Uso de Controladores no ROS 2

Manipuladores

Walter Fetter Lages

fetter@ece.ufrgs.br

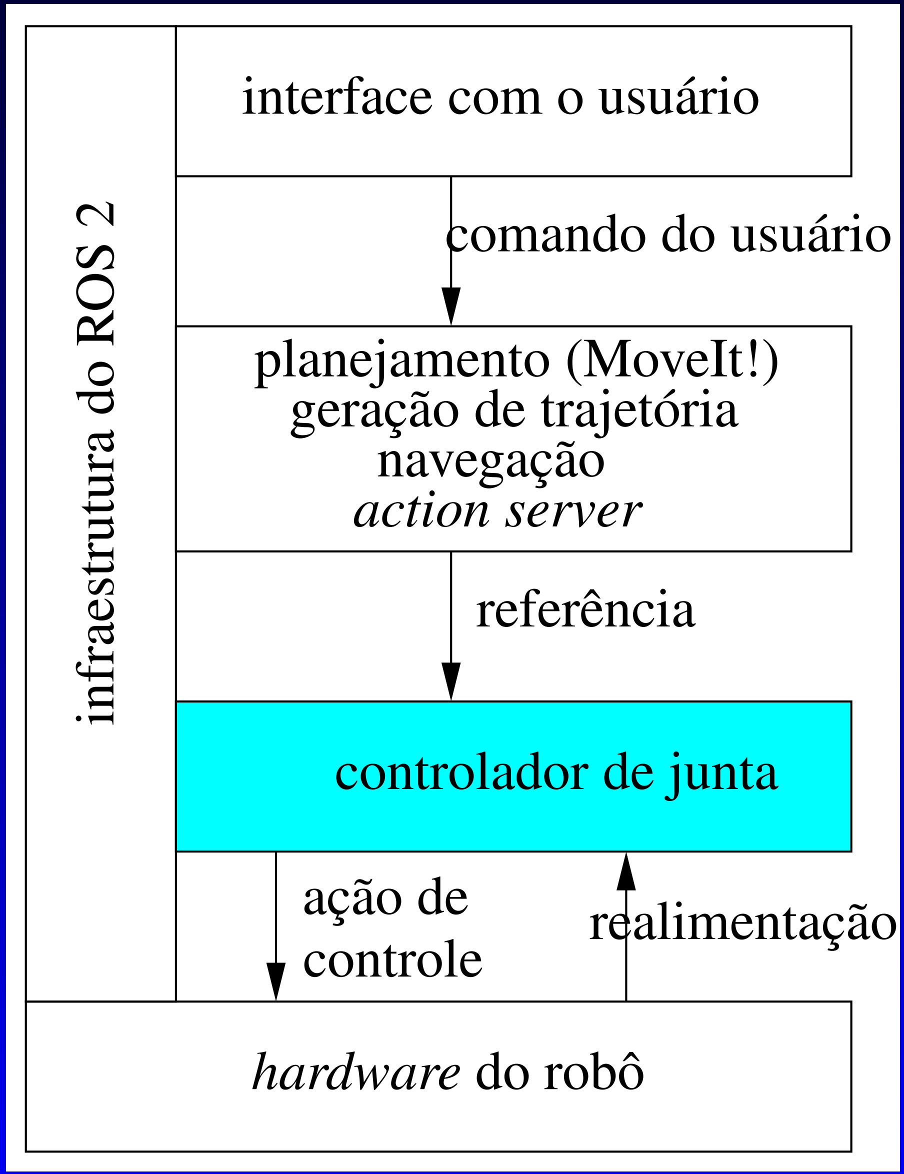
Universidade Federal do Rio Grande do Sul
Escola de Engenharia
Departamento de Sistemas Elétricos de Automação e Energia

ENG10052 Laboratório de Robótica

Introdução

- Existem várias formas de implementar controladores no ROS 2
  - A maioria delas não suporta operação em tempo-real
  - O *framework* `ros2_control` oferece ferramentas *real-time safe* para implementação de controladores
- Aqui será criado um pacote para usar os controladores já implementados no ROS 2
  - Simulador Gazebo Classic
  - Simulador Gazebo Ignition
  - Robô Quanser 2DSJFE

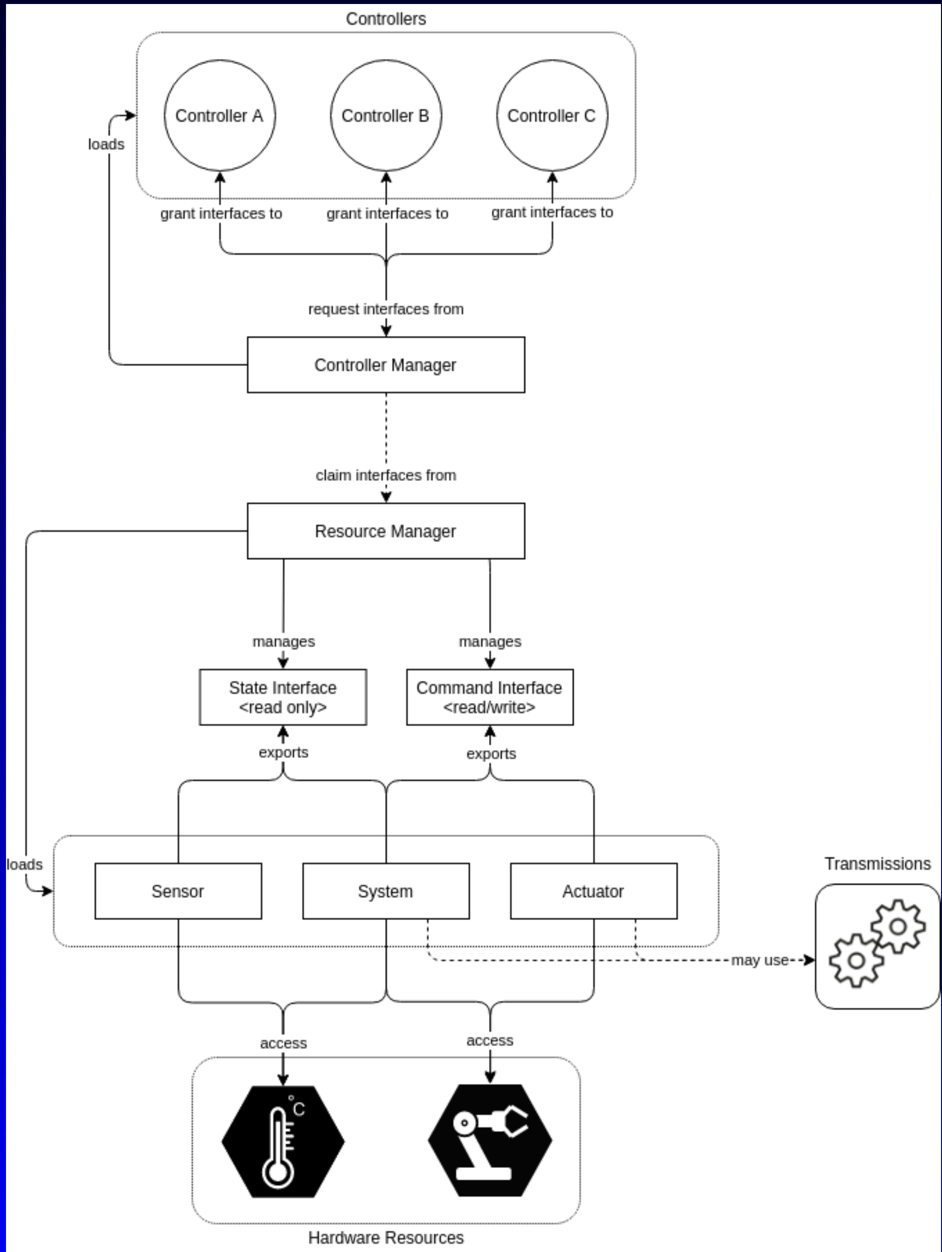
Controle de Juntas



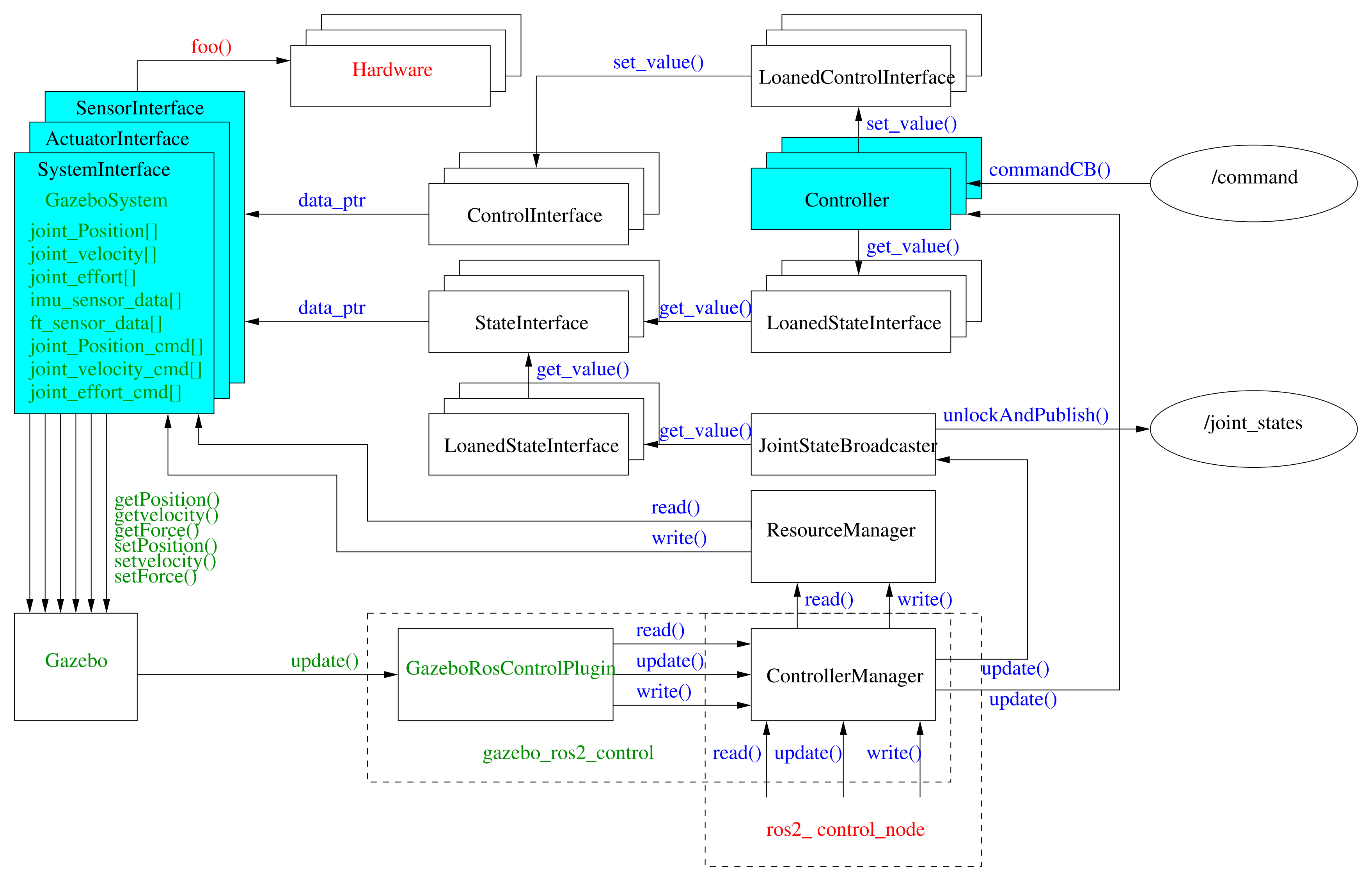
Framework `ros2_control`

- A definição de controlador do `ros_control` não é a clássica de Sistemas de Controle
  - Módulo que é carregado pelo gerenciador de controladores
  - Não necessariamente implementa um controlador
    - Alguns "controladores" implementam *drivers* para sensores
    - O exemplo clássico é o `joint_state_broadcaster`
  - Vários controladores implementados no `ros_controllers` ainda não foram implementadosno `ros2_controllers`

Framework `ros2_control`



Laço de Tempo Real



Plugin para o `ros2_control`

- Incluir na descrição URDF

```
<ros2_control name="Q2dSystem" type="system">
    <xacro:if value="${hardware == 'gazebo'}">
        <!-- Gazebo Classic -->
        <hardware>
            <plugin>gazebo_ros2_control/GazeboSystem<
/plugin>
        </hardware>
    </xacro:if>
    <xacro:if value="${hardware == 'ignition'}">
        <!-- Gazebo Ignition -->
        <hardware>
            <plugin>ign_ros2_control/IgnitionSystem</
plugin>
        </hardware>
    </xacro:if>
```

Plugin para o ros2_control

```
<xacro:if value="${hardware == 'real_robot'}">
    <!-- Actual Hardware -->
    <hardware>
        <plugin>q2d_hardware/Q2dSystemHardware</plugin>
    </hardware>
</xacro:if>

<joint name="shoulder_active_joint">
    <command_interface name="effort">
        <param name="min">-27.94</param>
        <param name="max">27.94</param>
    </command_interface>
    <state_interface name="position"/>
    <state_interface name="velocity"/>
    <state_interface name="effort"/>
</joint>
```

Plugin para o `ros2_control`

```
    <joint name="elbow_active_joint">
      <command_interface name="effort">
        <param name="min">-13.62</param>
        <param name="max">13.62</param>
      </command_interface>
      <state_interface name="position"/>
      <state_interface name="velocity"/>
      <state_interface name="effort"/>
    </joint>
  </ros2_control>
```

- Faz a interface do `ros2_control` com o *hardware*
  - O *hardware* pode ser um simulador

Plugin para o Gazebo

- Incluir na descrição URDF
- Possibilita que o Gazebo interprete os tags <ros2_control> e carregue o `controller_manager` e os controladores

```
<gazebo>
    <xacro:if value="${hardware == 'gazebo'}">
        <!-- Gazebo Classic -->
        <plugin filename="libgazebo_ros2_control.so" name="gazebo_ros2_control">
            <robot_param>robot_description</robot_param>
            <robot_param_node>robot_state_publisher</robot_param_node>
            <parameters>$(find q2d_description)/config/controller_manager.yaml</parameters>
        </plugin>
    </xacro:if>
```

Plugin para o Gazebo

```
<xacro:if value="${hardware == 'ignition'}">
    <!-- Gazebo Ignition -->
    <plugin filename="libign_ros2_control-system.so" name="ign_ros2_control::IgnitionROS2ControlPlugin">
        <robot_param>robot_description</robot_param>
        <robot_param_node>robot_state_publisher</robot_param_node>
        <parameters>$(find q2d_description)/config/controller_manager.yaml</parameters>
    </plugin>
</xacro:if>
</gazebo>
```

Tipo dos Controladores

- Usualmente o *namespace* indica a grandeza de saída do controlador
- Usualmente a classe indica a grandeza que é controlada ou a lei de controle
- No ROS 2 há uma tendência a fugir dessa semântica
- A sintaxe as vezes é de `path`, outras é de C++
- `namespace_do_controlador/ClasseDoControlador`
- `namespace_do_controlador::ClasseDoControlador`
- `effort_controllers/JointPositionController`
- `effort_controllers/JointVelocityController`
- `velocity_controllers::JointPositionController`

Estrutura do Pacote

```
q2d_bringup/
├─CMakeLists.txt
├─config/
│  ├─computed_torque.yaml
│  ├─group_bypass.yaml
│  ├─pid_plus_gravity.yaml
│  └─⋮
├─launch/
│  ├─computed_torque.launch.xml
│  ├─gazebo.launch.xml
│  ├─group_bypass.launch.xml
│  ├─pid_plus_gravity.launch.xml
│  └─⋮
├─package.xml
└─scripts/
   ├─group_torque_step.sh
   └─⋮
```

Dependências

- ros2_control
  - controller_interface
  - controller_manager
  - controller_manager_msgs
  - hardware_interface
  - joint_limits
  - ros2_control_test_assets
  - ros2controlcli
  - transmission_interface
- control_msgs
- control_toolbox
- realtime_tools

---

sudo apt install ros–$ROS_DISTRO–ros2–control

Dependências

- ros2_controllers
  - position_controllers
    - joint_position_controller
    - joint_group_position_controller
  - forward_command_controller
    - forward_command_controller
  - joint_state_broadcaster
    - joint_state_broadcaster

Dependências

- ros2_controllers
  - effort_controllers
    - joint_effort_controller
    - joint_position_controller
    - joint_velocity_controller
    - joint_group_effort_controller
    - joint_group_position_controller
  - velocity_controllers
    - joint_position_controller
    - joint_velocity_controller
    - joint_group_velocity_controller

---

sudo apt install ros–$ROS_DISTRO–ros2–controllers

Dependências

- `xacro`
- `test_interface_files`
- `test_msgs`
- Não incluídas na variante desktop do Humble

---

sudo apt install ros–$ROS_DISTRO–xacro

sudo apt install ros–$ROS_DISTRO–test–interface–files

sudo apt install ros–$ROS_DISTRO–test–msgs

Dependências

- `kdl_parser`
- `robot_state_publisher`
- `orocos_kdl`
- Já instalados na variante robot do ROS 2

Dependências - Gazebo Classic

- `gazebo`
- `gazebo_ros_pkgs`
  - `gazebo_dev`
  - `gazebo_msgs`
  - `gazebo_ros`
  - `gazebo_plugins`
- `gazebo_ros2_control`

```
sudo apt install gazebo
sudo apt install ros-$ROS_DISTRO-gazebo-ros-pkgs
sudo apt install ros-$ROS_DISTRO-gazebo-ros2-control
```

Criação do Pacote

```
cd ~/colcon_ws/src
ros2 pkg create --build-type=ament_cmake --dependencies
    effort_controllers joint_state_broadcaster q2d_bringup
```

`package.xml`

- Editar o arquivo `q2d_bringup/package.xml`
  - Descrição
  - Mantenedor
  - Licença
  - Dependências

CMakeLists.txt

- Editar `CMakeLists.txt` para descomentar e ajustar as *tags*:

```
install(PROGRAMS
  scripts/ijc_step.sh
  scripts/ijc_square.py
  scripts/group_torque_step.sh
  scripts/joint_trajectory_step.sh
  DESTINATION lib/${PROJECT_NAME})

install(DIRECTORY config launch
  DESTINATION share/${PROJECT_NAME})
```

Reconfigurar o Ambiente

---

cd ~/colcon_ws

colcon build −−symlink−install

source ~/colcon_ws/install/setup.bash

---

config/controller_manager.yaml


```yaml
controller_manager:
    ros__parameters:
        update_rate: 1000
        use_sim_time: true
```

config/group_bypass.yaml

```
group_controller:
    ros__parameters:
        joints:
            – shoulder_active_joint
            – elbow_active_joint
```

group_bypass.launch.xml

```
<launch>
  <arg name="config" default="$(find-pkg-share q2d_bringup)/
     config/group_bypass.yaml"/>

  <node name="group_controller_spawner" pkg="controller_manager
     " exec="spawner"
    args="-t effort_controllers/JointGroupEffortController -p $(var
     config) group_controller"/>

  <node name="joint_state_broadcaster_spawner" pkg="
     controller_manager" exec="spawner"
    args="-t joint_state_broadcaster/JointStateBroadcaster
     joint_state_broadcaster"/>
</launch>
```

gazebo.launch.xml

- q2d/q2d_bringup/launch/gazebo.launch.xml

```
<launch>
  <arg name="pause" default="true"/>
  <arg name="gui" default="true"/>
  <arg name="use_sim_time" default="true"/>
  <arg name="ignition" default="false"/>

  <arg name="controller" default="pid"/>
  <arg name="config" default="$(find-pkg-share q2d_bringup)/
    config/$(var controller).yaml"/>
```

gazebo.launch.xml

```
<include unless="$(var ignition)" file="$(find-pkg-share
   q2d_description)/launch/gazebo.launch.xml" >
  <arg name="pause" value="$(var pause)"/>
  <arg name="gui" value="$(var gui)"/>
  <arg name="use_sim_time" value="$(var use_sim_time)"/>
</include>

<include if="$(var ignition)" file="$(find-pkg-share
   q2d_description)/launch/ignition.launch.xml" >
  <arg name="pause" value="$(var pause)"/>
  <arg name="gui" value="$(var gui)"/>
  <arg name="use_sim_time" value="$(var use_sim_time)"/>
</include>
```

gazebo.launch.xml

```
  <include file="$(find-pkg-share q2d_bringup)/launch/$(var
     controller).launch.xml" >
    <arg name="config" value="$(var config)"/>
    <arg name="use_sim_time" value="$(var use_sim_time)"/>
  </include>
</launch>
```

Simulação no Gazebo

```
ros2 launch q2d/q2d_bringup gazebo.launch.xml controller:=
    group_bypass
```

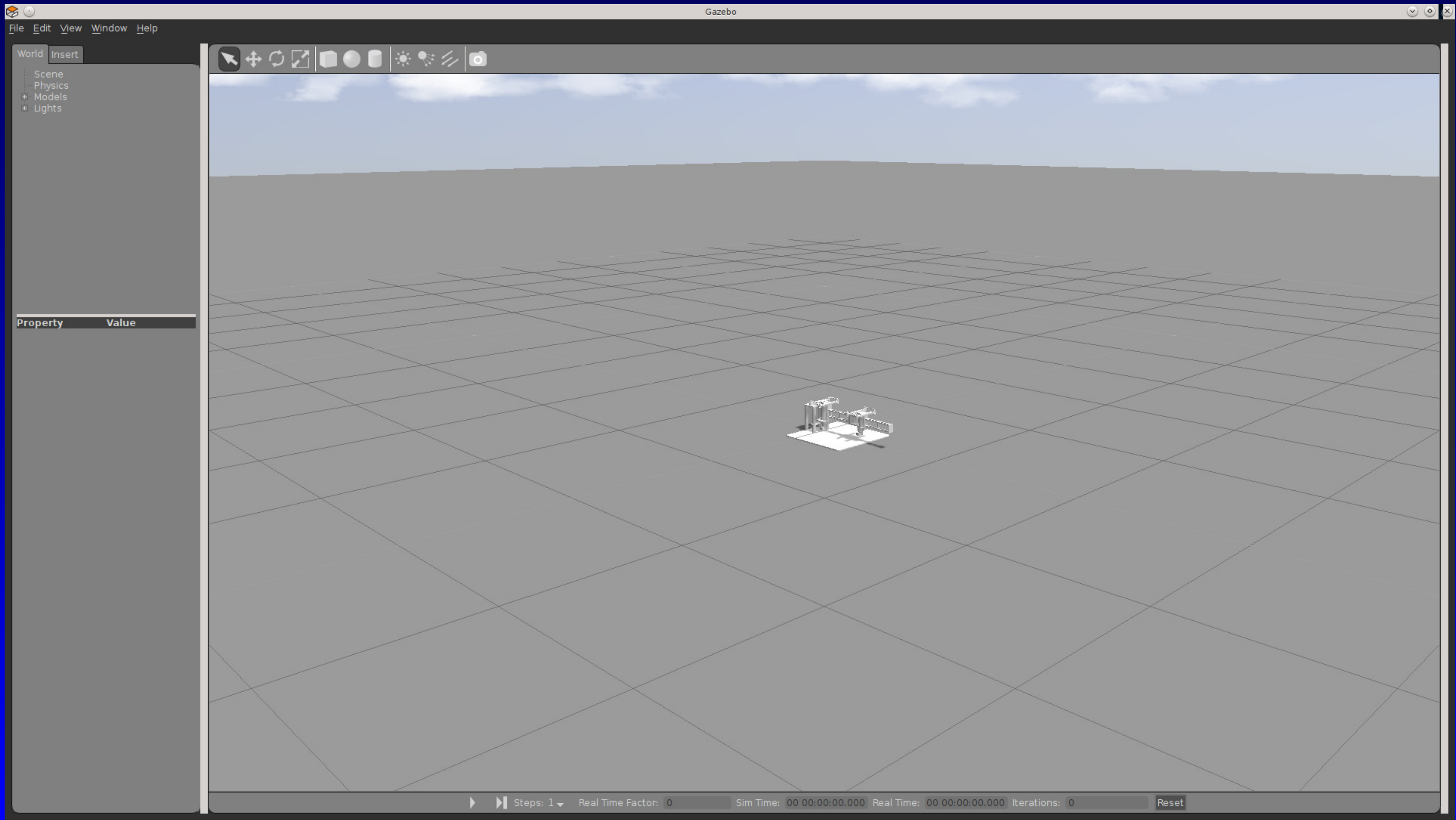
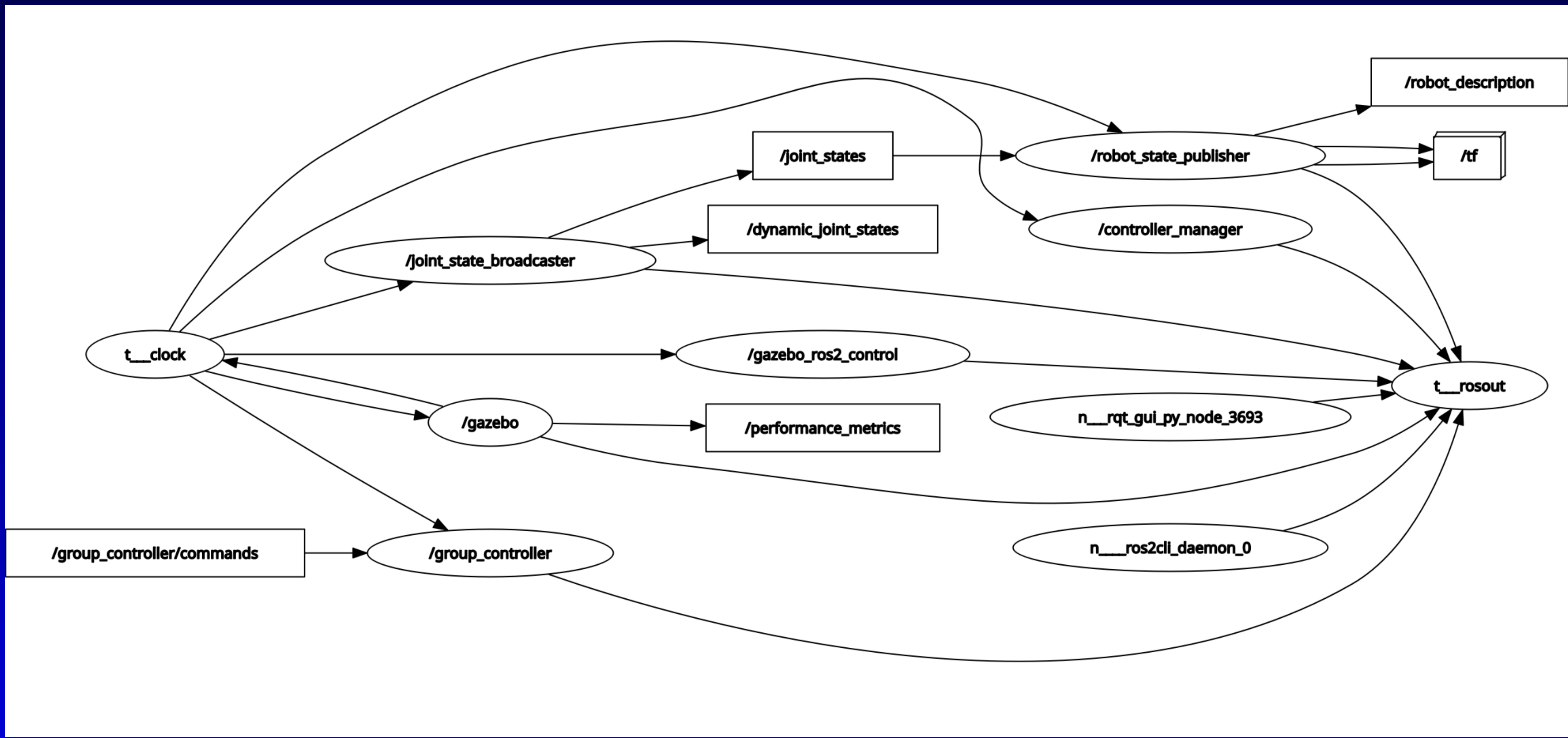


Gráfico de Computação

rqt_graph &



Movimentação do Robô

- Com o controlador *bypass* se aplica diretamente o torque nas juntas (malha aberta)

```
ros2 topic pub /group_controller/commands std_msgs/msg/
    Float64MultiArray '{data: [1.0, 1.0] }' –1
```

- ou

```
ros2 run q2d_bringup group_torque_step.sh 1 1
```

- O controlador fica aplicando o torque até receber outra referência
- Não é muito útil para movimentar o robô "na mão"
- Usado em testes em malha aberta

scripts/group_torque_step.sh

```
#!/bin/bash

if [ "$#" -ne 2 ]; then
        echo "Usage: $0 <shoulder torque> <elbow toque>"
        exit -1;
fi;

ros2 topic pub /group_controller/commands std_msgs/msg/
       Float64MultiArray "{data: [$1, $2] }" -1
```

Carregar a Interface com o Robô Real

```
ros2 launch q2d_hardware controller_manager.launch.xml
```

```
<launch>
    <node name="controller_manager" pkg="controller_manager" exec="ros2_control_node">
        <param name="robot_description" value="$(command 'xacro $(find-pkg-share q2d_description)/urdf/q2d.urdf hardware:=real_robot')" type="str"/>
        <param name="use_sim_time" value="false"/>
        <param name="update_rate" value="1000"/>
    </node>
</launch>
```

ros2 control CLI

- Comandos úteis para debugar controladores
- Listar as interfaces de *hardware*

---

ros2 control list_hardware_interfaces

- Listar os tipos de controladores disponíveis

---

ros2 control list_controller_types

- Listar os controladores carregados

---

ros2 control list_controllers

- Recarregar as bibliotecas de controladores

---

ros2 control reload_controller_libraries

---

ros2 control CLI

- Carregar um controlador

```
ros2 control load_controller <controller_name>
```

- Descarregar um controlador

```
ros2 control unload_controller <controller_name>
```

- Ajustar o estado de um controlador

```
ros2 control set_controller_state <controller_name> <configure |
    start | stop>
```

- Chavear controladores

```
ros2 control switch_controllers --stop <controller_list> --start
    <controller_list>
```

Controladores PID

- Os controladores
  - `effort_controllers/JointVelocityController`
  - `effort_controllers/JointPositionController`
  - `effort_controllers/JointGroupPositionController`

  não foram portados para o ROS 2
- Usam uma implementação baseada em *templates*
- Aqui será usado o controlador `pid_plus_gravity_controller`, que também será usado no Barrett WAM

Cálculo dos Ganhos do PID

- Desprezando-se o acoplamento dinâmico entre as juntas, pode-se refletir a inércia equivalente para cada junta
- Considerando a inércia equivalente e o atrito, cada junta pode ser representada por um sistema de segunda ordem
- A partir do desempenho desejado, pode-se calcular os polos desejados e calcular os ganhos do PID usando a função `pidtune` do Matlab

Cálculo dos Ganhos do PID

$$ {}^{i}P_{cij} = \left({}^{i}P_{ci}m_i + \left({}^{i}R_j(\theta_j)\, {}^{j}P_{cj} + {}^{i}P_j \right) m_j \right) / (m_i + m_j) $$

$$ {}^{cij}P_{cj} = \left({}^{i}R_j(\theta_j)\, {}^{j}P_{cj} + {}^{i}P_j \right) - {}^{i}P_{cij} $$

$$ {}^{cij}I_j = {}^{cj}I_j + m_j \left({}^{cij}P_{cj}^T\, {}^{cij}P_{cj} I - {}^{cij}P_{cj}\, {}^{cij}P_{cj}^T \right) $$

$$ {}^{cij}P_{ci} = {}^{i}P_{ci} - {}^{i}P_{cij} $$

$$ {}^{cij}I_i = {}^{ci}I_i + m_i \left({}^{cij}P_{ci}^T\, {}^{cij}P_{cj} I - {}^{cij}P_{ci}\, {}^{cij}P_{ci}^T \right) $$

$$ {}^{cij}I_{ij} = {}^{cij}I_j + {}^{cij}I_i $$

$$ G(s) = \frac{1/i_{33}}{s\left(s + f_a/i_{33}\right)} $$

Cálculo dos Ganhos do PID

```
% Mass, inertia tensor, and center of mass of shoulder active link
msa=0.19730261508;
Isa=[0.00038685518702305, 0.00000000055222416,
        –0.00000031340718614;
0.00000000055222416, 0.00010241438913870, –0.00000000015426019;
–0.00000031340718614 –0.00000000015426019, 0.00047879093657893];
Psa=[0.0252456823; –0.00000002723; 0.06470401873];

% Mass, inertia tensor, and center of mass of shoulder passive link
msp=1.26475817816;
Isp=[0.00346199967740929, –0.00010902049981923,
        –0.00401182173261703;
–0.00010902049981923, 0.03314904030482527, 0.00005087359051462;
–0.00401182173261703, 0.00005087359051462, 0.03113579694057124];
Psp=[0.16516344805; –0.00048428845; –0.00016382412];
```

Cálculo dos Ganhos do PID

```
% Mass, inertia tensor, and center of mass of elbow active link
mea=0.19712951877;
Iea=[0.00038850510800265, 0.00000000052121416, 0.00000404728675587;
0.00000000052121416, 0.00010146693248154, 0.00000000002789435;
0.00000404728675587, 0.00000000002789435, 0.00048091942023028];
Pea=[0.02548273493; -0.00000002263; 0.05254513577];

% Mass, inertia tensor, and center of mass of elbow passive link
mep=0.67529215765;
Iep=[0.00132247071698947, -0.00000000605403474,
        -0.00090893541574333;
-0.00000000605403474, 0.00774007102253750, 0.00000000624688369;
-0.00090893541574333, 0.00000000624688369, 0.00751638349361413];
Pep=[0.06204831581; 0.00000013809; 0.01489882531];
```

Cálculo dos Ganhos do PID

```
% Total mass of elbow
mec=mea+mep
% Center of mass of elbow (the origins of the systems are the same)
Pec=(Pea*mea+Pep*mep)/mec

% Position of the center of mass of the elbow passive link with respect to
% the combined center of mass
Pepc=Pep-Pec;
% Inertia tensor of elbow passive link at the combined center of mass
Iepc=Iep+mep*(Pepc'*Pepc*eye(3)-Pepc*Pepc');

% Position of the center of mass of the elbow active link with respect to
% the combined center of mass
Peac=Pea-Pec;
% Inertia tensor of elbow active link at the combined center of mass
Ieac=Iea+mea*(Peac'*Peac*eye(3)-Peac*Peac');
```

Cálculo dos Ganhos do PID

```
% Inertia tensor of elbow
Iec=Iepc+Ieac

% Time constant of elbow joint
Te=4e-3;

% Transfer function of elbow joint
Ge=tf(1/Iec(3,3),[1 1/Te 0])

[pide,pe]=pidtune(Ge,'pid',2*pi/Te/10.97)
```

Cálculo dos Ganhos do PID

```
% Position of elbow origin with respect to shoulder origin
Pse=[0.343;0;0];
% Total mass of shoulder
msc=msa+msp+mec
% Center of mass of shoulder (the origins of the systems are the same)
Psc=(Psa*msa+Psp*msp+(Pse+Pec)*mec)/(msc+mec)

% Position of the center of mass of the shoulder passive link with respect to
% the combined center of mass
Pspc=Psp-Psc;
% Inertia tensor of shoulder passive link at the combined center of mass
Ispc=Isp+msp*(Pspc'*Pspc*eye(3)-Pspc*Pspc');

% Position of the center of mass of the shoulder active link with respect to
% the combined center of mass
Psac=Psa-Psc;
```

Cálculo dos Ganhos do PID

```
% Inertia tensor of shoulder active link at the combined center of mass
Isac=Isa+msa*(Psac'*Psac*eye(3)-Psac*Psac');

% Position of the center of mass of the elbow with respect to
% the combined center of mass of shoulder
Psec=Pse+Pec-Psc;
% Inertia tensor of shoulder active link at the combined center of mass
Isec=Iec+mec*(Psec'*Psec*eye(3)-Psec*Psec');

% Inertia tensor of shoulder
Isc=Ispc+Isac+Isec

% Time constant of shoulderjoint
Ts=5e-3;
```


Cálculo dos Ganhos do PID

```
% Transfer function of shoulder joint
Gs=tf(1/Isc(3,3),[1 1/Ts 0])

[pids,ps]=pidtune(Gs,'pid',2*pi/Ts/10.97)
```

Download e Instalação do Pacote

- Instalação inicial

```
cd ~/colcon_ws/src
git clone –b $ROS_DISTRO http://git.ece.ufrgs.br/eng10051/q2d
cd ..
colcon build −−symlink−install
source ~/colcon_ws/install/setup.bash
```

- Atualização

```
cd ~/colcon_ws/src/q2d
rm q2d_bringup/COLCON_IGNORE
git pull
cd ../.. ; colcon build −−symlink−install
source ~/colcon_ws/install/setup.bash
```

PID com Compensação de Gravidade

- É o controlador *default* do WAM e da maioria dos robôs industriais
- Instalação

```
cd ~/colcon_ws/src
git clone –b $ROS_DISTRO http://git.ece.ufrgs.br/
    pid_plus_gravity_controller
cd ..
colcon build ––symlink–install
source ~/colcon_ws/install/setup.bash
```

pid_plus_gravity.launch.xml

```xml
<launch>
  <arg name="config" default="$(find-pkg-share q2d_bringup)/
     config/pid_plus_gravity.yaml"/>

  <node name="pid_plus_gravity_controller_spawner" pkg="
     controller_manager" exec="spawner"
    args="-t effort_controllers/PidPlusGravityController -p $(var
     config) pid_plus_gravity_controller"/>

  <node name="joint_state_broadcaster_spawner" pkg="
     controller_manager" exec="spawner"
    args="-t joint_state_broadcaster/JointStateBroadcaster
     joint_state_broadcaster"/>
</launch>
```

pid_plus_gravity.yaml

```
pid_plus_gravity_controller:
    ros__parameters:
        joints:
            – shoulder_active_joint
            – elbow_active_joint
        shoulder_active_joint: {p: 2310.0, i: 4640.0, d: 0.299,
    i_clamp_max: 27.94, i_clamp_min: –27.94}
        elbow_active_joint: {p: 339.0, i: 851.0, d: 0.351,
    i_clamp_max: 13.62, i_clamp_min: –13.62}
        gravity: {x: 0.0, y: 0.0, z: –9.8}
        chain: {root: "origin_link", tip: "tool_link"}
        priority: 99
```

Simulação no Gazebo

```
ros2 launch q2d_bringup gazebo.launch.xml controller:=
    pid_plus_gravity
```

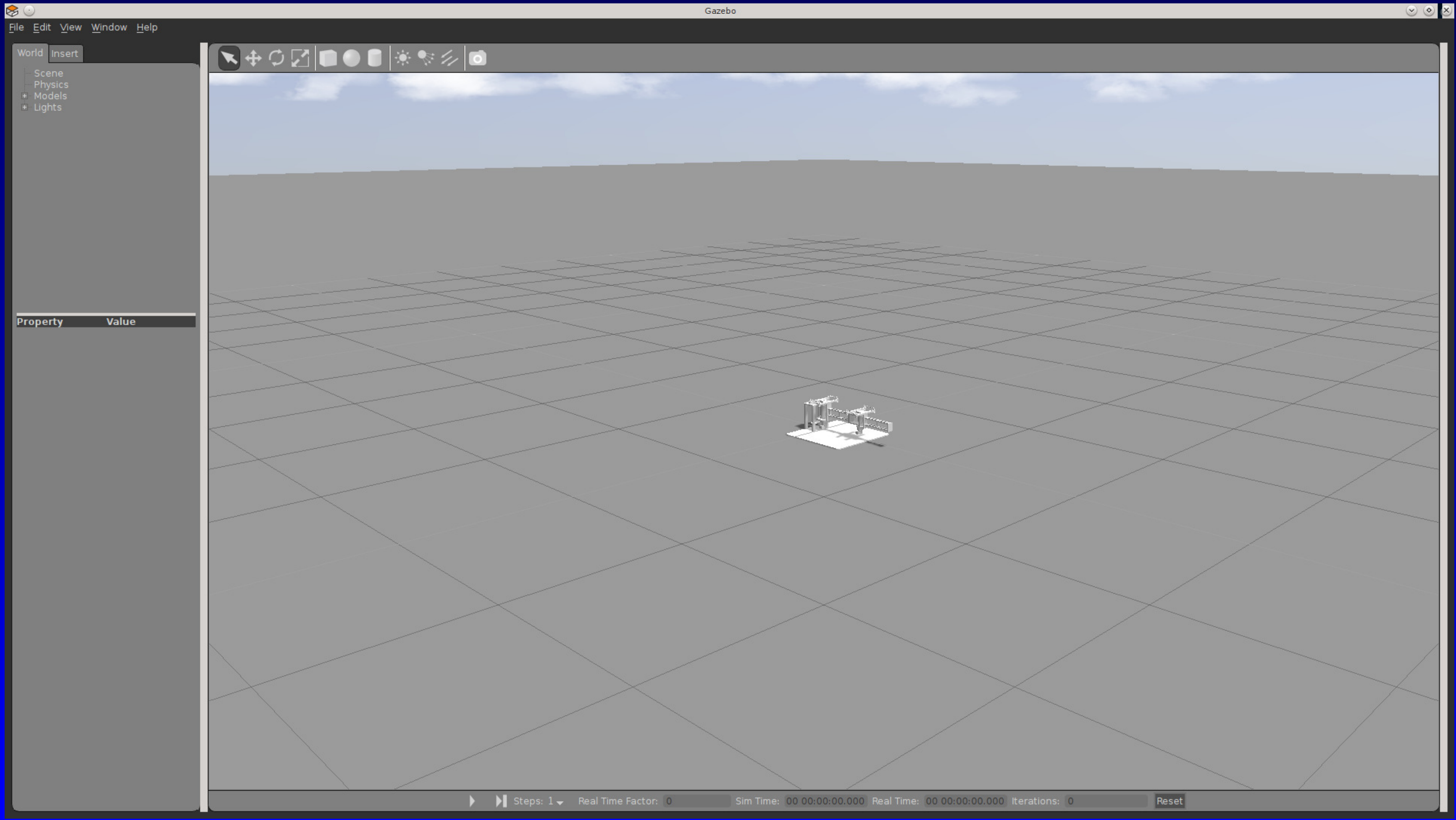
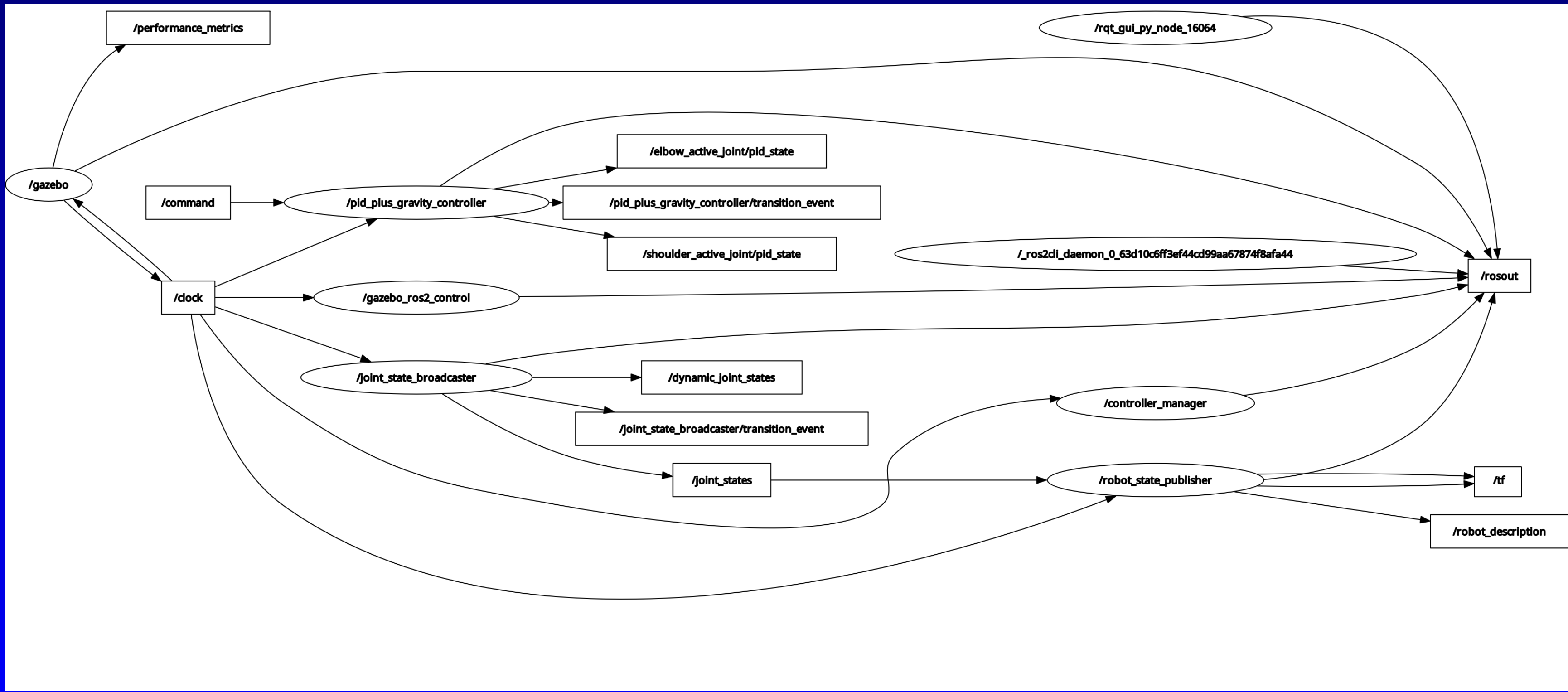


Gráfico de Computação

- É preciso dar *play* no simulador para o controlador terminar de carregar

rqt_graph &



Reconfiguração Dinâmica

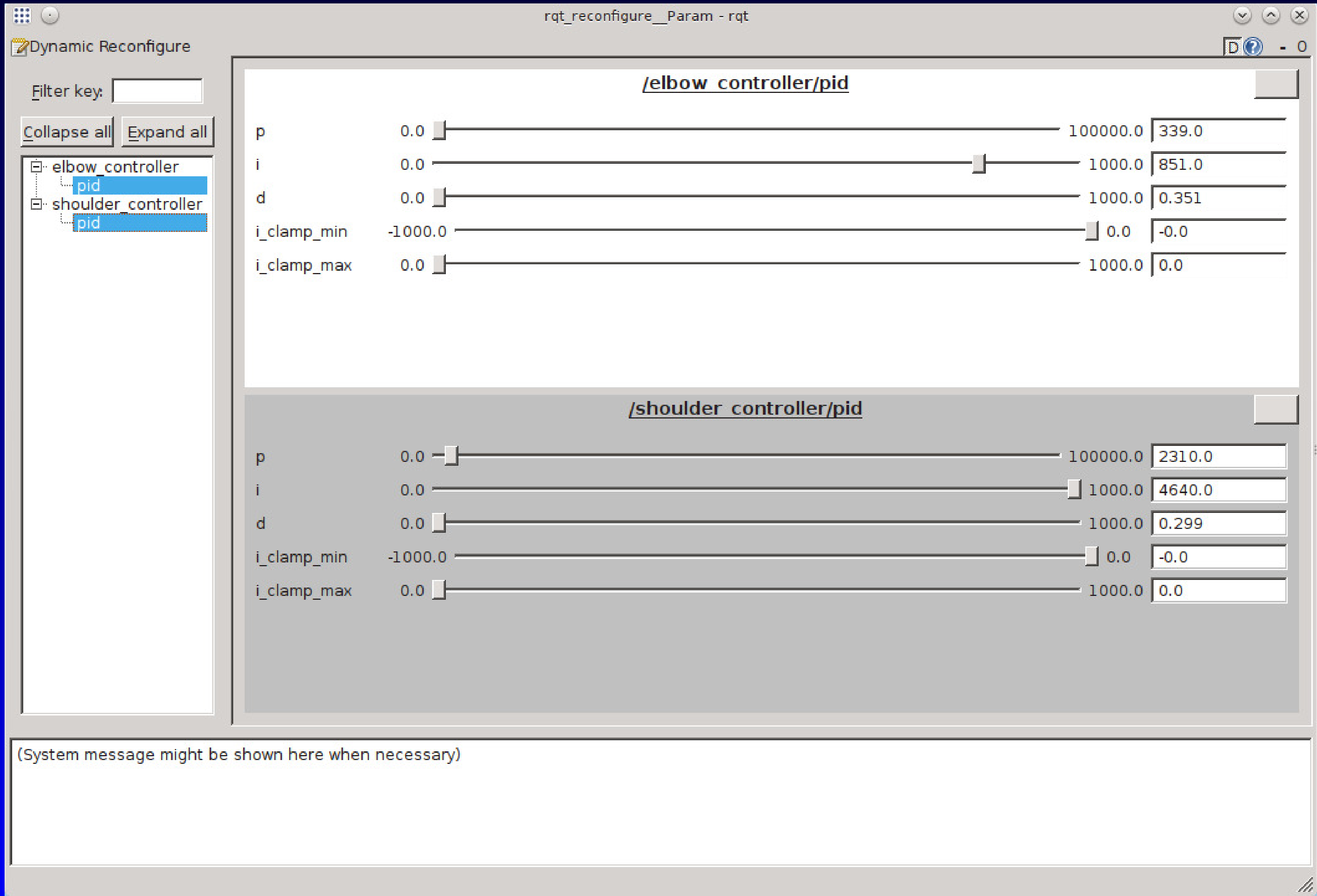
- Permite a alteração de parâmetros dos nodos em tempo de execução
- Nem todos os nodos do ROS utilizam
  - PID do `control_toolbox` utiliza
    - `pid_plus_gravity_controller`
    - `computed_torque_controller`
  - `joint_state_broadcaster` não utiliza
- Não é uma boa ideia alterar os ganhos dos controladores empiricamente

---

```
ros2 run rqt_reconfigure rqt_reconfigure &
```

---

rqt_reconfigure



Mover o Robô

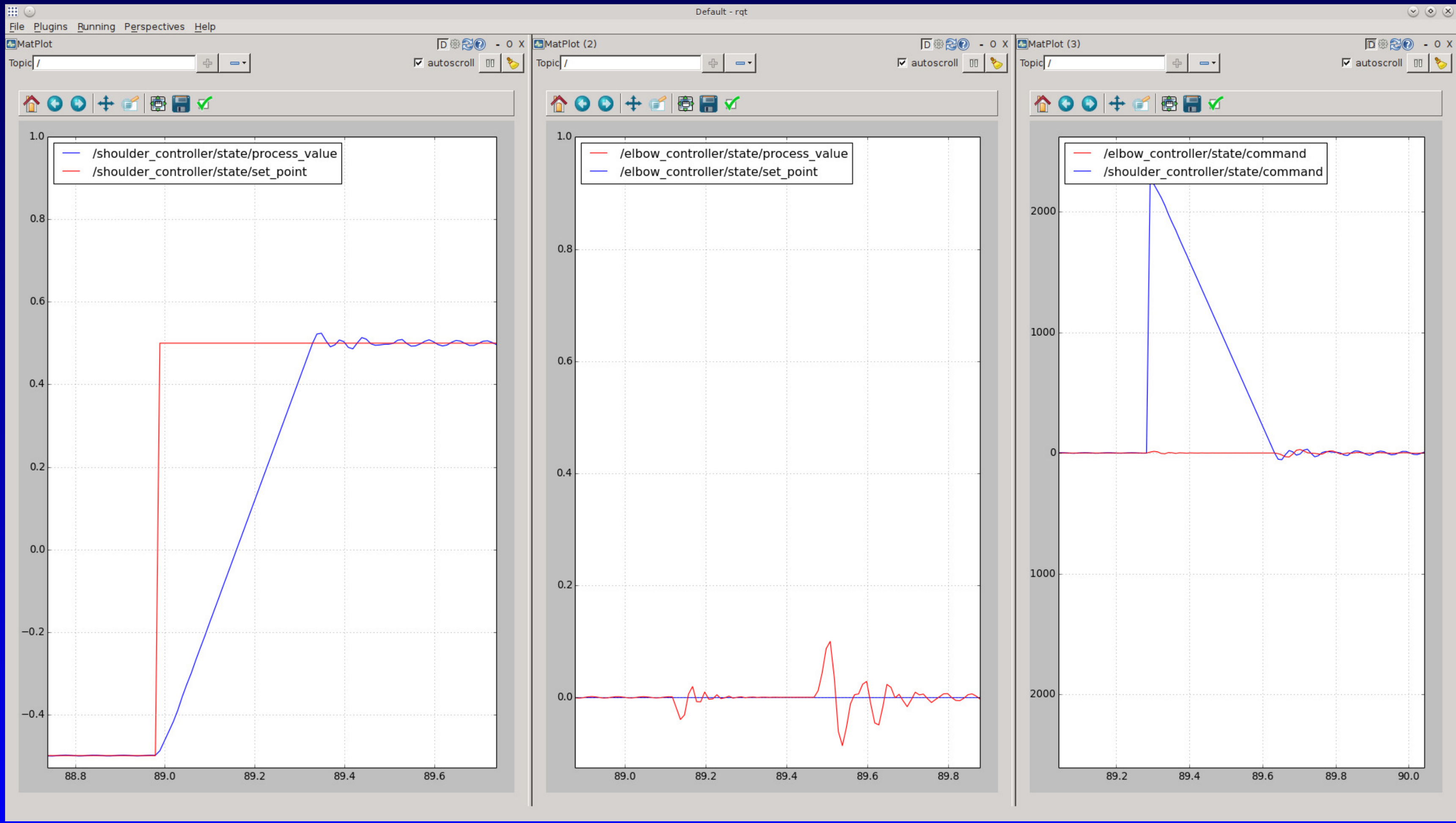
- Dar *play* no simulador
- Os mesmos comandos podem ser usados para o robô real
  - Lançar o `hardware.launch.xml` ao invés do `gazebo.launch.xml`
- Publicar nos tópicos das referências dos controladores

```
ros2 topic pub /command trajectory_msgs/msg/JointTrajectoryPoint "{
    positions: [–1.0, 1.0]}" –1
ros2 run q2d bringup joint_trajectory_step.sh –1.0 1.0
```

---

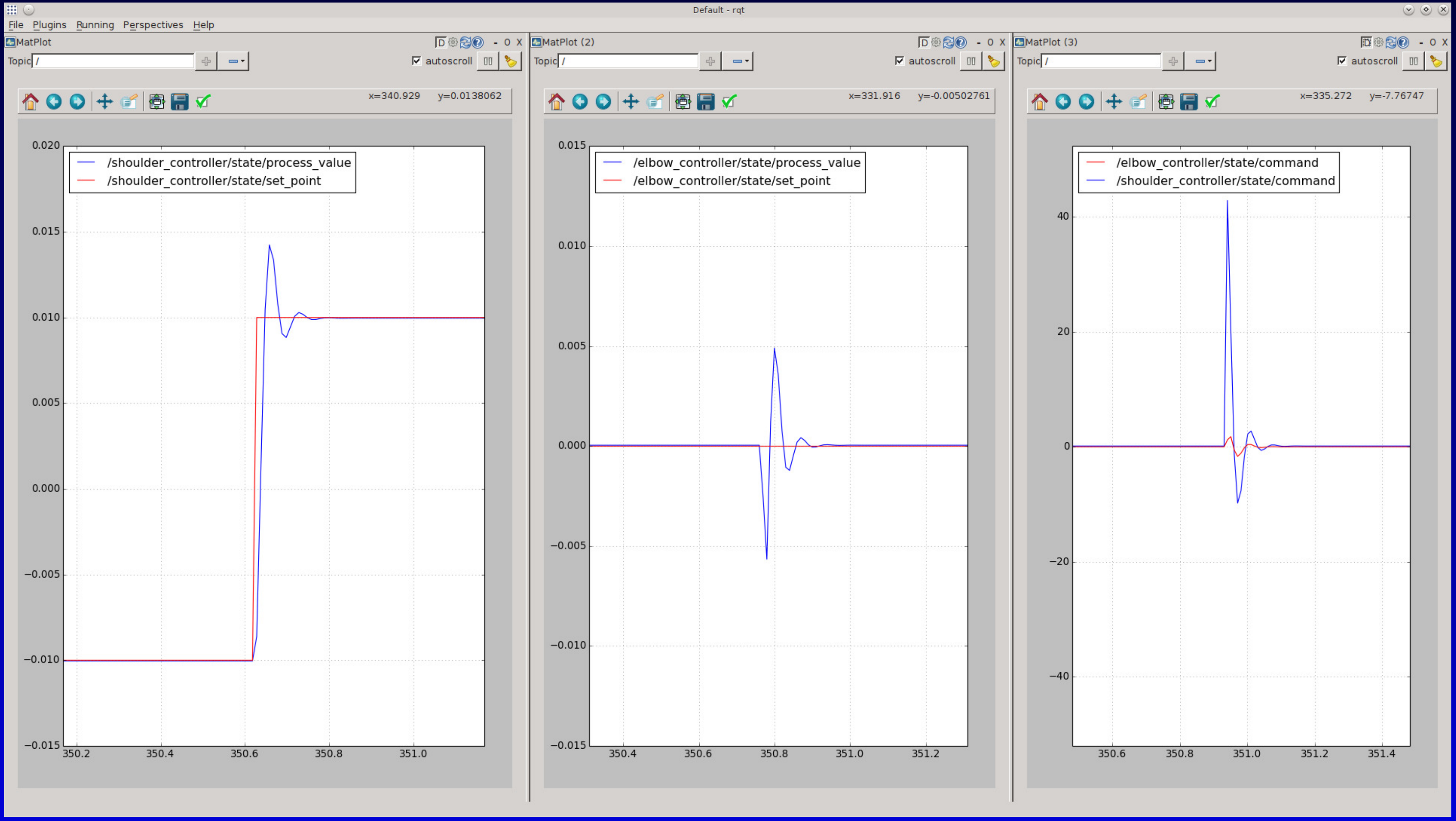
rqt

- *Framework* para GUIs
- Pode-se vizualizar várias instâncias do `rqt_plot` de forma conveniente



Sem Saturação

- Degrau de -0.01 rad to 0.01 rad



Execução com o Robô Real

ros2 launch q2d_bringup hardware.launch.xml controller:=
pid_plus_gravity

```
<launch>
    <arg name="gui" default="true"/>

    <arg name="controller" default="group_bypass"/>
    <arg name="config" default="$(find-pkg-share
    q2d_bringup)/config/$(var controller).yaml"/>

    <include file="$(find-pkg-share q2d_hardware)/launch
    /controller_manager.launch.xml"/>

    <include file="$(find-pkg-share q2d_bringup)/launch
    /$(var controller).launch.xml">
        <arg name="config" value="$(var config)"/>
        <arg name="use_sim_time" value="false"/>
    </include>
```

Carga do Gerenciador e Controlador

```
    <include if="$(var gui)" file="$(find-pkg-share
    q2d_description)/launch/display.launch.xml">
        <arg name="gui" value="false"/>
    </include>
</launch>
```