ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA

FACOLTÀ DI INGEGNERIA

CORSO DI LAUREA IN INGEGNERIA DELL'AUTOMAZIONE

ANALISI DI SOLUZIONI DI MOTION CONTROL PER AZIONAMENTI BLDC NEL PROGETTO ALMAX

Tesi in

INGEGNERIA DEI SISTEMI DI CONTROLLO

Relatore:

Chiar.mo Prof. Ing.

ANDREA TILLI

Presentata da:

FILIPPO GUARDA

Sessione III

Anno Accademico 2019-2020

A Giorgio, per avermi insegnato che prima viene il dovere.

Indice

Introduzione **1**
0.1 Pipeline . . . 2
0.2 Capitolo 1 . . . 2
0.3 Capitolo 2 . . . 2
0.4 Software . . . 3

1 Architettura e componenti Hardware **5**
1.1 Motori BLDC . . . 5
1.2 Computer di bordo . . . 7
1.3 Drive . . . 7
1.4 Convertitore MOXA e adattatore PCAN . . . 7
1.4.1 MOXA . . . 8
1.4.2 PCAN . . . 8
1.5 Adattatori . . . 8
1.5.1 Adattatore per MOXA . . . 8
1.5.2 Adattatore per PCAN . . . 10
1.6 Alimentatori . . . 11

2 Introduzione a ROS, CAN e CANopen. **13**
2.1 ROS . . . 13
2.1.1 ROS master . . . 14
2.1.2 Nodi . . . 15
2.1.3 Topics . . . 15
2.1.4 Servizi . . . 15
2.1.5 Strumenti . . . 16

2.1.5.1 rviz: . . . 16
2.1.5.2 roslaunch . . . 16
2.1.5.3 Catkin . . . 16
2.2 Bus CAN . . . 16
2.2.1 Livello fisico . . . 17
2.2.2 Livello trasmissione dati . . . 18
2.3 CANopen over CAN . . . 19
2.3.1 Object dictionary . . . 19
2.3.2 Protocolli di comunicazione, SDO . . . 20
2.3.3 Protocolli di comunicazione, PDO . . . 20
2.3.4 Profili di Moto, CiA 402 . . . 21

3 Architettura e componenti Software **23**
3.0.1 Setup drive . . . 23
3.0.1.1 Abilitare MOXA . . . 23
3.0.2 Installazione Drive Watcher . . . 25
3.0.3 Programmazione dei parametri . . . 26
3.1 Setup ROS . . . 27
3.2 Setup PCAN . . . 28
3.3 Can-utils . . . 29
3.4 CANopenSocket . . . 30
3.4.1 canopend . . . 30
3.4.2 canopencomm . . . 31
3.5 Kacanopen[KIT] . . . 31
3.6 ros_canopen [rc] . . . 33
3.7 File .edf .urdf . . . 34
3.7.1 EDF . . . 34
3.7.2 URDF . . . 35

Conclusioni **39**
3.8 Debugging del connettore RJ45 . . . 41
3.9 Utilizzare ros_canopen . . . 41
3.10 Utilizzare Kacanopen . . . 41

3.11 If everything else fails... 42

Bibliografia **43**

Elenco delle figure

1 Pipeline del Motion Control 2

1.1 Collegamento cavo encoder 6
1.2 Collegamento cavo motore 6
1.3 Adattatore DB9-RJ45(sinistra) insieme al MOXA 9
1.4 Schematica adattatore DB9-RJ45 per MOXA 9
1.5 Schematica adattatore DB9-RJ45 per PCAN 10
1.6 Adattatore DB9-RJ45(centro) insieme al PCAN(sinistra) e al cavo LAN . 11

2.1 Ruolo svolto da ROS nel motion control 14
2.2 Ruolo svolto da CAN e CANopen nel Motion control 17
2.3 Frame CAN standard 18
2.4 Schema degli standard CAN e CANopen nel modello OSI 19

3.1 Installazione MOXA, parte 1 24
3.2 Installazione MOXA, parte2 25
3.3 Communication Settings e Search Node Tool 26
3.4 lo strumento Parameters Configuration 27
3.5 Schema di funzionamento di ros_canopen 34
3.6 Ambiente di lavoro in cui il progetto è stato svolto 40

Introduzione

La ERC, European Rover Challenge, è una competizione a livello europeo, organizzata dal 2014 in Polonia. Anche l'università di Bologna, in collaborazione con la sede di Forlì, partecipa a questa competizione con il gruppo AlmaX, che ha il compito di sviluppare un rover che rispetti le specifiche fornite nel regolamento [Bog20], tra queste caratteristiche:

- Un peso pari o inferiore a 50kg.
- La capacità di movimento autonomo attraverso computer vision e mapping topografico, oltre alla possibilità di inviare comandi al rover da una base station via link radio.
- La capacità del rover di attraversare ostacoli e un terreno accidentato simile al suolo marziano.
- La capacità del rover di interagire con un pannello di controllo fisico

Inserito nel contesto più ampio dello sviluppo del rover, l'argomento di questa tesi è il benchmark della parte relativa al motion control.

Durante la durata del progetto è stato eseguito il setup completo dell'hardware e sono state testate le funzionalità dei vari software disponibili con lo scopo di trovare il miglior candidato per il completamento del progetto.

Tutti i materiali e la documentazione inclusi in questa tesi (manuali, drivers, immagini, bibliografia e file lyx e latex) sono disponibili su github al link `https://github.com/FilippoGuarda/MotionControlAlmax`.

0.1 Pipeline

L'obiettivo di questa tesi è di permettere ad una base station di controllare i motori BLDC a bordo del rover attraverso messaggi. Tali messaggi specificano un target di velocità da raggiungere che viene tradotto in un comando leggibile dal drive che si occupa di raggiungerlo controllando il motore in PWM con curve di accelerazione definite a priori.

Il drive a sua volta riceve un feedback dall'encoder del motore e deve inviare al computer di bordo la conferma di effettivo raggiungimento del target o, in alternativa, la presenza di errori.

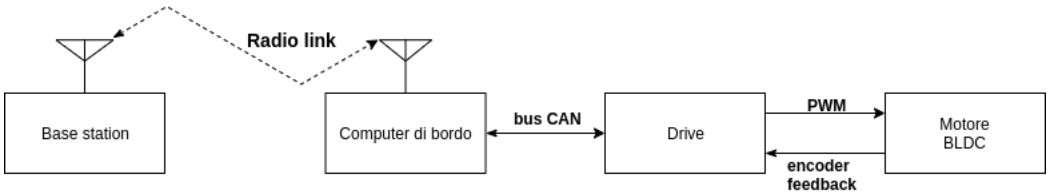


Figura 1: Pipeline del Motion Control

0.2 Capitolo 1

Nel primo capitolo si analizzano i componenti hardware che fanno parte del motion control del Rover: motore, drive, adattatori e alimentatori.

Si mostra inoltre il corretto cablaggio degli adattatori che permettono la comunicazione con il drive da parte del computer di bordo, oltre ai connettori di potenza e di feedback del motore.

0.3 Capitolo 2

Il secondo capitolo è dedicato ad una introduzione a ROS e CANopen: il primo, un middleware specifico per la robotica, il secondo, uno standard di comunicazione necessario a controllare il drive.

La prima sezione è dedicata a ROS e ai suoi componenti: master, nodi, messaggi e topics. Come ROS li utilizza per permettere la comunicazione tra macchine differenti e per ampliare le sue capacità con strumenti e librerie esterne.

La seconda sezione è dedicata a al bus CAN, attraverso il quale il protocollo CANopen(terza sezione) muove pacchetti di informazione (PDO ed SDO) attraverso i diversi nodi che fanno parte del sistema di motion control.

0.4 Software

Nel terzo capitolo si elencano i diversi strumenti utilizzati nei successivi tentativi di comunicare con il drive, partendo da quelli a livello più basso, fino ad arrivare ai due candidati più promettenti (due librerie di ROS: ros_canopen e Kacanopen) che possono eseguire il bridging tra messaggi ROS e frame CANopen.

Capitolo 1

Architettura e componenti Hardware

1.1 Motori BLDC

Obbiettivo del progetto è il controllo di un motore BLDC; nel caso in esame, i motori forniti sono Siboni S60 ed S40. Questi motori hanno le seguenti specifiche, che ci saranno utili più tardi per inserirle nei parametri del drive[Sibb]:

Tensione di alimentazione	32V
Potenza nominale	400W
Coppia di stallo	1.37Nm
Corrente di stallo	9.40A
Velocità nominale	3000rpm
Momento di inerzia rotorico	$0.30kg * m^2x10^{-4}$

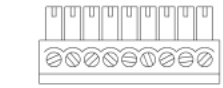
Tabella 1.1: Specifiche principali del motore Siboni ProPlanet S060 2B305

I motori dispongono di connettori di tipo molex a 6 e 12 pin, con le seguenti schematiche[Mice]:

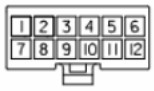
Collegamento motore **S0602B305P1P150 70 DA14 M5 BLU P14 S44 M 2048PPR**
COLLEGAMENTO CAVO SEGNALI
Cod.

CN4 Morsettiera Phoenix MC1,5/9-ST-3.81 lato *Micro digital One*		**Connettore Molex type 5557-12 poli lato motore**	
1	**GND**	**12**	SHIELD
2	**ENC A**	**8**	Ch. B
3	**ENC B**	**10**	Ch. A
4	**ENC Z**	**11**	Ch. Z
5	**+V (out)**	**1**	+5V dc
6	**GND**	**3**	0 V
7	**HALL1**	**5**	HALL W
8	**HALL2**	**4**	HALL V
9	**HALL3**	**6**	HALL U

Connettore Molex
Vista lato cavo



pin 1



12 CKT.

Figura 1.1: Collegamento cavo encoder

COLLEGAMENTO CAVO MOTORE
Cod.

POWER Morsettiera Phoenix MC1,5/5-ST-5.08 lato *Micro digital One*		**Connettore Molex type 5557-6 poli lato motore**	
1	+HV +		
2	GND -		
3	U	**1**	U
4	V	**2**	V
5	W	**4**	W

Connettore Molex
Vista lato cavo

pin 1

6 CKT.

Figura 1.2: Collegamento cavo motore

1.2 Computer di bordo

Nel rover, il computer di bordo è una UPboard[Ubb, Uba], molto comoda in quanto estremamente modulare e dotata di moduli extra. Tra i muduli disponibili ne è presente anche uno dotato di porte seriali CAN e rs422, molto utile ai fini del progetto in quanto riduce il numero di connettori extra.

Per questo progetto, non essendo disponibile la UPboard, è stato utilizzato un normale computer che utilizza Linux come OS.

1.3 Drive

Il drive che guida il motore in PWM è un Microphase Micro Digital One, con le seguenti specifiche[Micc]:

Range di tensione	19-84V
Corrente nominale	10A
Corrente di picco	20A
Potenza nominale	580W
Potenza di picco	1060W

Tabella 1.2: Specifiche principali del drive MicroDigitalOne

Il drive è collegato al motore attraverso due connettori. Uno di potenza, attraverso il quale passano le fasi della PWM, il ground e la linea che disattiva il freno integrato nel motore. L'altro responsabile della ricezione del feedback di posizione da parte dell'encoder incrementale e dei sensori di Hall.

Sul drive sono anche presenti due attacchi RJ45 che il drive usa per comunicare attraverso connessione seriale.

1.4 Convertitore MOXA e adattatore PCAN

Per connettere il drive ai due computer utilizzati, uno per la programmazione dello stesso e uno per il controllo, si usano due connettori da usb a seriale:

1.4.1 MOXA

Per programmare parametri e registri del drive è necessario utilizzare un software proprietario (di cui si parlerà nel dettaglio nel capitolo 2) che comunica attraverso una connessione seriale rs 422 resa possibile da un convertitore MOXA. Il convertitore è dotato di tre led di stato: uno che indica l'accensione, uno che si illumina in trasmissione e uno che si illumina in ricezione.

1.4.2 PCAN

La comunicazione con il computer di bordo avviene su bus CAN, attraverso l' adattatore PCAN[PEA].

L'adattatore è dotato di un led che ne comunica lo stato:

- LED acceso: usb connessa al computer, nessuna comunicazione
- LED lampeggia lentamente: adattatore attivo
- LED lampeggia velocemente: trasmissione in corso

1.5 Adattatori

Sia il convertitore MOXA che il PCAN utilizzano, per la connessione seriale, degli attacchi DB9, che utilizzano effettivamente solo una parte dei pin. Questo è un problema in quanto il drive dispone solo di attacchi RJ45. Per ovviarlo sono stati cablati a mano due adattatori.

1.5.1 Adattatore per MOXA

Benchè la documentazione del convertitore MOXA specifichi su quali pin passi la linea seriale rs422, nella confezione è fornito un attacco extra che presenta connessioni ben definite. Utilizzando la schematica degli attacchi RJ45 fornita nel datasheet del drive[Micd] è stato cablato a mano un cavo ethernet con un RJ45 da un lato e cavi scoperti dall'altro, in modo da poterli connettere all'attacco fornito.

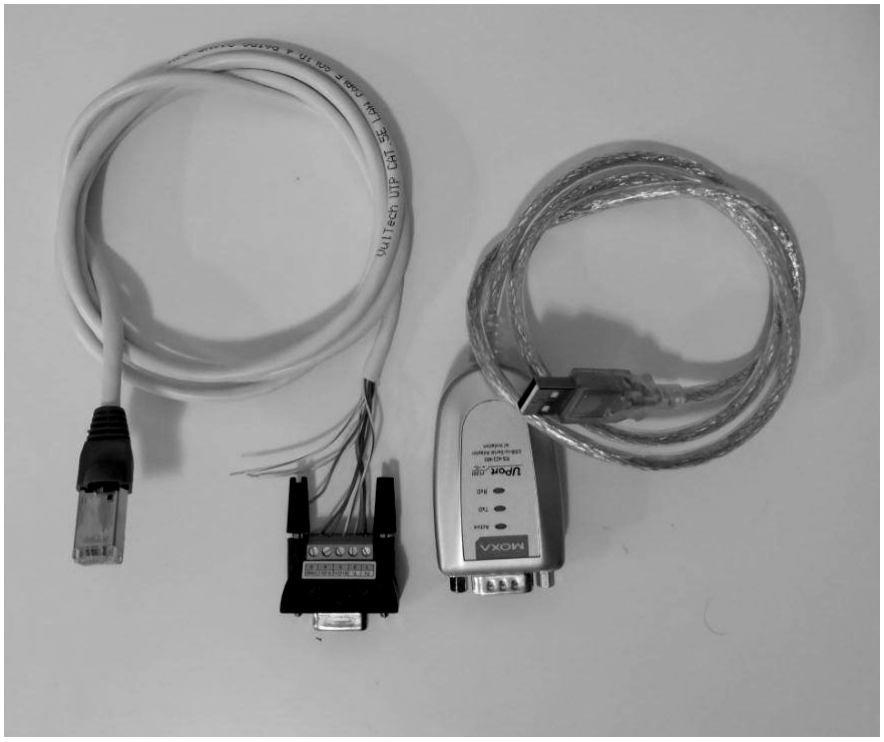


Figura 1.3: Adattatore DB9-RJ45(sinistra) insieme al MOXA

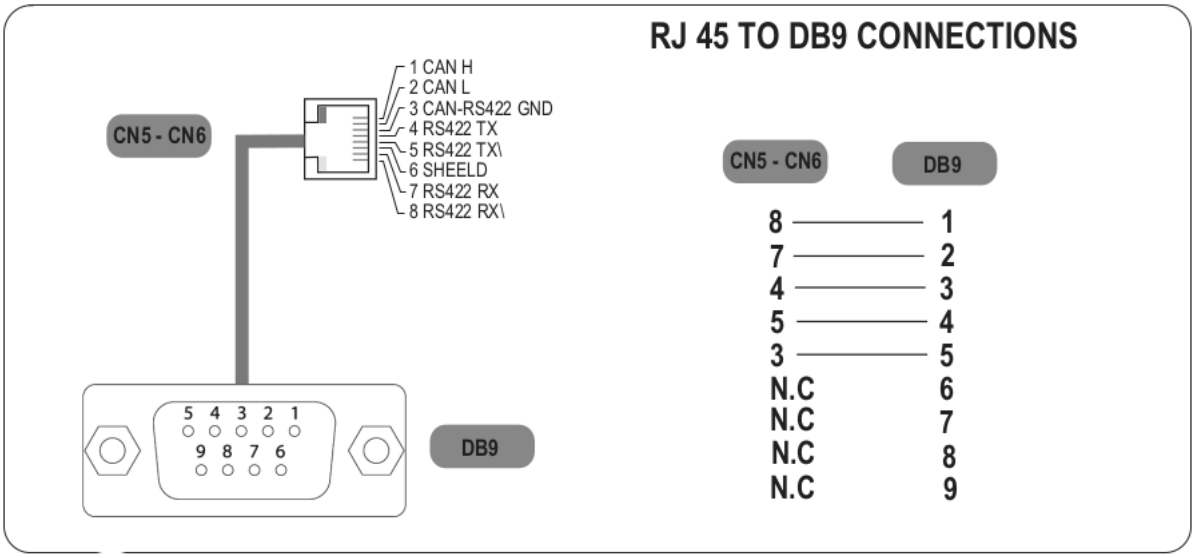


Figura 1.4: Schematica adattatore DB9-RJ45 per MOXA

1.5.2 Adattatore per PCAN

Anche l'adattatore PCAN dispone di un attacco DB9, che utilizza solo 4 pin. Il bus can passa attraverso i pin 2 (Can H) e 7 (Can L) questi due pin devono essere collegati ai pin n. 2 e 3 dell'attacco RJ45. Questa connessione si effettua tramite un adattatore cablabile a mano di facile reperibilità. Quello utilizzato in questo progetto nello specifico è stato acquistato su Amazon (`https://www.amazon.it/gp/product/B00006IRQA/ref=ppx_yo_dt_b_asin_title_o01_s00?ie=UTF8&psc=1`) ed ha i pin dell'attacco RJ45 femmina codificati tramite i colori dei cavetti che vanno inseriti manualmente nell'attacco DB9 femmina, secondo questa schematica[PEA]:

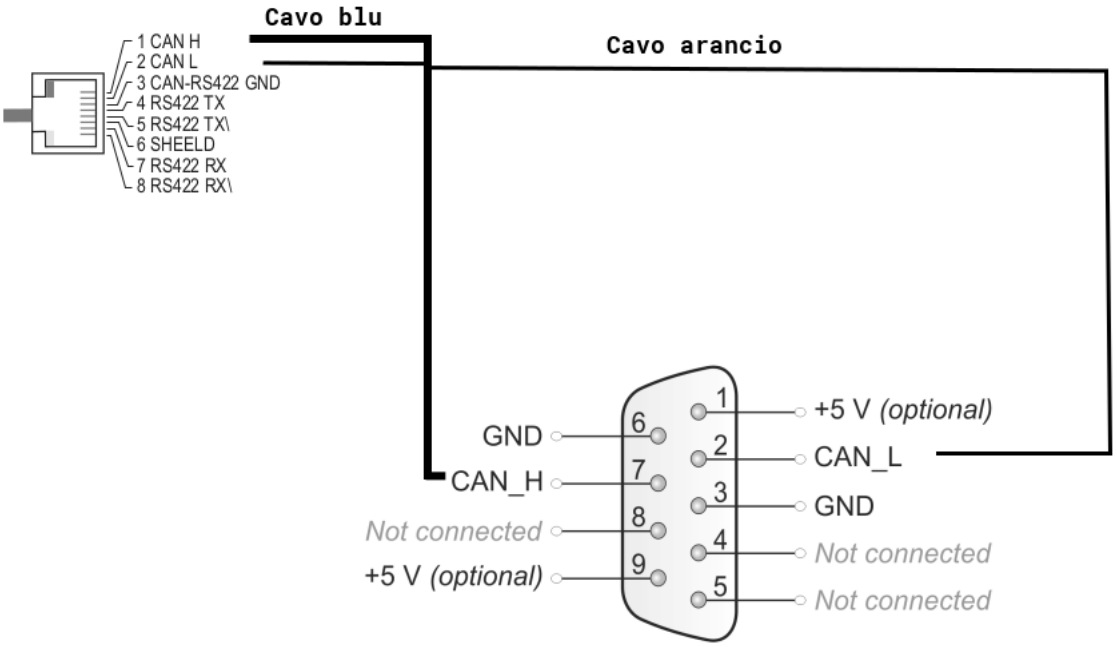


Figura 1.5: Schematica adattatore DB9-RJ45 per PCAN

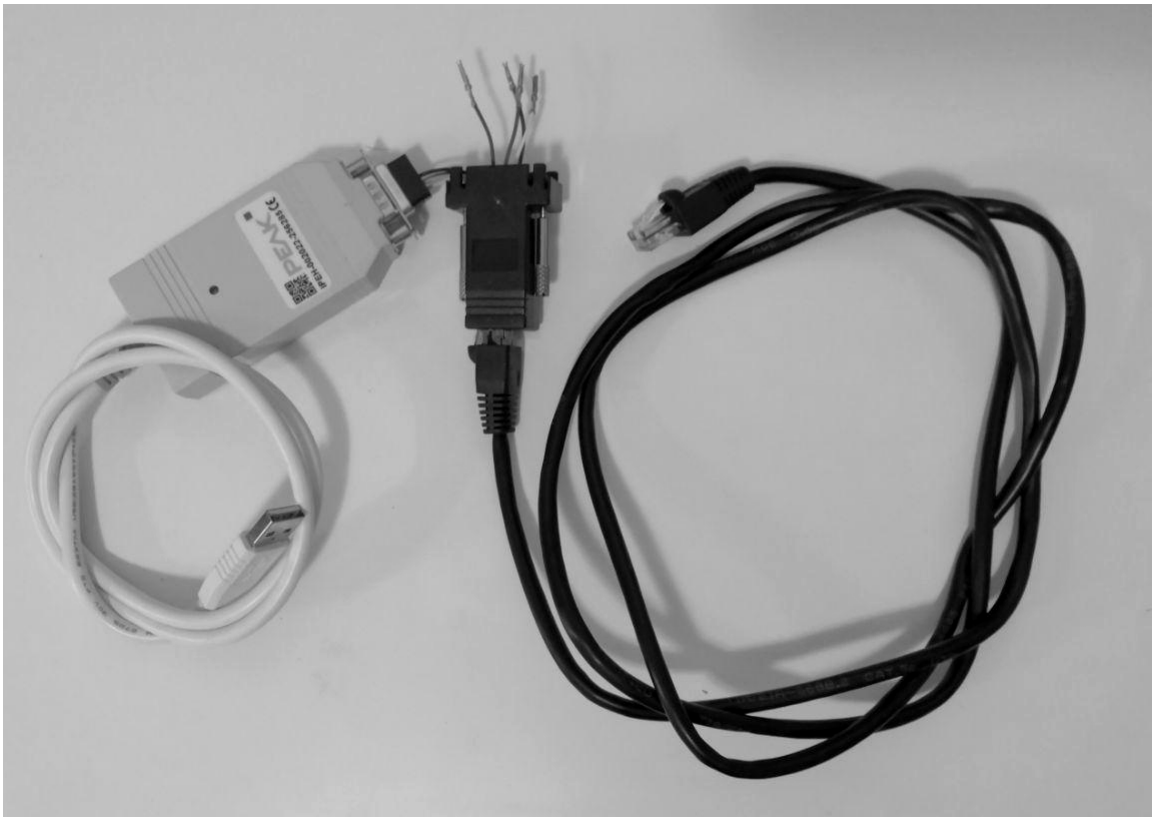


Figura 1.6: Adattatore DB9-RJ45(centro) insieme al PCAN(sinistra) e al cavo LAN

1.6 Alimentatori

Per il progetto servono due alimentatori separati, uno di potenza (BK precision 9104), che alimenta il controllo PWM a 48V (nel rover l'alimentazione sarà fornita da una batteria a 48V), e uno riservato alla parte logica, che alimenta il drive a 24V.

Capitolo 2

Introduzione a ROS, CAN e CANopen.

Il modello di motion control è basato su ROS, un software di connessione, che facilita la comunicazione tra diverse macchine sulle quali è installato; nel nostro caso, tra la base station e il computer di bordo attraverso un link radio.

Uno dei pregi di ROS è la possibilità di installare librerie aggiuntive che lo dotano di funzionalità extra, tra le quali la possibilità di codificare istruzioni in CANopen. Queste istruzioni servono a controllare il drive passando attraverso il datalink layer CAN.

In questo capitolo si parlerà, nello specifico, di ROS e CANopen over CAN.

2.1 ROS

ROS, acronimo di Robot Operative System, non è un sistema operativo in sè, ma un middleware open-source per la robotica. [ROSd]

Frutto dell'unione iniziale di diversi progetti dell'università di Stanford (tra cui Stanford AI Robot, STAIR e Personal Robot, PR) attraverso l'appoggio di Willow Garage (un incubatore per la robotica), ROS comprende una collezione di librerie, strumenti e convenzioni che hanno lo scopo di semplificare il controllo di un robot.

Lo sviluppo è basato su un modello "Federale" tale che un gruppo può scaricare il codice di ROS nel proprio server, lavorarci e decidere in seguito se renderlo disponibile pubblicamente sotto forma di open software. [ROSc]

I processi di ROS sono rappresentati come nodi in una struttura a grafi, e comuniCANo tra di loro inviando messaggi attraverso CANali chiamati topics. I nodi possono svolgere

funzioni o richiedere lo svolgimento di funzioni per e da altri nodi nonchè accedere a un database di informazioni comuni chiamato parameter server.

Per questo motivo ROS è utile per armonizzare diversi ambiti del controllo del rover; attraverso i topics è possibile mettere in comunicazione macchine differenti (in questo caso la base station e il computer di bordo) permettendo sia il controllo in remoto attraverso un HMI sulla base station, sia la navigazione autonoma, attraverso i sensori montati sul rover che inviano le informazioni raccolte al nodo incaricato dell'elaborazione dati. I comandi inviati dal pilota o dall'algoritmo incaricato della navigazione a partire dai rispettivi nodi, devono essere tradotti da un processo di ROS in CANopen frames (di cui parleremo nella sezione dedicata a CANopen) e poi comunicati sul bus CAN.

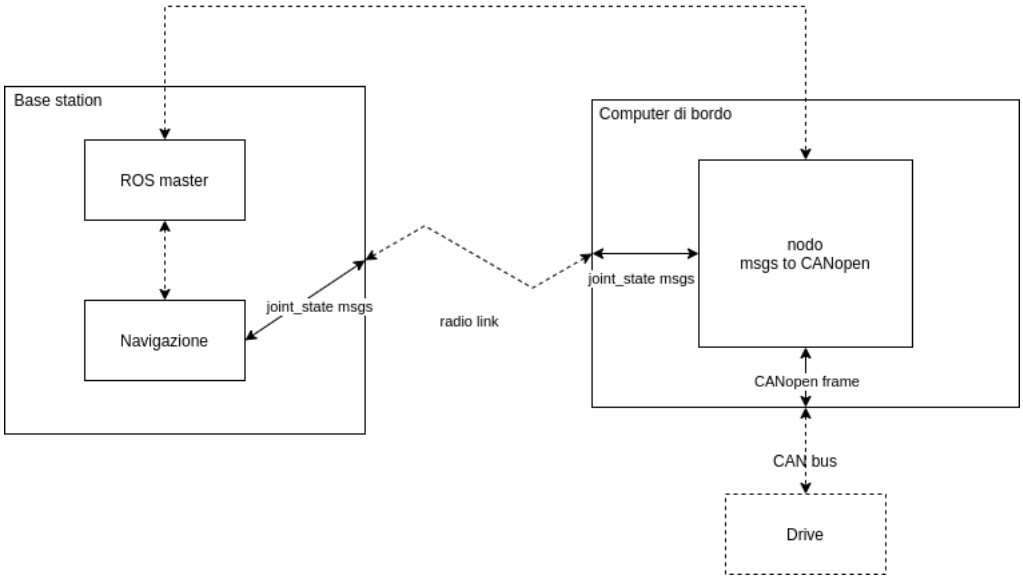


Figura 2.1: Ruolo svolto da ROS nel motion control

2.1.1 ROS master

Il nodo più importante nello schema di ROS è il master; esso infatti offre un servizio di naming e di registrazione al resto dei nodi. Essenzialmente quello che il master fa è permettere ai nodi di trovarsi a vicenda quando hanno la necessità di scambiare informazioni; svolge inoltre la funzione di controllo degli update del parameter server.

Attivare il master è la prima operazione da svolgere, attraverso il comando:

```
$ roscore
```

[ROSi]

2.1.2 Nodi

I nodi rappresentano singoli processi, ognuno con un nome specifico, e sono alla base di ROS.

Per funzionare ogni nodo deve essere iscritto al master, ed esegue delle azioni in base agli input ricevuti. Queste azioni possono comunicare con altri nodi attraverso i topics, modificare il parameter server, oppure offrire servizi. [ROSi]

2.1.3 Topics

Un topic è un CANale di comunicazione dotato di nome univoco, attraverso il quale i nodi inviano messaggi pubbliCANdoli sul topic, oppure li ricevono se iscritti al topic.

La pubblicazione e la sottoscrizione ad un topic è anonima, sicchè un nodo sa solo con quale topic sta interagendo, ma non con quali nodi.

I messaggi che passano attraverso un topic possono trasportare qualsiasi tipo di dato, ed hanno una struttura definita attraverso un file .msg.

Per esempio un messaggio può contenere due campi, x e y, uno definito come un int32 e uno definito come int16 nel seguente modo:

```
int32 x
int16 y
```

[ROSe]

Per trasmettere un messaggio, un nodo comunica al master che sta pubbliCANdo un dato su di un dato topic, ma non essendo nessun altro nodo iscritto a quel topic, il messaggio non viene inviato. Appena un altro nodo comunica al master che si sta iscrivendo ad un dato topic, il master arbitra una comunicazione tra di essi, permettendo al nodo 1 di inviare il messaggio direttamente al nodo 2, senza che l'informazione passi attraverso il master.

2.1.4 Servizi

Azioni complesse che portano ad un solo output vengono chiamate servizi. Un nodo può offrire un servizio (come l'acquisizione di un'immagine, il calcolo di un percorso, o l'invio

di un comando attraverso una linea seriale) o richiedere un servizio ad un altro nodo. Questo rende particolarmente facile delegare computazioni complesse ad una macchina esterna con più potenza di calcolo.

Un servizio è definito da una coppia di messaggi, uno per la richiesta e uno per la risposta, definiti attraverso un file .srv. [ROSh]

2.1.5 Strumenti

ROS dispone inoltre di strumenti utili a lavorare con nodi, topics e servizi che vengono forniti nell'installazione base. Tra i più importanti elenchiamo:

2.1.5.1 rviz:

Un visualizzatore in tre dimensioni utilizzato per testare i robot e l'ambiente in cui lavorano. [ROSg]

2.1.5.2 roslaunch

Roslaunch viene utilizzato per attivare più nodi contemporaneamente sia sulla macchina locale sia in remoto e permette di agglomerare più processi di inizializzazione in uno script, in modo da ridurli a un singolo comando.

2.1.5.3 Catkin

Basato su Cmake, catkin è un "build tool", vale a dire, un programma che automatizza la creazione di applicazioni eseguibili a partire da codice sorgente. Catkin permette di creare ambienti di sviluppo e di utilizzare librerie semplicemente clonando il loro codice sorgente da github. [ROSa]

2.2 Bus CAN

Il bus CAN (controller area network) è uno standard originariamente sviluppato per permettere una robusta comunicazione tra microcontrollori in campo automotive. Al momento fa parte degli standard EOBD e OBD-II (on-board diagnostics), che rendono la sua presenza obbligatoria su tutti i veicoli venduti nell'Unione Europea e negli Stati Uniti.

Oltre all'ambito automotive vede svariati utilizzi che variano dalle macchine agricole, all'areonautica, dall'automazione agli ascensori e alla strumentazione medica.

Il bus CAN occupa i due livelli più bassi del modello ISO-OSI: Fisico e Data-link. [FH, CiAc]

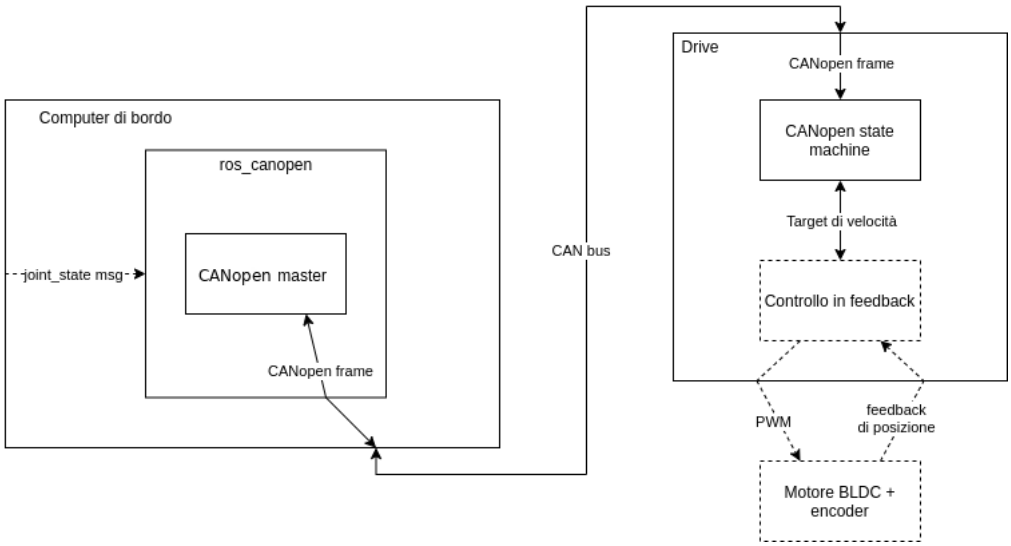


Figura 2.2: Ruolo svolto da CAN e CANopen nel Motion control

2.2.1 Livello fisico

CAN è un bus seriale producer-consumer nel quale i vari nodi sono tutti connessi tra loro tramite due cavi.

Nel nostro caso il bus collega il drive al connettore PCAN, ed entrambi sono considerati nodi. Questo bus usa una logica cablata differenziale con due segnali: CAN-H e CAN-L.

I segnali vengono impostati su uno stato dominante quando CAN-H ha una tensione più alta di CAN-L. Quando non viene più fornita tensione vengono riportati in uno stato recessivo da un resistore passivo, con CAN-H a tensione più bassa di CAN-L. Un bit a 0 codifica un segnale dominante, mentre un bit a 1 un segnale recessivo. [ric]

2.2.2 Livello trasmissione dati

Ogni nodo può richiedere il permesso di trasmissione di un frame in qualsiasi momento, ma i frame con identificativo più alto hanno la precedenza per l'utilizzo del bus. In questo modo si evitano conflitti di trasmissione.[CiAa] Tutti i frame sono trasmessi in broadcast e nel CAN classico il payload massimo è limitato a 8 bytes.

Un frame CAN è composto da 8 campi, come segue:

1. Start of frame (1 bit)
2. Standard Identifier (11 bit)
3. Remote Transmission Request (1 bit)
4. Control (6 bit)
5. Data (8 bytes)
6. Cyclic Redundancy Check (16 bit)
7. ACK (2 bit)
8. End Of Frame (7 bit)

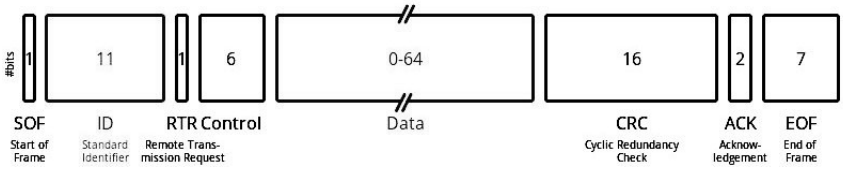


Figura 2.3: Frame CAN standard

[Elea]

2.3 CANopen over CAN

Mentre CAN rappresenta i due livelli più bassi del modello OSI, CANopen occupa il livello più alto, quello di applicazione (CiA 402[Ciab]).

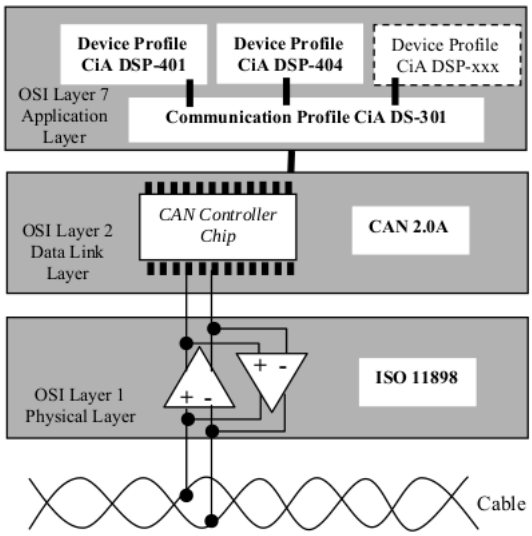


Figura 2.4: Schema degli standard CAN e CANopen nel modello OSI [Bot]

CANopen integra al suo interno diversi concetti:

- Object dictionary
- Protocolli di comunicazione
- Profili di moto

2.3.1 Object dictionary

Ogni nodo della reta CANopen deve essere provvisto dell'object dictionary (OD): una struttura standardizzata che contiene tutti i parametri e i registri che descrivono il suo comportamento, dal nome del device (un parametro statico), fino ad arrivare al target di velocità nel caso di un drive per motori (contenuto in un registro che deve poter essere modificato durante il funzionamento). Ad ogni voce dell'OD si accede tramite un indice

esadecimale, a cui è aggiunto un sotto-indice a 8 bit. Per esempio, per accedere alla voce che contiene la frequenza dell'heartbeat (un parametro che ci serve per capire se il nodo sta funzionando correttamente) utilizzeremo l'indice 0x1017, subindex 0.

Tra i registri del drive, quelli più importanti nella nostra applicazione sono Control-Word (0x6040), StatusWord (0x6041), Heartbeat (0x1017) e Target Velocity (0x800D)[Bot, Mica, Micb].

2.3.2 Protocolli di comunicazione, SDO

Per modificare e leggere le voci dell'object dictionary si utilizza l'SDO, Service Data Object. L'SDO è un protocollo di comunicazione relativamente lento in quanto ha un comportamento "client-server"; che comporta un overhead sostanziale, non è possibile utilizzarlo per trasmettere dati in real-time. Nel nostro caso il computer di bordo (client) inizia una comunicazione con il drive (server) e può richiedere una lettura di una voce dell'OD (SDO upload) oppure può modificare una voce dell'OD (SDO download).

Ogni CAN frame di un SDO può portare fino a 4 bytes, e ad ogni richiesta segue una risposta. Quindi, per leggere 4 bytes di informazione è necessario un CAN frame per inviare la richiesta, e uno per ricevere la risposta.

Gli SDO sono molto flessibili: permettono il trasporto di una grande quantità di dati utilizzando più CAN frames, e funzionano anche con un nodo in modalità pre-operational.

Per questo motivo solitamente vengono utilizzati per il setup iniziale di un nodo. [Bot, Eleb]

2.3.3 Protocolli di comunicazione, PDO

Il PDO è un servizio a basso overhead (solo 4 bytes), molto più veloce dell'SDO e viene utilizzato per trasmettere dati in real-time. Mentre per trasportare 8 bytes di dati ad un PDO basta un CAN frame, per trasportare la stessa informazione ad un SDO ne servirebbero 4 (due richieste e due risposte da 4 bytes ciascuna). Per questo motivo si usano i PDO per trasmettere feedback di posizione oppure per settare un target di velocità. [Eleb, Bot]

2.3.4 Profili di Moto, CiA 402

Il CiA 402 è un profilo standardizzato per drives elettrici, che implementa diverse modalità di funzionamento tra le quali Homing, Profile Position, Interpolated Position, Profile Torque e Velocity. Tutte queste modalità sono supportate dal drive Microphase, ma quella di nostro interesse è la modalità a target di velocità. Sono specificati inoltre un set di PDO generici di default disponibili per tutti i drives

Lo standard CiA 402 implementa anche una macchina a stati che permette l'attivazione e il blocco del drive e determina quali comandi siano accettati in base allo stato corrente o se sia possibile applicare potenza al motore. Questi stati cambiano quando ricevono una control-word dal controllore, ma possono cambiare anche in seguito ad eventi interni al drive (gestione errori). Lo stato attuale è indicato da una status-word che, insieme ad altri valori di stato, viene ricevuta dal controllore attraverso un TPDO (transmit process data object). La control-word e altri comandi, tra cui il target di velocità, vengono inviati attraverso un RPDO (receive process data object). [Ciab]

Capitolo 3

Architettura e componenti Software

Per permettere al computer di bordo di inviare comandi al drive e ricevere un feedback, è necessario trovare un modo per codificare i messaggi ROS in CAN-frames.

Benchè la comunicazione non sia ancora possibile, in questo capitolo si elencheranno i passaggi necessari per eseguire il setup del drive, dell'ambiente ROS, delle librerie e degli strumenti aggiuntivi necessari.

3.0.1 Setup drive

I parametri statici che definiscono il funzionamento del drive devono essere programmati attraverso una connessione seriale rs422 possibile attraverso il convertitore MOXA.

Per eseguire il setup del drive è stato necessario utilizzare Drive Watcher, un software proprietario fornito dal produttore dei motori. Questa suite sfortunatamente è compatibile solo con windows xp e necessita si un port non disponibile in una virtual machine, il che rende impossibile utilizzarla senza avere il sistema operativo installato. È stato quindi necessario utilizzare una vecchio computer esclusivamente per programmare i parametri del drive.

3.0.1.1 Abilitare MOXA

Per abilitare il convertitore MOXA è necessario per prima cosa installare il drivers da CD o file .zip della MOXA. Il programma è presente all'interno della cartella riferita al

convertitore MOXA che si possiede. Bisogna entrare nella cartella software e installare file .exe.

Per abilitare il port è necessario andare in Pannello di controllo > Sistema > Hardware > Gestione Periferiche.

1. Da Porte (COM e LPT) cliccare col tasto destro su MOXA USB e aprire Proprietà; in Port Settings impostare i seguenti valori : Baud Rate = 9600, Data its = 8, Parity = None, Stop bits = 1, Flowcontrol = None.

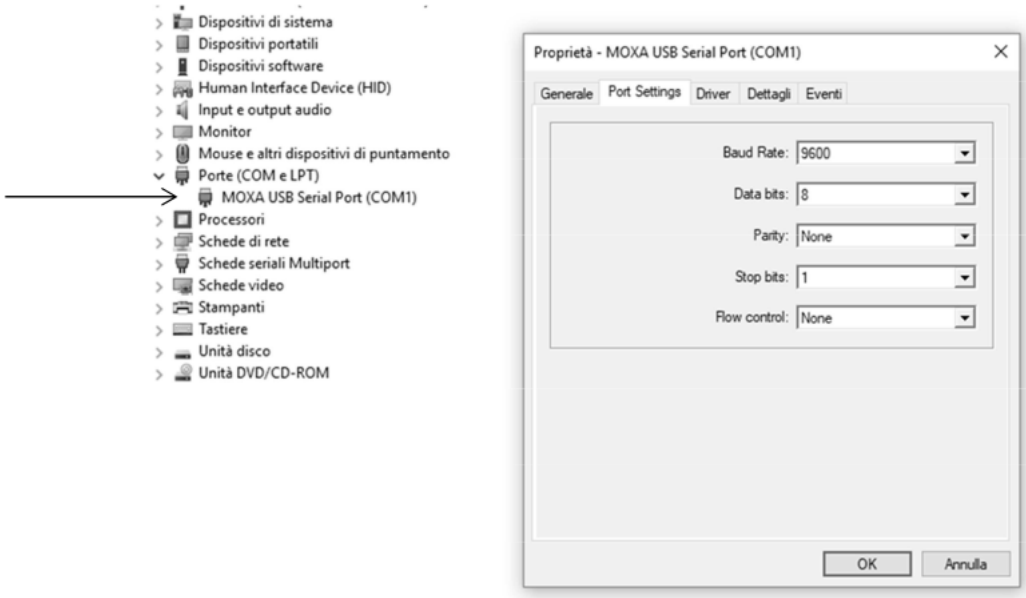


Figura 3.1: Installazione MOXA, parte 1

2. Da Schede Seriali Multiport cliccare col tasto destro su Uport 1130I e aprire Proprietà; in Ports Configuration aprire Port Setting della COM in uso dalla MOXA e impostare : Fast Flush su Enable e Interface su RS-422.

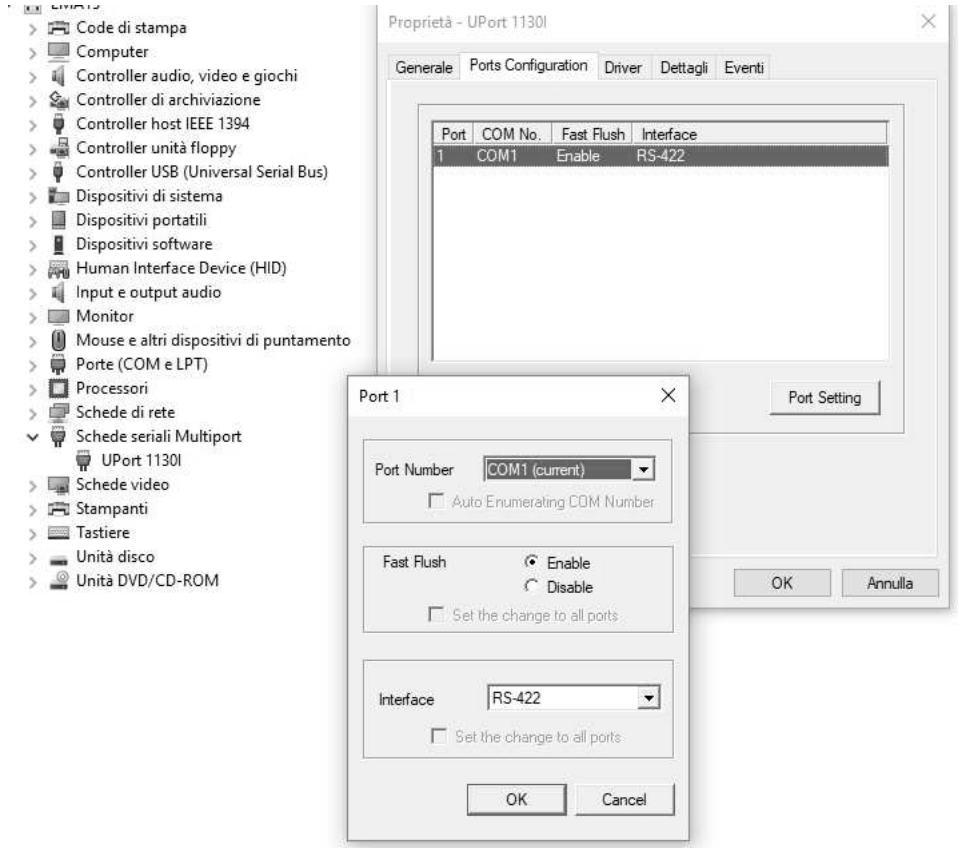


Figura 3.2: Installazione MOXA, parte2

3.0.2 Installazione Drive Watcher

Per installare il software basta semplicemente estrarre e aprire il file Setup-DriveWatcher-407.exe.

Prima di modificare i parametri bisogna accertarsi che la MOXA sia connessa al drive e che la parte logica del drive sia alimentata a 24V, con l'uscita negativa dell'alimentatore messa a terra.

Si esegue un settaggio della comunicazione attraverso la modalità "Communication Settings", impostando la ComPort designata in precedenza e cercando il nodo attivo attra-

verso il "search node tool". L'ultimo tool menzionato eseguirà uno sweep completo di tutti i nodi disponibili indicando quelli attivi in un colore differente.

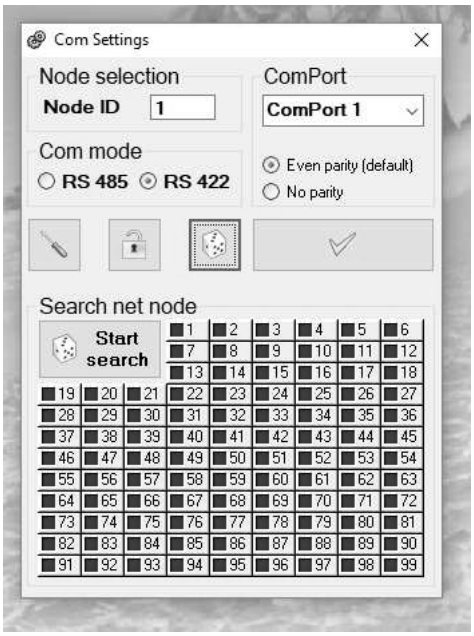


Figura 3.3: Communication Settings e Search Node Tool

Una volta impostato il nodo si controlla che la connessione sia attiva attraverso il tool "ComPort communication test" (icona a cacciavite).

3.0.3 Programmazione dei parametri

I parametri si possono visualizzare nella sezione "Parameters configuration"; ogni parametro dispone di una descrizione a lato, pertanto in questo documento non si elencheranno tutti, ma solo i più importanti ai fini del progetto. Per farsi un'idea più dettagliata si possono consultare il manuale Siboni "Istruzioni Drive Watcher" [Siba] e quello completo del drive Microphase "Micro Digital One", capitolo 9. [Mice]

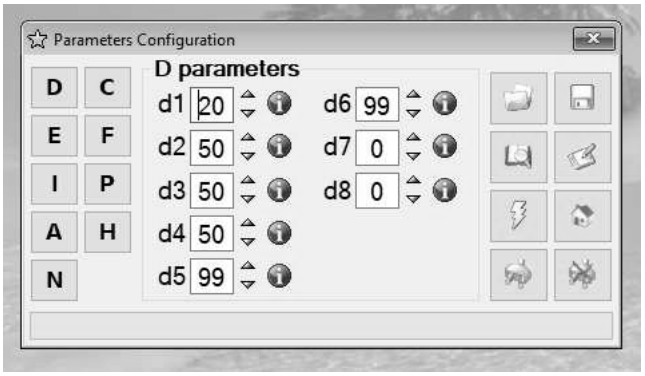


Figura 3.4: lo strumento Parameters Configuration

Fare particolare attenzione ai seguenti parametri:

C7 numero nodo = 1 di default

C9 modo di funzionamento del drive, impostato a 5 per il CANopen

D8 selezione del motore utilizzato, nel nostro caso brushless sinusoidale

N1 bitrate CAN, impostiamo a 5 per 250Kbaud, 6 per 500Kbaud

3.1 Setup ROS

L'installazione di ROS varia in base alle release. In questo progetto è stata utilizzata Melodic Morenia, ma in base alla distro di Linux installata sul computer di bordo potrebbe essere necessario installare una release differente. Ritenendo inutile descrivere il processo di installazione di una singola versione dal momento che è disponibile online ampia documentazione a riguardo, si rimanda al sito ufficiale di ROS, ove sono indicati tutti gli step necessari per le varie piattaforme: `https://wiki.ROS.org/ROS/Installation`.

Successivamente, si utilizza Catkin per creare un workspace in cui sviluppare il progetto:

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/
$ catkin_make
```

Per rendere utilizzabili con roscd e roslaunch i pacchetti aggiuntivi che installeremo nel workspace, è necessario aggiungere all'origine il nuovo file setup.*sh

Dall'interno della cartella catkin_ws, si scrive nel terminale:

```
$ source devel/setup.bash
$ echo $ROS_PACKAGE_PATH /home/youruser/catkin_ws/src:/opt/ROS/
kinetic/share
```

Questo comando, insieme a catkin_make, andrà ripetuto per sicurezza ad ogni utilizzo del workspace e, soprattuto, dopo l'installazione di nuovi pacchetti.

3.2 Setup PCAN

Benchè dei drivers per adattatore PCAN siano installati di default nel kernel Linux, sicchè i canali CAN sono gestiti come dispositivi netdev, per utilizzare il software di visualizzazione pcan_view e l'API pcan_basic è necessario installare i drivers proprietari.

I drivers sono disponibili sul sito della PEAK system all'indirizzo `https://www.peak-system.com/fileadmin/media/Linux/index.htm`.

Per installarli basta decomprimere il file e aprire la cartella su un terminale con i seguenti comandi:

```
$ make
$ sudo make install
```

Sullo stesso terminale, per attivare i drivers, si utilizza:

```
$ sudo modprobe peak_usb
```

Modprobe è un comando di Linux che aggiunge un modulo al kernel per estendere le sue funzionalità; in questo caso stiamo aggiungendo il modulo can0, necessario per utilizzare una linea seriale CAN e per il funzionamento dell'adattatore. Controlliamo che can0 sia attivo:

```
$ ip link show
```

Ip link è un altro comando che ci permette di controllare i dispositivi di rete disponibili e di modificarne le impostazioni. Ora si crea una linea can0, di tipo can, con baudrate a 250k in questo modo:

```
$ sudo ip link set up can0 type can bitrate 250000
```

Se vogliamo una baud a 500k basta modificare il bitrate. Ora il led dell'adattatore dovrebbe lampeggiare lentamente indicando una connessione attiva. Per disabilitare l'adattatore in caso di malfunzionamenti, si procede con il comando:

```
$ sudo ip link set down can0
```

3.3 Can-utils

Il primo tentativo di invio di frame CANopen ha utilizzato can-utils.

Can-utils è una suite di strumenti di analisi per terminale che comprende candump (log del traffico su linea can0) e cansend (invio di frame can).

È disponibile sul repository Debian, quindi per installarla basta il comando:

```
$ sudo apt-get install can-utils
```

Per vedere che messaggi passano attraverso la linea can0, il comando è:

```
$ candump can0
```

Per inviare un frame CAN, il comando è:

```
$ cansend can0 <CAN frame>
```

Il CAN frame ha questa forma: <can_id>#<data>

CAN_id definisce il nodo che deve ricevere i dati

data definisce il contenuto del frame, che a sua volta deve seguire lo standard CANopen

Cansend permette solo di inviare CAN frames codificati in binario o esadecimale, per inviare un frame CANopen è necessario specificare ogni volta il tipo di messaggio inviato (NMT, SDO, PDO) e leggere la risposta al livello più basso. I frames in uscita vengono registrati da candump, ma il drive non risponde, anche a causa dell'assenza di un segnale heartbeat da parte del computer, interpretabile dal drive come un errore di comunicazione.

3.4 CANopenSocket

Per ovviare a questo problema è stato utilizzato CANopenSocket, che permette di inviare frames CANopen ad un livello di astrazione più alto, risparmiando tempo ed evitando errori di codifica del messaggio. Esso permette inoltre di generare automaticamente un segnale di hearbeat. CANopenSocket fornisce anche il controllo della macchina a stati tramite comandi NMT di tipo verbose.

L'installazione di CANopenSocket si esegue clonando il repository in una cartella desiderata:

```
$ git clone https://github.com/CANopenNode/CANopenSocket.git
$ cd CANopenSocket
$ git submodule init
$ git submodule update
```

All'interno del pacchetto sono forniti due strumenti: canopend e canopencomm.

3.4.1 canopend

Canopend fornisce un'implementazione di un dispositivo CANopen con funzionalità di master, accessibile all'interno della cartella in cui la suite è installata con:

```
$ cd canopend
$ make
$ ./app/canopend --help
$ ./app/canopend can0 -i 2 -c " "
```

Il primo comando elenca le opzioni possibili di utilizzo con canopend, mentre il secondo è così strutturato:

can0 specifica l'interfaccia utilizzata

-i specifica l'indirizzo del nodo che si vuole creare; nel caso in cui non venga incluso l'indirizzo, esso viene preso dall'object dictionary contenuto nella cartella /objDict. Può essere qualsiasi indirizzo tranne 1, che è riservato al drive

-c abilita l'interfaccia di comando per funzionalità master; impostandolo come stringa vuota (" ") viene usato il socket di default, che ci permette di inviare messaggi agli altri nodi attraverso canopencomm

3.4.2 canopencomm

Mentre l'ultimo comando di canopend è attivo si apre un altro terminale e a partire dalla cartella principale di CANopenSocket:

```
$ cd canopencomm
$ make
$ ./canopencomm --help
```

Con le opzioni disponibili ora è possibile inviare frame CANopen attraverso l'interfaccia can0.

Per testarne il funzionamento si può andare a leggere la frequenza dell'heartbeat:

```
$ ./canopencomm [1] 4 read 0x1017 0 i16
```

Sorgono però una serie di complicazioni: innanzi tutto CANopenSocket permette di inviare frame con standard CiA 301, ma non con il CiA 402 che ci serve a controllare i profili di moto del drive. Sebbene sia possibile andare a modificare l'indirizzo dell'object dictionary che contiene ControlWord e VelocityTarget, il drive non risponde. Inoltre se si esegue un candump della linea seriale, si vede che la comunicazione viene interrotta dopo circa trenta secondi. Sono state tentate diverse soluzioni, e sebbene con un'interfaccia virtuale (vcan0), la comunicazione funzioni e non si riscontrino problemi di alcun genere, in questo caso non si è trovato nessuna soluzione al problema, che potrebbe essere causato da un adattatore difettoso, da un errore del driver o da interferenze sul bus.

3.5 Kacanopen[KIT]

Il fatto che non si sia avuto successo con CANopenSocket, ha suggerito di passare ad una libreria ROS che permettesse di connettersi direttamente con il drive attraverso l'adattatore PCAN e che contenesse al suo interno tutti gli strumenti necessari al controllo di un drive.

Kacanopen è sembrato un'ottima opzione, in quanto fornisce diversi strumenti utili: drivers per la maggior parte degli adattatori usb to CAN, un master CANopen integrato,

protocolli SDO, PDO ed NMT facilmente utilizzabili e un bridge ROS, che rende possibile pubblicare i nodi CANopen come nodi ROS in modo da poterli controllare semplicemente attraverso messaggi del tipo joint_state.

Un altro pregio della libreria è la documentazione estensiva.

L'installazione di Kacanopen si esegue scaricando la libreria da github all'interno del workspace ROS:

```
$ cd <catkin workspace>/src
$ git clone https://github.com/KITmedical/Kacanopen.git
$ cd ..
$ catkin_make –DDRIVER=socket
```

Lo studio della la libreria Kacanopen, ha evidenziato due problematiche che non la rendono adatta al progetto. In primo luogo la libreria Kacanopen non è aggiornata, è stata progettata per lavorare con ROS jade, che è una versione di ROS ormai deprecata.

Nonostante questo, la libreria potrebbe comunque funzionare, sebbene in maniera leggermente limitata. Infatti la modularità di ROS permette a nodi costruiti con una versione precedente di funzionare lo stesso, a patto che la sintasssi dei messaggi sia compatibile con le versioni più recenti, e che si possa eseguire la build della libreria a partire dal codice sorgente attraverso catkin_make.

Kacanopen funziona anche indipendentemente da ROS, utilizzando cmake per eseguire la build.

A rendere impossibile l'utilizzo di Kacanopen (almeno con il connettore PCAN) senza una sostanziale riscrittura del codice sono i drivers di cui ha bisogno per funzionare con un adattatore PCAN: il codice è infatti progettato per funzionare con una versione precedente dei drivers. È possibile installare i drivers compatibili con il comando:

```
$ wget http://www.peak-system.com/fileadmin/media/Linux/files/
peak-linux-driver-7.15.2.tar.gz tar -xzf peak-linux-driver-7.15.2.tar.gz
$ mv peak-linux-driver-7.15.2 ~/peak
$ cd ~/peak
$ make
```

[kac]

Una volta scaricati, non sembrano essere compatibili con la versione del sistema operativo, infatti durante il processo di installazione una grossa porzione del codice dà errori.

È stato fatto il tentativo di utilizzare Kacanopen con drivers aggiornati, aggiungendo i drivers peak_usb all'interno di /kacanopen/src/drivers_lgpl e utilizzando poi:

```
$ catkin_make –DDRIVER=peak_usb
```

ma non sono stati ottenuti i risultati sperati.

3.6 ros_canopen [rc]

Un'altra opzione che sembra promettente è la libreria ros_canopen. Parte del progetto ros-industrial[ri], che ha l'obiettivo di estendere le capacità di ROS ad hardware ed applicazioni industriali, ros_canopen è una libreria che fornisce il supporto di dispositivi CANopen all'interno di ROS.

Questo software fornisce gli stessi strumenti di Kacanopen, ma ha il pregio di essere aggiornato (il mantainer attuale è Mathias Lüdtke, `https://github.com/ipa-mdl`) e funziona con i drivers can integrati nel kernel Linux.

L'installazione è simile a Kacanopen, all'interno della cartella /src del workspace ROS:

```
$ git clone https://github.com/ros-industrial/ros_canopen
$ cd ..
$ catkin_make
```

La libreria è composta da diversi strumenti:

canopen_master implementa un master secondo lo standarc CiA 301, con supporto di sincronizzazione infra-processo.

canopen_chain_node controlla un bus CANopen con uno o più nodi che sono configurati attraverso un file EDS/DCF

canopen_motor_node fornisce un'interfaccia ROS_control a motori conformi allo standard CiA 402

socketcan_bridge espone i frames CAN ai messaggi ROS e viceversa

socketcan_interface fornisce un'interfaccia CAN generica e un'implementazione dei drivers basata su socketcan

canopen_402 contiene l'implementazione del protocollo CiA 402 DSP

can_msgs definisce un messaggio ros_canopen tipo

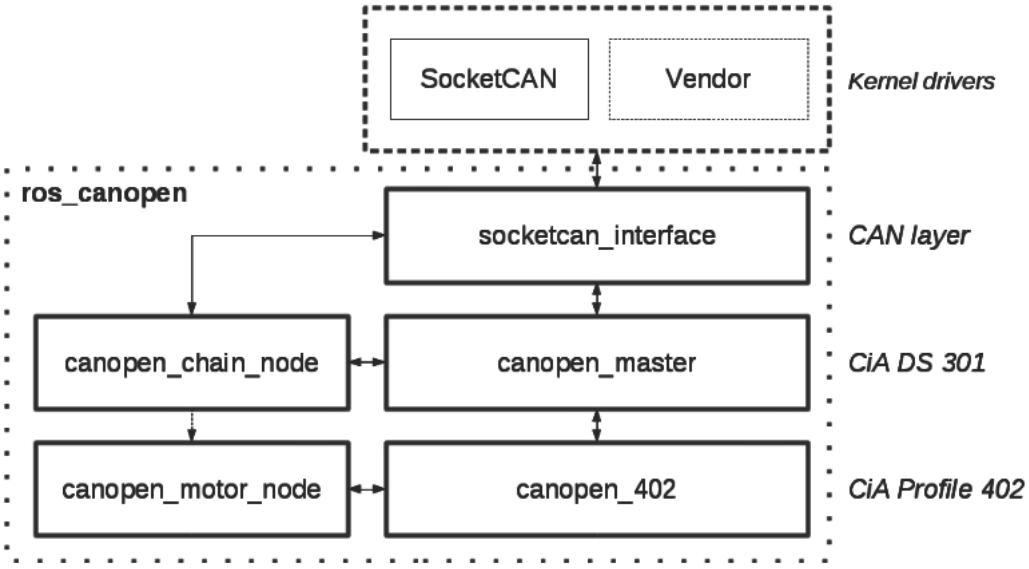


Figura 3.5: Schema di funzionamento di ros_canopen

3.7 File .edf .urdf

Per funzionare, sia Kacanopen che ros_canopen hanno bisogno di due file: EDS (Electronic Data Sheet) e URDF (Universal Robotic Description Format)

3.7.1 EDF

L'Electronic Data Sheet è un template fornito dall'azienda produttrice dell'object dictionary e contiene informazioni su tutti gli oggetti dell'OD, ma non sin i valori, che vengono specificati dal progettista finale.

Un esempio di oggetto all'interno del file .edf è il seguente:

```
[604E]
ParameterName=V1 Velocity Reference
ObjectType=0x07
```

```
DataType=0x0007
AccessType=rw
PDOMapping=0
```

Fortunatamente il drive Microphase usa lo standard CiA 402, per cui esiste già un file pronto all'uso tra quelli disponibili nella libreria Kacanopen.

Per utilizzi esterni a Kacanopen, il file 402.eds è stato inserito nella cartella /drivers e librerie/EDS nel repository github di questa tesi.

3.7.2 URDF

L'Universal Robotic Description Format descrive apputo il robot nella sua completezza da un punto di vista cinematico, ed è formattato in xml.

Per creare un file URDF si abbozza innanzitutto una lista dei collegamenti e delle articolazioni del robot[ROSb]:

```
<robot name="test_robot">
  <link name="link1" />
  <link name="link2" />
  <link name="link3" />
  <link name="link4" />

  <joint name="joint1" type="continuous">
    <parent link="link1"/>
    <child link="link2"/>
  </joint>

  <joint name="joint2" type="continuous">
    <parent link="link1"/>
    <child link="link3"/>
  </joint>

  <joint name="joint3" type="continuous">
    <parent link="link3"/>
    <child link="link4"/>
```

```
  </joint>
</robot>
```

Nel file vanno prima specificati i nomi dei vari collegamenti, poi i gradi di "parentela", vale a dire a quali collegamenti si attaccano le articolazioni.

In seguito vanno posizionate nello spazio le articolazioni attraverso il tag "origin" specificando la distanza e l'angolo di rotazione di ognuna rispetto alla sua origine; e infine con il tag "axis" viene specificato il modo in cui si muovono le articolazioni.

```
<robot name="test_robot">
  <link name="link1" />
  <link name="link2" />
  <link name="link3" />
  <link name="link4" />

  <joint name="joint1" type="continuous">
    <parent link="link1"/>
    <child link="link2"/>
    <origin xyz="5 3 0" rpy="0 0 0" />
    <axis xyz="-0.9 0.15 0" />
  </joint>

  <joint name="joint2" type="continuous">
    <parent link="link1"/>
    <child link="link3"/>
    <origin xyz="-2 5 0" rpy="0 0 1.57" />
    <axis xyz="-0.707 0.707 0" />
  </joint>

  <joint name="joint3" type="continuous">
    <parent link="link3"/>
    <child link="link4"/>
    <origin xyz="5 0 0" rpy="0 0 -1.57" />
    <axis xyz="0.707 -0.707 0" />
```

```
  </joint>
</robot>
```

Conclusioni

Dopo aver confrontato le possibili opzioni, l'approccio migliore per il completamento del progetto sembra essere quello di sfruttare la libreria ros_canopen.

Il problema principale è posto dalla mancata comunicazione del computer di bordo con il driver, di cui non si è riuscito a trovare la causa, nonostante estensivi tentativi di debug.

Il contesto "ambientale" dei mesi recenti, segnato dall'epidemia covid-19, ha influito negativamente sullo svolgimento del progetto e sui risultati ottenuti imponendo condizioni di lavoro e di osservazione limitative (lavoro completamente indipendente, assenza di supporto da parte di personale più qualificato, adattatori "artigianali", mancanza di un laboratorio e strumentazione adeguata...).

Ora, la riapertura delle sedi universitarie determina condizioni se non ottimali, certamente migliori, grazie alle quali è possibile attendere la soluzione di problemi sin qui incontrati.

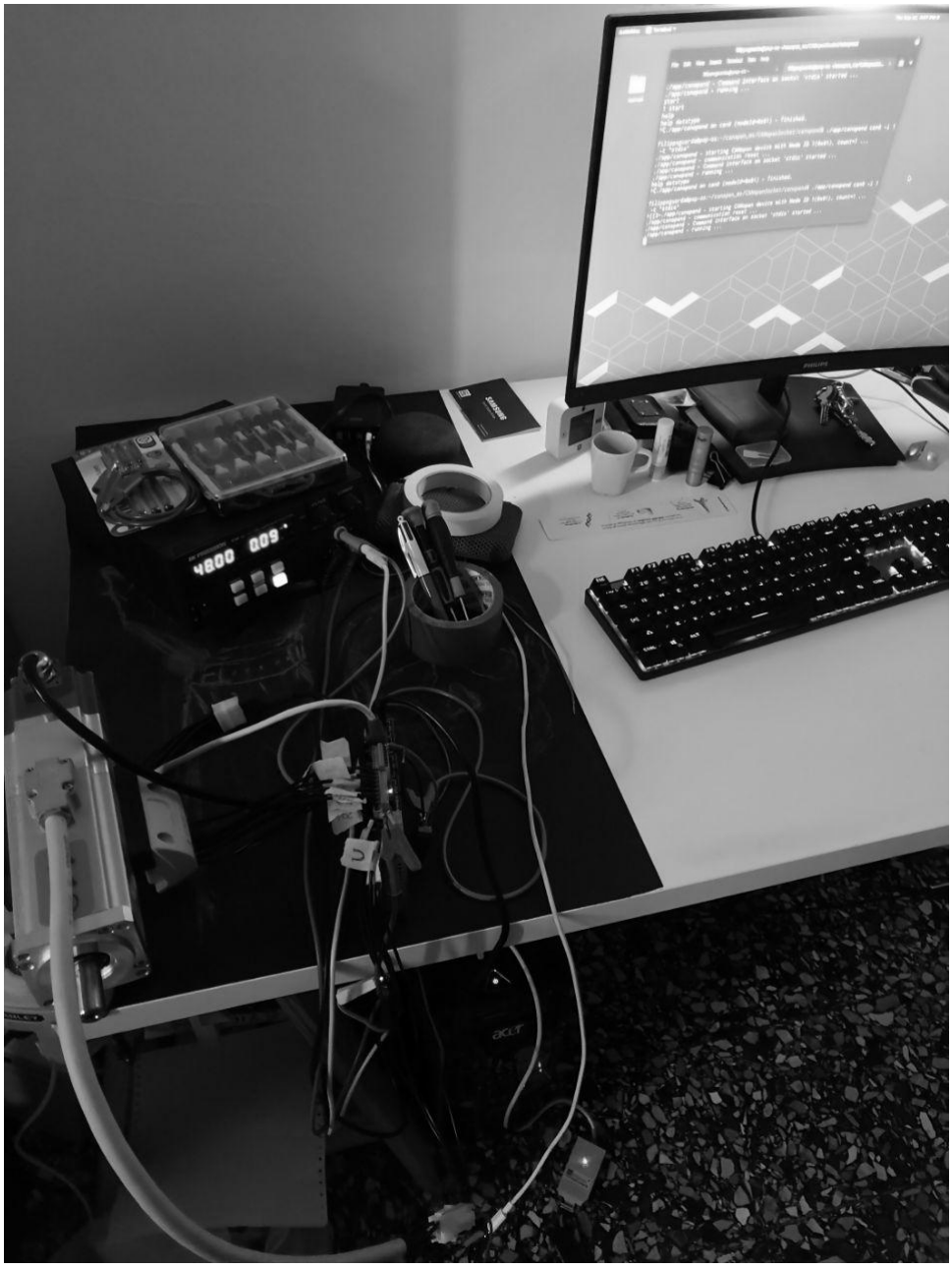


Figura 3.6: Ambiente di lavoro in cui il progetto è stato svolto

3.8 Debugging del connettore RJ45

Durante il lockdown e nei mesi successivi, in mancanza di un oscilloscopio e di un analizzatore logico, è stato impossibile accertarsi che il drive ricevesse effettivamente i segnali inviati. Il connettore PCAN dispone di un led che varia la frequenza di lampeggiamento in base alla presenza o meno di traffico sul bus, ma ciò non esclude che possano presentarsi dei problemi di interferenza o altre complicazioni nel trasporto dell'informazione al drive.

Per prima cosa sarebbe quindi opportuno accertare l'effettiva ricezione dei frame CAN da parte del drive concentrandosi sul controllo a un livello di astrazione più alto solo successivamente.

L'adattatore PCAN smette di funzionare dopo un determinato lasso di tempo: è da capire se ciò sia dovuto ad una incompatibilità con il sistema operativo o se il prodotto utilizzato sia difettoso. La seconda ipotesi può essere verificata utilizzando semplicemente un adattatore PCAN differente; per la prima invece risulta più difficile da accertare.

3.9 Utilizzare ros_canopen

La documentazione è abbastanza completa, anche se ridotta, e nel caso più disperato sarebbe sempre possibile contattare il mantainer per chiarimenti sul codice sorgente.

La sintassi dei messaggi è definita nel codice sorgente di /can_msgs.

Configurando i parametri del drive su /canopen_chain_node e /canopen_motor_node e collegando correttamente il bus dovrebbe essere possibile controllare i singoli nodi attraverso un messaggio del tipo joint_state, o in alternativa un interfaccia /ros_control[ROSf].

3.10 Utilizzare Kacanopen

Kacanopen è forse la libreria più facile da utilizzare; il suo unico problema è l'impossibilità di utilizzarla in sinergia con l'adattatore PEAK. Una soluzione possibile sarebbe quella di sostituire l'adattatore PEAK con un IXXAT usb-to-CAN FD[IXX], che tra i vantaggi ha anche quello di disporre di un attacco RJ45 automotive, senza quindi necessità di adattatori addizionali.

3.11 If everything else fails...

Il drive Micro Digital One può essere impostato per il controllo di velocità analogico attraverso un segnale +-10V, con una board arduino (facilmente controllabile con ROS) e con un circuito di pull-up si potrebbe creare un prototipo funzionante, anche se molto meno affidabile e preciso, con poche righe di codice.

Bibliografia

[Bog20] Mateusz Bogusz. Erc student rules. https://drive.google.com/file/d/1QK-eHO4zAySQ9jRCbzsc9qRgN-F9gWtm/view, 2020.

[Bot] H. Boterenbrood. Canopenhigh-level protocol for can-bus.

[CiAa] CiA. Can data lynk layer in some detail. https://can-cia.org/can-knowledge/can/can-data-link-layers/.

[Ciab] Cia. *CiA DSP 402 V 2.0*.

[CiAc] CiA. History of can technology. https://www.can-cia.org/can-knowledge/can/can-history/.

[Elea] Css Electronics. Can bus explained. https://www.csselectronics.com/screen/page/simple-intro-to-can-bus/language/en.

[Eleb] Css Electronics. Canopen explained, a simple intro. https://www.csselectronics.com/screen/page/canopen-tutorial-simple-intro/language/en.

[FH] Robert Bosch Florian Hartwich. The dawn of can. https://can-newsletter.org/uploads/media/raw/5f93ddcc3d04ff5d437a472f76c27af3.pdf.

[IXX] IXXAT. Usb-to-can fd. https://ixxat.com/products/automotive-solutions/overview/pc-interfaces/usb-to-can-fd.

[kac] kacanopen. Buid instructions. https://kitmedical.github.io/kacanopen/md_doc_Installation.html.

[KIT] KITmedical. kacanopen. https://kitmedical.github.io/kacanopen/md_doc_Installation.html.

[Mica] Microphase. *CANopen Additional Information*.

[Micb] Microphase. *Control Registers: Documentazione per le reti S-Net e S-CAN*.

[Micc] Microphase. *Datasheet MicroDigitalOne*.

[Micd] Microphase. *Micro Digital One interface*.

[Mice] Microphase. *Micro One Digital, Manuale di Istruzione*.

[PEA] PEAK. *PCAN-usb user manual*.

[rc] ros canopen. Package summary. https://wiki.ros.org/action/info/ros_canopen?action=info.

[ri] ros industrial. Ros industrial. https://rosindustrial.org/.

[ric] Pat richards. A can physical layer discussion. http://ww1.microchip.com/downloads/en/AppNotes/00228a.pdf.

[ROSa] ROS. catkin. https://wiki.ros.org/catkin.

[ROSb] ROS. Create your own urdf file. https://wiki.ros.org/urdf/Tutorials/Create%20your%20own%20urdf%20file.

[ROSc] ROS. History. https://www.can-cia.org/can-knowledge/canopen/canopen-lower-layers/.

[ROSd] ROS. Introduction. https://wiki.ros.org/ROS/Introduction.

[ROSe] ROS. msg. https://wiki.ros.org/msg.

[ROSf] ROS. ros control. https://wiki.ros.org/ros_control.

[ROSg] ROS. rviz. https://wiki.ros.org/rviz.

[ROSh] ROS. services. https://wiki.ros.org/Services.

[ROSi] ROS. Understanding ros nodes. https://wiki.ros.org/ROS/Tutorials/UnderstandingNodes.

[Siba] Siboni. *Istruzioni Drive Watcher ver. 4.07*.

[Sibb] Siboni. *ProPlanet servomotori brushless*.

[Uba] UP-board. Up core plus. https://up-board.org/upcore/specifications/.

[Ubb] UP-board. Up core plus specifications. https://up-board.org/wp-content/uploads/datasheets/UP-core-DatasheetV0.3.pdf.

Ringraziamenti

Grazie al professor Andrea Tilli per avermi fatto da relatore nonostante il poco preavviso, ai ragazzi di AlmaX, in particolare a Giosuè, per avermi introdotto ad un progetto interessante e pieno di sfide.

Grazie ad AlmaAutomotive per aver fornito l'hardware ed essere poi sparita, a Siboni per i motori, i drives e il software DriveWatcher.

Alla mia famiglia ed ai miei amici per il supporto continuativo nei momenti più frustranti, in particolare ad Adileo Barone, per le delucidazioni riguardanti il codice non documentato di ros_canopen e, insieme ad Annarita Angelini, per la revisione della bozza di questo testo.