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

Anno Accademico 2018–2019

Indice

1 ROS per NVIDIA Jetson TX2 **2**
1.1 Setup tramite tool Jetpack . . . 2
1.2 Backup e ripristino . . . 2
1.2.1 Backup . . . 2
1.2.2 Ripristino . . . 2
1.3 Possibile errore . . . 3
1.4 Setup di Ubuntu . . . 3
1.4.1 Setup tastiera . . . 3
1.4.2 Installazione ros . . . 3

2 ROS: COMUNICAZIONE SERIALE TRAMITE USB **4**

3 ZED **8**
3.1 ZED SDK . . . 8
3.2 ZED-ROS WRAPPER . . . 8
3.3 STAR-MARKER-FINDER . . . 9

4 eProsima DDS **11**
4.1 ROS 2 . . . 11
4.2 eProsima . . . 11
4.3 installazione . . . 12

1 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.

1.1 Setup tramite tool Jetpack

La scheda é dotata di pulsanti per il controllo diretto, per l'accensione della scheda é necessario 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 collegare la Scheda Jetson a internet attraverso un cavo ethernet connesso allo stesso router del pc su cui eseguiremo Jetpack mantenendo la scheda spenta. Sulla scheda sono presenti 4 tasti rossi evidenziati da un riquadro rosso nella 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. tendo 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 estremitá ad una porta usb sul pc su cui si eseguirá jetpack.

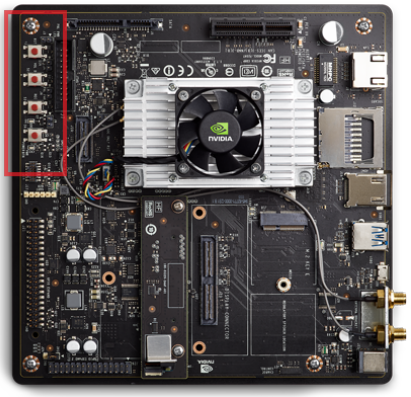

Figura 1: Jetson TX2.

Una volta scaricato l'archivio dal sito ufficiale e 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 presenterá 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 utilizare 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 presenterá 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.

1.2 Backup e ripristino

Una volta eseguito jetpack e scaricate le dipendenze verranno creati in specifiche cartelle script per eseguire operazioni manualmente sulla jetson quali backup e ripristino ad esempio.
Per fare ciò navighiamo nella cartella contenente L4T (nel mio calcolatore la directory era '64_TX2/Linux_for_Tegra') dove sarà presente il file 'flash.sh'.
Apriamo dunque un terminale in questa posizione e colleghiamo la scheda al pc in modalità recovery.

1.2.1 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' entrabi da conservare.

1.2.2 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.)

1.3 Possibile errore

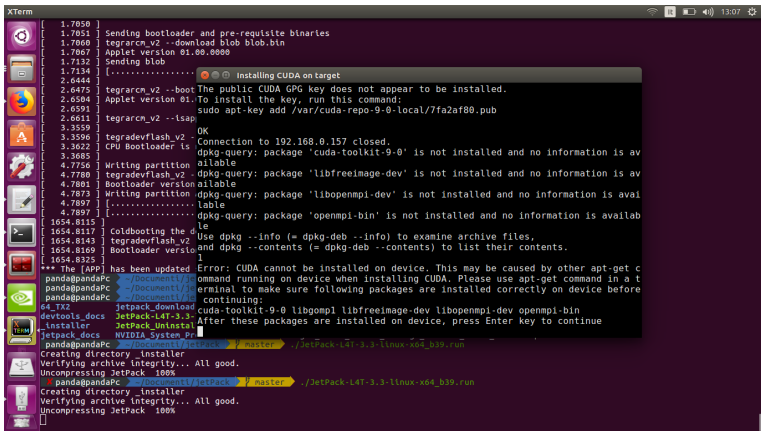

Figura 2: errore durante il flash.

Nella mia personale esperienza si è verificato un errore durante il flash della scheda dovuto all'interruzione della connessione internet, probabilmente causato da filtri sulla rete universitaria o a una momentanea instabilitá della connessione. Per risolvere il problema ho collegato la tastiera alla scheda jetson,ho installato manualmente i pacchetti mancanti tramite apt-get e infine premuto enter come indicato da jetpack nel terminale sul mio pc personale (n.b. eseguire "sudo apt-get update" sulla jetson prima di qualsiasi apt-get install per aggiornare le repositories)

1.4 Setup di Ubuntu

1.4.1 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

1.4.2 Installazione ros

Seguire le indicazioni per l'istallazione di ros-kinetic dal sito ufficiale per ubuntu ARM.

2 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 e 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 a una velocitá predefinita e con dati strutturati in modo ben preciso:

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

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

```
#include <unistd.h>
#include <iostream>
#include <boost/asio.hpp>
#include <boost/asio/serial_port.hpp>
#include <boost/system/error_code.hpp>
#include <boost/system/system_error.hpp>
#include <boost/bind.hpp>
#include <boost/thread.hpp>
#include <boost/date_time/posix_time/posix_time.hpp>
#include <boost/thread/thread.hpp>
#include <thread>

class SerialIO{
private:
        boost::asio::io_service io_svc;
        boost::asio::serial_port m_port;
public:
SerialIO(std::string portDir):m_port(io_svc,portDir){}

void write(std::string input){
        const int BUFFSIZE=20;
        boost::array<char,BUFFSIZE> buf;
        if(sizeof(input)>0){
                for(int i=0;i<sizeof(input)/8;i++){
                        buf[i]=input[i];
                }
                buf[sizeof(input)/8]=0;
                m_port.write_some(boost::asio::buffer(buf,BUFFER);
        }
}

void read(){
        const int BUFFSIZE=10;
        char read_msg_[BUFFSIZE];
        std::stringstream ss;
        int read_size=0;
        read_size=m_port.read_some(boost::asio::buffer(read_msg_,BUFFSIZE));
        while(read_msg_[read_size-1]!='\n'){//ciclo fino al finestringa
        read_msg_[read_size]=0;//terminatore stringa inserito manualmente
        ss<<read_msg_;
        read_size=m_port.read_some(boost::asio::buffer(read_msg_,BUFFSIZE));
        }
        read_msg_[read_size]=0;
        ss<<read_msg_;
        std::cout << ss.str();
}};
```

Le funzioni di read e write nella comunicazione asincrona stabilita sono bloccanti pertanto la coordinazione dei messaggi deve essere fatta a monte e non viene 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 puo 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_,BUFFSIZE));
```

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 BUFFSIZE, 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 puo essere modificata per eventualmente effettuare un post in un topic ros.
La funzione write è piuttosto simile alla read:

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

Viene creato un buffer di dimensione massima 20 (buffer troppo grossi rallentano notevolmente il programma e causano blocchi,ho trovato 20 un numero adeguato ma il valore puo 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 piu 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 piu facile inizializzazione della comunicazione seriale è stato scritta una libreria sempre in C++ per trovare un Arduino e ritornare il path del suo riferimento necessario per l'inizializzazione.

deviceFinder.h

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <malloc.h>
#include <iostream>
#include <string.h>
#define BUF_SIZE 1024
char addr[30]="";
int search(void)
{
    FILE *f;
    char* buf;
    f=popen("./finder.o | grep Arduino | awk '{print $1;}'", "r");
    if (f==NULL) {
        perror("1 - Error");
        return errno;
    }
   buf=(char*)malloc(BUF_SIZE);
   if (buf==NULL) {
        perror("2 - Error");
        pclose(f);
        return errno;
    }
        while(fgets(buf,BUF_SIZE,f)!=NULL) {
                //printf("arduino found at: %s, len %d\n",buf,strlen(buf));

                for(int i=0;i<strlen(buf);i++){
                        if(buf[i]==' '| buf[i]=='\n' ){
                                //printf("find it\n");
                                buf[i]=0;
                        }
                        else{
                                //printf("%c\n-",buf[i]);
                        }
                }
                strcpy(addr,buf); //copia l'ultimo risultato
    }
    pclose(f);
    free(buf);
    return 0;
}

char* getArduino(){
        search();
        //printf("%s-+-+-",addr);
        return addr     ;
}
```

finder.o

```
#!/bin/bash
for sysdevpath in $(find /sys/bus/usb/devices/usb*/ -name dev); do
    (   syspath="${sysdevpath%/dev}"
        devname="$(udevadm info -q name -p $syspath)"
        [[ "$devname" == "bus/"* ]] && continue
        eval "$(udevadm info -q property --export -p $syspath)"
        [[ -z "$ID_SERIAL" ]] && continue
        echo "/dev/$devname - $ID_SERIAL - $ID_VENDOR - $ID_TYPE"
    ) 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 al 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 }
```

3 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. É 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. É possibile selezionare tra immagini rettificate, non rettificate e in scala di grigi.

Profodità 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.

3.1 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 utizza un 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. É possibile verificare il corretto funzionamento della ZED utlizzando 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
```

3.2 ZED-ROS WRAPPER

il pacchetto *zed-ros-wrapper* consente di utilizzare le telecamere stereo ZED con ROS. Fornisce le immagini della fotocamera sinistra e destra, la mappa della profondità, la nuvola di punti, le informazioni sulla posa e supporta l'uso di più telecamere ZED.

Per poterlo utilizzare devono essere soddisfati alcuni prerequisiti

- Ubuntu 16.04

- ZED SDK $\geq$ 2.3 e la sua dipendenza CUDA
- ROS Kinetic

Per installare zed-ros-wrapper , aprire un terminale bash, clonare il pacchetto da Github e crearlo:

```
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
```

Aprire un terminale e utilizzare *roslaunch* per avviare il nodo ZED:

```
roslaunch zed_wrapper zed.launch
```

webcam interna jetson-TX2 La jetson-TX2 è fornita di una telecamera interna che avrà come indice */dev/video0* mentre la telecamera ZED, se non ci sono collegate altre camere, sarà */dev/video1*. Il pacchetto *zed-ros-wrapper* avrà un launch file che chiamerà la camera di defaut come indice 0. Modificare il *zed.launch* dentro il percorso `ROS/src/zed-ros-wrapper/zed_wrapper/launch` inserendo nel argomento relativa alla camera il valore 1.

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

Per vedere se correttamente la ZED comunica con ROS avviare RVIZ.
RVIZ è un'interfaccia grafica di ROS che consente di visualizzare molte informazioni, utilizzando plugin per molti tipi di topic disponibili.

Per una documentazione più approfondita sulla ZED e i nodi utilizzati è consigliabile consultare docs

3.3 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 per renderlo più facile.

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, consentendo la possibilità di applicare 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 una dimensione del marker di 4x4 è composta da 16 bit.

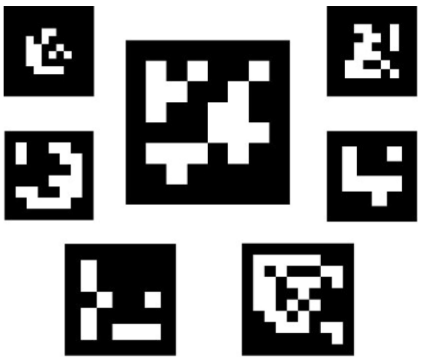

Figura 3: Esempio di immagini dei marcatori

Va notato che un marcatore può essere trovato ruotato nell'ambiente, tuttavia, il processo di rilevamento deve essere in grado di determinare la sua rotazione originale, in modo che ogni angolo sia identificato inequivocabilmente. Questo viene fatto anche in base alla codifica binaria.

Un dizionario di marcatori è un insieme di marcatori considerati in un'applicazione specifica. È semplicemente la lista delle codifiche binarie di ciascuno dei suoi marcatori.

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

Reindirizzamento topic Precedentemente lo *Star-marker-finder* utilizzava una camera usb per localizzare i marcatori e dedurre la propria posizione. Adesso al posto del nodo della camera usb viene utilizzato il nodo della ZED. Bisogna 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 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.

4.1 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 pubblicazione-sottoscrizione. Pertanto, DDS fornirebbe la scoperta, il trasporto di pubblicazione-sottoscrizione e almeno la serializzazione del messaggio sottostante per ROS. ROS 2.0 fornirà un'interfaccia ROS 1.x simile a DDS che nasconde gran parte della complessità di DDS per la maggior parte degli utenti di ROS, ma fornisce separatamente l'implementazione DDS sottostante per gli utenti che hanno casi di utilizzo estremi o necessità di integrarsi con altri sistemi DDS esistenti.

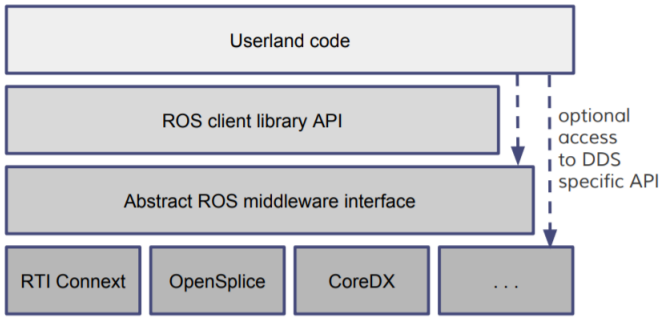

Figura 4: Layout API DDS e ROS

L'accesso all'implementazione DDS richiederebbe a seconda di un pacchetto aggiuntivo che non viene normalmente utilizzato. In questo modo puoi capire se un pacchetto si è legato a un particolare fornitore DDS semplicemente osservando le dipendenze del pacchetto. 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, nonché a ROS a prova di futuro contro le modifiche nelle opzioni del fornitore DDS.

DDS sostituirà completamente il sistema di rilevamento basato sul master ROS. ROS dovrebbe attingere all'API di DDS per ottenere informazioni come un elenco di tutti i nodi, un elenco di tutti gli topics e il modo in cui sono connessi. L'accesso a queste informazioni sarebbe nascosto dietro una API definita ROS, impedendo agli utenti di dover chiamare direttamente in DDS.

4.2 eProsima

Il protocollo DDSI-RTPS (DDS-Interoperability Real Time Publish Subscribe) sostituirà i protocolli ROSRO TCPROS e UDPROS per i protocolli publish/subscribe. *Prosima Fast RTPS* è un'implementazione relativamente nuova, leggera e open source di RTPS. Consente l'accesso diretto alle impostazioni e alle caratteristiche del protocollo RTPS, il che non è sempre possibile con altre implementazioni DDS. 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:

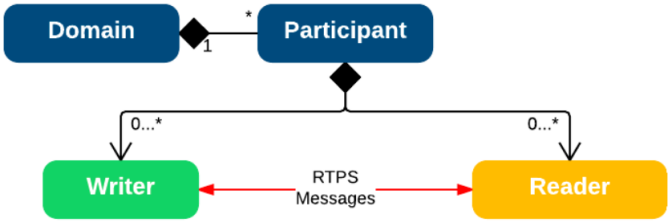

Figura 5: Layout API DDS e ROS

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

Un partecipante può avere un numero qualsiasi di scrittori e lettori 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

4.3 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 richede 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"*. Estrarre il contenuto del pacchetto *"eProsima-FastRTPS-1.3.1-Linux.tar.gz"* ed eseguire:

```
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. 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 richede 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"*. Estrare 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
```