CoppeliaSim Interface Documentation

Release 1.0

Sven Becker

Feb 11, 2021

EXPLANATION

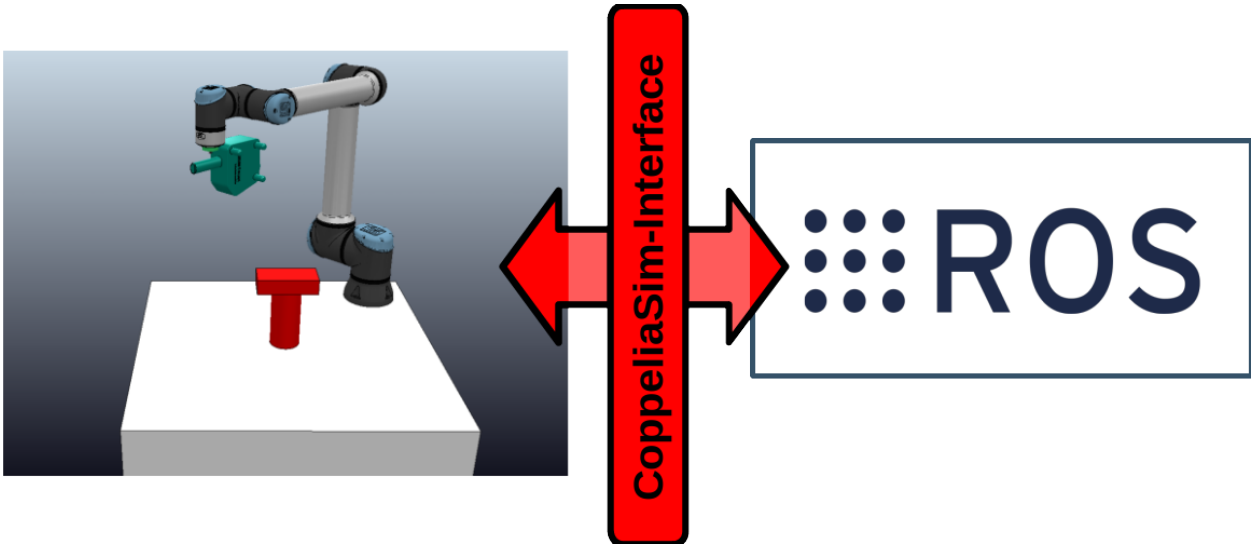


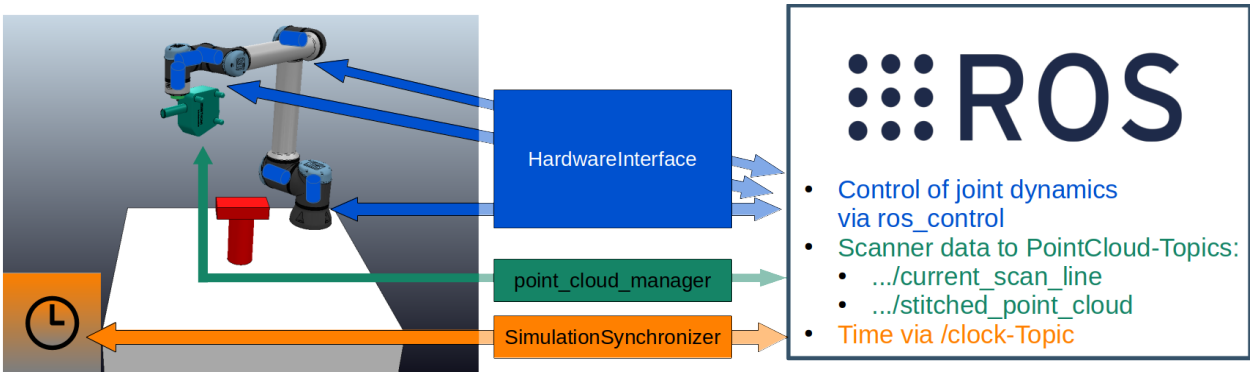
Fig. 1:[1]

[1] Source

CHAPTER

ONE

OVERVIEW



This package is the nexus between ROS and CoppeliaSim. The *SimulationSynchronizer C++-class* ensures that the time in ROS (available through '/clock'-topic) increases with the same rate as in CoppeliaSim. This is crucial for hard-timed tasks, like simulated real-time-control. This module also calls specified services at each simulation-step to for example read-out the Laser-Scanner or perform one control-cycle. For the latter, *HardwareInterface C++-class* mimics a real ros_control-interface but instead of controlling actual motors, the controller output is forwarded to the joints in CoppeliaSim's UR10e and the simulated joint-values are used as feedback for the ROS-controllers. This package was designed similarly to the drivers of the real UR-robots, which means that it works on the one hand seemlessly with every framework developed for real robots (like MoveIt) and on the other hand that movements evaluated in a simulation can be identically performed in reality when the real driver is loaded instead - *No remapping of ROS-topics, etc. is required*! The third component, *point_cloud_manager Python-module* , makes the Point-Cloud measured by the scanner in CoppeliaSim accessible to the ROS-system by transforming it. It also performs stitching of the measured Point-Cloud as well the application of an uncertainty-model to immediately review the measurement.

CHAPTER

TWO

HARDWAREINTERFACE C++-CLASS

class Hardware-Interface for the UR10e-model in CoppeliaSim that behaves just like the driver for the real robot. **Public Functions**

`coppeliasim_interface::HardwareInterface::`**`HardwareInterface`**`()`

Construct a new Hardware Interface object and initialize all class members that do not require a ROS- or CoppeliaSim-connection.

virtual Destroy the Hardware Interface object and handle disconnection from CoppeliaSim.

virtual Connects program to CoppeliaSim and obtains the required handles. Reads in the required ROS-parameters and initializes all interfaces that are implemented in the real hardware-interface.

`coppeliasim_interface::HardwareInterface::`**`~HardwareInterface`**`()` bool `coppeliasim_inte`

Whether connection to CoppeliaSim was successful and the hardware-interfaces could be set-up work with CoppeliaSim-Joints

Parameters

- `root_nh` - ROS-nodehandle for operations at root-level (names starting with '/...')
- `robot_nh` - ROS-nodehandle for operations in the hardware-interface-nodes namespace

virtual Gets the joint-data (position, velocity, torque) from CoppeliaSim in a blocking way.

void `coppeliasim_interface::HardwareInterface::`**`read`**`(`**`const`** ros::Time *&time*, **const** ros::Duration *&period*) **Parameters**

- `time` - Required by ROS-HardwareInterface specification (fullfills nothing here)
- `period` - Required by ROS-HardwareInterface (fullfills nothing here)

virtual Set the joint-target-values (velocities or positions, depending on active controller) in CoppeliaSim in a blocking way.

virtual

virtual Tries to turn on or off provided controllers. The provided controllers must be applicable with this *HardwareInterface*, otherwise they do not trigger any changes.

void `coppeliasim_interface::HardwareInterface::`**`write`**`(`**`const`** ros::Timebool `coppeliasim_inte`
&time, **const** ros::Duration *&period*)

- `start_list` - Controllers to start
- `stop_list` - Controllers to stop

bool `coppeliasim_interface::HardwareInterface::`**`zeroFTSensor`**(std_srvs::TriggerRequest &*req*, std_srvs::TriggerResponse &*res*)

Should normally tare the force-torque-sensor but is not implemented for performance reasons (service response gets delivered, but is always 'success == false')

Return

always 'true'

Parameters

- `req` - (contains no information)
- `res` - Trigger-Response (currently always 'success=false' with a corresponding message as this is not implemented)

bool `coppeliasim_interface::HardwareInterface::`**`setSpeedSlider`**(ur_msgs::SetSpeedSliderFraction &*req*, ur_msgs::SetSpeedSliderFraction &*res*)

Sets the speed-scaling as requested.

Return

always 'true'

Parameters

- `req` - Request with target value of the speed slider
- `res` - Response with information about whether the speed slider could be set (i.e. target lies in the interval [0.01,1])

bool `coppeliasim_interface::HardwareInterface::`**`setPause`**(std_srvs::SetBoolRequest &*req*, std_srvs::SetBoolResponse &*res*)

Controls if the robots running state (RUNNING = normal, PAUSED = halted, RAMPUP = Accelerating from PAUSED to RUNNING)

Return

always 'true'

Parameters

- `req` - Contains boolean flag about which run-state should be entered
- `res` - If the desired state could be set (false, if in RAMPUP or if the desired state is already set) with an descriptive message

CHAPTER

THREE

POINT_CLOUD_MANAGER PYTHON-MODULE

CHAPTER

FOUR

SIMULATIONSYNCHRONIZER C++-CLASS

class Synchronizes the ROS-time with the CoppeliaSim-time in the sense, that both 'clocks' are running at the same speed (but not necessarely at the same time), and ensures that time-critical tasks in the ROS-domain are performed in sync with each simulation-step. **Public Functions**

`coppeliasim_interface::SimulationSynchronizer::`**`SimulationSynchronizer`**`()`
Construct a new Simulation Synchronizer object and initialize all members that do not necessarely need a ROS- or CoppeliaSim-connection.

`coppeliasim_interface::SimulationSynchronizer::`**`~SimulationSynchronizer`**`()`
Destroy the Simulation Synchronizer object and stop the CoppeliaSim-Simulation and close the connection to CoppeliaSim.

bool `coppeliasim_interface::SimulationSynchronizer::`**`advanceSimulation`**`()`
Advances the simulation in CoppeliaSim exactly one time-step (dt = 'sim-dt'). Blocks until CoppeliaSim has finished the calculations for this advancement.

Return

Whether advancement was successful (could trigger calculations in CoppeliaSim and received success-feedback)

bool `coppeliasim_interface::SimulationSynchronizer::`**`synchronizeROS`**`()`
Increases ROS-time with the same dt as CoppeliaSim's 'sim-dt' and calls all ROS-services that should be handled during one time-step afterwards.

Return

Whether all Services were called successfully

bool `coppeliasim_interface::SimulationSynchronizer::`**`init`**`(`ros::NodeHandle *&nh*`)`
Connects instance to CoppeliaSim (registers it as client for the remoteApi and starts simulation in synchroneous mode) and ROS (preparing /clock-publication and registering Services that should be handled every simulation-step)

Return

Whether the method could connect to ROS and CoppeliaSim

Parameters

- `nh` - Nodehandle to that this instance uses to communicate with ROS

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search