Realizzazione di un braccio robotico e broadcasting della trasformata tra riferimento *solidale* e *fisso*

A Obiettivo del progetto

La seguente documentazione ha l'obiettivo di realizzare un robot grazie al middleware ROS e alcuni suoi pacchetti utili per lo sviluppo di applicazioni robotiche. Il progetto mira a costruire un robot manipolabile e pubblicare in broadcast la trasformata tra il riferimento fisso rispetto al robot **world** e il riferimento solidale all'end effector **ef** per essere visualizzata tramite lo strumento grafico *Rviz*.

B Configurazione Workspace

Il progetto è basato sul sistema operativo Linux 20.04, è importante verificare la propria versione perchè in base alla stessa è possibile scegliere quale versione del middleware installare. In particolare con la versione di Linux 20.04 la versione corrispondente di ROS è la versione **noetic**. L'installazione di ROS è presente sulla wiki al seguente link [1], a seguito dell'installazione e della verifica del corretto funzionamento dell'intera piattaforma è possibile procedere all'istanziazione dell'ambiente di sviluppo. Ecco un possibile modo per creare un workspace [2]:

1. creazione della cartella che rappresenta l'ambiente di lavoro

```
$ mkdir catkin_ws
```

2. posti all'interno della cartella creata al passo precedente, si utilizza il comando

```
$ cd ~/catkin_ws
$ catkin build
```

3. adesso è possibile creare il proprio package [3] al cui interno verranno inseriti tutti i file per la renderizzazione e la realizzazione del robot. Per creare un package si usa il seguente comando:

```
$ cd ~/catkin_ws/src
$ catkin_create_package <nome package> rospy urdf
```

   il comando prende come paramentri: il nome del pacchetto e le dipendenze, che rappresentano quei pacchetti o librerie necessarie per il corretto funzionamento delle componenti software presenti nel package stesso. Le dipendenze sopra scritte fanno riferimento rispettivamente al pacchetto wrapper di ros per il linguaggio di programmazione python e **urdf** è la libreria necessaria per la realizzazione e renderizzazione grafica del robot

Il pacchetto al termine dei passaggi appena eseguiti sarà in questo modo:

```
$    CMakeLists.txt          package.xml          src
```

il primo elemento corrisponde al file che viene eseguito al momento della compilazione del pacchetto stesso, insieme al file package.xml entrambi necessari per la definizione e descrizione di tutte le dipendenze presenti all'interno del progetto, con il comando eseguito al passaggio precedente non sarà necessario modificare nessuno dei due file. La cartella **src** rappresenta la cartella al cui interno è possibile inserire file o script di codice. Il package necessita della creazione di altre cartelle, attraverso i seguenti comandi:

```
$ cd ~/catkin_ws/src/own_package
$ mkdir launch rviz urdf
```

le cartelle appena create definiscono la struttura del pacchetto, la cartella launch ospiterà il file **.launch** utilizzabile per lanciare i nodi ros per configurare ed eseguire il programma di visualizzazione e manipolazione del robot. La cartella *rviz* al cui interno verrà inserito il file di configurazione per la visualizzazione tramite lo strumento grafico **Rviz**. L'ultima cartella al cui interno verrà inserito il file che definisce la corretta visualizzazione, in termini di *giunti* e *link*

C Realizzazione grafica del robot

Il file che è necessario modificare per la realizzazione del robot ha la seguente estensione:

```
$ cd /<nome_package>/urdf
$ sudo nano <nome_file>.urdf.xacro
```

la struttura tipica di questo file è quella di un file *.xml*, un esempio è il seguente:

```
<link name="base">
    <visual>
        <origin xyz="0 0 0" rpy="1.57 0 0"/>
        <geometry>
             <box size="0.5 0.5 0.125" />
        </geometry>
        <material name="c_base" />
    </visual>
</link>


<joint name="base_to_l1" type="revolute">
    <parent link="base" />
    <child link="l1" />
    <origin xyz="0 0.1 0" rpy="1.57 0 0"  />
    <axis xyz="0 0 1" />
    <limit effort="0.1" lower="-3.14" upper="3.14" velocity="0.2"/>
</joint>


<link name="l1">
    <visual>
        <origin xyz="0.0 0 0" rpy="0 0 1.57"/>
        <geometry>
             <cylinder length="0.05" radius="0.1" />
        </geometry>
        <material name="gray_j_side" />
    </visual>
    <visual>
        <origin xyz="0 0 -0.5" rpy="0 0 1.57"/>
        <geometry>
            <cylinder length="1.0" radius="0.04" />
        </geometry>
        <material name="c_l1" />
    </visual>
</link>
```

Come si può vedere ci sono dei tag con nome esplicitamente descrittivi: il tag <*material*> definisce il materiale o per essere più precisi il colore con cui è renderizzato il componente graficamente. Il tag <*link*> che prende come attributo il **nome** del link del robot, la **geometria** e la sua **posizione** relativa rispetto al giunto [4]

```
<link name="l1">
    <visual>
        <origin xyz="0.0 0 0" rpy="0 0 1.57"/>
        <geometry>
             <cylinder length="0.05" radius="0.1" />
        </geometry>
        <material name="gray_j_side" />
    </visual>
    <visual>
        <origin xyz="0 0 -0.5" rpy="0 0 1.57"/>
        <geometry>
            <cylinder length="1.0" radius="0.04" />
        </geometry>
        <material name="c_l1" />
    </visual>
</link>
```

il tag <*joint*> che rappresenta la definizione del giunto di collegamento tra due link in particolare a seconda del tipo di giunto si possono definire le **velocità** e gli **spostamenti** [5].

```
<joint name="base_to_l1" type="revolute">
    <parent link="base" />
    <child link="l1" />
    <origin xyz="0 0.1 0" rpy="1.57 0 0"  />
    <axis xyz="0 0 1" />
    <limit effort="0.1" lower="-3.14" upper="3.14" velocity="0.2"/>
</joint>
```

Il file sopra descritto può essere editato a piacere per ottenere strutture robotiche di qualsiasi complessità.

D File di launch

All'interno della cartella launch deve essere creato un file di launch. File tipico dell'ambiente ROS che da la possibilità di eseguire molteplici nodi con l'esecuzione di un singolo comando da terminale:

```
roslaunch <nome_package> <file.launch>
```

il file di launch all'interno viene definito anch'esso attraverso dei tag e deve avere la seguente struttura:

```
<launch>

  <arg name="model" default="$(find robo_exam_package)/urdf/robot_description.urdf.xacro"/>
  <arg name="gui" default="True" />

  <param name="robot_description" command="xacro $(arg model)" />
  <param name="use_gui" value="$(arg gui)"/>

  <!--!!!! inserire di seguito i comandi per il lancio dei nodi con eventuali argomenti-->

  <node name="joint_state_publisher_gui" pkg="joint_state_publisher_gui" type="joint_state_publisher_gui"/>
  <node name="robot_state_publisher"     pkg="robot_state_publisher"     type="robot_state_publisher"    />
  <node name="rviz"                      pkg="rviz"                      type="rviz"
   args="-d $(find robo_exam_package)/rviz/arm_description.rviz"       />
```

Al termine della scrittura del file di launch per poterlo lanciare è necessario eseguire il seguente comando:

```
$ cd ~/catkin_ws
$ source devel/setup.bash
```

All'interno del file sono stati definiti tre *nodi* principali:

1. **joint_state_publisher_gui** è un package ros che dà la possibilità di muovere conoscendo la struttura del robot i giunti effettuando così una sperimentazione grafica pratica per evidenziare criticità o il suo corretto funzionamento

2. **robot_state_publisher** è un package che in stretto legame con il pacchetto descritto in precedenza, pubblica lo stato dei link e giunti del robot.

3. **rviz** è il nodo che serve per lanciare lo strumento grafico su cui viene renderizzato il robot

E File python per il broadcasting della trasformata tra ef e world

L'obiettivo di questo progetto è realizzare graficamente un robot e pubblicare e renderizzare visualmente la trasformata tra l'end effector che rappresenta il riferimento solidale al robot e il **world** che rappresenta il riferimento fisso rispetto al robot. Il file ha un metodo che si occupa della pubblicazione della trasformata calcolata e ha questa forma [6]:

```
def get_transform():
    rospy.init_node('tf2_listener', anonymous=True)

    #Inizializzazione del broadcaster
    broadcaster = tf2_ros.TransformBroadcaster()
    tfBuffer = tf2_ros.Buffer()
    listener = tf2_ros.TransformListener(tfBuffer)
    rate = rospy.Rate(10.0)

    while not rospy.is_shutdown():
        try:

            transform = tfBuffer.lookup_transform('world','ef',rospy.Time())

            print("Traslazione rispetto a X: {}".format(transform.transform.translation.x))
            print("Traslazione rispetto a Y: {}".format(transform.transform.translation.y))
            print("Rotazione (Quaternione) rispetto a X: {}".format(transform.transform.rotation.x))

            #Nuova Trasformata per effettuare il broadcast
            transform_stamped = TransformStamped()

            # Header:
            # - stamp
            # - frame_id (Stringa)
            # - child_frame_id (Stringa)

            #stamp
            transform_stamped.header.stamp = rospy.Time.now()
            #frame_id
            transform_stamped.header.frame_id = "world"
            #child_frame_id
            transform_stamped.child_frame_id = "ef_world"

            #translation
            transform_stamped.transform.translation.x = transform.transform.translation.x
            transform_stamped.transform.translation.y = transform.transform.translation.y
            transform_stamped.transform.translation.z = transform.transform.translation.z


            #rotation (Quaternion x,y,z,w)
            transform_stamped.transform.rotation.x = transform.transform.rotation.x
            #rotation y
            transform_stamped.transform.rotation.y = transform.transform.rotation.y
            #rotation z
            transform_stamped.transform.rotation.z = transform.transform.rotation.z
            #rotation w
            transform_stamped.transform.rotation.w = transform.transform.rotation.w

            #broadcast della nuova trasfromata
            broadcaster.sendTransform(transform_stamped)

        except (tf2_ros.LookupException, tf2_ros.ConnectivityException, tf2_ros.ExtrapolationException) as e:
            print("Error obtaining transform: ", e)
        rate.sleep()


if __name__ == '__main__':
    get_transform()
    rospy.spin()
```

Il file python si può eseguire in due modo diversi lanciandolo dall'IDE o facendo un run da qualsiasi editor di codice o sfruttando la funzionalità del middleware ROS. Il robot operating system dà la possibilità di eseguire un qualsiasi script presente in un package ROS attraverso il comando **rosrun**. Prima di tutto il file *.py* va posizionato all'interno della cartella **src** del pacchetto e poi bisogna eseguire i seguenti comandi:

```
$ cd ~/catkin_ws
$ source devel/setup.bash
```

Dopo aver eseguito i comandi sopra descritti, si può effettuare il run dello script in questo modo:

```
$ rosrun <nome_package> <nome file.py>
```

F Reference

[1] https://wiki.ros.org/noetic/Installation/Ubuntu.

[2] https://wiki.ros.org/catkin/Tutorials/create_a_workspace.

[3] https://wiki.ros.org/ROS/Tutorials/CreatingPackage.

[4] https://wiki.ros.org/urdf/XML/link.

[5] https://wiki.ros.org/urdf/XML/joint.

[6] https://wiki.ros.org/tf2/Tutorials/Writing%20a%20tf2%20broadcaster%20%28Python%29.