franka_ros2

> **Note**
>
> `franka_ros2` is not supported on Windows.

You can access the changelog of franka_ros2 at this link

The franka_ros2 repo contains a ROS 2 integration of libfranka.

> **Caution**
>
> franka_ros2 is in rapid development. Anticipate breaking changes. Report bugs on GitHub.

Installation

Please refer to the README.md

MoveIt

To see if everything works, you can try to run the MoveIt example on the robot:

```
ros2 launch franka_fr3_moveit_config moveit.launch.py robot_ip:=<fci-ip>
```

Then activate the `MotionPlanning` display in RViz.

If you do not have a robot you can still test your setup by running on a dummy hardware:

```
ros2 launch franka_fr3_moveit_config moveit.launch.py robot_ip:=dont-care
use_fake_hardware:=true
```

Wait until you can see the green `You can start planning now!` message from MoveIt inside the terminal. Then turn off the `PlanningScene` and turn it on again. After that turn on the `MotionPlanning`.

Example Controllers

This repo comes with a few example controllers located in the `franka_example_controllers` package.

The following launch files are executed with the gripper by default. If you do not have the gripper attached you can disable the gripper in the launch file with `load_gripper:=false`.

Move-to-start

This controller moves the robot to its home configuration.

```
ros2 launch franka_bringup move_to_start_example_controller.launch.py arm_id:=fr3
robot_ip:=<fci-ip>
```

Gravity Compensation

This is the simplest controller that we have and is a good starting point to write your own. It sends zero as torque command to all joints, which means that the robot only compensates its own weight.

```
ros2 launch franka_bringup gravity_compensation_example_controller.launch.py
arm_id:=fr3 robot_ip:=<fci-ip>
```

Gripper Example

Demonstrates the Franka *Action Interface* for controlling the Franka Hand (aka: Gripper). The controller submits *Goals* to repeatedly close, then reopen, the gripper given a hard-coded target grasp size with epsilon. It evaluates whether the grasp is successful or failed based on the object's size and the defined tolerances.

```
ros2 launch franka_bringup gripper_example_controller.launch.py arm_id:=fr3 robot_ip:=
<fci-ip>
```

Joint Impedance Example

The example moves joints 4 and 5 in a periodic movement that is very compliant. You can try to move the joints while it is running.

```
ros2 launch franka_bringup joint_impedance_example_controller.launch.py arm_id:=fr3
robot_ip:=<fci-ip>
```

Joint Impedance With IK Example

The example uses the LMA-Orocos solver from MoveIt service to compute the joint positions for the desired pose. The desired pose is to move the end-effector periodically in x and z directions. You can change the kinematic solver in the franka_fr3_moveit_config package, kinematics.yaml file.

```
ros2 launch franka_bringup joint_impedance_with_ik_example_controller.launch.py
arm_id:=fr3 robot_ip:=<fci-ip>
```

Model Example Controller

This is a read-only controller which prints the coriolis force vector, gravity force vector, pose matrix of Joint4, Joint4 body jacobian and end-effector jacobian with respect to the base frame.

```
ros2 launch franka_bringup model_example_controller.launch.py arm_id:=fr3 robot_ip:=
<fci-ip>
```

Joint Position Example

This example sends periodic position commands to the robot.

```
ros2 launch franka_bringup joint_position_example_controller.launch.py arm_id:=fr3
robot_ip:=<fci-ip>
```

Joint Velocity Example

This example sends periodic velocity commands to the 4th and 5th joint of the robot.

```
ros2 launch franka_bringup joint_velocity_example_controller.launch.py arm_id:=fr3
robot_ip:=<fci-ip>
```

Cartesian Pose Example

This example uses the CartesianPose interface to send periodic pose commands to the robot.

```
ros2 launch franka_bringup cartesian_pose_example_controller.launch.py arm_id:=fr3
robot_ip:=<fci-ip>
```

Cartesian Orientation Example

This example uses CartesianOrientation interface to send periodic orientation commands around X axis of the end effector of the robot.

```
ros2 launch franka_bringup cartesian_orientation_example_controller.launch.py
arm_id:=fr3 robot_ip:=<fci-ip>
```

Cartesian Pose Elbow Example

This example sends periodic elbow commands while keeping the end effector pose constant.

```
ros2 launch franka_bringup cartesian_elbow_example_controller.launch.py arm_id:=fr3
robot_ip:=<fci-ip>
```

Cartesian Velocity Example

This example uses the CartesianVelocity interface to send periodic velocity commands to the robot.

```
ros2 launch franka_bringup cartesian_velocity_example_controller.launch.py arm_id:=fr3
robot_ip:=<fci-ip>
```

Cartesian Elbow Example

This example uses the CartesianElbow interface to send periodic elbow commands to the robot while keeping the end effector velocity constant.

```
ros2 launch franka_bringup elbow_example_controller.launch.py arm_id:=fr3 robot_ip:=
<fci-ip>
```

Package Descriptions

This section contains more detailed descriptions of what each package does. In general the package structure tries to adhere to the structure that is proposed here.

franka_bringup

This package contains the launch files for the examples as well as the basic `franka.launch.py` launch file, that can be used to start the robot without any controllers.

When you start the robot with:

```
ros2 launch franka_bringup franka.launch.py arm_id:=fr3 robot_ip:=<fci-ip>
use_rviz:=true
```

There is no controller running apart from the `joint_state_broadcaster`. However, a connection with the robot is still established and the current robot pose is visualized in RViz. In this mode the robot can be guided when the user stop button is pressed. However, once a controller that uses the `effort_command_interface` is started, the robot will be using the torque interface from libfranka. For example it is possible to launch the `gravity_compensation_example_controller` by running:

```
ros2 control load_controller --set-state active
gravity_compensation_example_controller
```

This is the equivalent of running the `gravity_compensation_example_controller.launch.py` launch file mentioned in Gravity Compensation.

When the controller is stopped with:

```
ros2 control set_controller_state gravity_compensation_example_controller inactive
```

the robot will stop the torque control and will only send its current state over the FCI.

You can now choose to start the same controller again with:

```
ros2 control set_controller_state gravity_compensation_example_controller active
```

or load and start a different one:

```
ros2 control load_controller --set-state active joint_impedance_example_controller
```

franka_description

> **Warning**
>
> As of version 0.1.14 the franka_description package is not available in the franka_ros2 repository. It is available in a separate repository Franka Description.

This package contains the xacro files and meshes that are used to visualize the robot. Further, it contains a launch file that visualizes the robot model without access to a real robot:

```
ros2 launch franka_description visualize_franka.launch.py load_gripper:=<true|false>
```

franka_example_controllers

This package contains a few controllers that can be seen as example of how to write controllers in ROS 2. Currently, a controller only has access to measured joint positions and joint velocities. Based on this information the controller can send torque commands. It is currently not possible to use other interfaces like the joint position interface.

franka_gripper

This package contains the `franka_gripper_node` for interfacing with the `Franka Hand`.

The `franka_gripper_node` provides the following actions:

- `homing` - homes the gripper and updates the maximum width given the mounted fingers.
- `move` - moves to a target width with the defined speed.
- `grasp` - tries to grasp at the desired width with the desired force while closing with the given speed. The operation is successful if the distance `d` between the gripper fingers is `width - epsilon.inner < d < width + epsilon.outer`
- `gripper_action` - a special grasping action for MoveIt.

Also, there is a `stop` service that aborts gripper actions and stops grasping.

Use the following launch file to start the gripper:

```
ros2 launch franka_gripper gripper.launch.py robot_ip:=<fci-ip>
```

In a different tab you can now perform the homing and send a grasp command.:

```
ros2 action send_goal /fr3_gripper/homing franka_msgs/action/Homing {}
ros2 action send_goal -f /fr3_gripper/grasp franka_msgs/action/Grasp "{width: 0.00,
speed: 0.03, force: 50}"
```

The inner and outer epsilon are 0.005 meter per default. You can also explicitly set the epsilon:

```
ros2 action send_goal -f /fr3_gripper/grasp franka_msgs/action/Grasp "{width: 0.00,
speed: 0.03, force: 50, epsilon: {inner: 0.01, outer: 0.01}}"
```

To stop the grasping, you can use `stop` service.:

```
ros2 service call /fr3_gripper/stop std_srvs/srv/Trigger {}
```

franka_hardware

> **Important**
>
> Breaking changes as of 0.1.14 release: `franka_hardware` robot_state and robot_model will be prefixed by the `arm_id`.
>
> - `panda/robot_model  -> ${arm_id}/robot_model`
> - `panda/robot_state  -> ${arm_id}/robot_state`
>
> There is no change with the state and command interfaces naming. They are prefixed with the joint names in the URDF.

This package contains the `franka_hardware` plugin needed for ros2_control. The plugin is loaded from the URDF of the robot and passed to the controller manager via the robot description. It provides for each joint:

- a `position state interface` that contains the measured joint position.
- a `velocity state interface` that contains the measured joint velocity.
- an `effort state interface` that contains the measured link-side joint torques.
- an `initial_position state interface` that contains the initial joint position of the robot.
- an `effort command interface` that contains the desired joint torques without gravity.
- a `position command interface` that contains the desired joint position.
- a `velocity command interface` that contains the desired joint velocity.

In addition

- a `franka_robot_state` that contains the robot state information, franka_robot_state.
- a `franka_robot_model_interface` that contains the pointer to the model object.

> **Important**
>
> `franka_robot_state` and `franka_robot_model_interface` state interfaces should not be used directly from hardware state interface. Rather, they should be utilized by the franka_semantic_components interface.

The IP of the robot is read over a parameter from the URDF.

franka_semantic_components

This package contains franka_robot_model, franka_robot_state and cartesian command classes. These classes are used to convert franka_robot_model object and franka_robot_state objects, which are stored in the hardware_state_interface as a double pointer.

For further reference on how to use these classes: Franka Robot State Broadcaster and Franka Example Controllers(model_example_controller)

- Cartesian Pose Interface:

This interface is used to send Cartesian pose commands to the robot by using the loaned command interfaces. FrankaSemanticComponentInterface class is handling the loaned command interfaces and state interfaces. While starting the cartesian pose interface, the user needs to pass a boolean flag to the constructor to indicate whether the interface is for the elbow or not.

```
auto is_elbow_active = false;
CartesianPoseInterface cartesian_pose_interface(is_elbow_active);
```

This interface allows users to read the current pose command interface values set by the franka hardware interface.

```
std::array<double, 16> pose;
pose = cartesian_pose_interface.getInitialPoseMatrix();
```

One could also read quaternion and translation values in Eigen format.

```
Eigen::Quaterniond quaternion;
Eigen::Vector3d translation;
std::tie(quaternion, translation) =
cartesian_pose_interface.getInitialOrientationAndTranslation();
```

After setting up the cartesian interface, you need to `assign_loaned_command_interfaces` and `assign_loaned_state_interfaces` in your controller. This needs to be done in the on_activate() function of the controller. Examples can be found in the assign loaned comamand interface example

```
cartesian_pose_interface.assign_loaned_command_interfaces(command_interfaces_);
cartesian_pose_interface.assign_loaned_state_interfaces(state_interfaces_);
```

In the update function of the controller you can send pose commands to the robot.

```
std::array<double, 16> pose;
pose = {1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0.5, 0, 0.5, 1};
cartesian_pose_interface.setCommanded(pose);
```

Or you can send quaternion, translation values in Eigen format.

```
Eigen::Quaterniond quaternion(1, 0, 0, 0);
Eigen::Vector3d translation(0.5, 0, 0.5);
cartesian_pose_interface.setCommand(quaternion, translation);
```

- Cartesian Velocity Interface:

This interface is used to send Cartesian velocity commands to the robot by using the loaned command interfaces. FrankaSemanticComponentInterface class is handling the loaned command interfaces and state interfaces.

```
auto is_elbow_active = false;
CartesianVelocityInterface cartesian_velocity_interface(is_elbow_active);
```

To send the velocity command to the robot, you need to assign_loaned_command_interface in your custom controller.

```
cartesian_velocity_interface.assign_loaned_command_interface(command_interfaces_);
```

In the update function of the controller you can send cartesian velocity command to the robot.

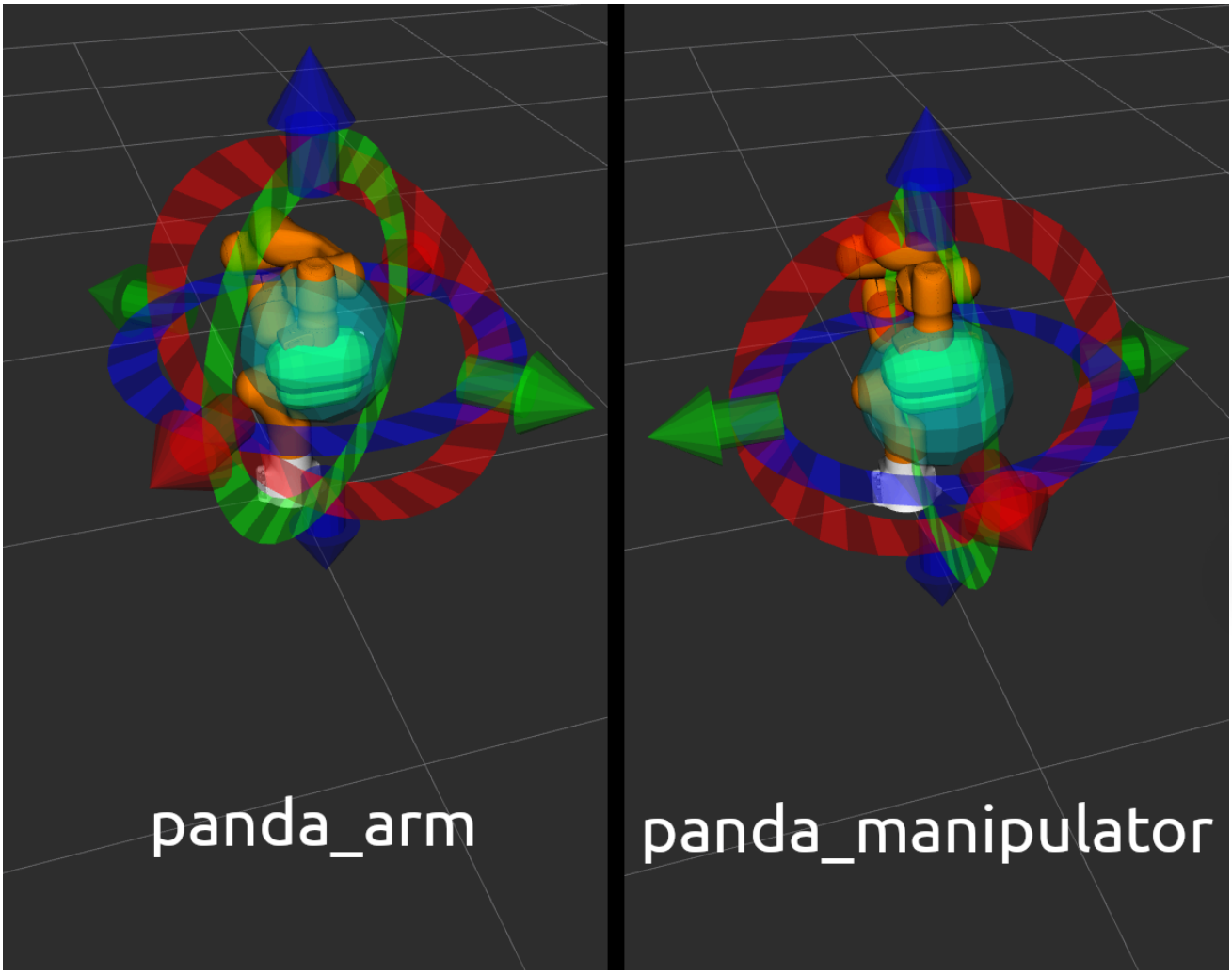
```
std::array<double, 6> cartesian_velocity;
cartesian_velocity = {0, 0, 0, 0, 0, 0.1};
cartesian_velocity_interface.setCommand(cartesian_velocity);
```

franka_robot_state_broadcaster

This package contains read-only franka_robot_state_broadcaster controller. It publishes franka_robot_state topic to the topic named */franka_robot_state_broadcaster/robot_state*. This controller node is spawned by franka_launch.py in the franka_bringup. Therefore, all the examples that include the franka_launch.py publishes the robot_state topic.

franka_fr3_moveit_config

This package contains the configuration for MoveIt2. There is a new move group called `panda_manipulator` that has its tip between the fingers of the gripper and has its Z-axis rotated by -45 degrees, so that the X-axis is now facing forward, making it easier to use. The `panda_arm` move group is still available for backward compatibility. New applications should use the new `panda_manipulator` move group instead.



Visualization of the old and the new move group

franka_msgs

This package contains the definitions for the different gripper actions and robot state message.

joint_effort_trajectory_controller

This package contains a modified joint_trajectory_controller that can use the effort interface of the `franka_hardware::FrankaHardwareInterface`. It is based on this Pull request.

> **Note**
>
> This package will be soon deleted as the fix is available in ros2_controllers master branch. As soon as, it's backported to Humble, it will be deleted from franka_ros2 repository.

franka_gazebo

> **Important**
>
> Minimum necessary *franka_description* version is 0.3.0. You can clone franka_description package from https://github.com/frankaemika/franka_description.

A project integrating Franka ROS 2 with the Gazebo simulator.

Launch RVIZ + Gazebo

Launch an example which spawns RVIZ and Gazebo showing the robot:

```
ros2 launch franka_gazebo_bringup visualize_franka_robot.launch.py
```

If you want to display another robot, you can define the arm_id:

```
ros2 launch franka_gazebo_bringup visualize_franka_robot.launch.py arm_id:=fp3
```

If you want to start the simulation including the franka_hand:

```
ros2 launch franka_gazebo_bringup visualize_franka_robot.launch.py load_gripper:=true
franka_hand:='franka_hand'
```

Joint Velocity Control Example with Gazebo

Before starting, be sure to build *franka_example_controllers* and *franka_description* packages. *franka_description* must have the minimum version of 0.3.0.

```
colcon build --packages-select franka_example_controllers
```

Now you can launch the velocity example with Gazebo simulator.

```
ros2 launch franka_gazebo_bringup gazebo_joint_velocity_controller_example.launch.py
load_gripper:=true franka_hand:='franka_hand'
```

Keep in mind that the gripper joint has a bug with the joint velocity controller. If you are interested in controlling the gripper please use joint position interface.

Joint Position Control Example with Gazebo

To run the joint position control example you need to have the required software listed in the joint velocity control section.

Then you can run with the following command.

```
ros2 launch franka_gazebo_bringup gazebo_joint_position_controller_example.launch.py
load_gripper:=true franka_hand:='franka_hand'
```

Joint Impedance Control Example with Gazebo

For running torque example. You must compile the *franka_ign_ros2_control* package located under *franka_gazebo*. You can compile *franka_ign_ros2_control* with the following command.

```
colcon build --packages-select franka_ign_ros2_control
```

Then source your workspace.

```
source install/setup.sh
```

Then you can run the impedance control example.

```
ros2 launch franka_gazebo_bringup gazebo_joint_impedance_controller_example.launch.py
load_gripper:=true franka_hand:='franka_hand'
```

Throubleshooting

If you experience that Gazebo can't find your model files, try to include the workspace. E.g.

```
export GZ_SIM_RESOURCE_PATH=${GZ_SIM_RESOURCE_PATH}:/workspaces/src/
```

Differences between franka_ros and franka_ros2

This section gives an overview of the fundamental changes between `franka_ros` and `franka_ros2`.

franka_gripper

- All topics and actions were previously prefixed with `franka_gripper`. This prefix was renamed to `panda_gripper` to enable, in the future, a workflow where all prefixes are based on the `arm_id` to effortlessly enable multi arm setups.
- The `stop` action is now a service action as it is not preemptable.
- All actions (apart from the `gripper_action`) have the current gripper width as feedback.

franka_visualization

This package does not exist anymore. However, franka_description provides a launch file to visualize the robot model without a connection to a robot.

franka_control

This package does not exist anymore. The connection to the robot is provided by the hardware plugin in the franka_hardware package. The actions and services that it provided are currently not offered in `franka_ros2`.

Writing Controllers

Compared to `franka_ros` we currently offer a reduced set of controller interfaces:

- Joint positions
- Joint velocities
- Measured torques
- Franka robot state
- Franka robot model

Important

Franka robot state is published through franka_robot_state_broadcaster package to the topic named */franka_robot_state_broadcaster/robot_state*

Important

Both Franka robot state and Franka robot model are advised to use through franka_semantic_components class. They are stored in the state_interface as double pointers and casted back to their original objects inside the franka_semantic_component class.

Example of using franka_model can be found in the franka_example_controllers package: model_example_controller.

You can base your own controller on one of the franka_example_controllers. To compute kinematic and dynamic quantities of the robot you can use the joint states and the URDF of the robot in libraries like KDL (of which there is also a ROS 2 package available).

Non-realtime robot parameter setting

Non-realtime robot parameter setting can be done via ROS 2 services. They are advertised after the robot hardware is initialized.

Service names are given below:

```
* /service_server/set_cartesian_stiffness
* /service_server/set_force_torque_collision_behavior
* /service_server/set_full_collision_behavior
* /service_server/set_joint_stiffness
* /service_server/set_load
* /service_server/set_parameters
* /service_server/set_parameters_atomically
* /service_server/set_stiffness_frame
* /service_server/set_tcp_frame
```

Service message descriptions are given below.

- `franka_msgs::srv::SetJointStiffness` specifies joint stiffness for the internal controller (damping is automatically derived from the stiffness).
- `franka_msgs::srv::SetCartesianStiffness` specifies Cartesian stiffness for the internal controller (damping is automatically derived from the stiffness).
- `franka_msgs::srv::SetTCPFrame` specifies the transformation from <arm_id>_EE (end effector) to <arm_id>_NE (nominal end effector) frame. The transformation from flange to end effector frame is split into two transformations: <arm_id>_EE to <arm_id>_NE frame and <arm_id>_NE to <arm_id>_link8 frame. The transformation from <arm_id>_NE to <arm_id>_link8 frame can only be set through the administrator's interface.
- `franka_msgs::srv::SetStiffnessFrame` specifies the transformation from <arm_id>_K to <arm_id>_EE frame.
- `franka_msgs::srv::SetForceTorqueCollisionBehavior` sets thresholds for external Cartesian wrenches to configure the collision reflex.
- `franka_msgs::srv::SetFullCollisionBehavior` sets thresholds for external forces on Cartesian and joint level to configure the collision reflex.
- `franka_msgs::srv::SetLoad` sets an external load to compensate (e.g. of a grasped object).

Launch franka_bringup/franka.launch.py file to initialize robot hardware:

```
ros2 launch franka_bringup franka.launch.py robot_ip:=<fci-ip>
```

Here is a minimal example:

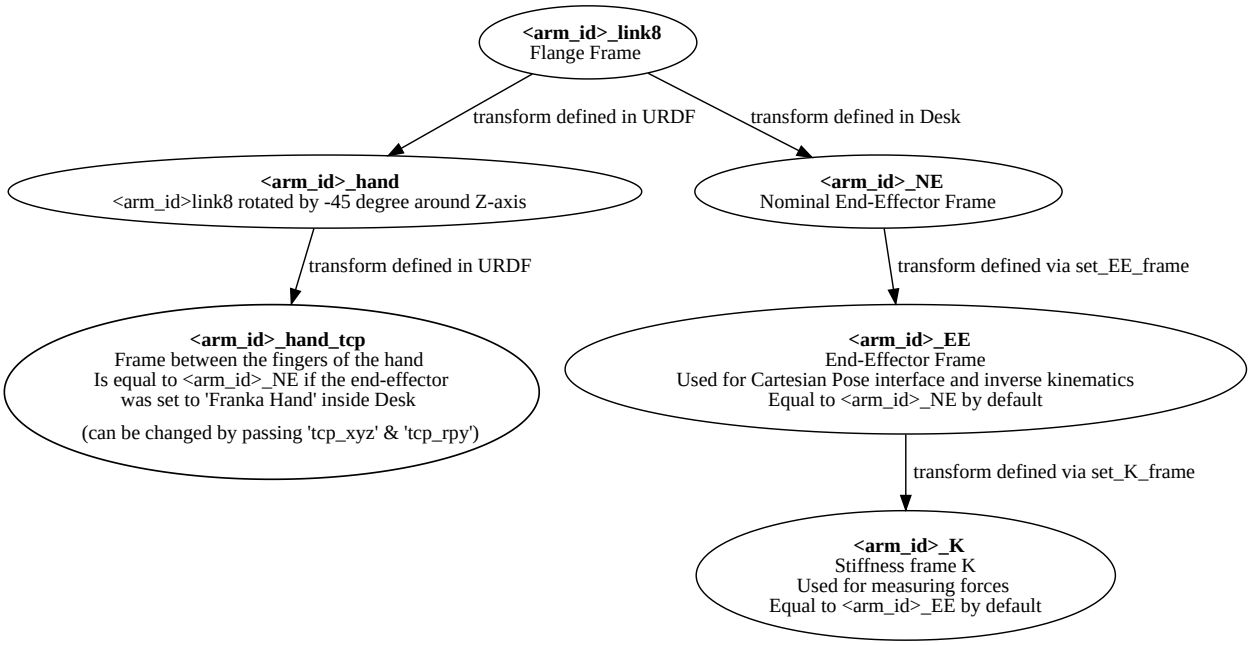
```
ros2 service call /service_server/set_joint_stif
fness franka_msgs/srv/SetJointStiffness "{joint_stiffness: [1000.0, 1000.0, 10
00.0, 1000.0, 1000.0, 1000.0, 1000.0]}"
```

Important

Non-realtime parameter setting can only be done when the robot hardware is in *idle* mode. If a controller is active and claims command interface this will put the robot in the *move* mode. In *move* mode non-realtime param setting is not possible.

Important

The <arm_id>_EE frame denotes the part of the configurable end effector frame which can be adjusted during run time through *franka_ros*. The <arm_id>_K frame marks the center of the internal Cartesian impedance. It also serves as a reference frame for external wrenches. *Neither the <arm_id>_EE nor the <arm_id>_K are contained in the URDF as they can be changed at run time*. By default, <arm_id> is set to "panda".



Overview of the end-effector frames.

Non-realtime ROS 2 actions

Non-realtime ROS 2 actions can be done via the *ActionServer*. Following actions are available:

- `/action_server/error_recovery` - Recovers automatically from a robot error.

The used messages are:

- `franka_msgs::action::ErrorRecovery` - no parameters.

Example usage::

```
ros2 action send_goal /action_server/error_recovery franka_msgs/action/ErrorRecovery
{}
```

Known Issues

- When using the `fake_hardware` with MoveIt, it takes some time until the default position is applied.