



Università degli Studi di Bergamo

FACOLTÀ DI INGEGNERIA
Corso di Laurea Magistrale in Ingegneria Informatica

Relazione di Robotica

Laureandi:

Pitturelli Leandro

Matricola 1043991

Sardi Dario

Matricola 1043991

Indice

1	[Introduzione]	2
2	ROS per NVIDIA Jetson TX2	2
2.1	Setup tramite tool Jetpack	2
2.2	Backup e ripristino	2
2.2.1	Backup	2
2.2.2	Ripristino	2
2.3	Possibile errore	3
2.4	Setup di Ubuntu	3
2.4.1	Setup tastiera	3
2.4.2	Installazione ROS	3
3	ROS: COMUNICAZIONE SERIALE TRAMITE USB	4
4	ZED	8
4.1	ZED SDK	8
4.2	ZED-ROS WRAPPER	9
4.3	RVIZ	9
4.4	STAR-MARKER-FINDER	9
4.5	Marker-finder su architettura distribuita	10
5	eProxima DDS	12
5.1	ROS 2	12
5.2	eProxima	12
5.3	installazione	13

1 [Introduzione

2 ROS per NVIDIA Jetson TX2

Questa parte del progetto ha richiesto l'installazione e la configurazione dell'ambiente ROS sulla scheda Jetson TX2. Tra le operazioni preliminari è stata effettuata l'installazione di L4T (Linux For Tegra) sulla scheda per fornire kernel, driver, librerie e un sistema operativo correttamente configurato per l'architettura specifica della Jetson.

2.1 Setup tramite tool Jetpack

La scheda Jetson TX2 é dotata di pulsanti per il controllo diretto. Per l'accensione bisogna premere il tasto contrassegnato sulla pcb con il label 'POWERBTN'.

La scheda inizialmente necessita di un flash tramite un tool chiamato 'JetPack' scaricabile dal sito NVIDIA da eseguire su un PC dotato di porta USB.

Come prima cosa occorre collegare la Scheda Jetson a internet attraverso un cavo ethernet connesso allo stesso router del pc su cui eseguiamo Jetpack mantenendo la scheda spenta. Sulla scheda sono presenti 4 tasti rossi evidenziati dal riquadro rosso nella Figura 1 che, dall'alto al basso, sono: RESET, VOL-, RECOVERY, POWER ON/OFF.

Avviare la scheda Jetson in 'recovery mode' seguendo questa sequenza di operazioni:

1. accendere la scheda premendo il pulsante di POWER ON/OFF .
2. tenendo premuto il pulsante RECOVERY premere e rilasciare il pulsante RESET.
3. rilasciare dopo 2 secondi dalla pressione del pulsante RESET il pulsante RECOVERY.
4. Collegare il cavo micro USB-B alla scheda e l'altra estremità ad una porta USB sul pc su cui si eseguirà Jetpack.

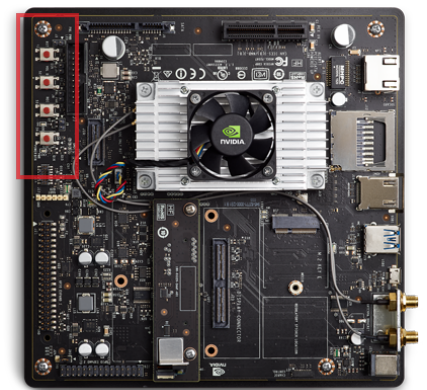


Figura 1: Jetson TX2.

Una volta scaricato l'archivio dal sito ufficiale ed estratto in una cartella su un pc (non quindi sulla Jetson) avviare da terminale (le istruzioni sono per S.O. linux) il file eseguibile *Jetpack-L4T-versione-linux-x64.run* tramite il comando `./Jetpack-L4T-versione-linux-x64.run`, seguire le istruzioni dell'interfaccia che si presenterà a video per selezionare cartella di installazione, modello della scheda e selezione dei pacchetti che si desidera installare sulla Jetson quali openCV e il toolkit CUDA. Accettare eventualmente i termini e le condizioni d'uso una volta premuto 'Next'. Nella scheda 'Network layout' selezionare la scelta coerente con la configurazione di rete attuale della nostra scheda ovvero la prima: *'Device access internet via router/switch'*. Selezionare la propria interfaccia di rete connessa a internet. Nel dubbio utilizzare il comando da terminale `'ifconfig'` e cercare a quale interfaccia di rete è assegnato un indirizzo IPv4. Dopo la schermata riassuntiva, alla pressione del tasto *next* si presenterà una finestra del terminale per visualizzare i progressi dell'installazione. In caso di errori relativi al collegamento della Jetson ripetere la sequenza di operazioni per porla in recovery mode.

2.2 Backup e ripristino

Una volta eseguito Jetpack e scaricate le dipendenze verranno creati in specifiche cartelle script per eseguire operazioni manualmente sulla Jetson quali, ad esempio backup e ripristino.

Per fare ciò bisogna navigare nella cartella contenente L4T (nel mio calcolatore la directory era `'64_TX2/Linux_for_Tegra'`) dove sarà presente il file `'flash.sh'`.

Si apra dunque un terminale in questa posizione e si colleghi la scheda al pc in modalità recovery.

2.2.1 Backup

Utilizzare il comando `"sudo ./flash.sh -r -k APP -G backup.img Jetson-tx2 mmcblk0p1"` che eseguirà il backup di tutta la memoria dell'emmc da 30Gb della scheda generando due files `'backup.img'` e `'backup.img.raw'` entrambi da conservare.

2.2.2 Ripristino

Per ripristinare un backup posizionare il `"backup.img"` nella directory contenente `flash.sh` e copiare il file `backup.img.raw` nella cartella bootloader rinominandolo `system.img`. Aprire ora un terminale nella cartella contenente `flash.sh` e digitare `"sudo ./flash.sh -r -k APP Jetson-tx2 mmcblk0p1"`, questo comando si occuperà di eseguire il flash della partizione p1 del disco `mmcblk0` (l'emmc della scheda.)

2.3 Possibile errore

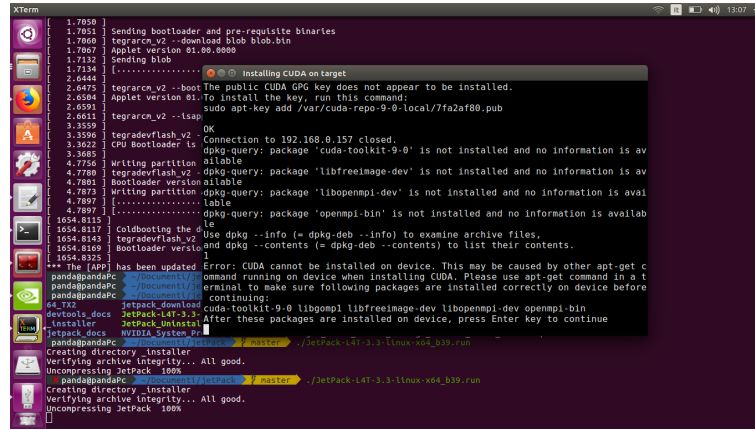


Figura 2: errore durante il flash.

Nel periodo di utilizzo si è verificato un errore durante il flash della scheda dovuto all'interruzione della connessione internet. Tra le possibili cause, i filtri sulla rete universitaria o a una momentanea instabilità della connessione. Per risolvere il problema è stato sufficiente collegare la tastiera alla scheda Jetson, installare manualmente i pacchetti mancanti tramite apt-get e infine premere enter come indicato da Jetpack nel terminale.

N.B. eseguire "sudo apt-get update" sulla Jetson prima di qualsiasi apt-get install per aggiornare le repositories.

2.4 Setup di Ubuntu

2.4.1 Setup tastiera

Avviato ubuntu il layout di tastiera potrebbe essere errato. Per cambiarlo digitare in un terminale 'sudo dpkg-reconfigure keyboard-configuration':

1. accettare 'Generic 105-key (intl) PC'
2. selezionare Italian nel menù del paese di origine della tastiera
3. accettare le restanti impostazioni senza modificarle

2.4.2 Installazione ROS

Seguire le indicazioni per l'installazione di ros-kinetic dal [sito ufficiale per ubuntu ARM](#).

3 ROS: COMUNICAZIONE SERIALE TRAMITE USB

Per interfacciarsi con board quali Arduino tramite ROS è risultato necessario un metodo per la comunicazione seriale tramite porta USB per poter ricevere e inviare dati in modo semplice ed efficace. Tale metodo è stato sviluppato pensando ad un utilizzo con board Arduino su macchina linux e sviluppato in C++ per essere compatibile con ROS. Tutte le board Arduino sono dotate di un chip UART (Universal asynchronous receiver-transmitter) incaricato della comunicazione seriale. Il metodo di comunicazione asincrono si basa sull'invio di dati ad una velocità predefinita e con dati strutturati in modo ben preciso:

1. Bit contenenti parte del messaggio complessivo di grandezza predefinita (dai 5 ai 9 in base alle impostazioni)
2. Bit di sincronizzazione per comunicare l'inizio e la fine di ciascun pacchetto di dati
3. Bit di parità per il controllo di errori
4. velocità di trasmissione dei dati per impostare su ogni singolo invio la velocità di invio e ricezione

Sono state sperimentate diverse librerie per la comunicazione seriale e metodi basati su programmazione seriale per sistemi POSIX scegliendo infine la libreria boost per la sua semplicità e per la documentazione online relativamente completa. Il codice scritto per la comunicazione è il seguente:

```
1  #include <unistd.h>
2  #include <iostream>
3  #include <boost/asio.hpp>
4  #include <boost/asio/serial_port.hpp>
5  #include <boost/system/error_code.hpp>
6  #include <boost/system/system_error.hpp>
7  #include <boost/bind.hpp>
8  #include <boost/thread.hpp>
9  #include <boost/date_time/posix_time/posix_time.hpp>
10 #include <boost/thread/thread.hpp>
11 #include <thread>
12
13 class SerialIO{
14 private:
15     boost::asio::io_service io_svc;
16     boost::asio::serial_port m_port;
17 public:
18     SerialIO(std::string portDir):m_port(io_svc,portDir){}
19
20     void write(std::string input){
21         const int BUFFSIZE=20;
22         boost::array<char,BUFFSIZE> buf;
23         if(sizeof(input)>0){
24             for(int i=0;i<sizeof(input)/8;i++){
25                 buf[i]=input[i];
26             }
27             buf[sizeof(input)/8]=0;
28             m_port.write_some(boost::asio::buffer(buf,BUFFER));
29         }
30     }
31
32     void read(){
33         const int BUFFSIZE=10;
34         char read_msg_[BUFFSIZE];
35         std::stringstream ss;
36         int read_size=0;
37         read_size=m_port.read_some(boost::asio::buffer(read_msg_,BUFFSIZE));
38         while(read_msg_[read_size-1]!='\n'){//ciclo fino al fine stringa
39             read_msg_[read_size]=0;//terminatore stringa inserito manualmente
40             ss<<read_msg_;
41             read_size=m_port.read_some(boost::asio::buffer(read_msg_,BUFFSIZE));
42         }
43         read_msg_[read_size]=0;
44         ss<<read_msg_;
45         std::cout << ss.str();
46     }
};
```

Le funzioni di read e write nella comunicazione asincrona stabilita sono bloccanti, pertanto la coordinazione dei messaggi deve essere fatta a monte, non essendo gestita dal programma. Questa libreria espone i due metodi di read e write e necessita di un'inizializzazione tramite il path del file che punta al dispositivo hardware nella cartella /dev/ (ad esempio un path valido può essere "/dev/tty1"). Il punto chiave della funzione read si trova alla riga 37 dove viene invocata la funzione di lettura dalla libreria boost:

```
37 read_size=m_port.read_some(boost::asio::buffer(read_msg_,BUFSIZE));
```

La funzione read_some legge in modo asincrono parte del messaggio, lo copia nel buffer creato ad hoc dal nome read_msg_ e di dimensione BUFSIZE, ritornando la dimensione del messaggio letto. Il ciclo while successivo serve per giungere al finestringa e copiare in uno stringstream il contenuto dei vari pacchetti del messaggio asincrono.

```
38 while(read_msg_[read_size-1]!='\n')
```

Viene ripetuta la lettura fino a quando non viene trovato come ultimo carattere letto il terminatore. Si presuppone quindi che il codice inviato tramite Arduino comunichi con una println sulla seriale o che sia inserito il terminatore di stringa manualmente al termine di un messaggio con una semplice print.

```
39 read_msg_[read_size]=0;  
40 ss<<read_msg_;
```

Poichè il buffer creato da asio risultava essere non vuoto, la copia del buffer nello stringstream senza aggiungere il terminatore manualmente generava stringhe errate. La riga 39 ovvia a questo problema.

```
45 std::cout << ss.str();
```

Il messaggio viene stampato sullo standard output utilizzando la funzione di conversione da stringstream a stringa. Questa parte del codice può essere modificata per eventualmente effettuare un post in un topic ros. La funzione write è piuttosto simile alla read:

```
21 const int BUFSIZE=20;  
22 boost::array<char,BUFSIZE> buf;
```

Viene creato un buffer di dimensione massima 20:buffer troppo grossi rallentano notevolmente il programma e causano blocchi. 20 è un numero adeguato ma il valore può essere modificato secondo necessità.

```
24 for(int i=0;i<sizeof(input)/8;i++){  
25     buf[i]=input[i];  
26 }
```

La stringa fornita come input viene copiata nel buffer. Il ciclo for è stato usato per l'impossibilità di usare funzioni quali strcpy o alternative più efficienti.

```
27 buf[sizeof(input)/8]=0;
```

Viene aggiunto manualmente un terminatore nel buffer onde evitare errori.

```
1 m_port.write_some(boost::asio::buffer(buf,sizeof(input)));
```

Viene inviato il messaggio contenuto nel buffer specificando la dimensione del messaggio da inviare.

Per una più facile inizializzazione della comunicazione seriale è stata scritta una libreria sempre in C++ per trovare un Arduino e ritornare il path del suo riferimento necessario per l'inizializzazione.

deviceFinder.h

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <errno.h>
4 #include <malloc.h>
5 #include <iostream>
6 #include <string.h>
7 #define BUF_SIZE 1024
8 char addr[30]="";
9 int search(void)
10 {
11     FILE *f;
12     char* buf;
13     f=popen("./finder.o | grep Arduino | awk '{print $1;}' ", "r");
14     if (f==NULL) {
15         perror("1 - Error");
16         return errno;
17     }
18     buf=(char*)malloc(BUF_SIZE);
19     if (buf==NULL) {
20         perror("2 - Error");
21         pclose(f);
22         return errno;
23     }
24     while(fgets(buf,BUF_SIZE,f)!=NULL) {
25         //printf("arduino found at: %s, len %d\n",buf,strlen(buf));
26
27         for(int i=0;i<strlen(buf);i++){
28             if(buf[i]==' ' | buf[i]=='\n' ){
29                 //printf("find it\n");
30                 buf[i]=0;
31             }
32             else{
33                 //printf("%c\n-",buf[i]);
34             }
35         }
36         strcpy(addr,buf); //copia l'ultimo risultato
37     }
38     pclose(f);
39     free(buf);
40     return 0;
41 }
42
43 char* getArduino(){
44     search();
45     //printf("%s-+-+-",addr);
46     return addr ;
47 }
```

finder.o

```
1 #!/bin/bash
2 for sysdevpath in $(find /sys/bus/usb/devices/usb*/ -name dev); do
3     (
4         syspath="${sysdevpath%/dev}"
5         devname="$(udevadm info -q name -p $syspath)"
6         [[ "$devname" == "bus/*" ]] && continue
7         eval "$(udevadm info -q property --export -p $syspath)"
8         [[ -z "$ID_SERIAL" ]] && continue
9         echo "/dev/$devname - $ID_SERIAL - $ID_VENDOR - $ID_TYPE"
10    ) done
```

La libreria deviceFinder.h sfrutta lo script per la bash finder.o per ricavare la stringa desiderata. È importante notare che il programma finder.o deve avere le autorizzazioni per essere eseguito (chmod +x finder.o) e deve essere eseguito su un sistema Linux. Lo script finder.o elenca semplicemente path dei device connessi e relativi id caratteristici:

finder.o

```
1 echo "/dev/$devname - $ID_SERIAL - $ID_VENDOR - $ID_TYPE"
```

Il programma in C esegue quindi lo script e tramite piping nella shell filtra il risultato avente "Arduino" come stringa nel vendor id e prende il primo token della stringa:

deviceFinder.h

```
1 f=popen("./finder.o | grep Arduino | awk '{print $1;}'", "r");
```

La stringa viene dunque copiata e l'indirizzo isolato viene restituito dalla funzione getArduino(). La libreria può dunque essere usata con la classe SerialIO per cercare automaticamente la porta a cui è connesso il device Arduino e inizializzare la comunicazione seriale nel seguente modo:

```
1 #include "SerialIO.h"
2 #include "deviceFinder.h"
3 int main(int argc, char* argv[]){
4     SerialIO arbitrary_name(getArduino());
5     arbitrary_name.write("something");
6     arbitrary_name.read();
7 }
```


4 ZED

Lo ZED è una fotocamera che riproduce il modo in cui funziona la visione umana. Usando due sensori RGB da 4MP e attraverso la triangolazione, la ZED fornisce una comprensione tridimensionale della scena che osserva, permettendo al robot di diventare consapevole dello spazio e del movimento.

Cattura video 3D ad alta definizione, fino a 2K, con un ampio campo visivo (110°) e apertura $f/2.0$. La camera genera due flussi video destro e sinistro sincronizzati in formato side-by-side su USB 3.0. È disponibile la retrocompatibilità con USB 2.0.

Sono disponibili diverse modalità video:

Modalità video	Risoluzione di uscita (affiancata)	Frame Rate (fps)	Campo visivo
2.2K	4416x1242	15	Largo
1080p	3840x1080	30, 15	Largo
720p	2560x720	60, 30, 15	Extra largo
WVGA	1344x376	100, 60, 30, 15	Extra largo

Lo ZED emette le immagini in diversi formati. È possibile selezionare tra immagini rettificate, non rettificate e in scala di grigi.

Profondità Lo ZED riproduce il funzionamento della visione binoculare umana. Gli occhi umani sono separati orizzontalmente di circa 65 mm in media. Quindi, ogni occhio ha una visione leggermente diversa del mondo circostante. Confrontando queste due viste, il nostro cervello può inferire non solo la profondità ma anche il movimento 3D nello spazio. Allo stesso modo, lo ZED ha due occhi separati da 12 cm che catturano un video 3D ad alta risoluzione della scena e stimano profondità e movimento confrontando lo spostamento dei pixel tra le immagini sinistra e destra.

Mappa di profondità Lo ZED memorizza un valore di distanza (Z) per ciascun pixel (X, Y) nell'immagine. La distanza è espressa in metri e calcolata dal retro dell'occhio sinistro della telecamera all'oggetto della scena.

Le mappe di profondità catturate dallo ZED non possono essere visualizzate direttamente poiché sono codificate su 32 bit. Per visualizzare la mappa di profondità, è necessaria una rappresentazione monocromatica (in scala di grigi) a 8 bit con valori compresi tra [0, 255], dove 255 rappresenta il valore di profondità più vicino possibile e 0 il valore di profondità più distante possibile.

Mappatura 3D La ZED scansiona continuamente i suoi dintorni e crea una mappa 3D di ciò che vede. Questa mappa viene aggiornata quando il dispositivo viene spostato acquisendo nuovi elementi nella scena. Poiché la fotocamera percepisce le distanze oltre la gamma dei tradizionali sensori RGB-D, è in grado di ricostruire rapidamente mappe 3D di grandi aree interne ed esterne.

4.1 ZED SDK

Prima di tutto bisogna scaricare e installare la SDK relativa. Essa contiene tutte le librerie e strumenti che consentono di testare le caratteristiche e modificare le impostazioni della ZED. La Jetson-TX2 utilizza una versione diversa di linux, chiamata L4T(linux for tegra). Dal sito bisogna cercare nella pagina del download ZED SDK per la versione per TX2. Si noti che l'SDK ZED è collegato con la versione CUDA utilizzata da JetPack. È possibile verificare il corretto funzionamento della ZED utilizzando i tool presenti nell'SDK.

```
/usr/local/zed/tools
```

Si possono avviare lo ZED Explorer che permette un'anteprima e la registrazione. Consente di modificare la risoluzione video, le proporzioni, i parametri della fotocamera e acquisire istantanee ad alta risoluzione e video 3D.

```
/usr/local/zed/tools/ZED Explorer
```

Oppure lo ZED Depth Viewer utilizza l'SDK per acquisire e visualizzare la mappa di profondità e la nuvola di punti 3D.

```
/usr/local/zed/tools/ZED Depth Viewer
```

4.2 ZED-ROS WRAPPER

il pacchetto *zed-ros-wrapper* consente di utilizzare le telecamere stereo ZED con ROS. Fornisce un nodo ROS che pubblica in tempo reale su più topic immagini relative alla fotocamera destra e sinistra, nuvole di profondità relative, i dati relativi alle sue specifiche video, informazioni relative alla sua posizione e altro ancora. Per poterlo utilizzare devono essere soddisfatti alcuni prerequisiti:

- Ubuntu 16.04
- ZED SDK ≥ 2.3 e la sua dipendenza CUDA
- ROS Kinetic

Per installare *zed-ros-wrapper*, clonare il pacchetto da Github e installarlo:

```
cd ~/catkin_ws/src/ #use your current catkin folder
git clone https://github.com/stereolabs/zed-ros-wrapper.git
cd ..
catkin_make -DCMAKE_BUILD_TYPE=Release
echo source $(pwd)/devel/setup.bash >> ~/.bashrc
source ~/.bashrc
```

Per avviare il nodo *zed*, utilizzare *roslaunch*:

```
roslaunch zed_wrapper zed.launch
```

webcam interna Jetson-TX2 La Jetson-TX2 è fornita di una telecamera interna che avrà come file nella dev-path */dev/video0* mentre la telecamera ZED, se non ci sono collegate altre camere, sarà */dev/video1*. Nel pacchetto *zed-ros-wrapper* il launch file utilizzerà di default la camera che ha come indice 0. Occorre dunque modificare il *zed.launch* nel percorso *ROS/src/zed-ros-wrapper/zed_wrapper/launch* inserendo come argomento relativo alla camera il valore 1.

```
<arg name="camera_model"          default="1"/>
```

4.3 RVIZ

Per vedere se correttamente la ZED comunica con ROS si utilizza RVIZ. Si apre il terminale digitando:

```
roslaunch rviz rviz
```

Per vedere, per esempio, l'immagine rettificata dalla telecamera ZED, si inserisce il seguente comando in un nuovo terminale:

```
roslaunch image_view image_view image:= / camera / rgb / image_rect_color
```

In basso a sinistra del menu RVIZ, si clicca sul pulsante **Add**. Viene mostrato l'elenco di tutti i topic disponibili. Di default quelli della ZED si trovano */camera*

Per una documentazione più approfondita sulla ZED e i nodi utilizzati è consigliabile consultare [docs](#)

4.4 STAR-MARKER-FINDER

La stima della posizione è di grande importanza in molte applicazioni di visione artificiale: la navigazione robotica, la realtà aumentata e molte altre. Questo processo si basa sulla ricerca di corrispondenze tra i punti nell'ambiente reale e la loro proiezione di immagini 2D. Questo è solitamente un passaggio difficile, e quindi è comune l'uso di marcatori artificiali facilitare il compito.

Uno degli approcci più popolari è l'uso di marker binari quadratici. Il vantaggio principale di questi marcatori è che un singolo marker fornisce abbastanza corrispondenze (i suoi quattro angoli) per ottenere la posizione della telecamera. Inoltre, la codifica binaria interna li rende particolarmente robusti, permettendo l'applicazione di tecniche di rilevamento e correzione degli errori.

Un marker *ArUco* è un marker quadrato sintetico composto da un ampio bordo nero e una matrice binaria interna che determina il suo identificatore (id). Il bordo nero facilita il rapido rilevamento nell'immagine e la codifica binaria ne consente l'identificazione e l'applicazione delle tecniche di rilevamento e correzione degli errori. La dimensione del marker determina la dimensione della matrice interna. Ad esempio un marker di dimensione 4x4 è composta da 16 bit.

Ros contiene nelle sue librerie un dizionario di marcatori. Un dizionario è un insieme di marcatori considerati per una specifica applicazione. È formato da una lista di codifiche binarie di ciascun marcatore

Le proprietà principali di un dizionario sono la dimensione del dizionario (il numero di marcatori) e la dimensione del marcatore (la dimensione effettiva del marcatore).

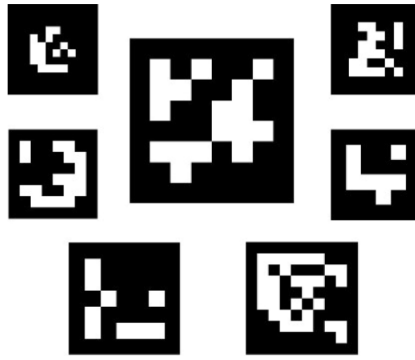


Figura 3: Esempio di immagini dei marcatori

Reindirizzamento topic Precedentemente lo *Star-marker-finder* utilizzava una camera usb per localizzare i marcatori e dedurre la posizione del robot. Adesso al posto del nodo della camera usb viene utilizzato il nodo della ZED. È necessario modificare i file lunch dello star-marker-finder, si trova nel percorso

```
\ROS\src\star_marker_finder\launch
```

E cambiare le voci di *remap*, è sufficiente scegliere una delle due camera sinistra o destra.

```
<remap from="image_rect" to="/left/image_rect_color"/>
<remap from="camera_info" to="/left/camera_info"/>
```

4.5 Marker-finder su architettura distribuita

Il programma è stato realizzato con l'obiettivo di rendere autonoma la guida di un robot esistente. Quest'ultimo è dotato di un computer di bordo responsabile del movimento delle quattro ruote omnidirezionali connesse attraverso un cavo RJ45. Si è voluto implementare la comunicazione tra questo pc e la scheda Jetson in modo da separare la parte di gestione della locomozione da quella di odometria e elaborazione dati. La connessione tra i due calcolatori viene effettuata attraverso un ulteriore cavo rj45 in configurazione crossover, creando una rete LAN. È necessario assegnare un ip statico alle due interfacce di rete, configurabile tramite il network-manager di ubuntu o con un semplice comando da terminale:

```
ifconfig eth0 192.168.0.2 netmask 255.255.255.0 up
```

In questo caso si presuppone che la porta ethernet sia riconosciuta dal sistema come "eth0" mentre l'indirizzo ip è un indirizzo ip scelto casualmente. Per la scelta dell'indirizzo ip del secondo pc si tenga conto della convenzione adottata nel primo. In questo caso, 24 bit di subnet e 8 di indirizzamento, l'ip della seconda scheda dovrà quindi essere del tipo *192.168.0.x*. Una volta stabilita la connessione, confermata generalmente da una notifica desktop di sistema, bisogna effettuare il setup dei terminali utilizzati per ROS.

In un sistema distribuito ROSCORE viene eseguito su una sola macchina, alle altre macchine connesse deve essere fornito l'ip della macchina che sta eseguendo roscore in modo da poter connettere i propri nodi ad esso.

facendo riferimento alla Figura 4 e supponendo di voler avviare ROSCORE sulla scheda Jetson, sulla stessa dovremmo esportare indirizzo ip e relativa porta su cui verrà avviato ROSCORE e un eventuale nome della macchina:

```
echo export ROS_MASTER_URI=http://localhost:11311
echo export ROS_HOSTNAME=JETSONTX2
roscore
```

È stato così avviato ROSCORE ad un indirizzo specifico a cui qualsiasi pc nella rete locale potrà connettersi per coordinare la gestione dei topic.

N.B. i comandi per la gestione degli indirizzi e del nome dell'host devono essere ripetuti ogni qualvolta un nuovo terminale viene aperto. Per convenienza questi possono essere aggiunti in fondo al file '.bashrc', collocato solitamente nella home, per essere eseguiti all'avvio di ciascuna BASH. Se si utilizza ZSH o FISH aggiungerli ai relativi file di configurazione. Per ogni altro pc che si vuole connettere alla scheda Jetson andrà specificato il suo indirizzo ip allo stesso modo:

```
echo export ROS_MASTER_URI=http://JETSONTX2:11311
```

Alternativamente si può utilizzare l'indirizzo ip della scheda nel caso ci fossero problemi con il riconoscimento dell'hostname:

```
echo export ROS_MASTER_URI=http://192.168.0.3:11311
```

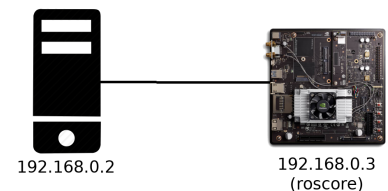


Figura 4: ipotetica configurazione

Si possono ora utilizzare tutte le funzionalità di ROS e avviare programmi come se tutti i nodi avviati dalle diverse macchine fossero presenti su ciascuna di esse. Nel caso del marker finder abbiamo avviato roscore sulla Jetson con connessa la telecamera ZED e fatto il setup sul secondo PC. Abbiamo quindi avviato il nodo marker finder sulla Jetson e visualizzato i risultati sul secondo pc attraverso RVIZ simulando in questo modo uno scambio dati attraverso due topic tra i due pc.

5 eProsima DDS

Il Data Distribution Service for Real Time Systems (DDS) è uno standard emanato dall'Object Management Group (OMG) che definisce un middleware per la distribuzione di dati in tempo reale secondo il paradigma publish/subscribe. Il Data Distribution Service è concepito come una soluzione infrastrutturale alla programmazione di applicazioni incentrate sui dati. Il suo scopo è di nascondere interamente le problematiche della gestione della comunicazione nella programmazione di applicazioni data-centric.

5.1 ROS 2

L'obiettivo è di rendere DDS un dettaglio di implementazione di ROS 2.0. Ciò significa che tutte le API specifiche di DDS e le definizioni dei messaggi devono essere nascoste. DDS offre funzionalità di rilevamento, definizione dei messaggi, serializzazione dei messaggi e trasporto di tipo publish-subscriber. ROS 2.0 fornirà un'interfaccia simile a ROS 1.x ma al livello di comunicazione utilizzerà DDS. In ROS 2 si potranno utilizzare diversi sistemi DDS per specifiche API.

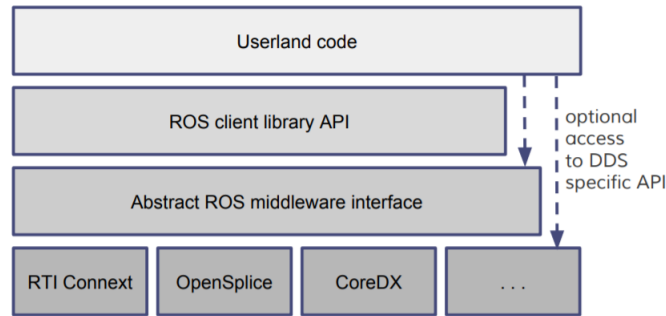


Figura 5: Layout API DDS e ROS

L'obiettivo dell'API ROS, che è in cima al DDS, dovrebbe essere quello di soddisfare tutte le esigenze comuni della comunità ROS, perché una volta che un utente accede al sistema DDS sottostante, perderà la portabilità tra i fornitori DDS. La portabilità tra i fornitori DDS non è intesa a incoraggiare le persone a scegliere frequentemente fornitori diversi, ma piuttosto a consentire agli utenti esperti di selezionare l'implementazione DDS che soddisfa i loro requisiti specifici.

DDS sostituirà completamente il sistema di rilevamento basato sul master ROS. ROS dovrà utilizzare le API di DDS per ottenere le informazioni sui nodi, topics e il modo in cui sono connessi. L'accesso viene mascherato dietro una API definita da ROS, impedendo agli utenti di dover chiamare direttamente DDS.

5.2 eProsima

Il protocollo DDSI-RTPS (DDS-Interoperability Real Time Publish Subscribe) sostituirà i protocolli ROS1 TCPROS e UDPROS per i protocolli publish/subscribe. *Prosima Fast RTPS* è un'implementazione nuova, leggera e open source di RTPS. Consente l'accesso diretto alle impostazioni e alle caratteristiche del protocollo RTPS. Nella parte superiore di RTPS, troviamo il dominio, che definisce un piano separato di comunicazione. Diversi domini possono coesistere contemporaneamente in modo indipendente. Un dominio contiene un numero qualsiasi di partecipanti, elementi in grado di inviare e ricevere dati. Per fare ciò, i partecipanti usano i loro endpoint:

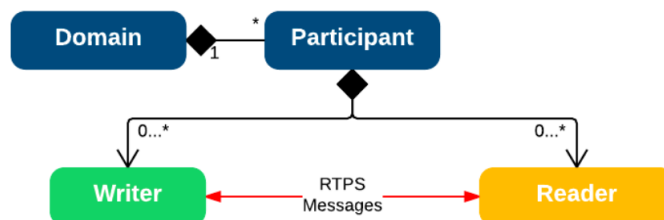


Figura 6: Layout API DDS e ROS

- Reader: Endpoint in grado di ricevere dati.
- Writer: Endpoint in grado di inviare dati.

Un partecipante può avere un numero qualsiasi Reader e Writer endpoint. La comunicazione ruota intorno ai Topics, che definiscono i dati scambiati. I Topics non appartengono ad alcun partecipante in particolare; invece, tutti i partecipanti interessati tengono traccia delle modifiche ai dati dell'argomento e si assicurano di tenersi aggiornati reciprocamente.

5.3 installazione

eProsima ha bisogno di due componenti:

- eProsima RPC su DDS
- eProsima Fast RTPS

Disponibili sul sito eProsima.

eProsima RPC su DDS eProsima RPC su DDS richiede l'installazione della libreria *eProsima FastCDR*. La libreria eProsima FastCDR si trova sotto la cartella *"requiredcomponents"*. Bisogna estrarre il contenuto del pacchetto *"eProsima-FastCDR-1.0.6-Linux.tar.gz"* ed eseguire sul terminale:

For 32-bit machines

```
cd eProsima_FastCDR-1.0.6-Linux;
./configure --libdir=/usr/lib;
make;
make install
```

For 64-bit machines

```
cd eProsima_FastCDR-1.0.6-Linux;
./configure --libdir=/usr/lib64;
make;
make install
```

richiede anche l'installazione della libreria *eProsima FastRTPS* su il tuo sistema. La libreria *eProsima FastRTPS* viene fornita nella cartella *"requiredcomponents"*. Si estrae il contenuto del pacchetto *"eProsima-FastRTPS-1.3.1-Linux.tar.gz"* e si esegue i seguenti comandi:

For 32-bit machines

```
cd eProsima_FastRTPS-1.3.1-Linux;
./configure --libdir=/usr/lib;
make;
make install
```

For 64-bit machines

```
cd eProsima_FastRTPS-1.3.1-Linux;
./configure --libdir=/usr/lib64;
make;
make install
```

eProsima RPC su DDS richiede anche librerie *Boost*. Se non sono presenti sulla macchina dovranno essere installati utilizzando il Gestore dei pacchetti di Linux. Fatto questo, si procede con l'installazione eProsima RPC su DDS.

For 32-bit machines

```
cd eProsima_RPCDDS-1.0.3-Linux;
./configure --libdir=/usr/lib;
make;
make install
```

For 64-bit machines

```
cd eProsima_RPCDDS-1.0.3-Linux;
./configure --libdir=/usr/lib64;
make;
make install
```

eProsima Fast RTPS Successivamente eProsima Fast RTPS richiede l'installazione della libreria *eProsima FastCDR* per eseguire alcuni degli esempi forniti e compilare il codice generato con *eProsima FASTRTPSGEN*. La libreria eProsima FastCDR è disponibile nella cartella *"requiredcomponents"*. Estrarre il contenuto del pacchetto *"eProsima-FastCDR-1.0.7-Linux.tar.gz"* ed eseguire:

```
cd eProsima_FastCDR-1.0.7-Linux;  
./configure --libdir=/usr/lib;  
make;  
sudo make install
```

Installare il software eProsima Fast RTPS.

```
cd eProsima_FastRTPS-1.5.0-Linux;  
./configure --libdir=/usr/lib;  
make;  
sudo make install
```