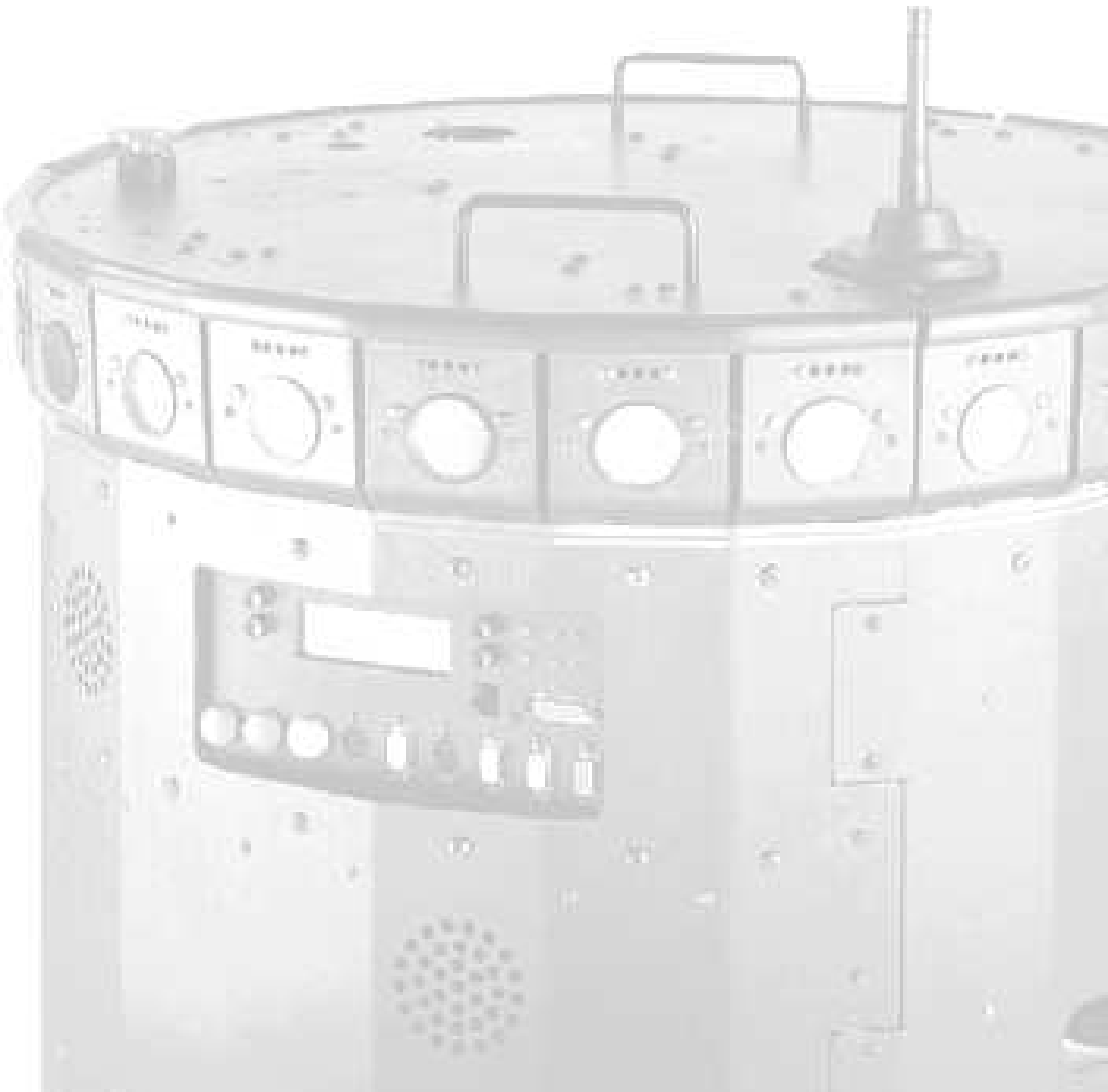
FABRIZIO ROMANELLI LUCIANO SPINELLO

GUIDA AL NOMAD XR4000

Manuale Hardware e Software del robot e della strumentazione integrata



Indice

Capitolo 1

Introduzione

Nel presente manuale gli autori si sono prefissi lo scopo di offrire all'utilizzatore una guida completa ed esaustiva all'hardware e al software del robot Nomad XR4000 della Nomadic Technologies. La documentazione di base ottenibile dai manuali a corredo del robot e della relativa strumentazione risultano infatti insufficienti ed obsoleti.

Gli autori, dunque si sono sforzati di produrre una nuova documentazione per rendere più accessibile lo sviluppo su tale piattaforma, grazie anche ai massivi aggiornamenti software che hanno reso possibile una provata affidabilità e robustezza rispetto alle condizioni originarie.

Il Nomad XR4000 del laboratorio di Controlli Automatici dell'Università di Roma Tor Vergata è stato prodotto nell'anno 2000 dalla Nomadic Technologies, società con sede nella Silicon Valley, CA, USA. Tale società però è stata in seguito assorbita dalla 3Com Technologies Inc. e il progetto Nomad è stato abbandonato, così come il supporto tecnico, lo sviluppo di software e driver aggiornati. Il Nomad XR4000 è composto da diversi tipi di sensori collegati tramite schede di interfaccia ad un computer dotato di microprocessore Intel Pentium III a 500 Mhz. Originariamente il sistema operativo funzionante sul Nomad era Red Hat Linux 5.1 con versione del kernel 2.0.35, poiché tutti i driver (motore, radio-lan, sintetizzatore vocale, sensori, frame-grabber) erano stati scritti per tale versione

del kernel, rendendo molto difficoltosa un eventuale aggiornamento a versioni più recenti di Linux.

L'affidabilità del sistema originario è risultata scarsa, in quanto serie di test sul funzionamento della strumentazione ha dato risultati poco soddisfacenti, causando blocchi di sistema spesso imprevedibili e quindi poca garanzia di funzionalità in un ambiente di sviluppo. Alcuni driver infatti erano ai primi stadi di sviluppo, quindi non privi di bachi, altri erano adattamenti da versioni ancora precedenti di Linux opportunamente modificati dalla Nomadic Technologies. Oltretutto il sistema pilotato di spegnimento non era correttamente implementato.

Nel manuale verrà estensivamente spiegato come è possibile installare e rendere pienamente funzionale il Nomad XR4000, partendo da un sistema *from the grass*[1].

[1]Sistema vergine, privo di dati.

Capitolo 2

L'hardware dell'XR4000

Il Nomad XR4000 è un sistema mobile avanzato che incorpora un sistema integrato di controllo, networking, power management e sensoristica utile allo studio nella manipolazione mobile, visione robotica, machine learning e navigazione sensoriale.

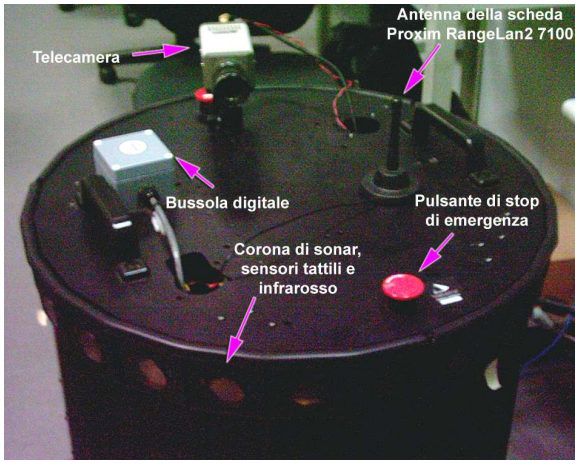


Figura 2.1: *Overview* del Nomad XR4000.

Il Nomad XR4000 (di cui è riportata una configurazione in figura 2.1) è costituito da un computer basato su processore Intel Pentium III collegato tramite

delle schede di interfacciamento al sistema sensoriale standard[1] (costituito da sensori sonar, sensori di prossimità ad infrarosso, e sensori tattili) al controller del motore e all'unità di potenza che fornisce alimentazione a tutte le apparecchiature. La strumentazione sensoriale del robot è inoltre completata da una bussola digitale, da un sistema laser di rilevamento ostacoli e da un sistema di visione ad alta velocità.

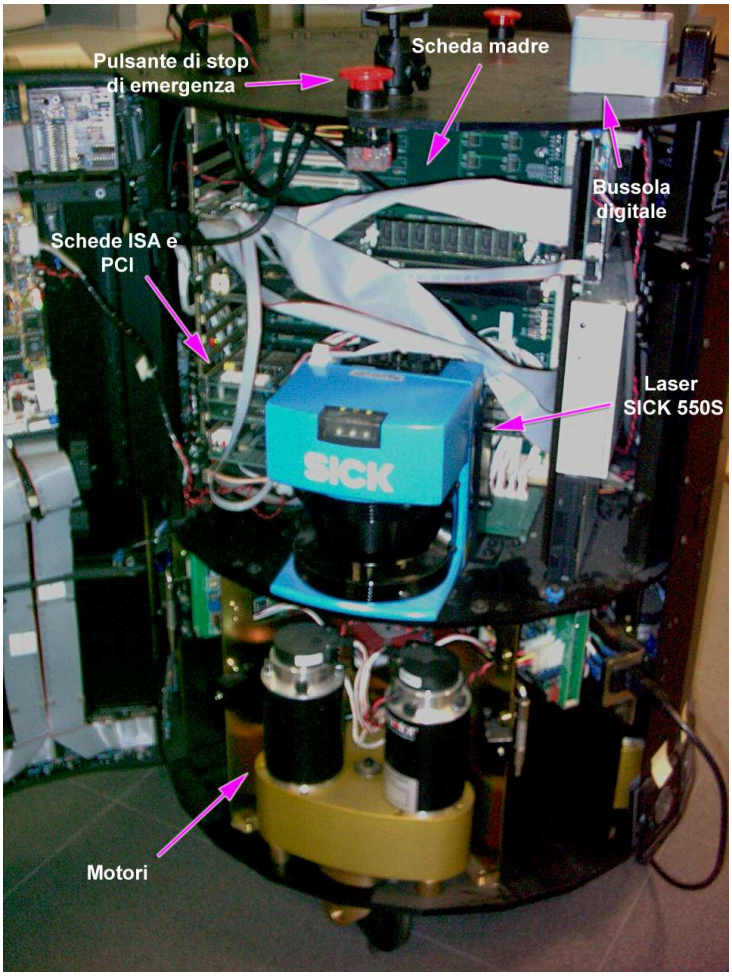


Figura 2.2: L'interno del Nomad XR4000.

[1] Si veda la sezione 2.5.1.

2.1 Operazioni elementari

Il Nomad XR4000 è un prisma a 24 facce di diametro $62cm$ e altezza $85cm$, sollevato da terra da quattro ruote di metallo stampato. Pesa circa $115Kg$ al netto delle sue quattro batterie[2] per un peso complessivo di $160Kg$.

In questa sezione vengono spiegate le operazioni di base da effettuare per la corretta attivazione del robot.

2.1.1 Aprire i portelli

Il robot ha tre portelli di accesso, mostrati in figura 2.3, ognuno con due fermi. Per aprire ogni porta è necessario premerli contemporaneamente fino allo scatto, e far ruotare con delicatezza il portello di metallo.

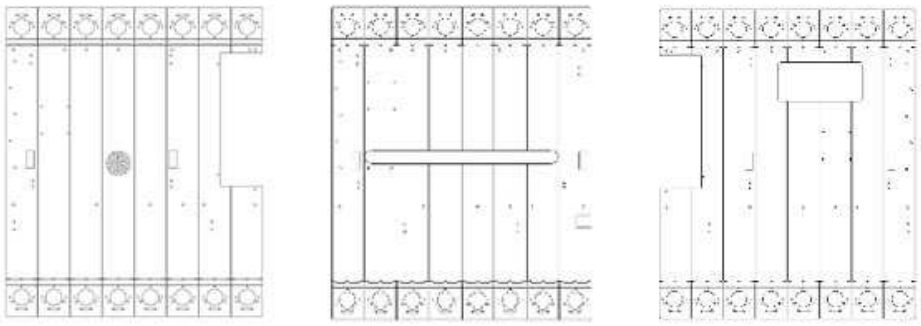


Figura 2.3: Portelli di accesso al robot XR4000.

2.1.2 Installare le batterie

A portello aperto, alzare il fermo posto davanti al vano batteria e inserirla con delicatezza fino allo scatto (nella figura 2.4 è riportata una foto del vano nel quale alloggiare la batteria). Rilasciare il fermo.

[2] Ogni batteria pesa 12kg.,

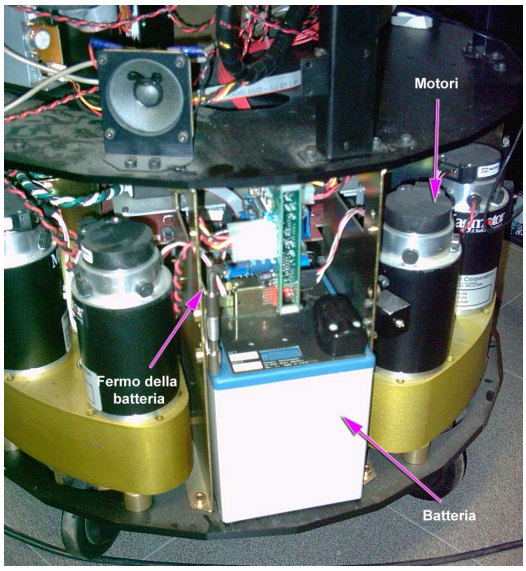


Figura 2.4: Vano batterie nel Nomad XR4000.

2.1.3 Chiudere i portelli

Spingere gentilmente, con le mani, sul portello vicino ai fermi fino a che è udibile lo scatto di chiusura. Se la porta non si chiude con facilità, non forzare. Verificare che i fermi delle batterie siano correttamente rilasciati e non creino ingombro.

2.1.4 Accendere il robot

Per accendere il robot è sufficiente premere il pulsante *On* sul pannello di controllo[3] ed attendere la corretta procedura di boot. Quando il Nomad XR4000 è operativo, per verificare le sue funzionalità basta eseguire dei semplici demo inclusi nel pacchetto di sviluppo.

[3] Si veda la sezione 2.7.

2.1.5 Comandi via *Joystick*

Appena il robot è acceso è possibile comandarlo in qualsiasi momento tramite un *joystick,* mostrato in figura 2.5, collegandolo all'apposito connettore del pannello di controllo.

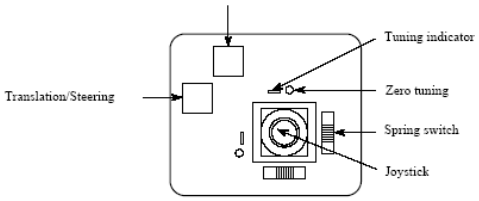


Figura 2.5: *Joystick* di comando.

Il *joystick,* può comandare il robot in tre diverse modalità:

modalità differenziale : tenendo premuto i pulsanti *steering* e *turret* il robot si muove in avanti utilizzando l'asse y del *joystick* e si gira utilizzando l'asse x del *joystick.* È la modalità di movimento più intuitiva.

modalità X-Y : tenendo premuto solo il pulsante *steering,* il movimento x-y del robot è comandato dal movimento degli assi x-y del *joystick*; il robot si muove nelle coordinate desiderate senza ruotare la base. Questa modalità dimostra la potente abilità del Nomad XR4000 di accelerare istantaneamente in qualsiasi direzione (movimento olonomo).

modalità *frisbee* : tenendo premuto solo il pulsante *turret* il robot si ferma, si muove lungo la direzione descritta dall' asse y del *joystick* e ruota tramite il movimento dell'asse x del *joystick*. Cambiare modo di movimento da *frisbee* a differenziale può essere fatto direttamente, creando un moto molto interessante.

2.2 Modalità di accensione

Il robot XR4000 ha 3 differenti stati di accensione:

1. Acceso
2. Standby
3. Spento

2.2.1 Acceso

Questa è la modalità di lavoro più frequentemente utilizzata nelle normali condizioni operative, in questa modalità tutti i sistemi sono alimentati compreso il pc di bordo. In questa modalità operativa se il robot viene collegato alla rete elettrica di alimentazione, le batterie vengono caricate e tutte le apparecchiature presenti vengono alimentate direttamente dalla rete elettrica. Il Nomad XR4000 viene posto nella modalità *Acceso* pigiando il pulsante *On* del pannello di controllo. Dopo qualche secondo dalla pressione del tasto, la rete SynapseNet (si veda la sezione 2.5.2) attiva i sonar, i sensori ad infrarosso e i sensori tattili di contatto. I sonar vengono attivati in maniera sequenziale su tutto il perimetro superiore ed inferiore del robot fino a che, dopo la corretta procedura di caricamento del sistema operativo, il demone *DRobot* provvede a porre nello stato quiescente la rete SynapseNet.

2.2.2 Standby

Questa modalità di risparmio energetico è utile, se il robot è collegato alla rete elettrica di alimentazione, per caricare le batterie senza fornire energia all'elettronica del robot, e quindi anche al suo pc. Si può attivare questa modalità semplicemente pigiando il tasto *Standby* sul pannello di controllo se il robot è spento. Se il robot è acceso, la pressione di questo tasto darà inizio a un processo di *clean shutdown* (si veda la sezione 2.2.3) che terminerà nello stato di *Standby.*

2.2.3 Spento

Il robot può essere spento con tre diverse procedure:

- Clean Shutdown
- Dirty Shutdown
- Auto Shutdown

Clean Shutdown

Questa è la procedura consigliata poichè spegne il pc di bordo in modo corretto, cioè dopo che il sistema operativo ha smontato i *file system* e terminato tutti i processi, spegne anche l'alimentazione a tutta la rete SynapseNet e porta il robot nello stato di *Spento.* Per attivare tale procedura premere una volta il pulsante rosso *Off* sul pannello di controllo, a seguito di un suono emesso dal robot premere nuovamente il pulsante *Off.* Il demone *DRobot* quindi darà inizio a un *clean shutdown* del pc; dopo circa 20s il robot si spegnerà.

Dirty Shutdown

Questo metodo forza il robot a uno spegnimento in qualsiasi momento della sua attività. È raccomandato che questa procedura sia applicata solo in caso di reale necessità, poichè non assicura lo spegnimento corretto del sistema operativo e della rete SynapseNet. Per attivare la procedura *dirty shutdown* basta tenere premuto per 10s il tasto *Off.*

Auto Shutdown

Quando la tensione delle batterie scende sotto i $10.8V$, l'XR4000 avvisa l'utente dello stato di scarsa carica, emettendo un suono acuto prolungato. Dopo aver udito tale suono è fortemente consigliato di spegnere il robot o collegarlo alla

rete elettrica; se entro 5 minuti l'utente non compie una delle due azioni, il sistema prova ad eseguire automaticamente un *clean shutdown.* Se anche questo tentativo fallisce, dopo altri 2 minuti di attesa, il robot esegue un *dirty shutdown.*

2.3 Sistema alto livello Intel Pentium III

All'interno del Nomad XR4000, è montata in verticale, saldamente ancorata a dei sostegni metallici, una particolare scheda stampata dotata di 4 slot ISA e 14 slot PCI. La scheda madre Portwell Robo-638Z è una scheda madre Pentium II/III utilizzata in processi industriali; poco più lunga di una scheda di espansione, essa occupa un particolare alloggiamento della scheda verticale. Questa scheda madre è molto compatta e ha un ottimo livello di integrazione, infatti su un così piccolo spazio è presente il doppio controller IDE UDMA/33, quattro slot per la RAM DIMM PC100, una scheda grafica AGP integrata, un controller ethernet, nonchè porte seriali e parallele[4].

Sulla scheda madre è presente un banco RAM da 128 MB. Il processore è una CPU Intel Pentium III a 500 Mhz[5]. Le unità di archiviazione di massa sono costituite da un hard disk da 20GB e un CD-ROM 24x. Questa è una configurazione che permette prestazioni computazionali assolutamente idonei ad applicazioni di robotica mobile e di fusione sensoriale.

Sulla scheda madre è integrata una scheda Ethernet Intel EtherExpress PRO 10/100 (per il collegamento alla rete tramite un comune cavo CAT-5 con connettore RJ-45) e una scheda video AGP ATI Mach64 8MB.

Un alloggiamento PCI è occupato dalla scheda di acquisizione, cuore del sistema di visione, basata sul chip Brooktree 878 che permette acquisizioni real-time fino a 30fps, con una massima risoluzione di 640x480. Le entrate disponibili sono

[4]Per una più completa trattazione si rimanda al datasheet della scheda madre Portwell Robo-638 fornito nel CD allegato.

[5]Originariamente la CPU Pentium II a 450Mhz, è stata sostituita per esigenze computazionali.

RC e S-Video. Sull'entrata RC è montata una telecamera a colori CCD Hitachi KP-D50, dotata di ottica Pentax regolabile manualmente (fuoco / otturatore) con un'ottima resa visiva.

Uno slot ISA è occupato dalla scheda di sintesi vocale DoubleTalk (*text-to-speech*), dotata dell'apposito regolatore di volume e collegata a un piccolo altoparlante.

Su un altro slot ISA è installata la scheda *radio-lan* Proxim Rangelan2 7100, collegata all'apposita antenna (posizionata all'esterno del robot) che permette il collegamento *wireless* verso altre postazioni compatibili. Sul lato di accesso della scheda sono presenti dei *dip-switch* per la configurazione dell'indirizzi di *I/O* e delle spie di indicazione di attività.

Un altro slot ISA è occupato da una scheda Nomadic Technologies *Intellisys 200 '12 axis servo controller'* che si occupa del controllo del motore e del processamento dei dati degli *encoder.*

L'ultima, ma non meno importante, scheda ISA presente è la scheda di interfacciamento con la rete *XR SynapseNet distributed network*[6], essa funge da ponte di comunicazione tra il sistema sensoriale di base e la scheda madre, in questo modo è possibile comunicare e ricevere dati l'uno dall'altra.

2.4 XR C8 Holonomic Drive System

Il sistema *XR C8 Holonomic Drive System*© è un sistema di guida olonomo a tre gradi di libertà (x,y,θ) senza restrizioni dovute allo spazio libero di operazione sul terreno, alle vibrazioni o alle complessità meccaniche. Questo risultato è stato raggiunto utilizzando due motori elettricamente indipendenti, uno per la sterzata e l'altro per il moto traslazionale, per ognuna delle quattro ruote che muovono il robot. L'utilizzo di queste quattro ruote lo rende un sistema ad otto assi sovravincolato. Per controllare questo sistema l'XR4000 utilizza uno

[6]Si veda la sezione 2.5.2.

speciale controller per il motore dotato di tre DSP e di un microcontrollore a 32-bit dedicato, in modo da controllare e stimare la posizione di questi otto assi.

Nella *tabella 2.1* è possibile valutare alcune caratteristiche tecniche del motore.

Max accel. traslazionale	$5m/s^2$
Max accel. rotazionale	$2rad/s^2$
Risoluzione encoder traslazione	$1120percm$
Risoluzione encoder rotazione	$422pergrado$

Tabella 2.1: Caratteristiche tecniche del motore.

2.5 Sistema sensoriale XR4000

Il *sistema sensoriale di base* del Nomad XR4000 consta di tre tipi di sensori:

- Sensori tattili a due livelli Sensus 150™
- Sensori di prossimità ad ultrasuoni Sensus 250™
- Sensori di prossimità ad infrarosso Sensus 350™

Questi sensori fanno parte e costituiscono la rete *SynapseNet distributed network* interna al robot.

Il *sistema sensoriale avanzato* è costituito da:

- Sistema Sensus 600 di orientamento
- Sistema laser di rilevamento degli ostacoli S550
- Sistema di visione

Il sistema sensoriale avanzato è collegato direttamente al pc di bordo, tramite collegamenti su porta seriale e non fa parte della rete *SynapseNet distributed network.*

2.5.1 Sistema sensoriale di base

Sensori tattili a due livelli Sensus 150™

Il sistema sensoriale tattile a due livelli Sensus 150 fornisce una completa copertura verticale delle superfici esterne del robot. Ci sono 48 elementi complessivamente installati sul perimetro superiore ed inferiore protetti da una robusta guaina di gomma che circonda il robot in alto e in basso. Ogni porta, inoltre, ha internamente 4 particolari interruttori flottanti che registrano l'urto della porta contro eventuali ostacoli, fornendo un informazione di collisione tra il perimetro superiore ed inferiore del robot. Nel momento dell'urto tra un ostacolo e i sensori tattili del perimetro superiore[7] (o inferiore) una spia rossa, sull'elemento in questione, si illuminerà, e resterà in tale stato per tutta la durata del contatto.

Sensori di prossimità ad ultrasuoni Sensus 250™

Il sistema Sensus 250™ utilizza 48 trasduttori sonar, mostrati schematicamente in figura 2.6, accuratamente posizionati a differenti livelli ed angolazioni sul perimetro superiore ed inferiore del robot, con sistema di rilevamento regolato tramite uno specifico ordine per ottenere una copertura completa, dalla superficie inferiore alla superficie superiore. L'ordine e la velocità di acquisizione può essere specificata via software a seconda dell'esigenza dell'utente. Il sistema Sensus 250™ ha un'accuratezza che va dai $15cm$ ai $650cm$ con un margine di errore dell 1% sul range massimo (cioè $6.5cm$). Il funzionamento del singolo sensore sonar è basato sulla registrazione del tempo intercorrente tra una emissione sonora acustica e il suo eco di ritorno; moltiplicando il tempo per la velocità del suono a cui viaggia l'impulso sonar, si ha una precisa misura di distanza. I 24 sensori dell'anello perimetrale superiore sono angolati di 2.5^0 verso il basso, viceversa i 24 sensori dell'anello perimetrale inferiore sono orientati di 2.5^0 verso l'alto; questo permette di ridurre al minimo le zone non coperte dalla rilevazione dei sonar al centro delle pareti verticali. È consigliato fare in modo che ogni coppia

[7] Comunemente chiamati *bumpers* o sensori di collisione.

di sensori (in alto e in basso) sia attivata simultaneamente, in questo modo è possibile ottenere che l'energia emessa dal sonar inferiore sia riflessa dall'ostacolo e acquisita dal sonar superiore e viceversa, in modo da conseguire una rilevazione più accurata e robusta. Quando un sonar viene attivato è possibile udire un piccolo *click* e nello stesso istante vedere brillare la spia verde dell'elemento.

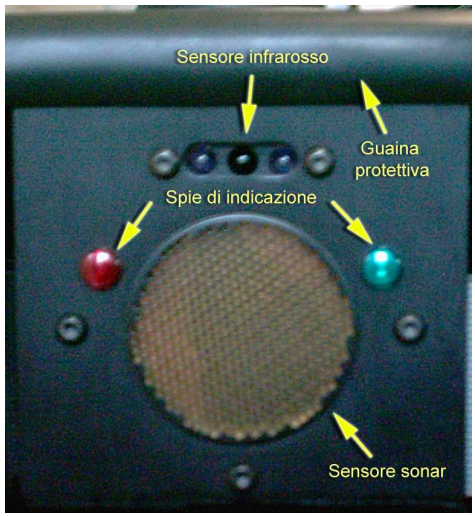


Figura 2.6: Elemento contenente il gruppo di sensori di collisione / sonar / infrarosso.

I sensori sonar sono comuni trasduttori Polaroid pilotati da una scheda di controllo *Polaroid 6500 ranging board*. Per la rilevazione dati, questa emette 16 periodi di un onda quadra a frequenza $49.4kHz$ attraverso un trasduttore elettrostatico, quindi un periodo di *blanking*[8] in cui il trasduttore si stabilizza. Il traduttore viene, adesso, fatto agire come un ricevitore, processando gli echi ricevuti attraverso un amplificatore di guadagno tempo-variabile in cui il fattore di guadagno aumenta con il passare del tempo per compensare le perdite di potenza del segnale, dovute all'attenuazione del suono nell'aria. L'uscita dell'amplificatore è sottoposta ad un circuito-soglia in cui se la soglia viene superata viene calcolata

[8] Il *blanking* è una finestra temporale in cui il sensore non viene attivato, viene cioè fatto stabilizzare.

la durata temporale del percorso dell'impulso sonar, e convertita in distanza tramite un appropriato fattore di correzione.

Il trasduttore sonar non emette energia omogeneamente in tutte le direzioni bensì forma lobi di intensità decrescente. Il trasduttore Polaroid, in particolare, emette energia in maniera non simmetrica nello spazio[9], e varia da trasduttore a trasduttore, questo effetto è di maggiore visibilità nei lobi più periferici; si veda a riguardo la figura 2.7. L'attenuazione dell'ultrasuono nell'aria aumenta con la frequenza e dipende dalla temperatura e dall'umidità. Valori tipici di attenuazione per frequenza di emissione 50kHz, temperatura $17 - 28^0 C$, umidità 15-70%, è di 0.6-1.8 dB/m. Il suono nell'aria è espresso tramite la costante $C = 33.4\sqrt{T/273} m/s$, dove T è la temperatura dell'ambiente in gradi Kelvin.

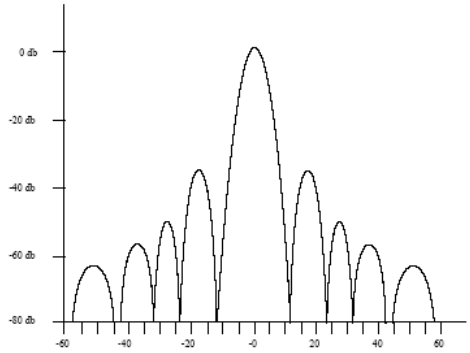


Figura 2.7: Distribuzione energetica del raggio ultrasonico di rilevamento.

I seguenti errori di misurazione sono dovuti a peculiarità tecnologiche della circuiteria del sensore[10]:

errori dovuti alla durata dell'impulso trasmesso : tutte le misurazioni del tempo sono basate sull'assunzione che la parte iniziale dell'impulso trasmesso sia la parte dell'eco di ritorno che supera la soglia del ricevitore. Se questo non avviene l'errore può arrivare fino a $23cm$.

[9] Frutto di risultati sperimentali di Lang, Kuc, Leonard.

[10] Risultati basati sugli studi di Leonard.

errori dovuti all'amplificatore a guadagno tempo variabile : La curva esponenziale ideale di guadagno che annullerebbe l'effetto dello *beam-spread*[11] e delle perdite di attenuazione è stata approssimata con una funzione a gradino costituita da 12 tratti costanti[12]. Visto che l'energia riflessa è funzione dell'angolo di incidenza l'angolo di visibilità cambia con la distanza.

errori dovuti al tempo di carica capacitiva nel circuito soglia : per segnali fortemente riflessi, tre periodi sono sufficienti per raggiungere la soglia[13]. Per segnali più debolmente riflessi, il tempo di carica si allunga notevolmente, rilevando distanze erroneamente allungate. Una delle maggiori fonti di incertezza sulla stima della distanza, a causa di queste caratteristiche, è la rilevazione successiva di segnali deboli e segnali forti. Un eco di ritorno forte possiede sufficiente energia per superare la soglia rapidamente, dando luogo, in questo modo, ad una misura molto precisa. Un eco di ritorno debole può causare un errore di distanza dovuto alla lunga attesa per oltrepassare la soglia, che viene superata solamente da una casuale combinazione del tempo di carica e delle variazioni del valore del guadagno, dovute alle discontinuità della caratteristica non lineare dell'amplificatore tempo variabile.

I *bersagli* possono essere divisi in due gruppi:

1. Oggetti riflettenti, di dimensioni più grandi della lunghezza d'onda ($6.95mm$ a 20^0C). Superfici lisce, muri dipinti e porte sono tipici oggetti riflettenti; oggetti che si comportano come perfetti specchi per le onde sonore.
2. Oggetti diffrattori, di dimensioni più piccole della lunghezza d'onda. Gli oggetti la cui dimensione complessiva è più piccola della lunghezza d'onda sono di poco interesse ma superfici ruvide come

[11] Fenomeno di perdita di energia dovuto al cono di rilevazione.

[12] La curva ideale è comunque funzione della temperatura e dell'umidità, per cui la funzione costante a tratti rappresenta una approssimazione piuttosto indicativa.

[13] Dopo una corretta calibrazione.

cemento, o pareti trattate, presentano piccole asperità che hanno un comportamento diffrattore.

Superfici arrotondate convesse di raggio paragonabile alla lunghezza d'onda producono un debole eco di ritorno. Una conseguenza delle superfici riflettenti è il multiplo effetto eco: l'onda sonora rimbalza diverse volte tra il muro e il sonar raggiungendo il ricevitore più volte. A causa delle perdite per attenuazione nell'aria il fenomeno dell'eco multiplo è quasi del tutto trascurabile, poichè i livelli energetici di tale impulso sono molto bassi ma in particolari situazioni potrebbero essere significativi.

Sensori di prossimità ad infrarosso Sensus 350^{TM}

Il sistema Sensus 350^{TM} utilizza 24 trasduttori infrarosso posizionati sia sul perimetro dell'anello superiore del robot, sia su quello inferiore. Questi sensori hanno l'utilità di essere sensori con una frequenza di acquisizione molto alta per una rilevazione di ostacoli vicini $(30 - 50cm)$. La rilevazione viene effettuata emettendo una radiazione infrarossa generata attraverso LED del tipo *high-current* e rilevando la quantità di energia di ritorno dall'ostacolo attraverso fotodiodi infrarossi. Sapendo che la quantità di energia riflessa è inversamente proporzionale alla distanza dell'oggetto, è possibile calcolarne la distanza da esso. L'energia di ritorno è anche funzione del coefficiente di riflessione dell'oggetto. Ostacoli con alto coefficiente di riflessione hanno la capacità di riflettere grandi quantità di energia emessa dalla sorgente infrarossa, viceversa oggetti con un basso coefficiente la riflettono in maniera inferiore. La differenza nel coefficiente di riflessione tra gli oggetti può causare errori sostanziali se non tenuto in conto.

Ogni sensore è composto di due emettitori infrarosso Siemens SFH 34-3GaAs e di un fotodiodo al silicio Siemens SFH 20030F[14]. I due emettitori e il ricevitore sono montati orizzontalmente, sulla stessa retta, con il fotodiodo ricevitore tra i due emettitori. Per rilevare l'energia riflessa, una lettura infrarosso viene effettuata con gli emettitori accesi, e un'altra con gli emettitori spenti, in tale modo

[14]Inserito in un involucro plastico (*delrin housing package*).

la differenza tra le due letture è proporzionale alla energia riflessa da un oggetto ma indipendente dall'energia infrarosso presente nell'ambiente di lavoro.

I fattori che influenzano le misurazioni ottenute dal Sensus 350^{TM} sono:

fattori geometrici : il principio di funzionamento del sensore di prossimità ad infrarosso è che più l'oggetto è lontano, meno quantità di emissione riflessa riceverà il fotodiodo. L'angolo di riflessione rispetto alla superficie dell'oggetto riveste un ruolo importante. Quando la superficie dell'oggetto riceve la radiazione infrarossa, parte dell'energia in arrivo viene riflessa e parte viene diffratta. Se la superficie illuminata è normale, o quasi, all'asse del ricevitore, la maggior parte della luce riflessa verrà rediretta all'indietro, contrariamente, man mano che la superficie diviene parallela all'asse dell'emettitore soltanto l'energia diffratta raggiungerà il ricevitore. La distanza e l'angolazione sono i due più importanti fattori che influenzano le misurazioni ma anche la geometria dell'oggetto ha un'importanza non trascurabile. Se l'oggetto è piccolo, la lettura sarà dipendente dall'area di oggetto illuminata, che è funzione della sua grandezza. Inoltre, se l'oggetto non ha superfici piane, la radiazione riflessa verso i sensori sarà frutto di una complessa combinazione di energia riflessa e diffratta proveniente dalle parti illuminate dell'oggetto. La figura 2.8 mostra i livelli di energia misurati da $0cm$ a $90cm$, per angoli da 0^0 a 75^0, le misure sono state effettuate al buio su un foglio di comune carta per fotocopiatrice.

fattori di illuminazione : il sensore calcola la differenza tra la luce ricevuta con l'emettitore spento e con quello acceso. Se la luce presente nell'ambiente ha sufficiente energia nello spettro dell'infrarosso, il sensore può saturare facilmente e la differenza tra le due misurazioni può essere molto piccola (poco contrasto). Il risultato è una perdita sostanziale di sensibilità. I dati di figura 2.9 sono stati rilevati in un ambiente illuminato con luce fluorescente, i dati di figura 2.10 invece con luce incandescente; comparando questa con la figura 2.8 è visibile quanto i valori energetici siano poco

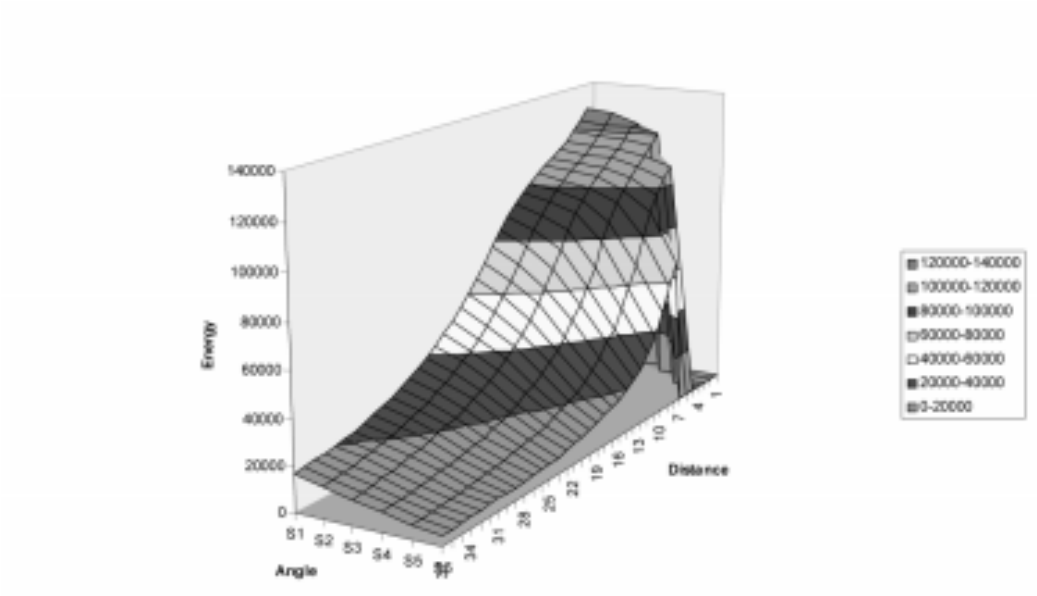


Figura 2.8: Grafico energia / angolazione / distanza su bersaglio di superficie opaca al buio.

influenzati dalla luce fluorescente, poichè contiene una piccola porzione di spettro infrarosso.

colore e caratteristiche superficiali : il colore dell'oggetto, o più precisamente il coefficiente di riflessione nello spettro infrarosso, influenza la precisione della misurazione, così come le caratteristiche superficiali dell'oggetto: superfici lucide evidenziano un picco energetico per angoli vicini alla normale dell'asse del ricevitore[15]. Materiali come cemento o tessuto assorbono molta più energia radiante. Il grafico di figura 2.11 mostra l'energia misurata a 0 gradi (superficie dell'oggetto normale all'asse del sensore), per una superficie nera lucida, per un foglio nero opaco, per una tavola bianca lucida, e per un comune foglio di carta bianca. Alcuni materiali riflettono in modo differente lo spettro della luce visibile e infrarossa. Oggetti che assorbono luce visibile[16] non necessariamente assorbono luce infrarossa e

[15] Cioè hanno un comportamento speculare.

[16] Chiamati anche *black objects*.

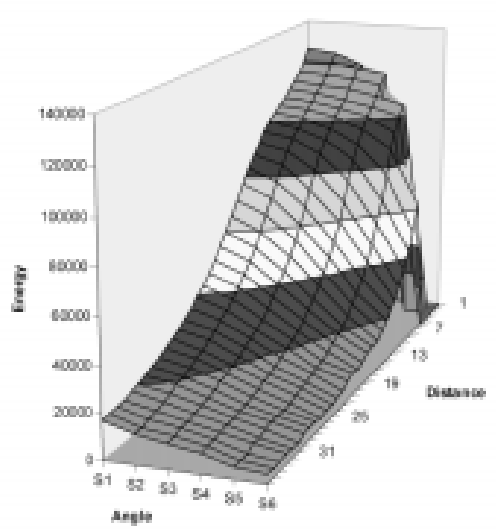


Figura 2.9: Grafico energia / angolazione / distanza su bersaglio di superficie opaca in luce fluorescente.

viceversa.

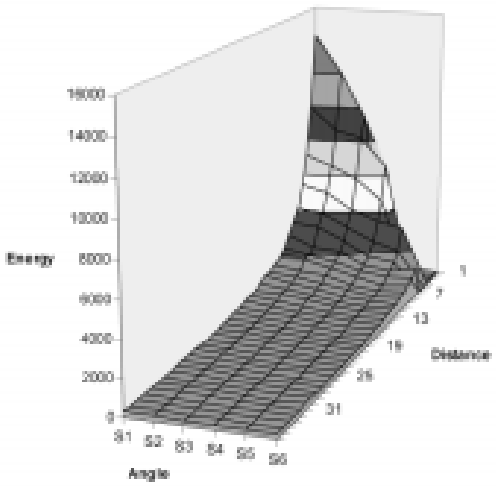


Figura 2.10: Grafico energia / angolazione / distanza su bersaglio di superficie opaca in luce incandescente.

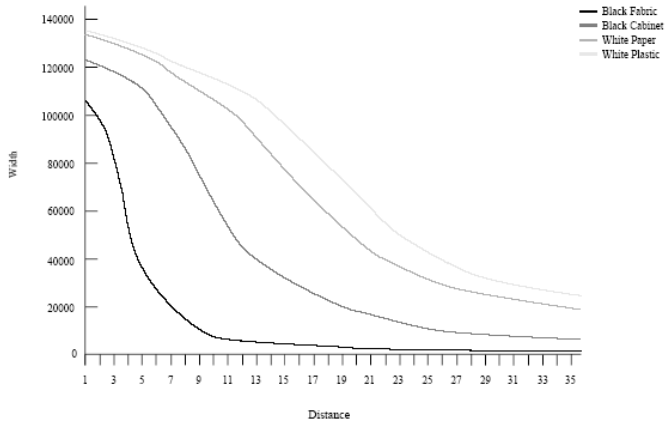


Figura 2.11: Grafico energia / distanza ad angolazione 0: materiale bianco / nero, materiale opaco / lucido.

2.5.2 XR SynapseNet distributed I/O network

La rete XR SynapseNet è una rete distribuita di controller embedded[17] collegati tra loro tramite una rete *ARCNET™ real-time token-passing*. Il network comunica tramite delle richieste basate su *remote-procedure call* (RPC) tra i vari controller e il *Sistema alto livello Intel Pentium III*. Questo permette la creazione di una rete espansibile, ad alta velocità e bassa latenza, composta di controller embedded. Il controller embedded Intellisys 160, è stato appositamente disegnato dalla Nomadic Technologies per l'XR4000, ed è basato sul processore Motorola 68HC11F1 con 128 KB di memoria FLASH aggiornabile tramite la rete stessa; grazie a questo tipo di tecnologia è possibile aggiornare il firmware in ogni controller senza bisogno di sostituire ROM.

[17] Il termine *sistema embedded* identifica genericamente dei sistemi elettronici a microprocessore progettati appositamente per una determinata applicazione, spesso con una piattaforma hardware ad hoc. In questa area si collocano sistemi di svariate tipologie e dimensioni, in relazione al tipo di microprocessore, al sistema operativo, ed alla complessità del software che può variare da poche centinaia di bytes a parecchi megabytes di codice.

Controller embedded Intellisys 160/200

Sono presenti all'interno del nomad XR4000 5 schede Intellisys 160 montate su ognuna delle tre porte, sulla PDB[18] e sulla PPC[19].

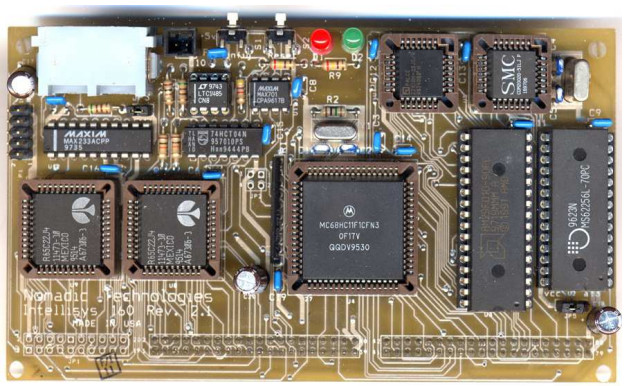


Figura 2.12: La scheda Intellisys 160.

Sulla scheda Intellisys 160 sono presenti due spie di attività: una rossa che indica che la scheda è correttamente alimentata, e una verde che indica la comunicazione verso un altro controller embedded. Tutti i controller Intellisys presenti nel robot sono collegati tra loro tramite comuni cavi RJ11[20].

Il controller embedded montato sulla porta riceve dati dai sensori montati su essa (sonar, infrarosso, sensori tattili).

Un altro controller embedded è montato sulla PDB del robot che alimenta tutte le periferiche, trasformando e stabilizzando le tensioni provenienti dalle batterie. L'Intellisys 160 verifica, e regola, le corrette tensioni dei vari canali di alimentazione. Ha inoltre il compito di controllare la visualizzazione dei dati sul display frontale e di gestire la pulsanteria del pannello di controllo. Il numero *ARCNET* di questo device è 0x21h[21].

[18] Acronimo di Power Distribution Board.
[19] Acronimo di Primary Power Controller.
[20] Cavo di collegamento telefonico standard.
[21] ANET_PDB in anet.h.

Un Intellisys 160 è montato, e fa parte, della scheda PPC. Posizionato nella parte inferiore del robot, subito sopra all'alloggiamento delle batterie, esso ha la funzione di controllo del circuito di carica delle batterie e di monitoraggio del corrente stato di carica.

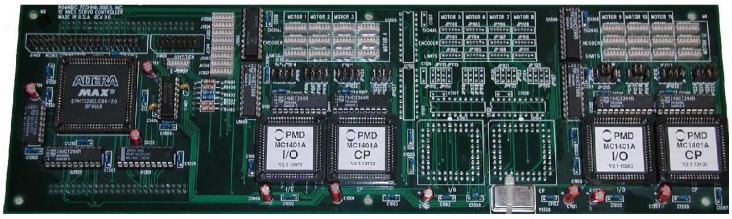


Figura 2.13: La scheda Intellisys 200.

La scheda *Intellisys 200 12 axis servo controller* controlla la base olonoma ad otto motori dell'XR4000, ed è connessa ad un pettine ISA della scheda madre. Sulla Intellisys 200 è presente un connettore seriale femmina 9 poli DSUB, dove è possibile ricevere e inviare comandi direttamente al controller del motore. Questo controller utilizza due integrati PMD MC1401[22]. Ogni PMD MC1401 controlla quattro assi ed è composto da due circuiti integrati: *Peripheral Input/Output IC* e *Command Processor IC.*

2.5.3 Sistema sensoriale avanzato

Sistema Sensus 600 di orientamento

Il sistema Sensus 600 è basato su una bussola digitale KVH-C100 che è un componente industriale di grande affidabilità, compattezza e precisione. È in grado di misurare i decimi di gradi ed ha una accuratezza di 0.5^0 anche a latitudini vicine ai poli[23]; anche grazie alle sue generose temperature di esercizio, da -25^0C

[22]Il PMD MC1401 è un sofisticato DSP programmabile che si occupa del *motion control* per motori in corrente continua.

[23]Si rimanda per completezza *Magnetic Dip Angle Effects on Accuracy* del manuale *Overview of Compass technology* del KVH-C100.

a 70^0C, è stato impiegato in applicazioni critiche (come robot sottomarini) e in ambito militare.

Il KVH-C100 si basa su una bussola a flussometro digitale capace di misure molto precise anche se montata non in orizzontale, o in ambienti con forti disturbi elettromagnetici. La bussola è montata esternamente in un contenitore metallico blindato e collegata alla scheda di interfacciamento *Nomadic compass interface* da cui, tramite una piattina, è connessa alla seconda porta seriale[24] della scheda madre. Il sistema è configurato per comunicare a 9600 baud, 8 bit di dati, nessun bit di parità e 1 bit di stop.

Sistema laser Sick Sensus 550

Il Sensus 550 è un sistema di telemetria basato sul sensore Sick elettro-ottico LMS-200. Esso fornisce 180^0 di rilevamento planare ad incrementi di 0.5^0, complessivamente 360 campioni. Ogni piano completo di rilevazione avviene con frequenza $20Hz$.

Sistema di visione

Il sistema di visione del Nomad XR4000 è composto da una scheda di acquisizione PCI PnP[25] basata sul chip Brooktree 878, che permette acquisizioni *real-time* fino a $30fps$ con una massima risoluzione di $640x480$, e da una telecamera CCD Hitachi KP-D50, dotata di ottica Pentax regolabile (riportata in figura 2.14). Grazie alla tecnologia PCI è possibile avere latenze molto limitate nell'acquisizione video, ottenendo risultati molto performanti. La telecamera è avvitata su un apposita staffa fissata sulla superficie superiore del robot.

[24] COM2 [dos/win] o /dev/ttyS1 [*nix].
[25] Plug-and-play.

Figura 2.14: Telecamera CCD Hitachi KP-D50.

2.6 Sistema di alimentazione

2.6.1 PPC: Primary Power Controller

Il cuore del sistema di alimentazione del Nomad XR4000 è il PPC, Primary Power Controller. Il PPC può essere raggiunto solamente rimuovendo una delle batterie e accedendo alla base del robot. Il circuito stampato sopra il connettore della batteria è proprio la scheda PPC, che ha la funzione di:

- Controllare l'alimentazione principale e lo stato dei pulsanti di emergenza
- Trasformare e regolare la corrente alternata della rete elettrica
- Controllare il caricamento della batteria
- Misurare le tensioni delle batterie
- Misurare le correnti assorbite e fornite dalle batterie
- Misurare nove importati tensioni per fini diagnostici

2.6.2 PDB: Power Distribution Board

La scheda PDB converte le tensioni ricevute dalla PPC ($45V/60V$) a tensioni ridotte e stabilizzate per l'alimentazione del computer di bordo, dei sensori e delle periferiche. La PDB è posta sul retro dell'alloggiamento della scheda madre, e può facilmente essere raggiunta aprendo una porta laterale del Nomad XR4000. La PDB, inoltre, fornisce corrente ai diversi connettori di alimentazione situati su essa, utilizzabili per il collegamento di periferiche e di accessori aggiuntivi.

2.6.3 Connettori sulla PDB

Sono presenti diversi tipi di connettori di alimentazione sulla PDB ognuno progettato per un diverso carico sostenibile:

- Connettori di alimentazione modulari, figura 2.16.
- Connettori per alimentazione telecamere, figura 2.17.
- Connettori per alimentazione dischi fissi, figura 2.18.
- Connettori per ventole di raffreddamento, figura 2.19.
- Connettori di alimentazione per scheda madre, figura 2.20
- Connettore di alimentazione principale, figura 2.21

Pin	Segnale
1	+12V
2	+5V
3	GND_CLEAN (riferito all'alimentazione del robot)

Tabella 2.2: *Pinout* del connettore di alimentazione modulare.

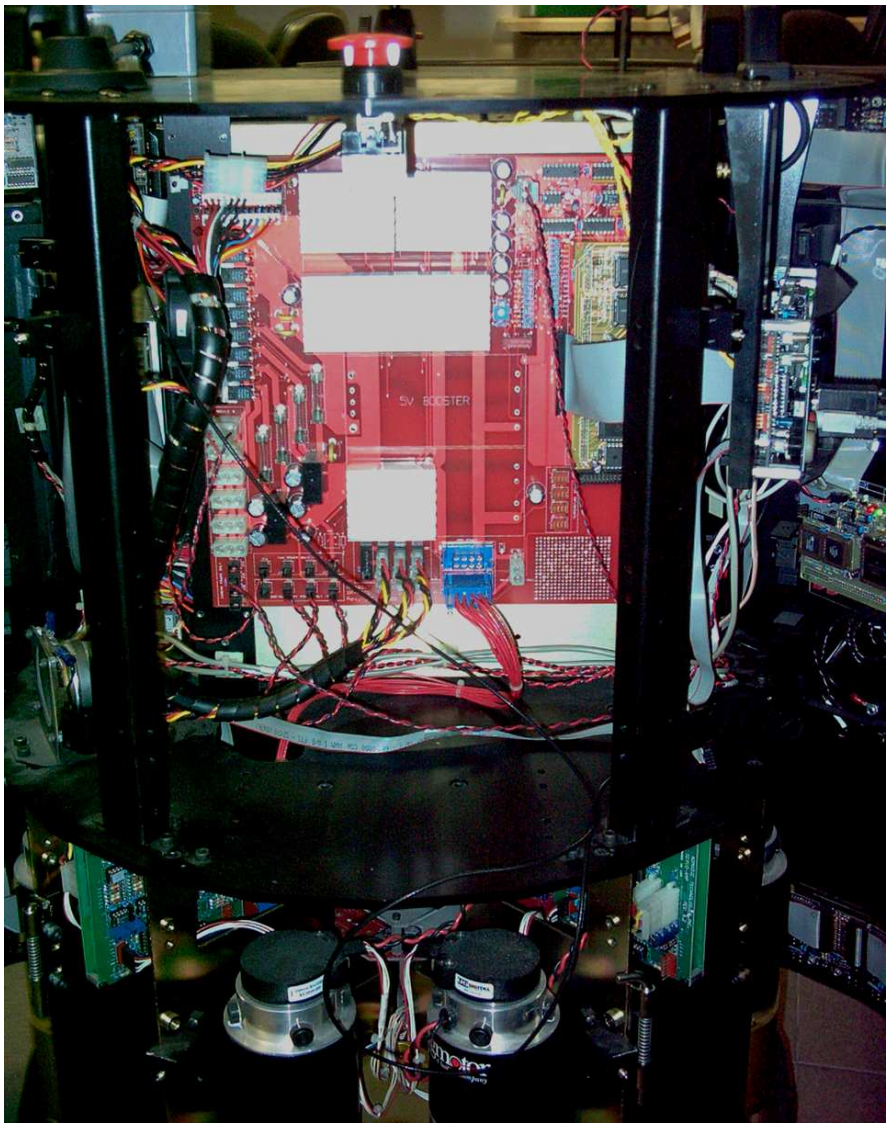


Figura 2.15: La scheda di alimentazione PDB.

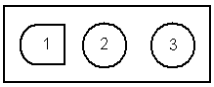


Figura 2.16: Connettore di alimentazione modulare.

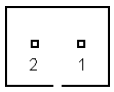


Figura 2.17: Connettore di alimentazione per telecamere.

Pin	Segnale
1	+12V
2	GND_CLEAN (riferito all'alimentazione del robot)

Tabella 2.3: *Pinout* del connettore di alimentazione per telecamera.

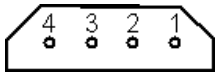


Figura 2.18: Connettore di alimentazione per dischi fissi.

Pin	Segnale
1	+12V
2	GND_CLEAN (riferito all'alimentazione del robot)
3	GND_CLEAN (riferito all'alimentazione del robot)
1	+5V

Tabella 2.4: *Pinout* del connettore per dischi fissi.

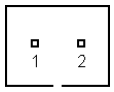


Figura 2.19: Connettore di alimentazione per ventole di raffreddamento.

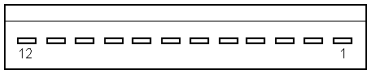


Figura 2.20: Connettore di alimentazione per scheda madre.

Pin	Segnale
1	+12V
2	GND_CLEAN (riferito all'alimentazione del robot)

Tabella 2.5: *Pinout* del connettore per ventole di raffreddamento.

Pin	Segnale
1	NC
2	+5V
3	+12V
4	-12V
5	GND_CLEAN (riferito all'alimentazione del robot)
6	GND_CLEAN (riferito all'alimentazione del robot)
7	GND_CLEAN (riferito all'alimentazione del robot)
8	GND_CLEAN (riferito all'alimentazione del robot)
9	+5V
10	+5V

Tabella 2.6: *Pinout* del connettore di alimentazione per scheda madre.

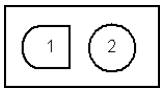


Figura 2.21: Connettore di alimentazione principale.

Pin	Segnale
1	+45V/+60V non stabilizzata
2	GND_CLEAN (riferito all'alimentazione del robot)

Tabella 2.7: *Pinout* del connettore di alimentazione principale.

2.7 Pannello di controllo

Il pannello di controllo del Nomad XR4000 è situato sulla porta laterale destra del robot, ed è composto da una pulsantiera, da dei connettori I/O, da un display

LCD di stato e da una serie di spie di indicazione.

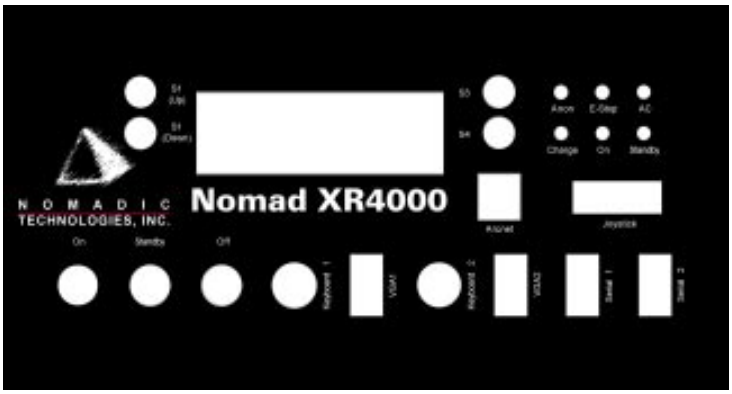


Figura 2.22: Pannello di controllo.

2.7.1 Pulsantiera

La pulsantiera è costituita da tre pulsanti

- On
- Standby
- Off[26]

si rimanda alla sezione 2.2 per le loro funzionalità.

2.7.2 Connettori I/O

Sistemati orizzontalmente sono stati alloggiati i seguenti connettori I/O:

Video : per l'uscita video VGA da collegare a un monitor

Mouse Entrata PS/2 per il collegamento di un mouse

[26] Di colore rosso.

Keyboard1 : entrata PS/2 per il collegamento di una tastiera

Serial : connessione DSub-9 poli per il collegamento ad una porta seriale del computer interno[27]

Joystick : connessione per modalità di controllo con *joystick.*

2.7.3 Display LCD di stato

Il display di LCD di stato risulta non funzionante, a causa di un danneggiamento elettrico del modulo di controllo. Il display è un Noritake Itron 4 righe x 20 colonne[28], tuttavia la scheda Intellisys 160 che lo controlla sembra avere un funzionamento corretto. Il display, con i suoi relativi pulsanti di scorrimento, era utilizzato per mostrate stringhe esplicative dello stato del robot, come la tensione ai capi delle batterie, oppure stringhe di informazione mandate direttamente da chiamate RPC da parte dell'utente.

2.7.4 Spie di indicazione

Sono presenti sei diverse spie di indicazione

Anon : non usato

Estop : indicatore E-Stop[29]

AC : indica se il robot è alimentato dalla rete elettrica

Charge : indica il caricamento delle batterie

On : è acceso solo se il robot è nello stato di *Acceso*

[27] Il connettore è scollegato.

[28] Modello CU20045SCPB-W5J.

[29] Emergency stop: pulsanti rossi di emergenza situati sulla parte superiore del robot, interrompono il moto del robot in maniera istantanea.

Standby : si illumina se è in modalità *Standby* o *Acceso*; questa spia non è accesa solo se il robot è completamente spento

2.8 Manutenzione e risoluzione dei problemi

2.8.1 Manutenzione batterie

Il sistema primario di alimentazione mobile del robot Nomad XR4000 è costituito da quattro batterie PowerSonic PS-12330 ($12V\ 22Ah$), per un erogazione totale di $1056Wh$.

Caricamento e durata

Le batterie sono poste in quattro vani laterali ricavati nella base del robot. È sufficiente aprire i portelli laterali, sollevare il fermo ed estrarle facendole scivolare.

Il design originale del circuito di carica del Nomad XR4000 effettua un caricamento serie delle batterie, in questo modo esse vengono spesso caricate in maniera irregolare[30], deteriorandole molto rapidamente. Per una maggiore longevità le batterie dovrebbero essere caricate in parallelo, è quindi sconsigliato utilizzare il robot come piattaforma di carica, ed è suggerito di far uso di caricatori esterni paralleli.

Se si utilizza il robot in maniera stazionaria è consigliato, prima di accenderlo[31], di scollegare le batterie estraendole dal loro vano, e collegare il robot alla rete elettrica.

Con le batterie completamente cariche, il robot ha un'autonomia di circa sei ore utilizzando le più comuni strumentazioni di bordo. Se la tensione di una delle quattro batterie scende sotto i $10.8V$, il robot emette un suono acuto continuo; è

[30] Con un caricabatteria serie sono frequenti i fenomeni di over/undercharging.

[31] Attenzione: non estrarre mai le batterie a robot acceso.

quindi necessario spegnerlo, e rimuovere le batterie per evitarne uno scaricamento eccessivo che provoca un danneggiamento permanente alla loro struttura chimica.

Se il Nomad non è in uso è consigliabile scollegare le batterie dal robot.

Quando tutte queste indicazioni sono correttamente eseguite, le batterie possono resistere a circa 200 cicli di carica e scarica. Quando le batterie iniziano a scaricarsi molto rapidamente, o non riescono ad accumulare sufficiente carica, è tempo di sostituirle.

2.8.2 Manutenzione dei portelli laterali

Rimozione dei portelli

Aprire il portello, scollegare il cavo di alimentazione rosso/nero e i cavi ARCNET dal controller embedded montato sulla porta. Scollegare il collegamento della massa sotto la cerniera del portello. Afferrare quindi la porta vicino alla cerniera e sollevare finchè non viene rimossa dal suo alloggiamento.

Montare i portelli

Tenere con una mano la parte superiore della porta, vicino alla cerniera e con l'altra tenera la cerniera. Allineare la cerniera con la staffa della base del robot e quindi abbassare la cerniera in modo che si inserisca nei due perni. Ricollegare il cavo rosso/nero di alimentazione, i cavi ARCNET[32] e il collegamento massa.

2.8.3 Fusibili

Due sole schede di controllo montano fusibili: la PPC, posta sulla base subito sopra le batterie, e la PDB, fissata dietro l'alloggiamento della scheda madre. Un altro fusibile è collegato alla presa di connessione per il collegamento alla rete elettrica, di $10A$ a $250V$.

[32] I cavi RJ11 ARCNET dei controller embedded non hanno ordine di collegamento, nei controller collegati con due cavi inserirli senza curarsi dell'ordine.

Fusibili PPC

La scheda PPC ha quattro fusibili. La posizione è schematizzata nella figura 2.23, che mostra una vista dall'alto. I fusibili sono alloggiati sulla superficie inferiore della scheda, e sono accessibili rimuovendo le batterie 3 e 4. I fusibili F3 ed F4 sono fissati in dei particolari alloggiamenti robusti, e possono essere rimossi tirando con decisione verso il basso. La disposizione delle batterie, sotto la PPC è mostrata in figura 2.24.

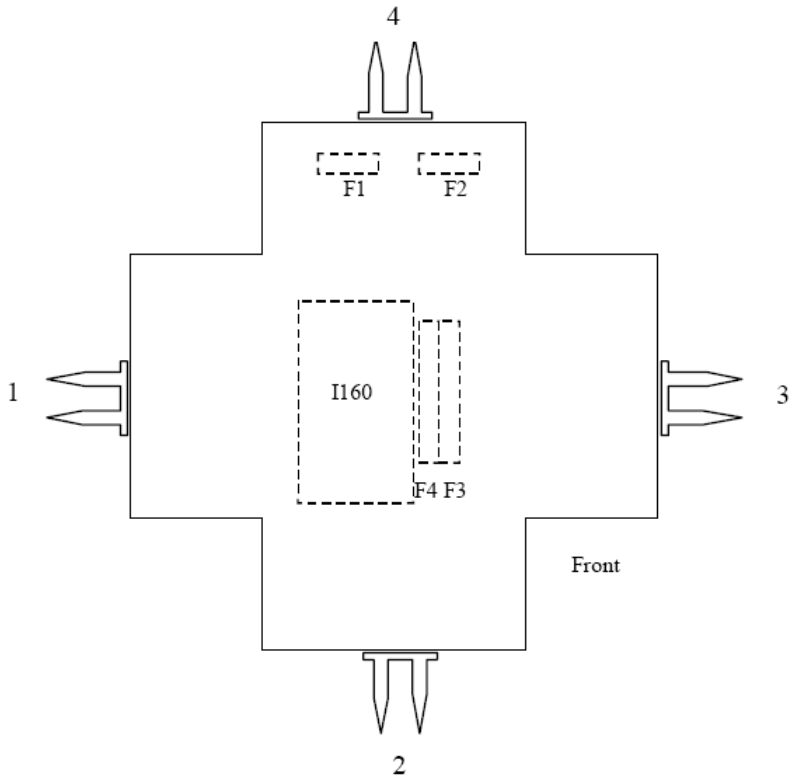


Figura 2.23: Posizionamento dei fusibili sulla PPC. La linea tratteggiata indica che l'oggetto non è visibile dall'alto

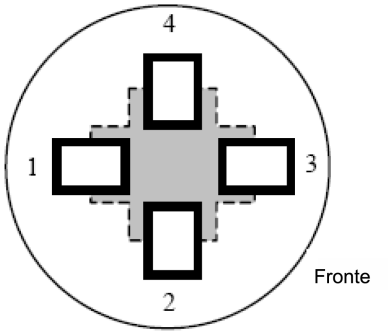


Figura 2.24: Posizionamento della batteria sotto la PPC.

Nome	Specifiche	Funzione
F1	3AG - 5A	fusibile Standby
F2	3AG - 15A	fusibile CPU
F3	CCMR LittleFuse-20A	fusibile Motore
F4	CCMR LittleFuse-30A	fusibile principale

Tabella 2.8: Specifiche fusibili PPC.

Fusibili PDB

Questa scheda, mostrata nella figura 2.25, ha quattro fusibili ed è comodamente accessibile aprendo il portello di sinistra del robot.

Nome	Specifiche	Funzione
F1	3AG - 5A	Modulo B 12V
F2	3AG - 5A	Modulo B 5V
F3	3AG - 5A	Modulo A 12V
F4	3AG - 5A	Modulo B 5V

Tabella 2.9: Specifiche fusibili PDB.

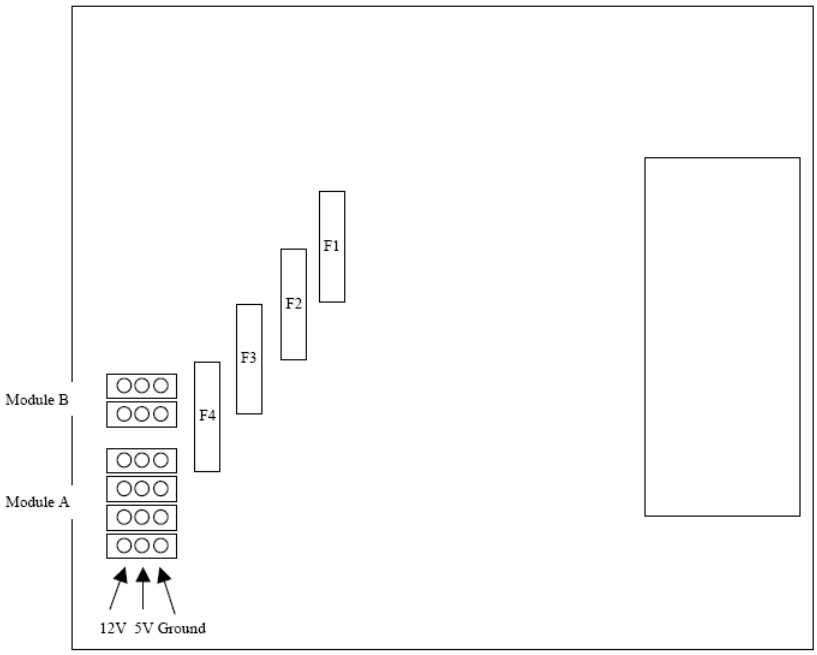


Figura 2.25: Posizionamento dei fusibili sulla PDB.

Capitolo 3

Il software dell'XR4000

Il Nomad XR4000 è un sistema robotico costituito da un apparato sensoriale complesso e da un computer basato su microprocessore Intel Pentium III integrato nel robot. Per rendere operativo l'XR4000, evitando di danneggiare l'hardware, è necessario che sia installato un sistema operativo Linux adeguato e che i driver delle periferiche connesse al robot e integrate siano correttamente installate; si devono inoltre modificare i file di configurazione dei driver in modo opportuno, affinché tutta la strumentazione sia appropriatamente configurata.

Come già discusso precedentemente, si è deciso, per motivi di prestazione e affidabilità, di utilizzare un sistema operativo più recente rispetto a quello fornito con l'XR4000, e la scelta è caduta sulla versione 7.3 della distribuzione Red Hat di Linux con kernel 2.4.18-3, sul quale è stato possibile installare versioni più aggiornate e quindi più stabili dei driver. Differenti prove sperimentali hanno dimostrato la reale affidabilità di questa configurazione, garantendo all'utilizzatore del Nomad un ambiente di lavoro e di sviluppo migliori rispetto alla configurazione originale.

Originariamente sul sistema Nomad XR4000 era installato un sistema operativo Red Hat 5.3 con kernel 2.0.35, sul quale erano montati i driver delle periferiche integrate e non; il passaggio ad una versione più aggiornata di Linux Red Hat e conseguentemente ad una più aggiornata del kernel è stato ritenuto

necessario per rendere più performante il sistema in vista delle applicazioni future di robotica nonché, come già detto, per renderlo più stabile; inoltre tutto il software compatibile con il kernel 2.4.x può essere installato sul Nomad.

L'installazione di un sistema operativo più aggiornato per il quale non sono presenti driver né supporto tecnico è un'operazione piuttosto complessa che richiede una profonda conoscenza dell'hardware, della programmazione in C nonché dell'ambiente di lavoro Linux; nonostante queste difficoltà si è riusciti a rendere operativo il robot con versioni più aggiornate dei driver, ottenendo ottimi risultati, ed è quindi fortemente consigliato seguire l'installazione fornita su questa guida piuttosto che ricorrere a versioni più datate dei manuali.

3.1 Installazione del sistema operativo

L'installazione di Linux Red Hat 7.3 può essere effettuata tramite CD-ROM[1], oppure tramite rete[2] (via FTP, HTTP). Durante l'installazione verrà richiesto di formattare l'hard disk e la scelta migliore è quella di creare due partizioni, una di swap da 127 MB ed una di root / in modo tale che occupi lo spazio restante. In seguito occorre operare una scelta accurata dei pacchetti di Linux da installare nel computer; è opportuno scegliere soltanto i pacchetti strettamente necessari, evitando così di appesantire inutilmente il sistema. Infine, dopo la copia dei file, occorre definire un *boot-loader* per avviare il sistema operativo[3] ed eventualmente di creare un disco di ripristino. Inoltre è opportuno inserire una password per l'amministratore, e si consiglia di default di inserire `N0mad1C`. Se non ci sono stati errori, il sistema operativo Linux è correttamente installato.

[1] Controllare che l'opzione di avvio da CD-ROM sia attivata nel BIOS.

[2] Avviare il robot dal disco di installazione *Boot-Net* e configurare la scheda di rete con i parametri opportuni.

[3] Si consiglia l'uso di *LILO*.

3.2 Installazione dei driver

Dopo aver effettuato l'installazione del sistema operativo e dopo averne appurato il corretto funzionamento, occorre eseguire il comando `make dep` nella directory del kernel corrente, poi occorre procedere nell'installazione dei driver (moduli) per la strumentazione *built-in* e periferica del robot che è di seguito elencata:

- scheda di rete Proxim RangeLan 2 - modulo *rlmod*;
- scheda di rete EtherExpress Pro 100 (*built-in*) - modulo *eepro100*[4];
- scheda di sintesi vocale DoubleTalk - modulo *dtlk*[5];
- scheda di acquisizione video BT878 - modulo *bttv*;
- scheda di controllo del motore (Intellisys 200 12 axis servo-controller) - modulo *xrm*;
- scheda di interfacciamento con la rete SynapseNet distributed network[6] - demone *DRobot*.

3.2.1 Installazione dispositivi di rete

In questa sezione verrà descritto il metodo per configurare correttamente i seguenti dispositivi di rete presenti nel Nomad XR4000:

1. Proxim RangeLan2 per la rete senza fili;
2. EtherExpress Pro 100 per la rete convenzionale.

[4]Il modulo *eepro100* è automaticamente riconosciuto ed installato dalla Red Hat versione 7.3.

[5]Il modulo *dtlk* è automaticamente riconosciuto ed installato dalla Red Hat versione 7.3.

[6]Per la descrizione hardware della scheda di interfacciamento con la rete SynapseNet, si rimanda alla sezione 2.5.2 del capitolo 2.

È necessario configurare il file `/etc/sysconfig/network` per settare i parametri di rete in modo tale da assicurarsi che siano presenti le seguenti impostazioni:

```
NETWORKING=yes
HOSTNAME=localhost
GATEWAY=10.10.10.1
```

le quali si riferiscono rispettivamente alla capacità di utilizzare la rete (`NETWORKING=yes`), al nome del computer (`HOSTNAME=localhost`) ed al gateway (`GATEWAY=10.10.10.1`[7]).

Installazione della scheda di rete Proxim RangeLan 2

Il driver per la scheda di rete wireless Proxim RangeLan2 è stato modificato opportunamente dagli autori per funzionare sulla versione di Linux Red Hat 7.3 ed è stato creato il pacchetto di installazione `rl2_driver_custom.tar.gz`[8]; per installare il pacchetto occorre eseguire come `root` il comando `tar -zxf rl2_driver_custom.tar.gz`, che creerà una directory denominata `rl2_driver_custom` e dalla directory corrente deve essere utilizzato il comando `make config`, rispondendo con esattezza alle domande e quindi eseguire `make modules modules_install` per installare il driver nel sistema. Inoltre occorre editare il file `/etc/sysconfig/network-scripts/ifcfg-eth0` nel modo seguente:

```
DEVICE=eth0
ONBOOT=yes
IPADDR=10.10.10.2
NETMASK=255.255.255.0
GATEWAY=10.10.10.1
```

[7] L'indirizzo specificato si riferisce all'IP del server della rete *N.I.N.E.* che verrà più ampiamente presentato nella sezione A.1 del capitolo A.

[8] Si ringrazia D.Komacke che ha scritto il modulo per Linux 2.2.x della RangeLan2 (`http://www.komacke.com/distribution.html`) e T.Schneider per i consigli sul porting sul kernel Linux 2.4.x.

le quali si riferiscono al tipo di dispositivo (`DEVICE=eth0`), all'attivazione dell'interfaccia di rete all'avvio (`ONBOOT=yes`), ed infine all'indirizzo IP del robot, alla subnet mask ed al gateway (per il dispositivo eth0). Inoltre è necessario modificare il file di configurazione `/etc/modules.conf`, in modo tale che sotto la riga:

```
alias eth0 rlmod
```

appaiano le seguenti righe di configurazione:

```
options rlmod CardType=0 io=0x270 irq=11
post-install rlmod proxcfg eth0 sta peer-to-peer yes
```

nella riga `options` appare il nome del modulo e come deve essere configurato: `CardType` indica il modello di RangeLan2 installata, `io` indica l'indirizzo di input/output ed `irq` indica l'interrupt request. L'opzione `post-install` viene utilizzata per configurare le impostazioni radio della RangeLan2: `proxcfg eth0` è il comando che applica delle modifiche alla configurazione radio del dispositivo eth0 ed i restanti sono parametri. In particolare `sta` indica che la scheda wireless è configurata come station[9], `peer-to-peer yes`, invece, indica che la configurazione tra i vari dispositivi wireless è di tipo *peer-to-peer.*

Installazione della scheda di rete EtherExpress Pro 100

L'installazione della EtherExpress Pro 100 avviene congiuntamente all'installazione del sistema operativo, è quindi opportuno verificare con il comando `lsmod`, che permette di visionare i moduli caricati in memoria (con l'opzione `-aux`), che il modulo *eepro100* sia presente[10]. È inoltre fondamentale editare il file `/etc/sysconfig/network-scripts/ifcfg-eth1`. Nel file `ifcfg-eth1`, in cui sono presenti i settaggi per il dispositivo eth1 cioé la EtherExpress Pro 100, occorre assicurarsi che siano presenti le seguenti impostazioni:

[9] Le possibili configurazioni sono: `master` e `alternate master`.

[10] Se il modulo *eepro100* non è presente occorre scaricare da internet il driver appropriato ed installarlo manualmente.

```
DEVICE=eth1
ONBOOT=yes
IPADDR=XXX.XXX.XXX.XXX
NETMASK=YYY.YYY.YYY.YYY
GATEWAY=ZZZ.ZZZ.ZZZ.ZZZ
```

le quali si riferiscono al tipo di dispositivo (`DEVICE=eth1`), all'attivazione dell'interfaccia di rete all'avvio (`ONBOOT=yes`), ed infine all'indirizzo IP del robot, alla subnet mask ed al gateway (per la configurazione del dispositivo eth1).

3.2.2 Installazione della scheda di sintesi vocale DoubleTalk

La scheda DoubleTalk è automaticamente installata con Red Hat Linux 7.3, non è quindi indicato utilizzare i driver forniti dalla Nomadic Technologies, che sono meno recenti. Per garantire la corretta inizializzazione all'avvio di Linux è comunque indicato verificare che nel file `/etc/rc.d/init.d/drobot` sia presente la seguente riga di configurazione:

```
modprobe -a dtlk
```

Il comando precedente garantisce il caricamento automatico del modulo `dtlk` all'avvio del sistema operativo. Come prova del corretto funzionamento il dispositivo di sintesi vocale produrrà la frase *DoubleTalk found* all'avvio e alla rimozione del modulo la frase *goodbye.*

3.2.3 Installazione della scheda di acquisizione video BT878

La scheda BT878[11] è automaticamente riconosciuta dal sistema operativo al momento dell'installazione, è comunque consigliato installare una versione più

[11] Chipset Brooktree 878.

recente dei driver, in quanto con la versione contenuta nella distribuzione Red Hat 7.3 sono riportati diversi bachi, che sono stati corretti in versioni successive[12].

Dopo aver decompresso l'archivio `bttv-0.9.15.tar.gz`, per installare la scheda di acquisizione è sufficiente eseguire il comando `make` nella directory appena creata. Il modulo `bttv` viene creato e per caricarlo automaticamente all'avvio occorre controllare che nel file `/etc/rc.d/init.d/drobot` sia presente la seguente riga di configurazione:

```
modprobe bttv
```

All'avvio del sistema operativo la corretta inizializzazione del driver è garantita dall'assenza di messaggi di errore in fase di caricamento.

Per testare, inoltre, la funzionalità della scheda di acquisizione e della telecamera è sufficiente installare l'applicazione `xawtv` ed eseguire il comando `xawtv` su una shell di KDE, assicurandosi di selezionare le impostazioni esatte per i dispositivi di visione; in particolare è da notare che la telecamera in dotazione[13] accetta come formato l'NTSC, e che è connessa alla scheda di acquisizione tramite cavo coassiale su *S-composite.* A schermo è possibile visualizzare l'immagine acquisita dalla telecamera in tempo reale, ed è possibile quindi configurare le impostazioni di fuoco e apertura dell'otturatore.

3.2.4 Installazione della scheda Intellisys 200 12 axis servo-controller

All'installazione dei driver della scheda di controllo del motore su un sistema operativo Linux differente dalla versione Red Hat 5.3 (kernel 2.0.35) consigliata dalla Nomadic Technologies, gli autori hanno preferito installare una versione più recente dei driver per il motore corretti da D. Austin[14]. L'installazione di questi

[12] La versione installata dei driver della BT878 è la 0.9.15.

[13] Hitachi color camera - KP-D50U.

[14] http://drobot.sourceforge.net/xr4000.html.

driver è fortemente consigliata in quanto, dopo varie prove sperimentali eseguite sull'XR4000, hanno fornito buoni risultati, migliori rispetto a quelli ottenuti con i driver originali forniti con il robot; da questo punto in poi ogni riferimento è relativo ai driver per il motore aggiornati.

Dopo aver scaricato e decompresso il pacchetto `DRobot-2.0.tgz`, per compilare ed installare i driver è sufficiente eseguire il comando `./compile_install`[15] nella directory appena creata. In questo modo i driver verranno compilati con le librerie aggiornate; se non ci sono stati errori di compilazione il driver del motore è stato correttamente installato.

In fase di avvio del sistema operativo se non ci sono stati errori, a schermo verrà visualizzato il messaggio di caricamento del modulo del motore; dopo aver inizializzato la scheda di controllo del motore, il sistema provvederà ad eseguire il comando `basezero` che porta il robot in posizione di home[16]. Durante il comando `basezero` non è possibile interagire con il sistema operativo, ma la corretta esecuzione dello *zeroing* è dettata dal movimento delle quattro ruote in successione.

Se il modulo del motore non dovesse essere caricato correttamente dal sistema operativo all'avvio, è presente un file readme con le possibili cause dell'errore. In ogni caso è indicato controllare che nel file `/etc/rc.d/init.d/drobot` sia presente la seguente riga di configurazione:

```
modprobe xrm
```

Successivamente è possibile verificare la corretta inizializzazione del motore dell'XR4000, eseguendo una serie di programmi di testing ed esempi, contenuti nella cartella `/usr/local/xrdev/bin` e nella cartella `/usr/local/xrdev/examples`.

[15] L'esecuzione del comando `make` comporta oltre all'installazione dei driver per il motore anche l'installazione dei driver per la scheda di interfacciamento con la rete SynapseNet e il demone *DRobot*.

[16] Durante la fase di *zeroing*, tutto l'output a schermo viene rediretto su una nuova console (tty8), la quale è visualizzabile tramite il comando `ALT + F8`.

3.2.5 Installazione della scheda di interfacciamento con la rete SynapseNet distributed network

Anche nell'installazione della SynapseNet, gli autori hanno preferito una versione più aggiornata dei driver e compatibile con la distribuzione Red Hat 7.3; di nuovo si sono utilizzati i driver forniti da D. Austin ai quali sono state apportate opportune modifiche per motivi di compatibilità con la versione installata di Linux.

L'installazione dei driver per la SynapseNet avviene contestualmente all'installazione dei driver del motore, come già riportato nella *nota 15* della *sezione 3.2.4.* Il software comunica con la scheda di interfacciamento grazie ad un demone (*DRobot*), che ha il compito di ascoltare i comandi e di comandare l'hardware (comunicando con i sensori, con il motore, e con altro hardware presente nel robot): il demone viene caricato automaticamente all'avvio del robot, e si occupa di effettuare, tra gli altri processi, lo *zeroing* del robot, riportandolo in posizione di home ed inizializzando i sensori e l'hardware. Nella *sezione 3.3* verrà ampiamente illustrata l'architettura *XRDev*.

3.3 L'architettura *XRDev*

XRDev[17] è una architettura multi processo destinata alle applicazioni robotiche composta dalle seguenti librerie ed eseguibili:

DRobot - È il demone caricato all'avvio del robot, addetto all'ascolto dei comandi impartiti dall'utente, i quali vengono poi diretti all'hardware presente sul Nomad. È possibile estendere questo programma, in modo che l'utente possa aggiungere altre caratteristiche, senza avere il codice sorgente.

Nhost_client.a - È la libreria di funzioni utilizzate da un programma client per comandare un robot da qualsiasi computer sulla rete. Per utilizzare questa caratteristica è sufficiente che l'utente, nella compilazione del proprio

[17] Acronimo per Xtreme Robotics Development Environment.

programma, lo linki a questa libreria, ed esegua il programma normalmente da qualsiasi macchina sulla rete; la libreria si prenderà cura della comunicazione attraverso la rete, per raggiungere e impartire comandi al robot.

Ngui - È l'interfaccia grafica[18] che fornisce una rappresentazione grafica dell'attività del robot e dei dati dei sensori, nonché permette all'utente di controllare il robot (tramite un joystick a schermo oppure con semplici comandi).

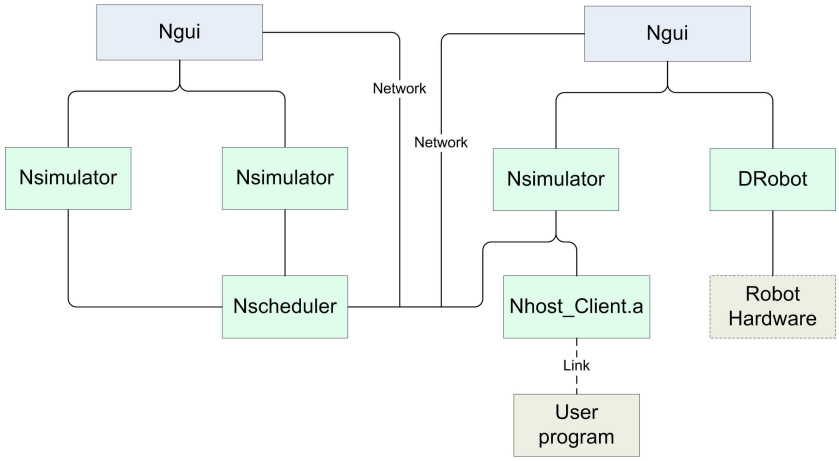


Figura 3.1: Schema dell'architettura *XRDev*

Una applicazione *XRDev* è composta da differenti processi: i processi del robot, dell'utente, i processi di interfaccia, etc., che comunicano attraverso la rete. Non c'è un limite (se non quello imposto dall'efficienza e dal carico sulla rete e sulla CPU) al numero di robot che possono essere controllati con *XRDev* e non c'è un limite al numero di programmi che li controllano. Inoltre ogni programma, che può controllare più robot contemporaneamente, può essere eseguito ovunque sulla rete.

[18] Per utilizzare questa applicazione è necessario avviare Linux in modalità grafica, utilizzando il comando `startx` sulla shell; all'interno dell'interfaccia grafica, è sufficiente aprire una shell e digitare `Ngui`.

Vi sono inoltre diversi tipi di processi e configurazioni che permettono di controllare differentemente il robot:

1. Un demone `DRobot` viene utilizzato per controllare un robot reale. Possono esserci molti demoni `DRobot` in una singola applicazione, ma un solo `DRobot` in esecuzione su un robot reale.

2. Un processo `Ngui` imposta un'interfaccia grafica. Da questo tipo di interfaccia l'utente può mandare comandi al robot e ricevere dati da qualsiasi processo del robot. Possono esserci molti processi `Ngui` connessi ad un numero qualsiasi di processi del robot.

3. Un processo dell'utente integra l'applicazione dell'utente che controlla i processi del robot direttamente.

Sono possibili differenti configurazioni per utilizzare varie combinazioni di questi processi.

La configurazione più semplice è quella con il demone `DRobot` in esecuzione sul robot ed `Ngui` in esecuzione su un'altra macchina, comunicando via radio Ethernet. L'utente in questa configurazione controlla direttamente il robot via interfaccia grafica.

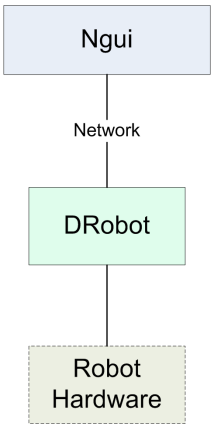


Figura 3.2: Una semplice configurazione - Una *GUI* ed un robot

Una configurazione più elaborata è composta da un processo `DRobot` e da una applicazione linkata alla libreria `Nhost_Client.a`:

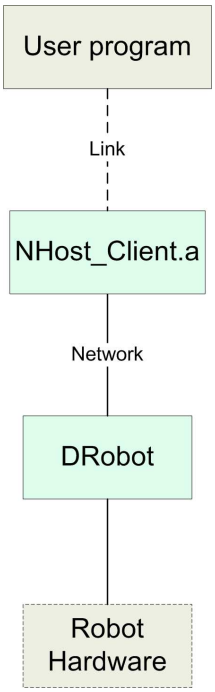


Figura 3.3: Una semplice configurazione - Un robot ed un programma utente in esecuzione sulla rete

Con il demone `DRobot` in esecuzione sul robot, l'applicazione dell'utente, linkata alla libreria `Nhost_Client.a`, può essere compilata ed eseguita su qualsiasi computer con una connessione (radio o cablata) al robot.

3.3.1 Il demone `DRobot`

`DRobot` è il server che comunica con l'hardware del robot (sensori, motore, etc.) mandando comandi e ricevendo i dati dai sensori. Ogni programma interagisce con il robot grazie a questo demone.

Il demone `DRobot` è avviato automaticamente al momento dello start-up; durante l'installazione del driver del motore (si veda a proposito la *sezione 3.2.4*),

viene generato il file `drobot` nella directory `/etc/rc.d/init.d`. Il file `drobot` è uno script della shell che provvede all'inizializzazione ed allo spegnimento[19] dell'hardware del robot e all'avvio o alla disattivazione del demone `DRobot`.

Opzioni di configurazione di *XRDev*

I programmi client, per poter stabilire una connessione dati con il robot, devono superare due fasi:

1. Stabilire una connessione TCP/IP con uno *scheduler*[20]
2. Il programma client richiede le informazioni di connessione al socket per l'ID del robot al quale vuole connettersi e lo *scheduler* glielo fornisce.

Alla partenza, il demone `DRobot` tenta di creare un *listener socket* TCP/IP, al quale i programmi client possono connettersi. Inoltre il demone `DRobot` si connette ad uno *scheduler* e lo informa sull'ID del robot, sull'hostname e sulla porta del *listener socket*. Da questo punto in poi, qualsiasi programma client che si connetta allo *scheduler* viene informato della presenza di questo robot.

Nella configurazione di default, il demone `DRobot` agisce anche da *scheduler*; in questo modo, i programmi client si connettono direttamente a `DRobot` e sono informati della presenza del robot, che risiede nella connessione già stabilita. Nel caso di più robot è possibile cambiare questa configurazione di base, utilizzando un robot come *scheduler* per tutti i robot. In questo modo un singolo programma client riesce a controllare due robot simultaneamente. È possibile editare il file di configurazione per `DRobot`; i seguenti parametri sono modificabili:

[19] Alla versione originale di `DRobot`, denominata `NRobot`, è stata preferita la più recente in quanto sulla versione originale, tra gli altri problemi, si è riscontrato un grave baco sulla procedura di spegnimento dell'hardware del robot, che causava uno shutdown non corretto, detto *dirty shutdown*. Nella nuova versione il problema è stato corretto ed il demone `DRobot` risponde alla pressione del tasto Off sulla console.

[20] Lo *scheduler* è un processo che mappa gli ID dei robot presenti sulla rete con le informazioni di connessione al socket.

Opzioni	Descrizione
`-help`	Visualizza un help sintetico
`-robotid n`	L'ID al quale registrarsi
`-port n`	La porta TCP alla quale `DRobot` ascolta le connessioni in ingresso
`-schedulerhost name`	L'hostname dello *scheduler*
`-schedulerport n`	La porta TCP alla quale lo *scheduler* ascolta
`-f filename`	Il nome di un file di configurazione alternativo
`-d`	Non forza il programma a funzionare in background

Tabella 3.1: Opzioni via linea di comando.

Robot ID : è il numero con il quale i programmi client si riferiscono al robot. È possibile un numero intero maggiore di zero. Il default è 1.

Listener port : è il numero della porta TCP alla quale il demone `DRobot` ascolta le connessioni dati. Il default è 7073. È possibile inserire qualsiasi numero da 1024 a 65535.

Scheduler location : è l'hostname ed il numero della porta TCP/IP che descrive la collocazione sulla rete dove `DRobot` dovrebbe registrare il proprio ID. Per default `DRobot` funge anche da *scheduler.*

È opportuno modificare il file di configurazione solo per cambiamenti permanenti alla configurazione di `DRobot`; a scopo di verifica è consigliato utilizzare le opzioni nella riga di comando riportate in tabella 3.1:

Modificando il file di setup `drobot.cfg` è possibile rendere permanente la configurazione di `DRobot`.

Un esempio di una sezione del file di configurazione è il seguente, in cui si vuole assegnare al robot l'ID 3 e come numero della porta TCP alla quale il demone `DRobot` ascolta le connessioni dati (`listener_port`) il valore 4500:

```
[robot]
robot_id = 3
listener_port = 4500
```

Per le altre sezioni, la sintassi è simile a quella riportata nel precedente esempio, e si consiglia di visionare il file `/usr/local/xrdev/etc/drobot.cfg` per ulteriori parole chiave e variabili d'ambiente.

Va tenuto presente, inoltre che i comandi impartiti a `DRobot` via linea di comando, hanno la priorità rispetto a quelli letti dal file di configurazione.

Quando il demone `DRobot` viene eseguito, questo legge nel file di configurazione `/usr/local/xrdev/etc/drobot.cfg` la sequenza di accensione dei sonar; se nel file sono presenti queste linee di configurazione, il demone comanda ogni singola porta del robot, inizializzando opportunamente i sonar. Come esempio si riporta il caso in cui si desidera configurare la corona inferiore dei sonar della porta di sinistra in modo tale da accendere alternativamente il primo e l'ottavo sonar; le seguenti righe di configurazione permettono di ottenere la configurazione precedente:

```
[sonset4]
firing_order = 0 7 255
```

dove il `firing_order` indica l'ordine con il quale devono essere accesi i sonar (il numero 255 è il valore speciale che indica il terminatore dell'ordine di accensione dei sonar).

Se sono presenti differenti dispositivi hardware, come il laser SICK oppure il meccanismo Nomad XRlift, il demone `DRobot` deve essere informato della presenza di questa strumentazione, per inizializzarla correttamente. Nel file `drobot.cfg` è presente la sezione `[lift]`, relativa al Nomad XRlift; un esempio della configurazione è il seguente:

```
[lift] type = xrlift
config_path = /usr/local/xrdev/etc/xrlift.cfg
```

Opzioni	Descrizione
`CreateS550`	Questa è una *keyword* e non deve essere cambiata
`name`	Nome arbitrario da assegnare al sensore
`pointcount`	Numero massimo di punti supportati da questo sensore
`x, y`	Posizione lungo X ed Y del centro della scansione del sensore relativa alla base
`cosine, sine`	Seno ed il coseno della parte anteriore dei sensori relativa alla base
`reference_object`	Nome dell'oggetto relativamente al quale è posizionato il sensore; utilizzare `base` come default
`port`	Porta seriale alla quale è collegato il sensore

Tabella 3.2: Lista di opzioni per la sezione `s550`.

Per quanto riguarda invece il laser SICK, la sezione `s550` è quella interessata all'inizializzazione. Un lista di *keyword* possibili per il laser SICK è riportata in *tabella 3.2.*

In generale una *keyword* ha la seguente struttura:

```
keyword = CreateS550 name pointcount x y cosine sine
          reference port
```

In particolare, per configurare un singolo laser SICK montato all'interno del robot e collegato sulla prima porta seriale è sufficiente utilizzare la seguente configurazione, da aggiungere nel file `/usr/local/xrdev/etc/drobot.cfg`:

```
[s550]
s550s = s550a
s550a = CreateS550 s550a 361 0 150 0 1 base /dev/ttyS0
```

3.3.2 Risoluzione dei problemi relativi a *XRDev*

È possibile eseguire, su uno stesso robot, una sola istanza del demone `DRobot` alla volta. Per prevenire l'esecuzione simultanea di più istanze di `DRobot`, all'avvio del demone questo crea un file *lock* nella directory `/tmp/.nrobot/`, il quale viene cancellato quando il demone viene chiuso. Se durante l'inizializzazione `DRobot` trovasse il *lockfile,* verrebbe visualizzato un messaggio d'errore; nel caso in cui fosse presente un'altra istanza di `DRobot`, la presenza del *lockfile* preverrebbe la possibilità di caricarne un'altra. Se invece fosse ancora presente il *lockfile* all'interno di `/tmp/.nrobot/`, ma il demone non fosse in esecuzione, è possibile che `DRobot` visualizzi un messaggio di errore e quindi non si avvii. Per risolvere questo problema, dopo aver verificato con il comando `ps -ef` che non è presente il demone `DRobot` nella lista, è sufficiente cancellare con il comando `rm -rf /tmp/.nrobot/*.lock` il *lockfile.* Generalmente il *lockfile* non viene cancellato automaticamente quando, per esempio il demone non viene chiuso correttamente, oppure quando il robot viene spento senza eseguire i comandi `shutdown -h now` oppure `halt`.

Capitolo 4

Programmare l'XR4000

In questo capitolo verranno illustrate le funzioni che permettono di programmare il Nomad XR4000. La struttura dati fondamentale, attorno alla quale ruota l'intera programmazione del robot, è `N_RobotState`, nella quale sono contenuti i dati e le configurazioni dei sensori; un puntatore a questa struttura è restituito dalla funzione `N_GetRobotState`. L'utente, per settare i parametri desiderati, dovrà modificare il contenuto di questa struttura. Ad esempio, per muovere il robot, i contenuti del campo *AxisSet* all'interno della struttura `N_RobotState` saranno modificati in modo tale da riflettere il movimento desiderato; per rendere poi effettivi i comandi, verrà chiamata la funzione `N_SetAxes()`. In modo analogo, per recuperare le informazioni degli assi del robot, dopo aver chiamato la funzione `N_GetAxes()`, occorre leggere i campi appropriati in `N_RobotState`. In generale, i nomi delle funzioni con le quali si leggono dati ed informazioni hanno il prefisso *get,* viceversa per impostare le configurazioni dello stesso sottosistema si usa il prefisso *set.*

4.1 I comandi del Nomad XR4000

In questa sezione vengono descritte le varie funzioni per il movimento e per il controllo dei sensori sul robot e come possono essere utilizzate.

4.1.1 Stabilire una comunicazione con il robot

Il programma *client* come primo passo deve connettersi ad uno *scheduler.* Lo *scheduler,* tra le altre cose, agisce come un timer globale per la simulazione, quando si utilizza un robot simulato ed è necessario ogniqualvolta ci siano più di due robot (reali o simulati) nel programma *client.* Quando ci si deve connettere ad un singolo robot reale, lo *scheduler* non è necessario poiché il robot reale può agire esso stesso come *scheduler.* Per connettersi allo *scheduler* è sufficiente chiamare la funzione `N_InitializeClient()`, che è dichiarata come segue:

```
int N_InitializeClient (const char *scheduler_hostname,
                        unsigned short scheduler_socket);
```

Dove `scheduler_hostname` è il nome sulla rete della macchina sulla quale è in esecuzione *Nscheduler* e `scheduler_socket` è il *socket* TCP/IP al quale il processo *scheduler* sta ascoltando. Nel caso particolare in cui è presente un solo robot (reale o simulato), `N_InitializeClient()` dovrebbe essere chiamata con l'*hostname* ed il *socket* del robot. I *socket* TCP/IP di `Nscheduler` e di `DRobot` vengono visualizzati al loro avvio.

Dopo aver stabilito la comunicazione con lo *scheduler,* la comunicazione con il robot deve essere effettuata; questo è possibile con la funzione `N_ConnectRobot`, che è dichiarata come segue:

```
int N_ConnectRobot (long RobotID);
```

dove `RobotID` è il numero identificativo del robot. Allo stesso modo, se l'utente desidera disconnettersi dal robot[1], basta chiamare la funzione `N_Disconnect Robot()`, che è dichiarata come segue:

```
N_DisconnectRobot (long RobotID);
```

[1]Ogni chiamata alla funzione `N_RobotConnect()` deve essere sempre seguita, alla fine del programma *client,* da una chiamata alla funzione `N_DisconnectRobot`.

4.1.2 Il timer del robot

Il timer del robot è una caratteristica intrinseca che impedisce al robot di continuare il movimento quando il programma *client* non è più in esecuzione. Questo è ottenuto utilizzando un timer interno che viene resettato ogni volta che un programma *client* esegue una chiamata alla libreria *client* con un comando *get* o *set.*

Quando il timer supera una soglia predefinita, i motori vengono automaticamente spenti, come se venisse premuto l'*emergency stop.* La soglia del timer è contenuta nella struttura `N_RobotState` nel campo `Timer`, che è definito così:

```
struct N_Timer
{
  long Timeout;
  unsigned long Time;
};
```

Dove `Timeout` è la soglia del timer, dopo la quale i motori vengono spenti[2]. `Time` rappresenta l'ammontare di tempo da quando il robot è stato acceso (il timer globale). Entrambi i parametri sono espressi in millisecondi.

Per leggere e scrivere il parametro `Timeout`, l'utente deve richiamare rispettivamente le funzioni `N_GetTimer` e `N_SetTimer`, che sono dichiarate così:

```
int N_GetTimer (long RobotID);
int N_SetTimer (long RobotID);
```

4.1.3 *Timestamps*

Dal momento che spesso alla lettura dei dati dai sensori è associata una latenza, ad ogni lettura dei sensori sono associati dei *timestamps,* i quali forniscono l'istante al quale è stata effettuata una misura. Il *timestamp* si trova nel campo

[2]Per motivi di sicurezza il parametro `Timeout` non può superare il valore 1500 (1,5 secondi).

TimeStamp nella struttura `N_RobotState`. La latenza dei sensori è regolata dalla frequenza di aggiornamento dei sensori e dal tempo necessario a mandare i dati dei sensori dal robot all'utente.

Per esempio, quando il robot si sta muovendo rapidamente ed ottiene le informazioni sia da sensori con un basso tasso di aggiornamento (come i sonar) che da sensori con un alto tasso di aggiornamento, è possibile che l'informazione è ottenuta dai due tipi di sensori in tempi inconsistenti. Questo è causa di grandi errori, se nell'applicazione è necessario fondere le informazioni da diversi tipi di sensori. I *timestamps* dei due sensori informano l'utente su quando è stata effettuata la misurazione in riferimento ad un timer globale[3], ed in questo modo è possibile riconciliare i dati dai due sensori.

4.1.4 La struttura `N_RobotState`

`N_RobotState` è una struttura nella quale sono contenuti i dati e le informazioni necessarie quando si stabilisce una comunicazione tra un robot (reale o simulato) ed un programma *client*. È definita come segue:

```
struct N_RobotState
{
  N_CONST long RobotID;
  N_CONST char RobotType;
  struct N_Integrator Integrator;
  struct N_AxisSet AxisSet;
  struct N_LiftController LiftController;
  struct N_Joystick Joystick;
  struct N_SonarController SonarController;
  struct N_InfraredController InfraredController;
  struct N_BumperController BumperController;
  struct N_Compass Compass;
```

[3] Il campo `Time` in `N_Timer`.

```
  struct N_LaserSet LaserSet;
  struct N_S550Set S550Set;
  struct N_BatterySet BatterySet;
  struct N_Timer Timer;
};
```

Per accedere ad `N_RobotState`, un programma *client* deve chiamare `N_Get RobotState()`, che è dichiarata così:

```
struct N_RobotState *N_GetRobotState (long RobotID);
```

La quale restituisce un puntatore ad `N_RobotState` per il robot specificato in `RobotID`. In generale `N_GetRobotState` viene chiamata una sola volta all'inizio del programma *client* per ogni robot al quale il programma desidera connettersi. Il puntatore che restituisce è quello che si utilizza nell'intero programma per scambiare informazioni con il robot. Quando il programma *client* vuole recuperare tutte le informazioni dal robot, occorre chiamare la funzione `N_GetState()`, che è dichiarata come segue:

```
int N_GetState (long RobotID);
```

Una chiamata alla funzione `N_GetState` equivale ad eseguire tutti i comandi *get* eccetto che `N_GetTimer()` che deve essere chiamata esplicitamente per aggiornare i campi della struttura `N_Timer`.

4.1.5 Movimentazione del robot

Il sistema olonomo Nomad XR4000

Un corpo rigido vincolato al piano ha fino a tre gradi di libertà. Nello spazio Cartesiano questi sono rappresentati da $\mathcal{X}$, $\mathcal{Y}$ e la rotazione. Le stesse regole si applicano ad un robot che si muove su un piano; così la posizione del Nomad XR4000 è rappresentata dai tre parametri appena definiti. Inoltre l'XR4000 può accelerare in ogni direzione in qualsiasi momento ed è quindi un sistema olonomo.

Modalità di movimento

Ci sono due possibili modalità di controllo del movimento del Nomad.

Joint mode Questa modalità tratta gli assi dell'XR4000 come giunti. In questo modo, il giunto ha una posizione, una direzione positiva, una negativa che sono definite rispetto al corpo al quale è collegato. Nella modalità *joint,* il Nomad XR4000 ha tre giunti collegati al centro del robot. Il giunto $\mathcal{Y}$ crea un movimento lineare lungo la direzione in avanti ed indietro rispetto al robot. Il giunto $\mathcal{X}$ crea un movimento lineare lungo la direzione di sinistra e di destra rispetto al robot. Il giunto di rotazione crea un movimento rotazionale rispetto al centro del robot.

Global mode Con questa modalità è possibile controllare gli assi del Nomad XR4000 rispetto ad un sistema di riferimento fisso (globale). Si può pensare al sistema di riferimento globale come ad un sistema che viene creato appena il robot viene avviato[4], e rimane fisso per tutto il tempo in cui il Nomad rimane acceso. Nella modalità *global,* il movimento lungo l'asse $\mathcal{Y}$, causa sempre un movimento lungo la direzione $\mathcal{Y}$ in riferimento all'angolo con cui è orientato il robot.

Queste due modalità di controllo sono specificate nella struttura `N_AxisSet` di `N_RobotState`. Se il campo `Global` nella struttura `N_AxisSet` è settato a `TRUE`, gli assi sono impostati nella modalità *Global*; se è settato a `FALSE`, gli assi sono impostati nella modalità *Joint* [5].

[4] Oppure analogamente quando viene richiamata la funzione `N_SetIntegratedConfiguration`.

[5] Si noti che il cambiamento della modalità di controllo degli assi è di tipo globale, ossia interessa tutti gli assi, così da non poter confondere le due modalità di controllo.

Modalità degli assi

Ogni asse del robot può essere controllato in una modalità separata[6]. Questa modalità è specificata nel campo `Mode` della struttura `N_Axis`, per ogni asse. La struttura `N_Axis` fa parte della struttura `N_AxisSet` di `N_RobotState`.

Velocity Mode Nella modalità *Velocity,* l'utente può selezionare la velocità da assegnare ad un asse; la velocità viene mantenuta a quel valore finché non ne viene inserito un altro. L'utente può anche selezionare il parametro accelerazione, che rappresenta quanto velocemente può cambiare la velocità, se essa deve diminuire o aumentare. Questa modalità è specificata impostando il campo `Mode` a `N_AXIS_VELOCITY`.

Position Mode Nella modalità *Position* l'utente seleziona una posizione di destinazione per un asse, e l'asse si muove fino a quella posizione e si ferma. L'utente specifica inoltre anche una velocità desiderata ed una accelerazione. La velocità desiderata è la velocità con cui si muove l'asse per raggiungere la posizione desiderata, mentre l'accelerazione rappresenta quanto velocemente può cambiare la velocità, se essa deve diminuire o aumentare. Le due modalità possibili per la posizione sono *absolute* o *relative* e vengono specificate impostando rispettivamente il campo `Mode` a `N_AXIS_POSITION_ABSOLUTE` oppure a `N_AXIS_POSITION_RELATIVE`. La modalità di posizione *absolute* consente il movimento degli assi alla posizione assoluta rispetto alla posizione zero degli assi. La modalità di posizione *relative* consente di muovere gli assi alla posizione relativa alla posizione corrente.

4.1.6 Le strutture `N_Axis` e `N_AxisSet`

All'interno della struttura `N_RobotState` è contenuta la struttura `N_AxisSet` che contiene tutte le informazioni riguardanti gli assi di movimentazione della

[6] Da non confondere con le modalità di controllo *Global* e *Joint.*

base del Nomad:

```
struct N_AxisSet
{
  BOOL Global;
  unsigned char Status;
  N_CONST unsigned int AxisCount;
  struct N_Axis Axis[N_MAX_AXIS_COUNT];
};
```

★ `Global`: quando è impostata a `TRUE`, la base del robot entra in modalità *Global,* mentre se è impostata a `FALSE`, la base del robot entra in modalità *Joint.*

★ `Status`: può assumere uno dei seguenti valori:

`N_AXES_READY`: indica che gli assi sono disponibili per il movimento.

`N_JOYSTICK_IN_USE`: indica che la base è al momento controllata via joystick.

`N_ESTOP_DOWN`: indica che uno o più pulsanti di emergenza sono premuti.

`N_MOTION_ERROR`: indica che la base del robot non può eseguire il movimento.

La struttura `N_Axis` contiene informazioni per ogni singolo asse:

```
struct N_Axis
{
  BOOL DataActive;
  BOOL TimeStampActive;
  BOOL Update;
  unsigned long TimeStamp;
  char Mode;
  long DesideredPosition;
  long DesiredSpeed;
```

```
  long Acceleration;
  long TrajectoryPosition;
  long TrajectoryVelocity;
  long ActualPosition;
  long ActualVelocity;
  BOOL InProgress;
};
```

★ `DataActive`: il valore `TRUE` per questo parametro implica che i valori nella struttura saranno aggiornati.

★ `TimeStampActive`: il valore `TRUE` per questo parametro implica che sarà aggiornato il valore di `TimeStamp`.

★ `Update`: il valore `TRUE` per questo parametro implica che i valori `DesiredSpeed`, `DesiredPosition` ed `Acceleration` saranno caricati, quando verrà eseguita la funzione `N_SetAxes`.

★ `TimeStamp`: il valore del tempo (in millisecondi) al quale sono stati misurati i valori.

★ `Mode`: può essere una dei seguenti:
`N_AXIS_POSITION_RELATIVE`: specifica che l'asse si muove relativamente alla posizione corrente.
`N_AXIS_POSITION_ABSOLUTE`: specifica che l'asse si muove verso una posizione rispetto alla posizione del riferimento assoluto.
`N_AXIS_VELOCITY`: specifica che l'asse si muove con una velocità costante.
`N_AXIS_STOP`: ferma l'asse.

★ `DesiredPosition`: specifica la posizione finale desiderata dell'asse. Le unità sono in millimetri per le traslazioni ed in milliradianti per le rotazioni. Quando il campo `Mode` è settato a `N_AXIS_POSITION_RELATIVE` oppure a `N_AXIS_POSITION_ABSOLUTE`, questo specifica rispettivamente la posizione finale relativa alla posizione corrente o a quella assoluta. Questo

parametro non viene utilizzato quando Mode è settato a N_AXIS_VELOCITY oppure a N_AXIS_STOP. Se è selezionata la modalità *Global* (*Global* =TRUE), il parametro DesiredPosition è definito rispetto al sistema di riferimento globale. Se invece è selezionata la modalità *Joint* (*Global* =FALSE), il parametro DesiredPosition è definito rispetto alle coordinate di giunto.

★ DesiredSpeed: specifica la velocità alla quale deve essere effettuato il movimento. Questo parametro non è utilizzato quando il campo Mode è settato a N_AXIS_STOP.

★ Acceleration: specifica l'accelerazione relativa al parametro DesiredSpeed oppure le accelerazioni successive se il valore di DesiredSpeed aumenta durante un movimento. Questo parametro specifica anche la decelerazione, con le stesse modalità dell'accelerazione. Le unità sono in millimetri/secondo, per le traslazioni ed in milliradianti/secondo per le rotazioni.

★ TrajectoryPosition: fornisce la posizione corrente del generatore di traiettoria. Se *Global* =TRUE (condizione di *Global mode*), il valore TrajectoryPosition è definito rispetto al sistema di riferimento fisso. Se *Global* =FALSE (condizione di *Joint mode*), il valore TrajectoryPosition è definito rispetto al sistema di coordinate di giunto.

★ TrajectoryVelocity: fornisce la velocità corrente del generatore di traiettoria. Se *Global* =TRUE (condizione di *Global mode*), il valore TrajectoryVelocity è definito rispetto al sistema di riferimento fisso. Se *Global* =FALSE (condizione di *Joint mode*), il valore TrajectoryVelocity è definito rispetto al sistema di coordinate di giunto.

★ ActualPosition: fornisce la posizione attuale dell'asse. Il valore di questo campo è basato sul settaggio del campo *Global*. Se *Global* =TRUE (condizione di *Global mode*), questo campo fornisce la posizione dell'asse rispetto al sistema di riferimento fisso (questo stesso valore può essere

trovato nella struttura `N_Integrator`). Se *Global* =`FALSE` (condizione di *Joint mode*), questo campo fornisce la posizione dell'asse rispetto al sistema di coordinate di giunto.

★ `ActualVelocity`: fornisce la velocità attuale dell'asse. Il valore di questo campo è basato sul settaggio del campo *Global.* Se *Global* =`TRUE` (condizione di *Global mode*), questo campo fornisce la posizione dell'asse rispetto al sistema di riferimento fisso. Se *Global* =`FALSE` (condizione di *Joint mode*), questo campo fornisce la posizione dell'asse rispetto al sistema di coordinate di giunto. Questo valore, come `ActualPosition` è basato sull'impostazione del campo *Global* in `N_AxisSet`. Se il campo *Global* è selezionato, questo campo fornisce un valore rispetto al sistema di riferimento fisso.

★ `InProgress`: fornisce un valore booleano che informa l'utente che un asse è in movimento.

Le informazioni nella struttura `N_AxisSet` sono aggiornate con la chiamata alla funzione `N_GetAxes`, che è dichiarata come segue:

```
int N_GetAxes (long RobotID);
```

I valori modificati tramite `N_AxisSet` sono caricati negli assi per il movimento, chiamando `N_SetAxes`, che è dichiarata come segue:

```
int N_SetAxes (long RobotID);
```

4.1.7 La *Configurazione Integrata*

Ogni robot della famiglia Nomad stima continuamente (durante il movimento) la propria posizione in coordinate Cartesiane ($\mathcal{X}$, $\mathcal{Y}$ e la rotazione) rispetto alle coordinate del sistema di riferimento fisso. Questa è chiamata in generale *dead-reckoning position*[7], ed è un ulteriore *sensore* che indica dove si trova il robot

[7] *Dead-reckoning* significa vagare ad occhi chiusi.

nell'ambiente (sensori odometrici). Tuttavia l'utilità di questo *sensore* è limitata, poiché è affetto da molti errori (slittamenti delle ruote, etc.). Questo è stimato misurando i cambiamenti di posizione ($\triangle x$, $\triangle y$ e $\triangle \theta$), su piccolissimi incrementi di tempo (circa 5 ms), e vengono integrati nel tempo (per questo viene chiamata *Configurazione Integrata*).

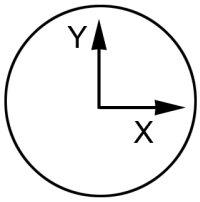


Figura 4.1: Posizione degli assi nel nomad XR4000

Per ottenere la *Configurazione Integrata* occorre chiamare la funzione `N_GetIntegratedConfiguration()` e recuperare i valori nel campo `N_Integrator` di `N_RobotState`. Allo stesso modo, la *Configurazione Integrata* può essere configurata modificando i contenuti di `N_Integrator` e chiamando la funzione `N_SetIntegratorConfiguration()`.

La struttura `N_Integrator` è definita come segue:

```
struct N_Integrator
{
  BOOL DataActive;
  BOOL TimeStampActive;
  long x;
  long y;
  long Rotation;
  unsigned long TimeStamp;
};
```

4.1.8 Sensori tattili

I sensori di contatto forniscono informazioni sullo stato di contatto fisico con gli oggetti presenti nell'ambiente. Il rilevamento degli ostacoli dovrebbe essere effettuato con i sensori di prossimità (sonar o infrarosso), ma questo non è garantito. I sensori di contatto, o sensori di collisione, sono 48 sensori tattili distribuiti sul perimetro superiore ed inferiore del robot. Sono presenti, inoltre, quattro switch flottanti dietro ogni portello di accesso del robot, i quali restituiscono rilevamento tra i due perimetri. I sensori tattili sono capaci di rilevare solo due livelli di contatto. Il Nomad XR4000 ha tre porte, numerate in senso antiorario, mostrate in figura 4.2, in cui sono installati due *set* di sensori (sonar, infrarosso, bumper) per porta, in totale sei *set.*

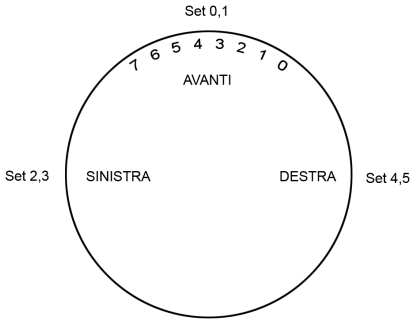


Figura 4.2: Disposizione dei sensori sul XR4000.

La rilevazione acquisita dal sensore di contatto è contenuta in `N_BumperController`, facente parte della struttura `N_RobotState`.

```
struct N_BumperController
{
   N_CONST unsigned int BumperSetCount;
   struct N_BumperSet BumperSet(N_MAX_BUMPER_SET_COUNT);
};
```

★ `BumperSetCount`: quantità di *bumper* collegati al controller

★ `N_BumperSet`: *array* di strutture contenenti informazioni sulle misurazioni dei sensori.

```
struct N_BumperSet
{
   BOOL DataActive;
   BOOL TimeStampActive;
   N_CONTS unsigned int BumperCount;
   struct N_Bumper Bumper[N_MAX_BUMPER_COUNT];
};
```

★ `DataActive`: viene impostato a `TRUE` se i dati del *bumper* per il *set* corrente sono stati aggiornati.

★ `TimeStampActive`: viene impostato a `TRUE` se il tempo[8] è stato aggiornato per il *set* corrente.

★ `BumperCount`: rappresenta il numero di *bumper* nel *set*. I valori dei quattro sensori flottanti di contatto montati su ogni porta del XR4000, sono assegnati agli ultimi 4 valori di ogni *set* riferito alla parte superiore di ogni portello (indice da 8 a 11). Per esempio, il *set* di *bumper* numero 2 del robot è il set di sensori della parte superiore della porta numero 2, quindi le letture effettuate sul secondo portello verranno immagazzinate sul secondo *set* di *bumper*; vengono quindi effettuate 12 letture nel *set* di *bumper* numero 2. Questo è il motivo per cui `N_MAX_BUMPER_COUNT` è 12 invece che 10. Le letture dei sensori di contatto posso essere `N_BUMPER_NONE`, `N_BUMPER_LOW` oppure `N_BUMPER_HIGH`.

La struttura `N_Bumper` è definita come:

```
struct N_Bumper
{
```

[8] Tempo in cui è avvenuta l'acquisizione del sensore tattile.

```
   char Reading;
   unsigned long TimeStamp;
};
```

I valori definiti per ogni *bumper* sono:

★ `Reading`: uno per ogni *bumper,* è una delle costanti definite in *Nclient.h.* Per il nomad XR4000, può avere valore `N_BUMPER_NONE`, `N_BUMPER_LOW` oppure `N_BUMPER_HIGH` per un urto particolarmente violento.

★ `Timestamp`: istante temporale di acquisizione della rilevazione

Per ottenere la lettura delle informazioni acquisite dai sensori di contatto è sufficiente effettuare una chiamata alla funzione `N_GetBumper()`, che aggiorna la struttura `N_BumperController`.

4.1.9 Sensori di prossimità all'infrarosso

I sensori ad infrarosso restituiscono informazioni sulla distanza di oggetti vicini (tipicamente tra i $30cm$ a $50cm$), si veda per completezza la sezione 2.5.1. Il Nomad XR4000 ha 48 sensori di prossimità all'infrarosso organizzati sul perimetro superiore ed inferiore del robot, in sei gruppi da otto. Su ognuna delle tre porte, numerate in senso antiorario, si veda la figura 4.2, sono montati due set di sensori, per un totale di sei *set.*

Le informazioni sul rilevamento infrarosso sono contenute nella struttura `N_InfraredController`:

```
struct N_InfraredController
{
   BOOL InfraredPaused;
   N_CONST unsigned int InfraredSetCount;
   struct N_InfraredSet InfraredSet[N_MAX_INFRARED_SET_COUNT];
};
```

★ `InfraredPaused`: se posto a `TRUE` l'acquisizione dei sensori viene interrotta.

★ `InfraredSetCount`: rappresenta la quantità di sensori presenti nel *set*.

★ `N_InfraredSet`: *array* di strutture contenenti informazioni sulle misurazioni dei sensori.

`N_InfraredSet` è così definita:

```
struct N_InfraredSet
{
   BOOL DataActive;
   BOOL TimeStampActive;
   N_CONST unsigned int InfraredCount;
   struct N_Infrared Infrared[N_MAX_INFRARED_COUNT];
};
```

Essa contiene un'*array* di strutture (una per ogni trasduttore del *set*), più un numero di parametri di configurazione.

★ `DataActive`: posto a `TRUE` se i dati sono stati aggiornati.

★ `TimeStampActive`: viene impostato a `TRUE` se il tempo[9] è stato aggiornato per il *set* corrente.

★ `InfraredCount`: quantità di sensori presenti nel *set*.

La struttura `N_Infrared` è definita come:

```
struct N_Infrared
{
   long Reading;
   unsigned long Timestamp;
};
```

[9] Tempo in cui è avvenuta l'acquisizione del sensore infrarosso.

I valori definiti per ogni sensore infrarosso sono:

★ Reading: un valore da 0 a 255 che rappresenta la quantità di energia riflessa dall'oggetto. Il valore 0 indica che non è stata riflessa radiazione infrarossa dall'oggetto (bersaglio distante), mente il valore di 255 rappresenta la massima quantità di energia riflessa (bersaglio vicino). Se è necessaria una misura di distanza, l'utente deve costruire una tabella di calibrazione misurando l'energia riflessa da un campione di materiale rappresentativo dell'ambiente di lavoro, a varie distanze.

★ Timestamp: istante temporale di acquisizione della rilevazione

Per ottenere la lettura delle informazioni acquisite dai sensori ad infrarosso è sufficiente effettuare una chiamata alla funzione `N_GetInfrared()`, che aggiorna la struttura `N_InfraredController`.

4.1.10 Sensori di prossimità ad ultrasuoni

Il sensore ad ultrasuoni fornisce misure di distanza per oggetti che sono a una distanza tra i $15cm$ e i $700cm$. Per una trattazione più accurata si rimanda alla sezione 2.5.1. La struttura del *controller* sonar racchiude tutti i dati validi per tutti i *set* di sonar del robot. Questi *set* sono gruppi di sonar che rilevano contemporaneamente; per esempio, l'ordine di attivazione può essere definito tra sonar dello stesso *set*. La struttura del *controller* sonar contiene un *array* di strutture `N_SonarSet`, ognuna che descrive un particolare *set* di sensori.

Il Nomad XR4000 ha 48 sensori sonar di prossimità, installati sul perimetro superiore ed inferiore, organizzati in sei gruppi da otto. Su ognuna delle tre porte, numerate in senso antiorario, si veda la figura 4.2, sono montati due set di sensori, per un totale di sei *set*

La struttura `SonarController` in `N_RobotState` è definita come:

```
struct N_SonarController
```

```
{
   N_CONST unsigned int SonarSetCount;
   struct N_SonarSet SonarSet[N_MAX_SONAR_SET_COUNT];
   BOOL SonarPaused;
};
```

★ `SonarSetCount`: rappresenta la quantità di sensori ad ultrasuono presenti nel *set*.

★ `N_SonarSet`: *array* di strutture contenenti informazioni sulle misurazioni dei sensori.

La struttura `N_SonarSet` è definita come:

```
struct N_SonarSet
{
   unsigned int FiringOrder[N_MAX_SONAR_COUNT + 1];
   long FiringDelay;
   long BlankingInterval;
   BOOL DataActive;
   BOOL TimeStampActive;
   N_CONST unsigned int SonarCount;
   struct N_Sonar Sonar[N_MAX_SONAR_COUNT];
};
```

Essa contiene un array `Sonar` di strutture sonar (una per ogni trasduttore del *set*), più altri parametri di configurazione.

★ `FiringDelay`: attesa, espressa in millisecondi, tra due attivazioni consecutive di un sensore sonar. Settare questo parametro a un valore alto previene fenomeni di eco multipli (si veda per completezza la sezione 2.5.1).

★ `Intervallo di` *`blanking`*: non implementato[10]

★ `DataActive`: viene impostato a `TRUE` se i dati del sensore sonar del *set* corrente sono stati aggiornati.

★ `TimeStampActive`: viene impostato a `TRUE` se il tempo[11] è stato aggiornato per il *set* corrente.

★ `SonarCount`: rappresenta la quantità di sonar nel *set* corrente

★ `FiringOrder`: è un array di indici dei sonar, terminato da `N_END_SONAR_FIRING_ORDER`, se la lunghezza dell'array è inferiore a quella rappresentata da `SonarCount`[12].

La struttura `N_Sonar` contiene le misurazioni effettuate dal sensore:

```
struct N_Sonar
{
   long Reading;
   unsigned long TimeStamp;
};
```

I valori definiti per ogni sensore sonar sono:

★ `Reading` Misura di distanza effettuata dal sensore in millimetri. Quando il sensore non riceve eco (per esempio a causa di una eccessiva distanza dal bersaglio) , questo valore viene imposto a `N_SONAR_TIMEOUT`

★ `Timestamp` istante temporale di acquisizione della rilevazione

[10]Intervallo di attesa, espresso in millisecondi, tra l'attivazione di un sensore sonar fino all'attivazione del sensore come ricevitore.

[11]Tempo in cui è avvenuta l'acquisizione del sensore sonar.

[12]Ogni indice rappresenta un sensore sonar del set.

Per ottenere la lettura delle informazioni acquisite dai sensori ad ultrasuono è sufficiente effettuare una chiamata alla funzione `N_GetSonar()`, che aggiorna la struttura `N_SonarController`.

Similmente, per ottenere informazioni riguardo alla configurazione corrente del sistema di rilevamento sonar, è necessario invocare `N_GetSonarConfiguration`; invece, per impostare la configurazione basta chiamare `N_SetSonarConfiguration()` dopo aver modificato le variabili `FiringDelay` e `FiringOrder`.

4.1.11 Sensore laser (Sensus 550)

Il Sensus 550 è un sistema di telemetria basato su un sensore Sick Elettro-ottico LMS. Per ottenere le misurazioni effettuate dal laser è sufficiente chiamare la funzione `N_GetS550` che si occupa di aggiornare la struttura `N_S550Set`, presente in `N_RobotState`, con i dati rilevati.

La struttura `N_S550Set` è così definita:

```
struct N_S550set
{
   N_CONST unsigned int S550Count;
   struct N_S550 S550[N_MAX_S550_COUNT];
};
```

★ `S550Count`: indica la quantità di S550 presenti nel robot.

★ `N_S550`: *array* di strutture contenenti informazioni sulle misurazioni dei sensori.

La struttura `N_S550` è definita come:

```
struct N_S550
{
   N_CONST unsigned int TotalPoints;
```

```
    unsigned int RequestedPoints;
    unsigned long Readings[N_MAX_S550_POINTS];
    unsigned char StatusFlags[N_MAX_S550_POINTS];
    unsigned char SummaryFlags;
    unsigned long TimeStamp;
    BOOL DataActive;
    BOOL TimeStampActive;
};
```

★ `TotalPoints`: numero di punti dell'array `Reading`. `TotalPoints` è impostato durante in fase di inizializzazione a 360 o 180 punti[13], e non dovrebbe essere modificato.

★ `RequestedPoints`: indica il numero di misurazioni che l'utente desidera per ogni scan laser. Può essere settato a una quantità limitata di valori: 9, 10, 15, 18, 30, 45, 90, 180, 361. Se viene selezionato un altro valore, non presente nella lista, sarà restituito `N_INVALID_ARGUMENT`. Se si sceglie un numero inferiore a 361, il sensore restituisce la quantità di misure effettuate, selezionandole in modo omogeneo sui 180^0 di scansione[14]

★ `Readings`: array in cui sono contenute tutte le misurazioni, in cui il primo elemento dell'array rappresenta la misurazione periferica destra e l'ultimo quella periferica sinistra. Il laser, cioè, effettua la scansione da destra a sinistra, spazzando uno spicchio di cerchio.

★ `StatusFlag`: non implementato[15]

★ `SummaryFlags`: non implementato[16]

[13] Dipende dal modello S550 installato nel robot.

[14] Ad esempio per un `RequestedPoints` di 10, la misurazione sarà effettuata ogni 18^0.

[15] *Flag* di stato di ogni *array* `Readings`. Potrebbe essere utilizzato per verificare se una misurazione è considerata affidabile o no.

[16] *Flag* di stato per un intero *set* di rilevazioni. Potrebbe essere utilizzato per indicare che una delle misurazioni non è stata affidabile.

★ `TimeStamp`: istante temporale di acquisizione della rilevazione da parte della scansione laser

★ `DataActive`: se questa variabile è impostata a `TRUE`, la struttura verrà aggiornata quando vengono effettuate chiamate alla libreria *client.*

★ `TimeStampActive`: quando impostato a `TRUE`, il campo TimeStamp verrà aggiornato quando vengono effettuate chiamate alla libreria *client*

4.1.12 Sistema di alimentazione

Le informazioni ottenibili sul sistema di alimentazione consistono nella lettura delle tensione ai capi di ogni batteria. Effettuando una chiamata a `N_GetBattery()`, verrà aggiornata la struttura `N_BatterySet`, di `N_RobotState`.

`BatterySet` è così definita:

```
struct N_BatterySet
{
   struct N_Battery Battery[N_MAX_BATTERY_COUNT];
   BOOL DataActive;
}
```

Nella figura 4.3 sono mostrate le correlazioni tra indici dell'array e posizione delle batterie.

La struttura `N_Battery` è definita come:

```
struct N_Battery
{
   long Voltage;
}
```

★ `Voltage`: fornisce, in millivolt, la tensione misurata della batteria.

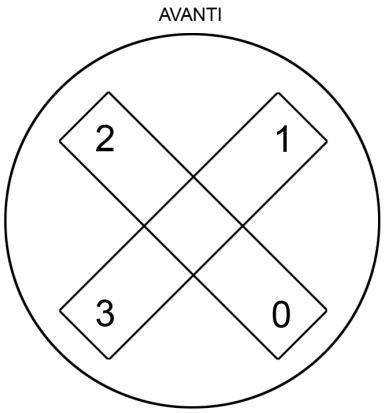


Figura 4.3: Posizione delle batterie rispetto agli indici dell'array `Battery`.

4.1.13 Sintetizzatore vocale DoubleTalk

Il sintetizzatore vocale DoubleTalk, è un sintetizzatore *text-to-speech,* che permette di trasformare stringhe ASCII, in inglese, in fonemi. Questo è facilmente ottenibile passando come argomento una stringa terminata da *fine stringa* alla funzione `N_speak()`, la quale è così definita:

```
N_SpeakN_Speak(unsigned long RobotID, char *Text);
```

4.1.14 Bussola magnetica digitale

Il robot è equipaggiato anche di una bussola magnetica digitale, capace di una precisione 0.5^0, si rimanda per completezza alla sezione 2.5.3. È possibile accedere alle informazioni sull'orientamento rilevato dalla bussola tramite una chiamata alla funzione `Compass_get()` che aggiorna la struttura `Compass_struct`.

La struttura `Compass_struct` è così definita:

```
struct Compass_struct
{
   int    Port_fd;
   char   Port [MAX_STR_LEN_PORT];
```

```
    int     Port_flags;
    char    Raw [MAX_STR_LEN_MSG];
    double Heading;
    char    Status;
    struct termios OldCfg;
};
```

★ `Port_fd`: contiene il *file descriptor* della porta seriale.

★ `Port`: contiene la stringa del device scelto per la porta seriale (/dev/ttySx).

★ `Port_flags`: contiene flag di configurazione della bussola.

★ `Raw`: rappresenta la stringa non processata della risposta restituita dalla bussola digitale.

★ `Heading`: contiene la rotta corrente, espressa in gradi (da 0.0 a 360.0) del robot.

★ `Status`: variabile contenente informazioni sullo stato della bussola.

★ `OldCfg`: variabile riservata contenente la configurazione della porta seriale precede all'attivazione della bussola.

Per ottenere le informazioni sulla rotta, è necessario, prima di chiamare la funzione `Compass_get()`, connettersi correttamente alla bussola magnetica con `Compass_open()`.

Le librerie della bussola magnetica, sono librerie appositamente scritte, ed incluse nell'ambiente di sviluppo. Per il corretto funzionamento è necessario includere il file `compass_lib.h`

4.2 Programming reference

COMPASS_CLOSE

NOME

`Compass_close`

SCOPO

Terminare la connessione con la bussola digitale.

SINTASSI

```
int Compass_close(struct Compass_Struct *Compass)
```

ARGOMENTI

`struct Compass_Struct *Compass`: puntatore a una struttura di tipo `Compass_Struct` contenente la configurazione della bussola digitale.

VALORI RESTITUITI

`COMPASS_CLOSE_SUCCESS`: esecuzione senza errori.
`COMPASS_CLOSE_ERROR`: impossibile terminare la connessione.

VARIABILI GLOBALI AGGIORNATE

Nessuna

DESCRIZIONE

Per l'utilizzo di questa funzione è necessario includere il file `compass_lib.h`. Questa funzione effettua la corretta disconnessione della bussola digitale. Deve essere chiamata dopo una corretta inizializzazione della periferica effettuata attraverso la funzione `Compass_open`. E' necessario terminare correttamente la connessione sulla porta seriale, altrimenti non sarà possibile ottenere nuovi dati dalla bussola digitale.

ESEMPI

`Compass_example.c`

```
#include "compass_lib.h"
#include <string.h>
#include <stdio.h>

int main(void)
{
   struct Compass_Struct Compass;
   strcpy(Compass.Port, "/dev/ttyS1");

   // Compass default configuration
   Compass.Port_flags = COMPASS_DEFAULT;

   // Open compass port
   Compass_open(&Compass);

   // Reads heading from compass
   if(Compass_get(&Compass))
      printf("Current Heading: %f Status:%d\n",
             Compass.Heading,Compass.Status);
```

```
    // Close compass port
    Compass_close(&Compass);
}
```

RIFERIMENTI

`Compass_get`, `Compass_open`.

COMPASS_GET

NOME

Compass_get

SCOPO

Ottenere la rotta rilevata dalla bussola digitale.

SINTASSI

```
int Compass_get(struct Compass_Struct *Compass);
```

ARGOMENTI

`struct Compass_Struct *Compass`: puntatore a una struttura di tipo `Compass_Struct` contenente la configurazione della bussola digitale.

VALORI RESTITUITI

`COMPASS_NO_ERROR`: rilevazione effettuata con successo.
`COMPASS_GET_READ_ERROR`: lettura errata.
`COMPASS_GET_COMMAND_ERROR`: comando non riconosciuto.
`COMPASS_GET_SERIAL_ERROR`: errore sulla comunicazione seriale.

VARIABILI GLOBALI AGGIORNATE

Nessuna

DESCRIZIONE

Per l'utilizzo di questa funzione è necessario includere il file `compass_lib.h`. Questa funzione effettua il rilevamento della rotta rilevata dalla bussola digitale. Il risultato è restituito in virgola mobile da $[0.0 - 360.0]$. Deve essere chiamata dopo una corretta inizializzazione della periferica effettuata attraverso la funzione `Compass_open`. E' necessario terminare correttamente la connessione sulla porta seriale, altrimenti non sarà possibile ottenere nuovi dati dalla bussola digitale.

ESEMPI

`Compass_example.c`

```
#include "compass_lib.h"
#include <string.h>
#include <stdio.h>

int main(void)
{
   struct Compass_Struct Compass;
   strcpy(Compass.Port, "/dev/ttyS1");

   // Compass default configuration
   Compass.Port_flags = COMPASS_DEFAULT;

   // Open compass port
   Compass_open(&Compass);

   // Reads heading from compass
   if(Compass_get(&Compass))
      printf("Current Heading: %f Status:%d\n",
             Compass.Heading,Compass.Status);
```

```
    // Close compass port
    Compass_close(&Compass);
}
```

RIFERIMENTI

`Compass_close`, `Compass_open`.

COMPASS_OPEN

NOME

Compass_open

SCOPO

Effettuare la connessione con la bussola digitale.

SINTASSI

```
int Compass_open(struct Compass_Struct *Compass)
```

ARGOMENTI

struct Compass_Struct *Compass: puntatore a una struttura di tipo Compass_Struct contenente la configurazione della bussola digitale.

VALORI RESTITUITI

N_NO_ERROR: esecuzione senza errori.
N_CONNECTION_FAILED: impossibile connettersi alla porta seriale selezionata, connessione fallita.

VARIABILI GLOBALI AGGIORNATE

Nessuna

DESCRIZIONE

Per l'utilizzo di questa funzione è necessario includere il file `compass_lib.h`. Questa funzione effettua una connessione alla porta seriale su cui è collegata la periferica con i parametri di collegamento specificati nella struttura `Compass`, che deve essere precedentemente allocata. Questo è il primo passo da effettuare per ottenere la rotta fornita dalla bussola digitale.

```
struct Compass_Struct
{
    int    Port_fd;
    char   Port [MAX_STR_LEN_PORT];
    int    Port_flags;

    char   Raw [MAX_STR_LEN_MSG];

    double Heading;
    char   Status;


    struct termios OldCfg;

};
```

★ `Port_fd`: contiene il *file descriptor* della porta seriale.

★ `Port`: contiene la stringa del device scelto per la porta seriale (/dev/ttySx).

★ `Port_flags`: contiene flag di configurazione della bussola. Assegnare questa variabile a `COMPASS_DEFAULT` se la bussola digitale KVH-C100 è collegata sulla seconda porta seriale (`/dev/ttyS2`) con parametri di comunicazione `9600,8,N,1` impostati dalla fabbrica.

★ `Raw`: rappresenta la stringa non processata della risposta restituita dalla bussola digitale.

★ `Heading`: contiene la rotta corrente, espressa in gradi (da 0.0 a 360.0) del robot.

★ `Status`: variabile contenente informazioni sullo stato della bussola. Se `Status` contiene `TRUE` la bussola ha effettuato una rilevazione corretta, se invece contiene `FALSE` l'ambiente di rilevazione risultava troppo disturbato (*overloaded*) per effettuare una misura e il risultato rilevato non è valido.

★ `OldCfg`: variabile riservata contenente la configurazione della porta seriale precede all'attivazione della bussola.

ESEMPI

`Compass_example.c`

```
#include "compass_lib.h"
#include <string.h>
#include <stdio.h>

int main(void)
{
   struct Compass_Struct Compass;
   strcpy(Compass.Port, "/dev/ttyS1");

   // Compass default configuration
   Compass.Port_flags = COMPASS_DEFAULT;

   // Open compass port
   Compass_open(&Compass);
```

```
   // Reads heading from compass
   if(Compass_get(&Compass))
      printf("Current Heading: %f Status:%d\n",
             Compass.Heading,Compass.Status);

   // Close compass port
   Compass_close(&Compass);
}
```

RIFERIMENTI

`Compass_close`, `Compass_get`.

N_CONNECTROBOT

NOME

`N_ConnectRobot`

SCOPO

Connettersi al robot.

SINTASSI

```
int N_ConnectRobot(long RobotID);
```

ARGOMENTI

`long RobotID`: numero identificativo del robot.

VALORI RESTITUITI

`N_NO_ERROR`: esecuzione senza errori.
`N_UNINITIALIZED`: non è stata chiamata `N_InitializeClient`.
`N_ROBOT_NOT_FOUND`: il robot con tale identificativo (`RobotID`) non è stato registrato nello *scheduler.*
`N_CONNECTION_FAILED`: connessione fallita.
`N_OUT_OF_MEMORY`: impossibile allocare le strutture di dati.

VARIABILI GLOBALI AGGIORNATE

`N_RobotState`

DESCRIZIONE

Questa funzione attua una connessione al robot con lo specificato RobotID. Questa connessione è necessaria per comunicare comandi e ricevere informazioni dal robot.

ESEMPI

N_Connect_Robot.c

```
#include <stdio.h>
#include "Nclient.h"
#define SCHEDULER_HOSTNAME "localhost"
#define SCHEDULER_PORT 7073

int main()
{
   N_InitializeClient(SCHEDULER_HOSTNAME, SCHEDULER_PORT);

   switch (N_ConnectRobot(1))
   {
      case N_NO_ERROR:
        printf("Successfully connected to Robot\n");
      break;
      case N_ROBOT_NOT_FOUND:
        printf("Robot not found\n");
      break;
      case N_CONNECTION_FAILED:
        printf("Connection failed\n");
      break;
   }
```

```
    N_DisconnectRobot(1);
    exit(0);
}
```

RIFERIMENTI

N_DisconnectRobot.

N_DISCONNECTROBOT

NOME

`N_DisconnectRobot`

SCOPO

Disconnettersi da un determinato robot.

SINTASSI

```
int N_DisconnectRobot(long RobotID);
```

ARGOMENTI

`long RobotID`: numero identificativo del robot.

VALORI RESTITUITI

`N_NO_ERROR`: esecuzione senza errori.
`N_ROBOT_NOT_FOUND`: il robot con tale identificativo (ID) non è stato registrato nello *scheduler.*
`N_CONNECTION_FAILED`: connessione fallita.

VARIABILI GLOBALI AGGIORNATE

Nessuna.

DESCRIZIONE

Questa funzione attua la disconnessione dal robot con lo specificato `RobotID`.

ESEMPI

N_Disconnect_Robot.c

```
#include <stdio.h>
#include "Nclient.h"
#define SCHEDULER_HOSTNAME "localhost"
#define SCHEDULER_PORT 7073

int main()
{
   N_InitializeClient(SCHEDULER_HOSTNAME, SCHEDULER_PORT);
   N_ConnectRobot(1);

   switch (N_DisconnectRobot(1))
   {
      case N_NO_ERROR:
        printf("Successfully disconnected from Robot\n");
      break;
      case N_CONNECTION_FAILED:
        printf("Connection was lost\n");
      break;
      case N_ROBOT_NOT_FOUND:
        printf("Robot not found\n");
      break;
   }

   exit(0);
}
```

RIFERIMENTI

`N_ConnectRobot`.

N_GETAXES

NOME

`N_GetAxes`

SCOPO

Aggiornare la struttura `N_RobotState` con i dati degli *encoder* provenienti dai sensori del robot.

SINTASSI

```
int N_GetAxes(long RobotID);
```

ARGOMENTI

`long RobotID`: numero identificativo del robot.

VALORI RESTITUITI

`N_NO_ERROR`: esecuzione senza errori.
`N_ROBOT_NOT_FOUND`: robot non trovato.
`N_CONNECTION_FAILED`: socket di connessione disconnesso, connessione fallita.
`N_UNKNOWN_ERROR`: Drobot non è riuscito ad ottenere l'informazione richiesta.

VARIABILI GLOBALI AGGIORNATE

`N_RobotState`

DESCRIZIONE

Questa funzione restituisce la configurazione corrente di tutti gli assi del robot e aggiorna la struttura `N_RobotState`.

```
struct N_AxisSet
{
   BOOL Global;
   unsigned char Status;
   N_CONST unsigned int AxisCount;
   struct N_Axis Axis[N_MAX_AXIS_COUNT];
};
```

★ `Global`: quando questo parametro è impostato `TRUE`, setta gli assi della base mobile del robot nella modalità *global*, o *world*, in cui il controllo della base avviene basandosi su coordinate di un riferimento fisso. Il riferimento fisso è impostato quando il robot viene acceso e compie l'operazione di *zeroing*[17] oppure quando è chiamata `N_SetIntegratedConfiguration()`. In modalità *global* il movimento di rotazione degli assi non influenza la direzione degli assi x e y. Per esempio un movimento simultaneo di rotazione e traslazione sull'asse y, provocherà un movimento del robot su linea retta e contemporaneamente una rotazione[18]. Quando questo parametro è impostato `FALSE`, gli assi dei motori sono controllati basandosi su un riferimento locale, o *joint.* In questa modalità il movimento di rotazione degli assi del robot influenza il moto degli assi di traslazione. Per esempio un movimento simultaneo di rotazione e traslazione sull'asse y, provocherà un movimento circolare del robot; in realtà il robot procederà a muoversi sull'asse y, ma la rotazione impostata causerà un cambiamento di direzione su quest'asse, generando un movimento circolare.

[17] Quell'operazione in cui muove le ruote affinchè gli encoder di ogni asse risultino essere zero.

[18] Movimento a pirouette.

★ Status: può essere impostato a uno dei seguenti valori:

N_AXES_READY: indica che gli assi sono pronti ad effettuare un movimento.
N_JOYSTICK_IN_USE: indica che il robot è controllato via *joystick*
N_ESTOP_DOWN: indica che uno o più *emergency stop* sono premuti e prevengono il movimento della base.
N_MOTION_ERROR: l'ultimo movimento eseguito è fallito.

```
struct N_Axis
{
   BOOL DataActive;
   BOOL TimeStampActive;
   BOOL Update;
   unsigned long TimeStamp;
   char Mode;
   long DesiredPosition;
   long DesiredSpeed;
   long Acceleration;
   long TrajectoryPosition;
   long TrajectoryVelocity;
   long ActualPosition;
   long ActualVelocity;
   BOOL InProgress;
   long TrajectoryVelocity;
};
```

★ DataActive: il valore TRUE per questo parametro implica che i valori in questa struttura saranno aggiornati. Precisamente verranno aggiornati: Mode, DesiredPosition, DesiredSpeed, Acceleration, TrajectoryPosition, TrajectoryVelocity, ActualPosition, ActualVelocity, InProgress e TrajectoryVelocity.

★ TimeStampActive: il valore TRUE per questo parametro implica che sarà aggiornato il valore di TimeStamp.

★ `Update`: il valore `TRUE` per questo parametro implica che i valori di *input*: `DesiredSpeed`, `DesiredPosition` ed `Acceleration` saranno caricati per l'asse corrente quando verrà eseguita la funzione `N_SetAxes`. Il parametro `Update` permette di assegnare simultaneamente a uno o più assi i nuovi valori di input.

★ `TimeStamp`: il valore del tempo (in millisecondi) al quale sono stati misurati i valori.

★ `Mode`: può essere una dei seguenti:

`N_AXIS_POSITION_RELATIVE`: specifica che l'asse si muove relativamente alla posizione corrente.

`N_AXIS_POSITION_ABSOLUTE`: specifica che l'asse si muove verso una posizione rispetto alla posizione del riferimento assoluto.

`N_AXIS_VELOCITY`: specifica che l'asse si muove con una velocità costante.

`N_AXIS_STOP`: ferma l'asse.

★ `DesiredPosition`: specifica la posizione finale desiderata dell'asse. Le unità sono in millimetri per le traslazioni ed in milliradianti per le rotazioni. Quando il campo `Mode` è settato a `N_AXIS_POSITION_RELATIVE` oppure a `N_AXIS_POSITION_ABSOLUTE`, questo specifica rispettivamente la posizione finale relativa alla posizione corrente o a quella assoluta. Questo parametro non viene utilizzato quando `Mode` è settato a `N_AXIS_VELOCITY` oppure a `N_AXIS_STOP`. Questo parametro può avere valore positivo o negativo.

★ `DesiredSpeed`: specifica la velocità alla quale deve essere effettuato il movimento verso la posizione desiderata (`DesiredPosition`), oppure la velocità costante con cui muoversi quando `Mode` è impostato a `N_AXIS_VELOCITY`. Le unità sono espresse in mm/s (millimetri al secondo) per l'asse di traslazione e $mRad/s$ (milliradianti al secondo) per l'asse di rotazione.

★ `Acceleration`: specifica l'accelerazione relativa al parametro `DesiredSpeed`. Le unità sono espresse in mm/s^2 (millimetri al secondo^2) per

l'asse di traslazione e $mRad/s^2$ (milliradianti al secondo^2) per l'asse di rotazione. Questo parametro può solo avere un valore positivo.

★ `TrajectoryPosition`: fornisce la posizione corrente del generatore di traiettoria.

★ `TrajectoryVelocity`: fornisce la velocità corrente del generatore di traiettoria.

★ `ActualPosition`: fornisce la posizione attuale dell'asse. Il valore di questa variabile è basato sul modo in cui è impostato il campo *Global.* Se *Global* =`TRUE` (condizione di *Global mode*), questo campo fornisce la posizione dell'asse rispetto al sistema di riferimento fisso (questo stesso valore può essere ricavato dal corrispondente campo della struttura `N_Integrator`). Se *Global* =`FALSE` (condizione di *Joint mode*), questo campo fornisce la posizione dell'asse rispetto al sistema di coordinate di giunto.

★ `ActualVelocity`: fornisce la velocità attuale dell'asse. Il valore di questa variabile è basato sul modo in cui è impostato il campo *Global.* Se *Global* =`TRUE` (condizione di *Global mode*), questo campo fornisce la posizione dell'asse rispetto al sistema di riferimento fisso. Se *Global* =`FALSE` (condizione di *Joint mode*), questo campo fornisce la posizione dell'asse rispetto al sistema di coordinate di giunto.

★ `InProgress`: fornisce un valore booleano che informa l'utente che un asse è in movimento.

L'interpretazione e i valori assumibili dagli assi sono riassunti nella tabella 4.1.

ESEMPI

`N_GetAxes.c`

```
#include <stdio.h>
```

Asse 0	X
Asse 1	Y
Asse 2	θ
Valori velocità asse 0	$[-1500mm/s, 1500mm/s]$
Valori velocità asse 1	$[-1500mm/s, 1500mm/s]$
Valori velocità asse 2	$[-5000mRad/s, 5000mRad/s]$
Valori accelerazione asse 0	$[0mm/s^2, 1500mm/s^2]$
Valori accelerazione asse 1	$[0mm/s^2, 1500mm/s^2]$
Valori accelerazione asse 1	$[0mm/s^2, 5000mm/s^2]$

Tabella 4.1: Caratteristiche degli assi del robot.

```
#include "Nclient.h"
#define SCHEDULER_HOSTNAME "localhost"
#define SCHEDULER_PORT 7073

int main()
{
   struct N_RobotState *state;
   struct N_Axis *axis;
   N_InitializeClient(SCHEDULER_HOSTNAME, SCHEDULER_PORT);
   N_ConnectRobot(1);
   state = N_GetRobotState(1);
   axis = &(state->AxisSet.Axis[N_XTRANSLATION]);

   while(1)
   {
      N_GetAxes(1);
      printf("Position %d mm\n", axis->CurrentPosition);
   }

   N_DisconnectRobot(1);
```

```
    exit(0);
}
```

RIFERIMENTI

`N_SetAxis`, `N_GetState`.

N_GETBATTERY

NOME

`N_GetBattery`

SCOPO

Aggiornare la struttura `N_RobotState` con le informazioni sulle batterie.

SINTASSI

```
int N_GetBattery(long RobotID);
```

ARGOMENTI

`long RobotID`: numero identificativo del robot.

VALORI RESTITUITI

`N_NO_ERROR`: esecuzione senza errori.
`N_ROBOT_NOT_FOUND`: robot non trovato.
`N_CONNECTION_FAILED`: socket di connessione disconnesso, connessione fallita.

VARIABILI GLOBALI AGGIORNATE

`N_RobotState`

DESCRIZIONE

La struttura `BatterySet` contiene semplicemente un array di puntatori alle strutture delle batterie.

```
struct N_BatterySet
{
   struct N_Battery Battery[N_MAX_BATTERY_COUNT];
   BOOL DataActive;
};
```

Il riferimento tra indice dell'array `Battery` e la disposizione fisica delle batterie è data dalla figura 4.3.

```
struct N_Battery
{
   long Voltage;
};
```

★ `Voltage`: fornisce la tensione della batteria corrente in mV (millivolt)

ESEMPI

```
#include <stdio.h>
#include "Nclient.h"
#define SCHEDULER_HOSTNAME "localhost"
#define SCHEDULER_PORT 7073

int main()
{
   struct N_RobotState *state
   struct N_Battery *battery;

   N_InitializeClient(SCHEDULER_HOSTNAME, SCHEDULER_PORT);
   N_ConnectRobot(1);

   state = N_GetRobotState(1);
```

```
    battery = &(state->BatterySet.Battery[0]);

    N_GetBattery(1);

    printf("Battery Voltage %d\n", battery->Voltage);

    N_DisconnectRobot(1);
    exit(0);
}
```

RIFERIMENTI

N_GetState.

N_GETBUMPER

NOME

`N_GetBumper`

SCOPO

Aggiornare il campo `N_BumperSet` nella struttura `N_RobotState` con i dati dei sensori di contatto.

SINTASSI

```
int N_GetBumper(long RobotID);
```

ARGOMENTI

`long RobotID`: numero identificativo del robot.

VALORI RESTITUITI

`N_NO_ERROR`: esecuzione senza errori.
`N_ROBOT_NOT_FOUND`: robot non trovato.
`N_CONNECTION_FAILED`: socket di connessione disconnesso, connessione fallita.

VARIABILI GLOBALI AGGIORNATE

`N_RobotState`

DESCRIZIONE

Questa funzione preleva i dati forniti dai sensori di collisione e aggiorna la struttura `N_RobotState`. La struttura del *controller* dei *bumper* contiene tutti i dati di configurazione validi per tutti i sensori del robot. I *bumper set* sono gruppi di sensori tattili che agiscono insieme. Il *controller* dei *bumper* contiene un array di strutture `N_BumperSet`, ognuna che descrive un particolare *set*. Quarantotto sensori bilivello circondano il perimetro superiore ed inferiore del robot, fornendo l'esatta locazione del contatto come pure la forza di impatto (nessuna, bassa, alta). Una collisione a bassa velocità accende una spia verde sull'elemento sensoriale, una pressione più intensa illumina una spia rossa. Una robusta guaina di gomma protegge i sensori dalle collisioni. Il Nomad XR4000 ha tre porte numerate in senso antiorario in cui su ognuna sono montati due set di sensori, si veda la figura 4.2, per un totale di sei.

```
struct N_BumperController
{
   N_CONST unsigned int BumperSetCount;
   struct N_BumperSet BumperSet[N_MAX_BUMPER_SET_COUNT];
};
```

I dati di configurazione utilizzati per tutti i *bumper set* del robot sono:

★ `BumperSetCount`: quantità di *bumper* collegati al controller

La struttura `N_BumperSet` è definita come:

```
struct N_BumperSet
{
   BOOL DataActive;
   BOOL TimeStampActive;
   N_CONST unsigned int BumperCount;
   struct N_Bumper Bumper[N_MAX_BUMPER_COUNT];
};
```

La struttura contiene `Bumper`, un array di strutture ed inoltre altri parametri di configurazione:

★ `DataActive`: viene impostato a `TRUE` se i dati del *bumper* per il *set* corrente sono stati aggiornati.

★ `TimeStampActive`: viene impostato a `TRUE` se il tempo[19] è stato aggiornato per il *set* corrente.

★ `BumperCount`: rappresenta il numero di *bumper* nel *set.* I valori dei quattro sensori flottanti di contatto montati su ogni porta del XR4000, sono assegnati agli ultimi 4 valori di ogni *set* riferito alla parte superiore di ogni portello (indice da 8 a 11). Per esempio, il *set* di *bumper* numero 2 del robot è il set di sensori della parte superiore della porta numero 2, quindi le letture effettuate sul secondo portello verranno immagazzinate sul secondo *set* di *bumper*; vengono quindi effettuate 12 letture nel *set* di *bumper* numero 2. Questo è il motivo per cui `N_MAX_BUMPER_COUNT` è 12 invece che 10. Le letture dei sensori di contatto posso essere `N_BUMPER_NONE`, `N_BUMPER_LOW` oppure `N_BUMPER_HIGH`.

La struttura `N_Bumper` è così definita:

```
struct N_Bumper
{
   char Reading;
   unsigned long TimeStamp;
};
```

I valori definiti per ogni bumper sono:

★ `Reading`: uno per ogni *bumper,* è una delle costanti definite in *Nclient.h.* Per il nomad XR4000, può avere valore `N_BUMPER_NONE`, `N_BUMPER_LOW` oppure `N_BUMPER_HIGH` per un urto particolarmente violento.

★ `Timestamp`: istante temporale di acquisizione della rilevazione

[19] Tempo in cui è avvenuta l'acquisizione del sensore tattile.

ESEMPI

N_GetBumper.c

```
#include <stdio.h>
#include "Nclient.h"
#define SCHEDULER_HOSTNAME "localhost"
#define SCHEDULER_PORT 7073

int main()
{
   struct N_RobotState *state;
   struct N_Bumper *bumper;
   short i;

   N_InitializeClient(SCHEDULER_HOSTNAME, SCHEDULER_PORT);
   N_ConnectRobot(1);

   state = N_GetRobotState(1);
   bumper = state->BumperController.BumperSet[0].Bumper;
   while(1)
   {
      N_GetBumper(1);
      printf("BumperSet 0: ");
      for (i = 0; i < 8; i++)
      {
         printf("%d ", bumper[i].Reading);
      }
   printf("\n");

   N_DisconnectRobot(1);
   exit(0);
```

```
}
```

RIFERIMENTI

`N_GetState`.

N_GETINFRARED

NOME

`N_GetInfrared`

SCOPO

Aggiornare la struttura `N_RobotState` con i dati dei sensori di prossimità all'infrarosso.

SINTASSI

```
int N_GetInfrared(long RobotID);
```

ARGOMENTI

`long RobotID`: numero identificativo del robot.

VALORI RESTITUITI

`N_NO_ERROR`: esecuzione senza errori.
`N_ROBOT_NOT_FOUND`: robot non trovato.
`N_CONNECTION_FAILED`: socket di connessione disconnesso, connessione fallita.

VARIABILI GLOBALI AGGIORNATE

`N_RobotState`

DESCRIZIONE

Questa funzione preleva i dati forniti dai sensori di prossimità all'infrarosso e aggiorna la struttura `N_RobotState`. La struttura del *controller* dei sensori all'infrarosso contiene tutti i dati di configurazione validi per tutti i *set* di sensori del robot. I *set* di sensori sono gruppi di sensori all'infrarosso che agiscono insieme. Il *controller* dei sensori contiene un array di strutture `N_InfraredSet`, ognuna che descrive un particolare *set*. Il Nomad XR4000 ha tre porte numerate in senso antiorario, in cui su ognuna sono montati due set di sensori, si veda la figura 4.2, per un totale di sei.

```
struct N_InfraredController
{
   BOOL InfraredPaused;
   N_CONST unsigned int InfraredSetCount;
   struct N_InfraredSet InfraredSet[N_MAX_INFRARED_SET_COUNT];
};
```

I dati di configurazione utilizzati per tutti i *set* di sensori di prossimità all'infrarosso del robot sono:

★ `InfraredPaused`: se posto a TRUE l'acquisizione dei sensori viene interrotta.

★ `InfraredSetCount`: rappresenta la quantità di sensori presenti nel *set*.

★ `N_InfraredSet`: *array* di strutture contenenti informazioni sulle misurazioni dei sensori.

La struttura `N_InfraredSet` è definita come:

```
struct N_InfraredSet
{
   BOOL DataActive;
   BOOL TimeStampActive;
```

```
    N_CONST unsigned int InfraredCount;
    struct N_Infrared Infrared[N_MAX_INFRARED_COUNT];
};
```

Essa contiene un'*array* di strutture (una per ogni trasduttore del *set*), più un numero di parametri di configurazione.

★ `DataActive`: posto a `TRUE` se i dati sono stati aggiornati.

★ `TimeStampActive`: viene impostato a `TRUE` se il tempo[20] è stato aggiornato per il *set* corrente.

★ `InfraredCount`: quantità di sensori presenti nel *set*.

La struttura `N_Infrared` è definita come:

```
struct N_Infrared
{
    long Reading;
    unsigned long Timestamp;
};
```

I valori definiti per ogni sensore infrarosso sono:

★ Reading: un valore da 0 a 255 che rappresenta la quantità di energia riflessa dall'oggetto. Il valore 0 indica che non è stata riflessa radiazione infrarossa dall'oggetto (bersaglio distante), mente il valore di 255 rappresenta la massima quantità di energia riflessa (bersaglio vicino). Se è necessaria una misura di distanza, l'utente deve costruire una tabella di calibrazione misurando l'energia riflessa da un campione di materiale rappresentativo dell'ambiente di lavoro, a varie distanze.

★ Timestamp: istante temporale di acquisizione della rilevazione.

[20] Tempo in cui è avvenuta l'acquisizione del sensore infrarosso.

ESEMPI

N_GetInfrared.c

```
#include <stdio.h>
#include "Nclient.h"
#define SCHEDULER_HOSTNAME "localhost"
#define SCHEDULER_PORT 7073

int main()
{
   struct N_RobotState *state;
   struct N_Infrared *infrared;
   short i;

   N_InitializeClient(SCHEDULER_HOSTNAME, SCHEDULER_PORT);
   N_ConnectRobot(1);

   state = N_GetRobotState(1);
   infrared = state->InfraredController.InfraredSet[0].Infrared;

   while(1)
   {
      N_GetInfrared(1);
      printf("InfraredSet 0: ");
      for (i = 0; i < 8; i++)
        printf("%d ", infrared[i].Reading);
      printf("\n");
   }

   N_DisconnectRobot(1);
   exit(0);
```

```
}
```

RIFERIMENTI

`N_GetState`.

N_GETINTEGRATEDCONFIGURATION

NOME

`N_GetIntegratedConfiguration`

SCOPO

Aggiornare la struttura `N_RobotState` con i dati della configurazione integrata.

SINTASSI

```
int N_GetIntegratedConfiguration(long RobotID);
```

ARGOMENTI

`long RobotID`: numero identificativo del robot.

VALORI RESTITUITI

`N_NO_ERROR`: esecuzione senza errori.
`N_ROBOT_NOT_FOUND`: robot non trovato.
`N_CONNECTION_FAILED`: socket di connessione disconnesso, connessione fallita.

VARIABILI GLOBALI AGGIORNATE

`N_RobotState`

DESCRIZIONE

Questa funzione preleva i dati della configurazione integrata dal robot e aggiorna la struttura `N_RobotState`. La struttura `N_Integrator` contiene le configurazioni geometriche del robot rispetto a un sistema di riferimento fisso. La

configurazione geometrica del Nomad XR4000 consiste nella sua posizione x,y e nel suo angolo di rotazione corrente o da quando è stata fatta una chiamata a N_SetIntegratedConfiguration.

```
struct N_Integrator
{
   BOOL DataActive;
   BOOL TimeStampActive;
   unsigned long TimeStamp;
   long x;
   long y;
   long Steering;
   long Rotation;
}
```

ESEMPI

N_GetIntegratedConfiguration.c

```
#include <stdio.h>
#include "Nclient.h"
#define SCHEDULER_HOSTNAME "localhost"
#define SCHEDULER_PORT 7073

int main()
{
   struct N_RobotState *state;
   struct N_Configuration *integrated_configuration;

   N_InitializeClient(SCHEDULER_HOSTNAME, SCHEDULER_PORT);
   N_ConnectRobot(1);
```

```
   state = N_GetRobotState(1);
   integrated_configuration =&state->Integrator;

   while (1)
   {
      N_GetIntegratedConfiguration(1);
      printf("Integrated Configuration: x %d y %d Steering %d
      Rotation %d\n",
      integrated_configuration->x,
      integrated_configuration->y,
      integrated_configuration->Steering,
      integrated_configuration->Rotation);
   }

   N_DisconnectRobot(1);
   exit(0);
}
```

RIFERIMENTI

N_SetIntegratedConfiguration.

N_GETS550

NOME

N_GetS550

SCOPO

Aggiornare la struttura N_RobotState con i dati rilevati dal sistema laser di rilevamento Sensus 550.

SINTASSI

```
int N_GetS550(long RobotID);
```

ARGOMENTI

long RobotID: numero identificativo del robot.

VALORI RESTITUITI

N_NO_ERROR: esecuzione senza errori.
N_ROBOT_NOT_FOUND: robot non trovato.
N_CONNECTION_FAILED: socket di connessione disconnesso, connessione fallita.
N_INVALID_ARGUMENT: punti richiesti non validi.
N_SENSOR_NOT_READY: il dispositivo non è ancora inizializzato.

VARIABILI GLOBALI AGGIORNATE

N_RobotState

DESCRIZIONE

Questa funzione preleva i dati rilevati dal sensore laser (Sensus 550) e riempie il campo `N_S550Set` di `N_RobotState`.

```
struct N_S550Set
{
   N_CONST unsigned int S550Count;
   struct N_S550 S550[N_MAX_S550_COUNT];
};
```

★ `S550Count`: indica la quantità di S550 presenti nel robot.

★ `N_S550`: *array* di strutture contenenti informazioni sulle misurazioni dei sensori.

La struttura `N_S550` è definita come:

```
struct N_S550
{
   N_CONST unsigned int TotalPoints;
   unsigned int RequestedPoints;
   unsigned long Readings[N_MAX_S550_POINTS];
   unsigned char StatusFlags[N_MAX_S550_POINTS];
   unsigned char SummaryFlags;
   unsigned long TimeStamp;
   BOOL DataActive;
   BOOL TimeStampActive;
};
```

★ `TotalPoints`: numero di punti dell'array `Reading`. `TotalPoints` è impostato durante in fase di inizializzazione a 360 o 180 punti[21], e non dovrebbe essere modificato

[21] Dipende dal modello S550 installato nel robot.

★ `RequestedPoints`: indica il numero di misurazioni che l'utente desidera per ogni scan laser. Può essere settato a una quantità limitata di valori: 9, 10, 15, 18, 30, 45, 90, 180, 361. Se viene selezionato un altro valore, non presente nella lista, sarà restituito `N_INVALID_ARGUMENT`. Se si sceglie un numero inferiore 361, il sensore restituisce la quantità di misure effettuate, selezionandole in modo omogeneo sui 180^0 di scansione[22]

★ `Readings`: array in cui sono contenute tutte le misurazioni, in cui il primo elemento dell'array rappresenta la misurazione periferica destra e l'ultimo quella periferica sinistra. Il laser, cioè, effettua la scansione da destra a sinistra, spazzando uno spicchio di cerchio.

★ `StatusFlag`: non implementato[23]

★ `SummaryFlags`: non implementato[24]

★ `TimeStamp`: istante temporale di acquisizione della rilevazione da parte della scansione laser

★ `DataActive`: se questa variabile è impostata a `TRUE`, la struttura verrà aggiornata quando vengono effettuate chiamate alla libreria *client.*

★ `TimeStampActive`: quando impostato a qTRUE, il campo TimeStamp verrà aggiornato quando vengono effettuate chiamate alla libreria *client*

ESEMPI

`N_GetS550.c`

```
#include <stdio.h>
```

[22] Ad esempio per un `RequestedPoints` di 10, la misurazione sarà effettuata ogni 18^0.

[23] *Flag* di stato di ogni *array* `Readings`. Potrebbe essere utilizzato per verificare se una misurazione è considerata affidabile o no.

[24] *Flag* di stato per un intero *set* di rilevazioni. Potrebbe essere utilizzato per indicare che una delle misurazioni non è stata affidabile.

```
#include "Nclient.h"
#define SCHEDULER_HOSTNAME "localhost"
#define SCHEDULER_PORT 7073

int main()
{
   struct N_RobotState *state;
   struct N_S550 *S550;
   short i;

   N_InitializeClient(SCHEDULER_HOSTNAME, SCHEDULER_PORT);
   N_ConnectRobot(1);

   state = N_GetRobotState(1);
   S550 = &state->S550Set.S550[0];

   while (1)
   {
      N_GetS550(1, 0);
      printf("Front Reading for S550 0: %d\n", S550->Readings[89]);
   }

   N_DisconnectRobot(1);
   exit(0);
}
```

RIFERIMENTI

N_GetState.

N_GETSONAR

NOME

`N_GetSonar`

SCOPO

Aggiornare la struttura `N_RobotState` con i dati rilevati dai sensori di prossimità ad ultrasuono.

SINTASSI

```
int N_GetSonar(long RobotID);
```

ARGOMENTI

`long RobotID`: numero identificativo del robot.

VALORI RESTITUITI

`N_NO_ERROR`: esecuzione senza errori.
`N_ROBOT_NOT_FOUND`: robot non trovato.
`N_CONNECTION_FAILED`: socket di connessione disconnesso, connessione fallita.

VARIABILI GLOBALI AGGIORNATE

`N_RobotState`

DESCRIZIONE

Questa funzione preleva i dati forniti dai sensori di prossimità ad ultrasuono e aggiorna la struttura `N_RobotState`, immagazzinando i dati nel campo `N_Sonar Controller`. La struttura del *controller* dei sensori ad ultrasuono contiene tutti i dati di configurazione validi per tutti i *set* di sensori del robot. I *set* di sensori sono gruppi di sensori sonar che agiscono insieme. Il *controller* dei sensori contiene un array di strutture `N_SonarSet`, ognuna che descrive un particolare *set.* Il Nomad XR4000 ha tre porte numerate in senso antiorario, in cui su ognuna sono montati due set di sensori, si veda la figura 4.2, per un totale di sei.

```
struct N_SonarController
{
   N_CONST unsigned int SonarSetCount;
   struct N_SonarSet SonarSet[N_MAX_SONAR_SET_COUNT];
   BOOL SonarPaused;
};
```

★ `SonarSetCount`: rappresenta la quantità di sensori ad ultrasuono presenti nel *set.*

★ `N_SonarSet`: *array* di strutture contenenti informazioni sulle misurazioni dei sensori.

La struttura `N_SonarSet` è definita come:

```
struct N_SonarSet
{
   unsigned int FiringOrder[N_MAX_SONAR_COUNT + 1];
   long FiringDelay;
   long BlankingInterval;
   BOOL DataActive;
   BOOL TimeStampActive;
```

```
    N_CONST unsigned int SonarCount;
    struct N_Sonar Sonar[N_MAX_SONAR_COUNT];
};
```

Essa contiene un array `Sonar` di strutture sonar (una per ogni trasduttore del *set*), più altri parametri di configurazione.

★ `FiringDelay`: attesa, espressa in millisecondi, tra due attivazioni consecutive di un sensore sonar. Settare questo parametro a un valore alto previene fenomeni di eco multipli (si veda il paragrafo 2.5.1).

★ `Intervallo di` *`blanking`*: non implementato[25]

★ `DataActive`: viene impostato a `TRUE` se i dati del sensore sonar del *set* corrente sono stati aggiornati.

★ `TimeStampActive`: viene impostato a `TRUE` se il tempo[26] è stato aggiornato per il *set* corrente.

★ `SonarCount`: rappresenta la quantità di sonar nel *set* corrente

★ `FiringOrder`: è un array di indici dei sonar, terminato da `N_END_SONAR_FIRING_ORDER`, se la lunghezza dell'array è inferiore a quella rappresentata da `SonarCount`[27].

La struttura `N_Sonar` contiene le misurazioni effettuate dal sensore:

```
struct N_Sonar
{
    long Reading;
    unsigned long TimeStamp;
};
```

[25] Intervallo di attesa, espresso in millisecondi, tra l'attivazione di un sensore sonar fino all'attivazione del sensore come ricevitore.

[26] Tempo in cui è avvenuta l'acquisizione del sensore sonar.

[27] Ogni indice rappresenta un sensore sonar del set.

I valori definiti per ogni sensore sonar sono:

★ `Reading` Misura di distanza effettuata dal sensore in millimetri. Quando il sensore non riceve eco (per esempio a causa di una eccessiva distanza dal bersaglio), questo valore viene imposto a `N_SONAR_TIMEOUT`

★ `Timestamp` istante temporale di acquisizione della rilevazione

ESEMPI

`N_GetSonar.c`

```
#include <stdio.h>
#include "Nclient.h"
#define SCHEDULER_HOSTNAME "localhost"
#define SCHEDULER_PORT 7073

int main()
{
   struct N_RobotState *state;
   struct N_Sonar *sonar;
   short i;

   N_InitializeClient(SCHEDULER_HOSTNAME, SCHEDULER_PORT);
   N_ConnectRobot(1);

   state = N_GetRobotState(1);
   sonar = state->SonarController.SonarSet[0].Sonar;

   while(1)
   {
      N_GetSonar(1);
```

```
      printf("SonarSet 0: ");
      for (i = 0; i < 8; i++)
        printf("%d ", sonar[i].Reading);
      printf("\n");
   }

   N_DisconnectRobot(1);
   exit(0);
}
```

RIFERIMENTI

`N_GetSonarConfiguration`, `N_GetState`.

N_GETSONARCONFIGURATION

NOME

N_GetSonarConfiguration

SCOPO

Aggiornare i dati dei sonar nella struttura `N_Robot_State`.

SINTASSI

```
int N_GetSonarConfiguration(long RobotID);
```

ARGOMENTI

`long RobotID`: ID del robot.

VALORI RESTITUITI

`N_NO_ERROR`: operazione eseguita senza errori.
`N_ROBOT_NOT_FOUND`: non è stato trovato il robot.
`N_CONNECTION_FAILED`: il *socket* è stato chiuso dall'ultima chiamata da parte del programma *client.*

VARIABILI GLOBALI AGGIORNATE

`N_RobotState`

DESCRIZIONE

Questo comando recupera la configurazione dei sonar dal robot ed aggiorna la struttura `N_RobotState`.

La struttura del *controller* sonar racchiude tutti i dati validi per tutti i *set* di sonar del robot. Questi *set* sono gruppi di sonar che rilevano contemporaneamente; per esempio, l'ordine di attivazione può essere definito tra sonar dello stesso *set*. La struttura del *controller* sonar contiene un *array* di strutture `N_SonarSet`, ognuna che descrive un particolare *set* di sensori.

Il Nomad XR4000 ha 48 sensori sonar di prossimità, installati sul perimetro superiore ed inferiore, organizzati in sei gruppi da otto. Su ognuna delle tre porte, numerate in senso antiorario sono montati due set di sensori, si veda la figura 4.2, per un totale di sei *set.*

La struttura `SonarController` in `N_RobotState` è definita come:

```
struct N_SonarController
{
   N_CONST unsigned int SonarSetCount;
   struct N_SonarSet SonarSet[N_MAX_SONAR_SET_COUNT];
   BOOL SonarPaused;
};
```

★ `SonarSetCount`: rappresenta la quantità di sensori ad ultrasuono presenti nel *set.*

★ `N_SonarSet`: *array* di strutture contenenti informazioni sulle misurazioni dei sensori.

La struttura `N_SonarSet` è definita come:

```
struct N_SonarSet
{
```

```
    unsigned int FiringOrder[N_MAX_SONAR_COUNT + 1];
    long FiringDelay;
    long BlankingInterval;
    BOOL DataActive;
    BOOL TimeStampActive;
    N_CONST unsigned int SonarCount;
    struct N_Sonar Sonar[N_MAX_SONAR_COUNT];
};
```

Essa contiene un array `Sonar` di strutture sonar (una per ogni trasduttore del *set*), più altri parametri di configurazione.

★ `FiringDelay`: attesa, espressa in millisecondi, tra due attivazioni consecutive di un sensore sonar. Settare questo parametro a un valore alto previene fenomeni di eco multipli (si veda il paragrafo 2.5.1).

★ `Intervallo di` *`blanking`*: non implementato[28]

★ `DataActive`: viene impostato a `TRUE` se i dati del sensore sonar del *set* corrente sono stati aggiornati.

★ `TimeStampActive`: viene impostato a `TRUE` se il tempo[29] è stato aggiornato per il *set* corrente.

★ `SonarCount`: rappresenta la quantità di sonar nel *set* corrente.

★ `FiringOrder`: è un array di indici dei sonar, terminato da `N_END_SONAR_FIRING_ORDER`, se la lunghezza dell'array è inferiore a quella rappresentata da `SonarCount`[30].

La struttura `N_Sonar` contiene le misurazioni effettuate dal sensore:

[28] Intervallo di attesa, espresso in millisecondi, tra l'attivazione di un sensore sonar fino all'attivazione del sensore come ricevitore.

[29] Tempo in cui è avvenuta l'acquisizione del sensore sonar.

[30] Ogni indice rappresenta un sensore sonar del set.

```
struct N_Sonar
{
   long Reading;
   unsigned long TimeStamp;
};
```

I valori definiti per ogni sensore sonar sono:

★ `Reading` Misura di distanza effettuata dal sensore in millimetri. Quando il sensore non riceve eco (per esempio a causa di una eccessiva distanza dal bersaglio) , questo valore viene imposto a `N_SONAR_TIMEOUT`

★ `Timestamp` istante temporale di acquisizione della rilevazione

ESEMPI

```
#include <stdio.h>
#include "Nclient.h"

#define SCHEDULER_HOSTNAME "localhost"
#define SCHEDULER_PORT 7073
int main()
{
   struct N_RobotState *state;
   struct N_SonarSet *sonar_set;
   short i;

   N_InitializeClient(SCHEDULER_HOSTNAME, SCHEDULER_PORT);
   N_ConnectRobot(1);

   state = N_GetRobotState(1);
   sonar_set = &state->SonarController.SonarSet[0];
```

```
   N_GetSonarConfiguration(1);

   printf("Configuration of Sonar Set 0\n");
   printf("Data Active: %s Time Stamp Active: %s\n",
          sonar_set->DataActive ? "Yes" : "No",
          sonar_set->TimeStampActive ? "Yes" : "No");
   printf("Firing order: ");
   i = 0;
   while (sonar_set->FiringOrder[i]!=N_END_SONAR_FIRING_ORDER)
   {
      printf("%d ", sonar_set->FiringOrder[i]);
      i+=1;
      printf("\n");
   }
   N_DisconnectRobot(1);
   exit(0);
}
```

RIFERIMENTI

N_GetSonar, N_SetSonarConfiguration, N_GetState.

N_GETSTATE

NOME

N_GetState

SCOPO

Aggiornare i dati dei sensori del robot nella struttura `N_Robot_State`.

SINTASSI

```
int N_GetState(long RobotID);
```

ARGOMENTI

`long RobotID`: ID del robot.

VALORI RESTITUITI

`N_NO_ERROR`: operazione eseguita senza errori.
`N_ROBOT_NOT_FOUND`: non è stato trovato il robot.
`N_CONNECTION_FAILED`: il *socket* è stato chiuso dall'ultima chiamata da parte del programma *client.*

VARIABILI GLOBALI AGGIORNATE

`N_RobotState`

DESCRIZIONE

Chiamare la funzione `N_GetState` equivale a chiamare tutti i comandi *get,* eccetto `N_GetTimer`, che deve essere chiamato esplicitamente per essere aggiornato.

`N_RobotState` è una struttura nella quale sono contenuti i dati e le informazioni necessarie quando si stabilisce una comunicazione tra un robot (reale o simulato) ed un programma *client.* La struttura `N_RobotState` è definita nel file `Nclient.h` come segue:

```
struct N_RobotState
{
  N_CONST long RobotID;
  N_CONST char RobotType;
  struct N_Integrator Integrator;
  struct N_AxisSet AxisSet;
  struct N_LiftController LiftController;
  struct N_Joystick Joystick;
  struct N_SonarController SonarController;
  struct N_InfraredController InfraredController;
  struct N_BumperController BumperController;
  struct N_Compass Compass;
  struct N_LaserSet LaserSet;
  struct N_S550Set S550Set;
  struct N_BatterySet BatterySet;
  struct N_Timer Timer;
};
```

ESEMPI

`N_GetState.c`

```
#include <stdio.h>
#include "Nclient.h"

#define SCHEDULER_HOSTNAME "localhost"
#define SCHEDULER_PORT 7073

int main()
{
   struct N_RobotState *state;
   struct N_SonarSet *sonar_set;
   short i;

   N_InitializeClient(SCHEDULER_HOSTNAME, SCHEDULER_PORT);
   N_ConnectRobot(1);

   state = N_GetRobotState(1);
   N_GetState(1);
   printf("Got all robot data in local state structure\n");

   N_DisconnectRobot(1);
   exit(0);
}
```

RIFERIMENTI

Nessuno.

N_GETTIMER

NOME

N_GetTimer

SCOPO

Ottenere le impostazioni correnti del *timeout* e del valore del timer globale.

SINTASSI

```
int N_GetTimer(long RobotID);
```

ARGOMENTI

`long RobotID`: ID del robot.

VALORI RESTITUITI

`N_NO_ERROR`: operazione eseguita senza errori.
`N_ROBOT_NOT_FOUND`: non è stato trovato il robot.
`N_CONNECTION_FAILED`: il *socket* è stato chiuso dall'ultima chiamata da parte del programma *client.*

VARIABILI GLOBALI AGGIORNATE

`N_RobotState`

DESCRIZIONE

La funzione legge l'impostazione corrente del *timeout* del robot ed aggiorna la struttura `N_RobotState`.

```
struct N_Timer
{
  long Timeout;
  unsigned long Time;
};
```

Dove `Timeout` è la soglia del timer, dopo la quale i motori vengono spenti[31]. `Time` rappresenta l'ammontare di tempo da quando il robot è stato acceso (il timer globale). Entrambi i parametri sono espressi in millisecondi.

ESEMPI

`N_GetTimer.c`

```
#include <stdio.h>
#include "Nclient.h"

#define SCHEDULER_HOSTNAME "localhost"
#define SCHEDULER_PORT 7073

int main()
{
   struct N_RobotState *state;
   struct N_Timer *timer;
   short i;

   N_InitializeClient(SCHEDULER_HOSTNAME, SCHEDULER_PORT);
   N_ConnectRobot(1);

   state = N_GetRobotState(1);
   timer = &(state->Timer);
```

[31] Per motivi di sicurezza il parametro `Timeout` non può superare il valore 1500 (1,5 secondi).

```
    N_GetTimer(1);

    printf("Timeout is currently set to %d seconds\n",
           timer->Timeout);

    N_DisconnectRobot(1);
    exit(0);
}
```

RIFERIMENTI

Nessuno.

N_INITIALIZECLIENT

NOME

N_InitializeClient

SCOPO

Inizializzare la comunicazione con lo *scheduler.*

SINTASSI

```
int N_InitializeClient(const char *scheduler_hostname,
                       unsigned short scheduler_socket);
```

ARGOMENTI

`const char *scheduler_hostname`: *hostname* o indirizzo IP.
`unsigned short scheduler_socket`: numero del *socket.*

VALORI RESTITUITI

Nessuno.

VARIABILI GLOBALI AGGIORNATE

Nessuna.

DESCRIZIONE

Questa funzione inizializza la libreria *client* e specifica l' *hostname* e la porta dello *scheduler.* Uno *scheduler* è necessario se sono presenti uno o più robot (reali o simulati) nell'applicazione. Quando è presente un solo robot la

funzione N_InitializeClient deve essere chiamata specificando l' *hostname* ed il *socket* reali del robot.

ESEMPI

N_InitializeClient.c

```
#include <stdio.h>
#include "Nclient.h"

#define SCHEDULER_HOSTNAME "localhost"
#define SCHEDULER_PORT 7073

int main()
{
   printf("Initializing the client with scheduler on machine %s,
          port %d\n", SCHEDULER_HOSTNAME, SCHEDULER_PORT);

   N_InitializeClient(SCHEDULER_HOSTNAME, SCHEDULER_PORT);
   N_ConnectRobot(1);

   /* Something useful... */

   N_DisconnectRobot(1);
   exit(0);
}
```

RIFERIMENTI

Nessuno.

N_SETAXES

NOME

N_SetAxes

SCOPO

Muovere il robot basandosi sui parametri assiali nella struttura `N_RobotState`.

SINTASSI

```
int N_InitializeClient(const char *scheduler_hostname,
                       unsigned short scheduler_socket);
```

ARGOMENTI

`long RobotID`: ID del robot.

VALORI RESTITUITI

`N_NO_ERROR`: operazione eseguita senza errori.
`N_INVALID_ARGUMENT`: argomento non valido.
`N_ROBOT_NOT_FOUND`: non è stato trovato il robot.
`N_CONNECTION_FAILED`: il *socket* è stato chiuso dall'ultima chiamata da parte del programma *client.*
`N_UNKNOWN_ERROR`: il demone *DRobot* non riesce a comunicare con il motore.
`N_AXES_NOT_READY`: gli assi non sono liberi di muoversi. Controllare il campo *Status.*

VARIABILI GLOBALI AGGIORNATE

`N_RobotState`

DESCRIZIONE

Questa funzione aggiorna gli assi del motore in modo da rispettare i parametri correnti degli assi contenuti nella struttura `N_AxisSet` di `N_RobotState` e causa il movimento desiderato.

```
struct N_AxisSet
{
  BOOL Global;
  unsigned char Status;
  N_CONST unsigned int AxisCount;
  struct N_Axis Axis[N_MAX_AXIS_COUNT];
};
```

★ `Global`: quando è impostata a `TRUE`, la base del robot entra in modalità *Global,* mentre se è impostata a `FALSE`, la base del robot entra in modalità *Joint.*

★ `Status`: può assumere uno dei seguenti valori:

`N_AXES_READY`: indica che gli assi sono disponibili per il movimento.

`N_JOYSTICK_IN_USE`: indica che la base è al momento controllata via joystick.

`N_ESTOP_DOWN`: indica che uno o più pulsanti di emergenza sono premuti.

`N_MOTION_ERROR`: indica che la base del robot non può eseguire il movimento.

La struttura `N_Axis` contiene informazioni per ogni singolo asse:

```
struct N_Axis
{
  BOOL DataActive;
  BOOL TimeStampActive;
  BOOL Update;
```

```
  unsigned long TimeStamp;
  char Mode;
  long DesideredPosition;
  long DesiredSpeed;
  long Acceleration;
  long TrajectoryPosition;
  long TrajectoryVelocity;
  long ActualPosition;
  long ActualVelocity;
  BOOL InProgress;
};
```

★ `DataActive`: il valore `TRUE` per questo parametro implica che i valori nella struttura saranno aggiornati.

★ `TimeStampActive`: il valore `TRUE` per questo parametro implica che sarà aggiornato il valore di `TimeStamp`.

★ `Update`: il valore `TRUE` per questo parametro implica che i valori `DesiredSpeed`, `DesiredPosition` ed `Acceleration` saranno caricati, quando verrà eseguita la funzione `N_SetAxes`.

★ `TimeStamp`: il valore del tempo (in millisecondi) al quale sono stati misurati i valori.

★ `Mode`: può essere una dei seguenti:

`N_AXIS_POSITION_RELATIVE`: specifica che l'asse si muove relativamente alla posizione corrente.

`N_AXIS_POSITION_ABSOLUTE`: specifica che l'asse si muove verso una posizione rispetto alla posizione del riferimento assoluto.

`N_AXIS_VELOCITY`: specifica che l'asse si muove con una velocità costante.

`N_AXIS_STOP`: ferma l'asse.

★ `DesiredPosition`: specifica la posizione finale desiderata dell'asse. Le unità sono in millimetri per le traslazioni ed in milliradianti per le rotazioni.

Quando il campo Mode è settato a N_AXIS_POSITION_RELATIVE oppure a N_AXIS_POSITION_ABSOLUTE, questo specifica rispettivamente la posizione finale relativa alla posizione corrente o a quella assoluta. Questo parametro non viene utilizzato quando Mode è settato a N_AXIS_VELOCITY oppure a N_AXIS_STOP. Se è selezionata la modalità *Global* (*Global* =TRUE), il parametro DesiredPosition è definito rispetto al sistema di riferimento globale. Se invece è selezionata la modalità *Joint* (*Global* =FALSE), il parametro DesiredPosition è definito rispetto alle coordinate di giunto.

★ DesiredSpeed: specifica la velocità alla quale deve essere effettuato il movimento. Questo parametro non è utilizzato quando il campo Mode è settato a N_AXIS_STOP.

★ Acceleration: specifica l'accelerazione relativa al parametro DesiredSpeed oppure le accelerazioni successive se il valore di DesiredSpeed aumenta durante un movimento. Questo parametro specifica anche la decelerazione, con le stesse modalità dell'accelerazione. Le unità sono in millimetri/secondo, per le traslazioni ed in milliradianti/secondo per le rotazioni.

★ TrajectoryPosition: fornisce la posizione corrente del generatore di traiettoria. Se *Global* =TRUE (condizione di *Global mode*), il valore TrajectoryPosition è definito rispetto al sistema di riferimento fisso. Se *Global* =FALSE (condizione di *Joint mode*), il valore TrajectoryPosition è definito rispetto al sistema di coordinate di giunto.

★ TrajectoryVelocity: fornisce la velocità corrente del generatore di traiettoria. Se *Global* =TRUE (condizione di *Global mode*), il valore TrajectoryVelocity è definito rispetto al sistema di riferimento fisso. Se *Global* =FALSE (condizione di *Joint mode*), il valore TrajectoryVelocity è definito rispetto al sistema di coordinate di giunto.

★ `ActualPosition`: fornisce la posizione attuale dell'asse. Il valore di questo campo è basato sul settaggio del campo *Global*. Se *Global* =`TRUE` (condizione di *Global mode*), questo campo fornisce la posizione dell'asse rispetto al sistema di riferimento fisso (questo stesso valore può essere trovato nella struttura `N_Integrator`). Se *Global* =`FALSE` (condizione di *Joint mode*), questo campo fornisce la posizione dell'asse rispetto al sistema di coordinate di giunto.

★ `ActualVelocity`: fornisce la velocità attuale dell'asse. Il valore di questo campo è basato sul settaggio del campo *Global*. Se *Global* =`TRUE` (condizione di *Global mode*), questo campo fornisce la posizione dell'asse rispetto al sistema di riferimento fisso. Se *Global* =`FALSE` (condizione di *Joint mode*), questo campo fornisce la posizione dell'asse rispetto al sistema di coordinate di giunto. Questo valore, come `ActualPosition` è basato sull'impostazione del campo *Global* in `N_AxisSet`. Se il campo *Global* è selezionato, questo campo fornisce un valore rispetto al sistema di riferimento fisso.

★ `InProgress`: fornisce un valore booleano che informa l'utente che un asse è in movimento.

Nella tabella 4.2 sono rappresentati l'interpretazione ed i *range* per gli assi del Nomad XR4000.

ESEMPI

`N_GetAxes.c`

```
#include <stdio.h>
#include "Nclient.h"

#define SCHEDULER_HOSTNAME "localhost"
#define SCHEDULER_PORT 7073
```

Assi	Descrizione
Asse 0	Asse $\mathcal{X}$
Asse 1	Asse $\mathcal{Y}$
Asse 2	Rotazione θ
Range di velocità per l'asse 0	[-1500mm/s, 1500mm/s]
Range di velocità per l'asse 1	[-1500mm/s, 1500mm/s]
Range di velocità per l'asse 2	[-5000mRad/s, 5000mRad/s]
Range di accelerazione per l'asse 0	[0mm/s^2, 1500mm/s^2]
Range di accelerazione per l'asse 1	[0mm/s^2, 1500mm/s^2]
Range di accelerazione per l'asse 2	[0mRad/s^2, 5000mRad/s^2]

Tabella 4.2: L'interpretazione degli assi ed i *range* del Nomad XR4000.

```
int main()
{
   struct N_RobotState *state;
   struct N_Axis *axis;
   struct N_AxisSet *axisSet;
   int inProgress;

   N_InitializeClient(SCHEDULER_HOSTNAME, SCHEDULER_PORT);
   N_ConnectRobot(1);

   state = N_GetRobotState(1);
   axisSet=&(state->N_AxisSet);
   axis = &(AxisSet->Axis[N_YTRANSLATION]);
   axisSet->Global=FALSE; /* local axis mode */
   axis->DataActive=TRUE;
   axis->Mode=N_AXIS_POSITION_RELATIVE;
   axis->DesiredSpeed=500; /* mm/s */
   axis->acceleration=500; /* mm/s/s */
```

```
   axis->DesiredPosition=1000; /* mm */
   axis->Update=TRUE;
   N_SetAxes(1);

   /* move 1 meter forward and stop */
   do
   {
      N_GetAxes(1);
      inProgress=axis->InProgress;
   }
   while(inProgress);
   N_DisconnectRobot(1);
   exit(0);
}
```

RIFERIMENTI

N_GetAxes.

N_SETINTEGRATEDCONFIGURATION

NOME

N_SetIntegratedConfiguration

SCOPO

Impostare i valori della configurazione integrata nella struttura `N_RobotState`.

SINTASSI

```
int N_SetIntegratedConfiguration(long RobotID);
```

ARGOMENTI

`long RobotID`: ID del robot.

VALORI RESTITUITI

`N_NO_ERROR`: operazione eseguita senza errori.
`N_ROBOT_NOT_FOUND`: non è stato trovato il robot.
`N_CONNECTION_FAILED`: il *socket* è stato chiuso dall'ultima chiamata da parte del programma *client.*
`N_UNKNOWN_ERROR`: il demone *DRobot* non riesce a comunicare con il dispositivo.

VARIABILI GLOBALI AGGIORNATE

`N_RobotState`.

DESCRIZIONE

Questa funzione imposta i valori della configurazione integrata del robot con i valori correnti del campo `N_Integrator` di `N_RobotState`.

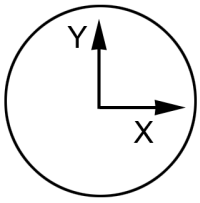


Figura 4.4: Configurazione geometrica del nomad XR4000

La struttura `N_Integrator` contiene la configurazione geometrica del robot rispetto al sistema di riferimento globale ed è definita come segue:

```
struct N_Integrator
{
  BOOL DataActive;
  BOOL TimeStampActive;
  long x;
  long y;
  long Rotation;
  unsigned long TimeStamp;
};
```

★ Il valore di `DataActive` è impostato a `TRUE` se i campi in questa struttura devono essere aggiornati.

★ Il valore di `TimeStampActive` è impostato a `TRUE` se il campo `TimeStamp` deve essere aggiornato.

★ Le coordinate `x` ed `y` rappresentano le coordinate integrate del punto di riferimento del robot nel sistema di riferimento fisso. I valori delle coordinate

sono espressi in millimetri. Il sistema di riferimento è allineato alla posizione del robot all'avvio.

★ Il parametro `Rotation` è l'angolo, misurato in milliradianti del corpo del robot rispetto al sistema di riferimento fisso.

★ Il parametro `TimeStamp` rappresenta il tempo al quale sono stati computati i valori integrati. È espresso in millisecondi.

ESEMPI

`N_SetIntegratedConfiguration.c`

```
#include <stdio.h>
#include "Nclient.h"

#define SCHEDULER_HOSTNAME "localhost"
#define SCHEDULER_PORT 7073

int main()
{
   struct N_RobotState *state;
   struct N_Integrator *configuration;
   N_InitializeClient(SCHEDULER_HOSTNAME, SCHEDULER_PORT);
   N_ConnectRobot(1);

   state = N_GetRobotState(1);
   configuration =&(state->Integrator);

   configuration->x = 0;
   configuration->y = 0;
   configuration->z = 0;
   configuration->Rotation = 0;
```

```
   N_SetIntegratedConfiguration(1);

   printf("Integrated Configuration set to: x %d y %d z %d
          Rotation %d\n",
          configuration->x,
          configuration->y,
          configuration->z,
          configuration->Rotation);

   N_DisconnectRobot(1);
   exit(0);
}
```

RIFERIMENTI

N_GetIntegratedConfiguration.

N_SETJOYSTICK

NOME

N_SetJoystick

SCOPO

Muovere il robot in base ai parametri nella struttura `N_RobotState`.

SINTASSI

```
int N_SetJoystick(long RobotID);
```

ARGOMENTI

`long RobotID`: ID del robot.

VALORI RESTITUITI

`N_NO_ERROR`: operazione eseguita senza errori.
`N_ROBOT_NOT_FOUND`: non è stato trovato il robot.
`N_CONNECTION_FAILED`: il *socket* è stato chiuso dall'ultima chiamata da parte del programma *client.*
`N_INVALID_ARGUMENT`: un valore nella struttura `N_Joystick` è non valido.
`N_UNKNOWN_ERROR`: il demone *DRobot* non riesce a comunicare con il dispositivo.

VARIABILI GLOBALI AGGIORNATE

Nessuna.

DESCRIZIONE

Questa funzione manda i valori nella sottostruttura `N_Joystick` della struttura di stato al robot, per far muovere il robot come se l'utente stesse utilizzando un joystick. La struttura `N_Joystick` ha i campi per i due assi del joystick (e per un terzo eventuale asse) e per tre pulsanti.

```
struct N_Joystick
{
   double X;
   double Y;
   double Theta;
   BOOL ButtonA;
   BOOL ButtonB;
   BOOL ButtonC;
};
```

★ `X`, `Y` e `Theta` rappresentano i valori degli assi e devono essere compresi in $[-1.0, 1.0]$.

★ I tre parametri `Button` sono impostati a `TRUE` per indicare che il pulsante è premuto. Se nessuno di questi è impostato a `TRUE`, questo indica che il joystick è stato rilasciato.

RIFERIMENTI

Nessuno.

N_SETSONARCONFIGURATION

NOME

N_SetSonarConfiguration

SCOPO

Impostare la configurazione dei sensori sonar di prossimità.

SINTASSI

```
int N_SetSonarConfiguration(long RobotID);
```

ARGOMENTI

`long RobotID`: ID del robot.

VALORI RESTITUITI

`N_NO_ERROR`: operazione eseguita senza errori.
`N_NO_ROBOT_FOUND`: non è stato trovato il robot.
`N_CONNECTION_FAILED`: il *socket* è stato chiuso dall'ultima chiamata da parte del programma *client.*

VARIABILI GLOBALI AGGIORNATE

`N_RobotState`.

DESCRIZIONE

Questa funzione configura i valori correnti dei sensori sonar di prossimità nella struttura `N_RobotState`.

La struttura del *controller* sonar racchiude tutti i dati validi per tutti i *set* di sonar del robot. Questi *set* sono gruppi di sonar che rilevano contemporaneamente; per esempio, l'ordine di attivazione può essere definito tra sonar dello stesso *set*. La struttura del *controller* sonar contiene un *array* di strutture `N_SonarSet`, ognuna che descrive un particolare *set* di sensori.

Il Nomad XR4000 ha 48 sensori sonar di prossimità, installati sul perimetro superiore ed inferiore, organizzati in sei gruppi da otto. Su ognuna delle tre porte, numerate in senso antiorario, sono montati due set di sensori, si veda la figura 4.2, per un totale di sei *set*.

La struttura `SonarController` in `N_RobotState` è definita come:

```
struct N_SonarController
{
   N_CONST unsigned int SonarSetCount;
   struct N_SonarSet SonarSet[N_MAX_SONAR_SET_COUNT];
   BOOL SonarPaused;
};
```

★ `SonarSetCount`: rappresenta la quantità di sensori ad ultrasuono presenti nel *set.*

★ `N_SonarSet`: *array* di strutture contenenti informazioni sulle misurazioni dei sensori.

La struttura `N_SonarSet` è definita come:

```
struct N_SonarSet
{
```

```
    unsigned int FiringOrder[N_MAX_SONAR_COUNT + 1];
    long FiringDelay;
    long BlankingInterval;
    BOOL DataActive;
    BOOL TimeStampActive;
    N_CONST unsigned int SonarCount;
    struct N_Sonar Sonar[N_MAX_SONAR_COUNT];
};
```

Essa contiene un array `Sonar` di strutture sonar (una per ogni trasduttore del *set*), più altri parametri di configurazione.

★ `FiringDelay`: attesa, espressa in millisecondi, tra due attivazioni consecutive di un sensore sonar. Settare questo parametro a un valore alto previene fenomeni di eco multipli (si veda il paragrafo 2.5.1).

★ `Intervallo di` *`blanking`*: non implementato[32]

★ `DataActive`: viene impostato a `TRUE` se i dati del sensore sonar del *set* corrente sono stati aggiornati.

★ `TimeStampActive`: viene impostato a `TRUE` se il tempo[33] è stato aggiornato per il *set* corrente.

★ `SonarCount`: rappresenta la quantità di sonar nel *set* corrente

★ `FiringOrder`: è un array di indici dei sonar, terminato da `N_END_SONAR_FIRING_ORDER`, se la lunghezza dell'array è inferiore a quella rappresentata da `SonarCount`[34].

La struttura `N_Sonar` contiene le misurazioni effettuate dal sensore:

[32] Intervallo di attesa, espresso in millisecondi, tra l'attivazione di un sensore sonar fino all'attivazione del sensore come ricevitore.

[33] Tempo in cui è avvenuta l'acquisizione del sensore sonar.

[34] Ogni indice rappresenta un sensore sonar del set.

```
struct N_Sonar
{
   long Reading;
   unsigned long TimeStamp;
};
```

I valori definiti per ogni sensore sonar sono:

★ `Reading` Misura di distanza effettuata dal sensore in millimetri. Quando il sensore non riceve eco (per esempio a causa di una eccessiva distanza dal bersaglio) , questo valore viene imposto a `N_SONAR_TIMEOUT`

★ `Timestamp` istante temporale di acquisizione della rilevazione

ESEMPI

`N_SetSonarConfiguration.c`

```
#include <stdio.h>
#include "Nclient.h"

#define SCHEDULER_HOSTNAME "localhost"
#define SCHEDULER_PORT 7073

int main()
{
   struct N_RobotState *state;
   struct N_SonarSet *sonar_set;
   short i;

   N_InitializeClient(SCHEDULER_HOSTNAME, SCHEDULER_PORT);
   N_ConnectRobot(1);
```

```
   state = N_GetRobotState(1);
   sonar_set = &state->SonarController.SonarSet[0];
   /* Always do this to initialize the local state structure
    * with correct data */

   N_GetSonarConfiguration(1);

   sonar_set->DataActive = TRUE;
   sonar_set->TimeStampActive = TRUE;

   sonar_set->FiringOrder[0] = 0;
   sonar_set->FiringOrder[1] = 2;
   sonar_set->FiringOrder[2] = 4;
   sonar_set->FiringOrder[3] = N_END_SONAR_FIRING_ORDER;
   N_SetSonarConfiguration(1);

   printf("Sonar Set 0 set to\n");
   printf("Data Active: %s Time Stamp Active: %s\n",
          sonar_set->DataActive ? "Yes" : "No",
          sonar_set->TimeStampActive ? "Yes" : "No");
   printf("Firing order: ");
   i = 0;
   while (sonar_set->FiringOrder[i]!=N_END_SONAR_FIRING_ORDER)
   {
      printf("%d ", sonar_set->FiringOrder[i]);
      i+=1;
   }
   N_DisconnectRobot(1);
   exit(0)
}
```

RIFERIMENTI

`N_GetSonarConfiguration`.

N_SETTIMER

NOME

N_SetTimer

SCOPO

Impostare il *timeout* del robot.

SINTASSI

```
int N_SetTimer(long RobotID);
```

ARGOMENTI

`long RobotID`: ID del robot.

VALORI RESTITUITI

`N_NO_ERROR`: operazione eseguita senza errori.
`N_ROBOT_NOT_FOUND`: non è stato trovato il robot.

VARIABILI GLOBALI AGGIORNATE

`N_RobotState`.

DESCRIZIONE

Questa funzione imposta il *timeout* del robot, in modo tale che se il robot non riceve un comando per più di un periodo di *timeout,* cessa il movimento. Questa funzione rappresenta una misura di sicurezza per prevenire che il robot

continui il suo movimento senza controllo se per qualche ragione (problemi di comunicazione, *crash* del sistema, etc.), il robot non riceve alcun comando.

```
struct N_Timer
{
   long Timeout;
   long Time;
};
```

`Timeout` è la soglia del timer, dopo la quale i motori vengono spenti[35].
`Time` rappresenta l'ammontare di tempo da quando il robot è stato acceso (il timer globale).
Entrambi i parametri sono espressi in millisecondi.

ESEMPI

`N_SetTimeout.c`

```
#include <stdio.h>
#include "Nclient.h"

#define SCHEDULER_HOSTNAME "localhost"
#define SCHEDULER_PORT 7073

int main()
{
   struct N_RobotState *state;
   struct N_Timer *timer;
   short i;

   N_InitializeClient(SCHEDULER_HOSTNAME, SCHEDULER_PORT);
```

[35] Per motivi di sicurezza il parametro `Timeout` non può superare il valore 1500 (1,5 secondi).

```
    N_ConnectRobot(1);

    state = N_GetRobotState(1);
    timer = &(state->Timer);

    timer->Timeout = 10000;

    N_SetTimer(1);

    printf("Timeout set to %d milliseconds\n", timer->Timeout);

    N_DisconnectRobot(1);
    exit(0);
}
```

RIFERIMENTI

N_GetTimer.

N_SPEAK

NOME

N_Speak

SCOPO

Inviare alla scheda di sintesi vocale *DoubleTalk* una stringa di testo da pronunciare.

SINTASSI

```
int N_Speak(long RobotID, char *Text);
```

ARGOMENTI

`long RobotID`: ID del robot.
`char *Text`: la stringa di testo che il robot pronuncerà.

VALORI RESTITUITI

`N_NO_ERROR`: operazione eseguita senza errori.
`N_ROBOT_NOT_FOUND`: non è stato trovato il robot.
`N_CONNECTION_FAILED`: il *socket* è stato chiuso dall'ultima chiamata da parte del programma *client.*

VARIABILI GLOBALI AGGIORNATE

Nessuna.

DESCRIZIONE

Questa funzione invia una stringa di testo alla scheda di sintesi vocale *DoubleTalk*, per far parlare il robot. Per i caratteri speciali di controllo si rimanda alla guida della scheda *DoubleTalk*.

ESEMPI

N_Speak.c

```
#include <stdio.h>
#include "Nclient.h"

#define SCHEDULER_HOSTNAME "localhost"
#define SCHEDULER_PORT 7073

int main()
{
   struct N_RobotState *state;
   struct N_Timer *timer;
   short i;

   N_InitializeClient(SCHEDULER_HOSTNAME, SCHEDULER_PORT);
   N_ConnectRobot(1);

   N_Speak(1, "Hello World");

   N_DisconnectRobot(1);
   exit(0);
}
```

RIFERIMENTI

Nessuno.

Appendice A

La rete N.I.N.E.

Gli autori hanno appositamente sviluppato un *network* tra i robot Nomad per estendere le funzionalità e gli studi possibili sulla robotica mobile, espandendoli nell'ambito delle ricerche sulla robotica collaborativa e multi-robot. La rete N.I.N.E., Nomadic Integrated Network Environment, è una rete costituita da due robot della famiglia Nomad (Nomad N150 e Nomad XR4000) e un computer che agisce da server. Lo scopo di questa rete è quello di rendere possibile una comunicazione tra i due robot, in modo tale che possano scambiarsi dati ed inoltre grazie alla presenza del server che completa la configurazione della rete, diventa possibile controllare e ricevere dati sulla rete Internet, cioè da qualunque parte del mondo.

L'aspetto della sicurezza della rete è stato uno degli obiettivi principali durante la costituzione di questa rete. I due robot sono tra loro collegati attraverso una rete WLan (Radio Ethernet), che li collega anche con il server. Come piattaforma di *gateway,* esso possiede una seconda scheda di rete che lo collega ad Internet[1]. Per accedere alla rete interna dei robot viene utilizzato il servizio *OpenVPN* con cifratura delle chiavi a 2048 bit, rendendo molto sicuro l'accesso alla rete interna.

I computer della rete N.I.N.E. utilizzano il sistema operativo Linux Red Hat 7.3.

[1]Su IP pubblico.

A.1 Struttura della rete N.I.N.E.

Il concetto di base della rete N.I.N.E. è avere due robot mobili computazionalmente indipendenti. Il Nomad N150[2] non è provvisto di un'unità di elaborazione propria, esso viene comandato attraverso una porta seriale. La soluzione ideata è stata di utilizzare un laptop in modo tale da rendere il Nomad N150 un'unità mobile indipendente capace di eseguire i compiti di robotica mobile (visione robotica, *obstacle avoidance*). Diversamente il Nomad XR4000 è di per sé computazionalmente autonomo; in questo modo la rete è composta da due robot mobili completamente indipendenti ed intercomunicanti.

La rete N.I.N.E. è composta da cinque elementi ethernet, riportati in tabella A.1.

Nome	Interfaccia	IP
Server N.I.N.E.	*internet access*	10.10.10.1
Nomad XR4000	Proxim RangeLan2 7100 (Master)	10.10.10.2
Nomad N150	Nomadic Mercury EN (Station)	10.10.10.4
Server N.I.N.E.	Nomadic Mercury EN (Station)	10.10.10.5
Laptop su N150	Interfaccia di rete	10.10.10.6

Tabella A.1: Dispositivi della rete N.I.N.E. e relativi indirizzi IP.

I dispositivi *wireless* che collegano tra loro i due robot ed il server sono dei dispositivi radio-ethernet che devono essere configurati in modo appropriato, in modo da rendere possibile una comunicazione robusta a delle distanze sufficienti per le tipiche applicazioni di robotica mobile. È necessario impostare un elemento tra i radio-ethernet che abbia la funzione di *Master,* cioè il controllore della comunicazione tra tutte le radiolan presenti nella rete senza fili. Il *Master* nella rete N.I.N.E. è costituito dalla scheda Proxim RangeLan2 7100 alloggiata nell'XR4000. Questa scelta è stata dettata da fattori di stabilità di comunicazione e di compatibilità con i Mercury EN presenti sulla rete N.I.N.E.. Tutti gli altri

[2]Si rimanda il lettore interessato al manuale *Nomad 150: manuale d'uso* di F. Romanelli.

dispositivi presenti nella rete sono configurati come *Station peer-to-peer,* cioè dispositivi che comunicano tra loro direttamente, senza passare dati al *Master.* Questo diminuisce la latenza di comunicazione quando le distanze tra i robot e il server sono considerevoli, guadagnando in affidabilità. Inoltre la comunicazione tra gli elementi della rete N.I.N.E. è completamente trasparente nel senso che tutti gli indirizzi IP sono raggiungibili all'interno della rete da qualsiasi robot o computer. Inoltre dopo aver effettuato l'identificazione (con chiave criptata) su *OpenVPN* il computer *client* che accede dall'esterno è trattato dalla rete come un elemento appartenente alla rete stessa. Uno schema della rete N.I.N.E. è riportato in figura A.1.

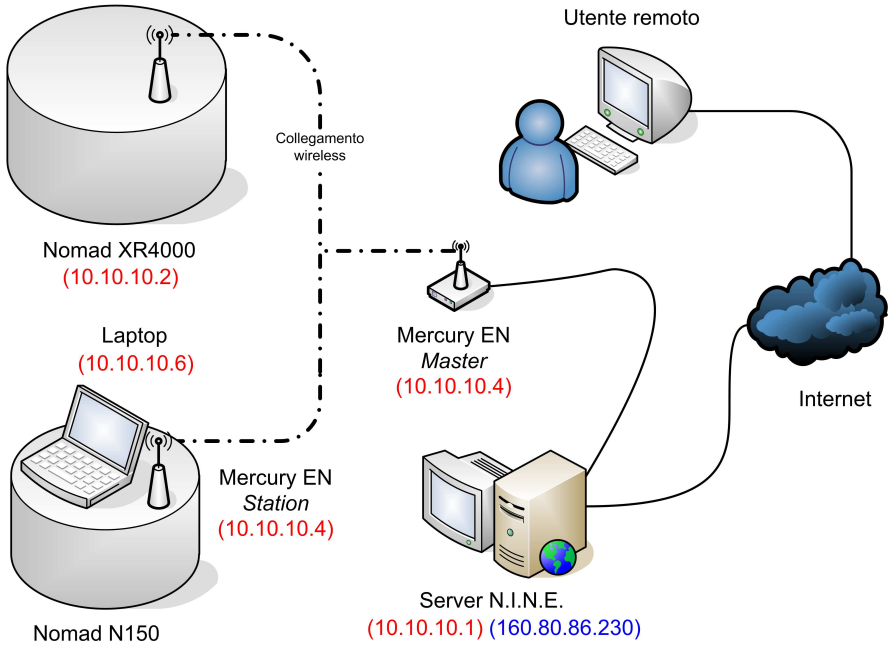


Figura A.1: Schema della rete N.I.N.E.

Il server N.I.N.E. ha due schede di rete, una che interfaccia il computer con la rete Internet e l'altra che collega il server al Nomadic Mercury EN con un cavo RJ45. Nel Nomad XR4000 è installata una scheda di rete wireless Proxim RangeLan2, su uno slot ISA; l'antenna, collegata alla scheda, è stata posizionata all'esterno del robot. Il Nomad N150 ha al suo interno un Mercury EN e una

connessione di tipo seriale. Il laptop sul Nomad N150 è collegato al Mercury EN con un cavo RJ45 ed è collegato con il robot con un cavo seriale.

A.2 Il protocollo *OpenVPN*

Il protocollo *OpenVPN* è un protocollo di *tunneling*[3] sicuro e opensource. Permette di accedere con un livello di sicurezza molto alto a reti private anche protette da *firewall*. Il principio di base dell'OpenVPN è quello di creare una rete virtuale tra il computer *server* e il computer *client*; vengono installati infatti dei dispositivi di rete virtuali, con indirizzi IP privati, su cui viaggiano i dati criptati tra i due punti estremi del tunnel.

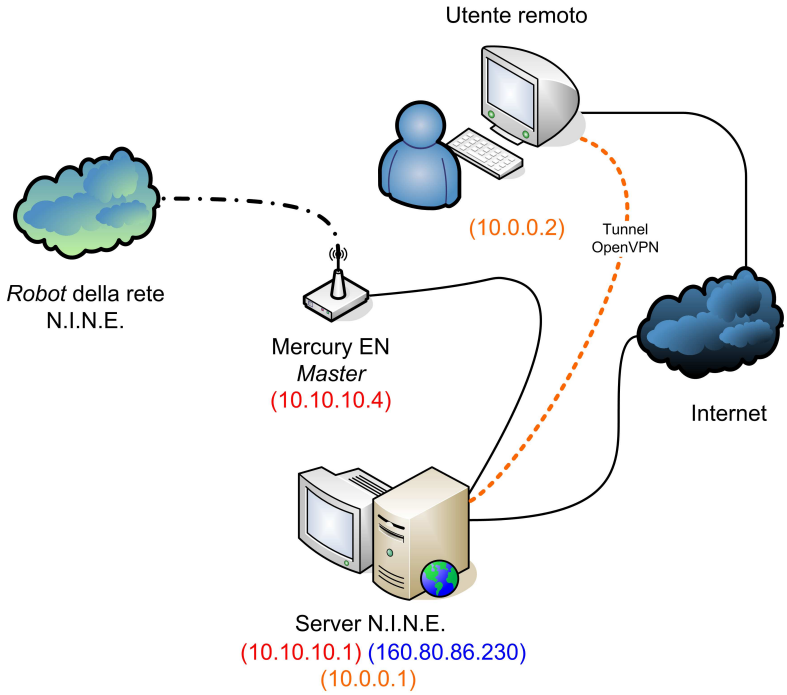


Figura A.2: Schema della configurazione con OpenVPN

[3]Particolare tecnica che consente di creare un canale privato e sicuro tra due punti di una rete.

A.2.1 Server OpenVPN

Per il server i requisiti di installazione sono che nel sistema sia presente il modulo `tun.o`, sia installato OpenSSH ed il sistema di compressione LZO. Il modulo `tun.o` è già presente nel sistema Red Hat 7.3, nel caso non lo sia compilare il kernel con tale modulo; anche OpenSSH è solitamente installato di *default,* altrimenti procedere all'installazione dopo averlo scaricato. In figura A.2 è rappresentato uno schema della configurazione di rete OpenVPN.

Compressione LZO

La compressione LZO è una libreria di compressione *real-time* di eccezionale velocità che permette di diminuire la quantità di dati trasmessi tra i due punti del *tunnel.*
Decomprimere il file in una *directory* ed eseguire in essa:

```
./configure
make
make check
make test
make install
```

OpenVPN 1.6

Se la procedura di installazione della libreria LZO viene eseguita senza errore eseguire l'installazione di *OpenVPN* 1.6. dopo aver decompresso il pacchetto tarball in una directory qualsiasi:

```
./configure
make
make install
```

È necessario quindi generare le chiavi statiche di accesso a $2048bit$ con il comando:
`openvpn -genkey -secret nine_network.key`

La chiave generata dovrà essere distribuita su un canale sicuro ai client che vogliono poter accedere al N.I.N.E. network. Creare la directory `/etc/openvpn`. È sufficiente quindi copiare il file di configurazione del firewall `nine_server.sh`, il file di configurazione OpenVPN `nine_server.conf` (ambedue forniti con questa documentazione) e il file di chiavi appena generato `nine_network.key` in `/etc/openvpn`. Copiare il file di configurazione del servizio openvpn in `/etc/init.d/openvpn`.

Installare il servizio eseguendo:
`chkconfig -add openvpn`

Il tunnel è adesso pronto all'uso.

La configurazione fornita installa il servizio `OpenVPN` sulla porta 5000 e attende una connessione da parte di qualunque ip. l'indirizzo IP del tunnel del client remoto deve essere configurato come 10.0.0.2[4].

Per configurazione avanzata consultare `vpn-howto.pdf` fornito insieme a questo manuale.

A.2.2 Client OpenVPN

Il client *OpenVPN* fornito è valido per le piattaforme windows (95/98/ME/2000/XP). Eseguire l'intuitiva procedura di installazione ed eventualmente aggiornare il file delle chiavi.

Dopo aver eseguito la corretta procedura di installazione riavviare il pc. Accendere i robot, verificare che il server abbia il servizio *OpenVPN* attivo, avviare il client *OpenVPN* e attendere il momento che nella sua finestra di stato compaia l'avviso di corretta connessione. Testare il corretto funzionamento del tunnel

[4] File di configurazione nine_server.conf.

eseguendo i comandi da una finestra *console*:

`ping 10.0.0.1` per verificare la connessione con il server.

`ping 10.10.10.2` per verificare la connessione con il robot XR4000.

Appendice B

Radio ethernet

In questa appendice verranno descritte le modalità di configurazione dei dispositivi radio ethernet. La rete N.I.N.E. annovera due tipologie di dispositivi *wireless:*

- Nomadic Mercury EN;
- Proxim RangeLan2.

Entrambi i dispositivi devono essere configurati opportunamente per operare in una rete *wireless* [1]. Il Nomadic Mercury EN è principalmente utilizzato come radio-modem, ossia sostituisce il collegamento seriale con un collegamento senza fili[2]. Il Proxim RangeLan2, invece è un dispositivo di rete *wireless ad hoc*.

[1] Essendo compatibili tra di loro, i due differenti dispositivi possono operare nella stessa rete.

[2] Originariamente il Mercury EN è stato configurato come radio-modem per comunicare con il Nomad N150 via seriale; nella configurazione della rete N.I.N.E., invece questo dispositivo si comporta da adattatore di rete *wireless* vero e proprio.

B.1 Configurazione Nomadic Mercury EN

Per configurare il Nomadic Mercury EN è necessario connettersi al dispositivo via seriale[3] o via ethernet[4]. Il collegamento via seriale è indispensabile se non è noto a priori l'indirizzo IP del dispositivo[5]. Per effettuare questo tipo di collegamento è necessario un software di emulazione terminale (è vivamente sconsigliato dagli autori di utilizzare come software *HyperTerminal* incluso in Microsoft Windows 95/98/ME/2000/XP). Dopo aver effettuato il collegamento via RS232 tra il dispositivo ed il computer (lasciando il Mercury EN non alimentato), occorre avviare il programma di emulazione terminale ed impostare la comunicazione seriale a 9600 `baud`, 8 `data bits`, `no parity` e 1 `stop bit`. Dopo aver effettuato il collegamento occorre alimentare l'unità. Successivamente è necessario premere il pulsante *configure,* situato sulla parte superiore dell'unità, aiutandosi con l'estremità di una graffetta. Dopo aver effettuato questa operazione, sullo schermo dovrebbe apparire il menu di configurazione dell'unità Mercury EN. Per connettersi al Mercury EN, conoscendo il suo indirizzo IP, è sufficiente collegare il modem con il cavo di rete RJ45 *crossed* ed utilizzare un software che realizzi una connessione *Telnet* con il dispositivo. È consigliato utilizzare il metodo di connessione via cavo ethernet, perché risulta più affidabile.

Configurare entrambi i Mercury EN come *station*, abilitare il *peer-to-peer*, impostare il relativo ip (si fa riferimento alla tabella A.1), e impostare come gateway la macchina su cui è installato il server *OpenVPN*. Nel menu `uart` impostare la velocità della porta a *38400baud*, protocollo di comunicazione `passthrough2` e configurare correttamente i menu `tcpbind2a` e `tcpbind2b` con i relativi IP dei due Mercury EN.

Per una lista delle impostazioni di configurazione dell'unità Mercury EN, si rimanda al manuale *Mercury - User's Guide (Version 2.0).* presente nel CD allegato.

[3] Per mezzo di un cavo seriale (cavo Null Modem), maschio-femmina.

[4] Per mezzo di un cavo di rete *crossed.*

[5] Per gli indirizzi IP dei dispositivi *wireless* si rimanda alla tabella A.1 dell'appendice A.

B.2 Configurazione Proxim RangeLan2

Per configurare il dispositivo di rete *wireless* Proxim RangeLan2, è necessario procedere come descritto nella sezione 3.2.1 del capitolo 3.

Appendice C

Quick Start

C.1 L'accensione

Collegare le batterie, inserendole nel vano apposito, collegare tastiera e monitor sul pannello di controllo (si rimanda alla sezione 2.1.4), quindi procedere all'accensione premendo il tasto `On`. Attendere qualche secondo; gli elementi presenti sulla circonferenza superiore ed inferiore del robot verranno prima attivati simultaneamente, e poi inizieranno ad attivarsi in modo sequenziale. Se il *bootstrap* è andato a buon fine apparirà a video il *prompt* di *login* di linux e il demone `DRobot` provvederà a mettere nella posizione di *home*[1] i motori e a disattivare i sensori attivi (sonar/infrarosso). Per accedere come amministratore[2] identificarsi come:

Utente : `root`

Password : `N0mad1C`

Il *login* per il server della rete *N.I.N.E.* è analogo a quello utilizzato per il Nomad XR4000.

[1] Per verificare che il robot sta effettivamente compiendo i comandi impartiti all'avvio è sufficiente premere i tasti `ALT + F8`, e verificare i comandi a schermo.

[2] Se si sono mantenute le impostazioni consigliate in questo manuale.

C.2 L'ambiente di sviluppo

L'ambiente di sviluppo è una suite integrata di librerie per la programmazione del Nomad XR4000 ed è presente nella directory `/usr/local/xrdev`. La struttura principale del pacchetto di sviluppo è la seguente:

`/usr/local/xrdev/examples` Directory dei file di esempio di ogni periferica del robot. Per compilare i singoli esempi basta eseguire nella directory `make nomeesempio`. Per compilare i propri sorgenti analogamente assicurarsi di includere il file header `Nclient.h` e di compilarli con la libreria `Nhost_client.a`, ad esempio `gcc nomefile.c ../lib/Nhost_client.a -o nomeesempio`.

`/usr/local/xrdev/compass` Directory dove sono installati i file di esempio e le librerie della bussola digitale. Per compilare il file di esempio eseguire nella directory `/usr/local/xrdev/compass`, il comando `gcc Compass_example.c compass_lib.c -o Compass_example`. La libreria `compass_lib.c` può essere compilata per una maggiore portabilità con il comando `gcc -c compass_lib.c`, in questo modo per utilizzarne le funzioni basta includere nel proprio programma il file header `compass_lib.h` e compilare il proprio file con `gcc nomefile.c compass_lib.o -o nomefile`.

`/usr/local/xrdev/bin` Directory dove sono installati i file di utilità come `Ngui`, interfaccia per *XWindow* del nomad XR4000.

`/usr/local/xrdev/sbin` Directory dove sono installati i file eseguibili di sistema del demone `DRobot`.

`/usr/local/xrdev/etc` Directory dove sono installati i file di configurazione per il demone `DRobot` ed il firmware del controller del motore (`xrm.prg`) caricato ad ogni avvio.

`/usr/local/xrdev/doc` Directory dove è contenuta una copia di questo manuale.

Dopo aver letto questo manuale è fortemente consigliato di prendere confidenza con questo ambiente di sviluppo, eseguendo i numerosi file di esempio e leggendoli con attenzione facendo riferimento al capitolo 4.2. Solo in questo modo è possibile acquisire una piena conoscenza della strumentazione presente e raggiungere un livello di esperienza tale da poter affrontare compiti di studio e ricerca nel campo della robotica mobile.

C.3 Lo spegnimento

Per spegnere correttamente il robot eseguire un *clean shutdown,* cioè premere una volta il pulsante rosso `Off`, attendere il segnale acustico e premerlo nuovamente. Il demone `DRobot` eseguirà lo spegnimento corretto del robot Nomad XR4000.

Elenco delle figure

Elenco delle tabelle