Blue Interface Documentation

Rachel Thomasson, Brent Yi

Oct 06, 2020

Contents

Blue Interface is a platform-agnostic Python API for controlling Blue robotic arms over a network connection.

It features:

- No dependency on ROS (or any particular version of Ubuntu)
- Easy connection to multiple robots
- Support for both Python 2 and 3
- Support for Mac, Windows, and Linux
- Support for Jupyter Notebooks

It's designed to be lightweight and easy-to-use! Sending a Blue "right" arm to its zero position, for example, is as simple as:

```
from blue_interface import BlueInterface

blue = BlueInterface(side="right", ip="127.0.0.1")
blue.set_joint_positions([0] * 7)
```

See Github for installation instructions and more usage examples.

---

class blue_interface.**BlueInterface**(*side*, *ip*, *port=9090*)

A Python interface for controlling the Blue robot through rosbridge.

Parameters

- **side** (*str*) – side of the arm, "left" or "right"
- **ip** (*str*) – The IP address of the robot, which by default should have a running rosbridge server.
- **port** (*int, optional*) – The websocket port number for rosbridge. Defaults to 9090.

calibrate_gripper()

Run the gripper position calibration process. This will automatically determine the gripper position by apply a closing torque and detecting when the gripper has fully closed.

cancel_gripper_command()

Cancel current gripper command, halting gripper in current position.

command_gripper(*position*, *effort*, *wait=False*)

Send a goal to gripper, and optionally wait for the goal to be reached.

Parameters

- **position** (*float64*) – gap size between gripper fingers in cm.
- **effort** (*float64*) – maximum effort the gripper with exert before stalling in N.

disable_control()

Set joint control mode to gravity compensation only.

disable_gripper()

Disables the gripper. The gripper will become compliant.

enable_gripper()

Enables the gripper. The gripper will begin to hold position.

get_cartesian_pose()

Get the current cartesian pose of the end effector, with respect to the world frame.

Returns Pose in the form {"position": numpy.array([x,y,z]), "orientation": numpy.array([x,y,z,w]} defined with respect to the world frame.

Return type dict

get_gripper_effort()
Get the current effort exerted by the gripper.

Returns the gripper effort in N

Return type float64

get_gripper_position()
Get the current gap between gripper fingers.

Returns the gripper gap in cm.

Return type float64

get_joint_positions()
Get the current joint angles, in radians.

Returns An array of 7 angles, in radians, ordered from proximal to distal.

Return type numpy.ndarray

get_joint_torques()
Get the current joint torques.

Returns An array of 7 joint torques, in Nm, ordered from proximal to distal.

Return type numpy.ndarray

get_joint_velocities()
Get the current joint velocities.

Returns An array of 7 joint torques, in Nm, ordered from proximal to distal.

Return type numpy.ndarray

gripper_enabled()
Check if gripper is enabled to take commands.

Returns True if enabled, False otherwise.

Return type bool

inverse_kinematics(*position*, *orientation*, *seed_joint_positions=[]*)
Given a desired cartesian pose for the end effector, compute the necessary joint angles. Note that the system is underparameterized and there are an infinite number of possible solutions; this will only return a single possible one.

Parameters

- **position** (*iterable*) – A length-3 array containing a cartesian position (x,y,z), wrt the world frame.
- **orientation** (*iterable*) – A length-4 array containing a quaternion (x,y,z,w), wrt the world frame.
- **seed_joint_positions** (*iterable, optional*) – An array of 7 joint angles, to be used to initalize the IK solver.

Returns An array of 7 joint angles, or an empty array if no solution was found.

Return type numpy.ndarray

set_joint_positions(*joint_positions*, *duration=0.0*, *soft_position_control=False*)
Move arm to specified position in joint space.

Parameters

- **joint_positions** (*iterable*) – An array of 7 joint angles, in radians, ordered from proximal to distal.
- **duration** (*float, optional*) – Seconds to take to reach the target, interpolating in joint space. Defaults to 0.
- **soft_position_control** (*bool, optional*) – Use "software" position control, which runs position control loop at the ROS-level, rather than on the motor drivers. This should be rarely needed. Defaults to False.

set_joint_torques(*joint_torques*)
Command joint torques to the arm.

Parameters

- **joint_torques** (*iterable*) – An array of 7 joint torques, in Nm,
- **from proximal to distal.** (*ordered*) –

shutdown()
Clean up and close connection to host computer. All control will be disabled. This can be called manually, but will also run automatically when your script exits.

Python Module Index

b

Index