

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

<b>Introduzione</b>	<b>1</b>
0.1 Pipeline . . . . .	1
0.2 Capitolo 1 . . . . .	2
0.3 Capitolo 2 . . . . .	2
0.4 Software . . . . .	3
<b>1 Architettura e componenti Hardware</b>	<b>5</b>
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
<b>2 Introduzione a ROS, CAN e CANopen.</b>	<b>13</b>
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	20
<b>3</b>	<b>Architettura e componenti Software</b>	<b>23</b>
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
<b>Conclusioni</b>	<b>39</b>	
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
<b>Bibliografia</b>	<b>43</b>



# 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 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 insieme al PCAN . . . . .	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 portato avanti . . . . .	40



# Introduzione

**L**a ERC, European Rover Challenge, è una competizione a livello europeo organizzata dal 2014 in Polonia. Anche l'~~U~~<sup>sedi di Bologna e Forlì</sup>iversità 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 cui:

- 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 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 ~~e' analisi~~ il ~~benchmark~~ della parte di motion control.

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~~ Obiettivo e architettura di riferimento

L'obiettivo di questa tesi è di permettere ad una base station di controllare i motori BLDC a bordo del rover attraverso messaggi in cui è specificato un target di velocità da raggiungere,

il quale 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, della presenza di errori.

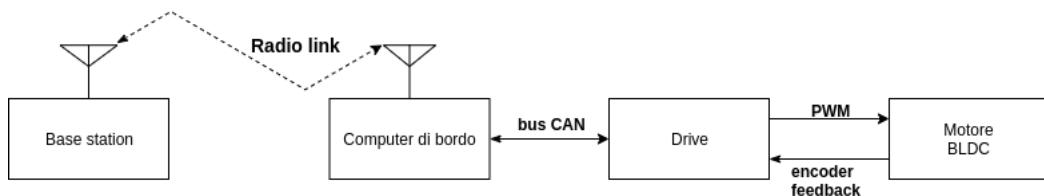


Figura 1: Pipeline del Motion Control

*Architettura del sistema di Motion control del Rover.*

## 0.2 Capitolo 1

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

Si mostrerà inoltre il corretto cablaggio degli adattatori che connettono il drive al computer di bordo, al motore e al PC windows necessario per la programmazione iniziale dei suoi parametri.

## 0.3 Capitolo 2

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

*per gestire → (cioè generare comandi/traiettorie target)  
("controllo", nel nostro ambito, lo lascerei alla  
accezione tipica dei controlli automatici)*

## 0.4 ~~Software~~ Capitolo 3

Nel terzo capitolo si elencheranno i diversi strumenti utilizzati nei successivi tentativi di comunicare con il drive, partendo da quelli a livello più alto, fino ad arrivare al candidato più promettente per eseguire un bridging efficace tra i messaggi di ROS e i frame CANopen.

## Introduzione

---

# Capitolo 1

## Architettura e componenti Hardware

### 1.1 Motori BLDC

O

bbiettivo del progetto è il controllo di un motore BLDC, in questo caso 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	$0.30kg * m^2 \times 10^{-4}$

→ adesso è sdoganato anche "Obiettivo",  
ma "Obiettivo" sarebbe molto meglio...  
  
non so se  
specificare  
che sono per  
la trazione  
o no.  
(o forse sono  
x altro?)

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 la seguenti schematiche[Mice]:

## CAPITOLO 1. Architettura e componenti Hardware

---

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



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



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 moduli disponibili 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 cui 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:

*comando*

### 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

Invece 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 lampeggiante lentamente: adattatore attivo
- LED lampeggiante velocemente: trasmissione in corso

## 1.5 Adattatori

Sia il convertitore MOXA che il PCAN utilizzano per la connessione seriale degli attacchi DB9, di cui solo una parte dei pin è effettivamente utilizzata. Questo è un problema in quanto il drive dispone solo di attacchi rj45. Per ovviare al problema sono stati cablati a mano due adattatori.

### 1.5.1 Adattatore per MOXA

Benchè dalla documentazione del convertitore moxa non venga specificato 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 con l'attacco fornito.



Figura 1.3: Adattatore db9-rj45 insieme al MOXA

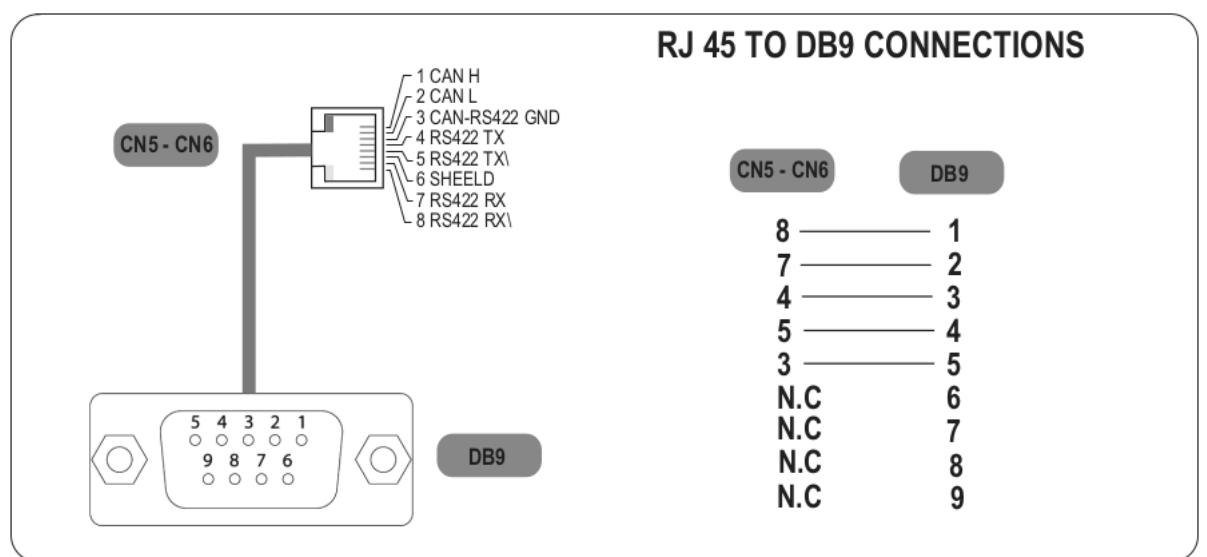


Figura 1.4: Schematica adattatore db9-rj45 per MOXA

### 1.5.2 Adattatore per PCAN

Anche l'adattatore PCAN dispone di un attacco db9, di cui solo 4 pin vengono utilizzati. 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 cablabilmente 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](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]:

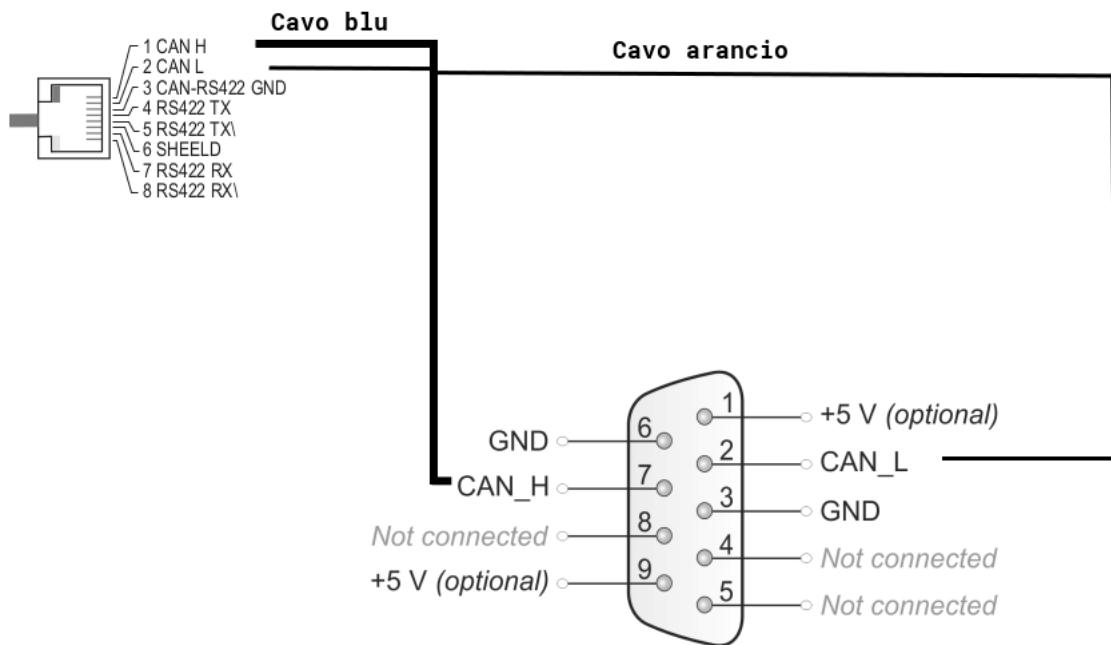


Figura 1.5: Schematica adattatore db9-rj45 per PCAN

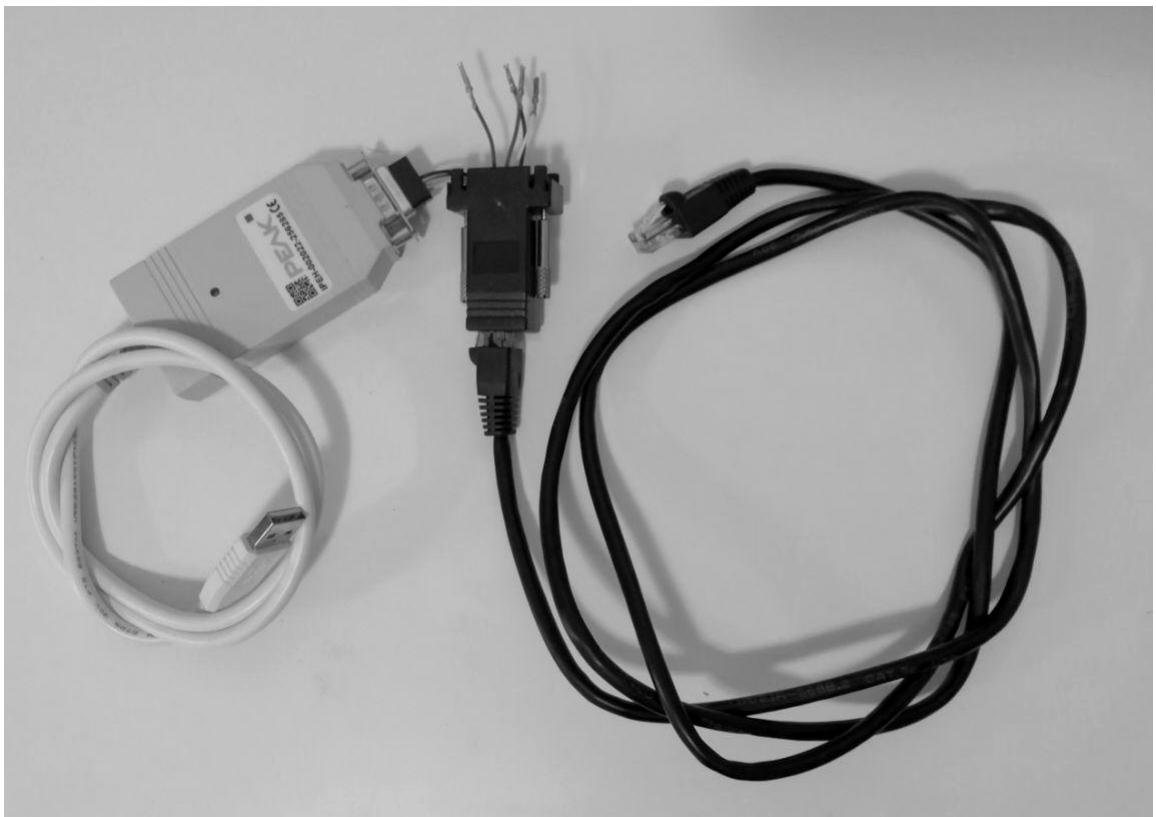


Figura 1.6: Adattatore db9-rj45 insieme al PCAN

## 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 1. Architettura e componenti Hardware

---

# Capitolo 2

## Introduzione a ROS, CAN e CANopen.

**I**l modello di motion control è basato su ROS, che facilita la comunicazione tra diverse macchine su cui 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, CANopen over CAN.

### 2.1 ROS

ROS, o 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" in cui 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, comunicano tra di loro inviando messaggi attraverso canali chiamati topics. I nodi possono svolgere fun-

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

Per questo 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 alla 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.

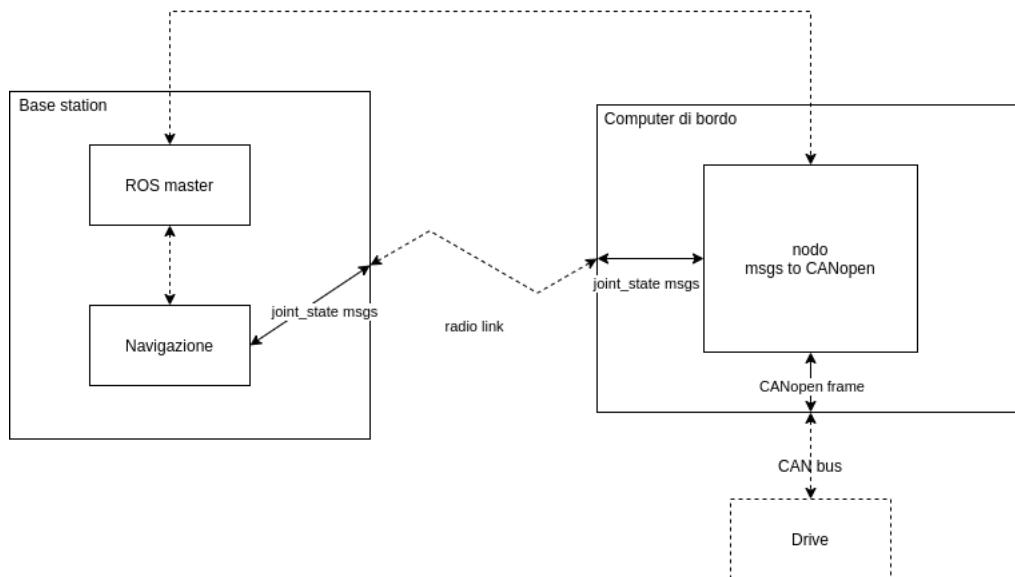


Figura 2.1: Ruolo svolto da ROS nel motion control

### 2.1.1 ROS master

Il nodo più importante nello schema di ROS è il master, infatti esso 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 definito, e sono alla base di ROS.

Ogni nodo per funzionare deve essere iscritto al master, ed esegue delle azioni in base agli input ricevuti. Queste azioni possono essere comunicate 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 cui i nodi inviano messaggi pubblicandoli sul topic, oppure li ricevono se iscritti al topic.

Pubblicare e sottoscriversi ad un topic è anonimo, quindi 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 richiederlo 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 che in remoto, 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", 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, automazione, ascensori e strumentazione medica.

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

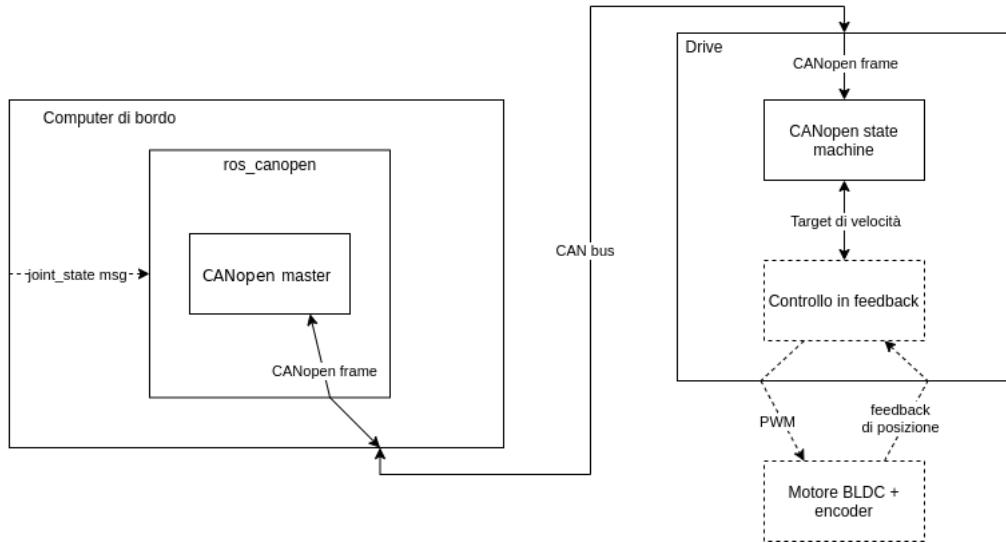


Figura 2.2: Ruolo svolto da CAN e CANopen nel Motion control

### 2.2.1 Livello fisico

CAN è un bus seriale producer-consumer in cui 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. Vengono riportati in uno stato recessivo da un resistore passivo, con CAN-H a tensione più bassa di CAN-L, quando non viene più fornita tensione. 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 byte.

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 byte)
6. Cyclic Redundancy Check (16 bit)
7. ACK (2 bit)
8. End Of Frame (7 bit)

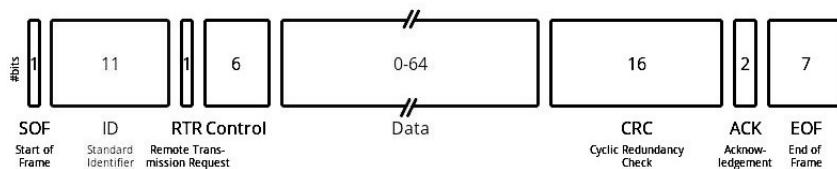


Figura 2.3: Frame CAN standard

[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]).

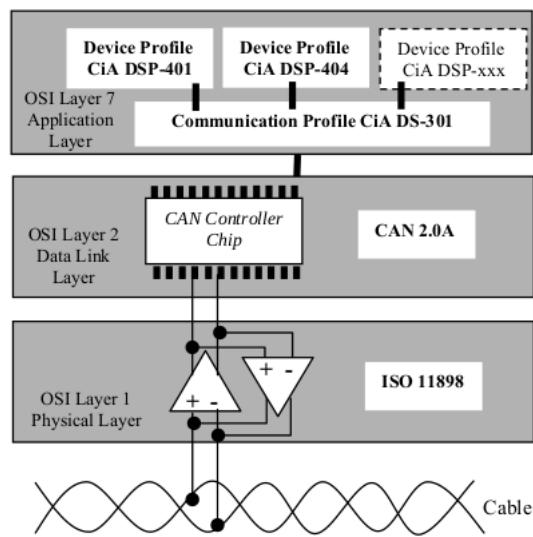


Figura 2.4: Schema degli standard CAN e CANopen nel modello OSI [Bot]

CANopen integra al suo interno diversi concetti:

### 2.3.1 Object dictionary

Ogni nodo della rete 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 byte, e ad ogni richiesta segue una risposta. Quindi per leggere 4 byte di informazione ho bisogno di 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 invece è un servizio a basso overhead (solo 4 byte) molto più veloce dell'SDO, viene utilizzato per trasmettere dati in real-time. Mentre per trasportare 8 byte 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 byte 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 cui 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 specificate inoltre uno set di PDO generiche 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 sono accettati in base allo stato corrente o se è possibile applicare potenza al motore. Questi stati cambiano quando ricevono una control-word dal controllore, possono anche cambiare 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 PDO(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 2. Introduzione a ROS, CAN e CANopen.

---

# Capitolo 3

## Architettura e componenti Software

**P**er permettere al computer di bordo di inviare comandi al drive e ricevere un feedback, è necessario trovare un modo di codificare i messaggi ROS in CANframes.

Benchè la comunicazione non sia ancora possibile, in questo capitolo si elencheranno i passaggi necessari ad eseguire il setup del drive, dell'ambiente ROS, delle librerie e 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 di un port non disponibile in una virtual machine il che rende impossibile utilizzarla senza avere il sistema operativo installato. È stato quindi necessario utilizzare un 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 driver 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.

## CAPITOLO 3. Architettura e componenti Software

---

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 bits = 8, Parity = None, Stop bits = 1, Flowcontrol = None.

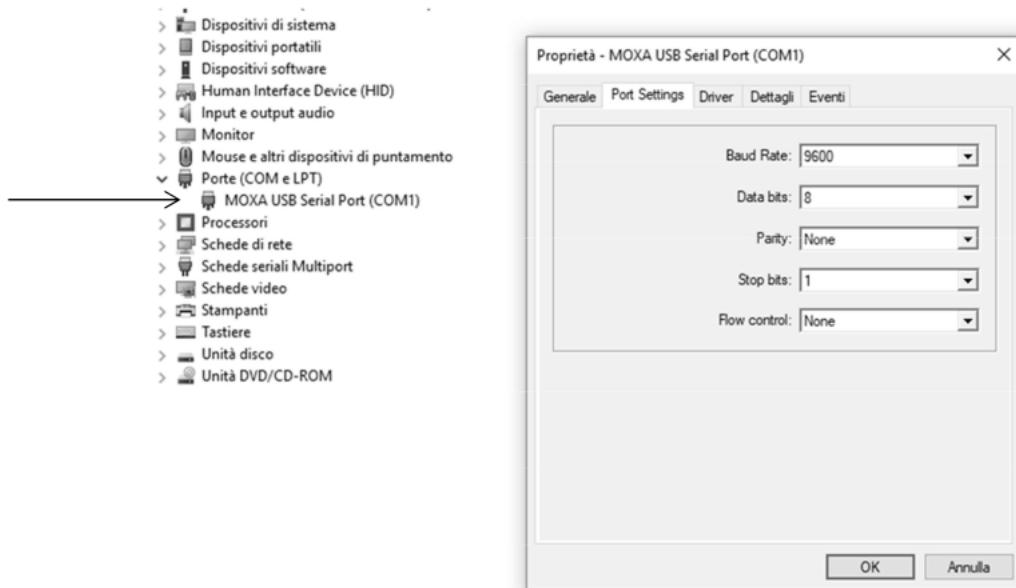


Figura 3.1: Installazione moxa, parte 1

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.

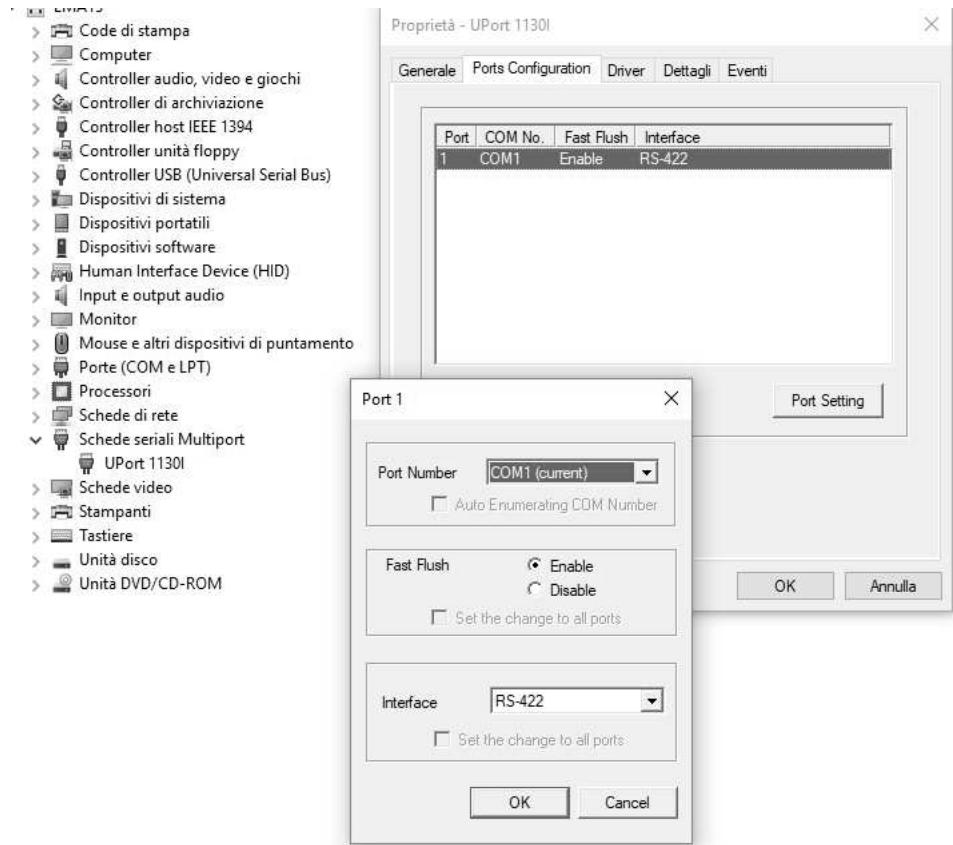


Figura 3.2: Installazione moxa, parte2

### 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 driver e che la parte logica del driver 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 con un colore differente.

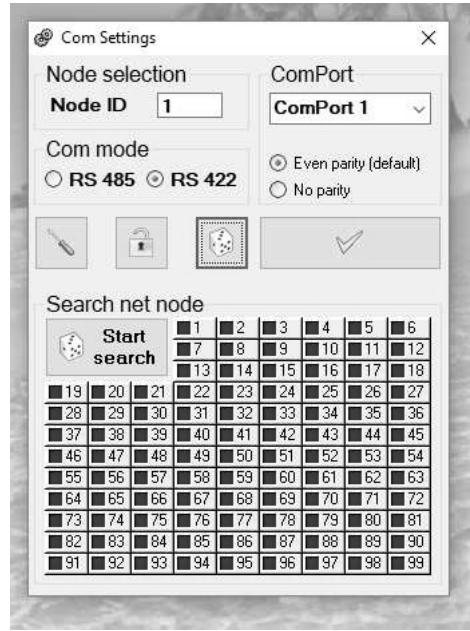


Figura 3.3: Communication Settings e Search Node Tool

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, quindi 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]

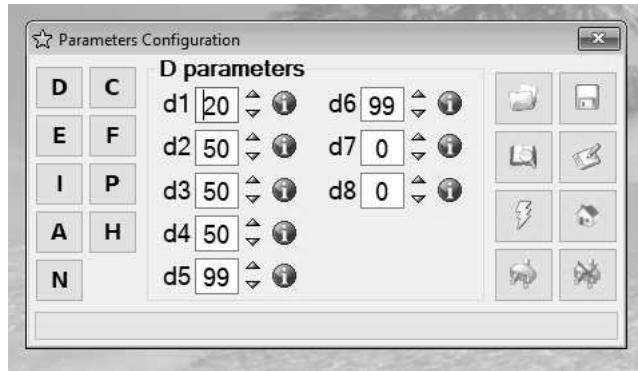


Figura 3.4: lo strumento Parameters Configuration

Fare particolare attenzione ai 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 quando è disponibile online ampia documentazione a riguardo, si include qui il link al sito ufficiale di ROS, dove 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 roscl e ros-launch 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 soprattutto dopo l'installazione di nuovi pacchetti.

## 3.2 Setup PCAN

Benchè dei driver per adattatore PCAN siano installati di default nel kernel linux, per cui 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 driver proprietari.

I driver 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, aprire la cartella su un terminale con i seguenti comandi:

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

Sullo stesso terminale, per attivare i driver, 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:

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

### 3.3 Can-utils

Il primo tentativo di invio di frame CANopen è stato fatto utilizzando 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:

```
$ candump can0
```

Per inviare un frame CAN:

```
$ 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 frame in uscita vengono registrati da candump, ma il drive non risponde, anche a causa dell'assenza di un segnale heartbeat da parte del computer, che potrebbe essere interpretata 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, oltre a generare automaticamente un segnale di heartbeat. CANopenSocket inoltre fornisce 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 vogliamo creare, se non viene incluso l'indirizzo viene preso dall'object dictionary incluso nella cartella /objDict, può essere qualsiasi indirizzo a parte 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 possiamo 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 provate diverse soluzioni, ma mentre con un’interfaccia virtuale (vcan0), la comunicazione funziona e non si hanno problemi di alcun genere, non si è trovato niente che potesse risolvere questo problema.

## 3.5 Kacanopen[KIT]

Non riscontrando successo con CANopenSocket, si è deciso 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 al suo interno 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 disponibili 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
```

Studiando la libreria Kacanopen, si sono incontrate due problematiche che non la rendono adatta al progetto. In primo luogo la libreria Kacanopen non è mantenuta, è stata progettata per lavorare con ROS jade, una versione di ROS ormai deprecata e mai aggiornata.

Nonostante questo la libreria potrebbe comunque funzionare, anche se 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 sintassi 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, basta utilizzare cmake per eseguire la build.

Il problema che rende impossibile utilizzare Kacanopen 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 driver compatibili con:

```
$ 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]

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

È stato fatto un tentativo di utilizzare Kacanopen con driver aggiornati, aggiungendo i driver 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** implementazione di 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 driver basata su socketcan

**canopen\_402** contiene l’implementazione del protocollo CiA 402 DSP

**can\_msgs** definizione di messaggio ROS per CAN

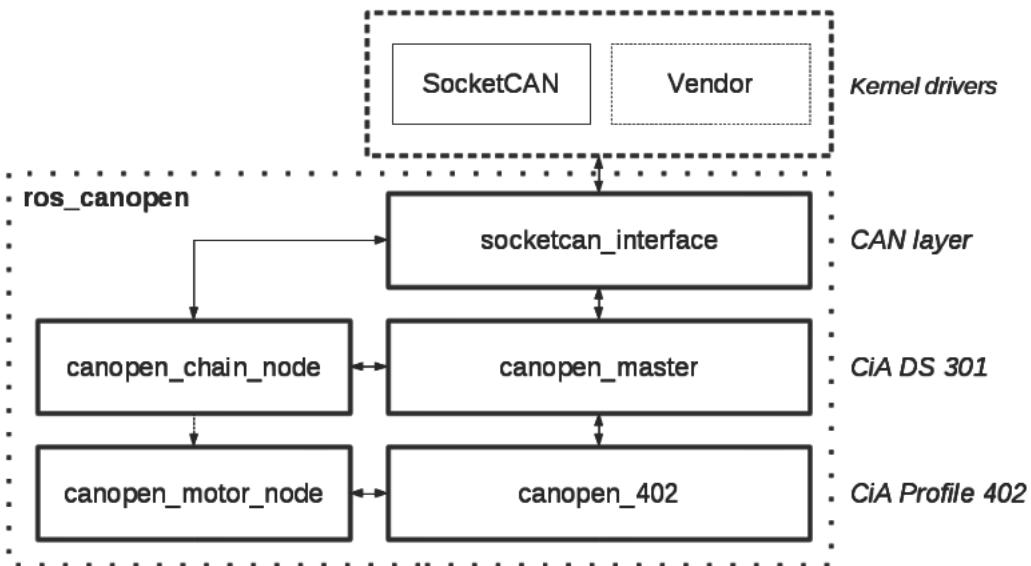


Figura 3.5: Schema di funzionamento di ros\_canopen

### 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, senza però 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 appunto il robot nella sua completezza da un punto di vista cinematico, ed è formattato in xml.

Per creare un file URDF si abbozza prima 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 specifichiamo prima i nomi dei vari collegamenti, poi i gradi di "parentela", cioè a quali collegamenti si attaccano le articolazioni.

In seguito posizioniamo 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" specifichiamo come 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>
```

Se possibile:

consiglierei di riportare un paragrafo anche breve con un report succinto dei tentativi sperimentali fatti, almeno per ROS - coreopen.

Metterei qui la foto del set up e qualche schermata che spieghi dove si è bloccata la comunicazione -

Questo darebbe conto esplicito di tutta la parte sperimentale -

Ho visto che ci sono dei paragrafi dopo le conclusioni (oltre tutto con numerazione del CAP 3), forse basta semplicemente spostare quei paragrafi.



# Conclusioni

 **C**onfrontando 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 ~~X~~, di cui non si è riuscito a trovare la causa, nonostante estensivi tentativi di debug.

Le particolari condizioni in cui è stato portato avanti questo progetto a causa del coronavirus (lavoro completamente indipendente, assenza di supporto da parte di personale più qualificato, adattatori "artigianali", mancanza di un laboratorio e strumentazione adeguata... ) ha sicuramente influito negativamente sui suoi risultati, ma con la riapertura dell'università si aprono nuove possibilità di risolvere le problematiche incontrate finora.

## Conclusioni

---



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

### 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. Benchè il connettore PCAN disponga di un led che varia la frequenza di lampeggiamento in base alla presenza o meno di traffico sul bus, potrebbero esserci dei problemi di interferenza o altre complicazioni nel trasporto dell'informazione al drive.

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

Inoltre l'adattatore PCAN smette di funzionare dopo un certo lasso di tempo, è da capire se il problema sia dovuto ad una incompatibilità con il sistema operativo o se il prodotto sia difettoso, la seconda ipotesi si può accantonare semplicemente utilizzando un adattatore PCAN differente, per la prima invece il problema sarebbe di più difficile soluzione.

### 3.9 Utilizzare ros\_canopen

Benchè non sia stato possibile terminare il lavoro in tempo utile, un'opzione è lavorare con 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 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 bisogno 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>.

## BIBLIOGRAFIA

---

- [kac] kacanopen. Buid instructions. [https://kitmedical.github.io/kacanopen/md\\_doc\\_Installation.html](https://kitmedical.github.io/kacanopen/md_doc_Installation.html).
- [KIT] KITmedical. kacanopen. [https://kitmedical.github.io/kacanopen/md\\_doc\\_Installation.html](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](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](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>.

## BIBLIOGRAFIA

---

# **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, 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 frustanti, in particolare ad Adileo che in un'ora è riuscito a spiegarmi quello che in una settimana non avevo capito da solo.