ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

DEIS Dipartimento di Elettronica Informatica e Sistemistica
CASY Center for Research on Complex Automated Systems

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

TESI DI LAUREA

in
Controlli Automatici T

Implementazione di un algoritmo di navigazione autonoma di alto livello e sviluppo software di una Control Ground Station

CANDIDATO
Enrico Damini

RELATORE:
Chiar.mo Prof. Lorenzo Marconi

CORRELATORE
Dott. Ing. Roberto Naldi

Anno Accademico [2014/2015]

Sessione III

Indice

1. Intoduzione

S.H.E.R.P.A. (Smart collaboration between Humans and ground-aErial Robots for imProving rescuing activities in Alpine environments) è un progetto europeo che svolge le proprie attività nell'ambito dei sistemi cognitivi e della robotica. Da febbraio 2013 questo progetto che coinvolge 7 università europee, incluso il Center for Research on Complex Automated System (C.A.S.Y.) dell'università di Bologna, nasce con l'idea di far collaborare umani e robot al fine di fornire un valido supporto in uno scenario alpino. L'obiettivo principale è la progettazione e la realizzazione di droni e di rover, affidabili e veloci, in grado di reagire nel minor tempo possibile. Tale tempestività e tale tecnologia renderanno essenziale il loro impiego in entrambi gli scenari montani, sia quello invernale che quello estivo.
L'esperienza maturata durante il tirocinio svolto al CASY è risultata fondamentale per la realizzazione dell'obiettivo da cui prende il titolo questa tesi.

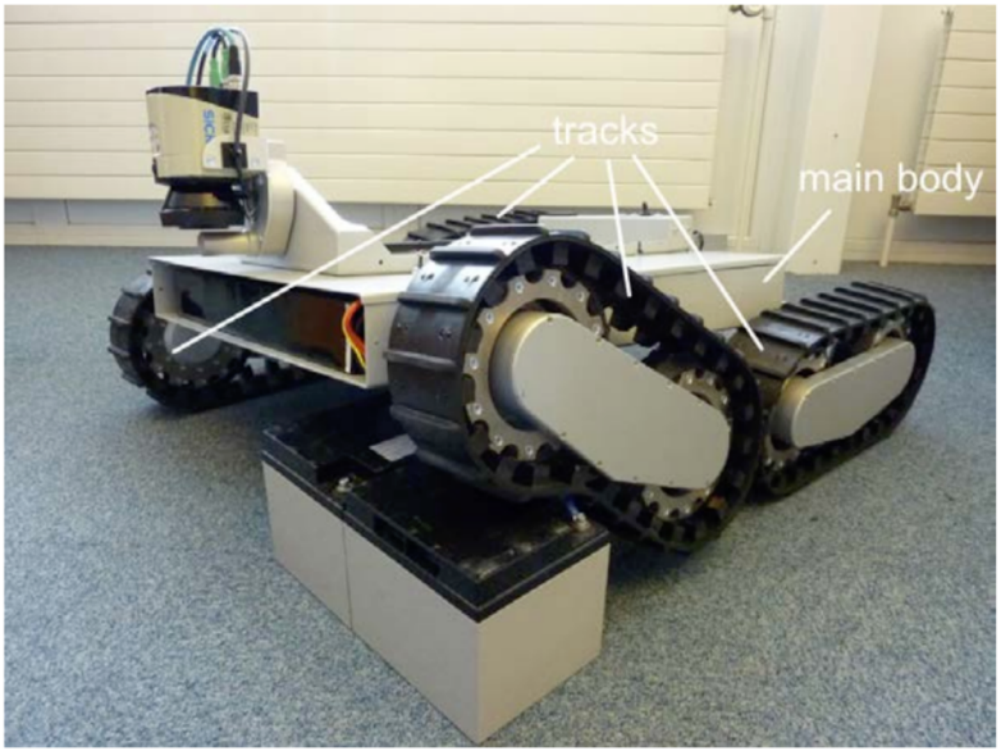
Uno degli obiettivi del lavoro di tesi qui presentato include il corretto funzionamento di navigazione autonoma e di obstacle avoidance del rover Sherpa che fungerà da base mobile per droni.

La struttura finale del rover prevede un braccio meccanico che, grazie all'uso di una stereo camera posizionata su di esso,consente di caricare sulla sommità i droni.

Il rover Sherpa, grazie ai suoi cingolati, sarà in grado di portare i droni in alta montagna, passando anche su terreni accidentati, e di poterli ricaricare quando necessario: è questo il motivo per cui viene denominato "Donkey".

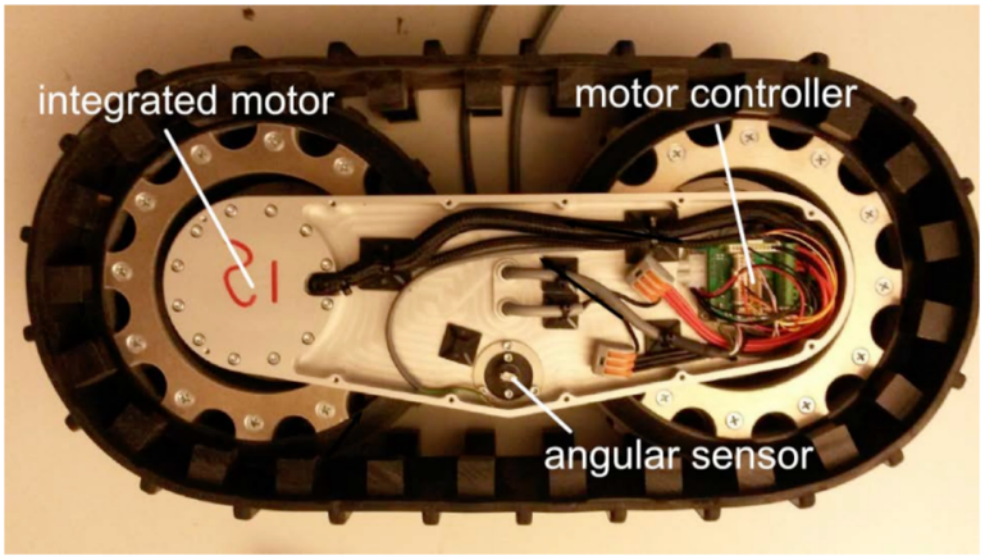
2. Presentazione piattaforma: rover Donkey

La piattaforma è composta da una struttura principale e da quattro cingoli motorizzati. Ogni cingolo ha un proprio motore, un regolatore del motore stesso, e può ruotare liberamente attorno al proprio asse.



2.1. Cingoli motorizzati

Tutti i cingoli sono identici: composti da due ruote, una guidata tramite un motore integrato e un'altra invece ruota liberamente. Il regolatore del motore è incorporato nelle ruote come mostrato in figura. Un sensore angolare è montato sull'asse delle ruote per misurare l'angolo relativo dei cingoli.



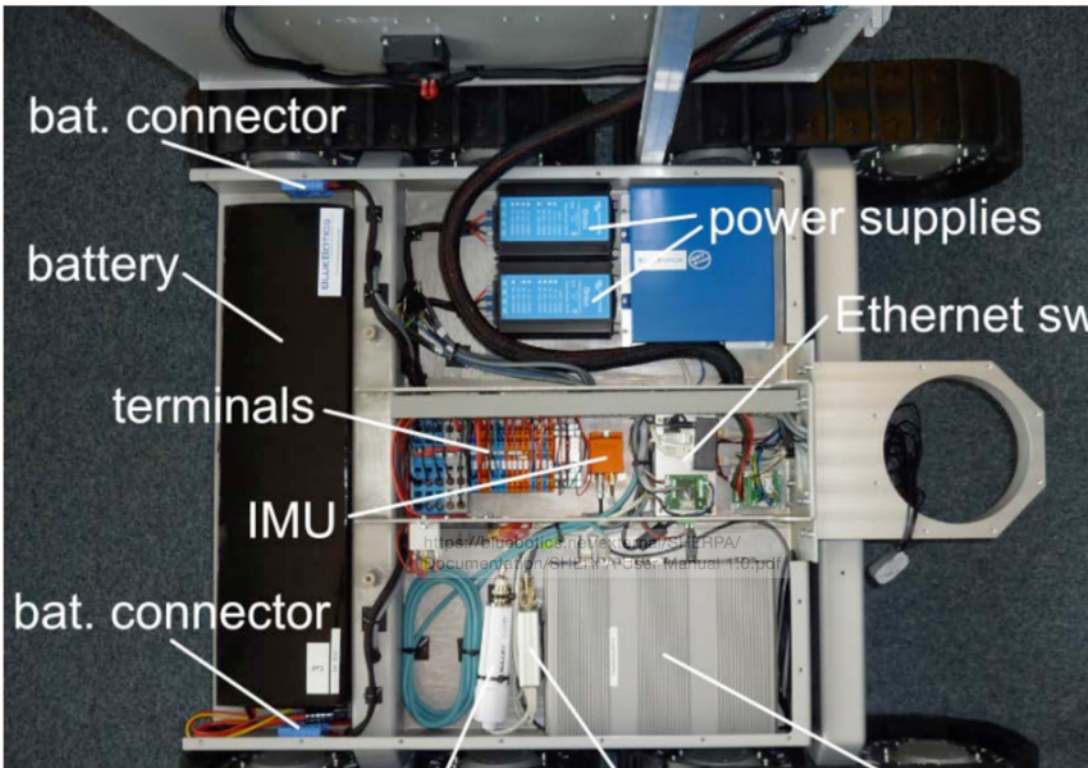
2.2. Struttura principale

I principali componenti elettrici della piattaforma sono:

- Alimentatori (voltage 48V, capacity 30Ah)
- Power terminals
- Embedded PC (Intel 3rd Generation Core i7 processor)
- Ethernet switch
- Convertitore USB to CAN
- WiFi, con antenna che è possibile posizionarla nella posizione più consona
- Batteria litio
- Due connettori per la batteria

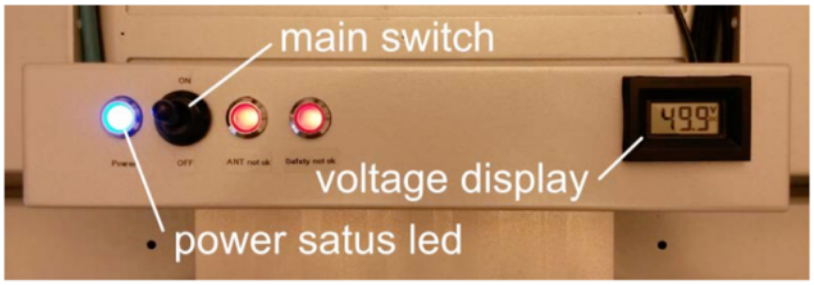
I sensori incorporati:

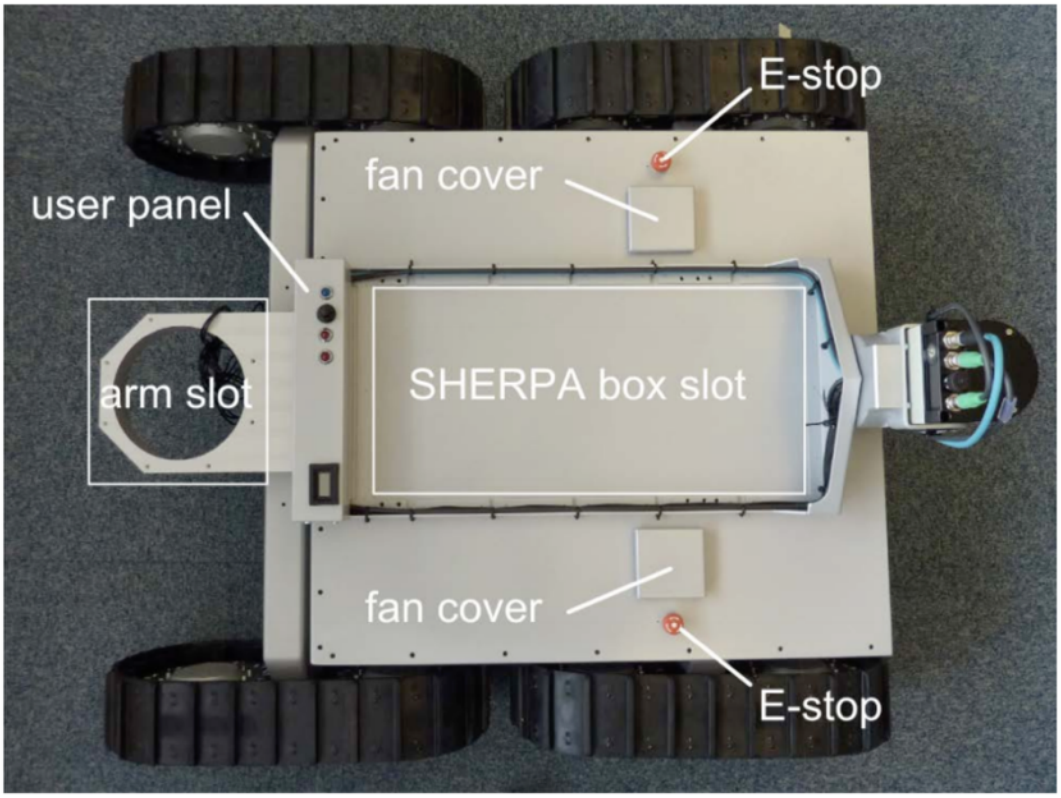
- Sensore 3D (un laser scanner 2D in grado di ruotare di +90°/-90°)
- Motion Tracker: IMU e GPS, l'IMU è all'interno della struttura mentre l'antenna GPS è possibile posizionarla nel posto più adatto.



Il coperchio della struttura principale è dotato di:

- Due bottoni di emergenza che fermano la struttura
- Ventole di raffreddamento
- Un pannello utente dotato di LED di stato, ON/OFF e un display con la tensione della batteria





È visibile in figura lo struttura in cui andrà integrato il braccio robotico citato nell'introduzione (arm slot) e dove invece andranno montate le strutture per i droni (SHERPA box slot).
Il laser scanner con il meccanismo di rotazione è montato davanti alla piattaforma.

2.3. Components references

- Embedded PC: Winmate IV70SB7-111 (P/N WIN-EPC.IV70IF0005).
- 2D laser scanner: SICK LMS-151 (P/N 1047607).
- IMU/GPS: XSens MTI-G-700-2A5G4 - MTi-G-700-GPS/INS RS232, USB
- USB-CAN: EMS-Wuensche (P/N CPC-USB/ARM7-GTI).
- WiFi: Ubiquiti Bullet M2

2.4. Embedded PC è dotato di:

- 4x USB.
- 2x Ethernet.
- 3x RS-232.
- VGA output.
- PS2 per mouse e tastiera.
- Audio microphone.
- Audio speaker out

Ubuntu 15.04 è il sistema operativo installato di default sul PC. Su questo PC si andrà a strutturare tutto il middle framework per integrare il sistema di controllo ANT con il framework ROS.
Il PC usa un USB to CAN converter per comunicare con i motori dei cingoli. È inoltre disponibile una libreria in C/C++ di basso livello in cui è possibile sfruttarne le API per creare leggi di guida e di osbtacle avoidance diverse da quelle già implementate da ANT.
Tuttavia l'obiettivo del lavoro svolto non è quello di creare nuovi algoritmi di leggi di guida o di obstacle avoidance, bensì di sfruttare quelli già realizzati e implementare una struttura software in grado di unire e sfruttare ciò che si ha a disposizione.

2.5. ANT

ANT è un sistema che permette il controllo della piattaforma su cui è installata. È un sistema molto portabile adatto a molteplici piattaforme. Consente un semplice, se pur non banale, utilizzo della libreria che mette a disposizione (LOS). I software forniti si basano su questa libreriar. Descrizioni più dettagliate sulla logica degli algoritmi del controllo di ANT e sulla libreria LOS verranno date in seguito.

Il sistema ANT è composta da:

- ANT box
- Joystick
- Configuration cable

3. MTi-G-700 GPS/INS: cenni teorici e tecnici

3.1. Introduzione

La gamma dei prodotti MTi dell'xsens sono attualmente 7 e variano in termini di funzionalità. Tutti i prodotti contengono un sensore inerziale 3D (giroscopi e accelerometri) e magnetometri 3D, con eventualmente un ricevitore GNSS.

Vi sono due serie di prodotti: MTi-10-series e Mti-100-series. I primi sono ad un livello base ed offrono un robusto livello di precisione ma hanno un output limitato, diversamente la seconda serie che è un del tutto rivoluzionaria: offre moduli di orientamento e precisione senza precedenti e una vasta gamma di output.

Tutte le serie MTi sono dotate di un multi-processore capace di elaborare roll,pitch e yaw a bassa latenza, inoltre è in grado di fornire un output calibrato di: accelerazione 3D, velocità di virata (gyro), campo magnetico terreste e pressione atmosferica (solo 100-series). L'MTi-G-700 GPS/INS offre anche la posizione e la velocità 3D.

All'interno del rover è presente un MTi-G-700 GPS/INS, il top di gamma: una soluzione completamente integrata che include un ricevitore di bordo GNSS e che è in grado di migliorare l'accuratezza della posizione in modo significativo attraverso i dati forniti dai sensori di movimento.

Una completa descrizione di tutte le funzionalità è riportata qui sotto:

Group Name	Type Name	Valid for MTi product								Max frequency[3]
		10	20	30	100	200	300	700	710	
Temperature		•	•	•	•	•	•	•	•	1 Hz
	Temperature	•	•	•	•	•	•	•	•	
Timestamp		•	•	•	•	•	•	•	•	2000 Hz
	UTC Time	•	•	•	•	•	•	•	•	
	Packet Counter	•	•	•	•	•	•	•	•	
	Integer Time of Week (ITOW)	•	•	•	•	•	•	•	•	
	GPS Age (legacy)									
	Pressure Age (legacy)									
	Sample Time Fine	•	•	•	•	•	•	•	•	
	Sample Time Coarse	•	•	•	•	•	•	•	•	
	Frame Range									
Orientation Data			•	•		•	•	•	•	400 Hz
	Quaternion		•	•		•	•	•	•	
	Rotation Matrix		•	•		•	•	•	•	
	Euler Angles		•	•		•	•	•	•	
Pressure					•	•	•	•	•	50 Hz
	Baro Pressure				•	•	•	•	•	
Acceleration		•	•	•	•	•	•	•	•	2000 Hz (see note)
	Delta V	•	•	•	•	•	•	•	•	
	Acceleration	•	•	•	•	•	•	•	•	
	Free Acceleration		•	•		•	•	•	•	
Trigger Indication		**MTw only; for MTi, see StatusWord**								
	TriggerIn1	MTw only; for MTi, see StatusWord								
	TriggerIn1	MTw only; for MTi, see StatusWord								
Position								•	•	400 Hz

	Altitude Ellipsoid							•	•	
	Position ECEF							•	•	
	LatLon							•	•	
GNSS								•	•	4 Hz
	GNSS PVT data							•	•	
	GNSS satellites info							•	•	
Angular Velocity		•	•	•	•	•	•	•	•	2000 Hz (see note)
	Rate of Turn	•	•	•	•	•	•	•	•	
	Delta Q	•	•	•	•	•	•	•	•	
GPS								•		4 Hz
	DOP							•		
	SOL							•		
	Time UTC							•		
	SV Info							•		
Sensor Component Readout (SCR)		•	•	•	•	•	•	•	•	2000 Hz
	ACC, GYR, MAG, temperature	•	•	•	•	•	•	•	•	
	Gyro temperatures	•	•	•	•	•	•	•	•	
Analog In										2000 Hz
	Analog In 1									
	Analog In 2									
Magnetic		•	•	•	•	•	•	•	•	50 Hz
	Magnetic Field	•	•	•	•	•	•	•	•	
Velocity								•	•	400 Hz
	Velocity XYZ							•	•	
Status		•	•	•	•	•	•	•	•	2000 Hz
	Status Byte	•	•	•	•	•	•	•	•	
	Status Word	•	•	•	•	•	•	•	•	
	RSSI									

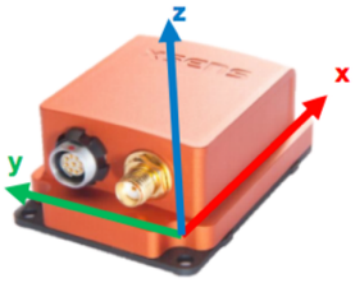
3.2. Filtri

L'algoritmo dei filtri dell'Xsens prevede un approccio "sensor fusion", ovvero con la combinazione dei vari sensori si cerca di dare una stima più accurata, dando inoltre la possibilità di avere tante funzionalità avanzate.

3.3. Coordinate

Le coordinate del sensore sono cartesiane e fissate nell'origine sull'accelerometro (con la regola della mano destra) in accordo con il sistema ENU che è lo standard di navigazione inerziale per le applicazioni geodetiche:

- X positiva verso Est (E)
- Y positiva verso Nord (N)
- Z positiva quando rivolta verso l'alto (U)



3.4. Yaw, Heading e Bearing

L'heading è definito come angolo tra tra il Nord e la proiezione orizzontale dell'asse del rover, mentre il bearing è la direzione tra la destinazione e il Nord.
Lo yaw è definito come l'angolo tra un asse di navigazione orizzontale e la proiezione dell'asse longitudinale nel piano orizzontale seguendo la regola della mano destra. Lo yaw ha dominio compreso tra $-\pi$ e π .
Usando la convenzione ENU, lo yaw risulta essere 0 quando il rover (asse X dell'MTi) sta puntando verso Nord.

Il profilo generale, con cui lavorano i filtri dell'Xsens, prevede il calcolo dello yaw a partire dal confronto tra l'accelerazione GPS e gli accelerometri di bordo. In questo modo più sarà elevata la velocità e più il valore di yaw sarà migliore.

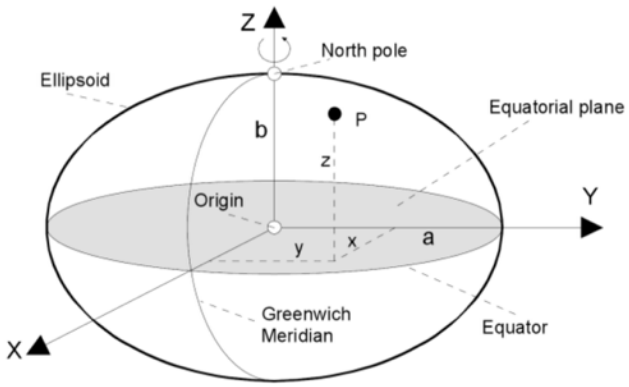
3.5. Dati di posizione

I dati della posizione sono calcolati sempre attraverso un approccio "sensor fusion" e sono latitudine, longitudine e altitudine.
I dati forniti dal GPS receiver sono rappresentati in ECEF (Earth Centered – Earth Fixed).

3.6. Coordinate globali

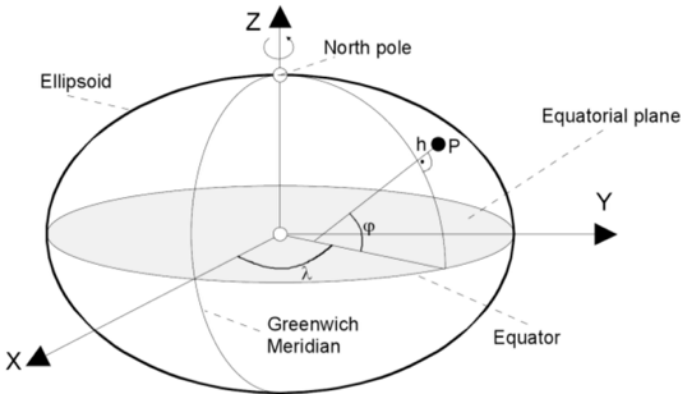
Al fine di calcolare la posizione di un ricevitore GNSS, è più conveniente utilizzare il sistema di coordinate cartesiane geocentrico che ruota in maniera solidale con la Terra, ed è appunto il ECEF.
Il piano X-Y è coincidente con il piano equatoriale con i rispettivi versori che puntano nelle direzioni di longitudine 0° e 90°; invece l'asse Z ortogonale a questo piano punta nella direzione del Polo Nord. Le coordinate X,Y,Z sono rappresentate in metri.
Risulta molto utile trasformare queste coordinate cartesiane in latitudine, longitudine e altitudine (nel caso di questa tesi sarà conveniente trasformare la latitudine e longitudine in coordinare cartesiane).
Per effettuare questa trasformazione, è necessario disporre di un modello fisico che descrive la Terra. Il modello fisico standard della Terra utilizzato per applicazioni GPS è il World Geodetic System 1984 (WGS84).
Il WGS84 è basato su un ellissoide di riferimento e costituisce un modello matematico della Terra da un punto di vista geometrico, geodetico e gravitazionale, costruito sulla base delle misure e delle conoscenze scientifiche e tecnologiche.
I principali parametri sono il semiasse maggiore a (= 6.378.137 m) e il semi-asse minore b (= 6.356.752 m) come descritto nella figura che segue.
Ci sono diversi modelli che incrementano l'accuratezza con diversi valori ma l'MTi in questione utilizza il modello di default.

Earth Centered Earth Fixed – ECEF
WGS-84 parameters:
a = 6,378,317 meter
b = 6,356,752 meter

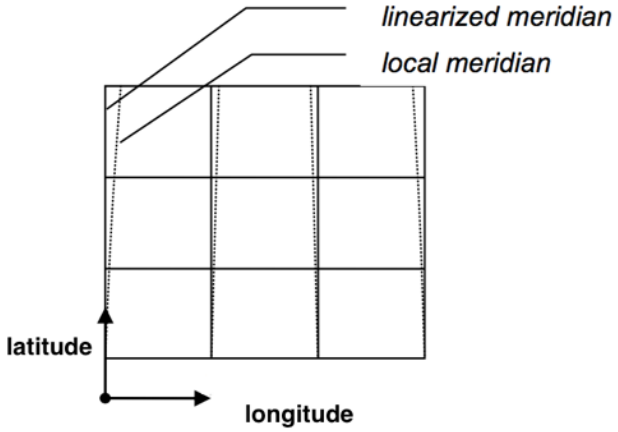


Definizione delle coordinate ellissoidali (latitudine, longitudine, altitudine) in WGS84:

λ = longitude
ϕ = latitude
h = altitude



Per combinare l'uscita del ricevitore GNSS con l'IMU, entrambi i sistemi devono essere convertiti in un sistema di coordinate appropriato.
I sensori inerziali MEMS utilizzati nel MTi-G non sono sufficientemente accurati per misurare la rotazione della terra nel momento in cui l'MTi, che si muove sulla superficie curva terrestre, è in movimento. Pertanto, senza commettere errori significativi, si può lavorare con un piano locale tangente linearizzato.
Questo sistema è chiamato LTP (locally tangent plane) ed è in effetti una linearizzazione locale delle coordinate ellissoidali (latitudine, longitudine, altitudine) nel sistema WGS-84.
Nella seguente figura si può vedere la distorsione:



Per ridurre al minimo l'errore di linearizzazione, le coordinate di riferimento devono essere scelte il più vicino possibile ai punti che vengono mappati. L'MTi-G esegue una linearizzazione locale per ogni aggiornamento GNSS valida secondo il seguente schema di linearizzazione. L'altezza è la stessa per entrambi i sistemi di coordinate.

$$\begin{cases} y = R \cdot \Delta\varphi \cdot \cos(\theta) \\ x = R \cdot \Delta\theta \end{cases}$$

dove R è il raggio della terra alla latitudine data mentre $\Delta\theta$ e $\Delta\varphi$:

$$\begin{cases} \Delta\theta = \theta - \theta_{ref} \quad \text{(differenza tra latitudine data e di riferimento)} \\ \Delta\varphi = \varphi - \varphi_{ref} \text{ (differenza tra longitudine data e di riferimento)} \end{cases}$$

3.7. Specifiche GPS receiver

GNSS Receiver specification	MTi-G-700 GPS	MTi-G-710 GNSS
Receiver Type:	50 channels, GPS L1, C/A code	72 channels, GPS/QZSS L1 C/A, GLONASS L10F,BeiDou B1
Datum, reference frame	WGS84	WGS84
GNSS Update Rate:	4 Hz	4 Hz
Horizontal Accuracy Position SPS:	2.5 m CEP	2.5 m CEP
SBAS:	2.0 m CEP	N/A
Vertical Accuracy Position SPS:	5 m	5m
Velocity accuracy	0.1 m/s @ 30 m/s	0.05 m/s
Heading (Course-over-Ground)	N/A	0.3 °
Start-up Time Cold start:	27 s	26 s (GPS+GLONASS)
Re-acquisition:	<1 s	<1 s
Tracking Sensitivity:	-161 dBm	-164 dBm (GPS+GLONASS)
Timing Accuracy:	30 ns RMS	30 ns RMS
Maximum Altitude:	18 km	50 km
Maximum Velocity:	515 m/s	500 m/s
Max dynamics GNSS:	4 g	4g

3.8. Protocollo di comunicazione

Tutti gli MTi utilizzano un protocollo di comunicazione binario chiamato “MT Communication Protocol”.
Tale protocollo è basato su messaggi che permettono all’utente di configurare il dispositivo. Esempi sono la frequenza di campionamento, la sincronizzazione di input e output, velocità di trasmissione e modalità di uscita.

3.8.1. Stato

Le modifiche di configurazione vengono eseguite nello stato denominato “Config State". In questo stato la MTi accetta messaggi che impostano la modalità di uscita o altre impostazioni.
Ogni volta che viene impostata la configurazione preferita l'utente può settare il dispositivo MTi nello "Measurement State". In questo stato la MT inizia a emettere i dati in base alle impostazioni di configurazione correnti.

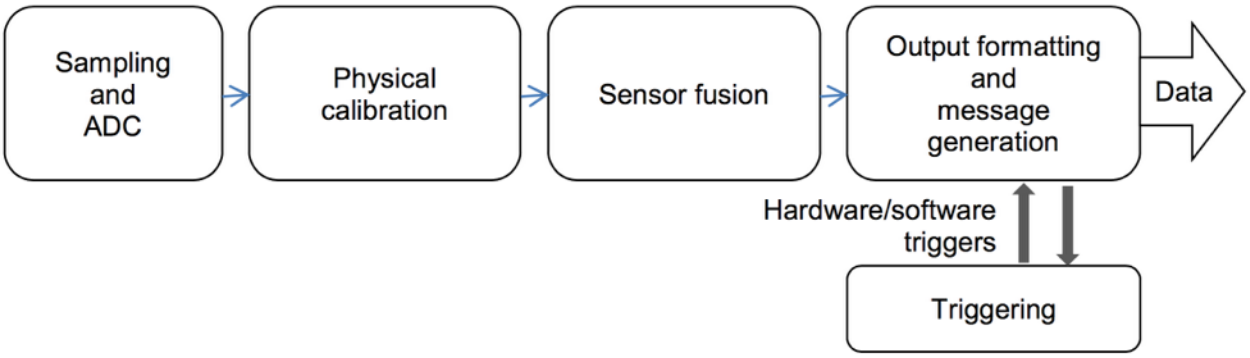
3.8.2. Messaggi

La comunicazione con il MTi è fatta da messaggi che sono costruiti secondo una struttura che prevede o uno con una lunghezza standard o uno con lunghezza estesa.
Il messaggio lunghezza standard ha un massimo di 254 byte di dati ed è usato più di frequente. In alcuni casi, il messaggio di lunghezza estesa deve essere utilizzato se il numero di byte di dati supera 254 byte.

Field	Field width	Description
Preamble	1 byte	Indicator of start of packet → 250 (0xFA)
BID	1 byte	Bus identifier or Address → 255 (0xFF)
MID	1 byte	Message identifier
LEN	1 byte	For standard length message: Value equals number of bytes in DATA field. Maximum value is 254 (0xFE) For extended length message: Field value is always 255 (0xFF)
EXT LEN	2 bytes	16 bit value representing the number of data bytes for extended length messages. Maximum value is 2048 (0x0800)
DATA (standard length)	0 – 254 bytes	Data bytes (optional)
DATA (extended length)	255 – 2048 bytes	Data bytes
Checksum	1 byte	Checksum of message

3.9. Communication Timing

Per molte applicazioni può essere fondamentale conoscere esattamente i vari ritardi e latenze in un sistema.
L' emissioni di dati possono essere attivate dal clock interno del dispositivo, oppure da trigger software esterni (polling), o trigger hardware.
Lo schema di sotto descrive bene tutti i passaggi necessari e può dare un idea di come i ritardi possono essere consistenti.



Il ritardo di tempo tra un evento fisico (ad esempio una variazione dell'orientamento o accelerazione) è dettata da due fattori:

- acquisizione interna, tempo di calcolo e la generazione dei messaggi (la durata di elaborazione del segnale)
- tempo di trasmissione seriale

Grazie all'architettura di sistema dell'algoritmo "sensor fusion", la durata dell'elaborazione del segnale è indipendente dal profilo di filtraggio.

Utilizzando unità di elaborazione multicore è possibile ridurre il tempo totale di un evento fisico per la trasmissione dei dati sull'uscita (solitamente al di sotto di 2 ms).

Il tempo di trasmissione può essere calcolato facilmente quando l dimensione del messaggio e la velocità di trasmissione sono noti:

$$\frac{(total\ bytes\ in\ message) * 10\ bits/byte}{communication\ baudrate\ (bits/s)} = \text{transmission time}$$

3.10. Note

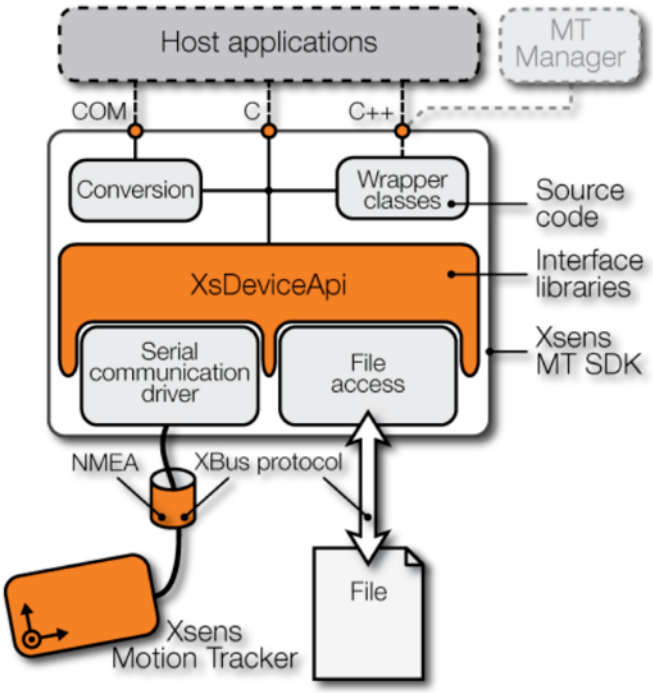
3.10.1.Tempi di comunicazione USB

Quando MTi viene utilizzato con il cavo USB, gran parte della temporizzazione dipende dallo scheduling dell'host (ad esempio Windows,Linux..) che deve interrogare i dati dai dispositivi USB.

3.10.2. Xsens Software: MTManager, FirmwareUpdater e MagfieldMapper

È stato utile il supporto di questi tre software di proprietà Xsens: il primo usato per interfacciarsi con il dispositivo, il secondo solo per aggiornare il firmware ed il terzo per calibrare il dispositivo.

L'Xsens mette a disposizione delle librerie (principalmente in C/C++ ma anche in Java e in Python) per potersi interfacciare con i dispositivi. Qui di sotto uno schema riassuntivo che mostra l'architettura implementata.
Per poter usare qualsiasi software Xsens (librerie comprese) è necessario essere in possesso di una serial key.



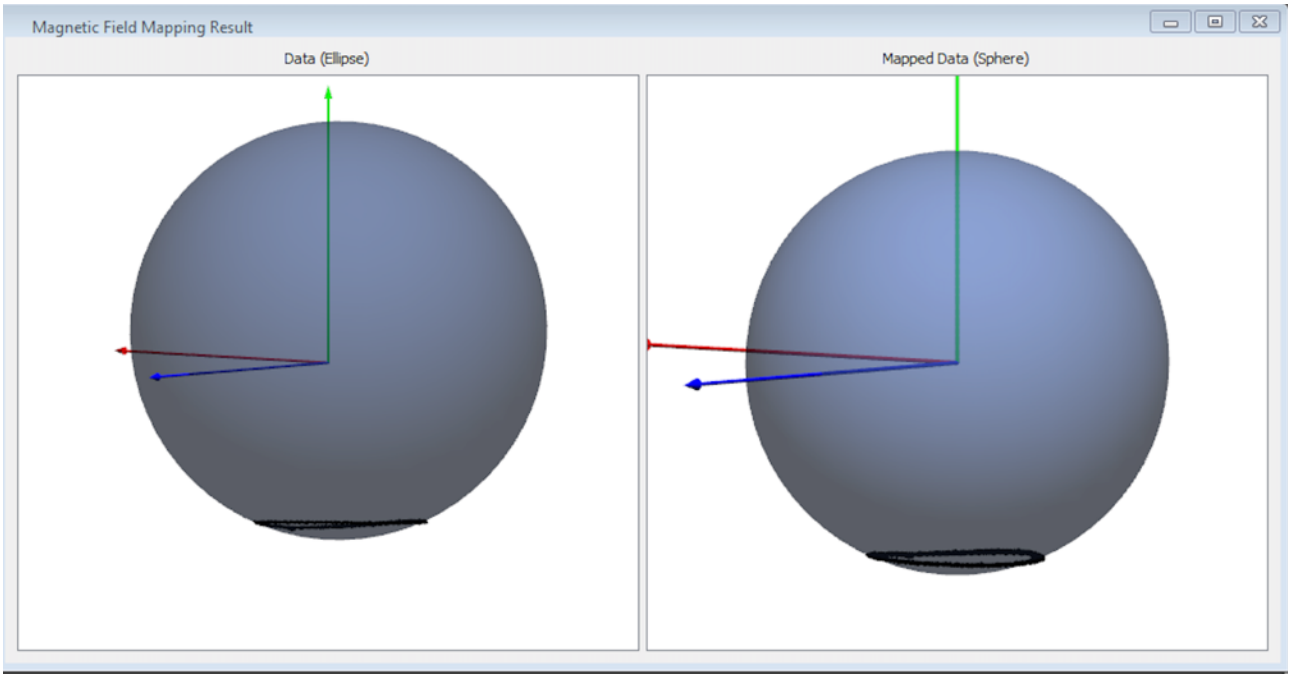
3.10.2.1. Calibrazione con MagfieldMapper

L'MTi è disturbato dai dei campi magnetici causati sia dall'ambiente che dalla piattaforma stessa.
Un campo magnetico locale causa un errore in orientamento che può essere molto rilevante. Il campo magnetico terrestre è alterato da materiali ferromagnetici, magneti permanenti o correnti molto forti (alcuni ampere).
La misurazione della direzione del campo magnetico terrestre viene utilizzata come bussola per determinare la direzione del nord (heading o yaw).

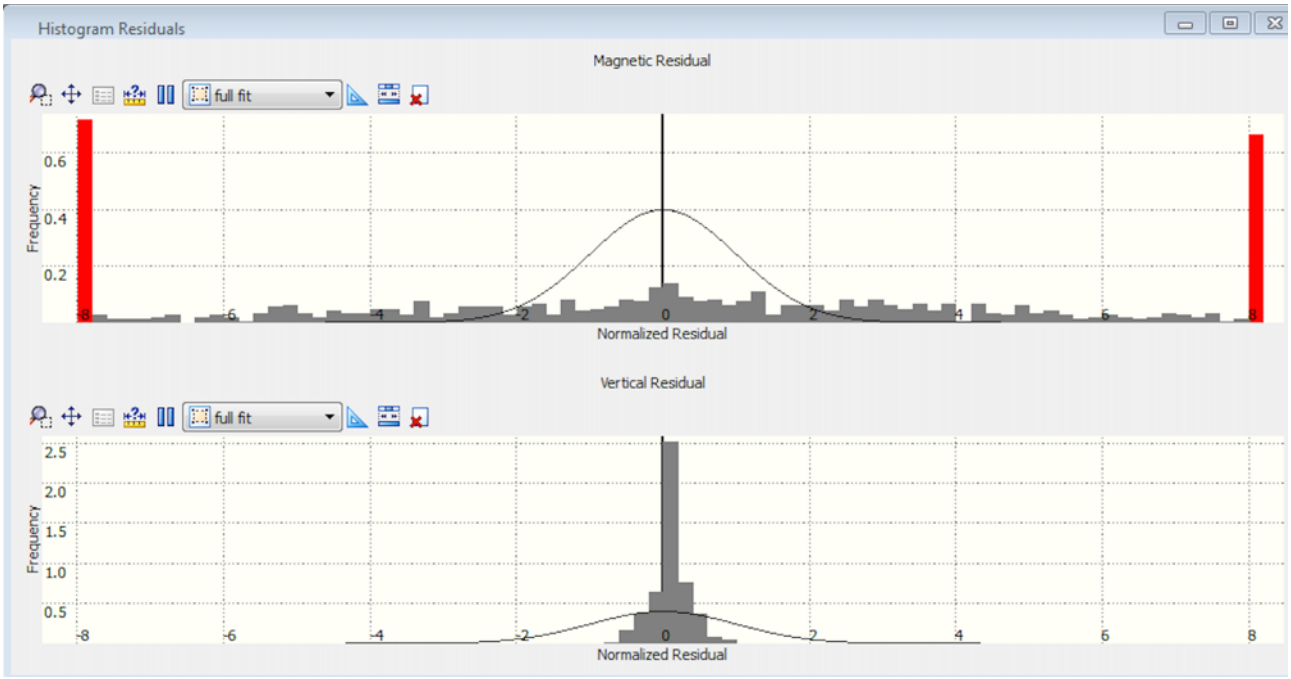
Quando i filtri che vengono applicati non sono sufficienti è necessario calibrare il dispositivo. Attraverso MagfieldMapper diventa un operazione abbastanza veloce.
Una calibrazione accurata si può ottenere registrando i segnali dell'MTi ruotando la piattaforma su cui è montato in uno spazio senza altri materiali ferromagnetici nelle vicinanze. Una volta che il rover viene ruotato per un numero sufficientemente grande (3 o 4 rotazioni), MagfieldMapper è in grado calcolare i nuovi parametri di calibrazione che possono essere immediatamente utilizzati dal MTi.

In un campo magnetico non disturbato, il vettore del campo magnetico misurato ha una grandezza pari a 1, quindi tutti i punti misurati si trovano sulla circonferenza di una sfera con centro nell'origine. Nel caso di un campo magnetico disturbato, questa sfera può essere spostata o deformata. La procedura di calibrazione di MagfieldMapper intende ricavare una funzione che associa il vettore del campo magnetico misurato ad una sfera.

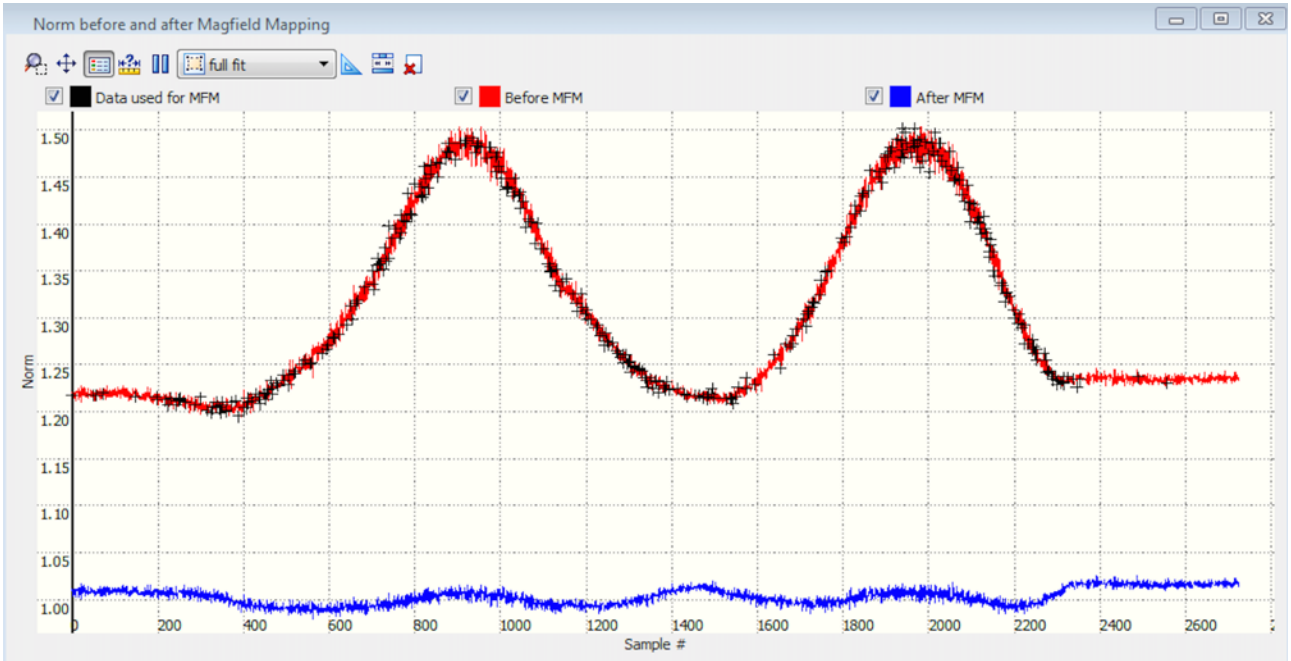
Si ricorda che è probabile che nella struttura dell'edificio (pavimento e soffitto) siano contenuti materiali magnetici.



Nei risultati, questa figura mostra il movimento totale del MTi lungo tutte le direzioni di una sfera ideale. In questo caso abbiamo fatto ruotare il rover di 360°; quindi come si vede ne deriva un cerchio.
Nell'immagine di sinistra sono mostrate le letture del campo magnetico originale, mentre a destra vengono mostrati i valori del campo magnetico dopo la procedura MFM.



In questa figura vengono mostrati i residui del campo magnetico che è stato corretto rispetto al nuovo modello del campo magnetico.
Quando le misurazioni sono visibili all'esterno della distribuzione Gaussiana, significa che i dati utilizzati dal MFM avevano errori.



Quest'ultimo risultato mostra la norma del campo magnetico prima e dopo l'intervento di MagfieldMapper. Inoltre vengono mostrati i punti che sono usati per la mappatura del campo magnetico.
Più piatta (e vicina ad 1) è la linea blu e migliore sarà la mappatura di MagfieldMapper.

4. Embedded PC

(IV70SB7-111)

1. Power Jack
2. PS/2
3. RS-232 Conn.
4. RS232/422/485 Conn.
5. VGA Conn.
6. USB 3.0
7. RJ-45
8. AudioJack

4.1. Caratteristiche:

System Specification

Processor	:	Intel® 3rd Generation Core ™ i7/i5/i3 processor (Ivy Bridge)
BIOS	:	AMI UEFI BIOS
System Chipset	:	Intel® **7** series Chipset (HM76)
System Memory	:	2 x SODIMM, Max 32GB DDR3 1333/1600
Graphic Chipset	:	Integrated Intel® HD Graphic 4000
Ethernet	:	Intel 82579-LM+Intel® 825742 GbE LAN
Audio	:	Realtek ALC886 HD codec (Line-in, Line out, Mic-in)
USB	:	6 x USB ports, 4 x USB 3.0, 2 x USB 2.0
Serial	:	2 x RS232, 1 x RS232/422/485 ports, 1 x RS232(Optional)
SSD Interface	:	1 x mSATA SSD slot

I/O Connectors

Rear Panel External I/Os	:	2 x RJ-45(10/100/1000 Mpbs)
		2 x RS-232 Conn
		1 x RS-232/422/485 Conn
		1 x RS-232 (Optional)
		1 x VGA Conn
		4 x USB 3.0
		2 x PS/2
		1 x Line in/Line out/Mic in
Front Panel External I/Os	:	2 x USB 2.0
		1 x Power/HDD LED
		1 x Reset / Power on Button

Mechanical Specification

Construction	:	Aluminum housing
Mouting	:	Desktop/wall-mounting
Dimension	:	264 x 57 x 192 mm (W x H x D)

Su questo PC era già installata una versione di Ubuntu (15.04). Di base il PC non ha assolutamente nessun compito all'interno del rover; è stato installato sulla piattaforma per dare la possibilità agli sviluppatori di sperimentare. Infatti è stato accostato ad ANT che invece già implementa tutte le funzionalità per poterlo comandare, sia in modalità autonoma sia attraverso il joystick.
Si è deciso di installare la penultima versione di ROS (Indigo) poiché ci permette di sfruttare a pieno la community di wikiROS.

Non bisogna dimenticare che sarebbe possibile far funzionare tutto senza l'utilizzo del framework ROS, ma l'obiettivo di questo lavoro è anche questo: riuscire a standardizzare e allo stesso tempo sfruttare ciò che già è stato implementato.
Difatti tutte le piattaforme e i vari componenti sono sviluppati in ambiente ROS ed è questo un requisito fondamentale da rispettare.

Per comunicare con il PC è possibile collegarsi fisicamente attraverso la porta VGA; tuttavia è sconsigliato poiché lo spazio e il design non è ottimizzato e risulta quindi estremamente difficoltoso riuscire a connetterlo. Soprattutto perché vi è il reale rischio di compromettere qualche cavo o connettore.

La comunicazione va quindi effettuata attraverso la rete WiFi e con il supporto di protocolli di comunicazione come FTP o SSH.
Se non si vuole lavorare solamente a riga di comando si deve optare per una comunicazione attraverso software VNC, già presente su Ubuntu.

5. LOS

Le librerie LOS (Lightweight Object Streaming) definiscono le API che sono usate nel sistema ANT per lo scambio di dati e il controllo della piattaforma.
Le API sono implementate come RPC (Remote Procedure Call) in una configurazione Client/Server: la piattaforma è il lato server e invece il client (il PC su cui vengono eseguiti i comandi) invoca le procedure a chiamata e riceve i risultati con le relative eccezioni. Il transport layer si basa sul protocollo TCP.

Le RPC e le risposte sono serializzate in un semplice e compatto formato binario chiamato Lightweight Object Streaming che permette l'efficiente trasmissione di un ampia varietà di dati, di array e di altri tipi di oggetto.

Le API si possono dividere in vari sottosistemi: localization, motion control, obstacle avoidance, odometry, scan data processing, etc.

La tecnologia ANT necessita di un ambiente conosciuto a priori e ciò potrebbe diventare un limite se consideriamo quali sono i nostri scopi.
È quindi necessario dare la possibilità al software, che verrà realizzato e che si appoggerà sopra il framework LOS/ROS, di non essere vincolato da alcuna mappa.

La libreria di LOS è implementata sia in Python che in Java: nel primo caso è estremamente utile, poiché consente lo sviluppo di software all'interno del framework ROS, nel secondo invece risulta utile per la Control Station (un' interfaccia grafica per un facile controllo della piattaforma). Quest'ultima verrà descritta nell'ultimo capitolo.

5.1. Autenticazione

Ogni connessione ha un livello di autenticazione associata e vengono autenticate all'inizio di ogni connessione, altrimenti si scatenerà un eccezione. Ogni livello di autenticazione da accesso ad una serie di funzioni. Si può cambiare il livello di autenticazione attraverso la funzione di accesso login().
Vi sono tre livelli: master, user e nobody. Il master ha accesso a tutte le funzioni mentre lo User solo ad alcune.

5.2. Risultati

Le procedure call restituiscono sempre un singolo oggetto, oppure un risultato vuoto (void). Per risultati multipli sono sfruttati gli array.

5.3. Argomenti

Gli argomenti delle procedure call sono posizionali, oppure determinati dalla posizione della lista degli argomenti. Alcune chiamate hanno argomenti opzionali, che possono essere lasciate fuori dalla lista degli argomenti, e che sono posti sempre alla fine della lista degli argomenti.

5.4. Procedure Call

I nomi delle procedure call sono strutturati in due parti, un nome del sottosistema e un nome della procedura, separati da un punto. Le seguenti sezioni ne descrivono solo alcuni dei sottosistemi disponibili e delle chiamate che forniscono.
Alcuni non avendo sottosistemi sono definiti solo dal nome della procedure call.

5.5. Configure

Questa chiamata consente di impostare i parametri che consentono di configurare i vari aspetti della piattaforma.

Name	`configure`	
Auth level	`User`	
Arguments		
`parameters`	`Struct`	A structure containing configuration parameters to be set
Results		
`unset`	`String[]`	The names of the parameters that could not be set
Exceptions		
–		

5.6. Login

Questa chiamata modifica le credenziali di una connessione fornendo un nome utente e una password. Per de- autenticare una connessione, cioè per tornare al livello di autenticazione più basso “nobody” si utilizza una stringa vuota come utente.

Name	`login`	
Auth level	`{nobody}`	
Arguments		
`user`	`String`	The authentication level to switch to
`password`	`String`	The password associated with the user
Results		
	`Void`	
Exceptions		
`LoginRefused`		The user / password pair is invalid.

5.7. Localization

L'algoritmo di localizzazione interno estrae periodicamente caratteristiche dell'ambiente dal laser scanner e li abbina a caratteristiche definite nella mappa, e corregge la stima della posizione corrente in base a queste informazioni. Deve essere disattivato se viene utilizzato un algoritmo di localizzazione esterna.

5.7.1. Paramentri di configurazione

È importante ricordare di disattivare questo fondamentale parametro di configurazione; altrimenti i comandi di Motion che si vogliono impartire alla piattaforma verranno ignorati e lo stato del rover rimarrà "Disable.Autonomous".

Name	Type	Default
`Localization.active`	`Boolean`	`true`
If true, activate the internal localization system. If false, deactivate it.		

5.7.2. localize

Questa chiamata esegue un ciclo di localizzazione utilizzando le funzionalità fornite esternamente.

Name	`Localization.localize`	
Auth level	`User`	
Arguments		
`time`	`Float64`	The time at which the features have been extracted as an absolute UTC time `[s]`
`features`	`Struct`	A structure with one field per feature type, mapping to `Float64[]` values containing the feature parameters.
Results		
`pairings`	`Int32`	The number of provided features that have matched features in the map.
Exceptions		
`Localization.InvalidTime`		The given timestamp cannot be found in the odometry history
`Localization .InvalidFeatureType`		The given feature type is not supported

5.7.3. snapToPose

Questa chiamata consente di inizializzare la posizione corrente in un percorso noto specificando una pozione in coordinate globali.È utile per specificare una posizione iniziale dopo l'accensione della piattaforma, nonché di ri-localizzare la piattaforma nel caso dell'algoritmo di localizzazione si perdesse. In particolare si sfrutta anche per dare comandi di movimento verso una posizione relativa.

Name	`Localization.snapToPose`	
Auth level	`User`	
Arguments		
`x`	`Float64`	The X coordinate of the pose to snap to in world coordinates `[m]`
`y`	`Float64`	The Y coordinate of the pose to snap to in world coordinates `[m]`
`theta`	`Float64`	The heading of the pose to snap to in world coordinates `[rad]`
Results		
	`Void`	
Exceptions		
–		

5.8. Map

La mappa contiene delle informazioni a priori necessarie per la localizzazione, obstacle avoidance e navigazione autonoma.

5.8.1. get

Questa chiamata ritorna un testo che rappresenta la mappa usata correntemente dalla piattaforma.

Name	`Map.get`	
Auth level	`User`	
Arguments		
–		
Results		
`map`	`String`	A text representation of the current map
Exceptions		
–		

5.8.2. set

Questa chiamata permette di impostare una nuova mappa, in caso di errore le linee dov'è presente l'errore vengono specificate nel messaggio di exception.

Name	Map.set	
Auth level	User	
Arguments		
map	String	A map in text format
Results		
	Void	
Exceptions		
Map.ParseError		A parse error occurred while parsing the map.

Vi è qui un esempio di una mappa in formato testuale:

```
Bin Navigation.Nodes
    Node id=1000 pose=3.67892872 3.93833403 3.14159265 links=1005 1025 ~
    Node id=1005 pose=1.46986459 3.98183969 3.14159265 links=1000 1010 1015 ~
    Node id=1010 pose=1.64 6.32 1.57079633 links=1005 1020 ~
    Node id=1015 pose=0.94990973 1.6634537 0.78539816 links=1005 ~
    Node id=1020 pose=2.99 7.45 0.00000001 links=1010 ~
    Node id=1025 pose=4.9 4.47 0.00000001 links=1000 ~
    Home node=1000 ~
~
```

Senza preoccuparci della sintassi si può vedere che è strutturata in vari elementi che potrebbero essere nodi, segmenti o muri virtuali tutti identificati da un codice univoco.

5.9. Motion

Il sottosistema Motion è adibito al controllo del movimento del rover e utilizza vari algoritmi. Inoltre fornisce lo stato ed informazioni varie sull'operazione movimento corrente.
La massima velocità lineare consentita è pari a 0.6 m/s mentre la massima velocità angolare è 1.57 m/s .

5.9.1. getSpeed

Questa chiamata restituisce la velocità lineare e di rotazione.

Name	`Motion.getSpeed`	
Auth level	`User`	
Arguments		
–		
Results		

`result`	`Float64[]`	`result[0]`: the time of the request as an absolute UTC time `[s]`
		`result[1]`: the current platform translation speed `[m/s]`
		`result[2]`: the current platform rotation speed `[rad/s]`

Exceptions		
–		

5.9.2. getStatus

“getStatus” restituisce lo stato dell’operazione in corso (null se è in movimento) e il risultato dell'ultima operazione appena terminata.
Per esempio quando il rover è in movimento autonomo lo stato sarà “Driven.Autonomous” o nel caso sia disabilitato “Disabled”.
Il risultato dell’ultima operazione potrebbe essere “Autonomous.Succes”, se andato a buon fine.

Description	**`state`**
Motion is disabled	`Disabled`
Motion controller is ready and not driven	`Ready`
Motion controller is driven in autonomous mode	`Driven.Autonomous`
Motion controller is driven in autonomous mode but platform is temporarily blocked	`Driven.Autonomous.Blocked`
Motion controller is driven in speed control mode, but is disabled (e.g. due to a pushed bumper)	`Disabled.SpeedControl`

Description	**`result`**
Motion operation is in progress	
Autonomous motion operation was successful	`Autonomous.Success`
Autonomous motion operation failed due to a planning error	`Autonomous.PlanError`
Autonomous motion operation was stopped gracefully	`Autonomous.Stopped`
Autonomous motion operation was aborted abnormally	`Autonomous.Aborted`
Current motion operation was stopped abruptly	`Stopped`
Speed control watchdog has triggered	`TimedOut`

5.9.3. moveToPose

Con questa chiamata si inizia un'operazione di movimento autonomo; in input si devono specificare le coordinate globali.
Inoltre vi è un flag per dare la possibilità alla piattaforma di muoversi all'indietro (la cui velocità limitata a 0.1 m/s).

Name	`Motion.moveToPose`	
Auth level	`User`	
Arguments		
`x`	`Float64`	The X coordinate of the pose to move to in world coordinates `[m]`
`y`	`Float64`	The Y coordinate of the pose to move to in world coordinates `[m]`
`theta`	`Float64`	The orientation of the pose to move to in world coordinates `[rad]`
`backward*`	`Boolean`	If false, move forward. If true, move backward. The default is false.
Results		
	`Void`	
Exceptions		
`Motion.Busy`		The motion controller is already in use

5.9.4. turn

La chiamata avvia un'operazione di moto autonomo in cui si fa girare la piattaforma di un angolo relativo o assoluto. Quando si gira di un angolo relativo, viene calcolato l'angolo in modulo 2π, e il senso di rotazione viene impostato in modo tale che sia percorso l'angolo minimo. Non è quindi consentito impostare un angolo relativo maggiore di π.

5.10. Script in Python

Dopo aver descritto tutte le funzioni e i meccanismi dell'environment di LOS, si presentano di seguito esempi di script in python con lo scopo di ottenere informazioni dal rover e di comunicarne comandi di movimento attraverso una connessione LAN di tipo WiFi.
È quindi necessario collegarsi dapprima alla rete SHERPA-WiFi (è sempre buona norma fare un ping di prova ad ANT per essere sicuri che tutto funzioni correttamente) e in seguito si può eseguire il programma in Python con la semplice sintassi di Python che prevede di collocarsi nella cartella dello script e digitare il seguente comando: python nomeScript.py (non è necessario compilare).
Ovviamente si deve installare la libreria "Los.py" che richiede una versione di Python a partire dalla 2.4. In Unix basta fare l'unpack di Los.py andare nel top folder e digitare il seguente comando "python setup.py install".
La libreria di Los è strutturata come un package contenente i seguenti moduli:

- Client (Una classe che incapsula una connessione LOS che può agire come un oggetto proxy per un server remoto)
- Readers (Un readers in grado di leggere i tipi di dati semplici e i corrispondenti tipi di matrice da uno stream)
- Serializers (Le classi che permettono di serializzare tutti i tipi di oggetto)
- Streams (un wrapper che fornisce un interfaccia per una socket)
- Types (La classe wrapper per tutti gli oggetti LOS)
- Writers (Un writer capace di scrivere tipi di dati semplici e i corrispondenti tipi di matrice su uno stream)

La tabella mostra come sono mappati gli oggetti Los nei corrispettivi tipi in Python.
Solo Void, Boolean, Float64, String e Array sono direttamente tipi nativi di Python, tutti gli altri tipi devono essere creati in modo esplicito e devono quindi essere specificati per esempio come Int32(5).

LOS object type	Python type	Inherits from
Void	None	
Boolean	bool	
Boolean[]	Los.Types.BooleanArray	list
Int8	Los.Types.Int8	int
Int8[]	Los.Types.Int8Array	list
Int16	Los.Types.Int16	int
Int16[]	Los.Types.Int16Array	list
Int32	Los.Types.Int32	int
Int32[]	Los.Types.Int32Array	list

Int64	Los.Types.Int64	long
Int64[]	Los.Types.Int64Array	list
Float32	Los.Types.Float32	float
Float32[]	Los.Types.Float32Array	list
Float64	float	
Float64[]	Los.Types.Float64Array	list
String	str	
String[]	Los.Types.StringArray	list
Array	list	
Call	Los.Types.Call	object
CallResult	Los.Types.CallResult	object
CallException	Los.Types.CallException	object
Struct	Los.Types.Struct	dict

5.10.1. Descrizione dello script:

- import delle librerie necessarie

```
2  import time
3  from Los.Client import Connection
4  from Los.Types import *
5  from Los.Util import *
```

- apertura della connessione e login al secondo livello

```
def main():

#**********connection**********
    proxy = Connection(("192.168.8.149", 1234), timeout=2.5)
    proxy.open()
    proxy.ping()
    proxy.login("User", "none")
```

- Le varie funzioni sono state descritte sopra; tuttavia è importante ricordare che affinché il rover vada in navigazione autonoma è importante invocare la funzione

“proxy.configure()”. Si può notare in proxy.Motion() che gli argomenti passati devono essere definiti esplicitamente secondo il modello di Los.

```
proxy.Watchdog.reset(1.0) #info about status
(t, state, result) = proxy.Motion.getStatus()
print "state=%r" % state

time.sleep(3)
#first of all make sure that the current state
#of the robot is in autonomous mode!
proxy.configure(Struct({"Localization.active": False}))
proxy.Motion.turn(Float64(3.14),True)
proxy.Motion.moveToPose(Float64(2.0),Float64(2.5),Float64(0.0))
```

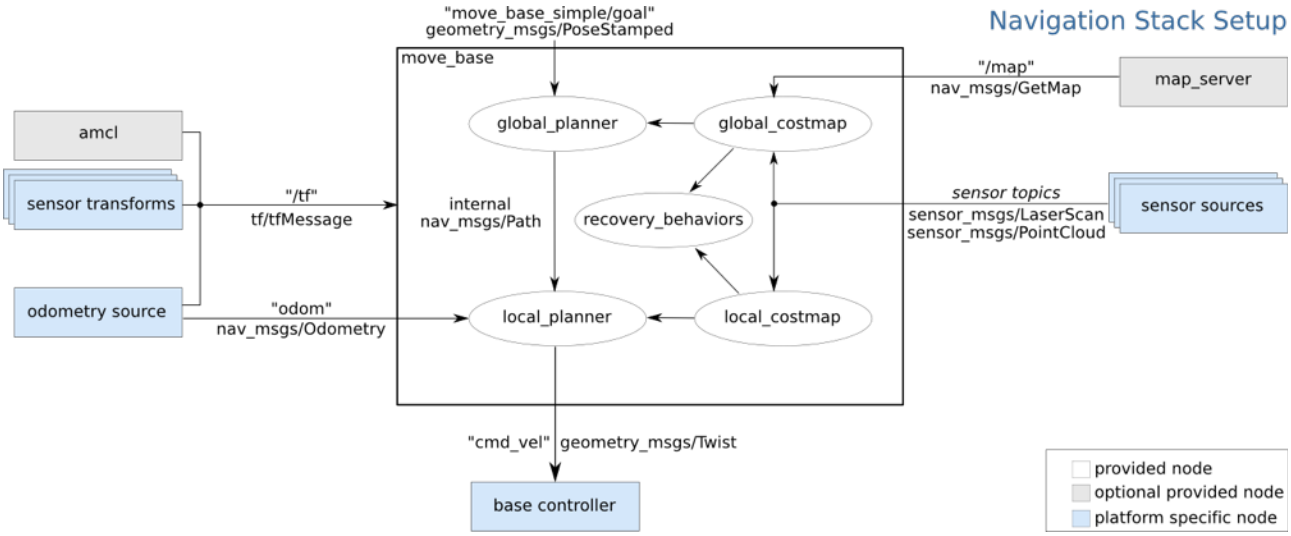
```
1
2  import time
3  from Los.Client import Connection
4  from Los.Types import *
5  from Los.Util import *
6  #
7  #TEMPLATE RemoteProcedureCall OVER LightweightObjectStreaming
8  #
9  def main():
10
11 #****************connection**********************
12       proxy = Connection(("192.168.8.149", 1234), timeout=2.5)
13       proxy.open()
14       proxy.ping()
15       proxy.login("User", "none")
16 #****************set the map*********************
17       try:
18       #proxy.Map.set("")
19       except RemoteException, e :
20           if e.name=="Map.ParseError"
21            print "A parse error occurred while parsing the map."
22 #*******************logic***************************
23
24       #***** example 2: information about the robot
25       print "example 2: information about the robot"
26       time.sleep(3)
27
28
29       while True:
30
31        proxy.Watchdog.reset(1.0)
32        (t, state, result) = proxy.Motion.getStatus()
33        speedInfo = proxy.Motion.getSpeed()
34        print "translation speed=%f" % speedInfo[1]
35        print "rotation speed=%f" % speedInfo[2]
36        print "state=%r" % state
37        print "result=%r" % result
38        print "time=%r" % t
39        time.sleep(1)
40
41 #************Close the connection************
42       proxy.close()
43
44 if __name__ == "__main__":
45     main()
46
```

Questo secondo esempio di script mostra come si ottengo alcune informazioni fondamentali.

```
1
2  import time
3  from Los.Client import Connection
4  from Los.Types import *
5  from Los.Util import *
6  #
7  #TEMPLATE RemoteProcedureCall OVER LightweightObjectStreaming
8  #
9  def main():
10
11 #*****************connection**************************
12       proxy = Connection(("192.168.8.149", 1234), timeout=2.5)
13       proxy.open()
14       proxy.ping()
15       proxy.login("User", "none")
16 #*****************set the map*********************
17       try:
18       #proxy.Map.set("")
19       except RemoteException, e :
20           if e.name=="Map.ParseError"
21            print "A parse error occurred while parsing the map."
22 #*********************logic*****************************
23       #**** example 1: move/turn the robot
24       print "example 1: move/turn the robot."
25       try:
26        proxy.Watchdog.reset(1.0) #info about status
27        (t, state, result) = proxy.Motion.getStatus()
28        print "state=%r" % state
29
30        time.sleep(3)
31        #first of all make sure that the current state
32        #of the robot is in autonomous mode!
33        proxy.configure(Struct({"Localization.active": False}))
34        proxy.Motion.turn(Float64(3.14),True)
35        proxy.Motion.moveToPose(Float64(2.0),Float64(2.5),Float64(0.0))
36
37        proxy.Watchdog.reset(1.0)
38        (t, state, result) = proxy.Motion.getStatus()
39        print "state=%r" % state  # state should be "Driven.Autonomous"
40        print "result=%r" % result
41       except RemoteException, e:  #exception handling
42          if e.name == "Motion.Busy"
43            print "The motion controller is already in use"
44
45 #************Close the connection************
46       proxy.close()
47
48 if __name__ == "__main__":
49     main()
50
```

6. Navigation Stack

Navigation Stack è un package molto corposo e complesso ma che è abbastanza semplice a livello concettuale: prende in ingresso informazioni relative all'odometria e ai sensori di bordo e li traduce in comandi di velocità da inviare ad una piattaforma che in questo caso è il rover Donkey.
L'uso di questo package su un rover, tuttavia, è un po' più complicato. L'architettura ROS è quindi un prerequisito fondamentale per l'utilizzo del Navigazione Stack che prevede un notevole scambio di messaggi tra i nodi.
Non entrando nel dettaglio è di seguito riportato l'architettura del package:



Il rover ha come obiettivo quello di spostarsi in una certa posizione e il Navigation Stack serve a guidarlo da una posizione a un'altra, evitando gli ostacoli in modo sicuro. Questo package si sostituisce completamente alla parte di controllo della navigazione di ANT senza nessuna mancanza. È però necessario eseguire alcuni comandi che inizializzano e acquisiscono la mappa prima di poterci interagire.
Per quanto riguarda il rover Donkey:

- roslaunch donkey_rover rovmap.launch (inizializza la mappa)
- roslaunch donkey_rover gmap.launch (acquisisce la mappa)

Infine si può eseguire il Navigation Stack:

- roslaunch donkey_rover_2dnav move_base.launch

A questo punto si potranno mandare i goal attraverso il seguente codice:

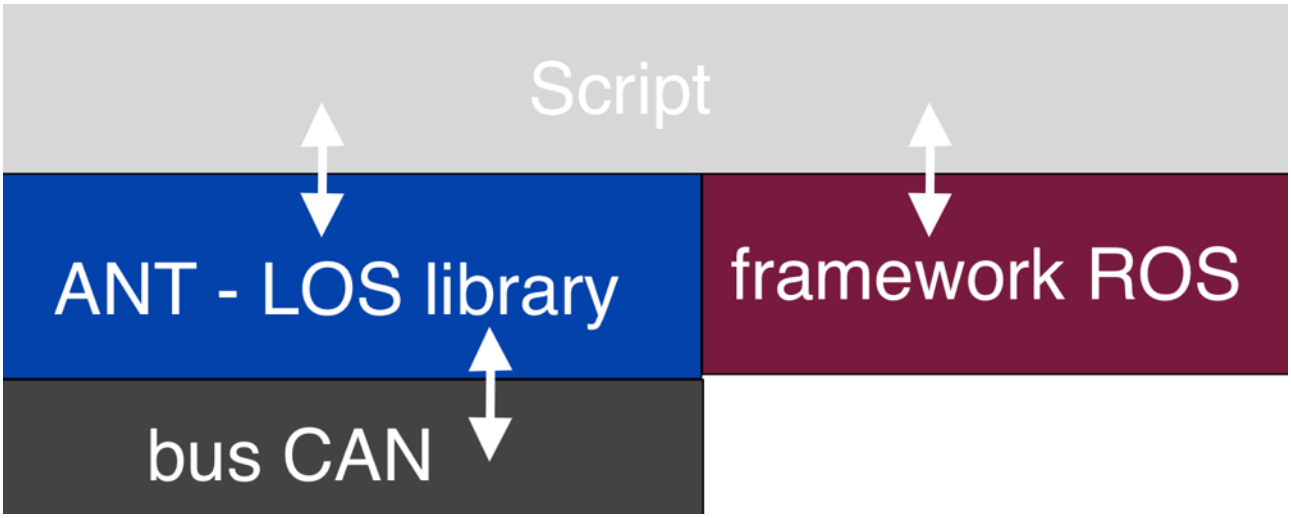
```
rospy.init_node('donkeyGPSListener', anonymous=True)
client = actionlib.SimpleActionClient('move_base', MoveBaseAction)
goal = MoveBaseGoal()


print "goal: x=%0.2f y=%0.2f" % (float(argv[3]),float(argv[4]))
goal.target_pose.pose.position.x = float(argv[3])
goal.target_pose.pose.position.y = float(argv[4])
goal.target_pose.pose.orientation.w=1.0
goal.target_pose.header.frame_id= "map"
print "frame_id= map"
goal.target_pose.header.stamp=rospy.Time.now()
client.wait_for_server()
client.send_goal(goal)
client.wait_for_result()

print "result:"
print client.get_result()
print "state:"
print client.get_state()
```

In questo caso il frame_id è “map” ma ve ne possono essere diversi e cambia il punto definito come origine all’interno della mappa. A seconda di come si vuole interpretare le coordinate si cambia il frame_id; per esempio se si vuole essere solidali con il rover il frame_id sarà “odom” oppure “base_link”. In pratica a seconda del frame_id viene applicata una trasformata diversa.
In ultima analisi si ricorda che le funzioni “wait_for_server()” e “wait_for_result()” sono sospensive.

7. framework LOS/ROS

Dopo aver illustrato la libreria LOS, è possibile comprendere il contenuto dello script che verrà presentato in questo capitolo.
Lo scopo è riuscire ad integrare il framework ROS con LOS, così da sfruttare il codice di basso livello implementato in ANT mantenendo allo stesso tempo l'ambiente di sviluppo necessario.

L'immagine seguente rappresenta la struttura dello script:



Per realizzare tale l'architettura è necessario creare un nodo ROS con all'interno uno script eseguibile. Non vi sono ulteriori dipendenze del package da specificare oltre che al tipo messaggio che vogliamo inviare.

Di seguito il codice dello script:

```python
#!/usr/bin/env python

import rospy
import os.path
import sys
from std_msgs.msg import String
#LOS
import time
from Los.Client import Connection
from Los.Types import *
from Los.Util import *



def talker(argv):


        if len(argv) < 4:
            sys.stderr.write("Usage: %s {X} {Y} {Theta}\n" % os.path.basename(argv[0]))
            return 2


        pub = rospy.Publisher('losTopic', String, queue_size=10)
        rospy.init_node('LOS_talker', anonymous=True)
        rate = rospy.Rate(10)

        proxy = Connection(("192.168.8.149", 1234), timeout=2.5)
        proxy.open()
        proxy.ping()
        proxy.login("User", "none")
        print "ANT is connected!"


        proxy.configure(Struct({"Localization.active": False})
        proxy.Motion.moveToPose(Float64(argv[1]),Float64(argv[2]),Float64(argv[3]),False)


        while not rospy.is_shutdown():

         proxy.Watchdog.reset(1.0)
         (t, state, result) = proxy.Motion.getStatus()
         speedInfo = proxy.Motion.getSpeed()

         stringa =  "%f-%r" % (speedInfo[1],t)

         rospy.loginfo(stringa)
         pub.publish(stringa)
         rate.sleep(1)



 if __name__ == '__main__':
     try:
         talker(sys.argv)
     except rospy.ROSInterruptException:
         pass
```

7.1. Descrizione del codice

Lo scopo di questo script è di pubblicare un topic usufruendo delle informazioni fornite da ANT.
Gli import da effettuare sono ovviamente le librerie del framework ROS e di LOS.
Dalla riga 23 alla 25 si effettua la pubblicazione del topic “losTopic” su cui si andranno a pubblicare i dati.
Dalla riga 27 alla 30 si effettua la connessione ad ANT e successivamente si danno le istruzione di movimento. In seguito quindi se entra nel ciclo while e si pubblica la stringa con i valori della velocità e del tempo.

7.2. Note:

Il nodo ovviamente deve essere inserito nel workspace del PC embedded e quindi,per l’utilizzo e la sperimentazione di questi nodi, sono risultati fondamentali i seguenti due programmi (già presenti nell’abilmente linux) : Putty e Firezilla.
Putty è una shell che utilizza il protocollo ssh. Firezilla utilizzato per scambio e modifica del file system che utilizza protocollo ftp.

8. xsens_mti_ros_node

In questo capitolo si illustra il nodo ROS usato per ottenere i dati di interesse dal dispositivo MTi installato sulla piattaforma.
Il nodo xsens_mti_ros_node è disponibile su github (github.com/xsens/xsens_mti_ros_node) e non serve alcuna serial-key.
Questo pacchetto funziona solo se il firmware del dispositivo MTi è aggiornato alla versione 1.4 . È per questo motivo che è stato utile il software fornito da Xsens FirmwareUpdater citato nel sottoparagrafo 3.10.2 .

8.1. Prerequisiti

Prima di installare il pacchetto è necessario installare tutti i driver per la porta USB:

- git clone https://github.com/xsens/xsens_mt.git
- cd ~/xsens_mt
- make
- sudo modprobe usbserial
- sudo insmod ./xsens_mt.ko

- sudo apt-get install ros-distro-gps-common

Affinché tutto funzioni assicurarsi di dare tutti i diritti di lettura e scrittura alla porta USB attraverso il comando chmod.

8.2. Contenuto pacchetto

Il pacchetto contiene due cartelle principali (oltre al CMakeLists.txt)situate in src: custom_msgs e xsens_driver. Nella prima troviamo la sottocartella msg che contiene la struttura di tutti i messaggi utilizzati nei topics di questo pacchetto.
La figura sottostante mostra l'header dei messaggi usati nell'ambito di questo progetto.

gnssSample.msg

```
# This is a message to hold data a GNSS unit
# Supported for MTi Devices with FW 1.4 and above.

Header header

float64 itow
float64 fix

float64 latitude
float64 longitude
float64 hEll
float64 hMsl

# ENU velocity
geometry_msgs/Vector3 vel

float64 hAcc
float64 vAcc
float64 sAcc

float64 pDop
float64 hDop
float64 vDop

float64 numSat
float64 heading
float64 headingAcc
```

positionEstimate.msg

```
# This is a message to hold filter data
# Supported for MTi Devices with FW 1.4 and above.

Header header

float64 latitude
float64 longitude
float64 hEll
```

orientationEstimate.msg

```
# This is a message to hold filter data
# Supported for MTi Devices with FW 1.4 and above.

Header header

float64 roll
float64 pitch
float64 yaw
```

La sostanziale differenza tra il primo messaggio e gli ultimi due risiede nel fatto che per ottenere i messaggi postionEstimate.msg e orientationEstimate.msg vengono applicati dei filtri, e quindi abbiamo in definitiva un valore molto più stabile (fino alla 6/7 cifra decimale). Il primo messaggio, invece, non ha un valore di latitudine e longitudine stabile; tuttavia fornisce molti più dati sullo stato del MTi.

Nella cartella src, oltre a custom_msgs, c'è anche xsens_driver/nodes dove sono contenuti gli script Python:

- mtdevice.py
- mtnode.py

Eseguendo mtdevice.py vi è la possibilità di configurare il pacchetto al fine di visualizzare alcuni dati piuttosto che altri.
Per dettagli sulla configurazione è possibile digitare il comando "rosrun xsens_driver mtdevice.py -h" :

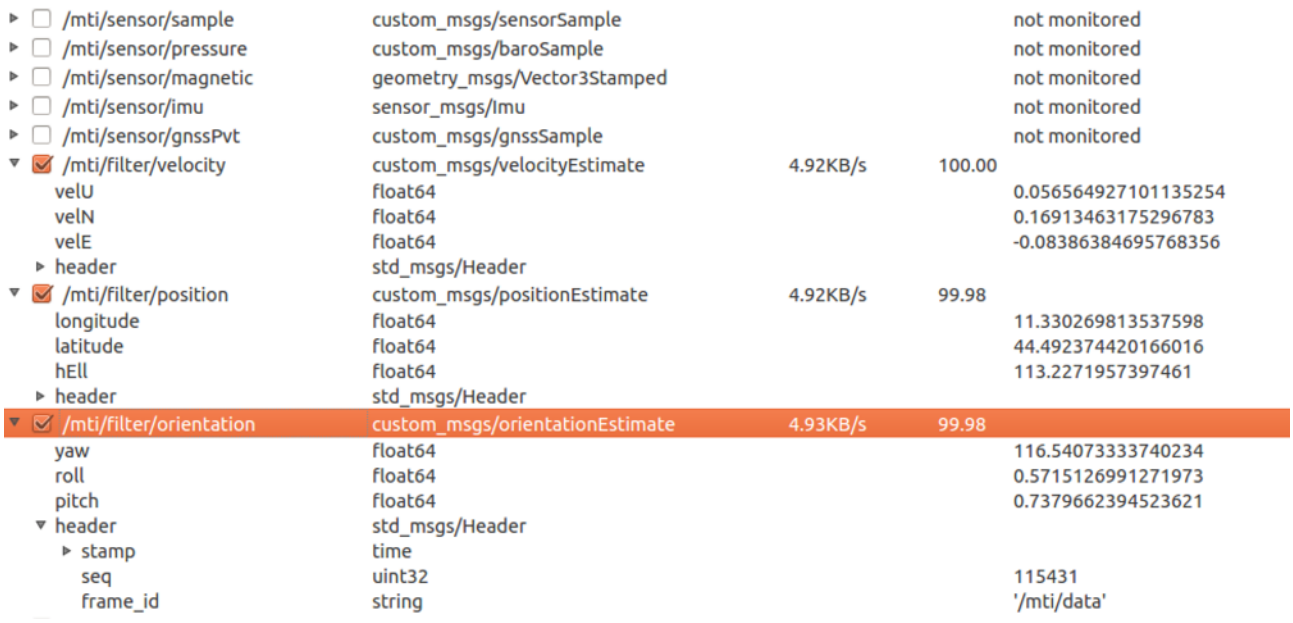
```
Commands:
        -h, --help
                Print this help and quit.
        -r, --reset
                Reset device to factory defaults.
        -f, --mtiSampleRate=SAMPLERATE
                Configures the device to the specified Output Data Rate (ODR).Po
ssible
                ODR's are 1,2,4,5,10,20,40,50,80,100,200 & 400 (the maximum outp
ut rate
                for mag, baro and GNSS sensor is 100Hz, 50Hz and 4Hz respectivel
y)
        -m, --sensorMode=SENSORMODE
                Configures the device to a particular sensor mode. The values ca
n be 1
                (for sensor data),2 (for sensor data with rate quantities) or
                3(for filter estimates). Use it in conjunction with -f command
        -a, --change-baudrate=NEW_BAUD
                Change baudrate from BAUD (see below) to NEW_BAUD.
        -e, --echo
                Print MTData. It is the default if no other command is supplied.
        -i, --inspect
                Print current MT device configuration.
        -x, --xkf-scenario=ID
                Change the current XKF scenario.
Options:
        -d, --device=DEV
                Serial interface of the device (default: /dev/ttyUSB0). If 'auto
', then
                all serial ports are tested at all baudrates and the first
                suitable device is used.
        -b, --baudrate=BAUD
                Baudrate of serial interface (default: 115200). If 0, then all
                rates are tried until a suitable one is found.
        -s, --skip-factor=SKIPFACTOR
                Number of samples to skip before sending MTData2 message
                (default: 0).
                The frequency at which MTData message is send is:
                        115200/(PERIOD * (SKIPFACTOR + 1))
                If the value is 0xffff, no data is send unless a ReqData request
                is made.
```

8.3. Esecuzione pacchetto

Dopo aver installato il pacchetto nel catkin workspace si è pronti per avviare gli eseguibili.
Prima di eseguire mtdevice.py è necessario eseguire il seguente comando ogni qualvolta avviamo la macchina: sudo chmod 777 /dev/ttyUSB0 .
Il passo successivo consiste nell'avviare l'eseguibile mtdevice.py con le relative impostazioni.

```
sherpa@Sherpa1:~$ rosrun xsens_driver mtdevice.py -m 3 -f 100
Timeout defaults to 0.016s based on output settings.
Device intiated at 100 Hz
Timeout changed to 0.016s based on current settings.
MTi-G-700/710 (GNSS/INS) device detected
Enabled publishing all filter estimates
Device configured at 115200 bps
sherpa@Sherpa1:~$ rosrun xsens_driver mtnode.py
[WARN] [WallTime: 1453212781.886324] Cannot find value for parameter: ~device, a
ssigning default: auto
[WARN] [WallTime: 1453212781.888755] Cannot find value for parameter: ~baudrate,
 assigning default: 0
Timeout defaults to 0.016s based on output settings.
[INFO] [WallTime: 1453212903.549676] Detected MT device on port /dev/ttyUSB0 @ 1
15200 bps
[INFO] [WallTime: 1453212903.550491] MT node interface: /dev/ttyUSB0 at 115200 b
d.
Timeout defaults to 0.016s based on output settings.
```

Se diamo il comando -m 1 impostiamo la modalità senza filtri. Con la modalità -m 3 si attiva la modalità con filtri.
Successivamente si dovrà avviare il nodo master (roscore) ed eseguire il mtnode.py, che si occupa di pubblicare tutti i topics .
Infine si possono visualizzare i topic con il comando rqt:



9. donkey_move (parte I)

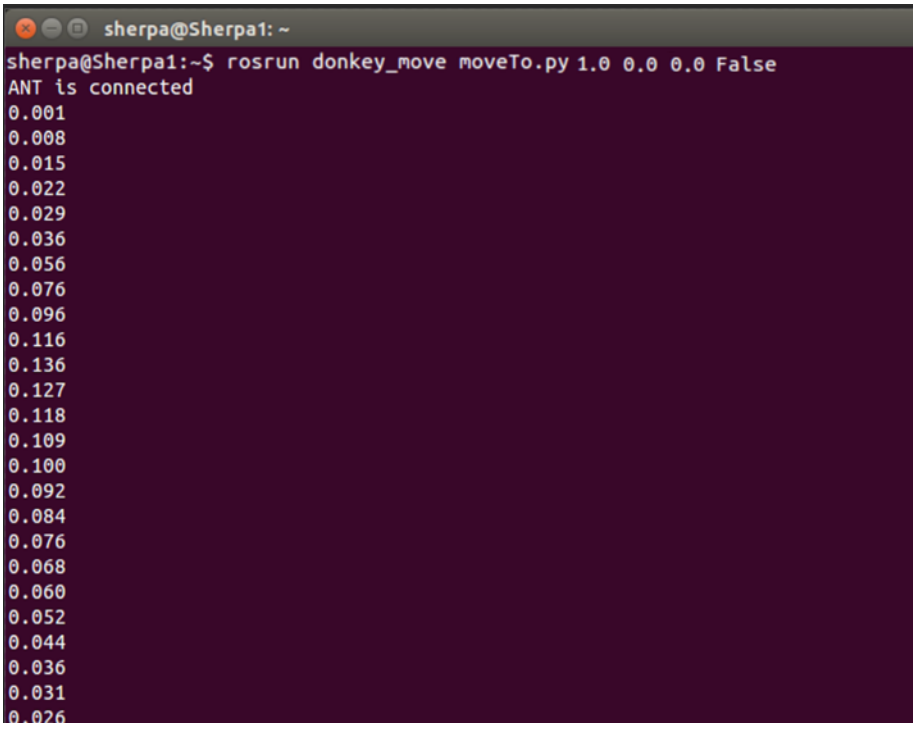
Questo pacchetto è stato creato per contenere uno script Python con la stessa architettura descritta nel capito 5.
Il pacchetto che viene ora descritto non sarà quello finale e si vedranno modifiche sostanziali nei prossimi capitoli. L'eseguibile contenuto nella cartella scripts è moveTo.py .
Si potrebbe dire che questo script non ha nulla a che vedere con ROS, difatti l'unica libreria importata è rospy: non vi è ora la necessità di sottoscriversi ad alcun topic e tantomeno di pubblicarne uno.

L'obiettivo di questo primo nodo è di muovere il rover di una posizione relativa.
Date quindi le coordinate x,y e theta, attraverso il comando moveToPose (riga 40), il rover si muoverà fino ad arrivare all'obbiettivo desiderato.
Inoltre il rover si muoverà in modalità autonoma fino alla posizione relativa facendo obstacle avoidance.

```
1  #!/usr/bin/env python
2
3  import rospy
4  import os.path
5  import sys
6
7  #LOS
8  import time
9  from Los.Client import Connection
10 from Los.Types import *
11 from Los.Util import *
12
13
14
15 def move(argv):
16
17         #check the input
18         if len(argv) < 5:
19             sys.stderr.write("Usage: %s {X} [m] {Y} [m] {Theta} [rad] {Localization.active} [bool]" % os.path.basename(argv[0]))
20             return 2
21
22         if(argv[4]!="True"):
23             if(argv[4]!="False"):
24                       sys.stderr.write("The 4th argument must be False or True \n\n")
25                       return 3
26
27        #*****************connection to ANT************************
28
29         proxy = Connection(("192.168.8.149", 1234), timeout=2.5)
30         proxy.open()
31         proxy.ping()
32         proxy.login("User", "none")
33         print "ANT is connencted."
```

```
33
34          #********************** MOVE TO RELATIVE POSE ****************************
35
36          if(argv[4]=="False")
37              proxy.configure(Struct({"Localization.active": False}))
38
39          proxy.Localization.snapToPose(0.0,0.0,0.0)
40          proxy.Motion.moveToPose(Float64(argv[1]),Float64(argv[2]),Float64(argv[3]),False)
41
42
43          while 1:
44
45           proxy.Watchdog.reset(1.0)
46           (t, state, result) = proxy.Motion.getStatus()
47           speedInfo = proxy.Motion.getSpeed()
48
49           stringa = "%0.3f" % speedInfo[1]
50           time.sleep(0.2)
51           if result :
52                  break
53
54
55
56
57
58  if __name__ == '__main__':
59      try:
60          talker(sys.argv)
61      except rospy.ROSInterruptException:
62          pass
```

Eseguendo il comando *rosrun donkey_move moveTo.py 1.0 0.0 0.0 False* il rover donkey si muoverà di un metro in linea retta evitando, se necessario, gli ostacoli. Si otterrà il seguente output:



10. Algoritmo outdoor

10.1. Navigazione autonoma

Attualmente il rover Donkey ha due possibili algoritmi di navigazione autonoma che attuano l'obstacle avoidance: uno è implementato da ANT e l'altro è stato realizzato all'interno di un altro progetto del team CASY.
Dal punto di vista teorico è possibile sfruttare entrambi i due algoritmi in maniera indistinta. Questi due algoritmi funzionano essenzialmente attraverso una mappa costruita tramite sensori esterni ed interni (laser scanner, motion tracker), necessari per effettuare l' odometria del rover.
Talvolta l'uso di una mappa per la navigazione è limitante, infatti non è possibile dare un obiettivo al di fuori del range della mappa.
Tale range dipende dalla visione che ha il rover e quest'ultima è caratterizzata dalla potenza del laser scanner.
In questo caso il SICK LMS-151 arriva fino a 30 m, una distanza assolutamente sufficiente per la navigazione indoor ma non per quella outdoor.
In dipendenza dal tipo di terreno su cui il rover si muove, insorgono ulteriore complicazioni relative agli algoritmi di basso livello: risulta difficile fare odometria e spesso ne consegue un malfunzionamento che manda il rover in modalità "disperso".

10.2. Algoritmo di alto livello

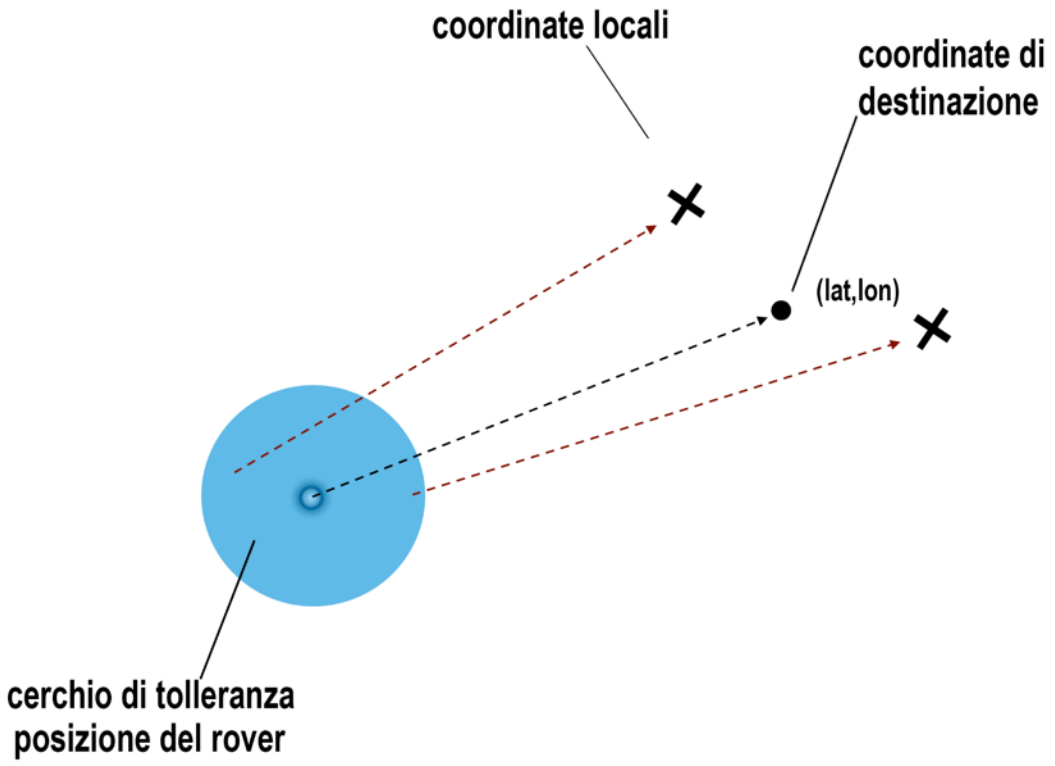
L'algoritmo implementato è di alto livello, cioè si appoggia su algoritmi di navigazione autonoma che necessitano solo dell'input di coordinate locali, calcolate partendo dall'algoritmo di alto livello, i cui input sono latitudine e longitudine.

Un problema non risolvibile è la precisione del segnale GPS che fornisce soltanto una stima della posizione all'interno di un certo range di valori.
A livello operativo la posizione tende a spostarsi nell'ordine dei metri e ciò può essere un vero e proprio ostacolo, soprattutto se si considerano le piccole distanze.
Il cerchio di tolleranza però non è costante: quando il rover è fermo questo è più grande, mentre quando è in movimento risulta essere molto più affidabile.
Altri fattori in gioco sono i filtri applicati all'MTi che in definitiva forniscono un valore decisamente più stabile.

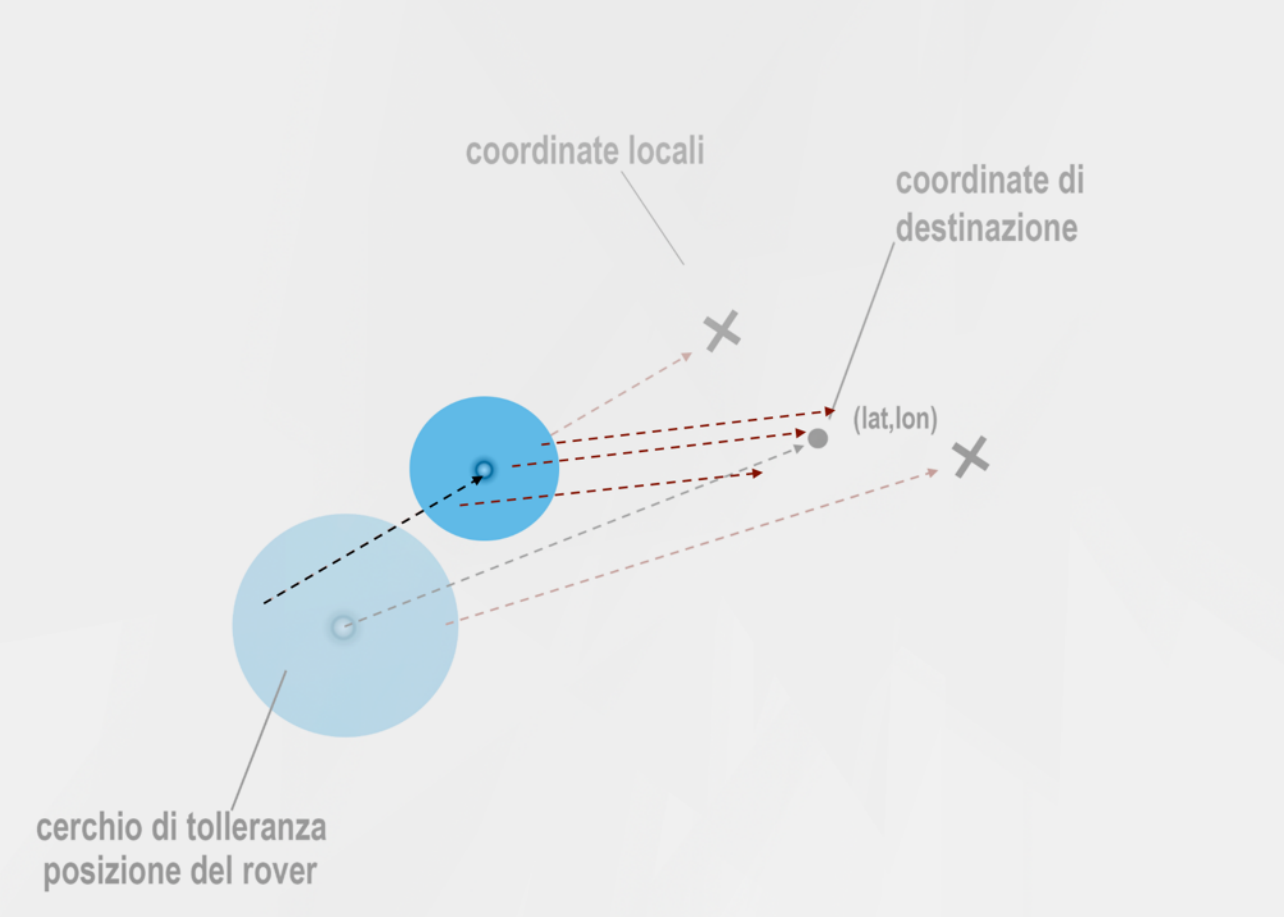
L'algoritmo prevede che in input sia data la posizione (latitudine e longitudine) di destinazione. Considerando anche la posizione corrente è possibile stimare le coordinate locali da fornire in input agli altri algoritmi di navigazione autonoma.
Se la posizione è al di fuori della mappa è necessario far muovere la piattaforma in quella direzione fino a un certo valore di saturazione.
Una volta che il rover è in movimento, è evidente la necessità di applicare dei filtri che ricalcolino la posizione di destinazione; infatti la prima posizione stimata dal MTi non sarà mai affidabile e si deve quindi ricalcolare dinamicamente le coordinate locali di destinazione da fornire in ingresso agli algoritmi di basso livello.

È fondamentale sottolineare che l'algoritmo implementato si appoggi e si affidi ad algoritmi di basso livello; tuttavia ne astrae il comportamento e in definita risulta essere portabile.
Potranno essere utilizzati diversi algoritmi, ad esempio uno che non necessiti della mappa.

Nelle immagini seguenti viene illustrato graficamente il comportamento dell'algoritmo:

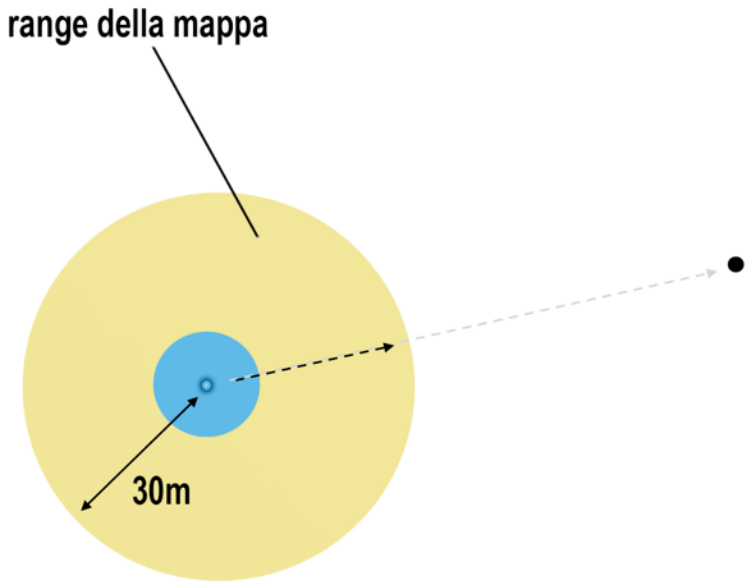


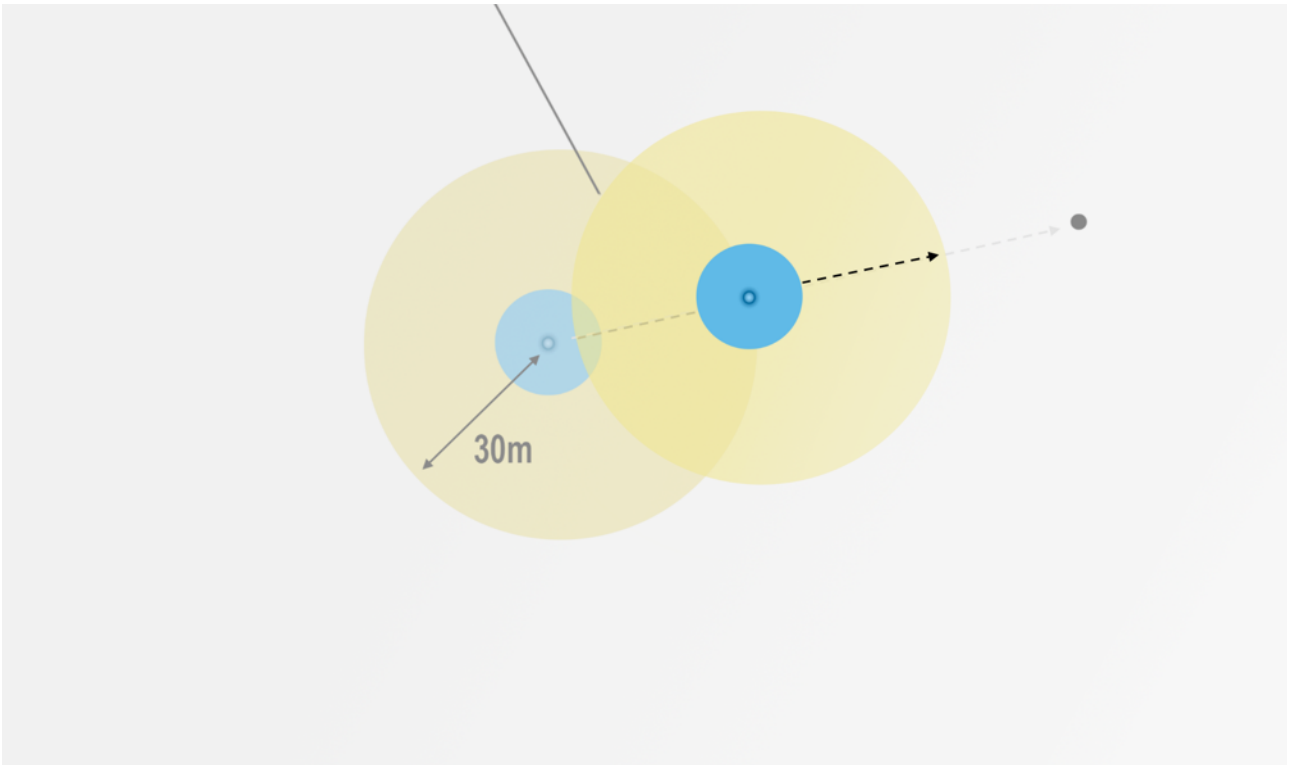
Si può constatare che l'entità dell'errore delle coordinate locali è proporzionale alla grandezza del cerchio di tolleranza.
Quando il rover comincia a muoversi, il cerchio si restringe e viene ricalcolata la destinazione in coordinate locali che, come si può notare, sono più vicine alla posizione di destinazione.



Questo può essere ripetuto più e più volte fino al raggiungimento della meta, cioè fino a quando si avvicina nel raggio desiderato.

È esemplificato di seguito il caso in cui la posizione di destinazione è al di fuori della mappa.





Grazie a questo algoritmo che è in grado di aumentare la distanza dell'obiettivo, si aumenta anche la potenzialità del rover Donkey, nonostante la limitatezza fisica del laser di bordo.

10.3. Calcolo delle coordinate locali

Per il calcolo delle coordinate locali è necessario determinare la propria posizione attraverso l'MTi-G.
Altri importanti parametri del sistema di coordinate descritto nel capitolo 3.6 sono:

- il semiasse maggiore $\mathbf{R_{ea}}$
- prima eccentricità **e** *(=0,081)*
- raggio di curvatura meridiana $\mathbf{M_E}$
- raggio di curvatura (verticale primo) $\mathbf{N_E}$

$$M_{\mathrm{E}} = \frac{R_{\mathrm{Ea}}(1-\mathbf{e}^2)}{(1-\mathbf{e}^2\sin^2\boldsymbol{\varphi})^{3/2}}, \qquad N_{\mathrm{E}} = \frac{R_{\mathrm{Ea}}}{\sqrt{1-\mathbf{e}^2\sin^2\boldsymbol{\varphi}}}.$$

dove *ϕ, λ, h,* sono rispettivamente latitudine, longitudine e altitudine; queste vengono trasformate in coordinate locali, come descritto nel capitolo (3.6).
Quindi:

$$\begin{cases} y = (h+N_E)\cdot \Delta\lambda \cdot \cos(\phi) \\ x = (h+M_E)\cdot \Delta\phi \end{cases}$$

11. *donkey_move (parte II)*

11.1. Implementazione dell'algoritmo di alto livello outdoor

Dopo tutte le informazioni teoriche si può ora procedere con la parte descrittiva del codice. Il package è sempre lo stesso del capitolo 9; lo si modificherà al fine di: integrare i dati GPS, implementare la logica dell'algoritmo e mandare i comandi di movimento all'algoritmo di basso livello.

Si ricorda che sarebbe possibile sfruttare in maniera quasi indistinta entrambi gli algoritmi di basso livello; tuttavia per questo algoritmo vi è un problema nell'usare ANT. Ricalcolando dinamicamente le coordinate si invieranno più richieste di movimento che quindi si andranno ad accavallare. Questo ANT non consente di farlo; si può aggirare il problema dandogli un comando di stop ogni qualvolta si deve interrompere un comando per darne uno nuovo. Tuttavia questo si riflette con una dinamica poco fluida.
Per questo motivo si è preferito usare l'altro algoritmo (Navigation Stack) adattato dal team CASY sul rover Donkey che consente di inviare dati in maniera dinamica ad un service del framework ROS.
Tale meccanismo fornisce una guida fluida al rover ed inoltre, cosa non meno importante, è del tutto integrata con ROS.

11.2. Librerie

Le librerie di cui si necessitano sono elencate qui di sotto:

```
#!/usr/bin/env python

#Python
import rospy
import os.path
import sys
import threading
import math
import time

#Navigation
import actionlib
from move_base_msgs.msg import *
from actionlib_msgs.msg import *

#GPS
from custom_msgs.msg import gnssSample
from custom_msgs.msg import positionEstimate
from custom_msgs.msg import orientationEstimate

```

La libreria actionlib e i relativi due messaggi sono utilizzati per il funzionamento dell'architettura di basso livello livello. I messaggi contenuti in custom_msgs.msg sono gli stessi visti nel capitolo 6.

11.3. Variabili globali

Al fine di poter aggiornare i dati in tempo reale si necessitano di alcune variabile globali che vengono modificate dalle callback in funzione dei dati pubblicati nei topics dal nodo xsens_mti_ros_node.
Le variabili sono:

- latitudine (lat)
- longitudine (lon)
- altitudine (h)
- angolo di yaw (yaw)

```
lat=0.0
lon=0.0
h=0.0
targetLat=0.0
targetLon=0.0
yaw =0.0
distance=0.0
x=0.0
y=0.0
sat=3.0
bearing=0.0
bearing0=0.0
earth_radius=6378137
earth_ecc=0.08181919
threshold=1
```

- distanza dall'obbiettivo (distance)
- coordinate locali (x,y)
- valore di saturazione della distanza (sat)
- angolo di bearing (bearing)
- angolo di bearing precedente (bearing0)
- circonferenza della soglia di arrivo (threshold)

11.4. Funzioni di callback

Le funzioni callback vengono invocate ogni qualvolta è pubblicato un messaggio all'interno di un topic a cui il nodo si è sottoscritto. Per come è strutturato il nodo xsens_mti_ros_node l'eseguibile deve essere subscriber di due topics: mti/filter/orientation e mti/filter/position.

```
40  def callbackPosition(data):
41      global lat,lon,h
42      lat=data.latitude
43      lon=data.longitude
44      h=data.hEll
45      print "callBack: latitude: %0.7f longitude: %0.7f h: %0.3f\n" % (lat,lon,h)
46
47  def callbackOrientation(data):
48       global yaw
49       yaw= data.yaw*math.pi/180 #rad
50       print "yaw: %0.2f" % yaw
51
```

Come si può notare le callback non fanno altro che aggiornare i dati forniti dall'MTi.

11.5. MoveTo

Dopo aver fatto i dovuti controlli sugli argomenti (saltando la parte di algoritmo indoor che viene spiegata nel capitolo successivo) si entra nel cuore dell'algoritmo; come accennato nel paragrafo precedente prima di tutto ci si deve iscrivere ai

```
118 def moveTo(argv):
119
120 # #*************************check the arguments*********************
121     if len(argv) < 5:
122         sys.stderr.write("Usage: %s {GPS} [bool] {X} [m] {Y} [m] {Theta} [rad] or %s {GPS} [bool] {
123         return 2
124     if(argv[1]!="True" and argv[1]!="False"):
125         sys.stderr.write("The 1th argument must be False or True \n\n")
126         return 3
127     if(type(argv[2])!=float or type(argv[3])!=float or type(argv[4])!=float):
128         sys.stderr.write("Not a floating point")
129
130
```

```
177     elif(argv[1]=="True") :
178  #*********************************MOVE TO (GPS)*****************************************
179      global distance,x,y,theta,bearing,bearing0,sat
180
181      rospy.init_node('donkeyGPSListener', anonymous=True)
182      #rospy.Subscriber("mti/sensor/gnssPvt", gnssSample, callback)
183      rospy.Subscriber("mti/filter/orientation",orientationEstimate,callbackOrientation)
184      rospy.Subscriber("mti/filter/position",positionEstimate,callbackPosition)
185
```

topics di interesse (riga 187-188).
È necessario attendere un paio di secondi affinché i dati vengano aggiornati, perciò si invoca la funzione time.sleep().
Come descritto nel capitolo 3 si calcolano le coordinate locali, partendo da M_E ed N_E:

```
187      time.sleep(4)
188     #calcolo le coordinate locali:
189
190      print "\nprima posizione stimata:\nlat=%0.7f lon=%0.7f h=%0.3f\n" % (lat,lon,h)
191
192      me = earth_radius*(1-earth_ecc*earth_ecc)/((1-earth_ecc*earth_ecc*math.sin(lat)*math.sin(lat))**1.5)
193      ne = earth_radius/((1-earth_ecc*earth_ecc*math.sin(lat)*math.sin(lat))**0.5)
194      print "me= %0.6f\n" % me
195      print "ne= %0.6f\n" % ne
```

Si inizializzano le coordinate di destinazione in input trasformando in radianti e si calcola x1 e y1 esattamente come già spiegato.

```
198       targetLat=float(argv[3])
199       targetLon=float(argv[4])
200       y1=((float(argv[4])* math.pi/180.0)-(lon* math.pi/180.0))*(ne+h)*math.cos(lat* math.pi/180.0)
201       x1=((float(argv[3])* math.pi/180.0)-(lat* math.pi/180.0))*(me+h)
202       print "x1=%0.3f"%x1
203       print "y1=%0.3f"%y1
204       print "yaw=%.2f"%yaw
205       x=x1*math.cos(-yaw) - y1*math.sin(-yaw)
206       y=x1*math.sin(-yaw) + y1*math.cos(-yaw)
207       print "CALCOLO ESEGUITO x: %f y: %f \n" % (x,y)
208
```

A riga 205/206 si applica una rotazione tenendo conto dello yaw.
Si calcola la direzione[-∏,∏],la distanza e il bearing:

```
208
209       theta=math.atan2(y,x)
210       bearing0=math.atan2(y,x)
211       distance = math.sqrt(x*x + y*y)
212       print "distance %0.3f" % distance
213       print "bearing %0.3f" % bearing0
214
```

Se la distanza fosse maggiore del valore di saturazione è necessario ridurla:

```
214
215       if(distance>sat):
216          distanceSat = distance/abs(distance) * sat
217          x= distanceSat*math.cos(bearing0)
218          y= distanceSat*math.sin(bearing0)
219          print "Saturazione x: %f y: %f" % (x,y)
220
```

A questo punto si possono inviare i goal al Navigation Stack e far partire il thread che aggiornerà in tempo reale le coordinate. Si sottolinea il frame_id posto uguale a “base_link”; questo significa che le coordinate andranno interpretate avendo come punto di origine il rover stesso.

```
212     try :
213
214             rospy.init_node('donkeyGPSListener', anonymous=True)
215             #Thread
216             targetUpdate = threading.Thread(target=updateTarget, args=())
217             targetUpdate.setDaemon(True)
218             client = actionlib.SimpleActionClient('move_base', MoveBaseAction)
219             goal = MoveBaseGoal()
220
221             goal.target_pose.pose.position.x = x
222             goal.target_pose.pose.position.y =y
223             goal.target_pose.pose.orientation.w=1.0
224             goal.target_pose.header.frame_id= "base_link"
225             goal.target_pose.header.stamp=rospy.Time.now()
226             client.wait_for_server()
227             client.send_goal(goal)
228
229             updateTarget.start()
230             print "thread started"
231
232
233
234     except Exception, e:
235        print str(e)
236
```

11.6. Thread Update Target

Il nome del thread è updateTarget ed è definito di seguito:

```
def updateTarget():
    global distance,x,y,theta,bearing,bearing0,targetLat,targetLon


    print distance

    while (distance > threshold)) :

            time.sleep(2)
            #ricalcolo destinazione :

            print "lat=%0.7f lon=%0.7f h=%0.3f" % (lat,lon,h)
            me = earth_radius*(1-earth_ecc*earth_ecc)/((1-earth_ecc*earth_ecc*math.sin(lat)*math.sin(lat))**1.5)
            ne = earth_radius/((1-earth_ecc*earth_ecc*math.sin(lat)*math.sin(lat))**0.5)

            y1=((targetLon* math.pi/180.0)-(lon* math.pi/180.0))*(ne+h)*math.cos(lat* math.pi/180.0)
            x1=((targetLat* math.pi/180.0)-(lat* math.pi/180.0))*(me+h)
            #theta=float(argv[5])
            #direzione:
            x=x1*math.cos(-yaw) - y1*math.sin(-yaw) #rotazione piano
            y=x1*math.sin(-yaw) + y1*math.cos(-yaw)

            print "CALCOLO ESEGUITO x: %f y: %f" % (x,y)

            #getDistance modulo:
            distance = math.sqrt(x*x + y*y)
            print "distance %0.3f" % distance
            bearing = math.atan2(y,x) - bearing0
            print "bearing %0.3f\n" % bearing

            if(distance>sat):
               distanceSat = distance/abs(distance) * sat
               x= distanceSat*math.cos(bearing)
               y= distanceSat*math.sin(bearing)
               print "Saturazione x: %f y: %f" % (x,y)


            if(distance>threshold) and not rospy.is_shutdown():
              print "target Update"
              bearing0=math.atan2(y,x)
              client = actionlib.SimpleActionClient('move_base', MoveBaseAction)
              goal = MoveBaseGoal()
              goal.target_pose.pose.position.x = x
              goal.target_pose.pose.position.y =y
              goal.target_pose.pose.orientation.w=1.0
              goal.target_pose.header.frame_id= "base_link"
              goal.target_pose.header.stamp=rospy.Time.now()
              client.wait_for_server()
              client.send_goal(goal)
```

Il thread entra in un ciclo in cui valuta se la distanza è maggiore della threshold e calcola le coordinate che, a questo punto, dovrebbero essere più accurate perché il rover dovrebbe essere già in movimento. Si arriva fino a riga 90 con gli stessi passaggi già spiegati. Infine si rivaluta la distanza dal target e se questa è maggiore si inviano i goal al Navigation Stack.

11.7. Conclusioni

Dall'immagine riportata di seguito si può vedere come il primo calcolo delle coordinate (1) sia di gran lunga impreciso rispetto alla reale posizione. Entrando nel ciclo e ricalcolando la posizione quando il rover è in movimento si nota che la pozione tende ad assumere il valore più realistico (2, 3) fino a convergere ad un valore ottimale (5,6).

```
sherpa@Sherpa1:~$ rosrun donkey_move moveTo.py True 44.4926007 11.3301258

prima posizione stimata:
lat=44.4928351 lon=11.3300422 h=113.227

me= 6366814.022619
ne= 6385460.575701

x1=-26.047
y1=6.646
yaw=116.54 [deg]
CALCOLO ESEGUITO x: 22.895023 [m] y: 14.087568 [m]
distance 26.882 [m]
bearing 0.552 [rad]

---thread started---

lat=44.4926997 lon=11.3299902 h=113.227
CALCOLO ESEGUITO x: -10.973336 [m] y: 10.776612 [m]
distance 15.380 [m]
bearing 1.814 [rad]

target Updated


lat=44.4926331 lon=11.3300542 h=113.227
CALCOLO ESEGUITO x: -3.591271 [m] y: 5.690311 [m]
distance 6.729 [m]
bearing -0.231 [rad]

target Updated


lat=44.4926035 lon=11.3301049 h=113.227
CALCOLO ESEGUITO x: -0.310357 [m] y: 1.660999 [m]
distance 1.690 [m]
bearing -0.378 [rad]

target Updated


lat=44.4926011 lon=11.3301191 h=113.227
CALCOLO ESEGUITO x: -0.044337 [m] y: 0.532473 [m]
distance 0.534 [m]
bearing -0.102 [rad]
```

12. donkey_move (parte III)

12.1. Implementazione dell'algoritmo di alto livello indoor

Non essendo possibile sfruttare i sensori GPS all'interno di un edificio è stato implementato un possibile utilizzo del nodo donkey_move che non fa uso dell'MTi.
L'algoritmo prevede l'utilizzo di una mappa ed è quindi necessario che il rover abbia ricostruito l'ambiente circostante prima che gli venga dato l'input.
L'input sono una o più coordinate locali (x,y) con l'origine che in questo caso non è riferita al rover, ma al primo punto in cui il rover ha cominciato a registrare i dati forniti dai sensori di bordo.

Anche in questo caso è possibile implementare l'algoritmo sfruttando ANT o Navigation Stack; dovendo utilizzare quest'ultimo per l'algoritmo outdoor si è scelto di usare lo stesso per quello indoor. Si ricorda però che non è assolutamente vincolante l'uso di ANT, come lo è per la logica dell'algoritmo outdoor.
Nelle prime fasi di debug è stato sempre utilizzato ANT.

È molto utile, o addirittura indispensabile, usare il programma Rviz per avere un idea sulla correttezza dei waypoint che gli vengono dati. Questo perché se si da una coordinata all'interno della mappa che si trovo proprio in un punto limite; gli algoritmi di basso livello dichiarano quel waypoint come irraggiungibile; facendo abortire la procedure lanciando un eccezione "It been not possible to calculate a route" .

È di seguito riportato la parte del codice moveTo.py in cui si implementa l'algoritmo indoor.

```
132     if(argv[1]=="False") :
133 # #*********************MOVE TO (WP in the map)*************************
134 
135         rospy.init_node('donkeyGPSListener', anonymous=True)
136         client = actionlib.SimpleActionClient('move_base', MoveBaseAction)
137         goal = MoveBaseGoal()
138 
139         wp_num = (len(argv)-3)/3
140         print "goal: x=%0.2f y=%0.2f" % (float(argv[3]),float(argv[4]))
141         goal.target_pose.pose.position.x = float(argv[3])
142         goal.target_pose.pose.position.y = float(argv[4])
143         goal.target_pose.pose.orientation.w=1.0
144         goal.target_pose.header.frame_id= "map"
145         print "frame_id= map"
146         goal.target_pose.header.stamp=rospy.Time.now()
147         client.wait_for_server()
148         client.send_goal(goal)
149         client.wait_for_result()
150 
151         print "result:"
152         print client.get_result()
153         print "state:"
154         print client.get_state()
155 
156         wp_num=wp_num -1
157 
158 
```

Da notare a riga 144 frame_id =“map”; l'origine della coordinate nella mappa non è il rover stesso ma il primo punto in cui si trova il rover alla inizializzazione della mappa.

Considerando la possibilità di avere più waypoint:

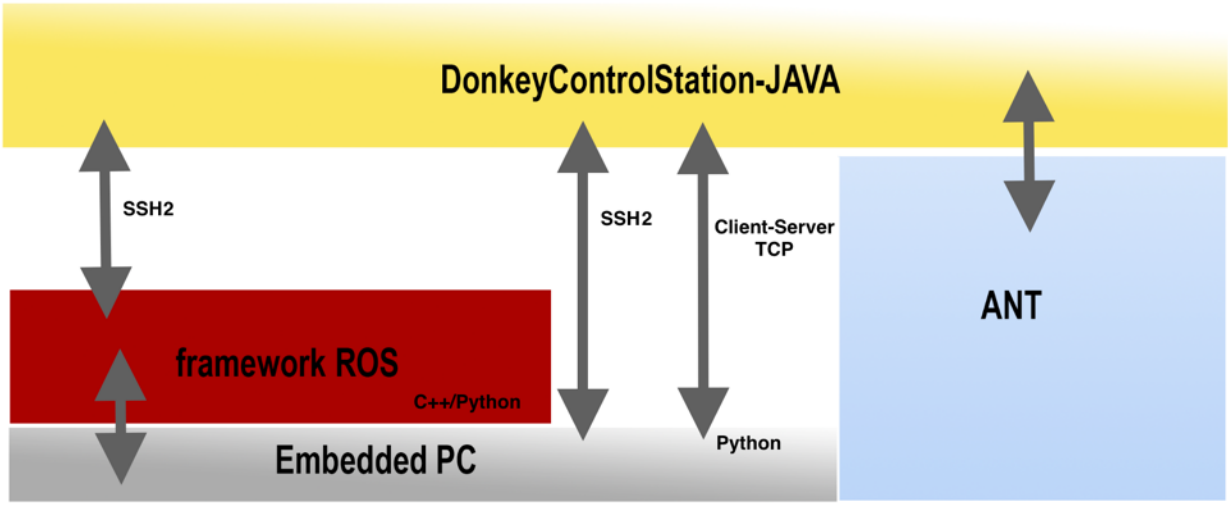
```
158
159         while(wp_num!=0) :
160
161
162           if(client.get_state()==3):
163
164              goal.target_pose.pose.position.x = float(argv[3+3*wp_num])
165              goal.target_pose.pose.position.y = float(argv[4+3*wp_num])
166              goal.target_pose.pose.orientation.w=1.0
167              goal.target_pose.header.frame_id= "map"
168              goal.target_pose.header.stamp=rospy.Time.now()
169
170              client.wait_for_server()
171
172              client.send_goal(goal)
173              client.wait_for_result()
174
175              print client.get_result()
176              wp_num=wp_num -1
177
178           else :
179              print client.get_state()
180
```

13. Graphical User Interface: Donkey Control Station

Per rendere tutto ciò che è stato realizzato utilizzabile da un utente medio si è scelto di implementare una stazione di controllo del rover Donkey che potesse appunto nascondere tutto il codice di più basso o medio livello.
L'idea di base è di permettere all'utente di inserire le coordinate di destinazione, selezionare se sono esterne o locali, eseguire il comando attraverso un apposito comando e infine visualizzarne l'output per verificare che sia andato tutto a buon fine.

La complessità dell'applicazione non è a livello logico, bensì risiede nella gestione e nella coordinazione di tutto il sistema implementato e dei vari nodi che si sono utilizzati. Infatti la Donkey Control Station (DCS) prevede l'utilizzo del nodo donkey_move in tutte e due le modalità implementate: indoor e outdoor. Da quest'ultima derivano le maggiori complicazione.

L'immagine che segue descrive l'architettura dell'applicazione: prevede l'utilizzo di connessioni che sfruttano il protocollo SSH e di una interazione client-server con connessione TCP attraverso socket.
A livello di questa applicazione si usa solo Java mentre ai livelli più bassi C++ e Python.



Si è scelto di utilizzare Java per la realizzazione di questa interfaccia poiché è eventualmente possibile integrare la libreria LOS. Da come è mostrato nella figura precedente questa libreria sostituisce e nasconde tutta l'architettura; quindi solleva da un grosso incarico il programmatore che non dovrà pertanto gestire nessuna connessione. Tuttavia la Control Station, che è stata in ultimo implementata, non ne prevede l'utilizzo.

13.1. Librerie

Le librerie che sono state aggiunte sono:
- ganymed-ssh2-build210.jar (libreria che implementano la connessione con protocollo SSH)
- los-1.4.jar (libreria LOS)

Non vi è un motivo particolare per quanto riguarda la scelta della libreria SSH; si tratta di una libreria molto comune utilizzata da svariati utenti.

13.2. Protocollo SSH2

Per poter gestire il computer di bordo attraverso un PC qualsiasi, previa connessione alla rete wifi del rover Donkey, è necessario eseguire comandi da remoto: il Secure SHell permette di stabilire una sessione remota cifrata.

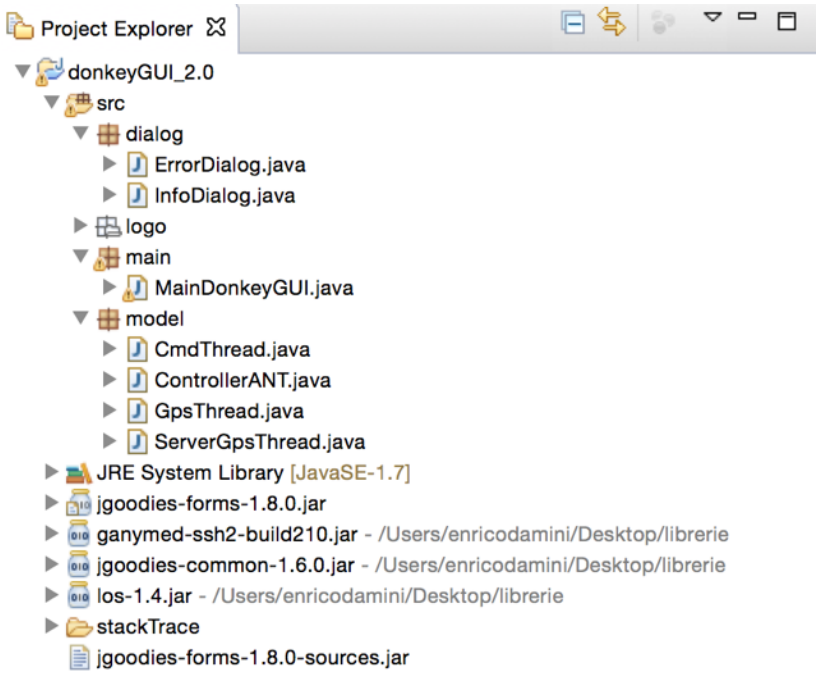
SSH fornisce un canale criptato per la registrazione di un computer all'interno di una rete; quindi offre la possibilità di eseguire comandi su un computer remoto e di spostare file da un computer ad un altro. Inoltre in SSH c'è un forte accoppiamento host-to-host per l'autenticazione degli utenti e la gestione della comunicazione.

SSH2 è una versione più sicura, efficiente, e portatile di SSH (che include SFTP).

Un problema che si è presentato durante la progettazione è l'impossibilità di eseguire diversi comandi in una singola sessione: questa non è una restrizione della libreria ma del protocollo SSH2.
Vi sono diverse soluzioni:
- eseguire diversi comandi incapsulandoli in uno (Session.execCommand("cd Desktop;ls")
- aprire una nuova sessione per ogni comando
- avviare una shell (Session.startShell())

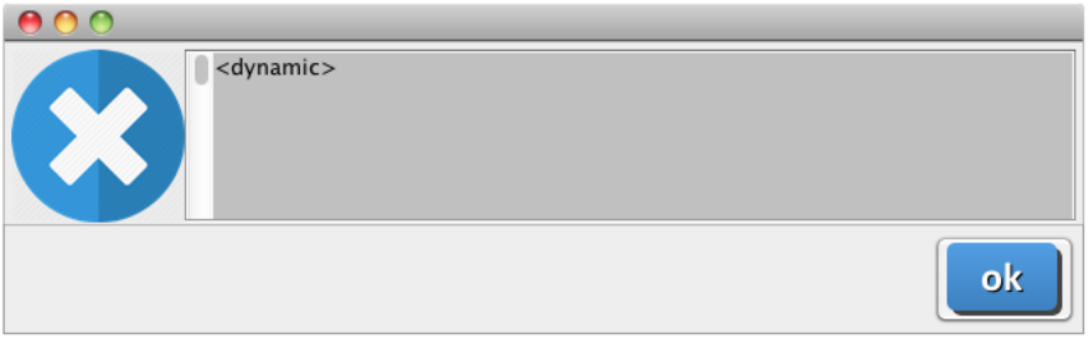
13.3. Descrizione delle classi

Il progetto è suddiviso in tre package: dialog, main e model:



13.4. ErrorDialog e InfoDialog

Nel primo package dialog vi sono le finestre di dialogo; ErrorDialog destinata alla visualizzazione di eventuali errori, mentre InfoDialog al fine di catturare lo stdOutput e stdError dei comandi inviate tramite SSH2. L'ultima è una comodità per il debug.



13.5. CmdThread

Questa classe ha il compito di prendere in ingresso un comando da far eseguire sul PC di bordo, prendersi carico della sua esecuzione e di intercettarne lo stdOutput e stdErr.

```
package model;


import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;

import ch.ethz.ssh2.Connection;
import ch.ethz.ssh2.Session;
import ch.ethz.ssh2.StreamGobbler;
import dialog.ErrorDialog;
```

La classe estende Thread e implementa Runnable; alcuni comandi sono sospensivi e la loro esecuzione deve partire ma mai arrestarsi, se non quando si chiude il programma. Un esempio è il nodo master di ROS, roscore, che deve vivere necessariamente. È la classe chiamante che si occupa di chiudere la connessione nel momento opportuno.

```
public class CmdThread extends Thread implements Runnable{

    private String user = "sherpa";
    private String password = "none";
    private String host = "192.168.8.100";
    private Connection conn;
    private String command;

    private String output;
    private String outputErr;
    private boolean ok=false;

    public CmdThread() {
        super();
        conn = new Connection(host);
        try
        {
            conn.connect();
            conn.authenticateWithPassword(user, password);
        }
        catch (IOException e)
        {
            new ErrorDialog("CmdThread:"+e.toString());
        }
    }
```

```
ublic void setCommand(String command){

    this.command=command;
}

ublic String getStdOut(){

    if(ok)
    return output;
    else return null;

}


ublic String getStdErr(){

    if(ok)
    return outputErr;
    else return null;

}

ublic void close(){

     conn.close();
}
```

Vi sono poi funzioni ausiliari vengono tutte invocate dalla classe chiamante.
Il metodo override run esegue il comando (che dev'essere prima impostato) e inizializza le due stringhe che vengono poi restituite attraverso i due metodi getStdOut e getStdErr.

Se si fosse chiusa la connessione nel finally all'interno del run e il comando da eseguire fosse sospensivo (ad esempio roscore) allora sarebbe impossibile eseguire altri comandi come rosrun che di fatto necessita del nodo master.
Di fatto sarebbe l'equivalente di aprire un terminale nel PC embedded; digitare roscore e subito dopo chiudere la finestra.

```
@Override
public void run(){

    Session sess = null;
    try{

        sess = conn.openSession();
        sess.execCommand(this.command);

        InputStream stdout = new StreamGobbler(sess.getStdout());
        BufferedReader br = new BufferedReader(new InputStreamReader(stdout));
        InputStream stderr = new StreamGobbler(sess.getStderr());
        BufferedReader br1 = new BufferedReader(new InputStreamReader(stderr));

        while (true) {
            String line = br.readLine();
            if (line == null) break;
            output+=line;

        }

        br.close();

        while (true) {
            String line2 = br1.readLine();
            if (line2 == null) break;
            outputErr+=line2;
        }

        br1.close();

        } catch (Exception ex) {
            new ErrorDialog("command: "+command+" \n"+ex.toString());

        }finally{
            ok=true;
            sess.close();
        }
}
```

13.6. ControllerANT

Questa classe implementa l'inizializzazione di ANT ed è stata realizzata secondo il pattern Singleton, poiché la sua istanza dev'essere unica all'interno dell'intero programma.
Le librerie importate sono:

```
package model;

import java.io.IOException;
import com.bluebotics.los.proxy.AntPlatform;
```

Il costruttore provvede ad inizializzare una ed una sola istanza di AntPlatform.

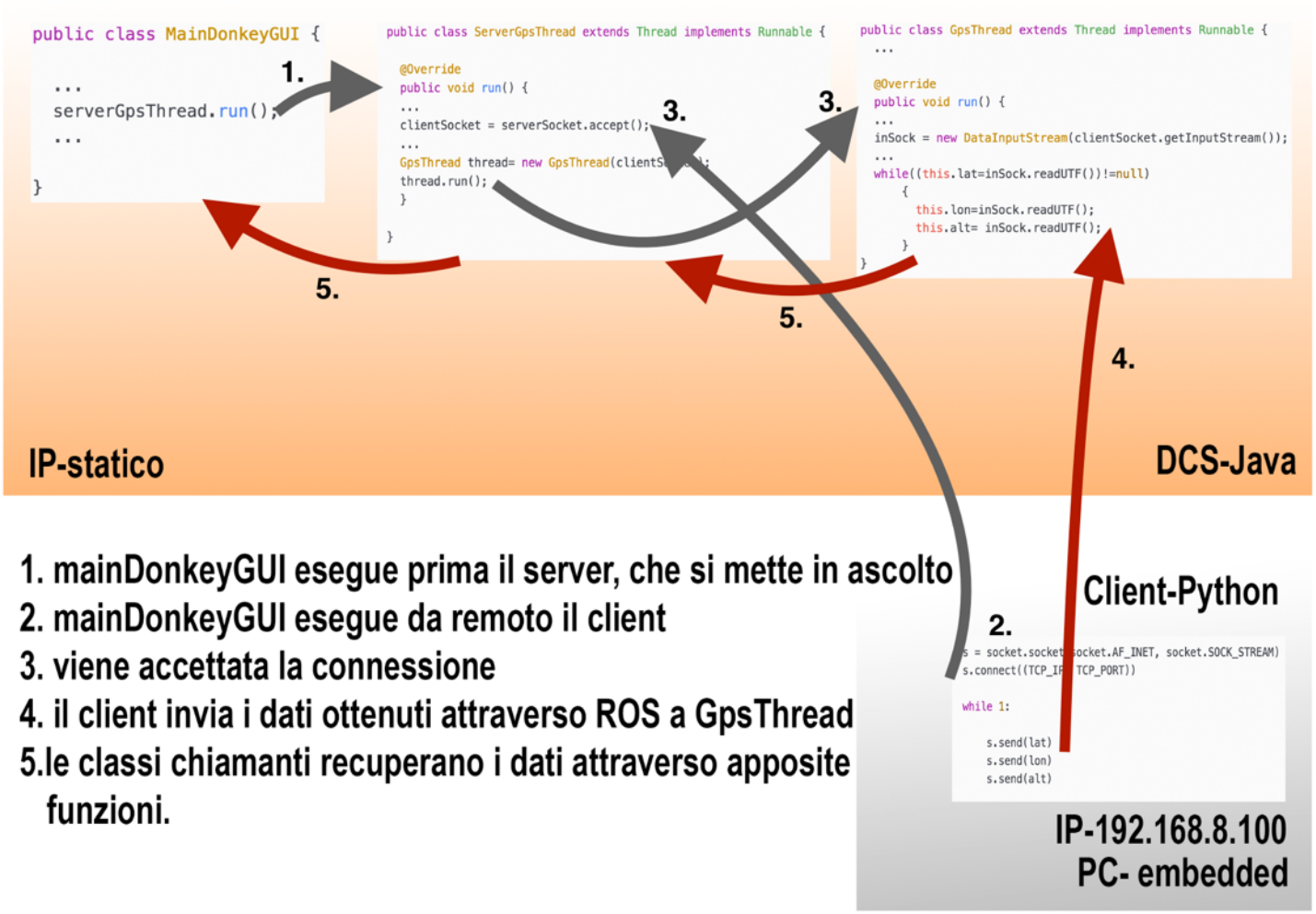
```
public class ControllerANT {

    private AntPlatform proxy;
    private static boolean init= false;

    public ControllerANT() throws IOException {

        if(!init)
           init();


    }

    private void init() throws IOException{


            proxy = new AntPlatform("192.168.8.149", 1234);
            proxy.setTimeout(2500);
            proxy.open();
            proxy.ping();
            proxy.login("User","none");

            init=true;

    }
```

Infine le funzioni getIstance() e close() invocate dal main.

```
public AntPlatform getIstance(){
    return this.proxy;
}

public void close() throws IOException {
        proxy.close();
}
```

13.7. ServerGpsThread e GpsThread

Per poter ottenere i dati relativi alla posizione dal MTi-G è necessaria un interazione client-server tra il computer di bordo, il client, e questa applicazione che funge da server. Per l'appunto anche la libreria LOS aveva fondamentalmente realizzato una struttura simile a quella qui implementata con un interazione client-server TCP.
Lo scopo è ottenere un riferimento GPS della posizione in tempo reale, la figura sottostante descrive l'interazione:



```
package model;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;

import dialog.ErrorDialog;
import dialog.InfoDialog;



public class ServerGpsThread extends Thread implements Runnable {

    public static final int PORT = 54421;
    ServerSocket serverSocket = null;
    private String lon, lat, alt;
    InfoDialog dialog;

    public ServerGpsThread() {
        super();
        dialog = new InfoDialog("ServerGpsThread");
        try
        {
            serverSocket = new ServerSocket(PORT,2);
            serverSocket.setReuseAddress(true);
            dialog.setText("ServerGsThread: avviato\n ");
            dialog.setText("Creata la server socket: " + serverSocket+"\n");
        }
        catch (Exception e)
        {
            dialog.setText("Problemi nella creazione della server socket: \n"+ e.getMessage());

        }
    }


 public String getLon() {
     return lon;
 }


 public String getLat() {
     return lat;
 }



 public String getAlt() {
     return alt;
 }
```

```
@Override
public void run() {
    try{
        Socket clientSocket = null;
        dialog.setText("\nIn attesa di richieste...");
        try
        {
            clientSocket = serverSocket.accept();
            //clientSocket.setSoTimeout(30000);
            dialog.setText("Connessione accettata: " + clientSocket + "\n");

        }
        catch (IOException e)
        {
            dialog.setText("TCP_Server: Problemi nella accettazione della connessione: "+ e.getMessage());

        }

        GpsThread thread= new GpsThread(clientSocket);
        thread.start();
        dialog.setText("GpsThread avviato...");
        while(true)
        {
        Thread.sleep(2000);

        lat=thread.getLat();
        dialog.setText(lat);
        lon=thread.getLon();
        dialog.setText(lon);
        alt=thread.getAlt();
        dialog.setText(alt);

        }

    }
    catch (Exception e)
    {
        new ErrorDialog("Server: Errore irreversibile: \n"+e.toString());

    }
}
```

```
package model;
import java.io.DataInputStream;

import java.io.EOFException;
import java.io.IOException;
import java.net.Socket;
import java.net.SocketTimeoutException;

import dialog.InfoDialog;

public class GpsThread extends Thread implements Runnable {

    private String lon, lat, alt;
    private Socket socket;
    InfoDialog dialog;

    public GpsThread(Socket clientSocket) {
        super();
        this.socket=clientSocket;
        dialog = new InfoDialog("GpsThread");
    }

 @Override
 public void run() {

     DataInputStream inSock = null;
     dialog.setText("GpsThread avviato.");
     try
     {
          inSock = new DataInputStream(socket.getInputStream());

     }
     catch (IOException e){dialog.setText("GpsThread: Errore "+e.toString());}


     try
     {
         dialog.setText("\nClient "+socket.getInetAddress().getHostName()+"\n");
         dialog.setText("\nlettura in corso...");

         while((lat=inSock.readUTF())!=null)  //aggiorno le variabili
         {
                 dialog.setText(lat);
                 this.lon=inSock.readUTF();
                 dialog.setText(lon);
                 this.alt= inSock.readUTF();
                 dialog.setText(alt);

         }
```

```
}
catch(EOFException e)
{
    dialog.setText("TCP_Server_Thread: "+this.getName()+" termina...");
}
catch(SocketTimeoutException e)
{
    dialog.setText("GpsThread: Socket Timeout Exception [Thread: "+this.getName()+"]");
}
catch(IOException e)
{
    dialog.setText("GpsThread:: I/O Exception [Thread: "+this.getName()+"]"+e.toString());

}
catch(Exception e)
{
    dialog.setText("GpsThread: Errore irreversibile [Thread: "+this.getName()+"]"+ e.toString());

}

finally
{
    dialog.setText("GpsThread: ["+this.getName()+"] chiusura Socket");
    try
    {
        socket.close();
        dialog.setText("socket chiusa.");
    }
    catch(IOException e)
    {
        dialog.setText("GpsThread: Errore chiusura Socket ["+this.getName()+"] :\n"+e.toString());
    }
}



public String getLon() {
    return lon;
}



public String getLat() {
    return lat;
}



public String getAlt() {
    return alt;
}
```

13.8. Script Client

Il Client si deve registrare al topic interessato, instaurare una connessione attraverso una socket e infine inviare i dati. Non è necessario mandarli tutti insieme perché si sta lavorando con una connessione TCP che garantisce l'ordine.

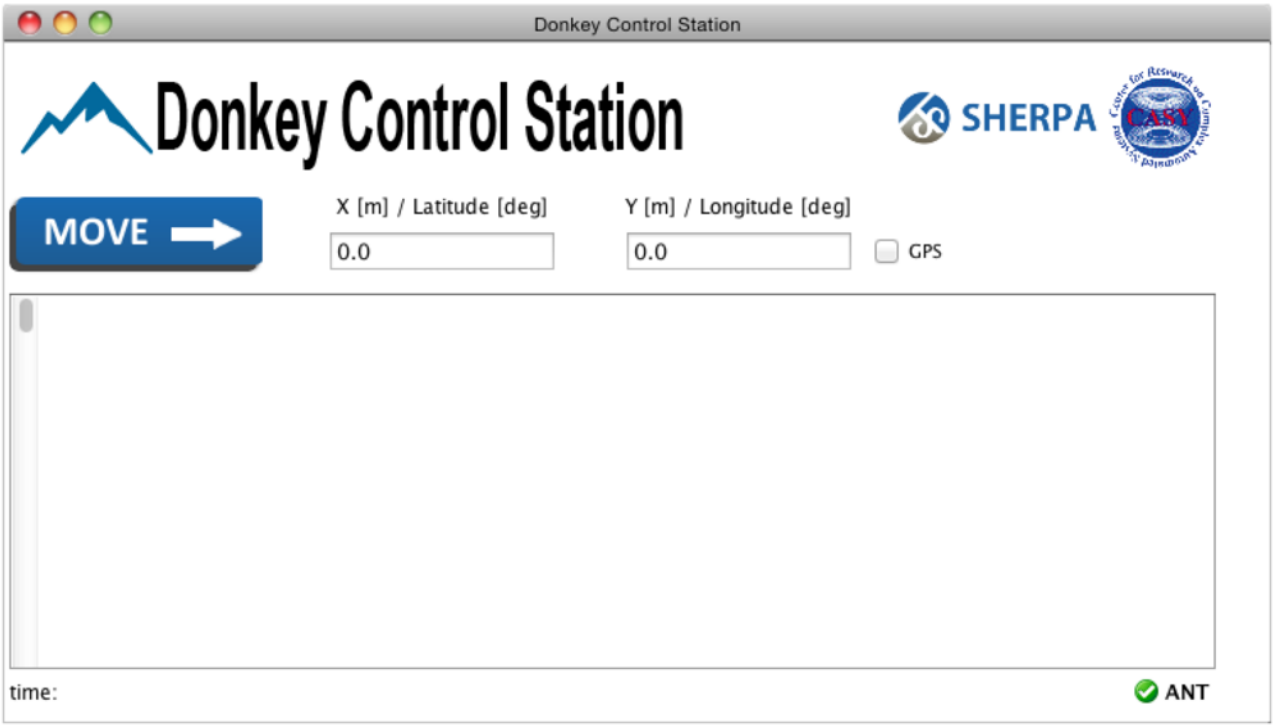
```
 #!/usr/bin/env python
import rospy
import os.path
import sys
import socket
import time

from custom_msgs.msg import estimantePosition

lat=0.0
lon=0.0
alt=0.0

def callback(data):
    global lat, lon, alt
    lat=data.latitude
    lon=data.longitude
    alt=data.altidute

def main(argv):

    rospy.init_node('donkeyGPSListener', anonymous=True)
    rospy.Subscriber("mti/filter/position", estimantePosition, callback)

    TCP_IP = '192.168.8.21' #IP statico
    TCP_PORT = 54321

    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((TCP_IP, TCP_PORT))

    while 1:

        s.send(lat)
        s.send(lon)
        s.send(alt)

        print "\nsend data: lat=%0.6f lon=%0.6f alt=%0.6f" % (lat,lon,alt)
        time.sleep(1)


if __name__ == '__main__':
    try:
        main(sys.argv)
    except rospy.ROSInterruptException:
        pass
```

13.9. MainDonkeyGUI

La GUI qui presentata è un prototipo ancora in fase di sviluppo. Risulta essere in continuo cambiamento poiché dal lavoro parallelo di altre attività del team CASY nascono sempre nuove possibile vie, la GUI pertanto deve essere sempre aperta ai cambiamenti proposti.
Per tale motivo all'interno del MainDonkeyGUI vi sono delle parti commentate che corrispondono, per esempio, all'utilizzo di ANT.

Questa applicazione prevedeva l'implementazione di una mappa. Tale lavoro si è dimostrato più oneroso di quello che si era calcolato inizialmente, tuttavia esso potrà essere in futuro la base di partenza per un altro lavoro di tesi. Per implementare una mappa è necessario disporre in tempo reale i dati della posizione. Per tale ragione si è dovuto implementare l'interazione Client-Server di cui sopra si è parlato.

La Donkey Control Station avrà un aspetto simile a questo:



13.9.1. Il codice

```
public static void main(String[] args) {
    EventQueue.invokeLater(new Runnable() {
        public void run() {
            try {
                MainDonkeyGUI window = new MainDonkeyGUI();
                window.frmDonkeyControlStation.setVisible(true);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    });
}
```

Il comportamento dell'applicazione prevede che se non vi sia la possibilità di connettersi alla rete wifi del rover non sia possibile nemmeno aprire il software. Se dovesse fallire la connect() non sarebbe possibile utilizzare l'applicazione, quindi per essere consistenti la GUI non viene inizializzata e al contrario viene mostrata una finestra di errore ErrorDialog.

```
/**
 * Create the application.
 */
public MainDonkeyGUI() {

    conn = new Connection("192.168.8.100");
   try {
        conn.connect();
        initialize();
        textArea.append("You are connected with SHERPA-Wifi.");
    } catch (IOException e1) {
        new ErrorDialog("You are not connected with SHERPA-Wifi:\n"+e1.toString());
    }finally{
        conn.close();
    }


}
```

Viene esclusa la creazione dei vari componenti perché non di particolare interesse.

```
    /**
     * Initialize the contents of the frame.
     */
    private void initialize() {

        try
        {
            ant = new ControllerANT();

        }
        catch (Exception e)
        {
            textArea.setText("Ant is not connected: "+e.toString());
        }
```

Eventualmente vengono chiuse all'interno del try-catch altre connessioni aperte all'interno del programma.

```
frmDonkeyControlStation.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
  frmDonkeyControlStation.addWindowListener(new WindowAdapter() {
      @Override
      public void windowClosing(WindowEvent event) {
         try
         {
             ant.close();

         }
         catch (IOException e)
         {
             new ErrorDialog(e.toString());
         }
         finally
         {
             frmDonkeyControlStation.dispose();
             System.exit(0);
         }

      }
  });
```

Al button “move” viene associato l’event handler che segue:

```
JButton btnMoveTo = new JButton("");
btnMoveTo.setSelectedIcon(new ImageIcon(MainDonkeyGUI.class.getResource("/logo/button.png")));
btnMoveTo.setFont(new Font("Tw Cen MT", Font.PLAIN, 16));
btnMoveTo.setIcon(new ImageIcon(MainDonkeyGUI.class.getResource("/logo/button.png")));
btnMoveTo.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {

        /* ********ANT********
        try {
            ant.getIstance().Motion_moveToPose(Double.parseDouble(textField.getText()),
                                               Double.parseDouble(textField_1.getText()), 0.0);
        } catch (NumberFormatException e2) {
            new ErrorDialog(e2.toString());
        } catch (IOException e2) {
            new ErrorDialog(e2.toString());
        }
         */
        String gps;

        if(chckbxGps.isSelected())gps="True";
        else gps = "False";

        if(textField.getText()==null || textField_1.getText()==null
                                        || !textField.getText().contains(".")
                                                || !textField_1.getText().contains(".") ){
            new ErrorDialog("please write all the arguments (ex: x= 1.0 y= 0.5)");
        }else{
        CmdThread command= new CmdThread();
        command.setCommand("cd catkin_ws;source devel/setup.bash;rosrun donkey_move moveTo.py "
                           +gps+" False "+textField.getText()+" "+textField_1.getText()+" 0.0");
        command.start();
```

Tramite la classe CmdThread viene eseguito il comando. In alternativa si poteva usare ANT con il codice commentato.

Finché command non restituisce lo stdOutput (o stdError) si deve rimanere in attesa. Un attesa attiva può non essere una soluzione molto fine, tuttavia sarà difficile che si aspetti per più di 500 ms.

Infine viene stampato il l’output nella textArea.

```
        while(command.getStdOut().compareTo("not ready")==0 || command.getStdErr().compareTo("not ready")==0)
        {try {Thread.sleep(500);}
        catch (InterruptedException e1){textArea.append("error, try again.\n");break;}}

        if(command.getStdOut()!=null)
            textArea.append("STDOUT:\n"+command.getStdOut());
        if(command.getStdErr()!=null)
            textArea.append("STDERR:\n"+command.getStdErr());
        command.close();
        }

    }
});
```

Il timer viene implementato con l'ausilio della classe TimerTask; ogni secondo si aggiorna il valore count. In alternativa si può usare ANT per sapere ogni secondo la posizione, lo stato, la velocità e volendo molte altre informazioni.

```
//******************timer-info***********************

Timer uploadCheckerTimer = new Timer(true);
uploadCheckerTimer.scheduleAtFixedRate(
    new TimerTask() {
      public void run() {

        //  Tuple3<Double, String, String> status;
            try {
                //********ANT********
                //status = ant.getIstance().Motion_getStatus();
                //Tuple2<Double, Pose> lista = ant.getIstance().Odometry_getPose();
                //textArea_1.append(lista.item1.toString());
                //textArea.append("state: "+status.item1+"\n");
                //textArea.append("speed: "+ant.getIstance().Motion_getSpeed()[1]+" m/s\n");
                count++;
                lblTime.setText("time: "+count +"s");
            } catch (Exception e) {
                textArea.append(e.toString()+"\n");
            }
      }
    }, 0, 1000);
```

Ogni volta che si vuole eseguire un comando relativo a ROS è necessario prima eseguire altri comandi: "cd catkin_ws" e "source devel/setup.bash" (Esempoio 2).
In alternativa è possibile scrivere un script sh (Esempio 1) dove si eseguono tutti i comandi necessari, che sono:

- modificare i permessi della porta USB (sudo chmod 777 /dev/ttyUSB0)
- eseguire lo script ros mtdevice.py (rosrun xsens_driver mtdevice.py -m 3 -f 100)
- eseguire il nodo master (roscore)
- eseguire lo script ros mtnode.py (rosrun xsens_driver mtnode.py)

Infine si dovrà prima far partire il server che si mette in ascolto e solo dopo eseguire lo script Python del Client.

Esempio:

1.
```
CmdThread command3= new CmdThread();
command3.setCommand("./cmdMtdevice.sh");
command3.start();
```

2.

```
CmdThread command1= new CmdThread();
command1.setCommand("cd catkin_ws;source devel/setup.bash;echo 'none'|sudo -S chmod 777 /dev/ttyUSB0");
command1.start();
this.waitForResult(command1);

CmdThread command2 =  new CmdThread();
command2.setCommand("cd catkin_ws;source devel/setup.bash;rosrun xsens_driver mtdevice.py -m 3 -f 100");
command2.start();
this.waitForResult(command2);
```

dove waitForResult:

```
private void waitForResult(CmdThread command){

    while(command.getStdOut().compareTo("not ready")==0 || command.getStdErr().compareTo("not ready")==0)
    {try {Thread.sleep(500);}
    catch (InterruptedException e1){textArea.append("error, try again.\n");break;}}

    if(command.getStdOut()!=null)
        textArea.append("STDOUT:\n"+command.getStdOut());
    if(command.getStdErr()!=null)
        textArea.append("STDERR:\n"+command.getStdErr());
    command.close();

}
```

Infine si potranno ottenere i dati GPS attraverso un metodo simile a questo:

```
public List<String> getGpsData(){

    List<String> coordinates = new ArrayList<String>();


    if(server.getState() == Thread.State.RUNNABLE)
    {
        coordinates.add(server.getLat());
        coordinates.add(server.getLon());
        coordinates.add(server.getAlt());
        return coordinates;

    }else return null;


}
```

Per concludere si mostra l'output della shell di Eclipse dove vengono stampati i valori ottenuti dal MTi. I valori ottenuti non corrispondono alle coordinate vere poiché estratti durante una fase di debug avvenuta all'interno del laboratorio dove il segnale GPS è quasi totalmente assente.

```
lat: 53.21856441
lon: 4.54243135
alt: 1676.608
---
lat: 53.21856445
lon: 4.54243135
alt: 1676.608
---
lat: 53.21856444
lon: 4.54243134
alt: 1676.608
---
lat: 53.21856445
lon: 4.54243137
alt: 1676.608
---
lat: 53.21856447
lon: 4.54243139
alt: 1676.608
---
```

14. Conclusioni

Gli scopi di questo lavoro sono stati due: implementare un algoritmo di navigazione autonoma di alto livello e sviluppare un interfaccia grafica destinata ad un utente medio che consenta di nascondere tutte le varie particolarità dei livelli sottostanti.
Più volte è stato necessario riconsiderare la soluzione trovata poiché, essendo un progetto in continuo cambiamento, possono entrare in gioco più variabili non considerate all'inizio (limiti software, hardware o relativi a protocolli).

Il risultati finali hanno avuto esito positivo ed è quindi stato possibile produrre codice funzionante, che fungerà da base per le future modifiche destinate ad ampliare ed ad ottimizzare il progetto di navigazione autonoma del rover Donkey.

Come è possibile consultare nella sezione progressi relativa a maggio-novembre 2015 sul sito del progetto sherpa (http://www.sherpa-project.eu/sherpa/content/progress), dal punto di vista della navigazione, è stato prodotto un nodo ROS per leggere la posizione attuale GPS e ricevere nuovi waypoint espressi in termini di latitudine e longitudine di destinazione; questi successivamente vengono letti da un ROS topic su cui il delegation framework pubblica i waypoint.
Sono poi usati per ottenere una traiettoria di destinazione nel sistema di riferimento cartesiano locale e vengono calcolati a partire dalla posizione corrente espressa in termini di latitudine e longitudine.
Questa traiettoria viene quindi eseguita dalla routine descritta in questa tesi sfruttando le capacità degli algoritmi di basso livello che consentono di evitare ostacoli dinamicamente e in autonomia.

Bibliografia

P. Ancilotti, M. Boari, A. Ciampolini, G. Lipari, “Sistemi Operativi”, McGraw-Hill, 2008

BlueBotics, “SHERPA Platform User Manual - Basic usage, handling & maintenance”, 2015-06-30

BlueBotics, “Platform communication interface - RPC over LOS”, 2015-02-06, Version: 1.5dev;

Xsens, “MTi User Manual, MTi 10-series and MTi 100-series”, 2015-02-27

Xsens, “MT Manager User Manual”, 2010-11-15

Xsens, “Firmware Update User Manual” , 2010-11-15

Xsens, “Magnetic Field Mapper Documentation” , 2010-11-15

Sitografia

www.sherpa-project.eu/sherpa/content/progress (ultima consultazione: 2016-1);

www.lia.disi.unibo.it (ultima consultazione: 2015-12);

www.wiki.ros.org (ultima consultazione: 2015-12);

www.stackoverflow.com (ultima consultazione: 2015-12);

www.docs.oracle.com (ultima consultazione: 2015-12);

www.docs.python.org (ultima consultazione: 2015-12);

www.archive.psas.pdx.edu/CoordinateSystem (ultima consultazione: 2015-11);

www.ganymed.ethz.ch (ultima consultazione: 2015-12);

www.support.ssh.com (ultima consultazione: 2015-12);