franka_ros_interface Documentation

Release 0.0.1

Saif Sidhik

Mar 16, 2020

Contents:

1 franka_interface

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)
Interface Class for an arm of Franka Panda robot

class RobotMode
Enum class for specifying and retrieving the current robot mode.

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

@rtype: dict({str:L{Limb.Point},str:L{Limb.Point}}) @return: force and torque at endpoint as named tuples in a dict

C{wrench = {'force': (x, y, z), 'torque': (x, y, z)}}

- '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}.

@rtype: dict({str:L{Limb.Point},str:L{Limb.Quaternion}}) @return: position and orientation as named tuples in a dict

C{pose = {'position': (x, y, z), 'orientation': (x, y, z, w)}}

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

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

@rtype: dict({str:L{Limb.Point},str:L{Limb.Point}}) @return: linear and angular velocities as named tuples in a dict

C{twist = {'linear': (x, y, z), 'angular': (x, y, z)}}

- '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.

@rtype: bool @return: True if the arm has error, False otherwise.

get_robot_status()
Return dict with all robot status information.

@rtype: dict @return: ['robot_mode' (RobotMode object), 'robot_status' (bool), 'errors' (dict() of errors and their truth value), 'error_in_curr_status' (bool)]

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

@rtype: bool @return: True if the arm is in safe state, False otherwise.

joint_angle(joint)
Return the requested joint angle.

@type joint: str @param joint: name of a joint @rtype: float @return: angle in radians of individual joint

joint_angles()
Return all joint angles.

@rtype: dict({str:float}) @return: unordered dict of joint name Keys to angle (rad) Values

joint_effort(joint)

Return the requested joint effort.

_ns @type joint: str @param joint: name of a joint @rtype: float @return: effort in Nm of individual joint

joint_efforts()
Return all joint efforts.

@rtype: dict({str:float}) @return: unordered dict of joint name Keys to effort (Nm) Values

joint_inertia_matrix()
Return joint inertia matrix (7,7)

@rtype: np.ndarray [7x7]

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

@rtype: [str] @return: ordered list of joint names from proximal to distal (i.e. shoulder to wrist).

joint_ordered_angles()
Return all joint angles.

@rtype: [double] @return: joint angles (rad) orded by joint_names from proximal to distal (i.e. shoulder to wrist).

joint_velocities()
Return all joint velocities.

@rtype: dict({str:float}) @return: unordered dict of joint name Keys to velocity (rad/s) Values

joint_velocity(joint)
Return the requested joint velocity.

@type joint: str @param joint: name of a joint @rtype: float @return: 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 to smooth the movement.

@type positions: dict({str:float}) @param positions: joint_name:angle command @type timeout: float @param timeout: seconds to wait for move to finish [15] @type threshold: float @param threshold: position threshold in radians across each joint when move is considered successful [0.008726646] @param test: optional function returning True if motion must be aborted @type use_moveit: bool @param use_moveit: 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.

@type timeout: float @param timeout: seconds to wait for move to finish [15] @type speed: float @param speed: 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 above)

@rtype: [bool, str] @return: [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

@type frame: [float (16,)] / np.ndarray (4x4) @param frame: transformation matrix of new EE frame wrt flange frame (column major) @rtype: [bool, str] @return: [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

@type frame_name: str @param frame_name: desired tf frame name in the tf tree @rtype: [bool, str] @return: [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.

@return True if service call successful, False otherwise @rtype: bool @param cartesian_forces: Cartesian force threshold for collision detection [x,y,z,R,P,Y] (robot motion stops if violated) @type cartesian_forces: [float] size 6 @param joint_torques: Joint torque threshold for collision (robot motion stops if violated) @type joint_torques: [float] size 7

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

@type timeout: float @param timeout: 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.

@type speed: float @param speed: 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.

@type positions: [float] @param positions: ordered joint angles (from joint1 to joint7) to be commanded

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)

@type positions: [float] @param positions: desired joint positions as an ordered list corresponding to joints given by self.joint_names() @type velocities: [float] @param velocities: 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 to the specified torques.

@type torques: dict({str:float}) @param torques: joint_name:torque command

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

@type velocities: dict({str:float}) @param velocities: joint_name:velocity command

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

@rtype: TipState object @return: pose, velocity, effort, effort_in_K_frame

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

@rtype: [str] @return: list of names of current errors in robot state

zero_jacobian()
Return end-effector jacobian (6,7)

@rtype: np.ndarray [6x7]

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'], ns='franka_ros_interface', calibrate=False)
Interface class for the gripper on the Franka Panda robot.

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.

@return True if command was successful, False otherwise. @rtype 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 $(ext{width} - ext{epsilon_inner}) < d < (ext{width} + ext{epsilon_outer})$.

@param width : Size of the object to grasp. [m] @param speed : Closing speed. [m/s] @param force : Grasping force. [N] @param epsilon_inner: Maximum tolerated deviation when the actual grasped width is smaller

than the commanded grasp width.

@param epsilon_outer: Maximum tolerated deviation when the actual grasped width is w
than the commanded grasp width.

@param cb : Optional callback function to use when the service call is done

@type width : float @type speed : float @type force : float @type epsilon_inner : float @type epsilon_outer : float

@return True if an object has been grasped, false otherwise.

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.

@param wait_for_result [if True, this method will block till response is] recieved from server

@type wait_for_result : bool

@return success @rtype bool

joint_effort(joint)
Return the requested joint effort.

@param joint : name of a joint @type joint : str

@rtype: float @return: effort in Nm of individual joint

joint_efforts()
Return all joint efforts.

@rtype: dict({str:float}) @return: unordered dict of joint name Keys to effort (Nm) Values

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

@rtype: [str] @return: ordered list of joint names from proximal to distal (i.e. shoulder to wrist).

joint_ordered_efforts()
Return all joint efforts.

@rtype: [double] @return: joint efforts ordered by joint_names.

joint_ordered_positions()
Return all joint positions.

@rtype: [double] @return: joint positions ordered by joint_names.

joint_ordered_velocities()
Return all joint velocities.

@rtype: [double] @return: joint velocities ordered by joint_names.

joint_position(joint)
Return the requested joint position.

@param joint : name of a joint @type joint : str

@rtype: float @return: position individual joint

joint_positions()
Return all joint positions.

@rtype: dict({str:float}) @return: unordered dict of joint name Keys to pos

joint_velocities()
Return all joint velocities.

@rtype: dict({str:float}) @return: unordered dict of joint name Keys to velocity (rad/s) Values

joint_velocity(joint)
Return the requested joint velocity.

@param joint : name of a joint @type joint : str

@rtype: float @return: velocity in radians/s of individual joint

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

@param width : Intended opening width. [m] @param speed : Closing speed. [m/s] @param wait_for_result : if True, this method will block till response is

recieved from server

@type width : float @type speed : float @type wait_for_result : bool

@return True if command was successful, False otherwise. @rtype bool

open()
Open gripper to max possible width.

@return True if command was successful, False otherwise. @rtype bool

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

@param value : speed value [m/s] @type value : float

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

@return True if command was successful, False otherwise. @rtype bool

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)
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

disable()
Disable all joints

enable()
Enable all joints

state()
Returns the last known robot state.

@rtype: intera_core_msgs/AssemblyState @return: Returns the last received AssemblyState message

1.4 RobotParams

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

class franka_interface.**RobotParams**
Interface class for essential ROS parameters on Intera robot.

get_joint_names()
Return the names of the joints for the specified limb from ROS parameter.

@rtype: list [str] @return: ordered list of joint names from proximal to distal

(i.e. shoulder to wrist). joint names for limb

get_robot_name()
Return the name of class of robot from ROS parameter.

@rtype: str @return: name of the robot

2

franka_moveit

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

@return: 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.

@rtype: moveit_commander.MoveGroupCommander

available_methods: http://docs.ros.org/jade/api/moveit_commander/html/classmoveit__commander_1_1move__group_1_1MoveGroupCommander.html

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

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

go_to_joint_positions(positions, wait=True, tolerance=0.005)
@return: status of joint motion plan execution @rtype: bool

@param positions: target joint positions (ordered) @param wait: if True, function will wait for trajectory execution to complete @param tolerance: maximum error in final position for each joint to consider

task a success

@type positions: [double] @type wait: bool @type tolerance: double

gripper_group

@return: 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.

@rtype: moveit_commander.MoveGroupCommander

available_methods: http://docs.ros.org/jade/api/moveit_commander/html/classmoveit__commander_1_1move__group_1_1MoveGroupCommander.html

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

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

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_joint_path(joint_position)
@return plan for executing joint trajectory

robot_state_interface
@return: The RobotCommander instance of this object @rype: moveit_commander.RobotCommander

available methods: http://docs.ros.org/jade/api/moveit_commander/html/classmoveit__commander_1_1robot_1_1RobotCommander.html

scene

@return: The RobotCommander instance of this object. This is an interface to the world surrounding the robot

@rype: moveit_commander.RobotCommander

available_methods: http://docs.ros.org/indigo/api/moveit_ros_planning_interface/html/classmoveit_1_1planning__interface_1_1PlanningSceneInterface.html

set_velocity_scale(value, group='arm')
Set the max velocity scale for executing planned motion. @param value: scale value (allowed (0,1])

2.2 ExtendedPlanningSceneInterface

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

class franka_moveit.**ExtendedPlanningSceneInterface**

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

@param name: name of object @param pose: desired pose for the box @param size: size of the box @param timeout: time in sec to wait while checking if box is created

@type name: str @type pose: geometry_msgs.msg.PoseStamped @type size: [float] (len 3) @type timeout: float

3 franka_tools

3.1 CollisionBehaviourInterface

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

class franka_tools.**CollisionBehaviourInterface**

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)

@return True if service call successful, False otherwise @rtype: bool @param joint_torques: Joint torque threshold for collision (robot motion stops if violated) @type joint_torques: [float] size 7 @param cartesian_forces: Cartesian force threshold for collision detection [x,y,z,R,P,Y] (robot motion stops if violated) @type cartesian_forces: [float] size 6

set_contact_threshold(joint_torques=None, cartesian_forces=None)

@return True if service call successful, False otherwise @rtype: bool @param joint_torques: Joint torque threshold for identifying as contact @type joint_torques: [float] size 7 @param cartesian_forces: Cartesian force threshold for identifying as contact @type cartesian_forces: [float] size 6

set_force_threshold_for_collision(cartesian_force_values)

@return True if service call successful, False otherwise @rtype: bool @param cartesian_force_values: Cartesian force threshold for collision detection [x,y,z,R,P,Y] (robot motion stops if violated) @type cartesian_force_values: [float] size 6

set_force_threshold_for_contact(cartesian_force_values)

@return True if service call successful, False otherwise @rtype: bool @param cartesian_force_values: Cartesian force threshold for contact detection [x,y,z,R,P,Y] @type cartesian_force_values: [float] size 6

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

@return True if service call successful, False otherwise @rtype: bool @param torque_lower: Joint torque threshold for contact detection @type torque_lower: [float] size 7 @param torque_upper: Joint torque threshold for collision (robot motion stops if violated) @type torque_upper: [float] size 7 @param force_lower: Cartesian force threshold for contact detection [x,y,z,R,P,Y] @type force_lower: [float] size

6 @param force_upper: Cartesian force threshold for collision detection [x,y,z,R,P,Y] (robot motion stops if violated) @type force_upper: [float] size 6

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)

controller_dict()
Get all controllers as dict

@return: name of the controller to be stopped @rtype: dict {'controller_name': ControllerState}

get_controller_config_client(controller_name)
@return The parameter configuration client object associated with the specified controller @rtype ControllerParamConfigClient obj (if None, returns False)

@param controller_name: name of controller whose config client is required @type controller_name: str

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()
@return The parameter configuration client object associated with the currently active controller @rtype ControllerParamConfigClient obj (if None, returns False)

@param controller_name: name of controller whose config client is required @type controller_name: str

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

@type controller_name: str @param controller_name: name of controller whose status is to be checked @return: True if controller is loaded, False otherwise @rtype: bool

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

@type controller_name: str @param controller_name: name of controller whose status is to be checked @return: True if controller is running, False otherwise @rtype: bool

list_active_controller_names(only_motion_controllers=False)
@return List of names active controllers associated to a controller manager namespace. @rtype [str]

@param only_motion_controller: if True, only motion controllers are returned @type only_motion_controller: bool

list_active_controllers(only_motion_controllers=False)
@return List of active controllers associated to a controller manager namespace. Contains both stopped/running controllers, as returned by the C{list_controllers} service, plus uninitialized controllers with configurations loaded in the parameter server. @rtype [ControllerState obj]

@param only_motion_controller: if True, only motion controllers are returned @type only_motion_controller: bool

list_controller_names()
@return List of names all controllers associated to a controller manager namespace. @rtype [str]

@param only_motion_controller: if True, only motion controllers are returned @type only_motion_controller: bool

list_controller_types()
@return List of controller types associated to a controller manager namespace. Contains both stopped/running/loaded controllers, as returned by the C{list_controller_types} service, plus uninitialized controllers with configurations loaded in the parameter server. @rtype [str]

list_controllers()
@return List of controllers associated to a controller manager namespace. Contains both stopped/running controllers, as returned by the C{list_controllers} service, plus uninitialized controllers with configurations loaded in the parameter server. @rtype [ControllerState obj]

list_loaded_controllers()
@return List of controller types associated to a controller manager namespace. Contains all loaded controllers, as returned by the C{list_controller_types} service, plus uninitialized controllers with configurations loaded in the parameter server. @rtype [str]

list_motion_controllers()
@return List of motion controllers associated to a controller manager namespace. Contains both stopped/running controllers, as returned by the C{list_controllers} service, plus uninitialized controllers with configurations loaded in the parameter server. @rtype [ControllerState obj]

load_controller(name)
Loads the specified controller

@type name: str @param name: name of the controller to be loaded

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

@return: name of currently active controller (can be used to switch back to this later) @rtype: str @type controller_name: str @param controller_name: name of controller to start

start_controller(name)
Starts the specified controller

@type name: str @param name: name of the controller to be started

stop_controller(name)
Stops the specified controller

@type name: str @param name: name of the controller to be stopped

unload_controller(name)
Unloads the specified controller

@type name: str @param name: name of the controller to be unloaded

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.

get_config(timeout=5)
@return the currently set values for all paramters from the server @rtype: dict {str : float}

@param timeout: time to wait before giving up on service request @type timeout: float

get_controller_gains(timeout=5)
@return the currently set values for controller gains from the server @rtype: ([float], [float])

@param timeout: time to wait before giving up on service request @type timeout: float

get_joint_motion_smoothing_parameter(timeout=5)
@return the currently set value for the joint position smoothing parameter from the server. @rtype: float

@param timeout: time to wait before giving up on service request @type timeout: float

get_parameter_descriptions(timeout=5)
@return the description of each parameter as defined in the cfg file from the server. @rtype: dict {str : str}

@param timeout: time to wait before giving up on service request @type timeout: float

is_running
@return True if client is running / server is unavailable; False otherwise @rtype bool

set_controller_gains(k_gains, d_gains=None)
Update the stiffness and damping parameters of the joints for the current controller.

@param k_gains: joint stiffness parameters (should be within limits specified in
franka documentation; same is also set in franka_ros_controllers/cfg/joint_controller_params.cf

@type k_gains: [float] @param d_gains: joint damping parameters (should be within limits specified in

franka documentation; same is also set in franka_ros_controllers/cfg/joint_controller_params.cfg)

@type d_gains: [float]

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

@param value: smoothing factor (should be within limit set in franka_ros_controllers/cfg/joint_controller_params.cfg)

@type value: [float]

start(timeout=5)
Start the dynamic_reconfigure client

@param timeout: time to wait before giving up on service request @type timeout: float

update_config(**kwargs)
Update the config in the server using the provided keyword arguments.

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

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**
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).

Note that all controllers have to be unloaded before switching frames. This has to be done externally (also automatically handled in PandaArm class).

frames_are_same(frame1, frame2)
@return True if two transformation matrices are equal @rtype: bool @param frame1: 4x4 transformation matrix representing frame1 @type frame1: np.ndarray (shape 4x4), or list (flattened column major 4x4) @param frame2: 4x4 transformation matrix representing frame2 @type frame2: np.ndarray (shape 4x4), or list (flattened column major 4x4)

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

@type as_mat: bool @param as_mat: if True, return np array, else as list @rtype: [float (16,)] / np.ndarray (4x4) @return: 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

@type as_mat: bool @param as_mat: if True, return np array, else as list @rtype: [float (16,)] / np.ndarray (4x4) @return: transformation matrix of K frame wrt EE frame

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

@rtype: bool @return: success status of service request

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

@rtype: bool @return: 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.

@type frame: [float (16,)] / np.ndarray (4x4) @param frame: transformation matrix of new EE frame wrt flange frame (column major) @rtype: bool @return: 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

@type frame_name: str @param frame_name: desired tf frame name in the tf tree @rtype: [bool, str] @return: [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.

@type frame: [float (16,)] / np.ndarray (4x4) @param frame: transformation matrix of new K frame wrt EE frame @rtype: bool @return: 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

@type frame_name: str @param frame_name: desired tf frame name in the tf tree @rtype: [bool, str] @return: [success status of service request, error msg if any]

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, ns='franka_ros_interface', controller_name='position_joint_trajectory_controller'

4

Indices and tables

- genindex
- modindex
- search

Python Module Index

f

Index

H

I

J

L

M

O

P

R

S

T

U

W

Z