Linux che rende disponibili gli stack TCP e UDP ma non solo, sono resi disponibili anche alcuni protocolli di gestione come: DHCP, SNMP, Zeroconf,.... Un web server residente può essere utilizzato per la gestione di pagine di configurazione i cui parametri possono essere inviati via seriale al microcontrollore.

Tenuto conto di tutte queste esigenze si è scelto il dsPIC30F6015 che dispone di un elevato numero di pin (64) in grado di svolgere tutte le funzioni richieste. In figura 4.3(pag.31) è riportato uno schema a blocchi della interfaccia motori-dsPIC sulla scheda SACT; in figura 4.4 (pag.32) è riportata uno schema a blocchi dell'interfaccia di comunicazione della scheda SACT; in figura 4.5 (pad.32) è riportata uno schema a blocchi generale della scheda; ed infine in figura 4.6 (pad.33) è riportata una fotografia della scheda ultimata.

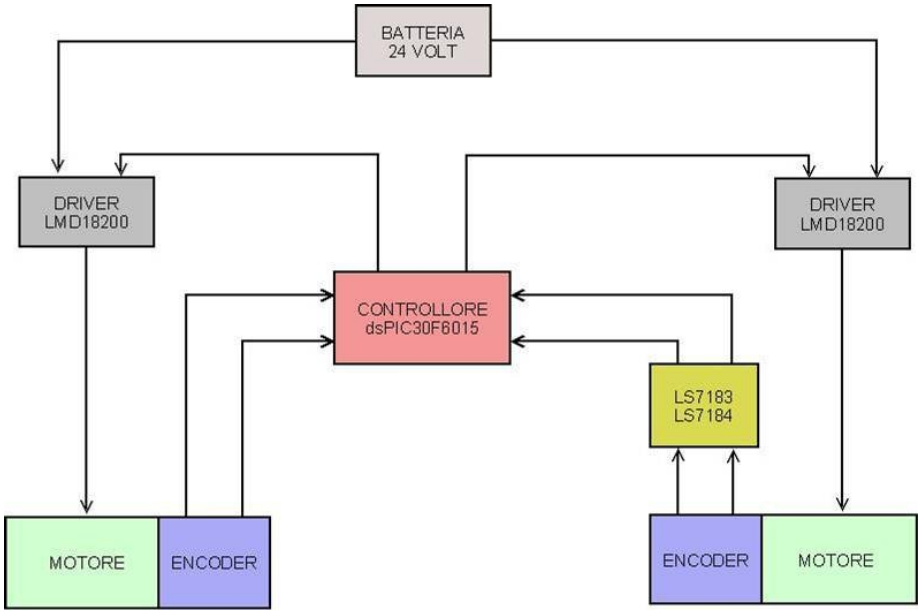

Figura 4.3: *Schema a blocchi dell'interfaccia motori sulla scheda SACT*

4.3 Software

Il firmware implementato sulla scheda SACT deve svolgere principalmente il compito di regolare i due motori con inseguimento di posizione dati i profili di velocità e accelerazione esponendo un opportuno protocollo comandi sulle proprie interfacce di comunicazione. Nello specifico il firmware svolgerà i seguenti compiti:

- eseguire una corretta inizializzazione dell'hw
- eseguire un corretto algoritmo di controllo di posizione su entrambe i motori
- impostare la velocità istantanea (anche durante il moto) dei motori mediante le interfacce di comunicazione

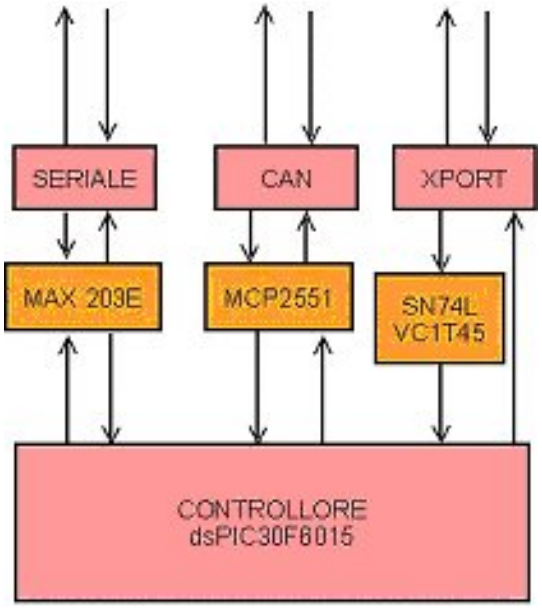

Figura 4.4: *Schema a blocchi dell'interfaccia di comunicazione sulla scheda SACT*

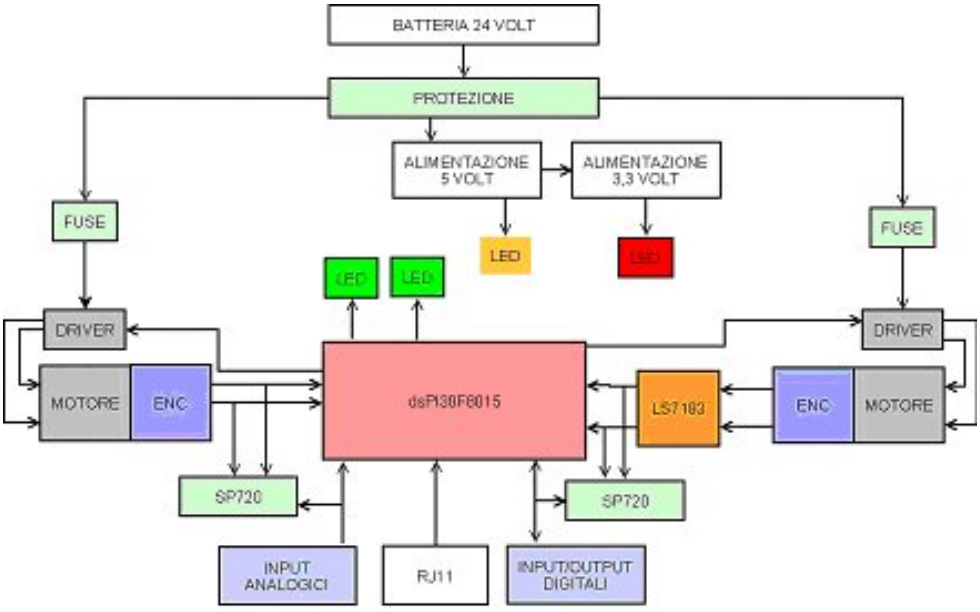

Figura 4.5: *Schema a blocchi generale della scheda SACT*

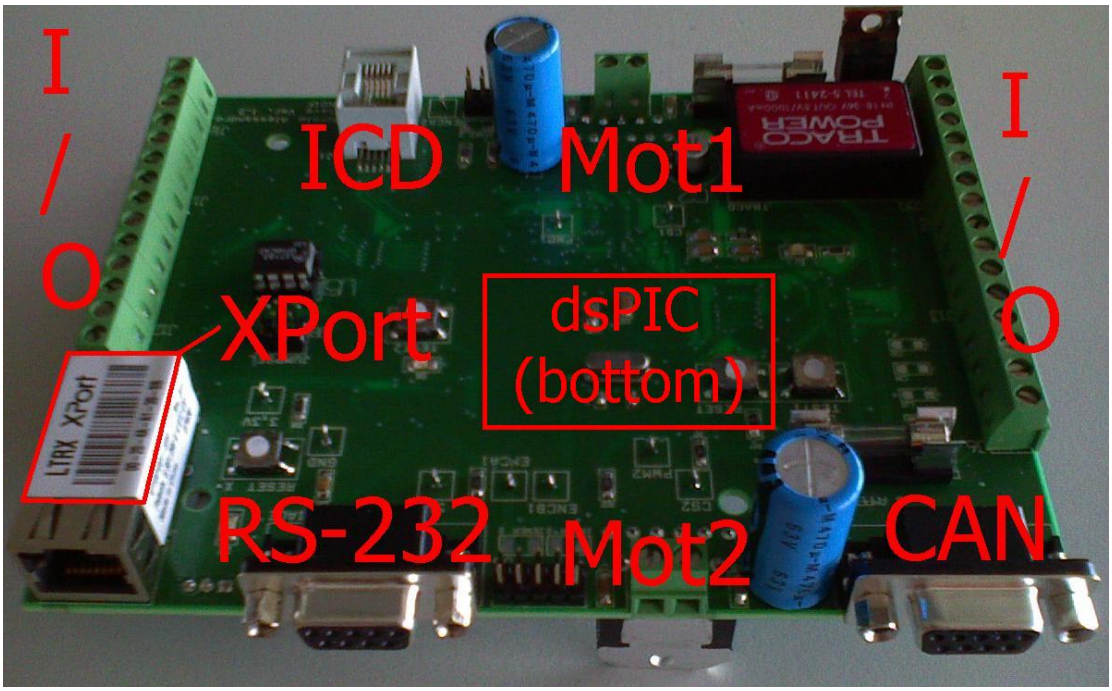

Figura 4.6: *Fotografia della scheda SACT*

- rilevare la posizione istantanea (anche durante il moto) di entrambe i motori mediante le interfacce di comunicazione
- impostare i parametri di regolazione (guadagni del PID, accelerazione massima e velocià massima) mediante le interfacce di comunicazione
- opzionalmente: rilevare eventuali sensori connessi sugli input analogici e digitali
- opzionalmente: comandare eventuali attuatori connessi sugli output digitali
- opzionalmente: impostare le caratteristiche degli encoder, del motoriduttore e della massima velocità a vuoto del motore per permettere la completa configurabilità della scheda di controllo.

4.3.1 Algoritmo di regolazione

Tutte le elaborazioni eseguite per il controllo del profilo di velocità dei motori sono eseguite con periodo pari a $T_S = 50\mu s$, all'interno della ISR a bassa priorità Low_ISR del dsPIC. I seguenti task sono eseguiti in sequenza nell'ordine riportato, per ciascuno dei motori:

1. Calcolo della posizione angolare del motore;
2. Calcolo della posizione desiderata per il motore;
3. Calcolo dell'errore;
4. Calcolo del nuovo valore di duty cycle del segnale PWM.

Calcolo della posizione angolare del motore

Il primo task è eseguito da una routine che calcola modulo e direzione della distanza angolare percorsa dal motore nell'intervallo di campionamento considerato. La QEI (qudrature encoder interface) opportunamente inizilizzata, mantiene il conteggio degli impulsi relativi alla rotazione dell'albero motore gestendo eventuali overflow sul registro di conteggio. Ai valori correnti del registro di conteggio, sono sottratti i rispettivi valori letti nel periodo di campionamento precedente.

$$mvelocity = (qei - prev.qei)$$

Il valore di mvelocity rappresenta la velocità misurata istantanea del motore in termini di numero di impulsi su intervallo di campionamento. La posizione angolare del motore viene mantenuta in una variabile intera a 32 bit. I 24 bit più significativi di questa variabile rappresentano le unità di impulsi conteggiati dall'encoder, mentre i rimanenti 8 bit meno significativi rappresentato frazioni di impulsi di conteggio. Il valore di velocità viene quindi sommato a quello di posizione ad ogni periodo di campionamento a partire dal nono bit meno significativo.

Lo scopo di degli 8 LSb della posizione, è quello estendere l'intervallo di velocità verso il basso, permettendo il controllo del motore a basse velocità. È stata infatti utilizzato l'aritmetica intera ed alcuni "shift" per ottenere una parte decimale in virgola fissa. Questo metodo è stato necessario perchè l'uso delle operazioni in virgola mobile richiede l'utilizzo di funzioni che salvano e ripristinano in continuazione lo stato della CPU. Nel nostro caso ciò avrebbe comportato un maggiore "overhead" computazionale.

La variabile *encoder* a 32bit memorizza anche la posizione angolare assoluta del motore. Con 32 bit la massima distanza lineare che può essere percorsa dal robot senza perdita di informazione sarà:

$$S_{max} = 2\pi r \frac{s^{32} - 1}{P * M * n}$$

dove r è il raggio delle ruote, P è il numero di passi dell'encoder e n è il rapporto di riduzione del riduttore. Nel caso si debbano percorrere distanze più elevate allora sarà necessario aumentare la lunghezza di tale variabile o ricorrere ad un reset della stessa. Nell'applicazione di **SabotOne** questa operazione non è necessaria in quanto la posizione assoluta viene conservata sulla scheda SSB ed è continuamente mutuata dall'algoritmo di autolocalizzazione. Le considerazioni fatte sino ad ora per il motore il cui encoder è connesso al QEI possono essere estese anche al motore due con alcune piccole differenze; in questo caso un contatore del dsPIC conta gli impulsi provenienti dal qudrature clock converter tenendo presente il livello del bit di direzione.

Calcolo della posizione desiderata del motore

Dopo aver stabilito qual è la distanza angolare che ha percorso il motore occorre calcolare la posizione in cui dovrebbe trovarsi sulla base della traiettoria comandata. Per calcolare la posizione comandata si utilizza una variabile a 32 bit. Come per la posizione misurata, gli otto bit meno significativi della variabile della posizione comandata hanno il significato di frazioni di impulsi di conteggio. Quando il valore della velocità comandata è costante nel tempo l'albero del motore è mantenuto fermo in posizione fissa. Per assegnare una velocità al motore è necessario aggiungere una costante alla posizione comandata ad ogni periodo di campionamento. A tale scopo è definita la variabile a 32 bit velact che rappresenta l'effettivo valore della velocità comandata al motore. I 24 bit meno significativi della variabile velact rappresentano valori frazionali degli

impulsi di conteggio. Ad ogni periodo di campionamento la velocità effettiva velact è sommata alla posizione comandata. L'accelerazione (decelerazione) dei motori viene realizzata in modo analogo al calcolo della velocità comandata. Il parametro che controlla l'accelerazione viene inserito come un numero intero senza segno a 16 bit tramite interfaccia comandi ed è memorizzato nella variabile accel. Il valore contenuto in accel è sommato (sottratto) alla velocità comandata velact ad ogni periodo della routine di controllo. In figura 4.7 (pag.35) è riportato il diagramma di flusso dell'algoritmo di calcolo della posizione comandata.

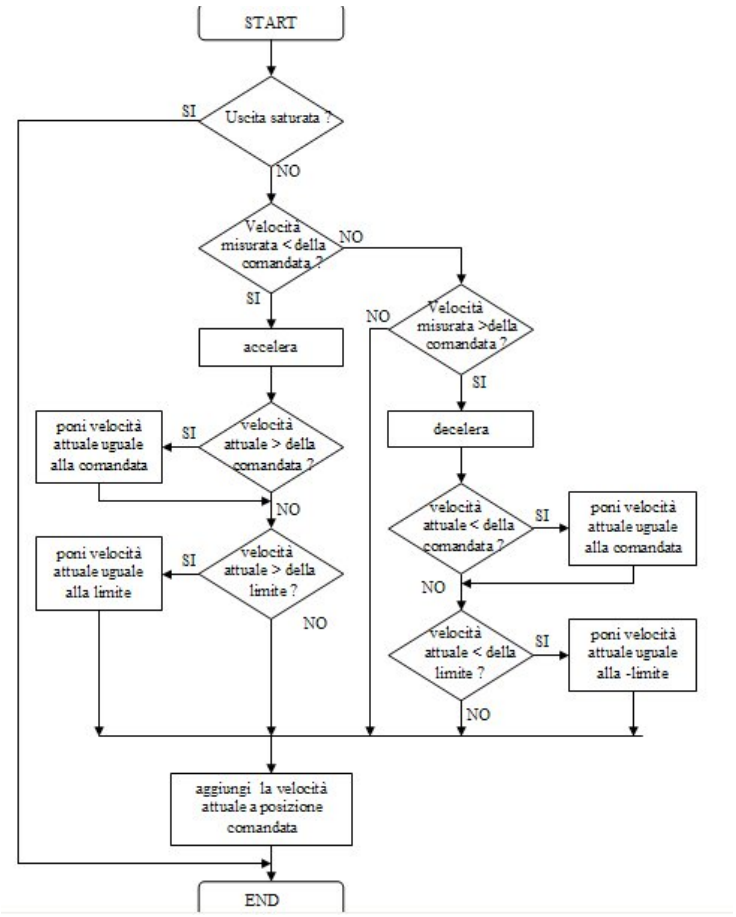

Figura 4.7: *Algoritmo di controllo sul profilo di velocità*

Calcolo dell'errore e algoritmo di PID

Successivamente si esegue il calcolo dell'errore di posizione del motore: il valore della posizione comandata è sottratto da quello della posizione misurata. Il risultato a 32 bit è posto nella variabile intera error. Uno shift sulla variabile error permette di eliminare la parte frazionaria. La rimanente parte intera a 24 bit dell'errore è arrotondata ad un intero a 16 bit per la successiva fase di elaborazione.

Il microcontrollore deve fornire il segnale per il corretto controllo del motore basandosi sul-

l'errore calcolato. È stato scelto di implementare un compensatore PID. In figura 4.8 (pag.36) è riportato il diagramma di flusso dell'algoritmo di calcolo dell'errore e del PID.

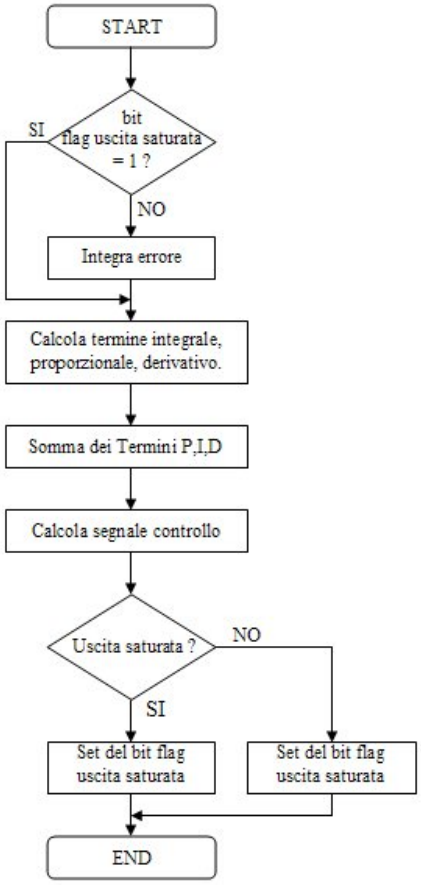

Figura 4.8: *Diagramma di flusso per la realizzazione dell'algoritmo PID*

Protocollo di comunicazione

Come è già stato detto nel capitolo 2 (pag.7) la scheda SACT dialoga con le altre unità presenti localmente a bordo del mobot mediante un protocollo punto/multipunto che è il protocollo ModBus. La scheda SACT è uno slave mentre il ruolo di master viene svolto dalla scheda SSB. Come già accennato, del rotocollo Modbus sono stati implementati solamente i due comandi 0x14 e 0x15 (Read file e Write file). In questo protocollo un file è un insieme di 10.000 record. Ogni record è costituito da due byte all'interno che possono rappresentare l'informazione in ASCII o in binario. All'interno del protocollo di **SabotOne** le informazioni sono organizzate come riportato in tabella 4.3.1 (pag.37)

Funzione	**File/Record**	**Read/Write**
Velocità motori	File: 1 Record: 1-2 Motore Left Record: 3-4 Motore Right	Write/Read
Accelerazione motori	File: 1 Record: 1-2 Motore Left Record: 3-4 Motore Right	Write/Read
Attiva il brake	File: 1 Record: 1 Motore Left Record: 22 Motore Right	Write/Read
Impostazione parametri PID	File: 1 Record: 1-2 P Motore Left Record: 3-4 I Motore Left Record: 5-6 D Motore Left Record: 7-8 P Motore Right Record: 9-10 I Motore Right Record: 11-12 D Motore Right	Write/Read
Errore di posizione motori	File: 1 Record: 1-2 Motore Left Record: 3-4 Motore Right	Read
Posizione motori	File: 1 Record: 1-2 Motore Left Record: 3-4 Motore Right	Read

Tabella 4.6: *Organizzazione dei file del protocollo Modbus per* ***SabotOne***

Parte II

Algoritmi implementati e definizione del protocollo di comunicazione *ControlML*

Capitolo 5

Path planning

5.1 Introduzione

Nel confronto fra vari metodi di pianificazione del moto le caratteristiche da tenere in considerazione sono principalmente:

- **completezza** ovvero la certezza che il metodo termini con la restituzione di una soluzione ammissibile, se questa esiste, altrimenti con il fallimento
- **complessità** sia spaziale che temporale, caratteristica particolarmente critica se si rilascia l'ipotesi di immobilità degli ostacoli
- **flessibilità** ossia possibilità di adottare il metodo su istanze diverse del problema

Sebbene completezza e flessibilità siano caratteristiche molto importanti, vengono spesso sacrificate in favore della complessità; accade cioè che metodi completi e generali, in casi particolarmente complessi, necessitino dei tempi di calcolo tali da renderli inutilizzabili. Questo problema è particolarmente sentito nei casi di ambienti fortemente dinamici in cui la pianificazione deve avvenire in tempi comparabili con i tempi di cambio della configurazione della scena. In letteratura il problema della pianificazione dei percorsi è stato risolto in svariati modi, di seguito ne vengono passati in rassegna i principali.

5.2 Metodi basati su roadmap

Si basano sulla costruzione di una mappa stradale $\mathcal{R}$ (i.e *roadmap*), di una rete cioè di cammini monodimensionali tutti contenuti in $\mathcal{C}_{free}$ o in $cl(\mathcal{C}_{free})$ ossia nella sua chiusura. Le soluzioni restituite sono tutte caratterizzate da una concatenazione di sottocammini di questo tipo:

- un sottocammino che connetta q_{init} a $\mathcal{R}$
- un sottocammino tutto contenuto in $\mathcal{R}$
- un sottocammino che connetta $\mathcal{R}$ a q_{goal}

Il punto cruciale del metodo è ovviamente il calcolo della *roadmap* che può essere fatto in svariati modi tra cui il grafo di visibilità e il diagramma di Voronoi.

5.2.1 Metodo basato sul grafo di visibilità

Il passo critico del metodo è la costruzione del grafo della roadmap. Nel caso peggiore, gli algoritmi calcolano il grafo con complessità pari a $\mathcal{O}(n^2)$ dove n è il numero dei vertici di $\mathcal{BW}$ (i.e dei poligoni che rappresentano gli ostacoli in $\mathcal{W}$). Alcuni di questi algoritmi eseguono una fase di preprocessamento di $\mathcal{BW}$ che ha complessità pari a $\mathcal{O}(n+c^2\log(n))$ per ottenere un grafo di visibilità ridotto dove c è il numero di poligoni convessi disgiunti in $\mathcal{W}$. Al termine di questo preprocessamento, il calcolo della roadmap ridotta avviene con complessità pari a $\mathcal{O}(c^2+n\log(n))$ Anche nel caso di complessità ridotta, questo metodo non si adatta ad ambienti con ostacoli in movimento in quanto sarebbe necessario rieseguire il preprocessamento a lato di ogni modifica di $\mathcal{BW}$

5.2.2 Metodo basato sul diagramma di Voronoi

La complessità dominante di questo algoritmo è anche in questo caso quella di costruzione del grafo e quella di ricerca dei cammini che per entrambe è di $\mathcal{O}(n\log(n))$. Anche in questo caso, la costruzione del grafo può essere eseguita a lato dell'inizializzazione del processo, ma il metodo non è adatto ad ambienti dinamici con ostacoli in movimento.

5.3 Metodi basati sulla decomposizione in celle

I metodi basati sulla decomposizione in celle hanno delle buone caratteristiche di efficienza. Essi si basano su una decomposizione in celle dello spazio libero $\mathcal{C}_{free}$ e sulla costruzione successiva di un grafo di connettivittà tra le celle. Le celle sono dei sottoinsiemi convessi dello spazio libero, la convessità assicura il fatto che un segmento rettilineo congiungente due configurazioni contenute nella stessa cella è sicuramente interno alla cella stessa e quindi libero. Il grafo di connettività ha come nodi le celle e due celle sono unite da un arco se e solo se sono adiacenti. Una volta diviso lo spazio in celle e costruito il grafo di connettività si deve ricercare una successione di celle che connetta quella contenente la configurazione iniziale a quella contenente la configurazione finale ed estrarre da questa successione un cammino. I metodi di decomposizione in celle sono divisi in due grandi classi in base al metodo di suddivisione dello spazio libero:

- decomposizione esatta dello spazio
- decomposizione approssimata

5.3.1 Metodi basati sulla decomposizione esatta dello spazio

I metodi di decomposizione esatta dello spazio produce una collezione di celle la cui unione è esattamente $\mathcal{C}_{free}$. È importante che le celle abbiano una geometria che permetta un semplice calcolo del cammino libero e che l'adiacenza tra due cella sia verificabile in modo efficiente. L'algoritmo di ricerca del percorso percorrerà i seguenti passi

- Generare una decomposizione esatta poligonale convessa $\mathcal{K}$ di $\mathcal{C}_{free}$
- Costruire un grafo di connettività $\mathcal{G}$ associato a $\mathcal{K}$
- Ricercare su $\mathcal{G}$ una sequenza di celle adiacenti tra quelle che comprendono q_{init} e q_{goal}

- Riportare in uscita la sequenza trovata se la ricerca ha successo

In uscita da questo algoritmo avremo in caso di successo, una successione di celle da cui estrarre il cammino. Un modo semplice per estrarre il cammino dalle celle è quello di considerare come punti del cammino i centri dei segmenti che delimitano una cella dall'altra, ed unirli tramite dei segmenti. In questo caso i passi dell'algoritmo sono dominati dalla complessità della costruzione della decomposizione e ricerca del cammino. Nel caso in cui si utilizzi una variante dell'algoritmo *sweep line* per la decomposizione e un algoritmo A^* per la ricerca della sequenza di celle, la complessità è per entrambe i processi di $\mathcal{O}(n\log(n))$ dove n sono i vertici dei poligoni.

5.3.2 Metodi basati sulla decomposizione approssimata dello spazio

Sono metodi che basano il loro funzionamento su alcune tecniche che scompongono lo spazio per via approssimata. Le celle in generale sono dei rettangoloidi la cui unione non restituisce l'intero spazio libero $\mathcal{C}_{free}$, ma una approssimazione conservativa. Dopo aver partizionato lo spazio le celle vengono classificate in tre categorie: **EMPTY** se sono completamente in $\mathcal{C}_{free}$, **FULL** se risiedono completamente nella zona $\mathcal{C}$-obstacle, **MIXED** se risiedono in parte in $\mathcal{C}_{free}$ e in parte in $\mathcal{C}$-obstacle. Un algoritmo che può essere utilizzato per la decomposizione dello spazio è il *quadtree*. Utilizzando questo metodo la complessità di spazio e tempo dell'algoritmo cresce esponenzialmente con le dimensioni dello spazio libero ed il metodo è quindi realisticamente applicabile per dimensioni inferiori a 4. Un altro aspetto critico di questo metodo è che non è completo, possono infatti verificarsi dei casi in cui esiste un cammino libero ma questo non viene trovato dall'algoritmo. In realtà la completezza dell'algoritmo dipende dalla risoluzione della decomposizione dello spazio.

5.4 Metodi basati sui campi di potenziale

I metodi basati sui campi di potenziale considerano il robot come un punto soggetto ad un campo di potenziale fittizio $\mathcal{U}$ che ne determina il moto. Di seguito si considera il caso di un punto materiale che può traslare liberamente nello spazio. In quest casi normalmente il potenziale viene costruito come somma di due funzioni che rappresentano rispettivamente un potenziale attrattivo $\mathcal{U}_a$ che sospinge il robot verso il goal e $\mathcal{U}_r$ che allontana il robot dagli ostacoli. La navigazione del robot avviene sotto l'effetto della forza fittizia $-\nabla\mathcal{U}(q)$. Avremo pertanto nel caso traslatorio che se $\mathcal{U} : \mathcal{C}_{free} \longmapsto \mathcal{R}$

$$\mathcal{U}(q) = \mathcal{U}_a + \mathcal{U}_r$$

la forza agente sul robot sarà pertanto data da

$$\mathcal{F}(q) = -\nabla_q\mathcal{U}(q) = -(\nabla_q\mathcal{U}_a(q) + \mathcal{U}_r(q)) = \mathcal{F}_a(q) + \mathcal{F}_r(q)$$

Quello che normalmente si fa, è definire due funzioni di potenziale, una per il campo lontano dal goal e una per il campo vicino al goal, definendo anche un valore di potenziale di raccordo per le due funzioni. Per il campo vicino si addotta la funzione

$$\mathcal{U}_{a1} = \frac{1}{2}k_a e^T(q)e(q)$$

che dà origine ad una forza di attrazione pari a

$$\mathcal{F}_{a1} = -\nabla\mathcal{U}_{a1}(q) = k_{a1}e(q)$$

Per il campo vicino si addotta invece la funzione

$$\mathcal{U}_{a2} = k_{a2}\|e(q)\|$$

che dà origine ad una forza di attrazione pari a

$$\mathcal{F}_{a2} = -\nabla\mathcal{U}_{a2}(q) = k_{a2}\frac{e(q)}{\|e(q)\|}$$

Viene poi definito un potenziale repulsivo associato agli ostacoli che normalmente ha forma

$$\mathcal{U}_r = \frac{1}{\gamma}\frac{k_r}{\eta(q)^\gamma} \text{ con } \eta(q) = \min_{q'\in\mathcal{CB}}\|q - q'\| \text{ e } \gamma = 1, 2$$

Il vantaggio di questo approccio è che nel metodo di pianificazione della traiettoria è inscritta anche una legge di moto e tale metodo si adatta alla così detta pianificazione *on-line* che permette di pianificare il moto *localmente*. Ciò significa che è possibile ripianificare la traiettoria molto frequentemente (grazie all'efficienza del metodo) considerando anzichè tutto $\mathcal{C}$ un suo sottoinsieme che possiamo chiamare *perceptual space*. Questo è la regione di $\mathcal{C}$ di sensing del sistema sensoriale. Questo metodo risulta particolarmente adatto alle situazioni in cui gli ostacoli sono in movimento e dove i metodi di pianificazione globale visti sopra comporterebbero un eccessivo carico computazionale. Il problema di questo metodo è legato alla possibilità di trovare delle funzioni per i potenziali attrattivi e per i potenziali repulsivi che non abbiano punti di minimo. Per questi casi sono stati studiati dei metodi di *fuga* dalle configurazioni di stallo. Altri metodi prevedono la generazione di campi che non abbiano minimo, è il caso dei campi vorticosi che tendono a far circumnavigare l'ostacolo da parte del mobot attraendolo sempre verso il goal. Questi metodi possono anche essere estesi ai casi in cui il robot oltre ad essere soggetto a gradi di libertà traslatori è soggetto a vincoli rotazionali.

5.5 Il metodo LPN

Il metodo LPN (*Linear Programming Navigation*) [LI],[Kon00] implementato sul Sabot, appartiene alla categoria dei metodi di pianificazione basati sui campi di potenziale. L'approccio in questo caso è puramente numerico e garantisce:

- basso costo computazionale
- flessibilità
- completezza

Il limite dell'algoritmo è quello di non generare sempre traiettorie a lunghezza minima che sarebbero quelle desiderate in molti ambiti, ma ha il vantaggio di poter considerare nella pianificazione la velocità relativa degli ostacoli permettendo di segliere in modo *intelligente* la traiettoria da seguire. In modo particolare in contesti in cui il mobot si muove in ambienti frequentati dalle

persone, è sempre preferibile fare in modo che vengano pianificate traiettorie che non *taglino la strada* alle traietorie che stanno seguendo le persone. Inoltre i vantaggi numerici che offre questo algoritmo, aggiunti al fatto che l'algoritmo è completo, lo rendono particolarmente adatto ai nostri scopi. L'idea di base dell'algoritmo è quello di costruire una *funzione di navigazione* che assegna un valore di potenziale ad ogni punto di $\mathcal{W}$. Seguendo il gradiente discendente della funzione calcolata si ottiene un cammino che conduce nella configurazione finale. Per l'applicazione di questo metodo, lo spazio $\mathcal{W}$ viene discretizzato in un insieme di punti di campionamento. In questo contesto un cammino non sarà altro che un insieme di punti di campionamento

$$P = \{p_1, \ldots, p_n\}$$

tale che:

- $p_i \in \mathcal{R}^2$
- p_i è adiacente a p_{i+1} $\forall i = 1, \ldots, n$

Ad ogni punto di campionamento viene attribuito un costo (valore del potenziale artificiale) mediante una funzione $F : \mathcal{P} \longrightarrow \mathcal{R}$ del tipo:

$$F(P_k) = \sum_{i=0}^{k} I(p_i) + \sum_{i=0}^{k-1} A(p_i, p_i + 1) \text{ con } p \in \mathcal{R}^2$$

dove:

- il dominio $\mathcal{P}$ è l'insieme dei cammini
- P_k è il cammino che dal punto p_k conduce al goal
- A viene chiamato *costo di adiacenza* ed assume il significato di costo di spostamento da una cella adiacente
- I viene chiamato *costo intrinseco* ed tiene in considerazione l'eventuale vicinanza di ostacoli o altri fattori quali ad esempio la pendenza del terreno

La *funzione di navigazione* sarà così definita:

Definizione 1. Dato un qualsiasi punto k dello spazio di lavoro, la funzione di navigazione nel punto N_k è pari al costo del cammino di costo minimo che parte da quel punto. Ovvero:

$$N_k = \min_{j=1,\ldots,m} F(P_k^j)$$

Dove:

- P_k^j è il j-esimo cammino che parte dal punto p_k ed arriva al goal
- m è il numero dei cammini che partono dal punto p_k ed arriva al goal

Se il costo intrinseco fosse nullo su tutto $\mathcal{W}$, la funzione di navigazione in un punto, sarebbe la distanza minima per quel punto dal goal. Seguendo il gradiente di N abbiamo la più veloce riduzione di costo del cammino, cioè il cammino a distanza minima che porta ad una configurazione finale.

5.5.1 Descrizione dell'algoritmo

Il metodo LPN permette il calcolo della funzione di navigazione precedentemente definita. Fare questo calcolo per via diretta in ogni punto dello spazio, avrebbe un costo proibitivo in quanto richiederebbe di iterare la procedura per tutti i cammini da ciascun punto dello spazio di lavoro. L'algoritmo LPN è una generalizzazione dell'algoritmo *wavefront* noto in letteratura [Lat91]. L'algoritmo *wavefront* consiste essenzialmente in due passi

1. **Inizializzazione** Al goal, o più in generale a tutti i punti appartenenti all'insieme delle configurazioni finali, viene assegnato un valore pari a 0. A Tutti gli altri punti si assegna un costo infinito. Si inseriscono i punti con costo pari a zero in una *lista attiva*.

2. **Espansione** Ad ogni iterazione tutti i punti con valore n vengono *espansi* dando ai loro quattro punti adiacenti un valore pari a n+1 se a questi non è ancora stato assegnato un valore oppure se sono punti appartenenti alla regione degli ostacoli. L'adiacenza è verificata solamente per celle che condividono un bordo e non in diagonale. In altri termini l'espansione viene eseguita partendo dai punti presenti in lista attiva. Cioè per ogni punto presente in lista attiva, rimuovilo dalla lista ed **aggiorna** tutti i suoi adiacenti. L'espansione prosegue fino a quando la lista attiva non è vuota.

Il processo di **aggiornamento** prosegue nel seguente modo:

1. per ciascuno degli otto punti q adiacenti al punto p secondo una griglia regolare, calcola calcola il costo secondo sommando al costo di p il costo di spostamento da p a q e il costo intrinseco di q

2. se il costo calcolato è inferiore al costo di q, aggiorna il costo di q e metti q nella lista attiva.

È possibile dimostrare [Kon00] la seguente proposizione

Proposizione 1. In ogni punto di $\mathcal{W}$ il gradiente della funzione di navigazione calcolata secondo l'algoritmo LPN, se esiste, punta nella direzione del cammino a minimo costo verso l'insieme delle configurazioni finali.

5.5.2 Benchmark

È stato realizzato l'algoritmo in C++. È stato altresì realizzato un simulatore che permettesse di inserire gli ostacoli con il relativo costo intrinseco, i punti della configurazione finale e il punto di partenza del mobot. Le simulazioni sono state eseguite su un PC con:

- Penthium III
- RAM 256 MB
- O.S WindowsXP

Il workspace $\mathcal{W}$ è stato partizionato in 10.000 celle. Sotto queste ipotesi l'algoritmo è riuscito a pianificare le traiettorie in un tempo massimo di 80ms con un impiego della CPU inferiore al 3%. Sono state inserite svariate configurazioni di posizionamento degli ostacoli per verificare la completezza dell'algoritmo. Anche questa prova ha avuto esito positivo.

Il metodo può essere integrato con un sistema che, mediante una opportuna fase di espansione, attribuisce un costo intrinseco alle celle "interessate" dal vettore velocità di un ostacolo in movimento. In altri termini se la velocità stimata di un ostacolo in movimento genera un'area di "interesse" che interseca l'area di interesse del mobot, alle celle appartenenti a quest'area viene attribuito un costo intrinseco più elevato che è funzione della velocità del mobot e della distanza. Il risultato sarà che il mobot tenderà a pianificare traiettorie alternative a quelle che lo porterebbero a *tagliare la strada* ad altri ostacoli in movimento.

Capitolo 6

Trajectory tracking

Nell'ottica di mantenere la struttura del sistema il più flessibile possibile, si è scelto di mantenere strutturalmente disaccoppiati i due sottosistemi di generazione delle traiettorie e di inseguimento delle traiettorie stesse (fig.6.1 a pag.49). In questo senso il path planner LPN si integra perfettamente in questa logica che, contrariamente ai metodi basati sui campi di potenziale, genera traiettorie puramente geometriche.

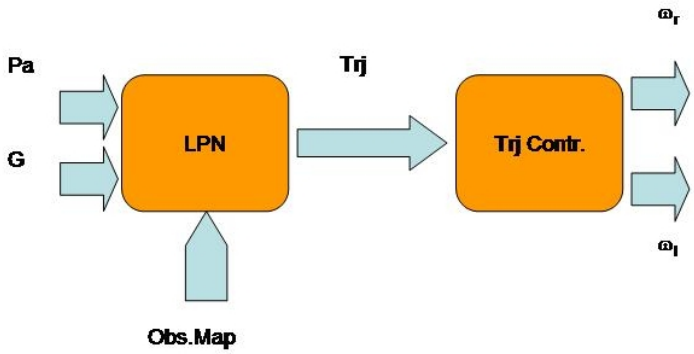

Figura 6.1: *Architettura del trajectory controller in relazione al path planner*

L'inseguimento della traiettoria è compito del controllore che deve "trasformare" le coordinate geometriche provenienti dal path planner in comandi elettrici da fornire agli attuatori per il corretto inseguimento tenendo conto della cinematica (ed eventualmente della dinamica) del mobot. Il controllore realizzato nel nostro caso considererà solamente la cinematica del sistema; si demanderà ad una futura attività la realizzazione di un controllore del secondo ordine.

6.1 Modello cinematico

Il moto di un mobot è un sistema fisico soggetto a p vincoli non olonomi (con $p < n$) che possono essere espressi nella forma generale[GO97]

$$A(X)\dot{X} = 0 \text{ dove } X \in \mathcal{R}^n \text{ è il vettore delle coordinate generalizzate} \tag{6.1}$$

La relazione 6.1 esprime il vincolo di rotolamento senza slittamento. Dato che in generale l'eq.6.1 non è integrabile la dimensione dello spazio delle configurazione non può essere ridotto. Tuttavia il vettore delle velocità generalizzate $\dot{X}$ deve soddisfare l'equazione

$$\dot{X} = G(X)u \text{ con } u \in \mathcal{R}^{n-p} \tag{6.2}$$

dove le $n-p$ colonne linearmente indipendenti di $G(X)$ sono una base dello spazio nullo di $A(X)$. L'equazione 6.2 è il modello cinematico del mobot.

La meccanica del mobot realizzato (fig.6.2 a pag.50) può essere modellata in modo approssimato con il modello fisico dell'uniciclo. Dato il vettore delle coordinate $X = \begin{bmatrix} x & y & \theta \end{bmatrix}$, le equazioni vincolari 6.1 saranno espresse come

$$\begin{bmatrix} \sin\theta & -\cos\theta & 0 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = 0 \tag{6.3}$$

mentre le equazioni 6.2 del modello cinematico nel caso dell'uniciclo possono essere riscritte come:

$$\begin{cases} \dot{x} = u_1 \cos\theta \\ \dot{y} = u_1 \sin\theta \\ \dot{\theta} = u_2 \end{cases} \tag{6.4}$$

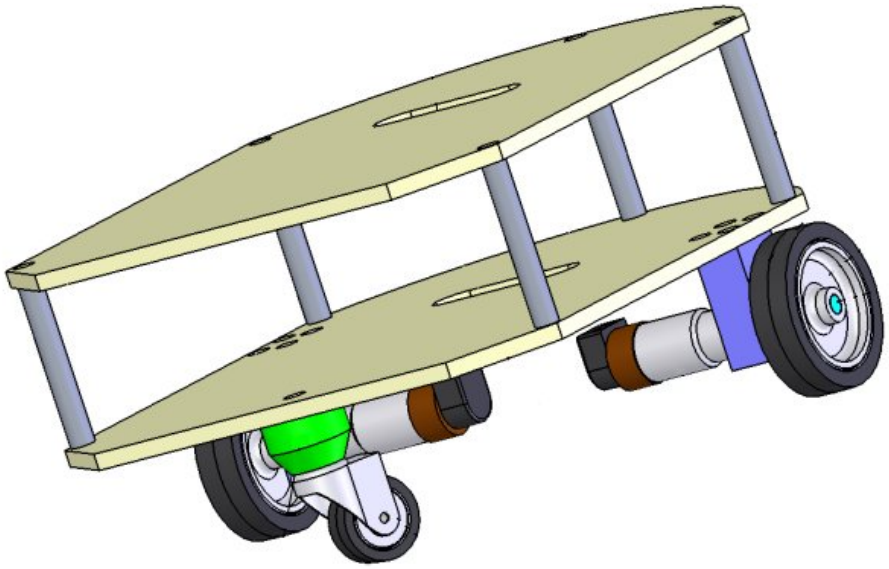

Figura 6.2: *Modello meccanico di* ***SabotOne***

Poichè si è scelto di azionare il mobot con il metodo differential drive, alle velocità generalizzate u_1 e u_2 che compaiono nelle equazioni 6.4 possiamo attribuire il seguente significato

$$\begin{cases} u_1 = r\frac{\omega_R + \omega_L}{2} \\ u_2 = r\frac{\omega_R - \omega_L}{d} \end{cases} \tag{6.5}$$

dove ω_R è la velocità angolare dell'asse destro e ω_L è la velocità agolare dell'asse sinistro. Pertanto u_1 rappresenta la velocità di traslazione mentre u_2 rappresenta la velocità di sterzata del mobot.

6.2 Soluzione del problema di controllo

Il modello del primo ordine 6.4 può essere esteso al modello del secondo ordine per includere la dinamica del sistema. Tuttavia tale estensione è immediata se si risolve il sistema al primo ordine. La presenza delle due variabili manipolabili per il controllo delle tre variabili di stato $X = \begin{bmatrix} x & y & \theta \end{bmatrix}$, rende il sistema *sottoattuato*. A supporto dell'algoritmo di controllo, viene eseguito un pretrattamento dei dati di traiettoria al fine di rendere tale curva una cubica di cui sono note le derivate prima e seconda in ogni punto. L'interpolazione è stata eseguita applicando l'algortimo *spline*[Wil02] mantendo parametrizzata la risoluzione dell'interpolatore la quale potrebbe essere stabilita a lato dell'applicazione dell'algoritmo di controllo. Sono stati realizzati due controllori differenti; quello che applica il metodo della *pseudoinversa* e quello che applica il metodo della *dynamic feedback linearization*.

6.2.1 Metodo della pseudoinversa

Data una traiettoria $X_d(t)$ un approccio semplice per la soluzione al problema di controllo è l'uso della pseudoinversa, ossia la ricerca della minimizzazione dell'errore $\|\dot{X}_d - \dot{X}\|$ tra la traiettoria desiderata e l'inseguimento mediante l'uso dei minimi quadrati[ADL94].

$$u = G^{\#}(X)\dot{X}_d = \left[G^T(X)G(X)\right]^{-1} G^T(X)\dot{X}_d \tag{6.6}$$

Applicando la relazione 6.6 al caso dell'uniciclo si ottengono i modelli delle attuazioni per eseguire l'inseguimento della traiettoria

$$u = G^{\#}(X)\dot{X}_d = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{x}_d \\ \dot{y}_d \\ \dot{\theta}_d \end{bmatrix} \tag{6.7}$$

Nel nostro caso la traiettoria viene fornita dall'algoritmo LPN di pianificazione della traiettoria. Poichè tale algoritmo viene eseguito per pianificare traiettorie in due dimensioni, l'output sarà costituito solamente da (x_d, y_d). Per ottenere θ_d si impone la seguente relazione:

$$\dot{\theta}_d = atan2(\dot{y}_d, \dot{x}_d) - \theta^{1} \tag{6.8}$$

suggerita dal fatto che l'uniciclo può eseguire istantaneamente un moto lineare solamente nella direzione del suo asse principale, viene in questo modo forzato un riallineamento di tale asse nella direzione data dal gradiente discendente della funzione di navigazione. Le espressioni dei comandi di feedforward del controllore che saranno pertanto[ADL94]

$$\begin{cases} u_1 = k_p(\dot{x}_d \cos\theta + \dot{y}_d) \\ u_2 = k_\theta(atan2(\dot{y}_d, \dot{x}_d) - \theta) \end{cases} \tag{6.9}$$

all'interno delle quali sono stati inseriti due coefficienti di guadagno k_p e k_θ per avere la possibilità di *pesare* l'azione di controllo. Si nota che le espressioni dei comandi di feedforward dipendono dallo stato attuale mediante il calcolo della pseudoinversa; il sistema nel complesso risulta quindi chiuso in retroazione.

[1] La funzione *atan2* è la funzione tan inversa sui 4 quadranti tale che $atan2(0,0) = 0$

Benchmark

Le simulazioni sono state condotte fissando una legge oraria per la conversione dal dominio geometrico al dominio del tempo. Al fine di ottenere la simulazione della risposta del sistema si sono integrate le eq.6.4 con il metodo di Runge-Kutta. Le simulazioni dimostrano che il controllore insegue la traiettoria con overshoot minimi in corrispondenza dei cambi di direzione. L'errore a regime viene annullato ma con transitori lunghi. Aumentando il valore dei parametri K del controllore si aumenta la prontezza del sistema generando tuttavia una lieve instabilità attorno al set-point a regime.

6.2.2 Metodo della Dynamic Feedback Linearization

Il secondo controllore implementato è il metodo della *Dynamic Feedback Linearization*. Il modello dell'uniciclo assicura la raggiungibilità di tutto il piano cartesiano, tuttavia il modello 6.4 linearizzato in qualsiasi punto non permette la controllabilità; in altri termini un controllore lineare non permette la stabilizzazione in un punto del mobot. È necessario pertanto progettare un controllore non continuo e/o non tempo invariante.

Si può invece dimostrare che è possibile inseguire la traiettoria con un controllore continuo a patto che in nessun punto della traiettoria stessa non vadano mai a zero simultaneamente sia la velocità lineare che quella angolare[GO02].

Inseguimento della traiettoria

Per l'inseguimento della traiettoria è necessario realizzare un controllo che generi dei comandi di *feedforward* combinati con una azione di *feedback*. Sia $X_d(t)$ la traiettoria generata dal path planner. Immaginando un punto che si debba muovere sulla traiettoria rispettando i vincoli dell'uniciclo, avremo che i comandi di feedforward possono essere espressi come:

$$v_d(t) = \pm\sqrt{\dot{x}^2(t) + \dot{y}^2(t)} \tag{6.10}$$

$$\omega_d(t) = \frac{\ddot{y}_d(t)\dot{x}_d(t) - \ddot{x}_d(t)\dot{y}_d(t)}{\dot{x}_d^2(t) + \dot{y}_d^2(t)} \tag{6.11}$$

$$\theta_d(t) = atan2(\dot{y}_d(t), \dot{x}_d(t)) + k\pi \tag{6.12}$$

dove k esprime un valore per il riallineamento iniziale del robot con la traiettoria da inseguire, in altri termini k viene scelto in modo che $\theta_d(0) = \theta_0$ con θ_0 direzione iniziale della traiettoria. Come si vedrà nel paragrafo 6.2.3, questi comandi possono essere espressi in forma puramente geometrica sostituendo tutte le derivate temporali con derivate spaziali mediante l'uso di una legge oraria che operi una mappatura tra punti di campionamento spaziali e punti di campionamento temporali.

Il feedback dinamico viene realizzato mediante un compensatore dinamico le cui equazioni

sono:

$$\dot{\xi} = u_1 \cos\theta + u_2 \sin\theta \tag{6.13}$$

$$v = \xi \tag{6.14}$$

$$\omega = \frac{u_2 \cos\theta - u_1 \sin\theta}{\xi} \tag{6.15}$$

e che permettono di vedere il sistema dal punto di vista degli I/O come:

$$\begin{aligned} z_1 &= x \\ z_2 &= y \\ z_3 &= \dot{x} = \xi \cos\theta \\ z_4 &= \dot{y} = \xi \sin\theta \end{aligned} \tag{6.16}$$

Il sistema esteso è quindi completamente linearizzato e descrivibile dalle due catene di integratori:

$$\begin{aligned} \ddot{z}_1 &= u_1 \\ \ddot{z}_2 &= u_2 \end{aligned} \tag{6.17}$$

Questo compensatore realizza un sistema dinamico lineare equivalente al modello dell'uniciclo in cui gli input sono disaccoppiati dagli output. Come già detto questo modello ha una potenziale singolarità nei punti della traiettoria in cui si annulla la velocità lineare. Di questo sistema è possibile realizzare una retroazione che stabilizza il sistema esponenzialmente e che ha un modello pari ad un compensatore PD[GO02]:

$$u_1 = \ddot{x}_d + k_{p1}(x_d - x) + k_{d1}(\dot{x}_d - x) \tag{6.18}$$

$$u_2 = \ddot{y}_d + k_{p2}(y_d - y) + k_{d2}(\dot{y}_d - x) \tag{6.19}$$

Stabilizzazione in un punto

Le funzionalità del tracking controller visto nella sezione precedente possono essere estese sotto alcune ipotesi al caso di stabilizzazione in un punto. Bisogna infatti evitare le configurazioni in cui $\xi = v = 0$, non solo in transitorio, ma asintoticamente (i.e durante l'avvicinamento al goal). Questo risultato può essere raggiunto mediante una opportuna scelta dei guadagni PD del controllore e una opportuna inizializzazione dello stato ξ del compensatore[ADL94]. Senza perdita di generalità possiamo assumere che l'origine sia la posizione finale, denotando

$$\mathcal{Q}^* = \{q \in \mathcal{Q} : (x = 0, \cos\theta \geq 0) \text{ OR } (y = 0, \cos\theta = -1) \text{ OR } (x = y = 0)\}$$

un sotto insieme di $\mathcal{Q}$ a cui prestare attenzione e la rimanente parte di $\mathcal{Q}$ con la sintassi $\mathcal{Q}/\mathcal{Q}^*$. Siano inoltre

$$\mathcal{Q}^r = \{q \in \mathcal{Q}/\mathcal{Q}^* : x \geq 0\} \tag{6.20}$$

$$\mathcal{Q}^l = \{q \in \mathcal{Q}/\mathcal{Q}^* : x < 0\} \tag{6.21}$$

Teorema 1. *Considerando il modello dell'uniciclo 6.4 sotto l'azione del compensatore dinamico 6.15, posto $x_d = y_d \equiv 0$ nella legge di controllo PD 6.19 scegliendo cioè:*

$$u_1 = -k_{p1}x - k_{d1}\dot{x} \tag{6.22}$$
$$u_2 = -k_{p2}y - k_{d2}\dot{y} \tag{6.23}$$

conduce ad una convergenza esponenziale da ogni configurazione iniziale $q_0 \in \mathcal{Q}/\mathcal{Q}^*$ sotto le seguenti ipotesi.

- *I guadagni $k_{pi} > 0$ e $k_{di} > 0$ (i=1,2) soddisfano la condizione*

$$k_{d1}^2 - 4k_{p1} = k_{d2}^2 - 4k_{p2} > 0 \tag{6.24}$$
$$k_{d2} - k_{d1} > 2\sqrt{k_{d2}^2 - 4k_{p2}} \tag{6.25}$$

- *Lo stato iniziale ξ^0 del compensatore viene scelto arbitrariamente come*

$$\xi^0 < 0 \quad \textit{moto indietro se} \quad q_0 \in \mathcal{Q}^r \tag{6.26}$$
$$\xi^0 > 0 \quad \textit{moto avanti se} \quad q_0 \in \mathcal{Q}^l \tag{6.27}$$

  e inoltre vale la seguente condizione

$$\xi^0 \neq 2\frac{k_{p1}x_0 \sin\theta_0 - k_{p2}y_0 \cos\theta_0}{k_{d2} - k_{d1}}$$

Per la dimostrazione di questo teorema si rimanda a[GO02]

6.2.3 Separazione tra legge oraria e geometria della traiettoria

Per la generazione dei comandi di feedforward, entrambe i metodi richiedono la conoscenza di una traiettoria $X_d(t)$ nel dominio del tempo. Il metodo LPN (come molti altri metodi di pianificazione), forniscono invece una traiettoria che è puramente geometrica. In un ottica di scalabilità di sistema, si desidera pertanto mantenere disaccopiati pianificatore delle traiettorie e controllore mediante una legge oraria $s = s(t)$ dove s è uno scalare. Il vincolo geometrico dato dalle eq. dell'uniciclo è così esprimibile come:

$$\begin{pmatrix} \frac{dx}{ds} \\ \frac{dy}{ds} \\ \frac{d\theta}{ds} \end{pmatrix} = \begin{pmatrix} \cos\theta \\ \sin\theta \\ 0 \end{pmatrix} v' + \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \omega' \tag{6.28}$$

In entrambe i metodi di controllo è possibile sostituire le derivate temporali con le derivate spaziali nelle equazioni dei comandi di feedforward potendo poi ricostruire i comandi stessi in funzione del tempo mediante le 6.30

$$v(t) = v'(t)\dot{s}(t) \tag{6.29}$$
$$\omega(t) = \omega'\dot{s}(t) \tag{6.30}$$

6.2.4 Benchmark

Il metodo della feedback linearization è stato simulato ottenendo traiettorie con overshoot minimi in corrispondenza dei cambi di direzione. L'errore a regime sull'inseguimento della traiettoria viene annullato ottenendo una risposta sufficientemente veloce del sistema e la stabilità asintottica sul setpoint inseguito. L'algoritmo è stato eseguito anche sul mobot reale facendogli eseguire delle traiettorie campione. La condizione più critica si è verificata nell'inseguimento di una traiettoria ad otto con lobo di larghezza pari a 2,5mt. L'errore riscontrato al ritorno sul punto di partenza è stato di 4cm. Da considerare tuttavia che la correzione è sempre stata eseguita sulla base della stima della posizione ottenuta in dead-reckonig con l'uso dei soli encoder montati sulle ruote.

6.3 Autolocalizzazione

Tutti gli algoritmi di trajectory tracking richiedono la conoscenza della posizione attuale del mobot. È pertanto indispensabile rendere disponibile questa informazione al regolatore. La stima della posizione può essere fatta in due modi differenti *dead reckoning* o *absolute position estimation*. Il primo metodo si basa esclusivamente sulle informazioni propriocettive del mobot senza alcun riferimento assoluto, mentre il secondo si basa sulla conoscenza da parte del mobot della posizione di alcuni riferimenti assoluti rispetto ai quali il mobot può misurare la propria posizione. Spesso questo secondo metodo viene utilizzato ad integrazione del primo al fine di correggere la deviazione dei sensori che normalmente è correlata temporalmente e spazialmente.

6.3.1 Dead-reckoning

Attualmente sul mobot è stata implementata esclusivamente una stima della posizione in dead-reckoning. Detto T_s il periodo di esecuzione dell'algoritmo di controllo, all'istante $t_k = kT_s$ la posizione può essere stimata mediante il modello [GO02]:

$$\hat{q}_k = \begin{pmatrix} \hat{x}_k \\ \hat{y}_k \\ \hat{\theta}_k \end{pmatrix} = \hat{q}_{k-1} + \begin{pmatrix} \cos\bar{\theta}_k & 0 \\ \sin\bar{\theta}_k & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \Delta s \\ \Delta\theta \end{pmatrix} \tag{6.31}$$

Nelle quali:

$$\Delta s = \tfrac{r}{2}(\Delta\phi_r + \Delta\phi_l), \quad \Delta\theta = \tfrac{r}{d}(\Delta\phi_r - \Delta\phi_l) \tag{6.32}$$

e $\bar{\theta}_k = \hat{\theta}_{k-1} + \Delta\theta/2$ è stata ottenuta integrando numericamente (Runge-Kutta al secondo ordine) il modello dell'uniciclo 6.4. Attualmente questa stima viene fatta utilizzando esclusivamente gli encoder ottici solidali all'albero motore. Tali encoder hanno una risoluzione di 512 impulsi giro che vengono successivamente moltiplicati per 4 dalla elettronica di condizionamento. La stima che viene fatta è abbastanza precisa se si fanno alcune ipotesi non sempre verificate: assenza di slittamenti, ruote indeformabili, contatto tra il pavimento e la ruota di tipo puntiforme (i.e pattinamento in curva nullo). Come si è già detto, quello che normalmente è possibile fare è integrare la stima della posizione eseguita in dead-reckoning con una stima della posizione assoluta che va a resettare la stima fatta in dead-reckoning. Il limite della procedura di stima della posizione assoluta basata sui landmark è quello di dover strutturare gli ambienti all'interno dei quali il robot si muove rendendo spesso eccessivamente costoso il processo di setup e complessa la calibrazione

dei mobot. D'altro canto oggi esistono algoritmi di *esplorazione/map-building/navigazione* che permettono di estrarre feature (i.e landmark naturali) e di adottarli come riferimento per il processo di autolocalizzazione[Gam01].

È quindi importante cercare di migliorare la stima eseguita in dead-reckoning con sensori di basso costo al fine di mantenere la stima della posizione il più possibile svincolata dalla risoluzione (i.e numero di landmark, dimensione del database delle feature naturali,...) dello stimatore della posizione assoluta[J.B], migliorandone in questo modo l'attendibilità[L.D03]. Nello schema di progetto, questo obiettivo verrà raggiunto utilizzando una piattaforma inerziale realizzata con sensori di tipo MEMS che hanno il pregio di avere un costo accessibile e delle prestazioni soddisfacenti. La stima della posizione viene poi *fusa* con la stima della posizione ottenuta mediante odometria. La *fusione* dei dati avviene mediante due EKF, uno che esegue un primo filtraggio dei dati provenienti dagli accelerometri e dai giroscopi[JSJ02], mentre un secondo EKF *fonde* i dati provenienti dal primo EKF con i dati odometrici (fig.6.3 a pag.56).

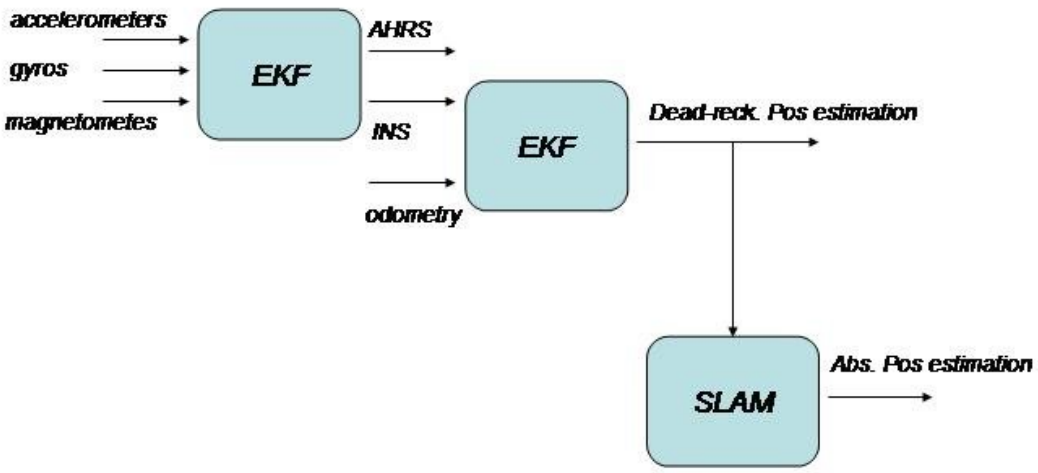

Figura 6.3: *Schema generale del posizionatore di* ***SabotOne***

6.3.2 Absolute position estimation

Come si è già avuto occasione di dire, esistono svariate tecniche per ottenere informazioni sulla posizione assoluta del mobot. Sicuramente uno dei metodi più affidabili è quello di adottare degli scanner laser per la misura della distanza del mobot dagli oggetti che secondo una determinata politica[Gam01] vengono o meno considerati degli ostacoli fissi (i.e pareti,...). Lo svantaggio degli scanner laser è che oggi questi dispositivi hanno costi inaccessibili per dei dispositivi orientati al mercato consumer (3.000 euro c.ca). Nello schema di progetto di **SabotOne** si utilizzerà piuttosto la visione artificiale per estrarre informazioni ambientali al fine di determinare la posizione reativa del mobot rispetto a dei riferimenti assoluti. Il punto debole della visione è che la riuscita di un algoritmo di elaborazione ha una forte dipendenza dalle condizioni

dall'illuminazione dell'ambiente e spesso anche dalla calibrazione di sistema. Con alcuni esperimenti preliminari, si è dimostrato che il primo problema (i.e scarsa illuminazione e controluce in prossimità delle finestre) potrebbe essere risolto utilizzando una telecamera con un filtro a IR totale illuminando poi l'ambiente con un illuminatore IR. Il problema della calibrazione verrà invece aggirato utilizzando un algoritmo di visione la cui precisione non dipenda dalla calibrazione del sistema di visione come succede nelle tecniche di stereovisione, o che richiedono un modello pin-hole della telecamera, piuttosto ci si affiderà ad algoritmi che prevedano l'estrazione di feature artificiali posizionate sul soffitto sotto l'ipotesi che il mobot si muova in piano. A tale scopo le feature artificiali che che si adotteranno saranno sagome in materiale catadiottrico con una elevata riflettività nell'ifrarosso.

Capitolo 7

Definizione dei modelli per il protocollo *ControlML*

7.1 Introduzione

I requisiti che devono essere soddisfatti da un protocollo di comunicazione orientato allo scambio di informazioni tra apparati controllati presenti su una rete e una o più centrali di controllo, possono essere classificati dai seguenti aspetti.

1. **Wireless** In quanto deve supportare dispositivi mobili
2. **Wide-band** In quanto deve supportare comunicazioni con informazioni ridondanti
3. **Session oriented** In quanto a tutti i dispositivi connessi deve essere riservata una sessione di comunicazione
4. **Client-server** In quanto deve poter accettare comandi da unità di controllo (i.e SCADA)
5. **Peer2Peer** In quanto deve poter "collaborare" in una "comunità" di simili
6. **Flessibile** In quanto deve poter supportare la definizione di nuove feature a seconda degli ambiti operativi del mobot

Quello che si andrà a definire nel corso del capitolo sarà la specifica per un protocollo di livello 7.

Il design del protocollo di comunicazione deve permette la distribuzione delle letture dei dati dei sensori in modo efficace. È possibile in questo modo soddisfare i seguenti requisiti:

- Rilevare i dispositivi presenti in rete
- Eseguire il logging di tutti i messaggi scambiati tra i dispositivi in rete, senza caricare i dispositivi stessi
- Gestire la sicurezza dell'accesso alla rete di controllo e rilevare in tempo reale nuovi dispositivi che richiedono l'accesso alla rete
- Disporre di un protocollo human readable per la gestione dei processi di debug

L'architettura di sistema è organizzata in modo che tutti i processi real time di elaborazione del segnale, vengano eseguiti sull'embedded PC; non è richiesto pertanto che il protocollo di comunicazione mobot-mobot e mobot-manager debba rispettare vincoli di determinismo stringenti.

7.2 Identità di un dispositivo/apparato

Definizione 2. Si definisce **Identità** di un *dispositivo/apparato*, l'insieme di tutti e soli i parametri necessari ad *identificare*, *localizzare* e *tracciare* il dispositivo di controllo.

7.2.1 Identificazione

Definizione 3. L'*identificazione* è il processo mediante il quale viene assegnato un ID unico all'interno di un dominio; questo viene utilizzato al fine dello scambio di messaggi tra i dispositivi

L'*identificazione* di un dispositivo di controllo è realizzata mediante un *JID* (Jabber ID) e un *NickName*.

JID Il Jabber ID viene assegnato al dispositivo dall'utente. Mediante un processo di registrazione-cancellazione presso un *Jabber/XMPP* server si garantisce l'unicità dei vari JID registrati all'interno di un determinato domino, accettando o rifiutando una richiesta di registrazione. In figura 7.1 a pag. 61 è riportato uno use-case del processo di registrazione. La semantica di un Jabber ID sarà

$$< JID >::=< name >'' @'' < server\_domain >$$

Ogni sessione di comunicazione mobot-mobot e mobot-manager, avviene facendo riferimento ai JID.

Nick name Il nick name è un mnemonico che può essere utilizzato per identificare il roulo del dispositivo all'interno di un processo di controllo. Trattandosi di un mnemonico svincolato da qualsiasi funzionalità, non viene fissata alcuna regola di codifica per questo attributo.

7.2.2 Localizzazione

La *localizzazione* è costituita da due parametri: il *parametro di localizzazione elettronica* e il *parametro di localizzazione geografica*.

Definizione 4. Definiamo il *parametro di localizzazione elettronica* l'URL dove è possibile trovare un web-portal che permette la definizione dei parametri di configurazione di un determinato dispositivo/apparato

Definizione 5. Definiamo il *parametro di localizzazione geografica* l'insieme delle stringhe contenenti la latitudine (in gradi decimali rispetto al Nord) e la longitudine (in gradi decimali rispetto a Est) di un dispositivo/apparato, valorizzate all'istante della richiesta della posizione.

Il parametro di localizzazione elettronica assume un particolare significato nei casi in cui si ha a che fare con un *apparato/dispositivo* il cui set-up dipende dalla dislocazione geografica.

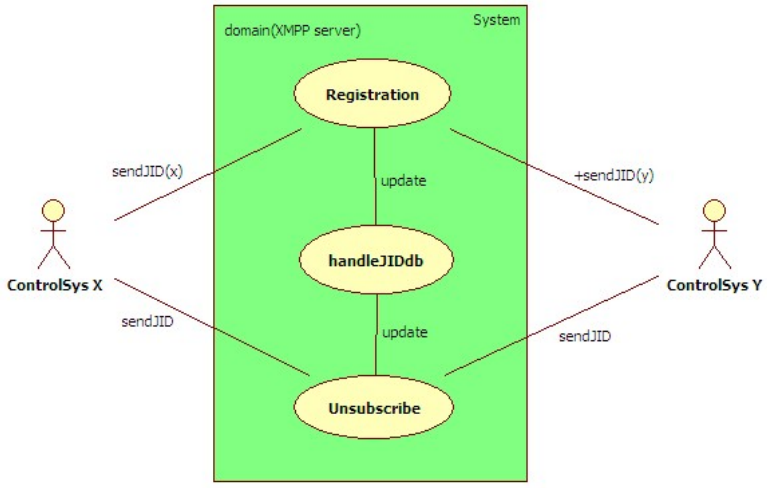

Figura 7.1: *Use case del processo di registrazione*

7.2.3 Tracciatura

Il processo di tracciatura permette all'ente che ha realizzato l'*apparato/dispositivo* di tracciare in modo univoco i parametri funzionali dei componenti meccanici, hw e sw, rilasciati su uno specifico *apparato/dispositivo* installato in campo. La tracciatura assume un particolare significato laddove si hanno molti apparati/dispositivi installati.

La tracciatura avviene mediante un URN (*Uniform Resource Name*) che rappresenta una sorta di matricola elettronica e un QR-CODE recante tutte le informazioni di sistema.

URN Viene utilizzato per associare al dispositivo un ID unico. L'unicità dell'ID generato e assegnato al sistema controllato, è garantita se si opera all'interno di un namespace appropriato. Il namespace che si decide di adottare è l'**UUID** (*Universally Unique Identifier*) all'interno del quale le chiavi sono di 128-bit e vengono generate con un algoritmo di hash SHA-1 (ISO/IEC 9834-8:2005, RFC4122). La struttura dell'URN definito dalla RFC2141 è:

$$< URN >::=" urn :"< NID >":"< NSS >$$

che nel nostro caso assumerà la forma:

$$< URN >::=" urn : uuid :"< NSS >$$

con NSS definito come:

$$< NSS >::=" \%0x.8 - \%x.4 - \%x.4 - \%x.4 - \%x.12"$$

Cioè l'NSS è rappresentato da 32 cifre esadecimali raggruppate in 5 insiemi di 8,4,4,4 e 12 cifre.

QRCode Nel QR-code viene codificato un documento xml che definisce le versioni dei vari componenti installati hw e sw installati sul sistema controllato.

Il QR-Code rappresenta una sorta di codice a barre evoluto che, con opportuni algoritmi, può codificare un grande quantità di informazioni. Si utilizza questa tecnica in quanto il Jabber/XMPP permette di inviare delle immagini a qualsiasi altro JID presente sulla rete. È possibile in questo modo codificare documenti di 4.296 caratteri, con algoritmi che hanno come output una immagine codificata; il ricevente potrà decodificare questa immagine con algoritmi avvalendosi di algoritmi particolarmente efficienti. L'uso di questa tecnica ci permetterà inoltre di contenere le modifiche al protocollo *Jabber/XMPP* limitatamente alla specializzazione di alcuni campi già presenti all'interno del protocollo stesso.

7.3 Presenza e livelli di presenza

Definizione 6. Definiamo il concetto di *presenza* come l'attitudine di un dispositivo/apparato a comunicare all'interno di una determinata infrastruttura di rete.

In questo senso la presenza in rete potrà essere di tipo *on-line* oppure *off-line*. Con l'accezione *on-line* non si è tuttavia definita l'attitudine di un dispositivo a portare a termine una determinata missione. Tale livello di specifica viene definita dai *livelli di presenza*. Un dispositivo/apparato *on-line* potrà essere:

- *Free* se disponibile ad accettare nuove missioni
- *Busy* se impegnato in un'altra missione
- *Away* se impegnato temporaneamente in attività di manutenzione

Busy definisce lo stato di un apparato di impossibilità a compiere una missione (e.g nuovo carico produttivo, un goal da raggiungere, ecc...), lo stato *Away* invece definisce l'impossibilità a prendere in carico una missione a breve termine a causa di attività di manutenzione che si stanno compiendo sull'apparato (e.g ricarica di batterie, esecuzione dei setpoint di sistema, ecc...).

7.4 Sistema percettivo e sistema motorio

Definizione 7. Definiamo il *sistema percettivo* come l'insieme di tutti i sottosistemi atti alla percezione delle variabili ambientali nell'intorno di un *apparato/dispositivo*

Al sistema percettivo è pertanto sempre associata una geometria spaziale che chiameremo spazio di percezione e che è definita dalla tipologia dei sensori utilizzati e dal loro posizionamento. Il sistema percettivo si riferisce sempre al sensing di variabili esterne all'*apparato/dispositivo* e non al sensing di parametri interni all'apparato stesso (e.g sensori di corrente per il controllo di un motore,...). Alcuni esempi di sistemi percettivi possono essere: degli scanner laser per l'obstacle avoidance posizionati su un mobile robot, delle barriere ottiche poste sugli accessi di aree pericolose di una macchina automatica...

Definizione 8. Definiamo il *sistema motorio* come l'insieme di tutti i sottosistemi mobili di un *apparato/dispositivo* che definisce la cinematica d'insieme del sistema

Il sistema motorio permette al mobot il movimento, secondo determinati vincoli cinematici, in uno spazio che è chiamato *workspace* [Lat91].

Definizione 9. Definiamo il concetto di *missione* come l'insieme di uno o più target che un determinato *apparato/dispositivo* deve raggiungere utilizzando il proprio *sistema percettivo* e il proprio *sistema motorio*

Capitolo 8

Design del protocollo *ControlML*

8.1 Introduzione

Nel corso di questo capitolo verrà definita la struttura protocollare del *ControlML* . Alcune strutture del protocollo vengono descritte nel corso del capitolo facendo ricorso ad uno pseudo DTD (Document Type Definition). Nell'Appendice B (pag.79) si trovano alcuni esempi applicativi di codice sviluppato per il sistema mobot SabotOne sviluppato nel corso di questa tesi.

8.2 Application layer

Regola 5. Il livello 7 dello standard ISO/OSI del protocollo che si adotta è il *Jabber/XMPP* (Extensible Messaging and Presence Protocol)

Rationale

- Gestione delle sessione di connessione
- Gestione dell'identità dei dispositivi connessi alla rete (il concetto di identità di un dispositivo verrà definito più avanti)
- Gestione della presenza in rete dei dispositivi
- Gestione del trasferimento file
- Gestione delle autorizzazioni alla connessione (roster list, certificati,...)
- Decentralizzazione di tutti i servizi associati al protocollo su una macchina esterna al mobot

8.3 *ControlML* layer

L'ultimo livello di definizione del protocollo è il *ControlML* layer che fornisce una specifica XML delle informazioni che vengono trasmesse attraverso il protocollo *Jabber/XMPP* . Questo layer si propone di definire alcune regole per incorporare il protocollo in via di standardizzazione denominato SensorML/SWE (Sensor Web Enablement) descritto in [MB07] e in parte fornisce dei *concetti* per alcune strutture di base del protocollo *Jabber/XMPP* . Nella figura 8.1 a pag.66 è rappresentata la stratificazione del protocollo ControlML.

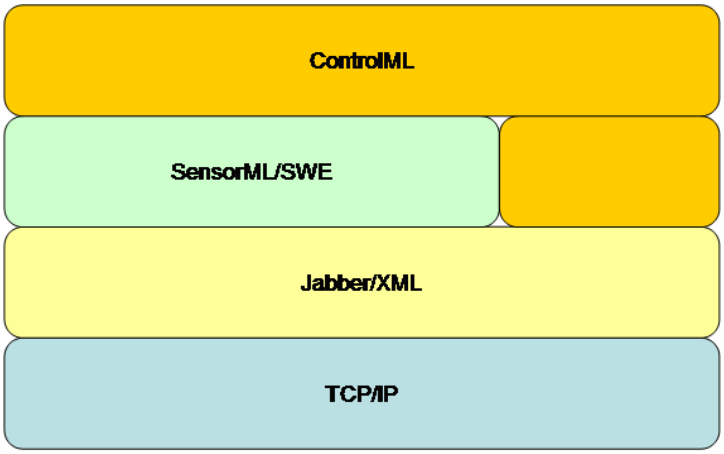

Figura 8.1: *Struttura del protocollo ControlML*

8.3.1 *Jabber/XMPP* : indentità

In riferimento alla **Definizione 3** (pag.60), viene data la seguente regola.

Regola 6. Nel protocollo *Jabber/XMPP* L'identità di un *apparato/dispositivo* è completamente specificata all'interno della struttura **vCard** definita dalla specifica RFC2426.

In particolare nel *ControlML* viene specificato un sottoinsieme dei campi definiti dalla RFC, lasciando gli altri ad un uso arbitrario. La definizione della vCard utilizzata in *ControlML* viene data di seguito utilizzando un DTD pseudo-codice (Document Type Definition)

```
<!-- Individual vCard container -->
<!ELEMENT vCard (
  (VERSION, FN, N),
  (NICKNAME?,
  PHOTO,
  ADR?,
  JABBERID,
  GEO?,
  ROLE?,
  ORG?,
  CATEGORIES?,
  REV?,
  SORT-STRING?,
```

```
  UID?,
  URL?,
  DESC?
)*)>
```

Identificazione

Regola 7. L'*identificazione* con il significato dato dalla **Definizione 3** (pag.60) è specificata dai campi:

- ```
  <!-- Nickname property. Multiple nicknames must be
      specified as a comma separated list value. -->
  <!ELEMENT NICKNAME (#PCDATA)>
  ```

- ```
  <!-- NOTE:   handle Jabber IDs; the value must be in the
       form of mobot_name.domain@host -->
  <!ELEMENT JABBERID (#PCDATA)>
  ```

Localizzazione

Regola 8. Il *parametro di localizzazione elettronica* nel senso dato dalla **Definizione 4** (pag.60) è specificato dal campo:

```
<!-- Electronic loc. parameter -->
  <!ELEMENT URL (#PCDATA)>
```

mentre il *parametro di localizzazione geografica* nel senso dato dalla **Definizione 5** (pag.60) è specificato dal campo:

```
<!-- Geographic loc. parameter -->
  <!-- The value should be specified to
    six decimal places.-->
  <!ELEMENT GEO (LAT, LON)>

    <!-- Latitude value. -->
    <!ELEMENT LAT (#PCDATA)>

    <!-- Longitude value. -->
    <!ELEMENT LON (#PCDATA)>
```

Tracciatura

Regola 9. L'URN definito nel paragrafo 7.2.3 a pag.61, è specificato dal campo

```
<!-- Unique identifier property. -->
  <!ELEMENT UID (#PCDATA)>
```

Regola 10. Il path all'immagine del *QR-Code* definito nel paragrafo 7.2.3 a pag.61, è specificato dal campo

```
<!-- URI to the external content. -->
  <!ELEMENT PHOTO (EXTVAL)>
```

Come si è già detto nel QR-Code viene codificato un documento la cui descrizione è rappresentata dal seguente pseudo-codice DTD (l'asterisco denota 0 o più occorrenze di un elemento, il + almeno una occorrenza dell'elemento)

```
<!-- QR-Code property container -->
<!ELEMENT QR-Code (
  (DTDDOC-VERSION),
  (MECHANIC*,
   ELECTRONIC*,
   SPECIAL_DEVICE*)
  )>

<!-- MECHANIC definition -->
<!ELEMENT
MECHANIC(DEVICE_ID,DEVICE_REV,MUL)>

<!-- ELECTRONIC definition -->
<!ELEMENT ELECTRONIC
(BOARD_NAME,BOARD_REV,BSP*,APPLICATION+,LIBRARY*)>

<!-- BSP/O.S definition -->
<!ELEMENT BSP (VERSION,PACKAGE*,MD5)>

<!-- APPLICATION definition -->
<!ELEMENT
APPLICATION(VERSION,NAME,MD5)>

<!-- LIBRARY definition -->
<!ELEMENT LIBRARY (VERSION,NAME,MD5)>

<!-- PACKAGE definition -->
<!ELEMENT PACKAGE (VERSION,NAME,MD5)>

<!-- SPECIAL_DEVICE definition -->
<!ELEMENT
SPECIAL_DEVICE(DEVICE_ID,DEVICE_REV,MUL,ROLE,DESC)>
```

La codifica del QR-code ricevuto avviene mediante la pipeline di elaborazione mostrata in figura 8.2 a pag.69:

Altri elementi della vCard opzionali

Ai rimanenti campi sono attribuiti i seguenti significati:

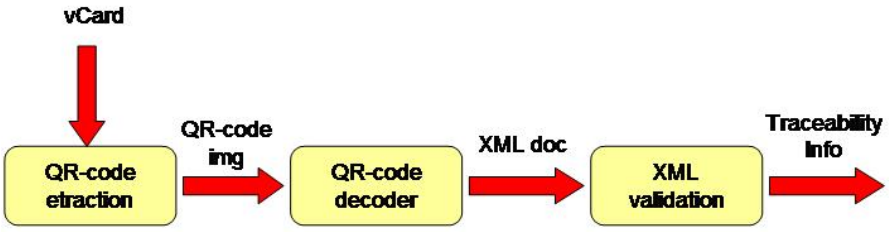

Figura 8.2: *Pipeline di elaborazione del QR-Code*

- **FN** È la categoria di un *apparato/dispositivo*
- **FAMILY** È il nome del costruttore
- **BDAY** È la data di rilascio o installazione in campo dell'*apparato/dispositivo*
- **ROLE** Contraddistingue la categoria di missioni a cui viene dedicato l'*apparato/dispositivo* (imbustatrice, service mobot,...)
- **ADR e TEL** Specifica gli estremi per contattare l'assistenza tecnica

8.3.2 *Jabber/XMPP* : presence signalling e presence levels

Per ogni dispositivo viene stabilito un euristico per l'abilitazione alla comunicazione con altri apparati connessi al medesimo server *Jabber/XMPP* .

Regola 11. Gli elementi che vengono utilizzati per la costruzione dell'euristico sono una combinazione logica di uno o più attributi contenuti nella vCard. A lato della connessione ad una rete, viene richiesta al server *Jabber/XMPP* la lista dei dispositivi attualmente connessi. Per ogni dispositivo non presente nella propria roster-list, viene eseguita una procedura di aggiornamento dei roster come indicato in figura 8.3 a pag.70

Questo processo viene eseguito solamente per i dispositivi non presenti nella roster-list. Per ogni dispositivo presente nella roster-list viene abilitato il meccanismo di *presence signalling* che permette di gestire in modo asincrono il *livello di presenza* (ref. definizione 6 a pag.62).

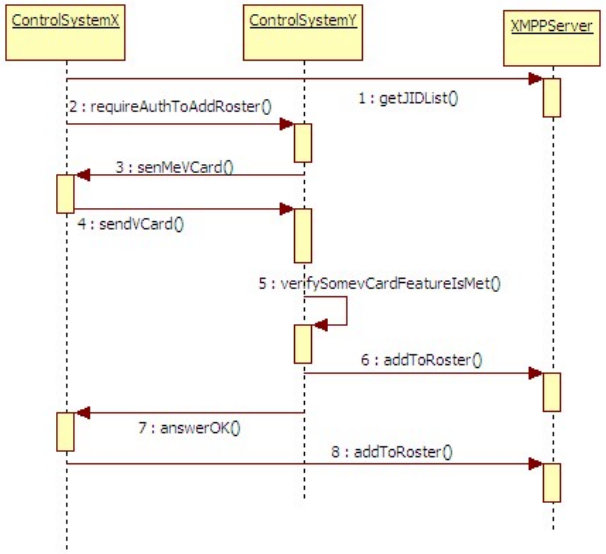

Figura 8.3: *Use case del processo di compilazione di una Roster List*

Se queste considerazioni sono valide per tutti i dispositivi che hanno un account presso uno stesso server *Jabber/XMPP* , viene invece introdotto un euristico diverso per abilitare la comunicazione con tutti i *apparato/dispositivo* che sono connessi a server *Jabber/XMPP* differenti.

Regola 12. Ogni dispositivo mantiene una lista locale di JID a cui connettersi con dominio diverso dal proprio. Per ogni JID con dominio diverso dal proprio non presente nella roster list, viene eseguito un processo di aggiornamento dei roster come riportato in figura 8.3 a pag.70 tranne per il fatto che non viene fatta alcuna verifica sugli attribuiti della vCard

8.4 Semantica del *ControlML*

La *Stanza* è un'unità di informazioni strutturate scambiate tra due entità in uno stream XML [Gro04]; in particolare nel *ControlML* la stanza rappresenta una sessione di comunicazione tra due apparati/dispositivi oppure tra un apparato e uno SCADA. La RFC-3920 definisce 3 tipi di stanza con i relativi attributi e elementi. Le stanza definite dal protocollo sono:

- $< message/ >$ per lo scambio di messaggi
- $< presence/ >$ per la segnalazione del livello di presenza
- $< iq/ >$ per transazioni di tipo request/response

Regola 13. Il protocollo *ControlML* oltre ad usare le stanza di base definite dallo standard, ne definisce una ad-hoc contraddistinta dal tag $< control - ml/ >$ inviato alla controparte all'inizio della stanza.

Definiamo tre attributi validi per questa stanza

- **type** Può assumere quattro valori: *get,set,error,result.*
- **id** È un identificativo unico della richiesta ed è codificato dalla regola

$$< REQ\_ID >::=< AAAAMMDD >":"< UINT32 >$$

- **res** È l'URN della risorsa su cui una entità vuole eseguire la richiesta

Il seguente pseudo-DTD definisce la struttura della stanza $< control-ml/ >$

```
<!ELEMENT control-ml (ERROR?|SENS-ML?))>
<!ATTLIST control-ml type (set|get|error|result) "get" #REQUIRED>
<!ATTLIST control-ml id ID #REQUIRED>
<!ATTLIST control-ml res CDATA #REQUIRED>
```

Una richiesta $< control-ml/ >$ di tipo *get* è un tag vuoto; la risposta è una stanza $< control-ml/ >$ di tipo $< result/ >$ se la richiesta era ben formata, oppure una stanza di tipo $< error/ >$ se la richiesta non era ben formata. Mentre per quest'ultima si adotta la semantica data dalla [Gro04], definiamo la prima come un elemento contenente uno o più sottoelementi SensML/SWE [MB07].

Una richiesta $< control-ml/ >$ di tipo *set* prevede un elemento di tipo SensML da fornire alla controparte; la risposta è una stanza $< control-ml/ >$ di tipo $< error/ >$ se la richiesta non era ben formata, oppure una stanza $< control-ml/ >$ di tipo $< result/ >$ riportante l'esito dell'operazione. Di seguito viene riportato un esempio di uno scambio tipico di richieste/risposte tra un ipotetico device richiedente e un device target.

Esempio 4.

```
Requesting                                                    Responding
  device                                                        device
----------                                                    ----------
    |                                                             |
    | <control-ml type='get' id='1234' res='cm.position'/>        |
    | ----------------------------------------------------------> |
    |                                                             |
    | <control-ml type='result' id='1234' res='cm.position'>      |
    | <sens-ml>  [....] </sens-ml>                                |
    | </control-ml>                                               |
    | <---------------------------------------------------------- |
    |                                                             |
    | <control-ml type='set' id='1235' res='cm.trajectory'>       |
    | <sens-ml>  [...]  </sens-ml>                                |
    | </control-ml>                                               |
    | ----------------------------------------------------------> |
    |                                                             |
    | <control-ml type='error' id='1235' res='cm.trajectory'>     |
    | <error>  [....] </error>                                    |
```

```
    | </control-ml>                                                          |
    | <---------------------------------------------------------------------  |
    |                                                                        |
```

Per la stanza $< control - ml/ >$ oltre agli attributi specifici definiti sopra, restano validi gli attributi comuni definiti per per tutte le altre stanza definite nel protocollo *Jabber/XMPP* . Di seguito è riportato un esempio con una richiesta $< control - ml/ >$ completa, mentre in Appendice B.3 è riportato il SensML/SWE di alcune grandezze definite a bordo del mobot SabotOne

Esempio 5.
```
<control-ml
from='workbench.unife@jabber.org'
to = 'sabotOne.unife@jabber.org'
    id='20081020:1234'
    type='get'>
   res = 'cm.position'/>
</control-ml>
```

8.5 Caratterizzazione del server *Jabber/XMPP*

Fermo restando che le caratteristiche del servizio *Jabber/XMPP* sono funzione del software scelto per realizzare il server, è tuttavia possibile identificare alcune caratteristiche comuni.

- **Flessibilità** tutte le caratteristiche del servizio sono configurabili mediante un file di configurazione.
- **Delay-delivery** il servizio si incarica di consegnare tutti i pacchetti anche qual'ora un *apparato/dispositivo* fosse off-line, a lato della sua riconnessione alla rete. Sarà cura del ricevente decidere se gestire il pacchetto o scartarlo.
- **Message logging** il servizio si incarica di mantenere un log (configurabile) riportante tutti i messaggi scambiati tra i *apparato/dispositivo*
- **Data storage** il servizio mantiene uno storage di tutte le vCard e le auth inserite
- **Data statistics** è possibile eseguire delle query al servizio al fine di ottenere delle statistiche sui messaggi scambiati

In Appendice B.4 a pag.82 sono presentati gli esiti di un benchmark del server *Jabber/XMPP* utilizzato nella nostra applicazione.

Conclusioni

E' stato realizzato un mobile robot avendo come obiettivo quello di ottenere una struttura scalabile sotto tutti gli aspetti: meccanico, hardware e software.

- Dal punto di vista meccanico, il sistema si presenta modulare al fine di poter espanso in altezza ed essere adeguato a svariati ambiti operativi.

- Dal punto di vista hardware è stata eseguita la progettazione della scheda di controllo assi sia per quanto concerne il dimensionamento delle parti analogiche, sia per quanto riguarda il progetto delle parti digitali. E' poi stato scritto il firmware per inizializzare correttamente tutti i componenti della scheda e gli algoritmi per eseguire un regolazione di velocità e di posizione di due motori in corrente continua

- Dal punto di vista del software è stata realizzata una architettura di astrazione del sistema operativo. Di questa si sono eseguiti due porting per i sistemi operativi Linux e vxWorks. Sul layer di astrazione del sistema operativo, è poi stata realizzata l'applicazione di gestione di bordo del mobot. In particolare si sono create le strutture per poter caricare dinamicamente gli algoritmi di controllo, di path planning e di autolocalization.

- Sono stati realizzati, simulati e integrati sul mobot i vari algoritmi path planning, autolocalization e trajectory control

- E' stato infine progettato un protocollo generico per la gestione di sistemi di controllo distribuiti che come caratteristiche principali supportasse:

  - Comunicazione real-time server driven
  - Gestione delle autenticazioni dei dispositivi e della segnalazione asincrona dello stato di presenza
  - Supporto alla lettura/scrittura di dati distribuiti

  Il protocollo così realizzato oltre a gestire il livello di presenza in una rete di un dispositivo mediante una sessione di autenticazione, permette la lettura/scrittura di variabili su dispositivi remoti in un formato IP/XML che specifica svariati attributi del dato stesso (unità di misura, istante di acquisizione,...)

La piattaforma è ora pronta per l'attività didattica di laboratorio e di ricerca nell'ambito della robotica mobile e dell'intelligenza artificiale.

Appendice A

Definizioni per il path planning

A.1 Spazio di lavoro e spazio degli ostacoli

Siano date le seguenti ipotesi:

1. Il robot è l'unico oggetto mobile nello spazio di lavoro
2. Non vengono prese in considerazione le propietà dinamiche del robot
3. Gli unici movimenti ammissibili sono quelli che non causano contatti tra gli oggetti presenti nel piano di lavoro

Sotto queste ipotesi il problema può essere così formulato:

> Sia $\mathcal{A}$ un singolo corpo rigido immerso in uno spazio euclideo $\mathcal{W} \equiv \mathcal{R}^n$ con $\mathcal{N}$ =2 o 3, detto *spazio di lavoro*. Siano $\mathcal{B}_1 \ldots \mathcal{B}_q$ gli ostacoli, ovvero oggetti rigidi fissi in $\mathcal{W}$. Allora il problema della pianificazione del moto si presenta come: data una locazione iniziale e una locazione finale di $\mathcal{A}$ in $\mathcal{W}$ generare un cammino τ significa generare una sequenza di locazioni di $\mathcal{A}$ che portano dalla locazione iniziale alla locazione finale prive di collisioni con $\mathcal{B}_i$.

A.2 Spazio delle configurazioni

Sia il robot $\mathcal{A}$ in una certa locazione un sottoinsieme compatto (cioè chiuso e limitato) di $\mathcal{W}$, e siano $\mathcal{B}_1 \ldots \mathcal{B}_q$ sottoinsiemi limitati di $\mathcal{W}$. Si indichino con $\mathcal{F}_\mathcal{W}$ e $\mathcal{F}_\mathcal{A}$ due sistemi di riferimento rispettivamente solidali con lo spazio di lavoro e con il robot. Una configurazione di $\mathcal{A}$ è quindi data dalla posizione ed orientazione di $\mathcal{F}_\mathcal{A}$ rispetto a $\mathcal{F}_\mathcal{W}$. Lo spazio delle configurazioni di $\mathcal{A}$ è lo spazio $\mathcal{C}$ di tutte le configurazioni possibili di $\mathcal{A}$. Una configurazione può essere descritta da un insieme di parametri reali, infatti una posizione generica di $\mathcal{F}_\mathcal{A}$ rispetto a $\mathcal{F}_\mathcal{W}$ è data dal vettore delle coordinate dell'origine di $\mathcal{F}_\mathcal{A}$ nel sistema $\mathcal{F}_\mathcal{W}$, mentre per quanto riguarda l'orientamento è possibile utilizzare una rappresentazione minima che faccia uso degli angoli relativi tra i sistemi.

Esempio 6. Per $\mathcal{N} = 2$ basta un angolo tra gli assi x dei due riferimenti $\mathcal{F}_\mathcal{A}$ e $\mathcal{F}_\mathcal{W}$, nel caso N=3 sono necessari invece un insieme di angoli di Eulero ϕ, τ, ψ.

Poichè ciascuna delle configurazioni di $\mathcal{A}$ può essere rappresentata da un numero di parametri reali indipendenti uguale alla dimensione di $\mathcal{C}$, detta n questa dimensione lo spazio $\mathcal{C}$ può essere rappresentato da $\mathcal{R}^n$.

Nota 1. E' importante osservare comunque che i due spazi non coincidono in quanto lo spazio $\mathcal{C}$ è più complesso di $\mathcal{R}^n$. Basti notare ad esempio che nel caso di un poligono in $\mathcal{R}^2$ le due configurazioni $q_1(\overline{x}; \overline{y}; 0)$ $q_2(\overline{x}; \overline{y}; 359)$ sono vicine in $\mathcal{C}$ ma non in $\mathcal{R}^3$. Lo spazio C ha cioè una topografia diversa rispetto ad $\mathcal{R}^n$, pertanto la struttura di $\mathcal{R}^n$ che ben serve a rappresentarlo non è in grado di catturarne tutte le proprietà.

Si definisce ora la funzione distanza per introdurre la nozione di continuità su $\mathcal{C}$

Definizione 10. La funzione distanza d su $\mathcal{C}$ è definita come

$$d : \mathcal{C} \times \mathcal{C} \longmapsto \Re$$

con

$$d(q, q') = \max_{a \in \mathcal{A}} \|a(q) - a(q')\|$$

dove:

- q, q' sono due configurazioni di $\mathcal{C}$
- a(q) rappresenta la posizione assunta dal punto a in $\mathcal{W}$ quando $\mathcal{A}$ è nella configurazione q
- $\|x - x'\|$ rappresenta la distanza euclidea tra i due punti x e x'

Con la definizione data di distanza è ora possibile introdurre la nozione di cammino nello spazione delle configurazioni

Definizione 11. Un cammino di $\mathcal{A}$ dalla configurazione iniziale q_{init} alla configurazione finale q_{goal} è una applicazione continua

$$\tau : [0, 1] \longmapsto \mathcal{C}$$

con

- $\tau(0) = q_{init}$
- $\tau(1) = q_{goal}$

A.3 Ostacoli nello spazio delle configurazioni

I concetti di spazio delle configurazioni e di cammino nello spazio delle configurazioni fino ad ora esposti non danno nessuna rappresentazione degli ostacoli presenti nello spazio di lavoro $\mathcal{W}$ Per caratterizzare i cammini che costituiscono una soluzione al problema della pianificazione del moto si deve dare una rappresentazione degli ostacoli nello spazio delle configurazioni. Si assume che gli ostacoli costituiscano una regione di $\mathcal{W}$ chiusa ma non necessariamente limitata.

Definizione 12. Per ogni ostacolo $\mathcal{B}_i$ consideriamo la sua immagine in $\mathcal{C}$ detta $\mathcal{C}$-obstacle definita come:

$$\mathcal{CB}_i = \{q \in \mathcal{C} : \mathcal{B}_i \cap \mathcal{A}(q) \neq \emptyset\}$$

Definizione 13. Si definisce la regione dei $\mathcal{C} - obstacle$ come:

$$\bigcup_{i=1}^{q} \mathcal{CB}_i$$

Definizione 14. Si definisce lo spazio libero delle configurazioni come:

$$\mathcal{C}_{free} = \mathcal{C} - \bigcup_{i=1}^{q} \mathcal{CB}_i \equiv \{q \in \mathcal{C} : \mathcal{A}(q) \cap \left(\bigcup_{i=1}^{q} \mathcal{CB}_i \right) \neq \emptyset\}$$

La regione $\mathcal{C}_{free}$ rappresenta in altri termini, l'insieme delle configurazioni che il robot può assumere senza incorrere in collisioni con gli ostacoli presenti in $\mathcal{W}$

Definizione 15. Un cammino libero tra due configurazioni q_{init} e q_{goal} è una applicazione continua definita come:

$$\tau : [0, 1] \longmapsto \mathcal{C}_{free}$$

con

- $\tau(0) = q_{init}$
- $\tau(1) = q_{goal}$

Appendice B

Definizione vCard per SabotOne

B.1 Definizione della vCard del mobot SabotOne

```
<vCard xmlns='vcard-temp'>
  <FN>Mobot</FN>
  <N>
    <FAMILY>Unife</FAMILY>
    <GIVEN>sabot</GIVEN>
    <MIDDLE/>
  </N>
  <BDAY>2008-12-31</BDAY>
  <NICKNAME>sabotOne</NICKNAME>
  <URL>http://192.168.1.10</URL>
  <ROLE>Demo mobot</ROLE>
  <ADR>
    <WORK/>
    <STREET>Via Saragat 1</STREET>
    <LOCALITY>Ferrara</LOCALITY>
    <PCODE>44100</PCODE>
    <CTRY>ITALY</CTRY>
  </ADR>
  <GEO>
    <LAT> 44,50 </LAT>
    <LON> 11,37  </LON>
  </GEO>
  <PHOTO>
    <TYPE>image/jpeg</TYPE>
    <BINVAL>
      Base64-encoded-QR-CODE-here!
    </BINVAL>
  </PHOTO>
  <JABBERID>sabotOne.unife@jabber.org</JABBERID>
  <DESC>
```

```
        Mobile robot di servizio, prototipo di sviluppo
    </DESC>
  </vCard>
```

B.2 Definizione del QR-Code del mobot SabotOne

```
<QR-Code xmlns='qr-code'>

<!-- Mechanical platform unicycle: base either top and bottom -->
<MECHANIC>
  <DEVICE_ID> base </DEVICE_ID>
  <DEVICE_REV> 1.1 </DEVICE_REV>
  <MUL> 2 </MUL>
</MECHANIC>

<!-- Mechanical platform unicycle: cylinders-->
<MECHANIC>
  <DEVICE_ID> cyl  </DEVICE_ID>
  <DEVICE_REV> 1.0  </DEVICE_REV>
  <MUL> 8 </MUL>
</MECHANIC>

<!-- Mechanical platform unicycle: engine brackets-->
<MECHANIC>
  <DEVICE_ID> eng_brk  </DEVICE_ID>
  <DEVICE_REV> 1.0  </DEVICE_REV>
  <MUL> 2 </MUL>
</MECHANIC>

<ELECTRONIC>
  <!-- Central intelligence -->
  <BOARD_NAME> SAI  </BOARD_NAME>
  <BOARD_REV> R10  </BOARD_REV>

  <!-- O.S Linux (Ubuntu 8.04LTS) -->
  <BSP>
    <VERSION> linux kernel 2.6.24-16 serrver</VERSION>
    <PACKAGE>
      <NAME> gcc </NAME>
      <VERSION>  4.2.3 </VERSION>
    </PACKAGE>
  </BSP>

  <APPLICATION>
```

```
    <NAME> SSB </NAME>
    <VERSION> 1.0 </VERSION>
  </APPLICATION>

  <!-- jabber/xmpp library -->
  <LIBRARY>
    <NAME> gloox </NAME>
    <VERSION> 0.9.9.5 </VERSION>
  </LIBRARY>

  <!-- logger library -->
  <LIBRARY>
    <NAME> log4cxx </NAME>
    <VERSION> N.A </VERSION>
  </LIBRARY>
</ELECTRONIC>

  <!-- SACT Sabot actuator -->
<ELECTRONIC>
  <BOARD_NAME> SACT board  </BOARD_NAME>
  <BOARD_REV> R2  </BOARD_REV>

  <APPLICATION>
    <NAME> sys_control</NAME>
    <VERSION> 1.1 </VERSION>
  </APPLICATION>
</ELECTRONIC>

<SPECIAL_DEVICE>
  <DEVICE_ID> electrical_engine </DEVICE_ID>
  <DEVICE_REV> N.A  </DEVICE_REV>
  <MUL> 2 </MUL>
  <ROLE> mobot traction</ROLE>
  <DESC> Gearhead 1:43 -- Torque 1,2 Nm -- Spee 111 g/min</DESC>
</SPECIAL_DEVICE>

<SPECIAL_DEVICE>
  <DEVICE_ID> inertial_sensor </DEVICE_ID>
  <DEVICE_REV> 4  </DEVICE_REV>
  <MUL> 1 </MUL>
  <ROLE> dead reckoning </ROLE>
  <DESC> 3 gyro, 3 accel, 3 mag </DESC>
</SPECIAL_DEVICE>

</QR-Code>
```

Il QR-Code corrispondente al documento XML di specifica dopra riportato è rappresentato dalla figura B.1 a pag.82

Figura B.1: *SabotOne QR-Code*

B.3 Esempio di definizione della grandezza posizione in SensML/SWE su SabotOne

fdfdf

B.4 Benchmark del server *Jabber/XMPP ejabberd*

I test sono stati eseguiti su

- **Hardware** P-IV@3GHz 1GB RAM
- **S.O** Ubuntu Linux kernel 2.6.24
- **Servizio** ejabberd 2.0.2

Il test consiste nell'invio di un certo numero di messaggi separati da un certo tempo. Viene misurato il tempo totale impiegato per l'invio di tutti i messaggi, il tempo di transito impiegato dal messaggio per andare dalla sorgente A al destinatario B e il carico del server. Il messaggio

inviato consiste in una stringa di 30Byte netti, esclusi quindi tutti i Byte di "Busta" aggiunti dal protocollo. I due attori della comunicazione sono connessi allo stesso server *Jabber/XMPP* su una stessa intranet.

Ritardo tra due messaggi consecutivi pari a 200usec:

- occupazione di CPU pari al 95% c.ca
- tempo totale $32.334sec$ per 100.000 messaggi
- tempo di transito $< 170\mu sec$

Ritardo tra due messaggi consecutivi pari a 50ms:

- occupazione di CPU $< 1\%$
- tempo totale $50.042sec$ per 1000 messaggi
- tempo di transito $< 1ms$

Ritardo tra due messaggi consecutivi pari a 5ms, si ottiene:

- occupazione di CPU $< 5\%$
- tempo totale $50.293sec$ per 10000 messaggi
- tempo di transito $< 2ms$

Bibliografia

[ADL94] Giuseppe Oriolo Alessando De Luca. Local incremental planning for nonholonomic mobile robot. *IEEE*, 1994.

[Dou06] Bruce Powel Douglass. *Real Time Design Patterns.* Addison Wesley, 2006.

[Eri98] Eric Gamma, Richrd Helm, Ralph Johnson, John Vlisides. *Design Patterns.* Addison Wesley-Professional Computing Series, 1998.

[Gam01] Gamini Dissanayake, Paul Newman, Steven Clark, Hugh F.Durrant-Whyte, M.Csorba. A Solution to the Simultaneous Localization and Map Building (SLAM) Problem. *IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATITION*, 17(3), 2001.

[GO97] Alessandro De Luca Giuseppe Oriolo. Planning Robot Motion. In Springer-Verlag, editor, *Chapter 4Feedback control of a nonholonomic car-like robot.* J.P.Lamound, 1997.

[GO02] Marilena Vendittelli Giuseppe Oriolo, Alessandro De Luca. WMR Control Via Dynamic Feedback Linearization: Design, Implementation, and Experimental Validation. *IEEE TRANSACTIONS ON CONTROL SYSTEMS TECHNOLOGY*, 10(6), 2002.

[gro02] Modbus group. MODBUS over Serial Line Specification and Implementation Guide V1.0, Febbraio 2002. http://www.modbus.org.

[Gro04] Network Working Group. Extensible messaging and presence protocol (XMPP): Core. RFC3920/Category: Standards Track, October 2004. http://xmpp.org/rfcs/rfc3920.htm.

[J.B] J.Borenstein, H.R Heverett, L.Feng. Where am I ? Sensor and Methods for Mobile Robot Positioning. Prepared by The University of Michigan for the Oak Ridge National Lab D&D Program and United State Departement of Energy.

[Jef06] Jeff McAffer, Jean-Michel Lemieux. *eclipse Rich Client Platform.* Addison Wesley, forth edition, 2006.

[JSJ02] Darren Liccardo Jung Soon Jang. Automatition of small uavs using a low cost mems sensor and embedded computing platform. *Giudance, Navigation, and Control Conference*, 2002.

[Kon00] Kurt Konolige. A gradient method for real time robot control. *AIROS*, 2000.

[Lat91] Jean-Claude Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, 1991.

[L.D03] L.D.L Perera, W.SWijesoma, S.Challa, M.D. Adams. Sensor Bias Correction in Simultaneous Localization and Mapping. *ISIF*, 2003.

[LI] Alessandro Farinelli Luca Iocchi. Planning trajectories in dynamic environments using a gradient method.

[MB07] University of Alabama in Huntsville Mike Botts, Alexandre Robin, editor. *OpenGIS Sensor Model Language (SensorML) Implementation Specification*, number OGC 07-000 in OpenGIS Implementation Specification. Open Geospatial Consortium, 2007.

[ROB04] ROB WARNER WITH,ROBERT HARRIS. *The Definitive Guide to SWT and JFace.* Apress, — edition, 2004.

[Wil02] William H. Press, William T. Vetterlin, Saul A. Teukolski, Brian P. Flannery. *NUMERICAL RECIPES The Art of Scientific Computing.* Cambrige University Press, second edition, 2002.