franka_ros_interface Documentation

Release 1.0.0-beta

Saif Sidhik

Aug 11, 2020

Contents:

A ROS interface library for the Franka Emika Panda robot, extending the franka_ros to expose more information about the robot, and providing low-level control of the robot using ROS and Python API.

Provides utilites for controlling and managing the Franka Emika Panda robot (real and simulated). Contains exposed controllers for the robot (joint position, velocity, torque), interfaces for the gripper, controller manager, coordinate frames interface, etc. Also provides utilities to control the robot using MoveIt! and ROS Trajectory Action & ActionClient. This package also provides almost complete sim-to-real / real-to-sim transfer of code with the panda_simulator package.

Features

- Low-level controllers (joint position, velocity, torque, impedance) available that can be controlled through ROS topics and Python API (including position control for gripper).
- Real-time robot state (end-effector state, joint state, controller state, etc.) available through ROS topics and Python API.
- Python API for managing controllers, coordinate frames, collision behaviour, controlling and monitoring the gripper.
- Python API classes and utility functions to control the robot using MoveIt! and ROS Trajectory Action Service.
- The panda_simulator package (which is Gazebo-based simulator for the robot) can also be controlled using this package (ROS and Python interface), providing almost complete sim-to-real transfer of code.

Go to Project Source Code.

1 Setup Instructions

1.1 Installation

ROS Kinetic / Melodic: 

NOTE: Tested on Ubuntu 18.04 with ROS Melodic. Version for ROS Kinetic is not maintained anymore. The latest updates to the package may not be compatible with Kinetic.

1.1.1 Dependencies

- libfranka (`sudo apt install ros-$ROS_DISTRO-libfranka` or install from source).
- franka-ros (`sudo apt install ros-$ROS_DISTRO-franka-ros` or install from source)
- (optional, but recommended) franka_panda_description (See Related Packages section for information about package). **NOTE**: If you do not want to use the franka_panda_description package, make sure you modify the `franka_interface/launch/interface.launch` file and replace all occurences of `franka_panda_description` with `franka_description` (two occurences).

Once the above dependencies are installed, the package can be installed using catkin:

```
$ cd <catkin_ws>
$ git clone https://github.com/justagist/franka_ros_interface src/franka_ros_
↪interface
$ catkin build franka_ros_interface # or catkin_make
$ source devel/setup.bash
```

After building the package:

- Copy/move the franka.sh file to the root of the catkin_ws `$ cp src/franka_ros_interface/franka.sh ./`
- Change the values in the copied file (described in the file).

1.2 Usage

The 'driver' node can be started by running (can only be used if run in 'master' environment - see Environments section below):

```
$ roslaunch franka_interface interface.launch # (use argument load_
↪gripper:=false for starting without gripper)
```

This exposes a variety of ROS topics and services for communicating with and controlling the robot. This can be accessed and modified using ROS topics and services (see below too find out about some of the available topics and services), or using the provided Python API.

Basic usage of the API is shown in the test_robot.py example file. See documentation for all available methods and functionalities.

1.2.1 The franka.sh environments

Once the values are correctly modified in the `franka.sh` file, different environments can be set for controlling the robot by sourcing this file.

- For instance, running `./franka.sh master` would start an environment assuming that the computer is directly connected to the robot (requires Real-Time kernel set up as described in the FCI documentation).
- On the other hand, `./franka.sh slave` would start an environment assuming that the robot is not connected directly to the computer, but to another computer in the network (whose IP must be specified in franka.sh). This way, if the 'master' is connected to the robot and running the driver node (see below), the 'slave' can control the robot (**no need for Real Time kernel!**) as long as they are in the same network.
- Simulation environment can be started by running `./franka.sh sim` (only required when using panda_simulator package).

1.2.2 Some useful ROS topics

1.2.2.1 Published Topics:

ROS Topic	Data
/franka_ros_interface/custom_franka_state_controller/robot_state	gravity, coriolis, jacobian, cartesian velocity, etc.
/franka_ros_interface/custom_franka_state_controller/tip_state	end-effector pose, wrench, etc.
/franka_ros_interface/joint_states	joint positions, velocities, efforts
/franka_ros_interface/franka_gripper/joint_states	joint positions, velocities, efforts of gripper joints

1.2.2.2 Subscribed Topics:

ROS Topic	Data
/franka_ros_interface/motion_controller/arm/joint_commands	command the robot using the currently active controller
/franka_ros_interface/franka_gripper/[move/grasp/stop/homing]	(action msg) command the joints of the gripper

Other topics for changing the controller gains (also dynamically configurable), command timeout, etc. are also available.

1.2.3 ROS Services:

Controller manager service can be used to switch between all available controllers (joint position, velocity, effort). Gripper joints can be controlled using the ROS ActionClient. Other services for changing coordinate frames, adding gripper load configuration, etc. are also available.

1.2.4 Python API

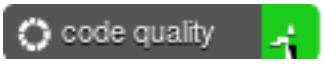
Python API Documentation.

Most of the above services and topics are wrapped using simple Python classes or utility functions, providing more control and simplicity. Refer README files in individual subpackages.

1.3 Related Packages

- panda_simulator : A Gazebo simulator for the Franka Emika Panda robot with ROS interface, providing exposed controllers and real-time robot state feedback similar to the real robot when using the franka_ros_interface package. Provides almost complete real-to-sim transfer of code.
- panda_robot : Python interface providing higher-level control of the robot integrated with its gripper control, controller manager, coordinate frames manager, etc. with safety checks and other helper utilities. It also provides the kinematics and dynamics of the robot using the KDL library.
- franka_panda_description : Robot description package modified from franka_ros package to include dynamics parameters for the robot arm (as estimated in this paper). Also includes transmission and control definitions required for the panda_simulator package.

2 Python API Documentation



2.1 franka_interface

2.1.1 ArmInterface

- Interface class that can monitor and control the robot
- Provides all required information about robot state and end-effector state
- Joint positions, velocities, and effort can be directly controlled and monitored using available methods
- Smooth interpolation of joint positions possible
- End-effector and Stiffness frames can be directly set (uses FrankaFramesInterface from franka_ros_interface/franka_tools)

class franka_interface.**ArmInterface**(synchronous_pub=False)
Bases: object

Interface Class for an arm of Franka Panda robot Constructor.

Parameters **synchronous_pub** (bool) – designates the JointCommand Publisher as Synchronous if True and Asynchronous if False.

Synchronous Publishing means that all joint_commands publishing to the robot's joints will block until the message has been serialized into a buffer and that buffer has been written to the transport of every current Subscriber. This yields predicable and consistent timing of messages being delivered from this Publisher. However, when using this mode, it is possible for a blocking Subscriber to prevent the joint_command functions from exiting. Unless you need exact JointCommand timing, default to Asynchronous Publishing (False).

class RobotMode
Bases: enum.IntEnum

Enum class for specifying and retrieving the current robot mode.

endpoint_effort()
Return Cartesian endpoint wrench {force, torque}.

Return type dict({str:np.ndarray (shape:(3,)),str:np.ndarray (shape:(3,))})

Returns

force and torque at endpoint as named tuples in a dict

- 'force': Cartesian force on x,y,z axes in np.ndarray format
- 'torque': Torque around x,y,z axes in np.ndarray format

endpoint_pose()
Return Cartesian endpoint pose {position, orientation}.

Return type dict({str:np.ndarray (shape:(3,)), str:quaternion.quaternion})

Returns

position and orientation as named tuples in a dict

- 'position': np.array of x, y, z
- 'orientation': quaternion x,y,z,w in quaternion format

endpoint_velocity()
Return Cartesian endpoint twist {linear, angular}.

Return type dict({str:np.ndarray (shape:(3,)),str:np.ndarray (shape:(3,))})

Returns

linear and angular velocities as named tuples in a dict

- 'linear': np.array of x, y, z
- 'angular': np.array of x, y, z (angular velocity along the axes)

error_in_current_state()
Return True if the specified limb has experienced an error.

Return type bool

Returns True if the arm has error, False otherwise.

exit_control_mode(timeout=0.2)
Clean exit from advanced control modes (joint torque or velocity).

Resets control to joint position mode with current positions.

@type timeout: float @param timeout: control timeout in seconds [0.2]

get_controller_manager()

Returns the FrankaControllerManagerInterface instance associated with the robot.

Return type franka_tools.FrankaControllerManagerInterface

get_frames_interface()

Returns the FrankaFramesInterface instance associated with the robot.

Return type franka_tools.FrankaFramesInterface

get_joint_limits()
Return the joint limits (defined in the parameter server)

Return type franka_core_msgs.msg.JointLimits

Returns JointLimits

get_movegroup_interface()

Returns the movegroup interface instance associated with the robot.

Return type franka_moveit.PandaMoveGroupInterface

get_robot_params()

Returns Useful parameters from the ROS parameter server.

Return type franka_interface.RobotParams

get_robot_status()
Return dict with all robot status information.

Return type dict

Returns ['robot_mode' (RobotMode object), 'robot_status' (bool), 'errors' (dict() of errors and their truth value), 'error_in_curr_status' (bool)]

gravity_comp()
Return gravity compensation torques.

Return type np.ndarray

Returns 7D joint torques compensating for gravity.

has_collided()
Returns true if either joint collision or cartesian collision is detected. Collision thresholds can be set using instance of `franka_tools.CollisionBehaviourInterface`.

in_safe_state()
Return True if the specified limb is in safe state (no collision, reflex, errors etc.).

Return type bool

Returns True if the arm is in safe state, False otherwise.

joint_angle(joint)
Return the requested joint angle.

Parameters **joint** (`str`) – name of a joint

Return type float

Returns angle in radians of individual joint

joint_angles()
Return all joint angles.

Return type dict({str:float})

Returns unordered dict of joint name Keys to angle (rad) Values

joint_effort(joint)
Return the requested joint effort.

Parameters **joint** (`str`) – name of a joint

Return type float

Returns effort in Nm of individual joint

joint_efforts()
Return all joint efforts.

Return type dict({str:float})

Returns unordered dict of joint name Keys to effort (Nm) Values

joint_inertia_matrix()

Returns joint inertia matrix (7,7)

Return type np.ndarray [7x7]

joint_names()
Return the names of the joints for the specified limb.

Return type [str]

Returns ordered list of joint names from proximal to distal (i.e. shoulder to wrist).

joint_ordered_angles()
Return all joint angles.

Return type [float]

Returns joint angles (rad) orded by joint_names from proximal to distal (i.e. shoulder to wrist).

joint_velocities()
Return all joint velocities.

Return type dict({str:float})

Returns unordered dict of joint name Keys to velocity (rad/s) Values

joint_velocity(joint)
Return the requested joint velocity.

Parameters **joint** (str) – name of a joint

Return type float

Returns velocity in radians/s of individual joint

move_to_joint_positions(positions, timeout=10.0, threshold=0.00085, test=None, use_moveit=True)
(Blocking) Commands the limb to the provided positions. Waits until the reported joint state matches that specified. This function uses a low-pass filter using JointTrajectoryService to smooth the movement or optionally uses MoveIt! to plan and execute a trajectory.

Parameters

- **positions** (dict({str:float})) – joint_name:angle command
- **timeout** (float) – seconds to wait for move to finish [15]
- **threshold** (float) – position threshold in radians across each joint when move is considered successful [0.00085]
- **test** – optional function returning True if motion must be aborted
- **use_moveit** (bool) – if set to True, and movegroup interface is available, move to the joint positions using moveit planner.

move_to_neutral(timeout=15.0, speed=0.15)
Command the Limb joints to a predefined set of "neutral" joint angles. From rosparam /franka_control/neutral_pose.

Parameters

- **timeout** (float) – seconds to wait for move to finish [15]
- **speed** (float) – ratio of maximum joint speed for execution default= 0.15; range= [0.0-1.0]

reset_EE_frame()
Reset EE frame to default. (defined by FrankaFramesInterface.DEFAULT_TRANSFORMATIONS.EE_FRAME global variable defined in franka_tools.FrankaFramesInterface source code)

Return type [bool, str]

Returns [success status of service request, error msg if any]

set_EE_frame(frame)
Set new EE frame based on the transformation given by 'frame', which is the transformation matrix defining the new desired EE frame with respect to the flange frame. Motion controllers are stopped for switching

Parameters **frame** ([float (16,)] / np.ndarray (4x4)) – transformation matrix of new EE frame wrt flange frame (column major)

Return type [bool, str]

Returns [success status of service request, error msg if any]

set_EE_frame_to_link(frame_name, timeout=5.0)
Set new EE frame to the same frame as the link frame given by 'frame_name' Motion controllers are stopped for switching

Parameters **frame_name** (str) – desired tf frame name in the tf tree

Return type [bool, str]

Returns [success status of service request, error msg if any]

set_collision_threshold(cartesian_forces=None, joint_torques=None)
Set Force Torque thresholds for deciding robot has collided.

Returns True if service call successful, False otherwise

Return type bool

Parameters

- **cartesian_forces** ([float] size 6) – Cartesian force threshold for collision detection [x,y,z,R,P,Y] (robot motion stops if violated)
- **joint_torques** ([float] size 7) – Joint torque threshold for collision (robot motion stops if violated)

set_command_timeout(timeout)
Set the timeout in seconds for the joint controller

Parameters **timeout** (float) – timeout in seconds

set_joint_position_speed(speed=0.3)
Set ratio of max joint speed to use during joint position moves (only for move_to_joint_positions).

Set the proportion of maximum controllable velocity to use during joint position control execution. The default ratio is 0.3, and can be set anywhere from [0.0-1.0] (clipped). Once set, a speed ratio will persist until a new execution speed is set.

Parameters **speed** (float) – ratio of maximum joint speed for execution default= 0.3; range= [0.0-1.0]

set_joint_positions(positions)
Commands the joints of this limb to the specified positions.

Parameters **positions** (dict({str:float}) – dict of {'joint_name':joint_position,}

set_joint_positions_velocities(positions, velocities)
Commands the joints of this limb using specified positions and velocities using impedance control. Command at time t is computed as:

$u_t = coriolis\_factor * coriolis\_t + K\_p * (positions - curr\_positions) + K\_d * (velocities - curr\_velocities)$

Parameters

- **positions** ([float]) – desired joint positions as an ordered list corresponding to joints given by self.joint_names()
- **velocities** ([float]) – desired joint velocities as an ordered list corresponding to joints given by self.joint_names()

set_joint_torques(torques)
Commands the joints of this limb with the specified torques.

Parameters **torques** (dict({str:float})) – dict of {'joint_name':joint_torque,}

set_joint_velocities(velocities)
Commands the joints of this limb to the specified velocities.

Parameters **velocities** (dict({str:float})) – dict of {'joint_name':joint_velocity,}

tip_states()
Return Cartesian endpoint state for a given tip name

Return type TipState object

Returns pose, velocity, effort, effort_in_K_frame

what_errors()
Return list of error messages if there is error in robot state

Return type [str]

Returns list of names of current errors in robot state

zero_jacobian()

Returns end-effector jacobian (6,7)

Return type np.ndarray [6x7]

2.1.2 GripperInterface

- Interface class to monitor and control gripper

- Gripper open, close methods
- Grasp, move joints methods

class franka_interface.**GripperInterface**(gripper_joint_names=('panda_finger_joint1', 'panda_finger_joint2'), calibrate=False, **kwargs)

Bases: object

Interface class for the gripper on the Franka Panda robot.

Parameters

- **gripper_joint_names** ([str]) – Names of the finger joints
- **ns** (str) – base namespace of interface ('frank_ros_interface'/'panda_simulator')
- **calibrate** (bool) – Attempts to calibrate the gripper when initializing class (defaults True)

close()

close gripper to till collision is detected. Note: This is not exactly doing what it should. The behaviour is faked by catching the error thrown when trying to grasp a very small object with a very small force. Since the gripper will actually hit the object before it reaches the commanded width, we catch the feedback and send the gripper stop command to stop it where it is.

Returns True if command was successful, False otherwise.

Return type bool

exists

Check if a gripper was identified as connected to the robot.

Returns True if gripper was detected, False otherwise

Return type bool

grasp(width, force, speed=None, epsilon_inner=0.005, epsilon_outer=0.005, wait_for_result=True, cb=None)

Grasps an object.

An object is considered grasped if the distance d between the gripper fingers satisfies $(ext{width} - ext{epsilon_inner}) < d < (ext{width} + ext{epsilon_outer})$.

Parameters

- **width** (float) – Size of the object to grasp. [m]
- **speed** (float) – Closing speed. [m/s]
- **force** (float) – Grasping force. [N]
- **epsilon_inner** (float) – Maximum tolerated deviation when the actual grasped width is smaller than the commanded grasp width.
- **epsilon_outer** (float) – Maximum tolerated deviation when the actual grasped width is wider than the commanded grasp width.
- **cb** – Optional callback function to use when the service call is done

Returns True if an object has been grasped, false otherwise.

Return type bool

home_joints(wait_for_result=False)
Performs homing of the gripper.

After changing the gripper fingers, a homing needs to be done. This is needed to estimate the maximum grasping width.

Parameters **wait_for_result** (bool) – if True, this method will block till response is recieved from server

Returns success

Return type bool

joint_effort(joint)
Return the requested joint effort.

Parameters **joint** (str) – name of a joint

Return type float

Returns effort in Nm of individual joint

joint_efforts()
Return all joint efforts.

Return type dict({str:float})

Returns unordered dict of joint name Keys to effort (Nm) Values

joint_names()
Return the names of the joints for the specified limb.

Return type [str]

Returns ordered list of joint names.

joint_ordered_efforts()
Return all joint efforts.

Return type [double]

Returns joint efforts ordered by joint_names.

joint_ordered_positions()
Return all joint positions.

Return type [double]

Returns joint positions ordered by joint_names.

joint_ordered_velocities()
Return all joint velocities.

Return type [double]

Returns joint velocities ordered by joint_names.

joint_position(joint)
Return the requested joint position.

Parameters **joint** (str) – name of a joint

Return type float

Returns position individual joint

joint_positions()
Return all joint positions.

Return type dict({str:float})

Returns unordered dict of joint name Keys to pos

joint_velocities()
Return all joint velocities.

Return type dict({str:float})

Returns unordered dict of joint name Keys to velocity (rad/s) Values

joint_velocity(joint)
Return the requested joint velocity.

Parameters **joint** (str) – name of a joint

Return type float

Returns velocity in radians/s of individual joint

move_joints(width, speed=None, wait_for_result=True)
Moves the gripper fingers to a specified width.

Parameters

- **width** (float) – Intended opening width. [m]
- **speed** (float) – Closing speed. [m/s]
- **wait_for_result** (bool) – if True, this method will block till response is recieved from server

Returns True if command was successful, False otherwise.

Return type bool

open()
Open gripper to max possible width.

Returns True if command was successful, False otherwise.

Return type bool

set_velocity(value)
Set default value for gripper joint motions. Used for move and grasp commands.

Parameters **value** (float) – speed value [m/s]

stop_action()
Stops a currently running gripper move or grasp.

Returns True if command was successful, False otherwise.

Return type bool

2.1.3 RobotEnable

- Interface class to reset robot when in recoverable error (use enable_robot.py script in scripts/)

class franka_interface.**RobotEnable**(robot_params=None)
Bases: object

Class RobotEnable - simple control/status wrapper around robot state

enable() - enable all joints disable() - disable all joints reset() - reset all joints, reset all jrcp faults, disable the robot stop() - stop the robot, similar to hitting the e-stop button

Parameters **robot_params** (RobotParams) – A RobotParams instance (optional)

disable()
Disable all joints

enable()
Enable all joints

is_enabled()
Return status of robot

Returns True if enabled, False otherwise

Return type bool

state()
Returns the last known robot state.

Return type str

Returns "Enabled"/"Disabled"

2.1.4 RobotParams

- Collects and stores all useful information about the robot from the ROS parameter server

class franka_interface.**RobotParams**
Bases: object

Interface class for essential ROS parameters on Intera robot.

get_gripper_joint_names()
Return the names of the joints for the gripper from ROS parameter server (/gripper_config/joint_names).

Return type [str]

Returns ordered list of joint names

get_joint_limits()
Get joint limits as defined in ROS parameter server (/robot_config/joint_config/joint_velocity_limit)

Returns Joint limits for each joints

Return type franka_core_msgs.msg.JointLimits

get_joint_names()
Return the names of the joints for the robot from ROS parameter server (/robot_config/joint_names).

Return type [str]

Returns ordered list of joint names from proximal to distal (i.e. shoulder to wrist). joint names for limb

get_neutral_pose()
Get neutral pose joint positions from parameter server (/robot_config/neutral_pose)

Returns Joint positions of the robot as defined in parameter server.

Return type [type]

get_robot_name()
Return the name of class of robot from ROS parameter server. (/robot_config/arm_id)

Return type str

Returns name of the robot

2.2 franka_moveit

2.2.1 PandaMoveGroupInterface

- Provides interface to control and plan motions using MoveIt in ROS.
- Simple methods to plan and execute joint trajectories and cartesian path.
- Provides easy reset and environment definition functionalities (See ExtendedPlanningSceneInterface below).

class franka_moveit.**PandaMoveGroupInterface**

arm_group

Getter The MoveGroupCommander instance of this object. This is an interface to one group of joints. In this case the group is the joints in the Panda arm. This interface can be used to plan and execute motions on the Panda.

Type moveit_commander.MoveGroupCommander

Note: For available methods for movegroup, refer MoveGroupCommander.

---

close_gripper(wait=False)
Close gripper. (Using named states defined in urdf.)

Parameters **wait** (bool) – if set to True, blocks till execution is complete

---

Note: If this named state is not found, your ros environment is probably not using the right panda_moveit_config package. Ensure that sourced package is from this repo –> https://github.com/justagist/panda_moveit_config

---

display_trajectory(plan)
Display planned trajectory in RViz. Rviz should be open and Trajectory display should be listening to the appropriate trajectory topic.

Parameters **plan** – the plan to be executed (from plan_joint_path() or plan_cartesian_path())

execute_plan(plan, group='arm', wait=True)
Execute the planned trajectory

Parameters

- **plan** – The plan to be executed (from `plan_joint_path()` or `plan_cartesian_path()`)
- **group** (`str`) – The name of the move group (default "arm" for robot; use "hand" for gripper group)
- **wait** (`bool`) – if set to True, blocks till execution is complete

go_to_joint_positions(positions, wait=True, tolerance=0.005)

Returns status of joint motion plan execution

Return type bool

Parameters

- **positions** (`[double]`) – target joint positions (ordered)
- **wait** (`bool`) – if True, function will wait for trajectory execution to complete
- **tolerance** (`double`) – maximum error in final position for each joint to consider task a success

gripper_group

Getter The MoveGroupCommander instance of this object. This is an interface to one group of joints. In this case the group is the joints in the Panda arm. This interface can be used to plan and execute motions on the Panda.

Type moveit_commander.MoveGroupCommander

Note: For available methods for movegroup, refer MoveGroupCommander.

---

move_to_neutral(wait=True)
Send arm group to neutral pose defined using named state in urdf.

Parameters **wait** (`bool`) – if set to True, blocks till execution is complete

open_gripper(wait=False)
Open gripper. (Using named states defined in urdf)

Parameters **wait** (`bool`) – if set to True, blocks till execution is complete

---

Note: If this named state is not found, your ros environment is probably not using the right panda_moveit_config package. Ensure that sourced package is from this repo –> https://github.com/justagist/panda_moveit_config.

---

plan_cartesian_path(poses)
Plan cartesian path using the provided list of poses.

Parameters **poses** (`[geomentry_msgs.msg.Pose]`) – The cartesian poses to be achieved in sequence. (Use `franka_moveit.utils.create_pose_msg()` for creating pose messages easily)

Returns the actual RobotTrajectory (can be used for `execute_plan()`), a fraction of how much of the path was followed

Return type [RobotTrajectory, float (0,1)]

plan_joint_path(joint_position)

Returns RobotTrajectory plan for executing joint trajectory (can be used for `execute_plan()`)

Parameters **joint_position** ([float]*7) – target joint positions

robot_state_interface

Getter The RobotCommander instance of this object

Type moveit_commander.RobotCommander

Note: For available methods for RobotCommander, refer RobotCommander.

---

scene

Getter The PlanningSceneInterface instance for this robot. This is an interface to the world surrounding the robot

Type franka_moveit.ExtendedPlanningSceneInterface

Note: For other available methods for planning scene interface, refer PlanningSceneInterface.

---

set_velocity_scale(value, group='arm')
Set the max velocity scale for executing planned motion.

Parameters **value** (float) – scale value (allowed (0,1])

2.2.1.1 Helper Functions

franka_moveit.utils.**create_pose_msg**(position, orientation)
Create Pose message using the provided position and orientation

Returns Pose message for the give end-effector position and orientation

Return type geometry_msgs.msg.Pose

Parameters

- **position** ([float]*3) – End-effector position in base frame of the robot
- **orientation** (quaternion.quaternion (OR) [float] size 4: (w,x, y,z)) – orientation quaternion of end-effector in base frame

franka_moveit.utils.**create_pose_stamped_msg**(position, orientation, frame='world')
Create PoseStamped message using the provided position and orientation

Returns Pose message for the give end-effector position and orientation

Return type geometry_msgs.msg.Pose

Parameters

- **position** ([float]*3) – End-effector position in base frame of the robot
- **orientation** (quaternion.quaternion (OR) [float] size 4: (w,x, y,z)) – orientation quaternion of end-effector in base frame

- **frame** (str) – Name of the parent frame

2.2.2 ExtendedPlanningSceneInterface

- Easily define scene for robot motion planning (MoveIt plans will avoid defined obstacles if possible).

class franka_moveit.**ExtendedPlanningSceneInterface**
Bases: moveit_commander.planning_scene_interface.PlanningSceneInterface

---

Note: For other available methods for planning scene interface, refer PlanningSceneInterface.

---

add_box(name, pose, size, timeout=5)
Add object to scene and check if it is created.

Parameters

- **name** (str) – name of object
- **pose** (geometry_msgs.msg.PoseStamped) – desired pose for the box (Use franka_moveit.utils.create_pose_stamped_msg())
- **size** ([float] (len 3)) – size of the box
- **timeout** (float) – time in sec to wait while checking if box is created

remove_box(box_name, timeout=5)
Remove box from scene.

Parameters

- **box_name** (str) – name of object
- **timeout** (float) – time in sec to wait while checking if box is created

2.3 franka_tools

2.3.1 CollisionBehaviourInterface

- Define collision and contact thresholds for the robot safety and contact detection.

class franka_tools.**CollisionBehaviourInterface**
Bases: object

Helper class to set collision and contact thresholds at cartesian and joint levels. (This class has no 'getter' functions to access the currently set collision behaviour values.)

set_collision_threshold(joint_torques=None, cartesian_forces=None)

Returns True if service call successful, False otherwise

Return type bool

Parameters

- **joint_torques** ([float] size 7) – Joint torque threshold for collision (robot motion stops if violated)

- **cartesian_forces** ([float] size 6) – Cartesian force threshold for collision detection [x,y,z,R,P,Y] (robot motion stops if violated)

set_contact_threshold(joint_torques=None, cartesian_forces=None)

Returns True if service call successful, False otherwise

Return type bool

Parameters

- **joint_torques** ([float] size 7) – Joint torque threshold for identifying as contact
- **cartesian_forces** ([float] size 6) – Cartesian force threshold for identifying as contact

set_force_threshold_for_collision(cartesian_force_values)

Returns True if service call successful, False otherwise

Return type bool

Parameters **cartesian_force_values** ([float] size 6) – Cartesian force threshold for collision detection [x,y,z,R,P,Y] (robot motion stops if violated)

set_force_threshold_for_contact(cartesian_force_values)

Returns True if service call successful, False otherwise

Return type bool

Parameters **cartesian_force_values** ([float] size 6) – Cartesian force threshold for contact detection [x,y,z,R,P,Y]

set_ft_contact_collision_behaviour(torque_lower=None, torque_upper=None, force_lower=None, force_upper=None)

Returns True if service call successful, False otherwise

Return type bool

Parameters

- **torque_lower** ([float] size 7) – Joint torque threshold for contact detection
- **torque_upper** ([float] size 7) – Joint torque threshold for collision (robot motion stops if violated)
- **force_lower** ([float] size 6) – Cartesian force threshold for contact detection [x,y,z,R,P,Y]
- **force_upper** ([float] size 6) – Cartesian force threshold for collision detection [x,y,z,R,P,Y] (robot motion stops if violated)

2.3.2 FrankaControllerManagerInterface

- List, start, stop, load available controllers for the robot
- Get the current controller status (commands, set points, controller gains, etc.)
- Update controller parameters through ControllerParamConfigClient (see below)

class franka_tools.**FrankaControllerManagerInterface**(ns='franka_ros_interface', synchronous_pub=False, sim=False)

Bases: object

Parameters

- **synchronous_pub** (bool) – designates the JointCommand Publisher as Synchronous if True and Asynchronous if False.

  Synchronous Publishing means that all joint_commands publishing to the robot's joints will block until the message has been serialized into a buffer and that buffer has been written to the transport of every current Subscriber. This yields predicable and consistent timing of messages being delivered from this Publisher. However, when using this mode, it is possible for a blocking Subscriber to prevent the joint_command functions from exiting. Unless you need exact JointCommand timing, default to Asynchronous Publishing (False).

- **ns** (str) – base namespace of interface ('frank_ros_interface'/'panda_simulator')

- **sim** (bool) – Flag specifying whether the robot is in simulation or not (can be obtained from franka_interface.RobotParams instance)

controller_dict()

Get all controllers as dict

Returns name of the controller to be stopped

Return type dict {'controller_name': ControllerState}

current_controller

Getter Returns the name of currently active controller.

Type str

effort_joint_position_controller

Getter Returns the name of effort-based joint position controller (defined in franka_ros_controllers, and specified in robot_config.yaml). Can be used for changing motion controller using FrankaControllerManagerInterface.set_motion_controller().

Type str

get_controller_config_client(controller_name)

Returns The parameter configuration client object associated with the specified controller

Return type ControllerParamConfigClient obj (if None, returns False)

Parameters **controller_name** (str) – name of controller whose config client is required

get_controller_state()

Get the status of the current controller, including set points, computed command, controller gains etc. See the ControllerStateInfo class (above) parameters for more info.

get_current_controller_config_client()

Returns The parameter configuration client object associated with the currently active controller

Return type ControllerParamConfigClient obj (if None, returns False)

Parameters **controller_name** (str) – name of controller whose config client is required

is_loaded(controller_name)
Check if the given controller is loaded.

Parameters **controller_name** (str) – name of controller whose status is to be checked

Returns True if controller is loaded, False otherwise

Return type bool

is_running(controller_name)
Check if the given controller is running.

Parameters **controller_name** (str) – name of controller whose status is to be checked

Returns True if controller is running, False otherwise

Return type bool

joint_impedance_controller

Getter Returns the name of joint impedance controller (defined in franka_ros_controllers, and specified in robot_config.yaml). Can be used for changing motion controller using FrankaControllerManagerInterface.set_motion_controller().

Type str

joint_position_controller

Getter Returns the name of joint position controller (defined in franka_ros_controllers, and specified in robot_config.yaml). Can be used for changing motion controller using FrankaControllerManagerInterface.set_motion_controller().

Type str

joint_torque_controller

Getter Returns the name of joint torque controller (defined in franka_ros_controllers, and specified in robot_config.yaml). Can be used for changing motion controller using FrankaControllerManagerInterface.set_motion_controller().

Type str

joint_trajectory_controller

Getter Returns the name of joint trajectory controller (defined in franka_ros_controllers, and specified in robot_config.yaml). Can be used for changing motion controller using FrankaControllerManagerInterface.set_motion_controller(). This controller exposes trajectory following service.

Type str

joint_velocity_controller

Getter Returns the name of joint velocity controller (defined in franka_ros_controllers, and specified in robot_config.yaml). Can be used for changing motion controller using FrankaControllerManagerInterface.set_motion_controller().

Type str

list_active_controller_names(only_motion_controllers=False)

Returns List of names active controllers associated to a controller manager namespace.

Return type [str]

Parameters **only_motion_controller** (bool) – if True, only motion controllers are returned

list_active_controllers(only_motion_controllers=False)

Returns List of active controllers associated to a controller manager namespace. Contains both stopped/running controllers, as returned by the list_controllers service, plus uninitialized controllers with configurations loaded in the parameter server.

Return type [ControllerState obj]

Parameters **only_motion_controller** (bool) – if True, only motion controllers are returned

list_controller_names()

Returns List of names all controllers associated to a controller manager namespace.

Return type [str]

Parameters **only_motion_controller** (bool) – if True, only motion controllers are returned

list_controller_types()

Returns List of controller types associated to a controller manager namespace. Contains both stopped/running/loaded controllers, as returned by the list_controller_types service, plus uninitialized controllers with configurations loaded in the parameter server.

Return type [str]

list_controllers()

Returns List of controllers associated to a controller manager namespace. Contains both stopped/running controllers, as returned by the list_controllers service, plus uninitialized controllers with configurations loaded in the parameter server.

Return type [ControllerState obj]

list_loaded_controllers()

Returns List of controller types associated to a controller manager namespace. Contains all loaded controllers, as returned by the list_controller_types service, plus uninitialized controllers with configurations loaded in the parameter server.

Return type [str]

list_motion_controllers()

Returns List of motion controllers associated to a controller manager namespace. Contains both stopped/running controllers, as returned by the list_controllers service, plus uninitialized controllers with configurations loaded in the parameter server.

Return type [ControllerState obj]

load_controller(name)
Loads the specified controller

Parameters **name** (str) – name of the controller to be loaded

set_motion_controller(controller_name)
Set the specified controller as the (only) motion controller

Returns name of currently active controller (can be used to switch back to this later)

Return type str

Parameters **controller_name** (str) – name of controller to start

start_controller(name)
Starts the specified controller

Parameters **name** (str) – name of the controller to be started

stop_controller(name)
Stops the specified controller

Parameters **name** (str) – name of the controller to be stopped

unload_controller(name)
Unloads the specified controller

Parameters **name** (str) – name of the controller to be unloaded

2.3.3 ControllerParamConfigClient

- Get and set the controller parameters (gains) for the active controller

class franka_tools.**ControllerParamConfigClient**(controller_name)
Interface class for updating dynamically configurable paramters of a controller.

Parameters **controller_name** (str) – The name of the controller.

get_config(timeout=5)

Returns the currently set values for all paramters from the server

Return type dict {str : float}

Parameters **timeout** (float) – time to wait before giving up on service request

get_controller_gains(timeout=5)

Returns the currently set values for controller gains from the server

Return type ([float], [float])

Parameters **timeout** (float) – time to wait before giving up on service request

get_joint_motion_smoothing_parameter(timeout=5)

Returns the currently set value for the joint position smoothing parameter from the server.

Return type float

Parameters **timeout** (float) – time to wait before giving up on service request

get_parameter_descriptions(timeout=5)

Returns the description of each parameter as defined in the cfg file from the server.

Return type dict {str : str}

Parameters **timeout** (float) – time to wait before giving up on service request

is_running

Returns True if client is running / server is unavailable; False otherwise

Return type bool

set_controller_gains(k_gains, d_gains=None)

Update the stiffness and damping parameters of the joints for the current controller.

Parameters

- **k_gains** ([float]) – joint stiffness parameters (should be within limits specified in franka documentation; same is also set in franka_ros_controllers/cfg/joint_controller_params.cfg)
- **d_gains** ([float]) – joint damping parameters (should be within limits specified in franka documentation; same is also set in franka_ros_controllers/cfg/joint_controller_params.cfg)

set_joint_motion_smoothing_parameter(value)

Update the joint motion smoothing parameter (only valid for position_joint_position_controller).

Parameters **value** ([float]) – smoothing factor (should be within limit set in franka_ros_controllers/cfg/joint_controller_params.cfg)

start(timeout=5)

Start the dynamic_reconfigure client

Parameters **timeout** (float) – time to wait before giving up on service request

update_config(**kwargs)

Update the config in the server using the provided keyword arguments.

Parameters **kwargs** – These are keyword arguments matching the parameter names in config file: franka_ros_controllers/cfg/joint_controller_params.cfg

2.3.4 FrankaFramesInterface

- Get and Set end-effector frame and stiffness frame of the robot easily
- Set the frames to known frames (such as links on the robot) directly

class franka_tools.**FrankaFramesInterface**
Bases: object

Helper class to retrieve and set EE frames

Has to be updated externally each time franka states is updated. This is done by default within the PandaArm class (panda_robot package: https://github.com/justagist/panda_robot).

EE_frame_already_set(frame)

Returns True if the requested frame is already the current EE frame

Return type bool

Parameters **frame** (np.ndarray (shape: [4,4]), or list (flattened column major 4x4)) – 4x4 transformation matrix representing frame

frames_are_same(frame1, frame2)

Returns True if two transformation matrices are equal

Return type bool

Parameters

- **frame1** (np.ndarray (shape: [4,4]), or list (flattened column major 4x4)) – 4x4 transformation matrix representing frame1
- **frame2** (np.ndarray (shape: [4,4]), or list (flattened column major 4x4)) – 4x4 transformation matrix representing frame2

get_EE_frame(as_mat=False)
Get current EE frame transformation matrix in flange frame

Parameters **as_mat** (bool) – if True, return np array, else as list

Return type [float (16,)] / np.ndarray (4x4)

Returns transformation matrix of EE frame wrt flange frame (column major)

get_K_frame(as_mat=False)
Get current K frame transformation matrix in EE frame

Parameters **as_mat** (bool) – if True, return np array, else as list

Return type [float (16,)] / np.ndarray (4x4)

Returns transformation matrix of K frame wrt EE frame

get_link_tf(frame_name, timeout=5.0, parent='/panda_link8')
Get $4 imes 4$ transformation matrix of a frame with respect to another. :return: $4 imes 4$ transformation matrix :rtype: np.ndarray :param frame_name: Name of the child frame from the TF tree :type frame_name: str :param parent: Name of parent frame (default: '/panda_link8') :type parent: str

reset_EE_frame()
Reset EE frame to default. (defined by DEFAULT_TRANSFORMATIONS.EE_FRAME global variable defined above)

Return type bool

Returns success status of service request

reset_K_frame()
Reset K frame to default. (defined by **DEFAULT_K_** FRAME global variable defined above)

Return type bool

Returns success status of service request

set_EE_frame(frame)
Set new EE frame based on the transformation given by 'frame', which is the transformation matrix defining the new desired EE frame with respect to the flange frame.

Parameters **frame** ([float (16,)] / np.ndarray (4x4)) – transformation matrix of new EE frame wrt flange frame (column major)

Return type bool

Returns success status of service request

set_EE_frame_to_link(frame_name, timeout=5.0)
Set new EE frame to the same frame as the link frame given by 'frame_name' Motion controllers are stopped for switching

Parameters **frame_name** (str) – desired tf frame name in the tf tree

Return type [bool, str]

Returns [success status of service request, error msg if any]

set_K_frame(frame)
Set new K frame based on the transformation given by 'frame', which is the transformation matrix defining the new desired K frame with respect to the EE frame.

Parameters **frame** ([float (16,)] / np.ndarray (4x4)) – transformation matrix of new K frame wrt EE frame

Return type bool

Returns success status of service request

set_K_frame_to_link(frame_name, timeout=5.0)
Set new K frame to the same frame as the link frame given by 'frame_name' Motion controllers are stopped for switching

Parameters **frame_name** (str) – desired tf frame name in the tf tree

Return type [bool, str]

Returns [success status of service request, error msg if any]

2.3.5 JointTrajectoryActionClient

- Command robot to given joint position(s) smoothly. (Uses the FollowJointTrajectory service from ROS control_msgs package)
- Smoothly move to a desired (valid) pose without having to interpolate for smoothness (trajectory interpolation done internally)

class franka_tools.**JointTrajectoryActionClient**(joint_names, controller_name='position_joint_trajectory_controller')

Bases: object

To use this class, the currently active controller for the franka robot should be the "joint_position_trajectory_controller". This can be set using instance of franka_tools.FrankaControllerManagerInterface.

add_point(positions, time, velocities=None)
Add a waypoint to the trajectory.

Parameters

- **positions** ([float]*7) – target joint positions
- **time** (float) – target time in seconds from the start of trajectory to reach the specified goal
- **velocities** ([float]*7) – goal velocities for joints (give atleast 0.0001)

Note: Velocities should be greater than zero (done by default) for smooth motion.

clear()
Clear all waypoints from the current trajectory definition.

result()
Get result from trajectory action server

start()
Execute previously defined trajectory.

stop()
Stop currently executing trajectory.

wait(timeout=15.0)
Wait for trajectory execution result.

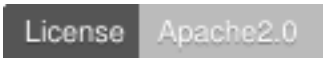
Parameters **timeout** (float) – timeout before cancelling wait

3 Indices and tables

- genindex
- modindex
- search

Go to Project Source Code.

LICENSE:



Python Module Index

f

Index

H

I

J

L

M

O

P

R

S

T

U

W

Z