panda_robot Documentation

Release 0.0.1-beta

Saif Sidhik

Apr 16, 2020

Contents:

A Python interface package built over the franka_ros_interface package for controlling and handling the Franka Emika Panda robot. Also works directly with panda_simulator.

Features

- Provides simple-intuitive interface classes with methods to directly and easily control the robot.
- Get real-time robot state, joint state, controller state, kinematics, dynamics, etc.
- Provides Kinematics computation (using kdl).
- Integrated with gripper control.
- Manage frames transformation and controller switching using simple utility functions
- Works directly on simulated robot when using panda_simulator providing direct sim-to-real and real-to-sim code transfer.

Go to Project Source Code.

1 Python API Documentation

1.1 panda_robot

1.1.1 PandaArm

class panda_robot.**PandaArm**(on_state_callback=None, reset_frames=True)

Constructor class. Methods from franka_interface.ArmInterface are also available to this object.

Bases franka_interface.ArmInterface

Parameters

- **on_state_callback** – optional callback function to run on each state update
- **reset_frames** – if True, EE frame is reset using franka_interface.ArmInteface (using franka_interface.ArmInterface and franka_tools.FrankaFramesInterface.)

angles(include_gripper=False)

Returns current joint positions

Return type [float]

Parameters **include_gripper** (bool) – if True, append gripper joint positions to list

base_link_name()

Returns name of base link frame

Return type str

cartesian_velocity(joint_angles=None)

Get cartesian end-effector velocity. To get velocity from franka_ros_interface directly, use method ee_velocity().

Returns end-effector velocity computed using kdl

Return type np.ndarray

Parameters **joint_angles** ([float]) – joint angles (optional)

ee_pose()

Returns end-effector pose as position and quaternion in global frame

Return type np.ndarray (pose), np.quaternion (orientation)

ee_velocity(real_robot=True)

Returns end effector velocity (linear and angular) computed using finite difference

Return type np.ndarray, np.ndarray

Parameters **real_robot** (bool) – if False, computes ee velocity using finite difference

If real_robot is False, this is a simple finite difference based velocity computation. Please note that this might produce a bug since self._goal_ori_old gets updated only if get_ee_vel is called.

enable_force_torque_transform_to_base_frame(boolval=True)
Enable transformation of force vector to base frame

Parameters **boolval** (bool) – set True to transform forces to base frame

enable_robot()
Re-enable robot if stopped due to collision or safety.

end_effector_link_name()

Returns name of end-effector frame

Return type str

exec_gripper_cmd(pos, force=None)
Move gripper joints to the desired width (space between finger joints), while applying the specified force (optional)

Parameters

- **pos** (float) – desired width [m]
- **force** (float) – desired force to be applied on object [N]

Returns True if command was successful, False otherwise.

Return type bool

exec_position_cmd(cmd)
Execute position control (raw positions). Be careful while using. Send smooth commands

Parameters **cmd** ([float]) – desired joint postions, ordered from joint1 to joint7 (optionally, give desired gripper width as 8th element of list)

exec_position_cmd_delta(cmd)
Execute position control based on desired change in joint position

Parameters **cmd** ([float]) – desired joint postion changes, ordered from joint1 to joint7

exec_torque_cmd(cmd)
Execute torque command at joint level directly

Parameters **cmd** ([float]) – desired joint torques, ordered from joint1 to joint7

exec_velocity_cmd(cmd)
Execute velocity command at joint level (using internal velocity controller)

Parameters **cmd** ([float]) – desired joint velocities, ordered from joint1 to joint7

forward_kinematics(joint_angles=None, ori_type='quat')

Returns position and orientaion of end-effector for the current/provided joint angles

Return type np.ndarray, np.ndarray/np.quaternion

Parameters

- **joint_angles** ([float]) – joint angles (optional) for which the ee pose is to be computed
- **ori_type** – to specify the orientation representation to return

get_gripper()

Returns gripper instance

Return type franka_interface.GripperInterface

gripper_state()
Return Gripper state {'position', 'force'}. Only available if Franka gripper is connected.

Return type dict({str:np.ndarray (shape:(2,)),str:np.ndarray (shape:(2,))})

Returns

dict of position and force

- 'position': np.array
- 'force': np.array

has_gripper

Returns True if gripper is initialised, else False

Return type bool

inertia(joint_angles=None)

Returns inertia matrix of robot at current state

Return type np.ndarray

Parameters **joint_angles** ([float]) – joint angles (optional)

inverse_kinematics(pos, ori=None, seed=None, null_space_goal=None, **kwargs)

Returns get the joint positions using inverse kinematics from the provided end-effector pose

Return type bool (success), [float]

Parameters

- **pos** ([float]) – end-effector position (x,y,z)
- **ori** ([float] or np.quaternion) – end-effector orientation (quaternion)

- **seed** ([float]) – seed joints to start ik computation
- **null_space_goal** ([float]) – null-space joint position if required

kwargs are to avoid breaking of sister classes for arguments that are not used in this class.

jacobian(joint_angles=None)

Returns jacobian matrix of robot at current state

Return type np.ndarray

Parameters **joint_angles** ([float]) – joint angles (optional) for which the jacobian is to be computed

joint_limits()

Returns joint limits

Return type [{'lower': float, 'upper': float}]

move_to_joint_pos_delta(cmd)
Execute motion (trajectory controller) based on desired change in joint position

Parameters **cmd** ([float]) – desired joint postion changes, ordered from joint1 to joint7

move_to_joint_position(joint_angles)
Move to joint position specified (using low-level position control)

Parameters **joint_angles** ([float]) – desired joint positions, ordered from joint1 to joint7

n_cmd()

Returns number of control commands (normally same as number of joints)

Return type int

n_joints()

Returns number of joints

Return type int

q_mean()

Returns mean of joint limits

Return type [float]

set_arm_speed(speed)
Set joint position speed (for joint trajectory controller [move_to_joint_positions] only)

set_gripper_speed(speed)
Set velocity for gripper motion

Parameters **speed** (float) – speed ratio to set

state()

Returns robot state as a dictionary

Return type dict {str: obj}

tip_state()

Returns tip_state dictionary

Return type dict {str: obj}

untuck()
Move to neutral pose (using trajectory controller, or moveit (if moveit is available))

velocities(include_gripper=False)

Returns current joint velocities

Return type [float]

Parameters **include_gripper** (bool) – if True, append gripper joint velocities to list

2 Indices and tables

- genindex
- modindex
- search

Go to Project Source Code.

LICENSE:

Python Module Index

p

Index