UR-Robotiq Integrated Driver Documentation

Release 0.0.1-alpha

Minh Tram

Mar 30, 2022

CONTENTS

The Robotic Vision Laboratory (RVL) UR-Robotiq Integrated Driver is a wrapper for existing ROS services and topics provided by the Universal Robots (UR) ROS Driver. It has additional support for a Robotiq gripper (e.g., 2F) directly attached to the UR tool port. The driver also provides functions for actuating the gripper in various modes (e.g., grasp on open/close, specific jaw gap, etc.) as well as integrating the MoveIt Motion Planning Framework for tracking the entire setup, UR robot, and Robotiq gripper.

At the time of this release, the driver is working on our setup with both Windows 10 (Version 10.0.19044 Build 19044) and Ubuntu 20.04.

CHAPTER

ONE

QUICKSTART

1.1 Universal Robots Setup

1.1.1 Setup RS485 URCap and `socat`

Follow the step-by-step instructions provided by Universal Robots to allow for tool communication through a socket on port 54321.

The control box filesystem can be accessed via `ssh root@[ROBOT-IP]` with the default password `easybot`. You should see the internal UR prompt, i.e.,

```
Universal Robots A/S Linux image

Production image
root@ur-[serial]:~#
```

`socat` can then be installed either from source or using a package manager. You may want to use the attached `socat-robotiq-gripper.service` so that `socat` starts up every time the system is powered on.

To setup the service file, just place it in `/etc/systemd/system/` and run

```
$ sudo systemctl enable socat-robotiq-gripper.service
```

which will start the `socat` service on boot.

1.2 Docker Environment Setup

1.2.1 Installing Docker

```
$ sudo apt-get install docker.io
```

or follow the WSL 2 backend installation guide at Get Docker.

Note: If you are using a Linux distribution, be sure to follow Docker Post-Installation Steps.

1.2.2 Clone the Source Code Repository

```
$ git clone --recurse-submodules https://github.com/robotic-vision-lab/UR-Robotiq-
↪Integrated-Driver
```

1.2.3 Build or Pull the Docker Image

You can build the image using the Dockerfile already included in the repository

```
$ cd UR-Robotiq-Integrated-Driver/docker
$ docker build -t rvl-ur-robotiq-driver -f Dockerfile.noetic .
```

or you can pull it directly from Docker Hub with

```
$ docker pull mqt0029/rvl-ur-robotiq-driver:latest`
```

If you pulled from Docker Hub, then be sure to retag your image to `rvl-ur-robotiq-driver:latest`

```
$ docker tag mqt0029/rvl-ur-robotiq-driver:latest rvl-ur-robotiq-driver:latest
```

or update the image name in the launch script accordingly.

1.2.4 Creating the ROS Container

Note: Be sure to check your image name before running the script.

The included `[windows_]launch_docker_container.sh` is a useful script for creating and accessing docker containers created from the built/pulled image. Simply run

```
$ sh launch_docker_container.sh
```

The first run will create the container if it does not already exist. Subsequent runs will attach the terminal to the docker container via `bash`. At this point, you should see the following prompt

```
[...]
root ~/catkin_ws
>
```

1.3 Inside the Container

With our setup using a UR5e and Robotiq 2F-85 gripper, we have customized the accompanying `xacro` (which also includes the generated `urdf`) to match our specific machine description. You may want to modify the launch file to use the correct files for your own setup.

While the base path is commonly `[...]/catkin_ws/src`, this is not always the case. Paths will be relative to where you place your ROS packages.

1.3.1 Robot Xacro and URDF

The included ROS package `rvl_robot_description` contains an example of our robot setup. The file `[...]/rvl_robot_description/xacro/rvl_ur5e.xacro` can be a good place to start and components should be fairly straightforward to swap in and move around.

1.3.2 Launching the UR Driver

The `ur_robot_driver` package launch file has been modified to work with the tool port gripper and our particular setup. You can use the launch file `[...]/rvl_ur_remote_dashboard/launch/rvl_ur5e_bringup.launch` as a reference, and modify it to fit your setup. The important sections are highlighted below.

```
<arg name="robot_ip" default="192.168.1.147" doc="IP address by which ..."/>
<arg name="robot_description_file" default="$(find rvl_ur_remote_dashboard)/launch/load_
↪rvl_ur5e.launch" doc="Robot description launch file."/>
<arg name="kinematics_config" default="$(find rvl_ur_remote_dashboard)/configs/ur5e_
↪calibration.yaml" doc="Kinematics config file used ..."/>

<!-- always start the UR side socat -->
<arg name="use_tool_communication" default="true" doc="On e-Series robots tool␣
↪communication can be enabled with this argument"/>
```

Warning: Double check that your calibration file is correct or retrieve it accordingly.

Warning: Make sure your robot description is correct! This is *crucial* to generating your own URDF and SRDF that MoveIt! uses to plan all subsequent motions.

1.3.3 Building and Running ROS

A few aliases have been provided to quickly setup ROS dependencies and paths. `bash` will also automatically source `devel/setup.bash`.

```
$ run_rosdep
[...] installing dependencies

$ rebuild_catkin
[...] building catkin workspace

$ roslaunch <your_stuff>
```

1.4 Debugging

Since this driver is built inside a Docker container, the most common issue is networking. A good place to start is to ensure that all the ports (or the network interface) are properly exposed, the robot IP is correct, and there is nothing interfering with ROS communications both ways. Otherwise, the host OS should not be a problem. However, should any issues arise, please open an issue on our GitHub repository.

CHAPTER

TWO

UR-ROBOTIQ INTEGRATED DRIVER API

Due to the way ROS structures Python source code and the workspace setup, the module/package usage terminology may be confusing here.

2.1 Imports

```
from rvl_robotiq_controller.RobotiqController import Robotiq2FController
from rvl_ur_remote_dashboard.URRemoteDashboard import URRemoteDashboard
from rvl_ur_motion_planner.URMoveitCommander import URCommander
```

These are the primary modules and classes that cover most of the use cases. The supporting functions might provide additional insights and are accessible as well.

`Robotiq2FController` is a wrapper to control the Robotiq 2F grippers.

`URRemoteDashboard` is a wrapper for handling services and status monitoring as described on the ROS Interface page in the UR Driver Repository.

`URCommander` is a wrapper for `MoveIt!` with some functions built-in (e.g., pose goal, joint goal, etc.). Additional features are planned for future releases.

2.2 Modules Quick Access

The links below provide access to the modules mentioned under the Import section. Full documentation of the entire driver is described in the next section. This documentation will continue to be updated and cover more modules as the driver is developed.

rvl_robotiq_controller.RobotiqController.Robotiq2FController

rvl_ur_remote_dashboard.URRemoteDashboard.URRemoteDashboard

rvl_ur_motion_planner.URMoveitCommander.URCommander

2.3 Full Modules Documentation

2.3.1 Gripper Control

rvl_robotiq_controller ROS Package

rvl_robotiq_controller package

Submodules

rvl_robotiq_controller.Robotiq2FSupport module

rvl_robotiq_controller.Robotiq2FSupport.**generate_2f_status_from_binary**(*binary*)

rvl_robotiq_controller.Robotiq2FSupport.**generate_binary_command_from_2f_msg**(*message*)

rvl_robotiq_controller.Robotiq2FSupport.**raw_to_rad_2f140**(*raw*)

rvl_robotiq_controller.Robotiq2FSupport.**raw_to_rad_2f85**(*raw*)

rvl_robotiq_controller.Robotiq3FSupport module

rvl_robotiq_controller.RobotiqController module

class rvl_robotiq_controller.RobotiqController.**Robotiq2FController**(*stroke: int*, *default_force: int = 100*, *default_speed: int = 100*, *initialize: bool = False*, *startup_reset: bool = False*, *calibrate: bool = False*, *bypass_power: bool = False*)

Bases: `object`

__init__(*stroke: int*, *default_force: int = 100*, *default_speed: int = 100*, *initialize: bool = False*, *startup_reset: bool = False*, *calibrate: bool = False*, *bypass_power: bool = False*) → None

Controller for basic Robotiq 2F Gripper operation.

Parameters

- **stroke** (*int*) – Stroke of the gripper. 2F grippers must use either 85mm or 140mm.
- **default_force** (*int, optional*) – Default gripping force. Defaults to 100.
- **default_speed** (*int, optional*) – Default gripping speed. Defaults to 100.
- **initialize** (*bool, optional*) – Register ROS publisher and subscriber on object instantiation. Defaults to False.
- **startup_reset** (*bool, optional*) – Invoke internal reset on instantiation. Defaults to False.
- **calibrate** (*bool, optional*) – Adjust binary limits of gripper. Defaults to False.
- **bypass_power** (*bool, optional*) – Ignoring power state monitor. Useful when gripper not attached to UR tool port. Defaults to False.

Raises **ValueError** – Invalid stroke width, must be 85mm or 140mm

activate() → None
[summary]

auto_close(*alt_speed: Optional[int] = None*, *alt_force: Optional[int] = None*, *blocking: bool = True*) → None
Fully close the gripper or until obstructed.

Parameters

- **alt_speed** (*int, optional*) – Override internal speed settings. Defaults to None.
- **alt_force** (*int, optional*) – Override internal force settings. Defaults to None.
- **blocking** (*bool, optional*) – Wait until gripper motion is completed. Defaults to True.

auto_open(*alt_speed: Optional[int] = None*, *alt_force: Optional[int] = None*, *blocking: bool = True*) → None
Fully open the gripper or until obstructed.

Parameters

- **alt_speed** (*int, optional*) – Override internal speed settings. Defaults to None.
- **alt_force** (*int, optional*) – Override internal force settings. Defaults to None.
- **blocking** (*bool, optional*) – Wait until gripper motion is completed. Defaults to True.

block() → None
Convenient snippet for idle looping until gripper finishes motion.

calibrate() → Tuple[int, int]
Fully open and closes the gripper to record internal binary. Must be executed without any obstructions as this can dramatically influence gripper operations!

Returns lower and upper binary limit (default 0x00 to 0xFF or 0 to 255)

Return type (int, int)

compensated_publish(*command: rvl_robotiq_controller.msg._Robotiq2FCommand.Robotiq2FCommand*, *lag: float = 0.1*)
Publish with a small wait between message for ROS topics to cope with the refresh rate of the gripper.

deactivate() → None
Deactivate the gripper. Can also be used to clear the reset bit.

grasp_hard(*opening: bool = False*, *alt_speed: Optional[int] = None*, *blocking: bool = True*) → None
Hard grasp preset, grasping with force set to maximum or 255 (approximately 235 N).

Parameters

- **opening** (*bool, optional*) – Grasp in the opening direction i.e. internal grasp. Defaults to False.
- **alt_speed** (*int, optional*) – Override internal speed settings. Defaults to None.
- **blocking** (*bool, optional*) – Wait until gripper motion is completed. Defaults to True.

grasp_medium(*opening: bool = False*, *alt_speed: Optional[int] = None*, *blocking: bool = True*) → None
Medium grasp preset, grasping with force set to 128 (approximately 128 N).

Parameters

- **opening** (*bool, optional*) – Grasp in the opening direction i.e. internal grasp. Defaults to False.
- **alt_speed** (*int, optional*) – Override internal speed settings. Defaults to None.

- **blocking** (*bool, optional*) – Wait until gripper motion is completed. Defaults to True.

grasp_soft(*opening: bool = False*, *alt_speed: Optional[int] = None*, *blocking: bool = True*) → None
Soft grasp preset, grasping with force set to 1 (approximately 20 N).

Parameters

- **opening** (*bool, optional*) – Grasp in the opening direction i.e. internal grasp. Defaults to False.
- **alt_speed** (*int, optional*) – Override internal speed settings. Defaults to None.
- **blocking** (*bool, optional*) – Wait until gripper motion is completed. Defaults to True.

grasp_soft_regrasp(*opening: bool = False*, *alt_speed: Optional[int] = None*, *blocking: bool = True*) → None
Soft grasp preset, grasping with force set to 1 (approximately 20 N) with Re-Grasp enabled.

Parameters

- **opening** (*bool, optional*) – Grasp in the opening direction i.e. internal grasp. Defaults to False.
- **alt_speed** (*int, optional*) – Override internal speed settings. Defaults to None.
- **blocking** (*bool, optional*) – Wait until gripper motion is completed. Defaults to True.

inps_to_raw_speed(*inps: Union[int, float]*) → int
Convert inches/s to raw binary value [0-255]

is_holding() → bool
Returns true when the gripper is holding an object

is_moving() → bool
Returns true when the gripper is in motion

lbf_to_raw_force(*lbf: Union[int, float]*) → int
Convert lbf to raw binary value [0-255]

mmps_to_raw_speed(*mmps: Union[int, float]*) → int
Convert mm/s to raw binary value [0-255]

newton_to_raw_force(*newton: Union[int, float]*) → int
Convert N to raw binary value [0-255]

open_gripper(*value: Union[int, float]*, *alt_speed: Optional[int] = None*, *alt_force: Optional[int] = None*, *unit: str = 'mm'*, *blocking: bool = True*) → None
Open/Close the gripper to specified gap between the gripper pads.

Parameters

- **value** (*Union[int, float]*) – Width of the jaw opening.
- **alt_speed** (*int, optional*) – Override internal speed settings. Defaults to None.
- **alt_force** (*int, optional*) – Override internal force settings. Defaults to None.
- **unit** (*str, optional*) – Unit of measurement. Defaults to 'mm'.
- **blocking** (*bool, optional*) – Wait until gripper motion is completed. Defaults to True.

open_in_to_raw(*inches: Union[int, float]*) → int
Convert opening in inches to raw binary value [0-255]

open_mm_to_raw(*mm: Union[int, float]*) → int
Convert opening in mm to raw binary value [0-255]

power_monitor_callback(*msg*)
Callback for power subsciber

raw_force_to_lbf(*raw: int*) → float
Convert raw value [0-255] to lbf

raw_force_to_newton(*raw: int*) → float
Convert raw value [0-255] to N

raw_speed_to_inps(*raw: int*) → float
Convert raw value [0-255] to in/s

raw_speed_to_mmps(*raw: int*) → float
Convert raw value [0-255] to mm/s

raw_to_open_in(*raw: int*) → float
Convert raw value [0-255] to in of opening

raw_to_open_mm(*raw: int*) → float
Convert raw value [0-255] to mm of opening

register(*timeout: int = 10*) → None
Register necessary ROS publishers and subscribers.

Parameters **timeout** (*int, optional*) – Wait time for topics to start publishing. Defaults to 10.

report_status(*verbose: bool = False*) → None
Output the current state of the gripper.

Parameters **verbose** (*bool, optional*) – Display additional raw status message. Defaults to False.

reset() → None
Reset the gripper to default state (may or may not be obstructed).

send_raw_position_command(*position: int*, *speed: int*, *force: int*, *blocking: bool = True*) → None
Send a position request command to the gripper ignoring internal settings.

Parameters

- **position** (*int*) – Raw position value [0-255].
- **speed** (*int*) – Raw speed value [0-255].
- **force** (*int*) – Raw force value [0-255].
- **blocking** (*bool, optional*) – Wait until gripper motion is completed. Defaults to True.

set_gripper_force(*value: Union[int, float]*, *unit: str = 'N'*) → None
Set the internal (default) force setting of the gripper.

Parameters

- **value** (*Union[int, float]*) – New force value.
- **unit** (*str, optional*) – Unit of measurement. Defaults to 'N'.

set_gripper_speed(*value: Union[int, float]*, *unit: str = 'mm/s'*) → None
Set the internal (default) speed setting of the gripper.

Parameters

- **value** (*Union[int, float]*) – New speed value.
- **unit** (*str, optional*) – Unit of measurement. Defaults to 'mm/s'.

status_monitor_callback(*msg*)
Callback for status subsciber

class rvl_robotiq_controller.RobotiqController.**Robotiq3FController**
Bases: object

__init__()

rvl_robotiq_modbus_server package

Submodules

rvl_robotiq_modbus_server.RobotiqModbusServer module

class rvl_robotiq_modbus_server.RobotiqModbusServer.**RobotiqRTUClient**(*unit_id=9*, *input_addr=1000*, *output_addr=2000*)
Bases: object

__init__(*unit_id=9*, *input_addr=1000*, *output_addr=2000*)

connect(*device_addr*, *delay=1*)

disconnect()

parse_registers(*recv_regs*, *nregs*)

request_status(*nbytes=6*)

send_command(*command*)

class rvl_robotiq_modbus_server.RobotiqModbusServer.**RobotiqTCPClient**
Bases: object

__init__()

2.3.2 UR Dashboard Control

rvl_ur_remote_dashboard ROS Package

rvl_ur_motion_planner package

Submodules

rvl_ur_motion_planner.URMoveitCommander module

class rvl_ur_motion_planner.URMoveitCommander.**URCommander**(*group_name='arm'*, *speed=0.1*, *accel=0.1*)
Bases: object

__init__(*group_name='arm'*, *speed=0.1*, *accel=0.1*)

all_close(*goal*, *actual*, *tolerance*)
Convenience method for testing if the values in two lists are within a tolerance of each other. For Pose and PoseStamped inputs, the angle between the two quaternions is compared (the angle between the identical

orientations q and -q is calculated correctly). @param: goal A list of floats, a Pose or a PoseStamped @param: actual A list of floats, a Pose or a PoseStamped @param: tolerance A float @returns: bool

define_preset_locations()

go_to_preset_location(*name*)

home()

report()

rvl_ur_remote_dashboard package

Submodules

rvl_ur_remote_dashboard.URInterfaceMapping module

class rvl_ur_remote_dashboard.URInterfaceMapping.**RobotModeMapping**(*value*)

Bases: enum.Enum

An enumeration.

BACKDRIVE = 6

BOOTING = 2

CONFIRM_SAFETY = 1

DISCONNECTED = 0

IDLE = 5

NO_CONTROLLER = -1

POWER_OFF = 3

POWER_ON = 4

RUNNING = 7

UPDATING_FIRMWARE = 8

class rvl_ur_remote_dashboard.URInterfaceMapping.**SafetyModeMapping**(*value*)

Bases: enum.Enum

An enumeration.

AUTOMATIC_MODE_SAFEGUARD_STOP = 12

FAULT = 9

NORMAL = 1

PROTECTIVE_STOP = 3

RECOVERY = 4

REDUCED = 2

ROBOT_EMERGENCY_STOP = 7

SAFEGUARD_STOP = 5

SYSTEM_EMERGENCY_STOP = 6

SYSTEM_THREE_POSITION_ENABLING_STOP = 13

UNDEFINED_SAFETY_MODE = 11

VALIDATE_JOINT_ID = 10

VIOLATION = 8

class rvl_ur_remote_dashboard.URInterfaceMapping.**SetIOFunctionMapping**(*value*)
Bases: enum.Enum

An enumeration.

SET_ANALOG_OUT = 3

SET_DIGITAL_OUT = 1

SET_FLAG = 2

SET_TOOL_VOLTAGE = 4

class rvl_ur_remote_dashboard.URInterfaceMapping.**SetIOPinMapping**(*value*)
Bases: enum.Enum

An enumeration.

class rvl_ur_remote_dashboard.URInterfaceMapping.**SetIOPinState**(*value*)
Bases: enum.Enum

An enumeration.

OFF = 0

ON = 1

class rvl_ur_remote_dashboard.URInterfaceMapping.**SetIOToolState**(*value*)
Bases: enum.Enum

An enumeration.

TOOL_VOLTAGE_0V = 0

TOOL_VOLTAGE_12V = 12

TOOL_VOLTAGE_24V = 24

rvl_ur_remote_dashboard.URRemoteDashboard module

class rvl_ur_remote_dashboard.URRemoteDashboard.**URRemoteDashboard**(*name: str = 'UR5e'*, *using_gripper: bool = False*, *using_urscript: bool = False*, *service_timeout: int = 5*)
Bases: object

__init__(*name: str = 'UR5e'*, *using_gripper: bool = False*, *using_urscript: bool = False*, *service_timeout: int = 5*) → None
The UR Remote Dashboard class. This is the primary extension overlaying the existing Universal Robot Driver code base to access mapped services.

Parameters

- **name** (*str, optional*) – Readable name to identify controller. Defaults to 'UR5e'.
- **using_gripper** (*bool, optional*) – Initialized the attached Robotiq gripper. Defaults to False.

- **using_urscript** (*bool, optional*) – Register appropriate publisher to send UR Script. Defaults to False.
- **service_timeout** (*int, optional*) – Wait time for services to come on. Defaults to 5.

clear_operational_mode() → bool
Allow PolyScope to change operational mode. User password will be enabled.

clear_protective_stop(*timeout: int = 30*) → bool
Clear a protective stop.

close_popup(*safety: bool = False*) → bool
Close a popup on the Teach Pendant or PolyScope.

Parameters **safety** (*bool, optional*) – Set to True if popup is a safety popup. Defaults to False.

Returns True if the targeted popup is closed.

Return type bool

cold_boot() → bool
Go directly to operational state (power on, brakes released). See release_brakes().

connect_dashboard(*quiet: bool = False*) → bool
Connect to the dashboard server. Need to be done before calling other services.

define_services()

disconnect_dashboard() → bool
Disconnect from the dashboard server.

get_loaded_program() → Optional[str]
Returns the name of the loaded program.

get_robot_mode() → Optional[int]
Returns the current robot mode.

get_safety_mode() → Optional[int]
Returns the current safety mode.

is_program_running() → bool
Returns true if the default or loaded program is running.

is_program_saved() → bool
Returns true if the default or loaded program is saved.

load_program(*filename: str*, *ptype: str*, *wait: int = 10*, *attempts: int = 10*) → None
Load a program or installation file.

Parameters

- **filename** (*str*) – Name of file with extension e.g., program.urp
- **ptype** (*str*) – Type of program. Accepting [‘prog’, ‘p’, ‘program’, ‘urp’] or [‘inst’, ‘i’, ‘installation’].
- **wait** (*int, optional*) – Wait time to handle known disconnection issue. Defaults to 10.
- **attempts** (*int, optional*) – Number of reconnection attempts. Defaults to 10.

log_to_pendant(*message: str*) → None
Log a message to PolyScope logs.

pause_loaded_program() → bool
Pause PolyScope program execution.

power_off_arm(*timeout: int = 30*) → bool
Power off the arm.

power_on_arm(*timeout: int = 30*) → bool
Power on the arm to idle state (brakes engaged).

query_program_state() → None
Display the name and execution state of the current PolyScope program.

raw_request(*query*)
Send any arbitrary message or request to the dashboard server.

register_robot_status() → None
Register necessary subscribers and callbacks to monitor robot operational status.

release_brakes(*timeout: int = 30*) → bool
Fully power on the robot with brakes released.

restart_safety() → bool
Clear a safety fault or violation. Arm will be powered off.

robot_iostate_callback(*msg*)

robot_safety_callback(*msg*)

robot_status_callback(*msg*)

send_popup(*message: str*) → bool
Send a message as a popup to Teach Pendant or PolyScope.

set_io(*function: int*, *pin: int*, *state: float*) → None
Set specific IO port on the robot. Currently not supporting specific domains (current/voltage).

Parameters

- **function** (*int*) – See SetIOFunctionMapping.
- **pin** (*int*) – Which pin to execute the function on.
- **state** (*float*) – 0/1 for digital IOs and value for analog IO.

set_payload(*mass: float*, *cx: float*, *cy: float*, *cz: float*) → None
Set the payload mass and center of gravity.

Parameters

- **mass** (*float*) – Mass of the payload in kg.
- **cx** (*float*) – Center of gravity of the payload.
- **cy** (*float*) – Center of gravity of the payload.
- **cz** (*float*) – Center of gravity of the payload.

set_speed_slider(*fraction: float*) → None
Set robot execution speed as a fraction. Only set less than 1 on scaled controllers.

Parameters **fraction** (*float*) – 0 to 1 if using scaled (default) controllers.

spam_connect(*attempts: int = 10*) → bool
Repeatedly calling connect() due to error prone and asynchronous status of the server.

Parameters **attempts** (*int, optional*) – Number of times connect() is called internally. Defaults to 10.

start_loaded_program() → bool
Start execution of default or loaded program.

stop_loaded_program() → bool
Stop PolyScope program execution.

system_shutdown() → None
Fully power down the robot (including control box).

terminate_external_control() → bool
Make the external_control node on PolyScope returns.

trigger_service(*serv_alias*)
Internal trigger service handling with exceptions

verify_services()

zero_force_torque_sensor() → bool
Zero the ft-sensor. Only work on e-Series in remote-control mode.

2.3.3 Utilities

rvl_utilities ROS Package

rvl_utilities package

Submodules

rvl_utilities.CustomLogger module

class rvl_utilities.CustomLogger.**ColorLogger**(*label=None*)
Bases: object

__init__(*label=None*)

define_sequences()

gen_output_str(*msg*, *esc*, *indent=0*)

log_error(*msg*, *indent=0*)

log_info(*msg*, *indent=0*)

log_success(*msg*, *indent=0*)

log_warn(*msg*, *indent=0*)

rvl_utilities.CustomLogger.**gen_ansi_rgb_esc**(*code*)

rvl_utilities.CustomLogger.**gen_esc**(*code*)

rvl_utilities.CustomLogger.**hex_to_rgb**(*hex*, *ansi=True*)

rvl_utilities.CustomLogger.**rgb_to_hex**(*rgb*)

CHAPTER

THREE

REFERENCES

The bulk of this driver is written based on old or current work, with some additional tweaks and tricks collected from browsing Universal Robots and Robotiq resources. Below is a tentative list of references, which will grow as we collect more.

- UniversalRobots/Universal_Robots_ROS_Driver
- ros-industrial/robotiq
- IntelRealSense/realsense-ros
- MoveIt! Motion Planning Framework
- Blog post by Jean-Philippe Roberge (last maintainer of the ROS Industrial Robotiq repository)
- ROS Tutorials

CHAPTER

FOUR

LICENSE

This repository is distributed under under the **Apache License, Version 2.0**. A nice tl;dr is provided here.

> **Warning:** When working with any robot, safety is of *utmost importance*. So please, **proceed with caution**.

4.1 Apache 2.0 Boilerplate

CHAPTER

FIVE

CHANGELOG

5.1 Releases

5.1.1 0.0.1-alpha - 2022/FEB/01

- Initial release of driver components: UR Remote Dashboard and Robotiq Controller
- Support for rudimentary forward and inverse kinematics
- Added pre-built Docker Image

5.2 Tasks

Don't worry if there's nothing here. I'm sure there's more…

PYTHON MODULE INDEX

r

INDEX

Symbols

A

B

C

D

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

Z