pyUR

Author: Lukas Rustler

CONTENTS:

CHAPTER

ONE

UR10E PYTHON SIMULATOR WITH VIRTUAL AIRSKIN

This folder contains necessary files to run the simulation of the UR10e robot with virtual AIRSKIN.

1.1 Contents

- *Installation*
- *Code*
- *Run*

1.2 Installation

The installation is the same as for the adaptive_skin project. Please see the README there.

1.3 Code

The documentation PDF can be found in ur10ewithairskin.pdf. Online documentation is available at lukas-rustler.cz/pyur.

1.4 Run

The simulator can be run in two mode:

1) ROS version - real-time, high-level planners (MoveIt!, ...)
2) Native - much faster than real-time, only low-level control

1.4.1 ROS version

- install necessary things and build the workspace
  - see *Installation*
- run the simulator
  - in the terminal, run

```
roslaunch bullet_ros simulation.launch
```

  - this will start the simulator and RVIZ with the robot #### Control the robot
  - to test the connection etc. you can just move the robot in RVIZ and plan from there
  - or, see examples.py
  - it shows how to control the robot, gripper. How to play trajectories and how to get IK without moving
  - the simulated robot provides `position_controllers/ScaledJointTrajectoryController` and `velocity_controllers/JointGroupVelocityController`, i.e., waypoint controller through moveit or direct assignment of joint velocities to the joints
  - the default is `ScaledJointTrajectoryController`. You need to switch with rosservice `/controller_manager/switch_controller` before running joint commands
    - see examples.py for an example

1.4.2 Non-ROS version

- install necessary things
  - see *Installation*
- you can run examples from examples folder
  - scripts cartesian_example.py and joints_example.py show how to control the robot in cartesian and joint space, respectively.

CHAPTER

TWO

PYUR

2.1 pyUR

2.1.1 pyur module

class pyur.**EndEffector**(*name*, *client*)

Bases: object

Help function for end-effector encapsulaation

Parameters

- **name** (*str*) – name of the end-effector
- **client** (*pointer to pyCub instance*) – parent client

get_position()

Function to get current position of the end-effector

class pyur.**Joint**(*name*, *robot_joint_id*, *joints_id*, *lower_limit*, *upper_limit*, *max_force*, *max_velocity*)

Bases: object

Help class to encapsulate joint information

Parameters

- **name** (*str*) – name of the joint
- **robot_joint_id** (*int*) – id of the joint in pybullet
- **joints_id** (*int*) – id of the joint in pycub.joints
- **lower_limit** (*float*) – lower limit of the joint
- **upper_limit** (*float*) – upper limit of the joint
- **max_force** (*float*) – max force of the joint
- **max_velocity** (*float*) – max velocity of the joint

class pyur.**Link**(*name*, *robot_link_id*, *urdf_link*)

Bases: object

Help function to encapsulate link information

Parameters

- **name** (*str*) – name of the link
- **robot_link_id** (*int*) – id of the link in pybullet

- **urdf_link** (*int*) – id of the link in pycub.urdfs["robot"].links

class pyur.**pyUR**(*config='default.yaml'*)

Bases: BulletClient

Client class which inherits from BulletClient and contains the whole simulation functionality

Parameters
config (*str, optional, default="default.yaml"*) – path to the config file

compute_skin()

Function to compute the skin activations

Returns

Return type

contactPoints = {'DISTANCE': 8, 'FLAG': 0, 'FORCE': 9, 'FRICTION1': 10, 'FRICTION2': 12, 'FRICTIONDIR1': 11, 'FRICTIONDIR2': 13, 'IDA': 1, 'IDB': 2, 'INDEXA': 3, 'INDEXB': 4, 'NORMAL': 7, 'POSITIONA': 5, 'POSITIONB': 6}

create_urdf(*object_path*, *fixed*, *color*, *suffix=''*)

Creates a URDF for the given .obj file

Parameters

- **object_path** (*str*) – path to the .obj
- **fixed** (*bool*) – whether the object is fixed in space
- **color** (*list of 3 floats*) – color of the object

dynamicsInfo = {'BODYTYPE': 10, 'DAMPING': 8, 'FRICTION': 1, 'INERTIAOR': 4, 'INERTIAPOS': 3, 'INTERTIADIAGONAL': 2, 'MARGIN': 11, 'MASS': 0, 'RESTITUTION': 5, 'ROLLINGFRICTION': 6, 'SPINNINGFRICTION': 7, 'STIFFNESS': 9}

find_joint_id(*joint_name*)

Help function to get indexes from joint name of joint index in self.joints list

Parameters
joint_name (*str or int*) – name or index of the link

Returns
joint id in pybullet and pycub space

Return type
int, int

find_link_id(*mesh_name*, *robot=None*, *urdf_name='robot'*)

Help function to find link id from mesh name

Parameters

- **mesh_name** (*str*) – name of the mesh (only basename with extension)
- **robot** (*int, optional, default=None*) – robot pybullet id
- **urdf_name** (*str, optional, default="robot"*) – name of the object in pycub.urdfs

Returns
id of the link in pybullet space

Return type
int

get_joint_state(*joints=None*)

Get the state of the specified joints

Parameters
joints (*int or list, optional, default=None*) – joint or list of joints to get the state of

Returns
list of states of the joints

Return type
list

get_link_id(*link_name*)

init_robot()

Load the robot URDF and get its joints' information

Returns
robot and its joints

Return type
int or list

is_alive()

Checks whether the engine is still running

Returns
True when running

Return type
bool

jointInfo = {'AXIS': 13, 'DAMPING': 6, 'FLAGS': 5, 'FRICTION': 7, 'INDEX': 0, 'LINKNAME': 12, 'LOWERLIMIT': 8, 'MAXFORCE': 10, 'MAXVELOCITY': 11, 'NAME': 1, 'PARENTINDEX': 16, 'PARENTORN': 15, 'PARENTPOS': 14, 'QINDEX': 3, 'TYPE': 2, 'UINDEX': 4, 'UPPERLIMIT': 9}

jointStates = {'FORCES': 2, 'POSITION': 0, 'TORQUE': 3, 'VELOCITY': 1}

kill_open3d()

linkInfo = {'ANGVEL': 7, 'INERTIAORI': 3, 'INERTIAPOS': 2, 'LINVEL': 6, 'URDFORI': 5, 'URDFPOS': 4, 'WORLDORI': 1, 'WORLDPOS': 0}

motion_done(*joints=None*, *check_collision=True*)

Checks whether the motion is done.

Parameters

- **joints** (*int or list, optional, default=None*) – joint or list of joints to get the state of
- **check_collision** (*bool, optional, default=True*) – whether to check for collision during motion

Returns
True when motion is done, false otherwise

Return type
bool

move_cartesian(*pose*, *wait=True*, *velocity=1*, *check_collision=True*)

Move the robot in cartesian space by computing inverse kinematics and running position control

Parameters

- **pose** (utils.Pose) – desired pose of the end effector
- **wait** (*bool, optional, default=True*) – whether to wait for movement completion
- **velocity** (*float, optional, default=1*) – joint velocity to move with
- **check_collision** (*bool, optional, default=True*) – whether to check for collisions during motion

move_position(*joints*, *positions*, *wait=True*, *velocity=None*, *set_col_state=True*, *check_collision=True*)

Move the specified joints to the given positions

Parameters

- **joints** (*int, list, str*) – joint or list of joints to move
- **positions** (*float or list*) – position or list of positions to move the joints to
- **wait** (*bool, optional, default=True*) – whether to wait until the motion is done
- **velocity** (*float, optional, default=1*) – velocity to move the joints with
- **set_col_state** (*bool, optional, default=True*) – whether to reset collision state
- **check_collision** (*bool, optional, default=True*) – whether to check for collision during motion

move_velocity(*joints*, *velocities*)

Move the specified joints with the specified velocity

Parameters

- **joints** (*int or list*) – joint or list of joints to move
- **velocities** (*float or list*) – velocity or list of velocities to move the joints to

prepare_log()

Prepares the log string

Returns
log string

Return type
str

print_collision_info(*c=None*)

Help function to print collision info

Parameters
c (*list, optional, default=None*) – one collision

run_vhacd()

Function to run VHACD on all objects in loaded URDFs, and to create new URDFs with changed collision meshes

stop_robot(*joints=None*)

Stops the robot

toggle_gravity()

Toggles the gravity

update_simulation(*sleep_duration=0.01*)

Updates the simulation

Parameters

sleep_duration (*float, optional, default=0.01*) – duration to sleep before the next simulation step

visualShapeData = {'COLOR': 7, 'DIMS': 3, 'FILE': 4, 'GEOMTYPE': 2, 'ID': 0, 'LINK': 1, 'ORI': 6, 'POS': 5, 'TEXTURE': 8}

wait_motion_done(*sleep_duration=0.01*, *check_collision=True*)

Help function to wait for motion to be done. Can sleep for a specific duration

Parameters

- **sleep_duration** (*float, optional, default=0.01*) – how long to sleep before running simulation step
- **check_collision** (*bool, optional, default=True*) – whether to check for collisions during motion

2.1.2 utils module

class utils.**Config**(*config_path*)

Bases: object

Class to parse and keep the config loaded from yaml file

Parameters

config_path (*str*) – path to the config file

set_attribute(*attr*, *value*, *reference*)

Function to recursively fill the instance variables from dictionary. When value is non-dict, it is directly assigned to a variable. Else, the dict is recursively parsed.

Parameters

- **attr** (*str*) – name of the attribute
- **value** (*str, float, int, dict, list, ... - and other that can be loaded from yaml*) – value of the attribute
- **reference** (*pointer or whatever it is called in Python*) – reference to the parent class. "self" for the upper attributes, pointer to namedtuple for inner attributes

Returns

0

Return type

int

class utils.**CustomFormatter**(*fmt=None*, *datefmt=None*, *style='%'*, *validate=True*)

Bases: Formatter

Custom formatter that assigns colors to logs From https://stackoverflow.com/a/56944256

Initialize the formatter with specified format strings.

Initialize the formatter either with the specified format string, or a default as described above. Allow for specialized date formatting with the optional datefmt argument. If datefmt is omitted, you get an ISO8601-like (or RFC 3339-like) format.

Use a style parameter of '%', '{' or '$' to specify that you want to use one of %-formatting, `str.format()` ({}) formatting or `string.Template` formatting in your format string.

Changed in version 3.2: Added the `style` parameter.

```
FORMATS = {10:  '\x1b[38;20m%(module)s %(levelname)s:  %(message)s\x1b[0m', 20:
'\x1b[38;20m%(module)s %(levelname)s:  %(message)s\x1b[0m', 30:
'\x1b[33;20m%(module)s %(levelname)s:  %(message)s\x1b[0m', 40:
'\x1b[31;20m%(module)s %(levelname)s:  %(message)s\x1b[0m', 50:
'\x1b[31;1m%(module)s %(levelname)s:  %(message)s\x1b[0m'}
```

```
bold_red = '\x1b[31;1m'
```

format(*record*)

Format the specified record as text.

The record's attribute dictionary is used as the operand to a string formatting operation which yields the returned string. Before formatting the dictionary, a couple of preparatory steps are carried out. The message attribute of the record is computed using LogRecord.getMessage(). If the formatting string uses the time (as determined by a call to usesTime(), formatTime() is called to format the event time. If there is exception information, it is formatted using formatException() and appended to the message.

```
grey = '\x1b[38;20m'
```

```
red = '\x1b[31;20m'
```

```
reset = '\x1b[0m'
```

```
yellow = '\x1b[33;20m'
```

class utils.**Pose**(*pos*, *ori*)

Bases: `object`

Mini help class for Pose representation

Init function that takes position and orientation and saves them as attributes

Parameters

- **pos** (*list*) – x,y,z position
- **ori** (*list*) – rpy orientation

class utils.**URDF**(*path*)

Bases: `object`

Class to parse URDF file

Parameters
path (*str*) – path to the URDF file

```
ROOT_TAGS = []
```

dereference()

Make parent/child again as names to allow urdf write

find_root_tags()

Finds tags that are 'root', i.e., they have child 'inside'

fix_urdf()

Fix the URDF file by converting non-mesh geometries to mesh and saving them as .obj files. If changes were made, write the new URDF to a file.

make_references()

Make parent/child in joint list as references to the given link

read(*el*, *parent*)

Recursive function to read the URDF file. When there are no children, it reads the attributes and saves them.

Parameters

- **el** (*xml.etree.ElementTree.Element*) – The current element in the XML tree.
- **parent** (*xml.etree.ElementTree.Element*) – The parent element in the XML tree.

write_attr(*attr_name*, *attr*, *level=1*, *skip_header=False*)

Write an attribute to the new URDF string.

Parameters

- **attr_name** (*str*) – The name of the attribute.
- **attr** (*any*) – The attribute value.
- **level** (*int, optional, default=1*) – The indentation level for the attribute.
- **skip_header** (*bool, optional, default=False*) – Whether to skip writing the attribute header.

write_urdf()

Write the URDF object to a string.

2.1.3 visualizer module

class visualizer.**Visualizer**(*client=None*)

Bases: object

Class to help with custom rendering

Parameters

client (*int, optional, default=None*) – The client to be used for the visualizer.

class MenuCallback(*menu_id*, *parent*)

Bases: object

Class to handle menu callbacks.

Initialize the MenuCallback class.

Parameters

- **menu_id** (*int*) – The id of the menu.
- **parent** (*pointer to the class of visualizer.Visualizer type*) – The parent class (Visualizer).

input_completed(*text=None*)

save_image(*im*, *mode*)

Save the image. It shows FileDialog to find path for image save. It saves it with the current resolution of the window.

Parameters

- **im** (*open3d.geometry.Image*) – The image to be saved.
- **mode** (*int*) – The mode of the image. 0 for RGB, 1 for depth.

wait_for_dialog_completion()

Help function to keep the gui loop running

find_xyz_rpy(*mesh_name*, *urdf_name='robot'*)

Find the xyz, rpy and scales values.

Parameters

- **mesh_name** (*str*) – The name of the mesh.
- **urdf_name** (*str, optional, default="robot"*) – The name of the urdf.

Returns

The xyz, rpy, and scales, link_name

read_info(*obj_id*)

Read info from PyBullet

Parameters

obj_id (*int*) – id of the object; given by pybullet

Returns

0 for success

Return type

int

render()

Render all the things

show_first(*urdf_name='robot'*)

Show the first batch of meshes in the visualizer. It loads the meshes and saves the to dict for quicker use later

Parameters

urdf_name (*str, optional, default="robot"*) – The name of the urdf to be used.

show_mesh()

Function to parse info about meshes from PyBullet

2.1.4 gripper module

class gripper.RG6(*client*)

Bases: object

Class for the RG6

Parameters

client (*Client*) – instance of Client, holding info about the scene

JOINTS = ['left_inner_knuckle_joint', 'left_inner_finger_joint', 'right_outer_knuckle_joint', 'right_inner_knuckle_joint', 'right_inner_finger_joint']

SIGNS = [-1, 1, -1, -1, 1]

prepare_gripper()

Sets constraints for joints to works as mimic joint in URDF

Returns

Return type

reset_constraints()

Remove all constraints

Returns

Return type

set_gripper_pose(*position*, *velocity=None*, *wait=False*)

Set goal position of gripper

Parameters

- **position** (*float*) – position of the gripper
- **wait** (*bool*) – whether to wait for the motion to finish

Returns

Return type

stop()

2.2 Examples

2.2.1 cartesian_example module

Author

Lukas Rustler

cartesian_example.**move**(*client*)

2.2.2 joints_example module

Author

Lukas Rustler

joints_example.**move**(*client*)

CHAPTER

THREE

BULLET_ROS

3.1 bullet_ros

3.1.1 main module

class main.**BulletROS**

Bases: object

activate_airskin()

Function to activate the airskin based on variables that had been set through service

Returns

Return type

activate_airskin_cb(*request*)

Callback for airskin activation request

Parameters
request (*bullet_ros.srv.activateAirskinRequest*) – request with all the info

Returns

Return type

change_obj_pose(*request*)

Change pose of other than robot object

Parameters
request (*bullet_ros.srv.changeObjPoseRequest*) – request fot the pose change

Returns

Return type

grip(*request*)

Grip request callback

Parameters
request (*bullet_ros/srv/gripRequest*) – request to close/open the gripper

Returns

Return type

move()

Based on move_type (velocity or position) move the robot in the given pose or fix it in place

Returns

Return type

publish_other_objects()

Function to publish other than robot objects to RVIZ

Returns

Return type

read(*msg*)

Function to read the message from the RobotHW interface and send it to the bullet

Parameters
msg (*bullet_ros.msg.ros_to_bullet*) – msg with velocity and joint angles

Returns

Return type

read_state()

Read current robot state to be sent back to RobotGH interface

Returns

Return type

send_finger_joint()

Get state of finger joint and publish it

Returns

Return type

send_skin()

Function to send current airskin values to /airskin_status topic

Returns

Return type

3.1.2 motion_interface module

Classes for robot movement

Edited from official ROS tutorial by Lukas Rustler

class motion_interface.**MoveGroupPythonInterface**(*group_*)

Bases: object

all_close(*goal*, *actual*, *tolerance*)

Convenience method for testing if a list of values are within a tolerance of their counterparts in another list @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

apply_planning_scene(*scene_msg*)

close_gripper()

display_trajectory(*plan*)

Display trajectory in Rviz, the following code is needed for proper displaying.

execute_plan(*plan*, *wait=True*)

Note: The robot's current joint state must be within some tolerance of the first waypoint in the **`RobotTrajectory`_** or `execute()` will fail

get_current_state()

get_ee_pose(*scale=1*)

go_to_joint_position(*joint_goal*, *wait=True*)

go_to_pose(*pose*, *wait*)

open_gripper()

plan_cartesian_path(*wpose*, *collisions=True*)

Cartesian Paths

You can plan a Cartesian path directly by specifying a list of waypoints for the end-effector to go through:

stop_robot()

3.1.3 robot_kinematics_interface module

@author: Jan Behrens

This source code is derived from the dmp_gestures project (https://github.com/awesomebytes/dmp_gestures) Copyright (c) 2013, Willow Garage, Inc., licensed under the BSD license, cf. 3rd-party-licenses.txt file in the root directory of this source tree.

class robot_kinematics_interface.**ForwardKinematics**

Bases: `object`

Simplified interface to ask for forward kinematics

closeFK()

getCurrentFK(*fk_link_names*, *frame_id='base_link'*)

Get the forward kinematics of a set of links in the current configuration

getFK(*fk_link_names*, *joint_names*, *positions*, *frame_id='base_link'*)

Get the forward kinematics of a joint configuration @fk_link_names list of string or string : list of links that we want to get the forward kinematics from @joint_names list of string : with the joint names to set a position to ask for the FK @positions list of double : with the position of the joints @frame_id string : the reference frame to be used

class robot_kinematics_interface.**InverseKinematics**(*ik_srv=None*)

Bases: object

Simplified interface to ask for inverse kinematics

closeIK()

getIK(*group_name*, *ik_link_name*, *pose*, *avoid_collisions=True*, *robot_state=None*, *constraints=None*, *timeout=1*)

Get the inverse kinematics for a group with a link a in pose in 3d world. @group_name string group i.e. right_arm that will perform the IK @ik_link_name string link that will be in the pose given to evaluate the IK @pose PoseStamped that represents the pose (with frame_id!) of the link @avoid_collisions Bool if we want solutions with collision avoidance @attempts Int number of attempts to get an Ik as it can fail depending on what IK is being used @robot_state RobotState the robot state where to start searching IK from (optional, current pose will be used if ignored)

class robot_kinematics_interface.**StateValidity**

Bases: object

close_SV()

getStateValidity(*robot_state*, *group_name='both_arms_torso'*, *constraints=None*)

Given a RobotState and a group name and an optional Constraints return the validity of the State

3.1.4 rg_finger_publisher module

@author Lukas Rustler

rg_finger_publisher.**send_tf**()

3.2 Examples

3.2.1 examples module

PYTHON MODULE INDEX

INDEX

T

U

V

W

Y