

Nozioni utili

Talker.ccp

```
#include "ros/ros.h"
```

Include tutte le intestazioni necessarie per utilizzare le parti pubbliche più comuni del sistema ROS.

```
#include "std_msgs/String.h"
```

```
#include <sstream>
```

```
int main(int argc, char **argv)
```

```
{
```

```
    ros::init(argc, argv, "talker");
```

Inizializza ROS. Assegna a questo nodo il nome "talker"

```
    ros::NodeHandle n;
```

Inizializza il nodo.

```
    ros::Publisher chatter_pub = n.advertise<std_msgs::String>("chatter", 1000);
```

La call "advertise" avverte il nodo master che siamo intenzionati a pubblicare un messaggio di tipo std_msgs/String sul topic Chatter. Il secondo parametro indica la dimensione della coda di pubblicazione. Se mandiamo messaggi troppo veloci, appena si riempie il buffer vengono cancellati i messaggi vecchi.

Il return di questa funzione finisce nell'oggetto "chatter_pub" in quale serve per:

- Permette di pubblicare messaggi sul topic.
- Si auto discrive dal topic quando termina.

```
    ros::Rate loop_rate(10);
```

Specifica la frequenza di pubblicazione del messaggio sul topic. In modo tale che lo "sleep" duri un certo tempo fissato. In questo caso la velocità di loop è di 10Hz.

```
    int count = 0;
```

Tengo traccia di quanti messaggi sto pubblicando per fare in modo che ognuno sia univoco.

```
    while (ros::ok())
```

```
    {
```

Ros::ok() ritorna false e quindi esce dal ciclo quando

- Ctrl-C
- Veniamo espulsi dalla rete di nodi da un nodo con il nostro stesso nome
- Tutti i NodeHandles vengono distrutti
- Viene invocato `ros::shutdown()`

```
        std_msgs::String msg;
```

```
        std::stringstream ss;
```

```
        ss << "hello world " << count;
```

```
        msg.data = ss.str();
```

Costruisco la stringa da pubblicare sul topic

```
ROS_INFO("%s", msg.data.c_str());
```

E' una printf di ros.

```
chatter_pub.publish(msg);
```

Pubblico il messaggio costruito prima sul topic.

```
ros::spinOnce();
```

Serve per chiamare le callback in caso il nodo creasse un topic sui quali gli altri vanno a pubblicare.

```
loop_rate.sleep();
```

Invoco la sleep per avere la frequenza di publish di 10Hz.

```
++count;
```

```
}
```

```
return 0;
```

```
}
```

Listener.ccp

```
#include "ros/ros.h"
```

```
#include "std_msgs/String.h"
```

```
void chatterCallback(const std_msgs::String::ConstPtr& msg)
```

```
{
```

```
    ROS_INFO("I heard: [%s]", msg->data.c_str());
```

```
}
```

Questa funzione viene invocata ogni qualvolta arriva un nuovo messaggio viene pubblicato sul topic "Chatter".

```
int main(int argc, char **argv)
```

```
{
```

```
    ros::init(argc, argv, "listener");
```

```
    ros::NodeHandle n;
```

```
    ros::Subscriber sub = n.subscribe("chatter", 1000, chatterCallback);
```

Il nodo avvisa il master che vuole sottoscrivere un nuovo topic di nome "Chatter" e che venga invocata la funzione chatterCallback ogni volta un nuovo messaggio viene pubblicato.

Chi si sottoscrive ad un topic vuole ricevere delle notifiche di messaggi pubblicati sopra.

```
    ros::spin();
```

Qua non c'è un ciclo come nel talker. Perché "spin" non ha return, il nodo si ferma in spin e continuano ad essere processati i messaggi pubblicati sul topic.

SpinOnce invece ritorna poi all'esecuzione del nodo, se voglio pubblicare altri messaggi devo inserirlo all'interno di un loop, stabilendo anche una frequenza di pubblicazione.

```
    return 0;
```

```
}
```

Logbook

0- Installazione full desktop di ROS Melodic su Ubuntu 18.04 LTS

1- Installazione di DVRK con qualche warning

2- Esecuzione di rviz con modello del PSM, ma con errore di un nodo

3- Occhiata al funzionamento dei nodi listener e talker.

4- Errori di file non trovati. Da internet nel "sawIntuitiveResearchKit-master" ho preso i file che mi mancavano.

Console PSM1 KIN SIMULATED.json

```
{
  "arms":
  [
    {
      "name": "PSM1",
      "type": "PSM",
      "simulation": "KINEMATIC",
      "pid": "sawControllersPID-PSM.xml",
      "kinematic": "psm-large-needle-driver.json"
    }
  ]
}
```

Mancano i due file citati in pid e kinematics.

5- Nuovo errore:

----- ERROR:

You should have a "arm" file for each arm in the console file. The arm file should contain the fields "kinematic" and options specific to each arm type.

E- Class mtsIntuitiveResearchKitPSM: File: mtsIntuitiveResearchKitArm.cpp Line: 599 - Configure PSM1:

----- ERROR:

You should have a "arm" file for each arm in the console file. The arm file should contain the fields "kinematic" and options specific to each arm type.

mtsIntuitiveResearchKitArm.cpp riga 599

jsonKinematic = jsonConfig["kinematic"];
ma in jsonConfig che è: /share/psm-large-needle-driver.json
non ci sono camp "kinematic" quindi ritorna errore

ALTERNATIVA1:

PSM1

```
roslaunch dvrk_robot dvrk_arm_rviz.launch arm:=PSM1 \
config:=/home/filippo/catkin_ws/src/cisst-saw/sawIntuitiveResearchKit/share/console/console-
PSM1_KIN_SIMULATED.json
```

Mettendo un /console davanti funziona però la cinematica che va a prendere è /kinematic/psm.json

anzichè large_needle_driver.json

La cinematica di psm.json comprendere sono 3 giunti. I primi 3.

Oppure posso sempre lanciarlo senza specificare il config file. Perchè senza specificare il config file lui prende direttamente quello in "CONSOLE/CONSOLE-...." PERCHÈ C'È SCRITTO NEL *.LAUNCH FILE.

ALTERNATIVA2:

Usare "kinematic": "psm-large-needle-driver.json",
nel file PSM_KIN_SIMULATED_LARGE_NEEDLE_DRIVER_400006.js
problema è che tool e tool-detection sono diversi. Quindi non so se sia la stessa cosa.

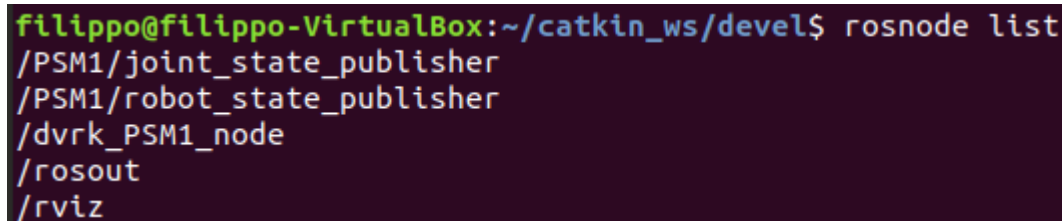
A questo scopo ho creato dei miei file:

```
roslaunch dvrk_robot dvrk_arm_rviz.launch arm:=PSM1 \
config:=/home/filippo/catkin_ws/src/cisst-saw/sawIntuitiveResearchKit/share/myconsole.json
```

Il file /arm/myarmfile.json è uguale a PSM_KIN_SIMULATED_LARGE_NEEDLE_DRIVER_400006.js
Ma senza le ultime due righe.

Seguendo l'alternativa 2, non ci sono errori durante il roslaunch.

Con il comando "**rostopic list**" vedo i nodi in esecuzione:



```
filippo@filippo-VirtualBox:~/catkin_ws/devel$ rostopic list
/PSM1/joint_state_publisher
/PSM1/robot_state_publisher
/dvrk_PSM1_node
/rosout
/rviz
```

Dopodichè vedo quali topic ci sono con il comando "**rostopic list -v**" ma per il momento non vedo topic che hanno a che fare con la cinematica.

Con il comando "**rostopic info nome_nodo**" è possibile vedere tutte le info per i singoli nodi.

La tabella DH si trova in /share/psm_large_needle_driver.json. Ma ci sono delle informazioni/limiti di giunto anche nel file /share/sawControllersPID-PSM.xml

DUBBIO:

IL FILE /SHARE/CONSOLE_MTML_KIN_SIMULATED.JSON:

```
/* -*- Mode: Javascript; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- */
{
  "arms":
  [
    {
      "name": "MTML",
      "type": "MTM",
      "simulation": "KINEMATIC",
      "pid": "sawControllersPID-MTML.xml",
      "kinematic": "mtm.json"
    }
  ]
}
```

IL FILE /SHARE/CONSOLE_PSM_KIN_SIMULATED.JSON:

```
/* -*- Mode: Javascript; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- */
{
  "arms":
  [
    {
      "name": "PSM1",
      "type": "PSM",
      "simulation": "KINEMATIC",
      "pid": "sawControllersPID-PSM.xml",
      "kinematic": "psm-large-needle-driver.json"
    }
  ]
}
```

Capire i file di configurazione:

<https://github.com/jhu-dvrk/sawIntuitiveResearchKit/wiki/Configuration-File-Formats#psms>

Using the ROS topics

The best way to figure how to use the ROS topics is to look at the files `dvrk_python/src/robot.py` and `dvrk_matlab/robot.m`.

Guida per python:

Start a single arm using

> rosrun dvrk_robot dvrk_console_json -j <console-file>

To communicate with the arm using ROS topics, see the python based example
dvrk_arm_test.py:

> rosrun dvrk_python dvrk_arm_test.py <arm-name>

1. #Solito errore con questa console

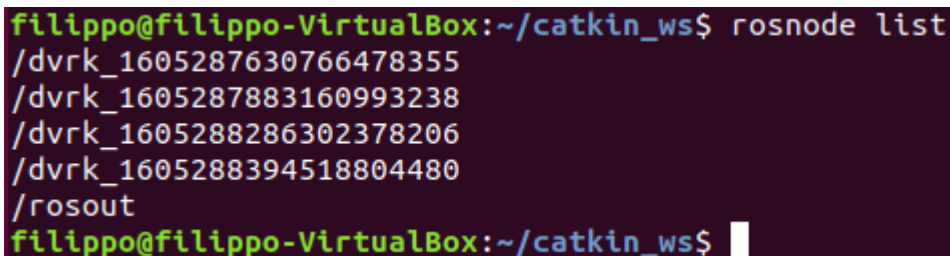
```
roslaunch dvrk_robot dvrk_console_json -j /home/filippo/catkin_ws/src/cisst-  
saw/sawIntuitiveResearchKit/share/console-PSM1_KIN_SIMULATED.json
```

2. #Funziona ma usa la cinematica con 3 giunti

```
roslaunch dvrk_robot dvrk_console_json -j /home/filippo/catkin_ws/src/cisst-  
saw/sawIntuitiveResearchKit/share/console/console-PSM1_KIN_SIMULATED.json
```

Apri una console e funziona.

Con il comando "roslaunch list":



```
filippo@filippo-VirtualBox:~/catkin_ws$ roslaunch list  
/dvrk_1605287630766478355  
/dvrk_1605287883160993238  
/dvrk_1605288286302378206  
/dvrk_1605288394518804480  
/rosout  
filippo@filippo-VirtualBox:~/catkin_ws$
```

Ora come suggerisce la guida, eseguo lo script python: **roslaunch dvrk_python dvrk_arm_test.py PSM1**, con errore:

```

filippo@filippo-VirtualBox:~/catkin_ws$ rosrun dvrk_python dvrk_arm_test.py PSM1
Traceback (most recent call last):
  File "/home/filippo/catkin_ws/src/dvrk-ros/dvrk_python/scripts/dvrk_arm_test.py", line 22, in <module>
    import dvrk
  File "/home/filippo/catkin_ws/devel/lib/python2.7/dist-packages/dvrk/__init__.py", line 34, in <module>
    exec(__fh.read())
  File "<string>", line 17, in <module>
  File "/home/filippo/catkin_ws/src/dvrk-ros/dvrk_python/src/dvrk/arm.py", line 29, in <module>
    import crtk
  File "/home/filippo/catkin_ws/devel/lib/python2.7/dist-packages/crtk/__init__.py", line 34, in <module>
    exec(__fh.read())
  File "<string>", line 20, in <module>
ImportError: No module named arm
filippo@filippo-VirtualBox:~/catkin_ws$

```

Nel file arm.py:

```

"""

```

This class presents a arm api for the da Vinci Research Kit. Remember that for this program to work, you will need to import the arm class, this can be done by ``from dvrk.arm import arm`` as well as initialize the arm. For example, if we want to create a arm called ``r``, for arm ``PSM1``, we will simply type ``r = arm('PSM1')``.

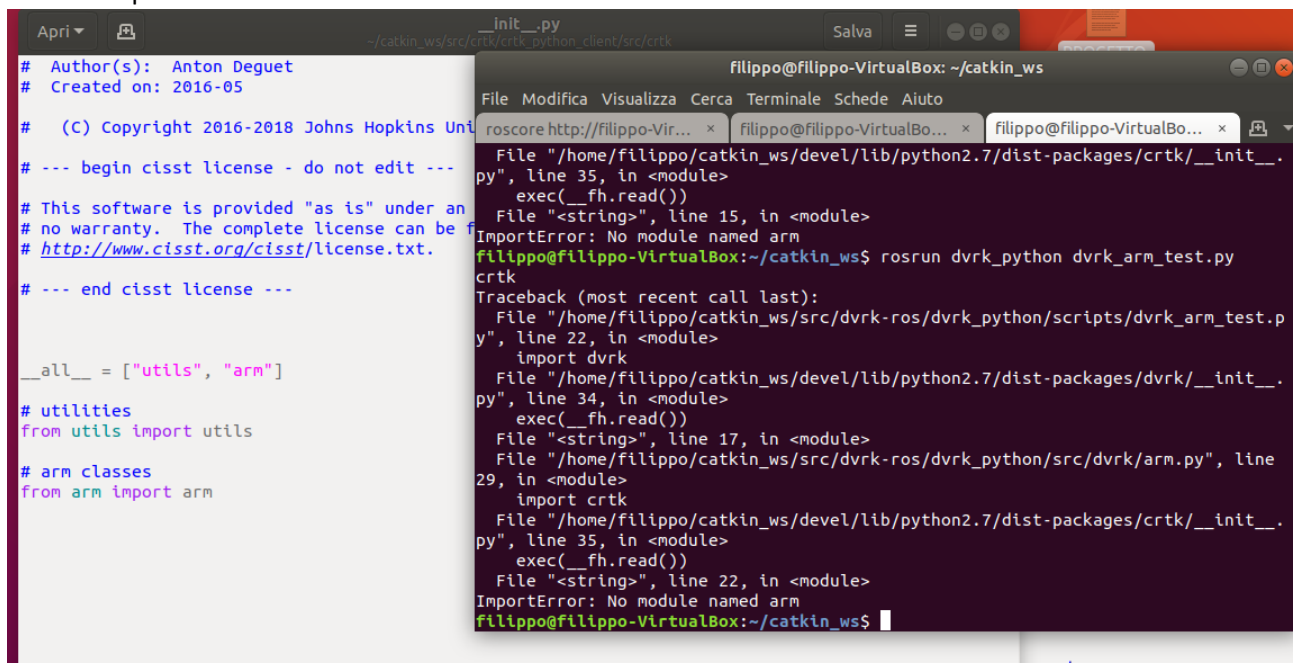
For arm specific features, import the class psm or mtm (e.g. ``from dvrk.psm import psm``) and initialize your instance using ``psm1 = psm('PSM1')``.

```

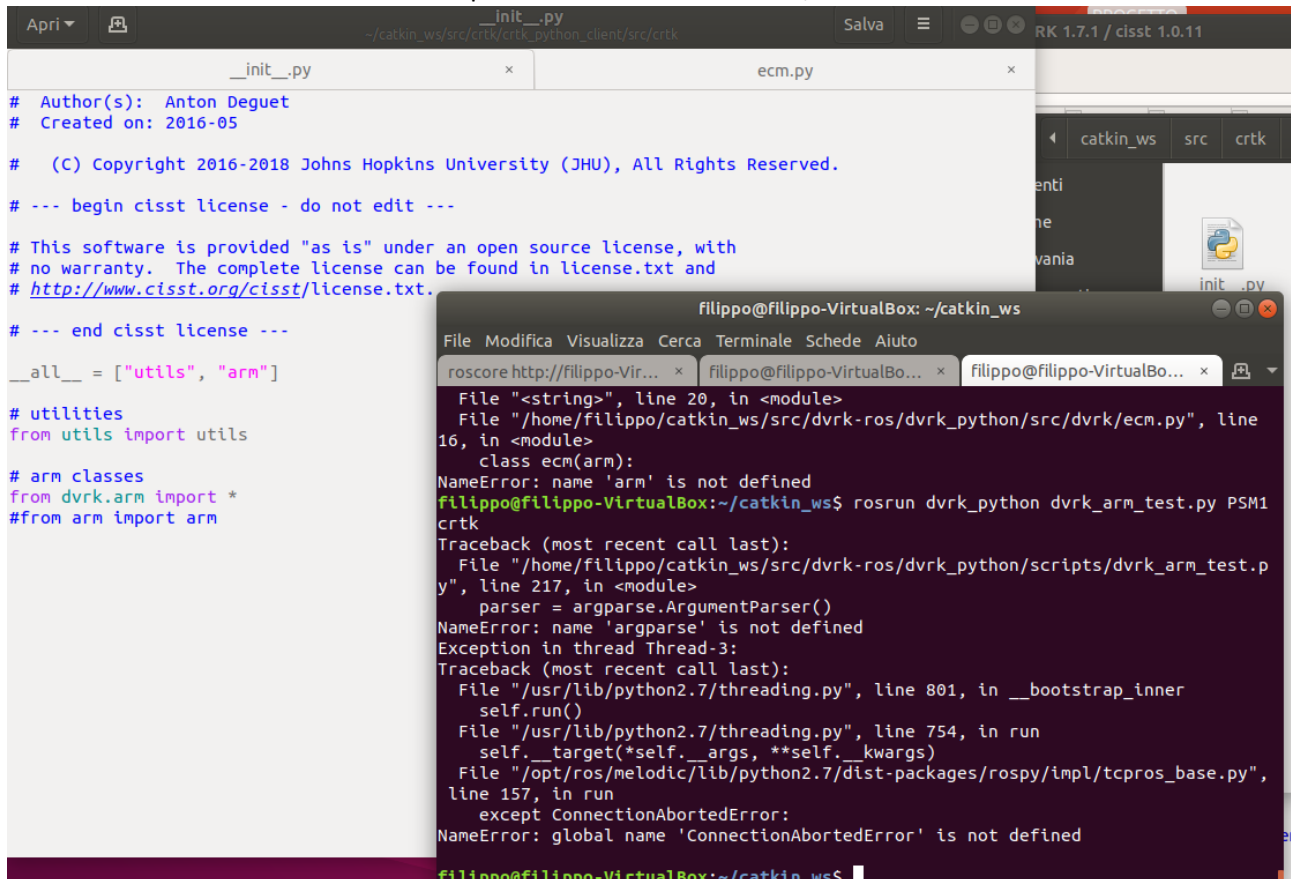
"""

```

Localione precisa dell'errore:



Ho modificato il codice cambiando l'import e sembra essere risolto, l'errore è diventato:



The screenshot shows a code editor with two tabs: `__init__.py` and `ecm.py`. The `__init__.py` file contains the following code:

```
# Author(s): Anton Deguet
# Created on: 2016-05

# (C) Copyright 2016-2018 Johns Hopkins University (JHU), All Rights Reserved.

# --- begin cisst license - do not edit ---

# This software is provided "as is" under an open source license, with
# no warranty. The complete license can be found in license.txt and
# http://www.cisst.org/cisst/license.txt.

# --- end cisst license ---

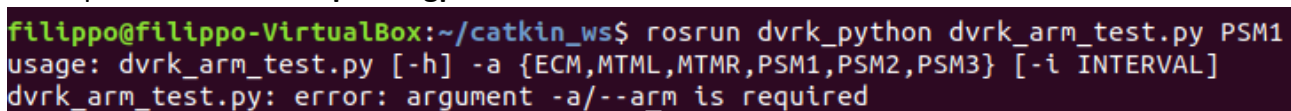
__all__ = ["utils", "arm"]

# utilities
from utils import utils

# arm classes
from dvrk.arm import *
# from arm import arm
```

The `ecm.py` file is open but its content is not visible. Below the code editor is a terminal window titled `filippo@filippo-VirtualBox: ~/catkin_ws`. It shows the command `roscore http://filippo-Vir... x filippo@filippo-VirtualBo... x filippo@filippo-VirtualBo... x` and the output of `rosrun dvrk_python dvrk_arm_test.py PSM1 crtk`. The output shows a `NameError: name 'arm' is not defined` and a `NameError: name 'argparse' is not defined`. The terminal also shows a `Traceback (most recent call last):` and a `ConnectionAbortedError`.

Ho importato nel file `"import argparse"` e ora sembra funzionare:



The screenshot shows a terminal window with the command `rosrun dvrk_python dvrk_arm_test.py PSM1` and its output:

```
usage: dvrk_arm_test.py [-h] -a {ECM,MTML,MTMR,PSM1,PSM2,PSM3} [-i INTERVAL]
dvrk_arm_test.py: error: argument -a/--arm is required
```

Da qui deduco che il funzionamento scritto ad inizio file non è del tutto corretto: `rosrun`

`dvrk_python dvrk_arm_test.py <arm-name>`

Adesso con `"rosrun dvrk_python dvrk_arm_test.py -a PSM1"` sembra partire per poi terminare con un nuovo errore:


```
filippo@filippo-VirtualBox: ~/catkin_ws
File Modifica Visualizza Cerca Terminale Schede Aiuto
roscore http://filippo-Vir... x filippo@filippo-VirtualBo... x filippo@filippo-VirtualBo... x
out of 7
/dvrk_arm_test_18085_1605366874571 -> servo_jp complete in 5.16 seconds (expected 5.00)
/dvrk_arm_test_18085_1605366874571 -> starting move_jp
/dvrk_arm_test_18085_1605366874571 -> testing goal joint position for 2 joints out of 7
/dvrk_arm_test_18085_1605366874571 -> move_jp complete
/dvrk_arm_test_18085_1605366874571 -> starting servo_cp
Traceback (most recent call last):
  File "/home/filippo/catkin_ws/src/dvrk-ros/dvrk_python/scripts/dvrk_arm_test.py", line 228, in <module>
    application.run()
  File "/home/filippo/catkin_ws/src/dvrk-ros/dvrk_python/scripts/dvrk_arm_test.py", line 208, in run
    self.servo_cp()
  File "/home/filippo/catkin_ws/src/dvrk-ros/dvrk_python/scripts/dvrk_arm_test.py", line 133, in servo_cp
    initial_cartesian_position.p = self.arm.setpoint_cp().p
  File "/home/filippo/catkin_ws/src/crtk/crtk_python_client/src/crtk/utils.py", line 341, in __setpoint_cp
    raise RuntimeError('unable to get setpoint_cp')
RuntimeWarning: unable to get setpoint_cp
filippo@filippo-VirtualBox:~/catkin_ws$
filippo@filippo-VirtualBox:~/catkin_ws$
```

Se utilizzo il file “dvrk_psm_test” al posto di “dvrk_arm_test” sembra funzionare senza errori.
Però in console appaiono messaggi di errore in giallo:

Arms

Power Off
Power On
Home
PSM1
Tele operation
Start
Stop
scale 0.20
Inputs
Operator
Clutch
Camera
Audio
Direct control
Component Viewer

PSM1 PID PSM1

Current position (deg)	Desired position (deg)	Current effort (Nm)	Desired effort (Nm)
0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000

PGain 100.000 100.000 6000.00 4.000 4.000 4.000 4.000
Dgain 3.000 3.000 80.000 0.030 0.030 0.030 0.030
IGain 0.00000 0.00000 0.00000 0.00200 0.00200 0.00200 0.00200

Index
0
Current position
Desired position
Current velocity
Current effort
Desired effort

☐ Direct control ☒ Enable PID ☒ Enable tracking error

16:18:00 status #16: PSM1: current state CHANGING_COUPLING_ADAPTER
16:18:00 status #17: PSM1: current state ENGAGING_ADAPTER
16:18:00 status #18: PSM1: current state HOMED
16:18:00 warning #1: PSM1: tool type requested from user
16:18:11 warning #2: PSM1: jaw_move_jp, arm not ready
16:18:41 warning #3: PSM1: jaw_move_jp, arm not ready
16:19:11 warning #4: PSM1: jaw_move_jp, arm not ready (2 errors)

```
filippo@filippo-VirtualBox: ~$ rosrun dvrk_python dvrk_psm_test.py -a PSM1
[dvrk_PSM1_node-2] process has died [pid 32695, exit code 255, cmd /h
16:18:00 status #16: PSM1: current state CHANGING_COUPLING_ADAPTER
16:18:00 status #17: PSM1: current state ENGAGING_ADAPTER
16:18:00 status #18: PSM1: current state HOMED
16:18:00 warning #1: PSM1: tool type requested from user
16:18:11 warning #2: PSM1: jaw_move_jp, arm not ready
16:18:41 warning #3: PSM1: jaw_move_jp, arm not ready
16:19:11 warning #4: PSM1: jaw_move_jp, arm not ready (2 errors)
```

INSTALLAZIONE

- 0- *sudo apt update*
sudo apt-get upgrade
- 1- Configurazione ROS MELODIC + UBUNTU BIONIC 18.04LTS
- 2- Ci sono da installare alcuni pacchetti:
Ubuntu 18.04 with ROS Melodic: `sudo apt install libxml2-dev libraw1394-dev libncurses5-dev qtcreator swig sox espeak cmake-curses-gui cmake-qt-gui git subversion gfortran libcppunit-dev libqt5xmlpatterns5-dev python-wstool python-catkin-tools`
- 3- *source /opt/ros/melodic/setup.bash*
- 4- Creo il workspace catkin
mkdir ~/catkin_ws
- 5- *cd ~/catkin_ws*
- 6- Viene utilizzato wstool per pullare tutto il codice da git
wstool init src
- 7- Inizializzo il workspace di catkin
catkin init
- 8- Il codice deve essere compilato in release mode
catkin config --cmake-args -DCMAKE_BUILD_TYPE=Release
- 9- *cd src*
- 10- Il file dvrk_ros.install è stato creato da me e non è altro che una copia del file presente su https://raw.githubusercontent.com/jhu-dvrk/dvrk-ros/devel/dvrk_ros.rosinstall ma con tutti i branch di git settati a "master". Serve per indicare quali repositories pullare.
wstool merge /home/filippo/dvrk_ros.rosinstall
- 11- Pullo I repositories indicate precedentemente
wstool up
- 12- Compilo tutto quanto
catkin build
- 13- Faccio in modo che il source di /home/filippo/catkin_ws/devel/setup.bash venga fatto in automatico all'avvio del terminale, quindi:
gedit ~/.bashrc
E in fondo aggiungo:
if [-f ~/catkin_ws/devel/setup.bash]; then
. ~/catkin_ws/devel/setup.bash
fi
- 14- Questi pacchetti non sono più supportati. E probabilmente nel branch "devel" stanno usando la nuova versione di ros-hydro ma io sto usando la versione "master" quindi non so se l'assenza di questi pacchetti possa dare problemi.

You will also need some extra Python packages:

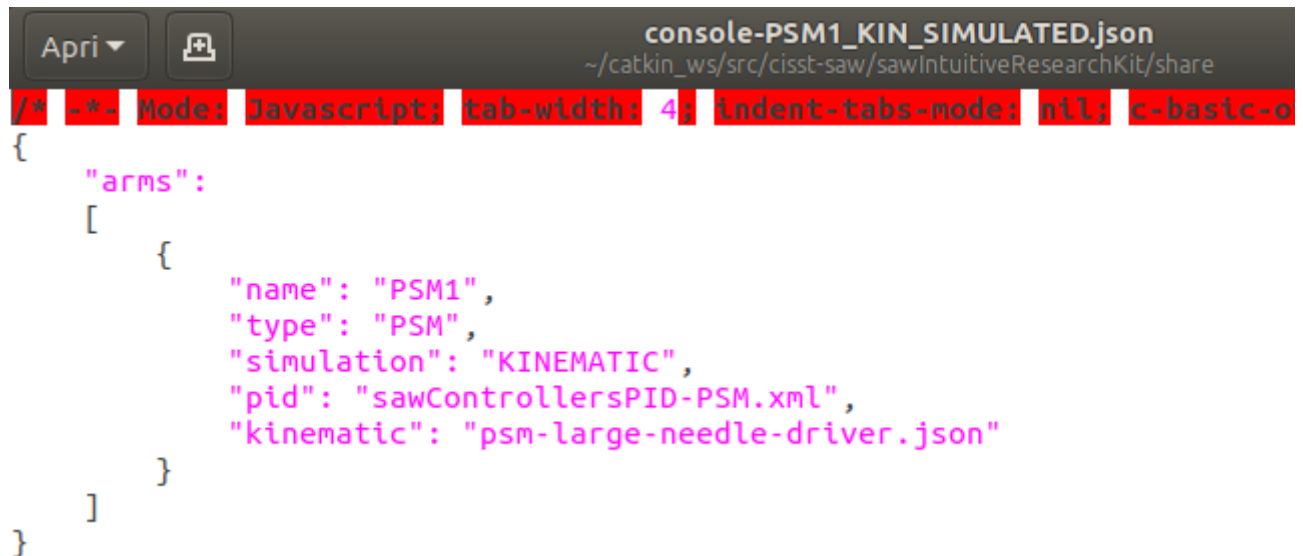
```
sudo apt-get install ros-hydro-python-orocos-kdl ros-hydro-orocos-kinematics-dynamics ros-hydro-tf
```

15- Launch script per il braccio che desidero, in questo caso PSM1, con RVIZ

(è consigliato modificare nel file “**dvrk_arm_rviz.launch**” il joint_state (deprecato) con robot_joint_state)

roslaunch dvrk_robot dvrk_arm_rviz.launch arm:=PSM1

config:=/home/filippo/catkin_ws/src/cisst-saw/sawIntuitiveResearchKit/share/console-PSM1_KIN_SIMULATED.json



```
console-PSM1_KIN_SIMULATED.json
~/catkin_ws/src/cisst-saw/sawIntuitiveResearchKit/share
/* -*- Mode: Javascript; tab-width: 4; indent-tabs-mode: nil; c-basic-o
{
  "arms":
  [
    {
      "name": "PSM1",
      "type": "PSM",
      "simulation": "KINEMATIC",
      "pid": "sawControllersPID-PSM.xml",
      "kinematic": "psm-large-needle-driver.json"
    }
  ]
}
```

- Il “name” del braccio è uno tra: MTML, MTMR, PSM1, PSM2, PSM3, ECM o SUJ.
- Il “type” è uno tra MTM, PSM, ECM o SUJ.
- Il “PID” è richiesto per qualunque braccio, sia in simulazione che non.
- Il campo “Simulation” indica che si tratta di una simulazione e c’è solo il tipo “kinematic” al momento.

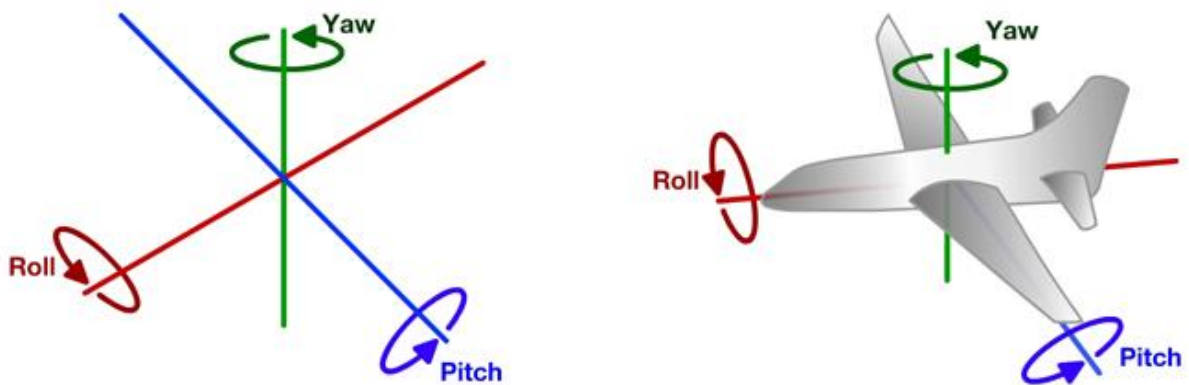
Il file “**sawControllersPID-PSM.xml**” contiene delle specifiche per ogni **GIUNTO** (7) indicando il nome, il tipo, e i limiti di giunto.

Il gripper è considerato revolute?

Il file “**psm-large-needle-driver.json**” fornisce:

- La tabella “**DH**” del manipolatore PSM e vengono quindi specificati i **LINK** (il valore 1.5708 viene tradotto come $\pi/2$).
- “**Tooltip-offset**” offre una matrice di rotazione per combinare l’offset di rotazione/traslazione rispetto al centro del gripper.
(con l’aggiunta di un offset di traslazione è possibile definire le cinematiche quando la pinza è chiusa)
- “**coupling**” : è necessario fornire 4 matrici, due per l'accoppiamento di posizione e 2 per l'accoppiamento di sforzo.

- **“tool-engage-position”**: Questi valori sono le posizioni del giunto superiore / inferiore utilizzate durante la procedura di innesto (valori del giunto, non dell'attuatore!). Poiché le prime 3 articolazioni non devono essere attivate, l'attuale implementazione C++ ignorerà i valori forniti. Vengono utilizzati gli ultimi 4 valori. Notare che per un driver ago l'ultimo giunto (apertura ganascia) non viene utilizzato (rimane chiuso) ma, a causa dell'accoppiamento, tutti gli ultimi 4 attuatori si muoveranno comunque.
- **“tool-joint-limit”**
- **“homing-zero-position”**: il braccio non andrà a zero durante la procedura di homing. E' possibile sovrascriverlo utilizzando questo flag ed impostarlo ad 1.



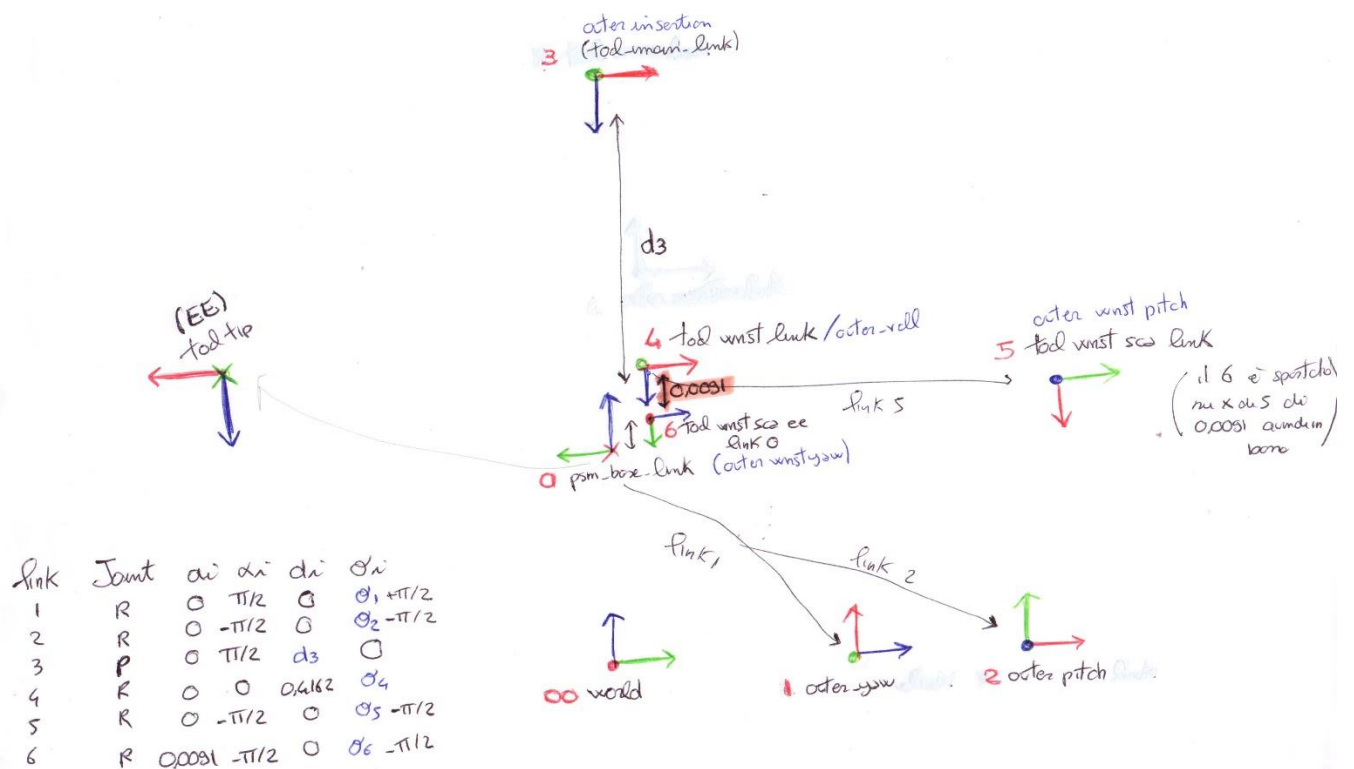
Il file **“dvrk_arm_rviz.launch”** che è il file lanciato. Include in modo ricorsivo i file:
 (Continuos è come Revolute ma non ha limiti di giunto)
 (Fixed non è un vero e proprio giunto perché non può muoversi)

- PSM1.urdf.xacro
 - /model/psm.urdf.xacro
 - /model/psm.base.urdf.xacro
 - Link world
 - Joint fixed F
 - Link 0 psm_base_link
 - Joint 1 outer_yaw R
 - Link 1 outer_yaw_link
 - Joint 2 outer_pitch R
 - Link 2 outer_pitch_link
 - Joint 2-1 outer_pitch_1 C
 - Link 2-1 outer_pitch_back_link
 - Joint 2-2 outer_pitch_2 C
 - Link 2-2 outer_pitch_front
 - Joint 2-3 outer_pitch_3 C
 - Link 2-3 outer_pitch_bottom
 - Joint 2-4 outer_pitch_4 C
 - Link 2-4 outer_pitch_top
 - Joint 2-5 outer_pitch_5 C
 - Link 2-5 outer_insertion_link

- Joint 3 outer_insertion P
 - Link 3 tool_main_link
 - Joint 4 outer_roll R
 - Link 4 tool_wrist_link
 - Joint 4-1 outer_roll_shaft F
 - Link 4-1 tool_wrist_shaft_link
-
- /model/psm.tool.sca.urdf.xacro
 - Viene scelto settando PSM1
 - Joint 5 outer_wrist_pitch R
 - Link 5 tool_wrist_sca_link
 - Joint 6 outer_wrist_yaw R
 - Link 6 tool_wrist_sca_shaft_link
 - Joint 7 jaw R
 - Link 7 tool_wrist_sca_ee_link_0
 - Joint 7-1 jaw_mimic_1 R
 - Link 7-1 tool_wrist_sca_ee_link_1
 - Joint 7-2 jaw_mimic_2 R
 - Link 7-2 tool_wrist_sca_ee_link_2
 - Joint tool_tip F

 - /model/psm.tool.caudier.urdf.xacro
 - Viene scelto settando PSM2
 - /model/psm.snake.urdf.xacro
 - Viene scelto settando PSM3
 - Macro per psm con snake/sca/caudier

Avviando RVIZ e settando i frame visibili ho notato che molti erano frame di “supporto”, sono andato nel file “**dvrk_arm_rviz.launch**” che includeva un file “**PSM1.urdf.xacro**” che ne includeva altri in modo ricorsivo (vedi sopra). Ho analizzato singolarmente ogni link/giunto e ho stabilito quali fossero le terne portanti partendo dal primo giunto e applicando la tabella DH (che si trova nel file “**psm_large_needle_driver.json**” per trovare tutti gli altri, in modo da verificare con mano le implementazioni già fatte.



Dopo aver eseguito “dvrk_arm_rviz.launch” ed eseguito il comando “rostopic list”, ci sono 5 nodi attivi:

```

filippo@filippo-VirtualBox:~$ rostopic list
/dvrk/PSM1/joint_state_publisher
/dvrk/PSM1/robot_state_publisher
/rosout
/rviz
/sawIntuitiveResearchKitdvrk

```

Con “rostopic list” è possibile vedere tutti i topic attivi.

Per il momento, sulla guida al link <https://github.com/jhu-dvrk/sawIntuitiveResearchKit/wiki/Kinematics-Simulation> è consigliato di interfacciarsi con ROS mediante script presenti in “/catkin_ws/src/dvrk-ros/dvrk_python”

16- Per eseguire l’applicazione QT senza eseguire il nodo di RVIZ

`roslaunch dvrk_robot dvrk_console_json -j /home/filippo/catkin_ws/src/cisst-saw/sawIntuitiveResearchKit/share/console-PSM1_KIN_SIMULATED.json`

C’è poi uno script già fatto per poter comunicare tramite nodi ros (che si trova nel percorso indicato allo step 15) con la console

`roslaunch dvrk_python dvrk_arm_test.py PSM1` o
`roslaunch dvrk_python dvrk_psm_test.py`

Sono script già fatti da provare.

17- Al link https://github.com/jhu-dvrk/dvrk-ros/tree/master/dvrk_python c’è uno script con le possibili funzioni che possono essere chiamate. (Funzioni definite in

/home/filippo/catkin_ws/src/dvrk-ros/dvrk_python/src/dvrk/arm.py). L'idea adesso è quella di indagare sulle varie funzioni per capire come creare uno script che possa ottenere delle informazioni riguardo la cinematica.

18- La sottoscrizione/pubblicazione sui nodi ROS avviene tramite le funzioni invocate nello script python e definite in **"arm.py"**.

Altrimenti posso vedere la lista di topic attivi e sottoscrivermi ad un topic per ricevere le informazioni che vengono pubblicate, come ad esempio:

rostopic echo /dvrk/PSM1/state_joint_current

Ma comunque le funzioni implementate in arm.py gestiscono questi topic tramite funzioni, evitando così di utilizzare il terminale e di fare tutto tramite script python.

Le spiegazioni sui vari topic possono essere trovate al link <https://github.com/jhu-dvrk/sawIntuitiveResearchKit/wiki/Components-APIs>

Topic di sola lettura (subscribe):

- GetPositionJoint = Posizione
- GetPositionJointDesired = L'ultima posizione dei giunti desiderata, usata poi dal PID controller.
- GetStateJoint = Posizione + Velocità
- GetSateJointDesired
- GetPositionCartesian = Posizione cartesiana in base agli encoder. (Se c'è un base frame viene incluso, es. PSM su SUJ)
- GetPositionCartesianDesired =
- GetPositionCartesianLocal = Posizione cartesiana in base agli encoder ma si basa solo sulla catena cinematica e non include alcun base frame.
- GetJacobianBody =
- GetJacobianSpatial =

Topic di sola scrittura (public):

- SetBaseFrame = Serve per impostare un frame di base dal quale poi vengono fatti tutti i calcoli cinematici diretti e inversi.
(Questa trasformazione viene anteposta alla catena cinematica, quindi diventa, **Base_frame * base_offset * DH_chain * tooltip_offset**)
- SetRobotControlState = Serve per impostare lo stato desiderato del braccio robotico. (<https://github.com/jhu-dvrk/sawIntuitiveResearchKit/blob/master/components/code/mtsIntuitiveResearchKitArmTypes.cdg>)
- SetPositionJoint = Serve per settare la posizione del giunto desiderata. La posizione desiderata viene inviata al controllore PID. Il braccio deve essere in **DVRK_POSITION_JOINT**, la modalità si può cambiare tramite SetRobotControlState.
Nessuna traiettoria verrà generate.
- SetPositionGoalJoint = Imposto la posizione obiettivo. Viene generate una traiettoria che verrà poi inviata al controllore PID. Quando l'obiettivo è raggiunto viene attivato un evento e sarà possibile indicare altri obiettivi.

Il braccio deve essere in modalità **DVRK_POSITION_GOAL_JOINT**.

- SetPositionCartesian = Viene impostata la posizione cartesiana desiderata, il controller calcola la cinematica inversa e invierà la posizione del giunto desiderata al PID.

Nessuna traiettoria verrà generate.

Il braccio deve essere in modalità **DVRK_POSITION_CARTESIAN**.

- SetPositionGoalCartesian = Come quella precedente ma viene generate la traiettoria.

19- Funzioni da capire:

- get_current_joint_position() - TOPIC: position_joint_current(state_joint_current) ==
- get_current_position() – TOPIC: position_cartesian_current ==
- get_desired_joint_position() – TOPIC: position_joint_desired == ritorna l'ultima posizione dei ginti richiesta usata dal PID controller.
- get_desired_position() – TOPIC: position_cartesian_desired ==
- get_current_position_local() – TOPIC: position_cartesian_local_current ==
- dmove_joint_one(-0.05, 2) == Muove un singolo giunto, l'indice dei giunti parte da 0, quindi in questo caso muove il giunto 2.
Il primo parametro -0,05 indica di quanto “**incrementalmente**” muovere il giunto. (In pratica viene sommato il valore indicato al valore attuale del giunto)
- move_joint_one(0.2, 0) == Come la funzione precedente ma il primo parametron indica il valore “**assoluto**” di quanto muovere il giunto. (Non viene tenuto conto del valore attuale del giunto)
- dmove_joint_some(numpy.array([1.2217, -0.12]), numpy.array([0, 2])) == per muovere più ginuti per volta, devo passare l'array dei valori seguito da un array degli indici dei giunti.
- p.dmove_joint(numpy.array([1.2217, 0.0, -0.12, 0.0, 0.0, 0.0])) == Se invece voglio indicare il valore di ogni giunto e quindi spostarli tutti posso fare così.
- p.move(PyKDL.Vector(0.0, 0.0, -0.113)) (o la variante dmove) == Non è che altro che la **cinematica inversa**. Indico un punto sul piano cartesiano dove voglio il mio end effector. Ovviamente viene assegnato ai giunti un valore tale da portare l'EE in quel punto.

Ora l'idea è di provare le varie funzioni, cercando di capire i valori di ritorno.

Le funzioni invocate sono rispettivamente:

- `get_current_joint_position()`
- `get_current_position()`

```
Current Joint position ---- TOPIC: position_joint_current(state_joint_current) :
[ 0.    0.    0.12  0.    0.    0. ]

Current position ---- TOPIC: position_cartesian_current :
[[-2.69849e-11,      1, -9.91219e-17;
    1, 2.69849e-11, 7.34641e-06;
    7.34641e-06, 9.91194e-17,      -1]
[ 4.16909e-07, 4.50335e-07,      -0.1135]]

Current Joint position ---- TOPIC: position_joint_current(state_joint_current) :
[ 0.    0.    0.07  0.    0.    0. ]

Current position ---- TOPIC: position_cartesian_current :
[[-2.69849e-11,      1, -9.91219e-17;
    1, 2.69849e-11, 7.34641e-06;
    7.34641e-06, 9.91194e-17,      -1]
[ 2.33249e-07, 2.66675e-07,      -0.0635]]
```

Current_Joint_Position:

name: [outer_yaw, outer_pitch, outer_insertion, outer_roll, outer_wrist_pitch, outer_wrist_yaw]

position: [0.0, 0.0, 0.12, 0.0, 0.0, 0.0]

Outer_yaw	0.0 degree
Outer_pitch	0.0 degree
Outer_insertion	0.12 mm
Outer_roll	0.0 degree
Outer_wrist_pitch	0.0 degree
Outer_wrist_yaw	0.0 degree

Current position

x: 4.1690877922e-07

y: 4.50334945663e-07

z: -0.1134999999998

Mentre la matrice 3x3 precedente indica l'orientazione: **p.dmove_joint_one(-0.05, 2)** la quale muove il giunto 3 (prismatico) di -0,05 infatti come si vede nello screenshot i valori cambiano in modo corretto. La matrice di rotazione dell'end effector rimane invariata perchè non sono stati attribuiti valori ai giunti rotazionali.

La cosa strana è che anche i valori di x e y variano, seppur di pochissimo.

Dopo aver ottenuto i valori, è stata invocata la funzione:

Per capire meglio i valori, mi sono sottoscritto manualmente ai due topic, rispettivamente:

- `rostopic echo /dvrk/PSM1/state_joint_current`
- `rostopic echo /dvrk/PSM1/position_cartesian_current`

Provando a muovere il giunto 1 R (outer_yaw) di 90°:

p.dmove(1.5708, 0)

possiamo notare che il valore finale del giunto sia di 1.22173048 che corrisponde a 70°, questo perchè nel file “sawControllersPID-PSM.xml” sono specificati i vari limiti di giunto come ad esempio per il giunto 1 R:

```
<joints>
  <joint index="0" name="outer_yaw" type="Revolute">
    <pid PGain="100.0" DGain="3.0" IGain="0.0" OffsetTorque="0.0" Forget="1.0"
Nonlinear="0.0"/>
    <limit MinILimit="-20" MaxILimit="20" ErrorLimit="0.5" Deadband="0.0" Units="rad"/>
    <pos LowerLimit="-70.0" UpperLimit="70.0" Units="deg"/>
  </joint>
```

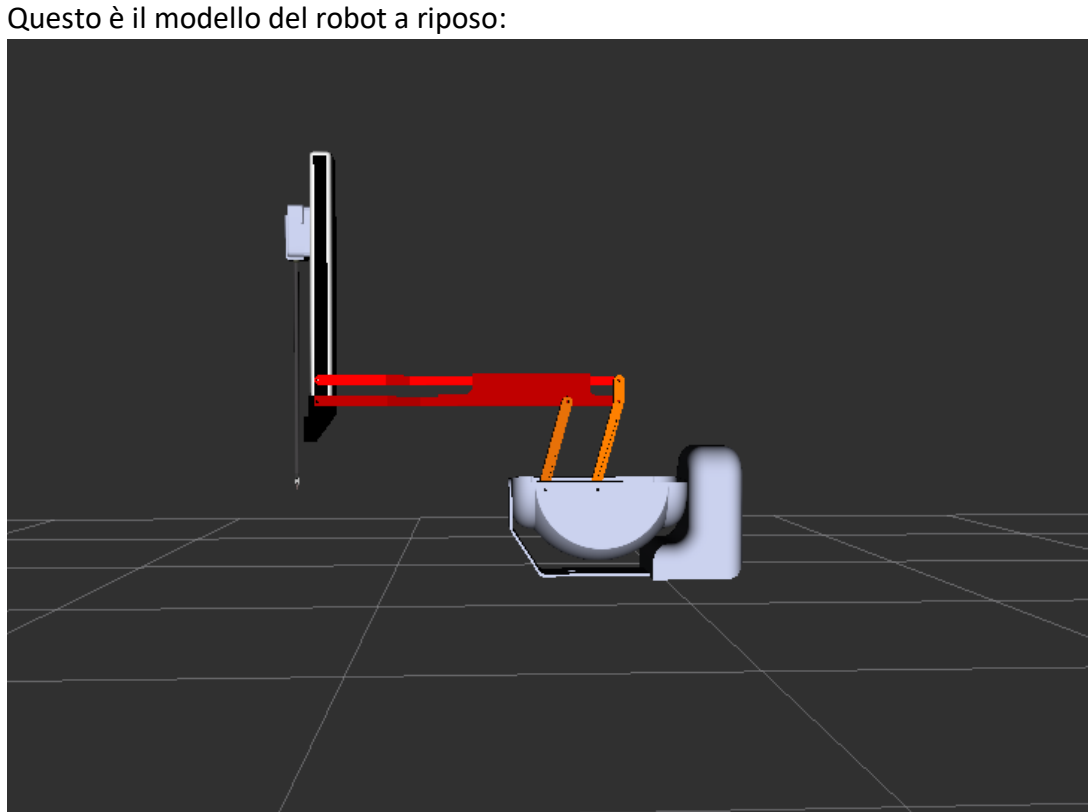
Avendo fatto ulteriori prove, assegnando vari dati a giunti, posso confermare che la terna base rispetto la quale viene data la posizione dell’end-effector è “**psm_base_link**”, visibile in RVIZ.

Altra prova, porto il prismatico a d3=0 e alzo il secondo giunto rotoidale (-50 gradi). Per verifica dovrei ottenere un valore della Z di EE alto e un valore della Y negativo e non più tendente a 0. Ed infatti così accade.

Muovere il robot

Per poter muovere il robot, si uniscono i concetti visti prima. Quindi prima di tutto si esegue il launch file per avviare i vari nodi e RVIZ (punto 15). In seguito si può eseguire uno script python per dare comandi al robot e quindi farlo muovere “**roslaunch dvrk_python myScript.py**”.

Qui di sotto viene riportato un esempio di codice con il prima-dopo del robot:



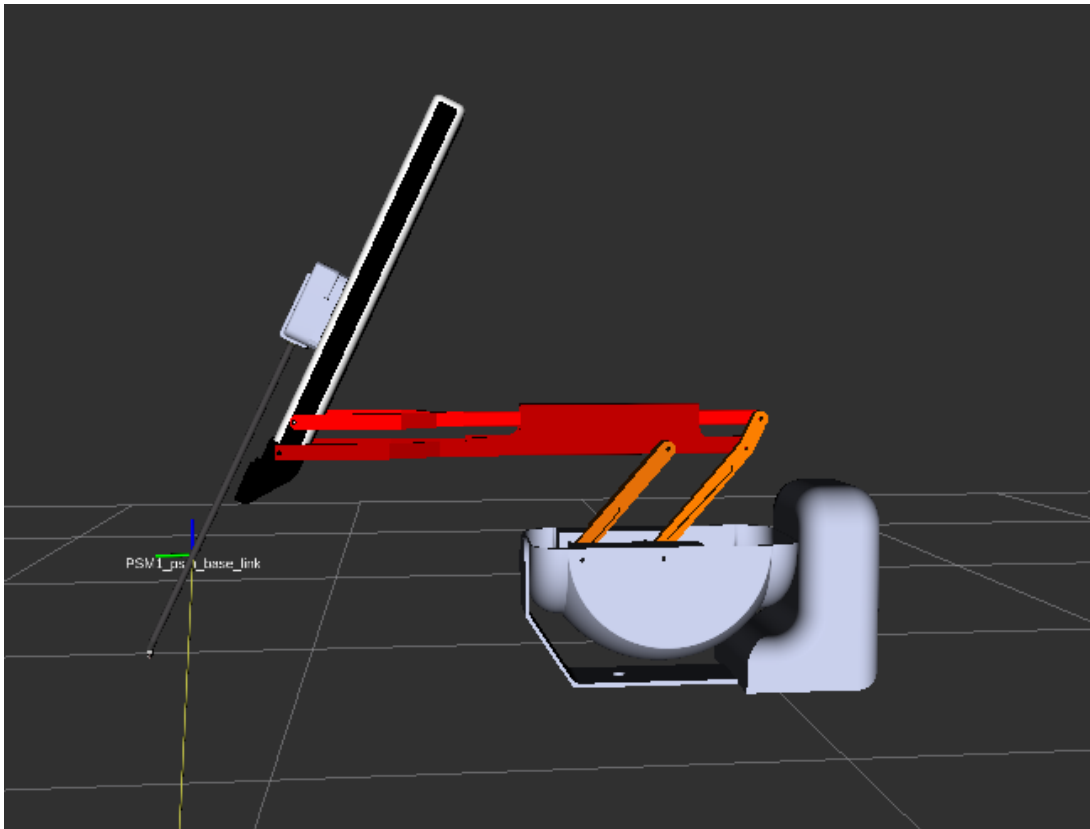
Nello script eseguito sono presenti questi due comandi:

- `p.home()`
- `p.dmove(PyKDL.Vector(0.0, 0.05, 0.0))`

Dove rispettivamente il primo porta il robot in uno stato dove la posizione dell'EE è $x=0$ $y=0$ $z=-0,1135$

Mentre il secondo essendo una "dmove" quindi una move relativa, quello che accade è che indico al robot di spostare l'EE di $x=0$ $y=0.05$ $z=0.0$ (5cm sulla y).

Al termine dello script l'EE si trova in $x=0$ $y=0.05$ $z=-0,1135$



Valori se sposto solo il giunto 3 P:

- $z = -0.2285$ in estensione massimo
- $z = -0.1135$ home
- $z = +0.065$ minima estensione

L'unico dubbio è che, se parto da $x, y = 0$ e $z = -0.1135$ ed eseguo la cinematica inversa indicando Move/dmove di 0 0 5.0 con una z esagerata, mi aspetto che l'EE finale sia molto in alto ma quello che ottengo è:

$x = 0$

$y = 0$

$z = -0.035$

Come se non potesse andare oltre, però se sposto il giunto 3 di 2.0 metri con una classica dmove allora l'EE raggiunge una posa di

$X = 0$

$Y = 0$

$Z = 0.065$

Questa è una move della **cinematica 3** al punto 0,0,0:

```
Current Joint position ---- TOPIC: position_joint_current(state_joint_current) :
[ 0.  0.  0.12 0.  0.  0. ]

Current position ---- TOPIC: position_cartesian_current :
[[ 1, 0, 0;
  0, 1, 0;
  0, 0, 1]]

Current Joint position ---- TOPIC: position_joint_current(state_joint_current) :
[ -1.22173048e+00 -8.72664626e-01  4.00000000e-02  4.15023416e-02
  3.67318820e-06 -1.57079265e+00]

Current position ---- TOPIC: position_cartesian_current :
[[ 0.733418, -0.604019, -0.311862;
  0.642231,  0.766047,  0.0266694;
  0.222792, -0.219848,  0.949753]
[ -0.0202346,  0.0256626, -0.00736489]]
```

Non ci può arrivare.

Però se invece All'inizio muovo il giunto 3 (**cinematica diretta**) riesco ad arrivare al punto 0,0,0:

```
Current Joint position ---- TOPIC: position_joint_current(state_joint_current) :
[ 0.  0.  0.12 0.  0.  0. ]

Current position ---- TOPIC: position_cartesian_current :
[[ -2.69849e-11, 1, -9.91219e-17;
  1, 2.69849e-11, 7.34641e-06;
  7.34641e-06, 9.91194e-17, -1]
[ 4.16909e-07, 4.50335e-07, -0.1135]]

Current Joint position ---- TOPIC: position_joint_current(state_joint_current) :
[ 0.  0.  0.0065 0.  0.  0. ]

Current position ---- TOPIC: position_cartesian_current :
[[ -2.69849e-11, 1, -9.91219e-17;
  1, 2.69849e-11, 7.34641e-06;
  7.34641e-06, 9.91194e-17, -1]
[ -1.19286e-19, 3.34262e-08, 1.84215e-13]]
```

Credo che il problema dietro a tutto ciò sia in quei valori tendenti allo zero ma non del tutto a zero.

Come viene calcolata la cinematica inversa?

L'obiettivo ora è di capire dove viene implementata(file) la cinematica inversa:

```
grep -lri 'inverse kinematic' /home/filippo/catkin_ws/src
```

Ho cercato tra i vari file fino ad ottenere dei riscontri e file interessati sono i seguenti (nei primi due viene utilizzata e negli ultimi due viene implementata):

- `~/catkin_ws/src/cisst-saw/sawIntuitiveResearchKit/components/code/mtsIntuitiveResearchkitPSM.cpp`
- `~/catkin_ws/src/cisst-saw/sawIntuitiveResearchKit/components/include/sawIntuitiveResearchKit/mtsIntuitiveResearchKitPSM.h`
- `~/catkin_ws/src/cisst-saw/cisst/cisstRobot/code/robManipulator.cpp`
- `~/catkin_ws/src/cisst-saw/cisst/cisstRobot/robManipulator.h`

robManipulator.h

```
/// Evaluate the forward kinematics
/**
 * Compute the position and orientation of each link wrt to the world frame
 * \param[input] q The vector of joint positions
 * \param[input] N The link number (0 => base, negative => end-effector)
 * \return The position and orientation, as a 4x4 frame
 */
virtual
vctFrame4x4<double>
ForwardKinematics( const vctDynamicVector<double>& q, int N = -1 ) const;

/// Evaluate the inverse kinematics
/**
 * Compute the inverse kinematics. The solution is computed numerically using
 * Newton's algorithm.
 * \param[input] q An initial guess of the solution
 * \param[output] q The inverse kinematics solution
 * \param Rts The desired position and orientation of the tool control point
 * \param tolerance The error tolerance of the solution
 * \param Niteration The maximum number of iterations allowed to find a solution
 * \return SUCCESS if a solution was found within the given tolerance and
 *         number of iterations. ERROR otherwise.
 */
virtual
robManipulator::Errno
InverseKinematics( vctDynamicVector<double>& q,
                  const vctFrame4x4<double>& Rts,
                  double tolerance=1e-12,
                  size_t Niteration=1000,
                  double LAMBDA=0.001 );

virtual
robManipulator::Errno
InverseKinematics( vctDynamicVector<double>& q,
                  const vctFrm3& Rts,
                  double tolerance=1e-12,
                  size_t Niteration=1000 );
```

Da come si può capire, la soluzione numerica della cinematica inversa viene calcolata mediante l'**algoritmo di newton**. (E' stata scelta una soluzione numerica perché probabilmente quella analitica o era di difficile derivazione o non esiste).

CENNI TEORICI

Per il calcolo della cinematica inversa ci sono due approcci possibili:

- Soluzione Analitica (in forma chiusa)
 - o Da preferire se può essere trovata (vedi condizione sufficiente)
 - o Viene fatta un'ispezione geometrica del manipolatore
 - o Si ricorre alla soluzione di equazioni polinomiali
- Soluzione Numerica (in forma iterativa)
 - o Serve nel caso di $n > m$ (manipolatore ridondante)
 - o Molto lento computazionalmente ma semplice da implementare
 - o Sfrutta la matrice dello Jacobiano Analitico

$$J_r(q) = \frac{\partial f_r(q)}{\partial q}$$

- o I principali due metodi sono quello del gradiente e di newton

La condizione **sufficiente** affinché un manipolatore a 6 DOF abbia una soluzione in forma chiusa è:

- O 3 giunti rotazionali consecutivi hanno gli assi che si intersecano in un punto (es. polso sferico)
- Oppure 3 giunti rotazionali consecutivi hanno tutti gli assi paralleli

$n > m \rightarrow$ manipolatore ridondante perché ho più gradi di libertà di quelli che mi servono per eseguire un compito. (Si parla di ridondanza del robot rispetto ad un task)

ALGORITMO DI NEWTON (metodo usato)

$$q^{k+1} = q^k + J_r^{-1}(q^k) [r_d - f_r(q^k)]$$

- **Calcolo dello Jacobiano Pseudo-Invertito**
- converge se q_0 (stima iniziale) sufficientemente vicina a q^* : $f_r(q^*) = r$
- ha problemi vicino alle singolarità dello Jacobiano $J_r(q)$
- in caso di robot ridondanti ($n > m$) serve l'uso della pseudo-inversa $J_r^\#(q)$
- ha convergenza quadratica se è vicino alla soluzione (rapido!)

Metodo del gradiente (altro metodo non usato)

$$q^{k+1} = q^k + \alpha J_r^T(q^k)(r_d - f_r(q^k))$$

- **Calcolo dello jacobiano trasposto**
- L'obiettivo è di minimizzare la funzione di errore mediante il feedback (retroazione)
- Si applica anche a robot ridondanti
- Può non convergere ad una soluzione ma non diverge mai

La dimensione di α (passo scalare) deve essere scelta in modo da garantire una diminuzione della funzione di errore ad ogni iterazione:

- Se troppo grande si rischia di perdere il "minimo"
- Se troppo piccolo la convergenza è estremamente lenta

Una soluzione numerica efficiente prevede:

1. fare iterazioni iniziali con il metodo del gradiente per avere una “convergenza sicura” ma più lenta avendo convergenza lineare
2. fare iterazioni finali con il metodo di newton per avvicinarsi alla soluzione in modo rapido (convergenza con tasso quadratico)

Scelte che devono essere fatte:

1. Inizializzazione di q_0 (genera una sola delle soluzioni)
2. Passo ottimale α nel metodo del gradiente
3. Criteri di arresto (quando fermarsi)

START

console

- *roscore*
- *roslaunch dvrk_robot dvrk_console_json -j /home/filippo/catkin_ws/src/cisst-saw/sawIntuitiveResearchKit/share/console-PSM1_KIN_SIMULATED.json*
- *rostopic echo /dvrk/PSM1/state_joint_current*
- *rostopic echo /dvrk/PSM1/position_cartesian_current*
- *roslaunch dvrk_python myScript.py*

rviz

- *roslaunch dvrk_robot dvrk_arm_rviz.launch arm:=PSM1 config:=/home/filippo/catkin_ws/src/cisst-saw/sawIntuitiveResearchKit/share/console-PSM1_KIN_SIMULATED.json*
- *roslaunch dvrk_python myScript.py*

Domande:

- dmove è cinematica inversa?
- Per la cinematica inversa devo conoscere le dimensioni dei link del robot? Dove le posso trovare
- Ha senso pensare ad una cinematica prima sui primi 3 giunti e in seguito sugli ultimi 3?
- Il RCM dov'è di preciso? Coincide ed è il base_frame?
- devo provare con le viste side e top? Prendo come Sistema di riferimento il “psm_base_link” come il Sistema base da cui esprimere Il punto cartesiano dell'EE? (Ma quindi devo rappresentare il robot in un altro modo visto che il Sistema di riferimento base non parte dalla vera base del robot)

Cose utili

PID (Controllo Proporzionale-Integrale-Derivativo) = è un Sistema in retroazione negativa impiegato nei sistemi di controllo. Dato un input cerca di mantenere l'errore che si può verificare a 0.

RETROAZIONE = è la capacità di un Sistema dinamico di tenere conto dei risultati del Sistema per modificare le caratteristiche del Sistema stesso. (Imparando dagli errori)

ENCODER = Gli "encoder" in robotica sono utilizzati per il controllo del movimento. E' un dispositivo di rilevamento che fornisce feedback, questo segnale di feedback può essere usato per determinare informazioni come velocità, posizione, direzione.

Mentre la **dinamica inversa** richiede coppie che producono una certa traiettoria temporale di posizioni e velocità, la **cinematica inversa** richiede solo un insieme statico di angoli articolari tale che un certo punto (o un insieme di punti) del personaggio (o robot) sia posizionato in una determinata posizione designata.

```
grep -lri 'joint_state_publisher' /home/filippo/catkin_ws/
```