



› Instituto Infnet

## **ASSESSMENT**

**Elizabeth Maria de Carvalho**

**Matrícula: 11681797771**



Instituto Infnet

## **Assessment**

Trabalho apresentado como recurso avaliativo da disciplina de Fundamentos de Sistemas Robóticos com ROS ministrada pelo M.e. Miguel Fensterseifer.

Rio de Janeiro  
2020

SUMÁRIO

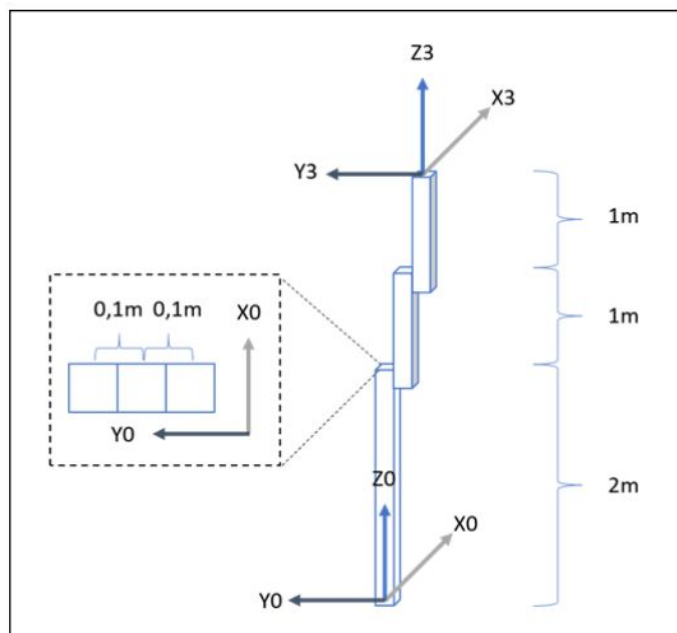
Objetivo	3
Tarefa Única	4

## OBJETIVO

Reforçar os conceitos aprendidos na disciplina de “Fundamentos de Sistemas Robóticos com ROS”.

## TAREFA ÚNICA

1. Explique com suas próprias palavras como funciona a medição de distância com ultrassom.
2. Explique com suas próprias palavras o que é e para que serve uma matriz de transformação.
3. Explique com suas próprias palavras para que servem os modelos cinemáticos direto e reverso de um braço robótico.
4. Descreva o modelo cinemático do braço robótico abaixo utilizando os parâmetros de Denavit-Hartenberg.



Articulação	$\theta$	$d$	$a$	$\alpha$
1				
2				

5. Explique com suas próprias palavras em que consiste o planejamento de trajetória de um braço robótico.
6. Implementar um nó de comunicação do tipo publisher para envio de mensagens do tipo `std_msgs/Float64` em ROS-1. O código fonte pode ser desenvolvido na

linguagem de programação de sua preferência e deverá ser anexado à etapa correspondente do Moodle.

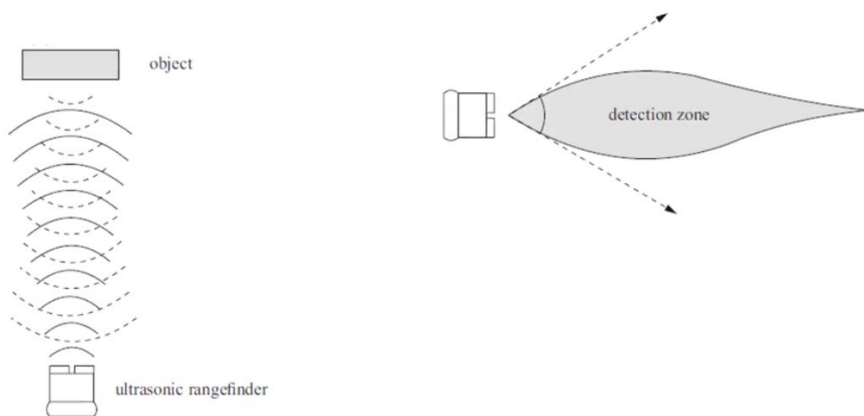
7. Implementar um nó de comunicação do tipo subscriber para recebimento de mensagens do tipo `std_msgs/Float64` em ROS-2. Este nó deverá exibir as mensagens publicadas pelo nó anterior. O código fonte pode ser desenvolvido na linguagem de programação de sua preferência e deverá ser anexado à etapa correspondente do Moodle.
8. Exibir a comunicação entre ambos os nós através da captura de tela dos terminais utilizados para inicializar o mestre ROS-1, inicializar o publisher em ROS-1 e inicializar o subscriber em ROS-2. A imagem capturada deverá ser anexada à etapa correspondente do Moodle.
9. Exibir a comunicação entre ambos os nós através do `rqt_graph`. A imagem do grafo deverá ser capturada e anexada à etapa correspondente do Moodle.
10. Explique com suas próprias palavras as diferenças entre os arquivos URDF utilizados para visualização de um mesmo robô no Rviz e no Gazebo.
11. Elabore um arquivo URDF para descrever um braço robótico do tipo RR com as dimensões apresentadas na figura da questão 4. O XML deverá ser anexado à etapa correspondente do Moodle.
12. Elabore um arquivo de inicialização para simulação do robô acima no Gazebo com controle das articulações enviados a partir da interface `rqt`. O XML deverá ser anexado à etapa correspondente do Moodle, assim como a captura da tela da simulação.
13. Implementar uma máquina de estados para que o robô acima execute um ciclo com 4 poses predeterminadas ( $\neg$ ,  $\perp$ ,  $\bot$ ,  $\top$ ). O código fonte pode ser desenvolvido na linguagem de programação de sua preferência e deverá ser anexado à etapa correspondente do Moodle.
14. Exibir a sequência anterior através do SMACH. A imagem do grafo deverá ser capturada e anexada à etapa correspondente do Moodle.

15. Alterar o arquivo URDF da questão 11 para incluir um sensor do tipo ultrassom do extremo do elo superior do robô acima. A leitura do sensor deverá ser exibida em um terminal através da função `rostopic echo`.
16. Exibir a comunicação entre o nó do sensor e o nó de visualização da sua leitura através do `rqt_graph`. A imagem do grafo deverá ser capturada e anexada à etapa correspondente do Moodle.

----- RESPOSTAS -----

1)

O sensor de ultrassom calcula a distância emitindo um som de frequência bastante elevada (inaudível ao ser humano, daí ultrassom) e analisando o tempo que o som leva para alcançar o objeto e ser refletido de volta para o sensor. Logo, baseado na velocidade de propagação do ar é possível calcular a distância com base no tempo  $[(\text{tempo de ida e volta})/2 * \text{velocidade de propagação do som no ar}]$ .



2)

São as matrizes de rotação e translação que servem, por exemplo, para saber o quanto um robô se deslocou em relação a um ponto e o quanto o sistema de coordenadas girou em relação ao mesmo ponto. De forma prática, serve para calcular a posição de uma articulação e em relação a outra ou a uma origem, podendo assim estimar sua pose.

Acontece também, por exemplo, ao fazer mapeamento e localização simultâneo com um LIDAR.

3)

O modelo cinemático direto serve para estimar a posição e orientação da garra levando em conta a posição de cada articulação. Já o modelo de cinemática reversa analisa o quanto que tem que mover cada articulação para que o braço fique em uma determinada posição.

4)

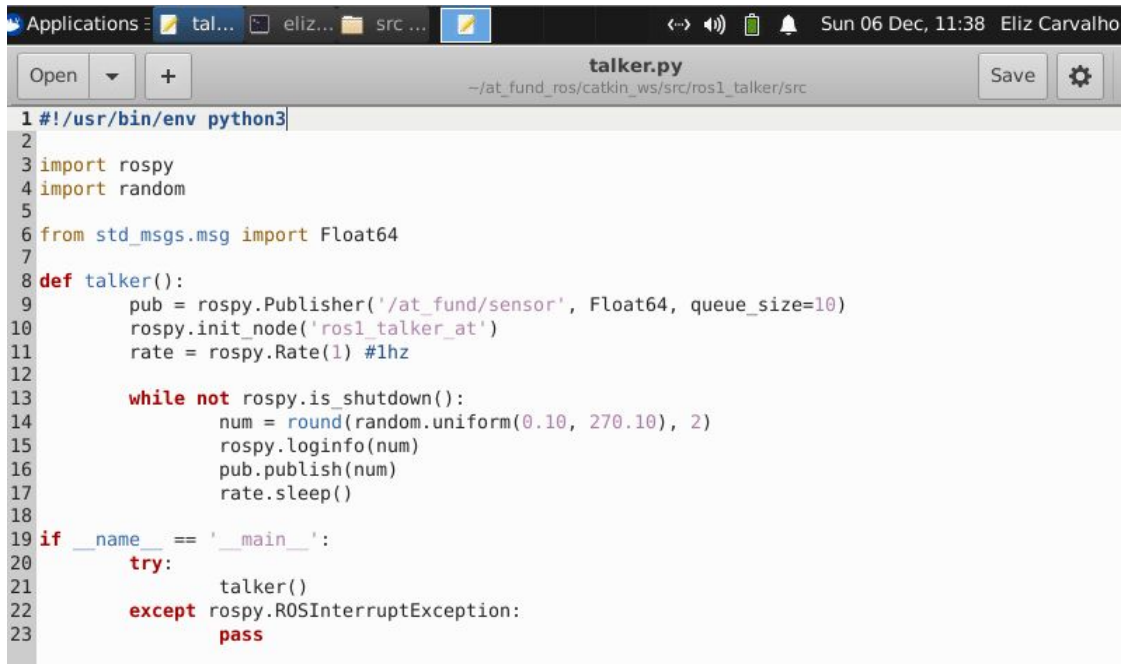
Articulação	$\Theta$	$d$	$a$	$\alpha$
Articulação 0 [0-1]	90°	2m	0.1m	90°
Articulação 1 [1-2]	Q1	0.1m	0.95m	0
Articulação 2 [2-3]	-90°	1m	0.1	-90°

5)

Consiste em planejar adequadamente a posição de cada articulação (pose atual e nova pose), além da trajetória a ser realizada, para que o braço robótico altere sua trajetória e se movimente de forma síncrona para atingir a pose nova, sem esbarrar em algum obstáculo.

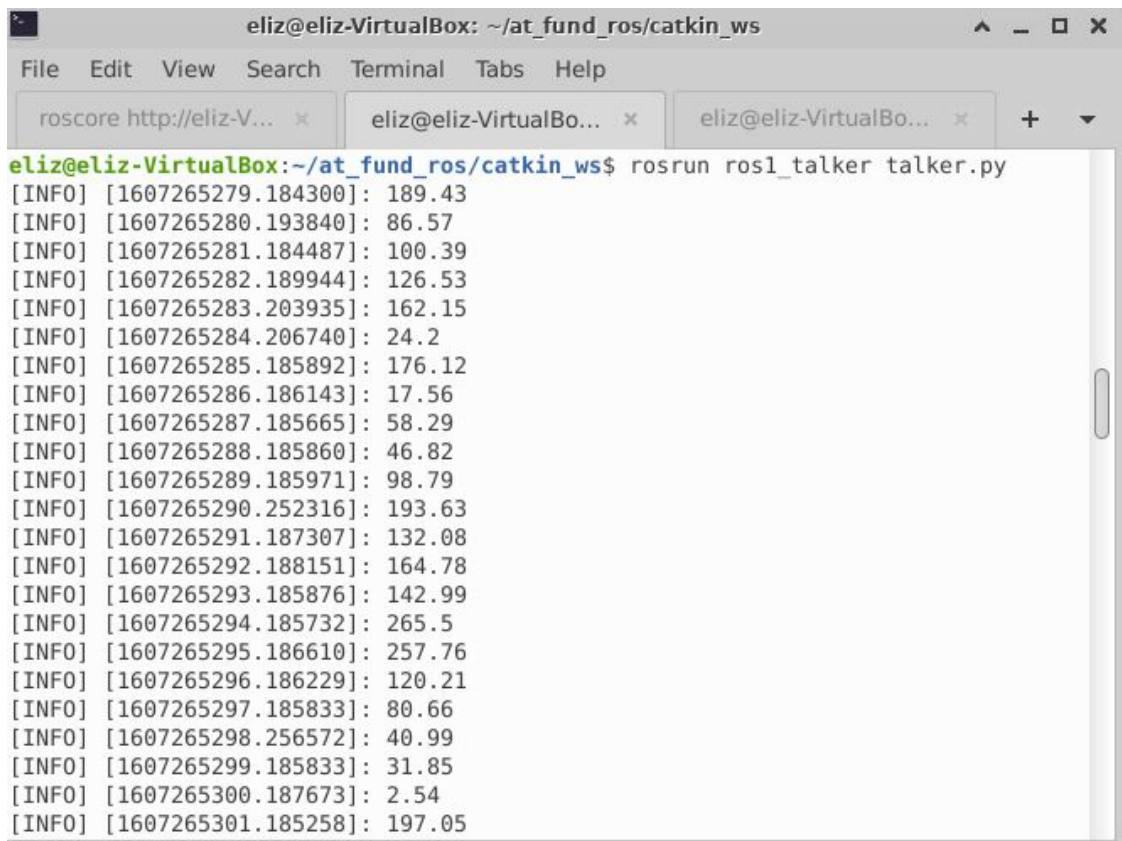


6)



The screenshot shows a code editor window titled 'talker.py' with the file path '~/\_at\_fund\_ros/catkin\_ws/src/ros1\_talker/src'. The code is a Python script that uses the rospy library to create a ROS node named 'ros1\_talker\_at'. It publishes random Float64 values to the 'sensor' topic at a rate of 1 Hz. The code includes a try-except block to handle ROSInterruptException.

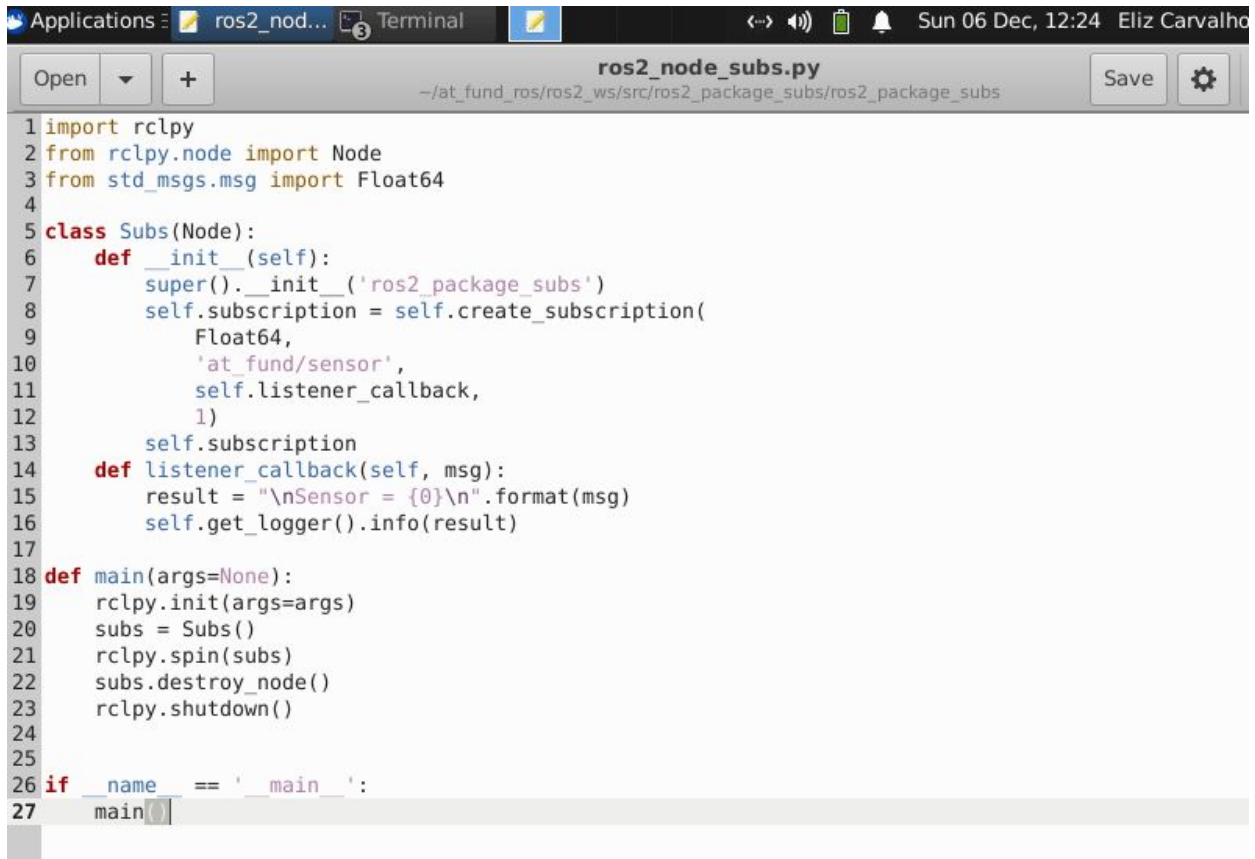
```
1#!/usr/bin/env python3
2
3import rospy
4import random
5
6from std_msgs.msg import Float64
7
8def talker():
9    pub = rospy.Publisher('/at_fund/sensor', Float64, queue_size=10)
10    rospy.init_node('ros1_talker_at')
11    rate = rospy.Rate(1) #1hz
12
13    while not rospy.is_shutdown():
14        num = round(random.uniform(0.10, 270.10), 2)
15        rospy.loginfo(num)
16        pub.publish(num)
17        rate.sleep()
18
19if __name__ == '__main__':
20    try:
21        talker()
22    except rospy.ROSInterruptException:
23        pass
```



The screenshot shows a terminal window titled 'eliz@eliz-VirtualBox: ~/\_at\_fund\_ros/catkin\_ws'. The terminal displays the command 'roscore http://eliz-V...' and 'roslaunch ros1\_talker talker.py'. The output shows a series of INFO messages, each containing a timestamp and a random Float64 value.

```
eliz@eliz-VirtualBox:~/_at_fund_ros/catkin_ws$ roscore http://eliz-V...
[INFO] [1607265279.184300]: 189.43
[INFO] [1607265280.193840]: 86.57
[INFO] [1607265281.184487]: 100.39
[INFO] [1607265282.189944]: 126.53
[INFO] [1607265283.203935]: 162.15
[INFO] [1607265284.206740]: 24.2
[INFO] [1607265285.185892]: 176.12
[INFO] [1607265286.186143]: 17.56
[INFO] [1607265287.185665]: 58.29
[INFO] [1607265288.185860]: 46.82
[INFO] [1607265289.185971]: 98.79
[INFO] [1607265290.252316]: 193.63
[INFO] [1607265291.187307]: 132.08
[INFO] [1607265292.188151]: 164.78
[INFO] [1607265293.185876]: 142.99
[INFO] [1607265294.185732]: 265.5
[INFO] [1607265295.186610]: 257.76
[INFO] [1607265296.186229]: 120.21
[INFO] [1607265297.185833]: 80.66
[INFO] [1607265298.256572]: 40.99
[INFO] [1607265299.185833]: 31.85
[INFO] [1607265300.187673]: 2.54
[INFO] [1607265301.185258]: 197.05
```

7)



The screenshot shows a code editor window titled "ros2\_node\_subs.py" with the file path "/at\_fund\_ros/ros2\_ws/src/ros2\_package\_subs/ros2\_package\_subs". The code is as follows:

```
1 import rclpy
2 from rclpy.node import Node
3 from std_msgs.msg import Float64
4
5 class Subs(Node):
6     def __init__(self):
7         super().__init__('ros2_package_subs')
8         self.subscription = self.create_subscription(
9             Float64,
10            'at_fund/sensor',
11            self.listener_callback,
12            1)
13         self.subscription
14     def listener_callback(self, msg):
15         result = "\nSensor = {0}\n".format(msg)
16         self.get_logger().info(result)
17
18 def main(args=None):
19     rclpy.init(args=args)
20     subs = Subs()
21     rclpy.spin(subs)
22     subs.destroy_node()
23     rclpy.shutdown()
24
25
26 if __name__ == '__main__':
27     main()
```



The screenshot shows a terminal window titled "eliz@eliz-VirtualBox: ~/at\_fund\_ros/ros2\_ws". The terminal output displays five log messages from the "ros2\_package\_subs" node, each showing a sensor value received from the "at\_fund/sensor" topic.

```
[INFO] [1607269176.194767294] [ros2_package_subs]:
Sensor = std_msgs.msg.Float64(data=212.69)

[INFO] [1607269177.174765305] [ros2_package_subs]:
Sensor = std_msgs.msg.Float64(data=203.78)

[INFO] [1607269178.194451874] [ros2_package_subs]:
Sensor = std_msgs.msg.Float64(data=21.42)

[INFO] [1607269179.196584529] [ros2_package_subs]:
Sensor = std_msgs.msg.Float64(data=47.75)

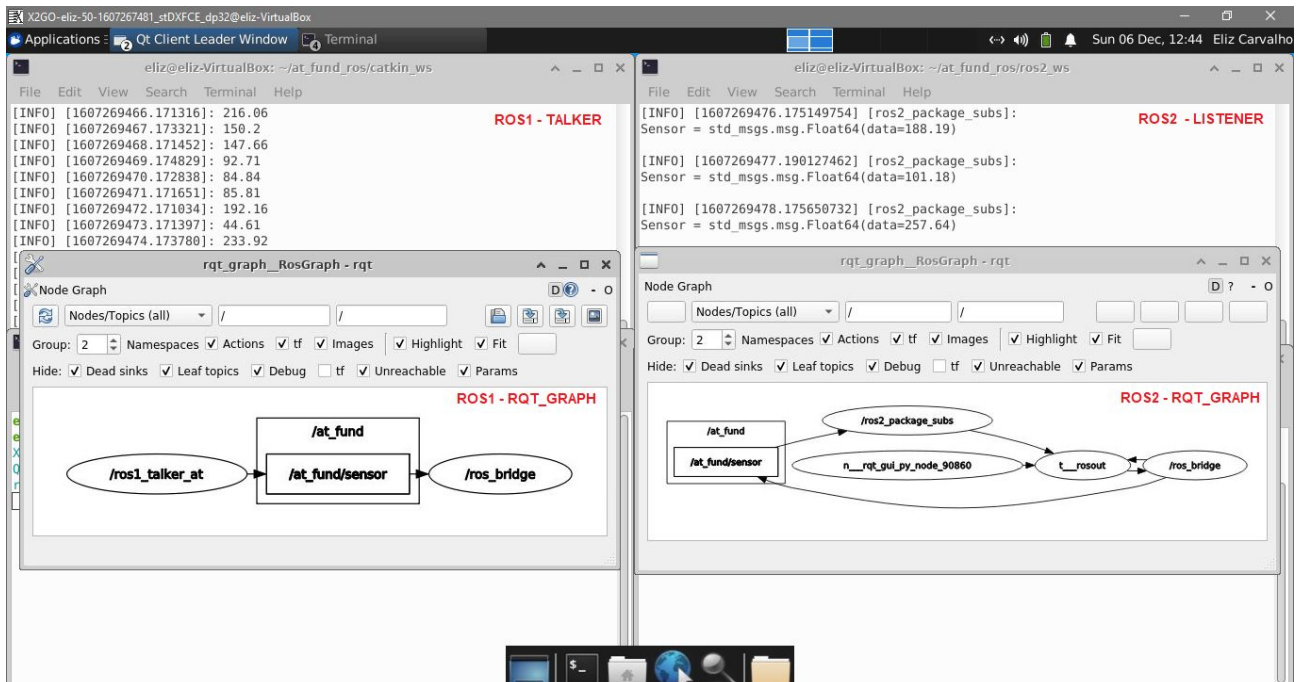
[INFO] [1607269180.207576720] [ros2_package_subs]:
Sensor = std_msgs.msg.Float64(data=96.16)
```

8)

The screenshot shows four terminal windows in a VirtualBox environment:

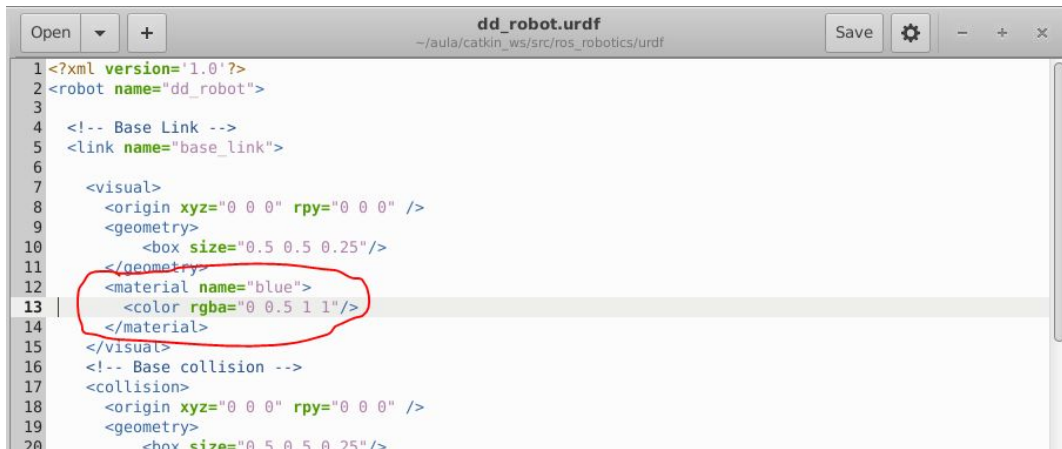
- ROS1 - TALKER:** Displays a list of sensor data points, including timestamps and values for topics like `/at_fund/sensor`.
- ROS2 - LISTENER:** Shows the reception of messages from the ROS2 package `std_msgs.msg.Float64` with data values like 42.38, 155.14, etc.
- BRIDGET:** Contains the command-line execution of `colcon build` and `ros2 run ros1_bridge dynamic_bridge`, along with a summary of the bridge configuration.
- ROSCORE:** Displays the status of the ROS2 master, including parameters like `/rostdistro: noetic` and `/rosversion: 1.15.9`, and the list of nodes running on the master.

9)



10)

O Gazebo não sabe interpretar no XML a sessão “material” (utilizada, por exemplo, para dar cor ao elo).



```
1 <?xml version='1.0'?>
2 <robot name="dd_robot">
3
4 <!-- Base Link -->
5 <link name="base_link">
6
7   <visual>
8     <origin xyz="0 0 0" rpy="0 0 0" />
9     <geometry>
10       <box size="0.5 0.5 0.25"/>
11     </geometry>
12     <material name="blue">
13       <color rgba="0 0.5 1 1"/>
14     </material>
15   </visual>
16 <!-- Base collision -->
17 <collision>
18   <origin xyz="0 0 0" rpy="0 0 0" />
19   <geometry>
20     <box size="0.5 0.5 0.25"/>
```

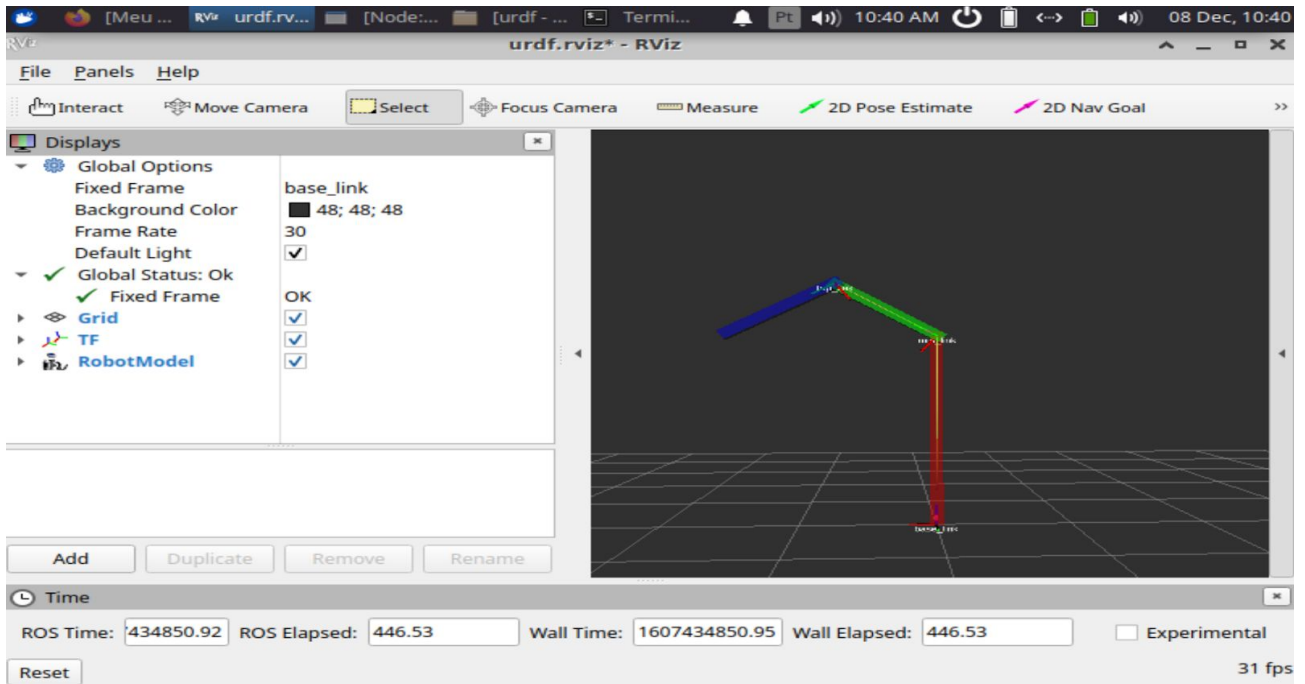
No caso do Gazebo, faz-se necessário o uso da sessão “gazebo” dando como referência o nome do elo que sofrerá o efeito desejado.



```
54 <!-- Right Wheel -->
55 <link name="right_wheel">
56   <visual>
57     <origin xyz="0 0 0" rpy="1.570795 0 0" />
58     <geometry>
59       <cylinder length="0.1" radius="0.2" />
60     </geometry>
61     <material name="black">
62       <color rgba="0.05 0.05 0.05 1"/>
63     </material>
64   </visual>
65 <!-- Right Wheel collision -->
66 <collision>
67   <origin xyz="0 0 0" rpy="1.570795 0 0" />
68   <geometry>
69     <cylinder length="0.1" radius="0.2" />
70   </geometry>
71 </collision>
72 <inertial>
73   <mass value="0.5"/>
74   <inertia ixx="0.01" ixy="0.0" ixz="0.0" iyy="0.005" iyz="0.0" izz="0.005"/>
75 </inertial>
76 </link>
77
78 <gazebo reference="right_wheel">
79   <material>Gazebo/Black</material>
80 </gazebo>
81
82
```

Além disso, se o arquivo não tiver as descrições de <visual> e <collision>, o Gazebo vai interpretar os objetos como transparentes.

11)



Arquivo "rrbot-q11.xacro":

```
<?xml version="1.0"?>
<!-- Revolute-Revolute Manipulator -->
<robot name="rrbot" xmlns:xacro="http://www.ros.org/wiki/xacro">

  <!-- Constants for robot dimensions -->
  <xacro:property name="width" value="0.1" />    <!-- Beams are square
in length and width -->
  <xacro:property name="height1" value="2" />    <!-- Link 1 -->
  <xacro:property name="height2" value="1" />    <!-- Link 2 -->
  <xacro:property name="height3" value="1" />    <!-- Link 3 -->
  <xacro:property name="axle_offset" value="0.05" /> <!-- Space between
joint and end of beam -->
  <xacro:property name="damp" value="0.7" />    <!-- damping
coefficient -->

  <!-- Default Inertial -->
  <xacro:macro name="default_inertial" params="z_value i_value mass">
    <inertial>
```



```

    <origin xyz="0 0 ${z_value}" rpy="0 0 0"/>
    <mass value="${mass}" />
    <inertia ixx="${i_value}" ixy="0.0" ixz="0.0"
            iyy="${i_value}" iyz="0.0"
            izz="${i_value}" />
  </inertial>
</xacro:macro>

<!-- Import Rviz colors -->
<xacro:include filename="$(find ros_robotics)/urdf/materials.xacro"
/>

<!-- Base Link -->
<link name="base_link">
  <visual>
    <origin xyz="0 0 ${height1/2}" rpy="0 0 0"/>
    <geometry>
<box size="${width} ${width} ${height1}"/>
    </geometry>
    <material name="red"/>
  </visual>

  <collision>
    <origin xyz="0 0 ${height1/2}" rpy="0 0 0"/>
    <geometry>
<box size="${width} ${width} ${height1}"/>
    </geometry>
  </collision>

  <xacro:default_inertial z_value="${height1/2}" i_value="1.0"
mass="1"/>
</link>

<!-- Joint between Base Link and Middle Link -->
<joint name="joint_base_mid" type="revolute">
  <parent link="base_link"/>
  <child link="mid_link"/>
  <origin xyz="0 ${width} ${height1 - axle_offset}" rpy="1.57 0

```

```

0"/>
    <axis xyz="0 0 1"/>
    <dynamics damping="${damp}"/>
    <limit effort="100.0" velocity="0.5" lower="-3.14" upper="3.14"
/>
</joint>

<!-- Middle Link -->
<link name="mid_link">
    <visual>
        <origin xyz="0 ${height2/2 - axle_offset} 0" rpy="1.57 0 0"/>
        <geometry>
<box size="${width} ${width} ${height2}"/>
        </geometry>
        <material name="green"/>
    </visual>

    <collision>
        <origin xyz="0 ${height2/2 - axle_offset} 0" rpy="1.57 0 0"/>
        <geometry>
<box size="${width} ${width} ${height2}"/>
        </geometry>
    </collision>

    <xacro:default_inertial z_value="${height2/2 - axle_offset}"
i_value="1.0" mass="1"/>
</link>

<!-- Joint between Middle Link and Top Link -->
<joint name="joint_mid_top" type="revolute">
    <parent link="mid_link"/>
    <child link="top_link"/>
    <origin xyz="0 ${height2 - axle_offset*2} -${width}" rpy="0 0
0"/>
    <axis xyz="0 0 1"/>
    <dynamics damping="${damp}"/>
    <limit effort="100.0" velocity="0.5" lower="-3.14" upper="3.14"
/>

```

```

</joint>

<!-- Top Link -->
<link name="top_link">
  <visual>
    <origin xyz="0 ${height3/2 - axle_offset} 0" rpy="1.57 0 0"/>
    <geometry>
<box size="${width} ${width} ${height3}"/>
    </geometry>
    <material name="blue"/>
  </visual>

  <collision>
    <origin xyz="0 ${height3/2 - axle_offset} 0" rpy="1.57 0 0"/>
    <geometry>
<box size="${width} ${width} ${height3}"/>
    </geometry>
  </collision>

  <xacro:default_inertial z_value="${height3/2 - axle_offset}"
i_value="1.0" mass="1"/>
</link>

</robot>

```

Arquivo "rrbot-q11.urdf" (gerado através do comando "roslaunch xacro xacro rrbot-q11.xacro > rrbot-q11.urdf"):

```

<?xml version="1.0" ?>
<!--
=====
----- -->
<!-- |      This document was autogenerated by xacro from
rrbot-q11.xacro      | -->
<!-- |      EDITING THIS FILE BY HAND IS NOT RECOMMENDED
      | -->
<!--

```



```

=====
===== -->
<!-- Revolute-Revolute Manipulator -->
<robot name="rrbot">
  <!-- damping coefficient -->
  <material name="black">
    <color rgba="0.0 0.0 0.0 1.0"/>
  </material>
  <material name="blue">
    <color rgba="0.0 0.0 0.8 1.0"/>
  </material>
  <material name="green">
    <color rgba="0.0 1.0 0.0 1.0"/>
  </material>
  <material name="grey">
    <color rgba="0.2 0.2 0.2 1.0"/>
  </material>
  <material name="orange">
    <color rgba="1.0 0.4235294117647059 0.0392156862745098 1.0"/>
  </material>
  <material name="brown">
    <color rgba="0.8705882352941177 0.8117647058823529
0.7647058823529411 1.0"/>
  </material>
  <material name="red">
    <color rgba="0.8 0.0 0.0 1.0"/>
  </material>
  <material name="white">
    <color rgba="1.0 1.0 1.0 1.0"/>
  </material>
  <!-- Base Link -->
  <link name="base_link">
    <visual>
      <origin rpy="0 0 0" xyz="0 0 1.0"/>
      <geometry>
        <box size="0.1 0.1 2"/>
      </geometry>
      <material name="red"/>
    </visual>
    <collision>

```

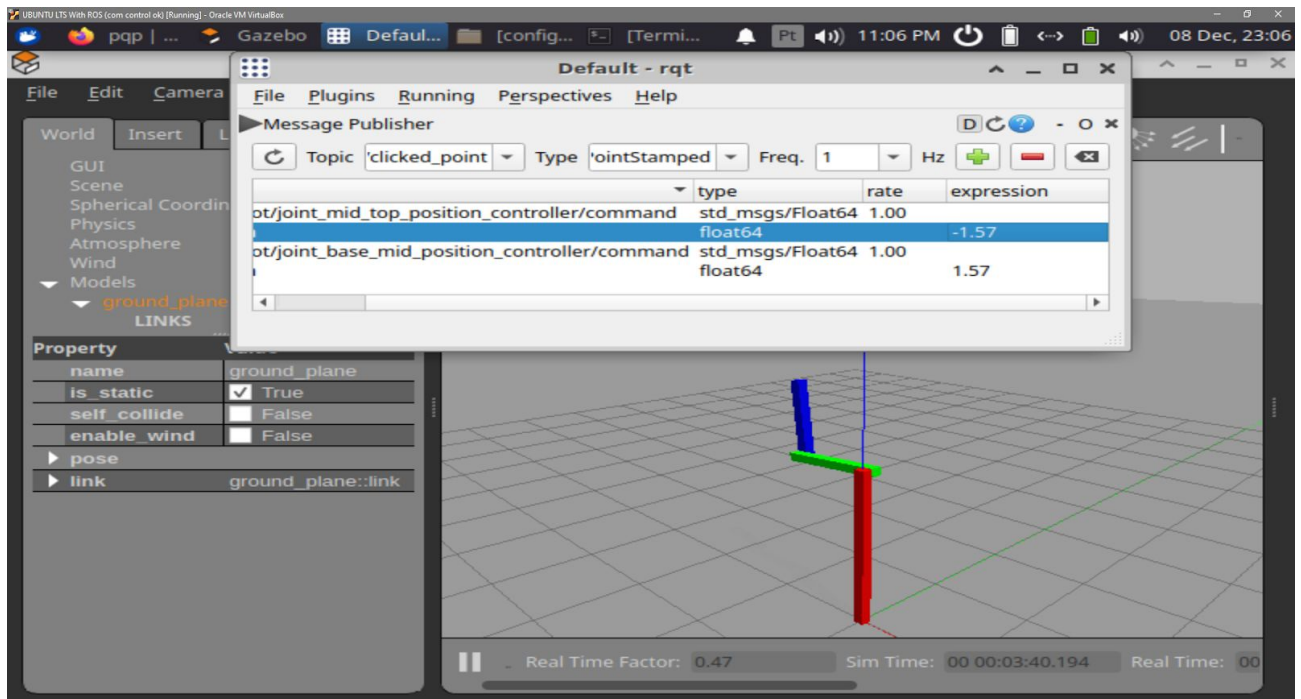
```

    <origin rpy="0 0 0" xyz="0 0 1.0"/>
    <geometry>
    <box size="0.1 0.1 2"/>
    </geometry>
    </collision>
    <inertial>
    <origin rpy="0 0 0" xyz="0 0 1.0"/>
    <mass value="1"/>
    <inertia ixx="1.0" ixy="0.0" ixz="0.0" iyy="1.0" iyz="0.0"
izz="1.0"/>
    </inertial>
  </link>
  <!-- Joint between Base Link and Middle Link -->
  <joint name="joint_base_mid" type="revolute">
    <parent link="base_link"/>
    <child link="mid_link"/>
    <origin rpy="1.57 0 0" xyz="0 0.1 1.95"/>
    <axis xyz="0 0 1"/>
    <dynamics damping="0.7"/>
    <limit effort="100.0" lower="-3.14" upper="3.14" velocity="0.5"/>
  </joint>
  <!-- Middle Link -->
  <link name="mid_link">
    <visual>
    <origin rpy="1.57 0 0" xyz="0 0.45 0"/>
    <geometry>
    <box size="0.1 0.1 1"/>
    </geometry>
    <material name="green"/>
    </visual>
    <collision>
    <origin rpy="1.57 0 0" xyz="0 0.45 0"/>
    <geometry>
    <box size="0.1 0.1 1"/>
    </geometry>
    </collision>
    <inertial>
    <origin rpy="0 0 0" xyz="0 0 0.45"/>
    <mass value="1"/>
    <inertia ixx="1.0" ixy="0.0" ixz="0.0" iyy="1.0" iyz="0.0"

```

```
    izz="1.0"/>
    </inertial>
  </link>
  <!-- Joint between Middle Link and Top Link -->
  <joint name="joint_mid_top" type="revolute">
    <parent link="mid_link"/>
    <child link="top_link"/>
    <origin rpy="0 0 0" xyz="0 0.9 -0.1"/>
    <axis xyz="0 0 1"/>
    <dynamics damping="0.7"/>
    <limit effort="100.0" lower="-3.14" upper="3.14" velocity="0.5"/>
  </joint>
  <!-- Top Link -->
  <link name="top_link">
    <visual>
      <origin rpy="1.57 0 0" xyz="0 0.45 0"/>
      <geometry>
        <box size="0.1 0.1 1"/>
      </geometry>
      <material name="blue"/>
    </visual>
    <collision>
      <origin rpy="1.57 0 0" xyz="0 0.45 0"/>
      <geometry>
        <box size="0.1 0.1 1"/>
      </geometry>
    </collision>
    <inertial>
      <origin rpy="0 0 0" xyz="0 0 0.45"/>
      <mass value="1"/>
      <inertia ixx="1.0" ixy="0.0" ixz="0.0" iyy="1.0" iyz="0.0"
    izz="1.0"/>
    </inertial>
  </link>
</robot>
```

12)



13)

Arquivo at\_machine\_state.py:

```
#!/usr/bin/env python3
import rospy
from smach import State, StateMachine
from time import sleep
import smach_ros
from std_msgs.msg import Float64

class A(State):
    def __init__(self):
        State.__init__(self, outcomes=['1','0'], input_keys=['input'],
        output_keys=[''])

    def execute(self, userdata):
        pub_base_mid =
        rospy.Publisher('/rrbot/joint_base_mid_position_controller/command',
        Float64, queue_size=10)
```

```

        pub_base_mid.publish(-1.57)
        pub_mid_top =
rospy.Publisher('/rrbot/joint_mid_top_position_controller/command',
Float64, queue_size=10)
        pub_mid_top.publish(-1.57)
        sleep(20)
        if userdata.input == 1:
            return '1'
        else:
            return '0'

class B(State):
    def __init__(self):
        State.__init__(self, outcomes=['1','0'], input_keys=['input'],
output_keys=[''])
    def execute(self, userdata):
        pub_base_mid =
rospy.Publisher('/rrbot/joint_base_mid_position_controller/command',
Float64, queue_size=10)
        pub_base_mid.publish(-1.57)
        pub_mid_top =
rospy.Publisher('/rrbot/joint_mid_top_position_controller/command',
Float64)
        pub_mid_top.publish(1.57)
        sleep(20)
        if userdata.input == 1:
            return '1'
        else:
            return '0'

class C(State):
    def __init__(self):
        State.__init__(self, outcomes=['1','0'], input_keys=['input'],
output_keys=[''])
    def execute(self, userdata):
        pub_base_mid =
rospy.Publisher('/rrbot/joint_base_mid_position_controller/command',
Float64)
        pub_base_mid.publish(1.57)
        pub_mid_top =

```

```

rospy.Publisher('/rrbot/joint_mid_top_position_controller/command',
Float64)
    pub_mid_top.publish(-1.57)
    sleep(20)
    if userdata.input == 1:
        return '1'
    else:
        return '0'

class D(State):
    def __init__(self):
        State.__init__(self, outcomes=['1','0'], input_keys=['input'],
output_keys=[''])
    def execute(self, userdata):
        pub_base_mid =
rospy.Publisher('/rrbot/joint_base_mid_position_controller/command',
Float64)
        pub_base_mid.publish(1.57)
        pub_mid_top =
rospy.Publisher('/rrbot/joint_mid_top_position_controller/command',
Float64)
        pub_mid_top.publish(1.57)
        sleep(20)
        if userdata.input == 1:
            return '1'
        else:
            return '0'

if __name__ == '__main__':

    rospy.init_node('test_fsm', anonymous=True)
    sm = StateMachine(outcomes=['success'])
    sm.userdata.sm_input = 1
    with sm:
        StateMachine.add('A', A(), transitions={'1':'B','0':'D'},
remapping={'input':'sm_input','output':'input'})
        StateMachine.add('B', B(), transitions={'1':'C','0':'A'},
remapping={'input':'sm_input','output':'input'})
        StateMachine.add('C', C(), transitions={'1':'D','0':'B'},
remapping={'input':'sm_input','output':'input'})

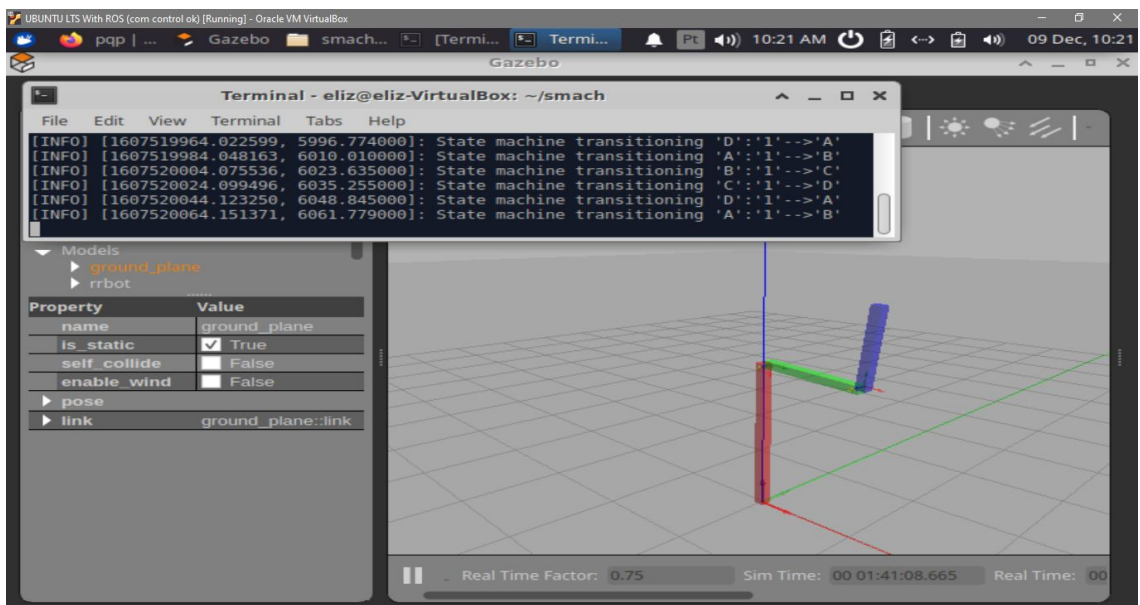
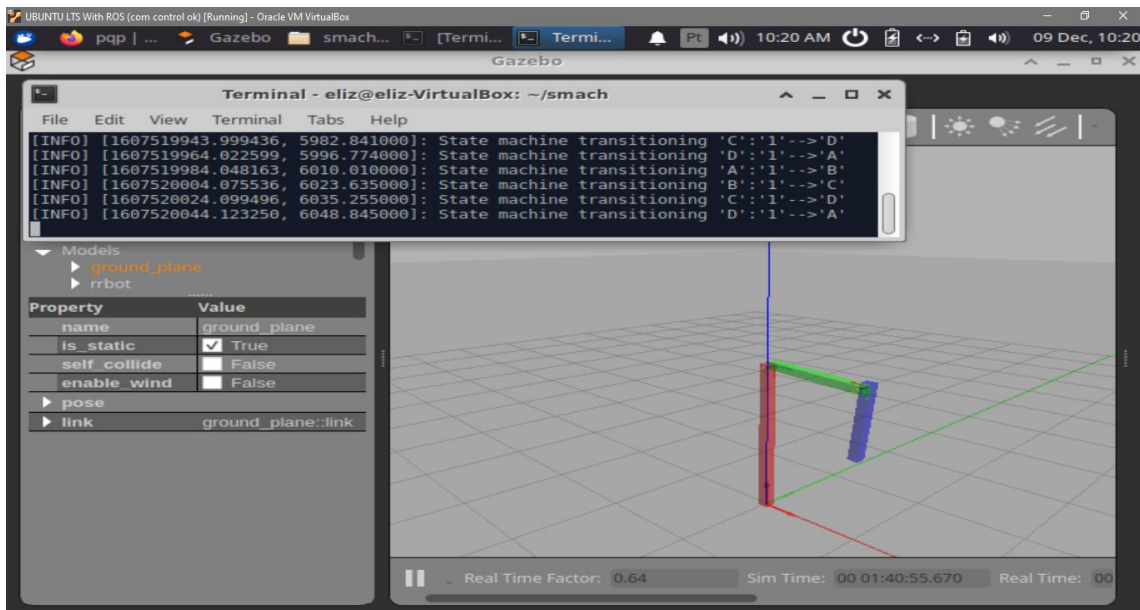
```

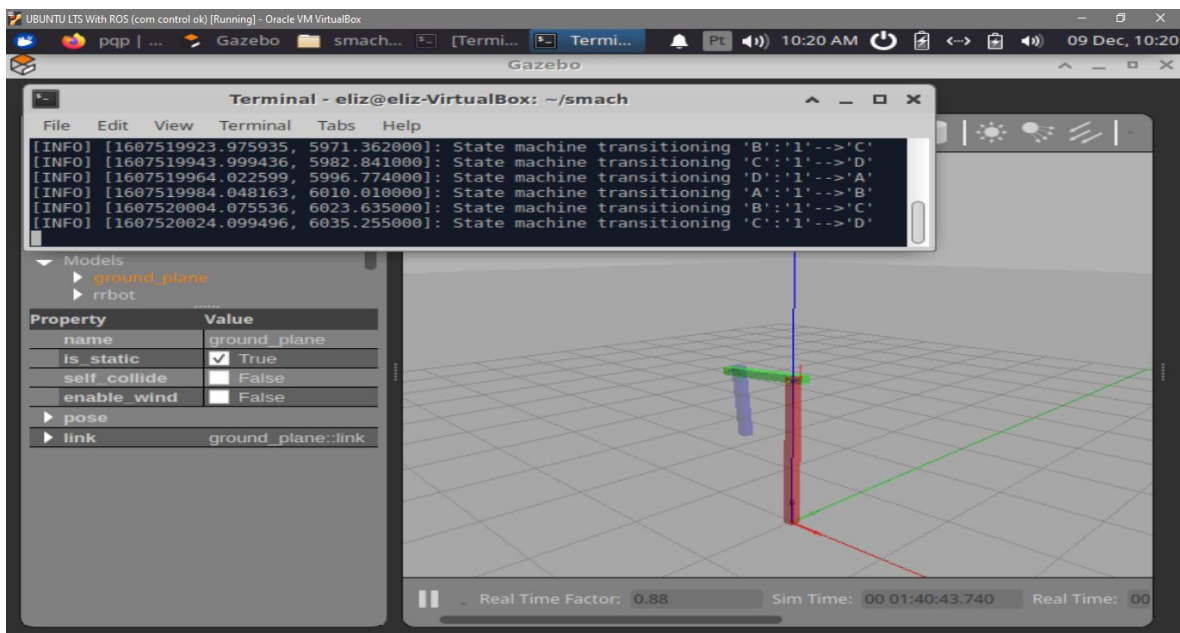
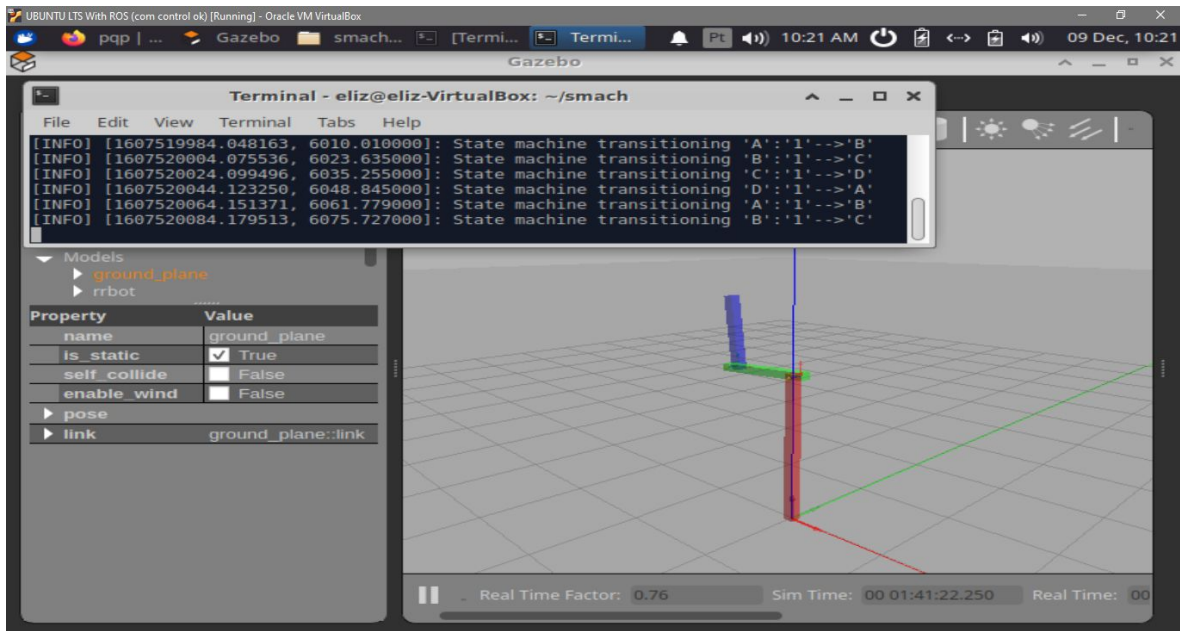
```

    StateMachine.add('D', D(), transitions={'1':'A','0':'C'},
remapping={'input':'sm_input','output':'input'})
    #sis = smach_ros.IntrospectionServer('server_name', sm, '/SM_ROOT')
    #sis.start()

    sm.execute()
    rospy.spin()
    #sis.stop()

```

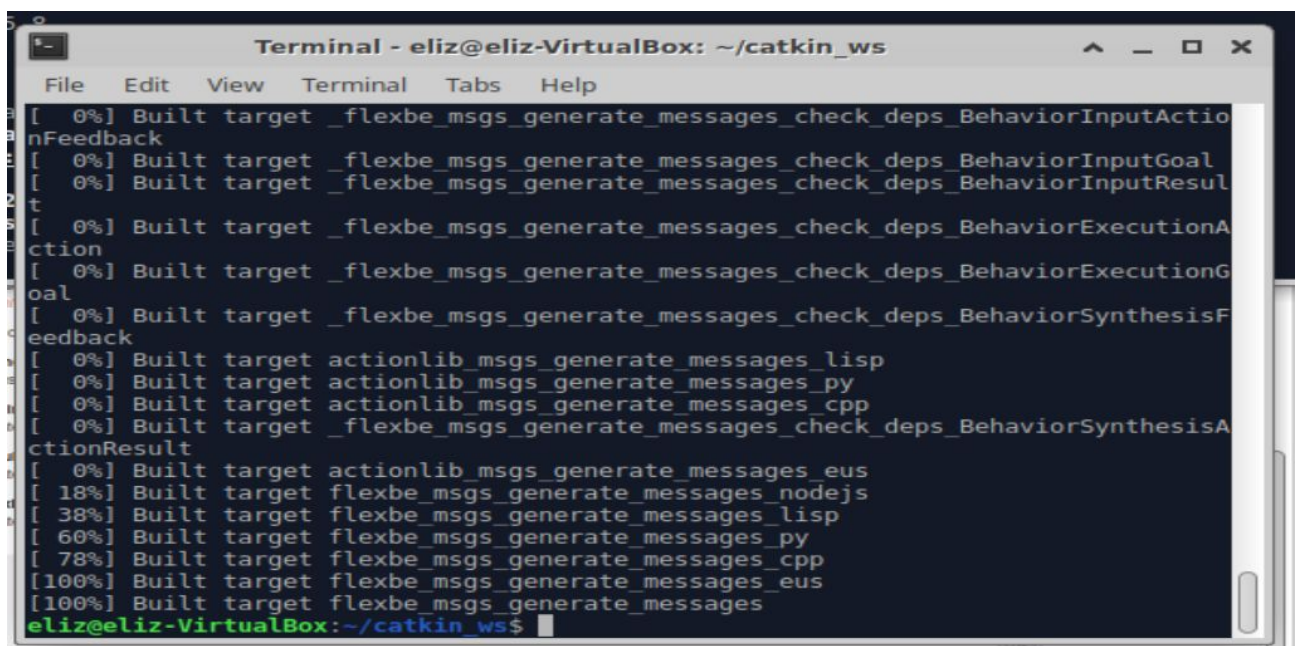
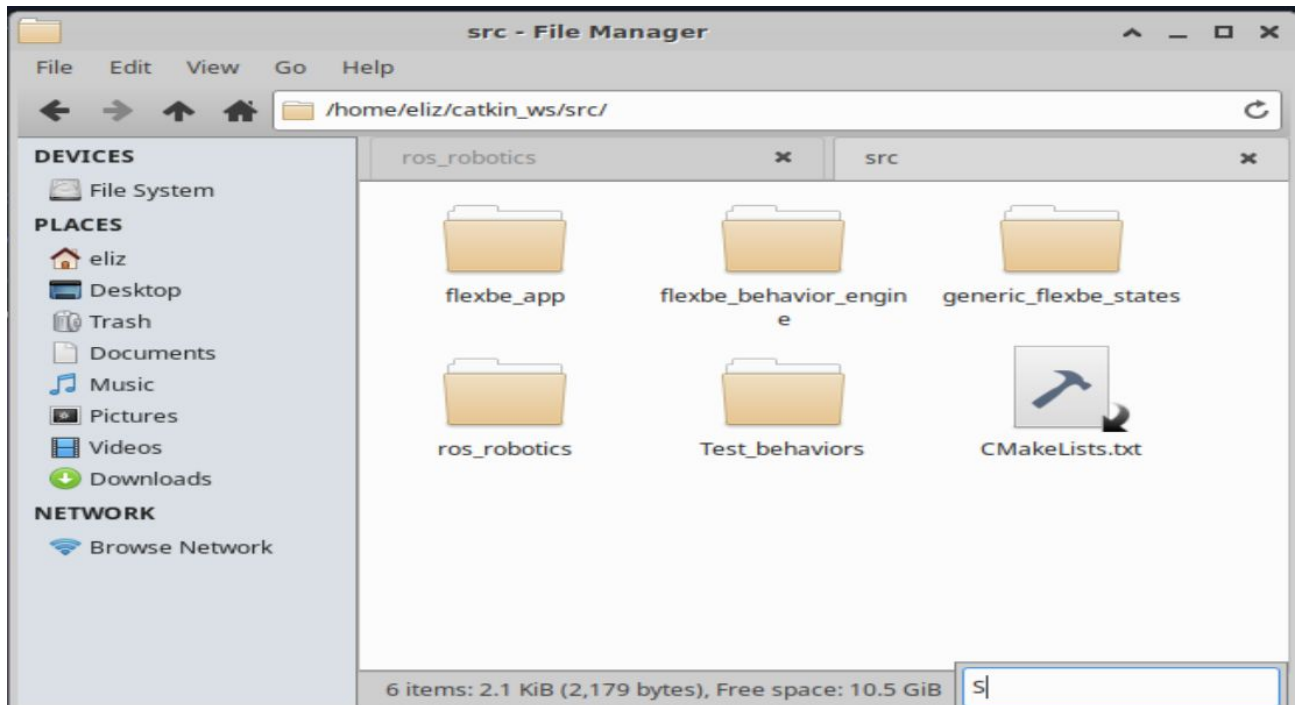




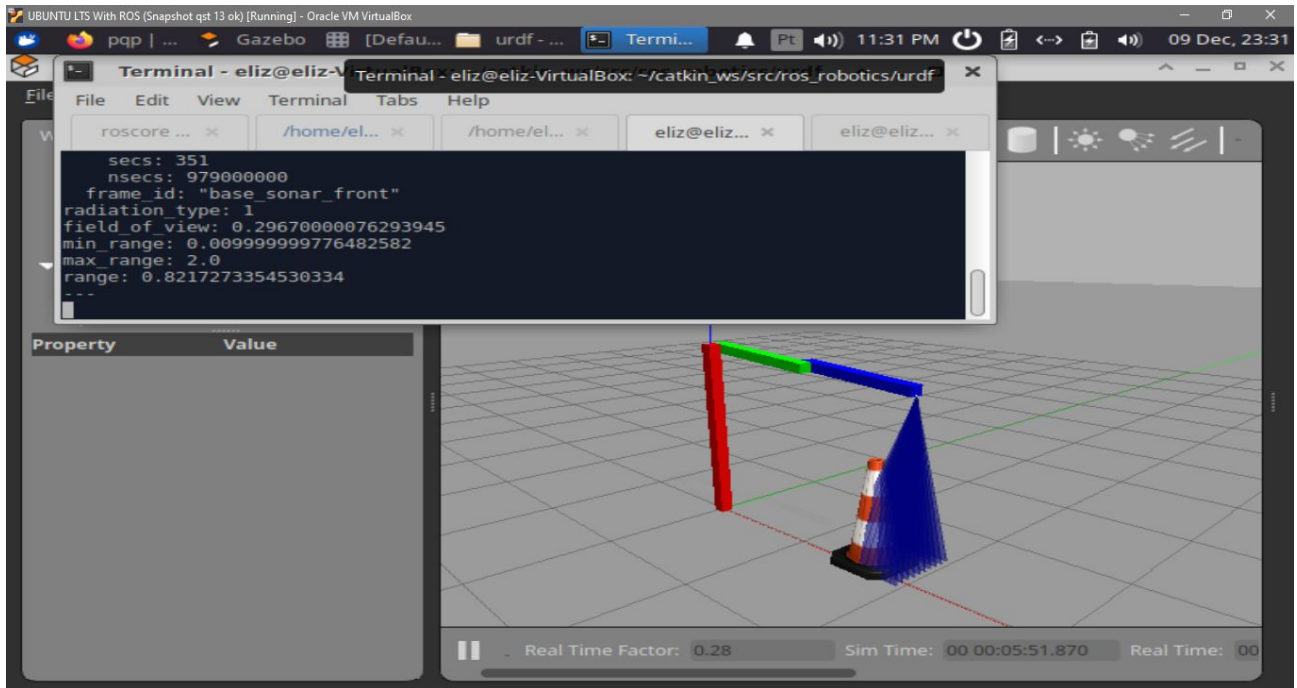
14)

Foi realizado o clone dos repositórios do FlexBe, bem como criação do projeto e build da solução. Peço desculpas mas infelizmente não gerenciei meu tempo bem o suficiente a ponto de investir mais tempo na pesquisa para resolução desta questão.





15)



Trecho adicionado ao arquivo rrobot-q15.xacro:

```
<joint name="sonar_front_joint" type="fixed">
  <axis xyz="0 1 0" />
  <origin rpy="0 0 0" xyz="0.045 0.955 0" />
  <parent link="top_link"/>
  <child link="base_sonar_front"/>
</joint>
<link name="base_sonar_front">
  <collision>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <box size="0.01 0.01 0.01"/>
    </geometry>
  </collision>
  <visual>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <box size="0.01 0.01 0.01"/>
    </geometry>
  </visual>
</link>
```

```

    <inertial>
      <mass value="1e-5" />
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <inertia ixx="1e-6" ixy="0" ixz="0" iyy="1e-6" iyz="0" izz="1e-6"
/>
    </inertial>
  </link>

  <gazebo reference="base_sonar_front">
    <sensor type="ray" name="TeraRanger">
      <pose>0 0 0 0 0 0</pose>
      <visualize>true</visualize>
      <update_rate>50</update_rate>
      <ray>
        <scan>
          <horizontal>
            <samples>10</samples>
            <resolution>1</resolution>
            <min_angle>-0.14835</min_angle>
            <max_angle>0.14835</max_angle>
          </horizontal>
          <vertical>
            <samples>10</samples>
            <resolution>1</resolution>
            <min_angle>-0.14835</min_angle>
            <max_angle>0.14835</max_angle>
          </vertical>
        </scan>
        <range>
          <min>0.01</min>
          <max>2</max>
          <resolution>0.02</resolution>
        </range>
      </ray>
      <plugin filename="libgazebo_ros_range.so"
name="gazebo_ros_range">
        <gaussianNoise>0.005</gaussianNoise>
        <alwaysOn>true</alwaysOn>
        <updateRate>50</updateRate>
        <topicName>sensor/sonar_front</topicName>

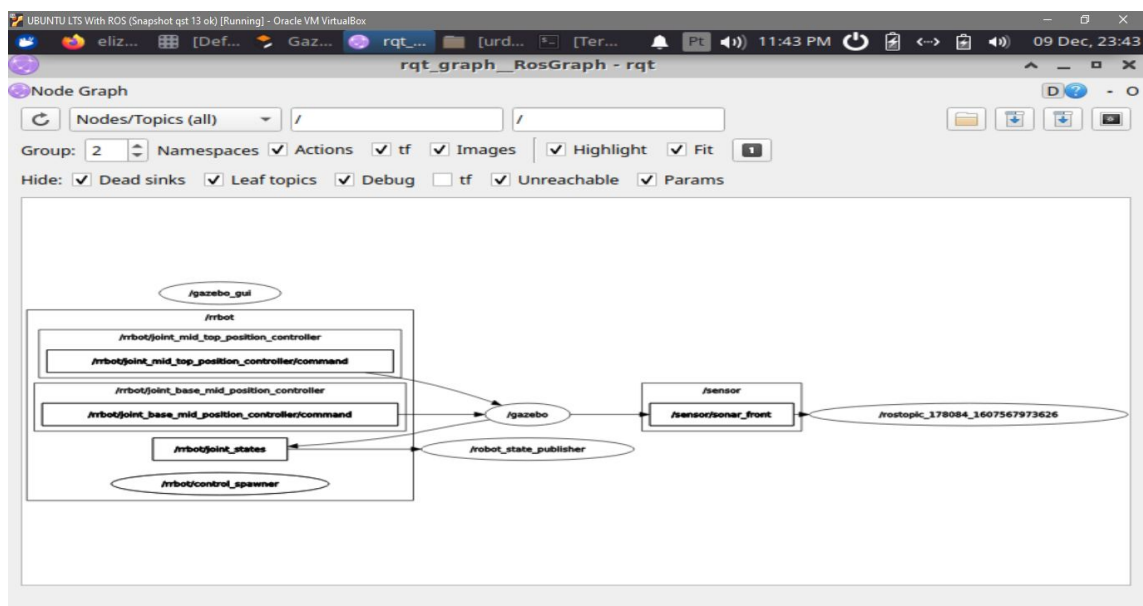
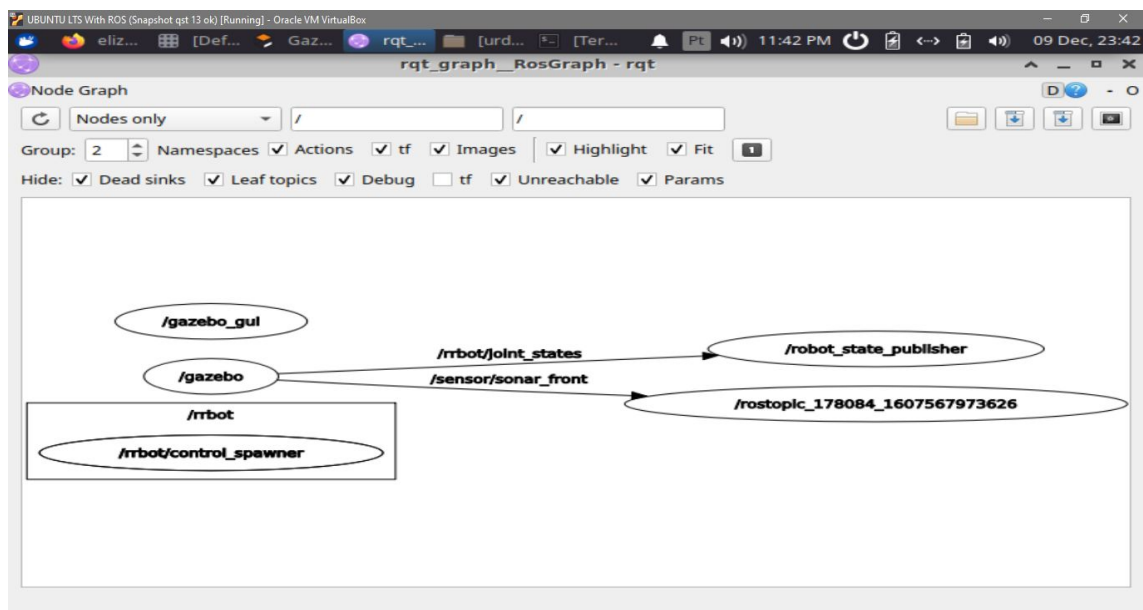
```

```

    <frameName>base_sonar_front</frameName>
    <radiation>INFRARED</radiation>
    <fov>0.2967</fov>
  </plugin>
</sensor>
</gazebo>

```

16)



----- FIM -----

