SIMULATE-ROBOTICS-FOR-TEACHING – DOCUMENTATION

13. März 2022

Inhaltsverzeichnis

1 Structure of the Project

```
simulate-robotics-for-teaching
├─ logs
│  └─ run_time.log
├─ Tasks
│  ├─ ex{2-5}_tasks.py
│  └─ ex{2-5}_tasks_sol.py
├─ Simulation
│  ├─ Data
│  │  └─ *
│  ├─ Environment
│  │  └─ build_scenario.py
│  ├─ Robot
│  │  └─ *
│  ├─ Tests
│  │  └─ ex{2-4}_tests.py
│  ├─ UI
│  │  └─ *
│  └─ utils
│     └─ *
├─ run_exercise{2-5}.py
├─ requirements.txt
└─ README.md
```

- The idea of this project structure is to apply an easy installation of the programming tasks, because most of the mechanical stundents don't have any or only a few programming experiences. This should help to prevent confusions in the exercise tasks.
- The most important files for the students are the $run\_exercise\{2-5\}.py$, for executing the simulation on their personal computer, and the folder $Tasks$ for doing the necessary programming tasks $ex\{2-5\}\_tasks.py$. Everything else that depends on the simulation is in the background and should not be touched by the students!
- All necessary packages that needs to be installed before running the programming tasks are in the $requirements.txt$ file. In most cases by using an IDE's, the packages can be installed in the integraded terminal with the follwing command:

  pip install -r requirements.txt

- Another option would be to read the $README.md$ file and follow the instructions
- Short discription of the $Simulation$ folder:
  - $Data$: All data for testing programming tasks.
  - $Environment$: Build and load an pybullet environment.
  - $Robot$: Model of the robot, algorithm for the robot, etc.
  - $Tests$: Test programming exercises and show if the result is correct implemented.
  - UI: User Interface of the specific programming tasks, like inverse kinematic, dynamic and planning.
  - $utils$: Everything else that depends on pybullet, operating system, threading, etc.
- For more information, see Chapter 2.
- For showing solution, copy everything in the $ex*\_tasks\_sol.py$ in $ex*\_tasks.py$ and $run\_exercise*.py$.

2 Description of the folders in $Simulation$

2.1 UI

```
└─ UI
   ├─ robot_control.py
   ├─ robot_dynamic.py
   └─ robot_planning.py
```

- $robot\_control$: User Interface (UI) for computing forward- and inverse kinematic. Set configuration in pybullet simulation. The stundents can control all joints and set it in the simulation or can choose a target end-effector position of the TCP-position and search a suitable inverse kinemtatic for all joints.
- $robot\_dynamic$: User Interface (UI) for computing dynamics. Get accelerleration/forces from UI and set configuration in pybullet simulation. Computes dynamics with the Recursive-Newton-Euler algorithm (RNE).
- $robot\_planning$: User Interface (UI) for planning a trajectory in the pybullet simulation. Execution can be done in task space or cartesian space. Students are enabled for adding new TCP-position, switch two TCP-position, or delete a TCP-position for a specific trajectory.

- UI's: TODO add figures

2.2 Robot

```
└─ Robot
   ├─ never_collisions.py
   ├─ robot_primitives.py
   ├─ robot_utiles.py
   ├─ robot.py
   └─ panda
      ├─ franka_description
      │  └─ *
      ├─ franka_panda
      │  └─ *
      └─ panda.urdf
```

- *never_collisions.py*: Discribes a list full of tuples of links and joints. This list can be generated from **ROS (Robot Operating System)** and must be written out of ROS manually. In pybullet, every contact with a link and/or joints that is connected, is detected as a collision. This means, we have to sort out all collisions, because they are put together.
- *robot_primitives.py*: Discribes all important subroutines for the robot, like, calculating inverse kinematic, calculate dynamics, compute a trajectory, grasping or motion control in the simulation.
- *robot_utiles.py*: Helper class for the *robot_primitive.py* class. Everything else that is important for the robot, but is not essential for the robot subroutines. Example for this class are set standard arm configuration, transform reference system, or draw coordinate systems in the pybullet simulation.
- *robot.py*: Class that discribes the robot. Creates a robot setup with all necessary robot information, like joint limits, max joint velocity, ID's of the joints and gripper, or creates a **RTB (robotic-toolbox)** object.

- *panda*: Folder for discribing the robot model. The *panda.urdf*-file discribes all robot specific inforamtion, like, inertia, center of mass, or joints limits. With the *franka_description*, we can generate a new $*$*.urdf* file from a $*$*.xacro* file. It can easily generated by **ROS 1 - Melodic** with the following guide:
  1) Install ros-melodic
  2) Create a ros workspace:
     2.1) source /opt/ros/melodic/setup.bash
     2.2) mkdir -p /catkin_ws/src
     2.3) cd /catkin_ws/
     2.4) catkin_make
     2.5) source /devel/setup.bash
  3) Copy *_description into src/ folder
  4) Go to src folder with cd-command
  5) craete *.urdf file:
     5.1) rosrun xacro xacro *_description/robots/*.urdf.xacro > <name>.urdf
     5.2) Example: rosrun xacro xacro franka_description/robots/panda.urdf.xacro > panda.urdf
  6) Important: Everytime you open a new terminal, you HAVE to do the following command in the catkin_ws folder:
     6.1) source /devel/setup.bash

2.3 Data

```
Data
├─ accel_solution.npy
├─ coefficient_sol.npy
├─ default_transformation_matrix.npy
├─ default_workspace.npy
├─ force_solution.npy
├─ load_trajectory.npy
├─ qddt_sol.npy
├─ qdt_sol.npy
├─ qt_sol.npy
└─ test_trajectory.npy
```

- In this folder, all exercise tasks needs to be checked. We load default solutions from this folder and we check the shape of the return values and if it correct programmed.
  - Exercise 2: Checks the shape of the workspace and the resutl of the transformation matrix
  - Exercise 3: Checks the shape and result of the force and acceleration
  - Exercise 4: Checks shape and result of the coefficient, the joint position (qt), joint velocity (dqt), joint acceleration (ddqt) and test trajectory.
  - Exercise 5: No checks needed. If exercise 2-4 is correct implemented, exercise 5 can be executed with no errors.
- If an error appears, the pybullet simulation do nothing. It only shows a message in the terminal, that the exercise task is not right implemented. Due to the lack of experience, students are overwhelmed with errors in the code.

2.4 utils

```
utils
├─ logging
│  ├─ collision_handler.py
│  └─ runtime_handler.py
├─ models
│  ├─ box.urdf
│  ├─ floor.urdf
│  └─ table_collision
│     ├─ table.urdf
│     └─ *
├─ pybullet_tools
│  ├─ os_utils.py
│  └─ pybullet_utils.py
└─ utils.py
```

- *logging*: Logging handler for collision and runtime. Save all messages in *logs/run_time.log*.
- *models*: All other models for the environment, that should be loaded into the pybullet simulation, like, box, floor and a table.
- *pybullet_tools*: Everything that have to do with pybullet or the operating system.
- *utils.py*: Everything else, that does not have to do with the robot or pybullet.

3 Installation

- Download IDE (Pycharm, Visual Studio Code, etc.)
- Download Python Version 3.6 - 3.9.*
  - Python Version 3.10.* is currently not supported. Problem with threading and Tkinter.

- Depending on the IDE, install $requirements.txt$ file (see section 3.2 for more details) via **terminal** or use the **project interpreter**

pip install -r requirements.txt   or   pip install <package_name>==<version>

- Sometimes the package **roboticstoolbox-python** is not always detected by the project interpreter. Close IDE and open the project again.

3.1 Operating Systems

- Supported Operating Systems are
  - Windows 10
  - Unix (Ubuntu (18.06, 20), Debian, etc.),
  - MacOS (Big Sur Version 11.6)
- MacOS is not always supported, because the Python Package **Tkinter**.

3.2 Requirements

- Package list:
  - pybullet==3.2.0
  - numpy==1.20.3
  - scipy==1.7.0
  - matplotlib==3.4.2
  - roboticstoolbox-python==0.11.0
  - sympy==1.9

4 Introduction

4.1 Build World

In this Project, every simulation environment is an object. This objects defines the most important arguments. All these different environments are located into the same folder **Environment**. Every floor, robot, shelf or wall gets an individual identification number (*body*) from the simulation, that is loaded into the simulation environment.

This identification number is useful for checking collision with different objects in the simulation, or getting joint limits, velocities or forces out of the urdf file.

4.1.1 __$init$__

Input Parameters:

	Parameters	Type	Description
optional	place_obstacle	bool	Place an obstacle in the simulation for collision testing task. This obstacle is needed for the exercise 5.

Initialized Parameters:

Parameters	Type	Description
movable_bodies	list	List of unique id, that is returned by load_pybullet_model. The variable movable_bodies discriebes all objects in the simulation environment, that can be moved around.
env_bodies	list	List of unique id, that is returned by load_pybullet_model. Examples are floor, plants, etc.
regions	list	List of unique id, that is returned by load_pybullet_model. Examples are tables, shelves, etc.
robots	list	List of unique id, that is returned by load_pybullet_model.
all_bodies	list	List of unique id, that is returned by load_pybullet_model. The variable all_bodies describes all spawned objects in the simulation environment. Needed for collision checking.

4.1.2 $configuration$

In this method, all position, orientation of the spawned models should be configured here. For more details of setting all environment specific configuration see explanation in Sec. 4.3.

Setting orientation, position or joint position, while spawning *body* into the simulation environment causes problems. Spawn the body into the pybullet environment and then change the position, orientation, etc. of the body.

Use the methods, like, *set_position and set_orientation* (see Sec. 4.5), *set_arm_config* (see Sec. 4.5) and *open_gripper or close_gripper* (see Sec. 4.5) TODO change ref

Input Parameters:

	Parameters	Type	Description
optional	place_obstacle	bool	Place an obstacle in the simulation for collision testing task. This obstacle is needed for the exercise 5.

4.2 *Connect, Disconnect*

PyBullet has to connect to the physics simulation, by sending different commands to the client-server API. The client connects to the physics server and the server retruns the status. You can read more about it in the Pybullet Documentation: `https://docs.google.com/document/d/10sXEhzFRSnvFcl3XxNGhnD4N2SedqwdAvK3dsihxVUA/edit#heading=h.2ye70wns7io3`

The connect method will automatically set the debug visualizer to False and set the time step to t = 1 / 60 (See Sec. 4.4.1). This is the update cycle of the pybullet simulation. Any trajection should be built according to this cycle.

Input Parameters:

	Parameters	**Type**	**Description**
optional	c	int	PyBullet has some built-in physics servers: DIRECT and GUI. Both GUI and DIRECT connections will execute the physics simulation and rendering in the same process as PyBullet.
optional	dt	float	Each time you call 'stepSimulation' the timeStep will proceed with 'timeStep'.

The disconnect method closes the pybullet simulation.

4.3 Load URDF in Simulation

4.3.1 *load_pybullet_model*

Load model into simulation. Return an integer value for calling the object in the simulation. The loadURDF will send a command to the physics server to load a physics model from a Universal Robot Description File (URDF). The URDF file is used by the ROS project (Robot Operating System) to describe robots and other objects.

Input Parameters:

	Parameters	**Type**	**Description**
optional	filename	str	a relative path to the URDF file on the filesystem of the physics server
optional	fixed_base	bool	Set model or model base as fixed (cannot be moved)
optional	scale	float	scaling will apply a scale factor to the URDF model

Attention: Do not load a model with a start position and start orientation with this function. The internal pybullet method has already a method that can load a model with a given start position and orientation, but this causes troubles in the simulation. First load a model with *load_pybullet_model* and then change the orientation and start position with *set_position* (see Sec. 4.5), *set_orientation* (see Sec. 4.5) or *set_position_and_orientation* (see Sec. 4.5).

4.4 Step Simulation

4.4.1 *set_time_step*

You can set the physics engine timestep that is used when calling 'stepSimulation'. It is best to only call this method at the start of a simulation. Don't change this time step regularly.setTimeStep can also be achieved using the new setPhysicsEngineParameter API. It will be automatically called in the connect method (see Sec. 9.2.3).

Input Parameters:

	Parameters	**Type**	**Description**
optional	dt	float	Each time you call 'stepSimulation' the timeStep will proceed with 'timeStep'.

4.5 Set Position and Orientation

4.5.1 *set_position*

Set position of a body in the pybullet simulation.

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc
required	position	list	reset the base of the object at the specified position in world-space coordinates [X,Y,Z]

4.5.2 *set_orientation*

Set orientation of a body in the pybullet simulation.

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc
required	orientation	list	reset the base of the object at the specified orientation as worldspace quaternion [X,Y,Z,W]

4.5.3 *set_position_and_orientation*

Set position and orientation of a body in the pybullet simulation.

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc
required	position	list	reset the base of the object at the specified position in world-space coordinates [X,Y,Z]
required	orientation	list	reset the base of the object at the specified orientation as worldspace quaternion [X,Y,Z,W]

4.5.4 *set_arm_config*

Set joint configuration for a body in the simulation environment.

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc
required	limbs	str	Name of the robot limb part, i.e. *gripper* or *arm*. Limbs have to define in *Simulation/Robots/robot_utils.py* in the dictionary ROBOT_GROUPS. Loads name of the joints that should be set.
required	config	list/numpy (flatten)	Joint configuration, that should be set.

4.5.5 *open_gripper*

Set gripper joints to maximum limits.

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc

4.5.6 *close_gripper*

Set gripper joints to minimum limits.

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc

Attention: You can reset the position and orientation of the base (root) of each object. Do it only in the beginning of the simulation start. Do changing position and orientation only at the start, and not during a running simulation, since the command will override the effect of all physics simulation.

4.6 *activate_gravity*

By default, there is no gravitational force enabled. setGravity lets you set the default gravity force for all objects. Gravitiy is automatically set to $g = 8.91$

Input Parameters:

	Parameters	Type	Description
optional	gravity	float	Set individual gravity to the simulation. If not set, use standard earth gravity

5 Tasks

All exercise task that should be programmed by the students.

5.1 Exercise 2 - Kinematics

5.1.1 *initialize_workspace*

Initialize an approximately workspace of the robot model. It is not needed for the following exercises the get a perfect workspace. Later for the planning task in exercise 4, it will be always evaluated, if the target point is reachable. If its reachable, then it will be added the planning path, else you get a message in the user interface.

Input Parameters:

	Parameters	**Type**	**Description**
required	robot	RobotSetup	Object description of the robot model with all important information

5.1.2 *inverse_kinematic*

Calculates the inverse kinematic of a target point.

Input Parameters:

	Parameters	**Type**	**Description**
required	ik_solver	InverseKinematic	Object description of the InverseKinematic
required	input_tuple	tuple	Input tuple of the target-position [X, Y, Z], target-orientation [rX, rY, rZ] (as euler angles) and a list of target bodies in the environment . Target bodies can be also None.
required	search_type	bool	Search a new joint position, when target position is already reached. Used only for in exercise 2

5.1.3 *set_joints*

Set joint in the pybullet simulation.

Input Parameters:

	Parameters	**Type**	**Description**
required	robot	RobotSetup	Object description of the robot model with all important information
required	joint_positions	list	All movable joint position that should be set

5.1.4 Rx, Ry, Rz

Generate the rotation matrix R_x, R_y, R_z with a the euler angles and return it as a $4x4$-numpy array.

Input Parameters:

	Parameters	**Type**	**Description**
required	alpha	float	Radiant angle of the rotation

5.1.5 *translation*

Calculates the translation matrix and return a $4x4$ numpy array.

Input Parameters:

	Parameters	Type	Description
optional	position	list/numpy	Target position as a flatten numpy array or a list

5.1.6 *compute_transformation*

Computes the target transformation matrix and return it as a $4x4$-numpy array.

Input Parameters:

	Parameters	Type	Description
optional	position	list/numpy	Target position as a [X, Y, Z] list or a flatten numpy array
optional	orientation	list/numpy	Target orientation as euler angles. [rX, rY, rZ] as a list or a flatten numpy array

5.2 Exercise 3 - Dyanmics

5.2.1 Recursive Newton Euler - *rne*

Computes the torques with the recursive newton euler methods and return it as a flatten numpy array

Input Parameters:

	Parameters	Type	Description
required	dyn	Dynamic	Object description of the dynamic robot model
required	q	numpy	Joint position as a flatten numpy array
required	qd	numpy	Joint velocity as a flatten numpy array
required	qdd	numpy	Joint acceleration as a flatten numpy array
optional	gravity	list/numpy	Set a specific gravitational force. If None, it sets a default gravity force (9.81)

5.2.2 Acceleration - *accel*

Computes the acceleration with the recursive newton euler methods and return it as a flatten numpy array

Input Parameters:

	Parameters	Type	Description
optional	dyn	Dynamic	Object description of the dynamic robot model
optional	q	numpy	Joint position as a flatten numpy array
optional	qd	numpy	Joint velocity as a flatten numpy array
optional	torques	numpy	Torques as a as a flatten numpy array

5.2.3 *euler_step*

Calculate one euler step with a specific time step. Return new position q, velocity qd and acceleration qdd as a flatten numpy array

Attention: This method is not used and completly useless. It will only checked, if the students has programmed it correctly, but we are using the method *integration* in Simulation/utils/utils.py for the euler step. The reason is pybullet, the design of the user interface and the knowlege of the students. To make it simpler, the $euler\_step$ methods get only one parameter for the acceleration and this is not the right way for computing it. The method $integration$ gets an additional parameter qdd_t, because in the app you can do both ways, setting acceleration or setting a specific force to a joint. It depends on which mode the students want to see in the simulation and how changing one slider in the user interface affects the robot in the simulation.

When qdd_t is set, then the parameter tau in $integration$ is *None* and return the new qdd_t as qdd. This means we are one iteration behind. If the students selects a specific force to a joints, then we are updating and caluclating the acceleration qdd.

Input Parameters:

	Parameters	**Type**	**Description**
optional	dyn	Dynamic	Object description of the dynamic robot model
optional	q	numpy	Joint position as a flatten numpy array
optional	qd	numpy	Joint velocity as a flatten numpy array
optional	torques	numpy	Torques as a as a flatten numpy array
optional	dt	float	Time step for the current epoch

5.2.4 $integration$

Right implementation of the euler method, with different name. Calculate one euler step with a specific time step. Return new position q, velocity qd and acceleration qdd as a flatten numpy array

Input Parameters:

	Parameters	**Type**	**Description**
optional	accel_method	method	Function of the acceleration method (has to be programmed from the students)
optional	dyn	Dynamic	Object description of the dynamic robot model
optional	q	numpy	Joint position as a flatten numpy array
optional	qd	numpy	Joint velocity as a flatten numpy array
optional	qdd	numpy	Current joint acceleration as a flatten numpy array
optional	qdd_t	numpy	New acceleration from the dynamic app as a flatten numpy array
optional	tau	numpy	New torque from the dynamic app as a flatten numpy array
optional	dt	float	Time step for the current epoch

5.2.5 $gravload$

Calculates the force compensation for a resting robot model and print result.

Input Parameters:

	Parameters	**Type**	**Description**
optional	dyn	Dynamic	Object description of the dynamic robot model
optional	q	numpy	Joint position as a flatten numpy array

5.3 Exercise 4 - Trajectory

5.3.1 $compute\_coefficient$

Calculates the coefficients for a specific trajectory.

Input Parameters:

	Parameters	Type	Description
required	q	tuple	Tuple of the start joint position and the end joint position as a flatten numpy array
required	qd	tuple	Tuple of the start joint velocity and the end velocity position as a flatten numpy array . Also possible for using ints or flaots, if you want every joints have the same start- and end-velocity
required	qdd	tuple	Tuple of the start joint acceleration and the end joint acceleration as a flatten numpy array. Also possible for using ints or flaots, if you want every joints have the same start- and end-acceleration
required	t	tuple	Start and end time as integer values (execution time) with $t_0 = 0$

5.3.2 *q_t*

Calculates the trajectory of the joint position for every time step and return the joint position for a trajectory as a (n, dof) numpy array

Input Parameters:

	Parameters	Type	Description
required	a_0	numpy	Trajectory coefficient as a (n, dof) numpy array
required	a_1	numpy	Trajectory coefficient as a (n, dof) numpy array
required	a_2	numpy	Trajectory coefficient as a (n, dof) numpy array
required	a_3	numpy	Trajectory coefficient as a (n, dof) numpy array
required	a_4	numpy	Trajectory coefficient as a (n, dof) numpy array
required	a_5	numpy	Trajectory coefficient as a (n, dof) numpy array
required	ti	numpy	Time steps for the trajectory as a flatten numpy array of the lenght of n

5.3.3 *qd_t*

Calculates the trajectory of the joint position for every time step and return the joint position for a trajectory as a (n, dof) numpy array.

Input Parameters:

	Parameters	Type	Description
required	a_0	numpy	Trajectory coefficient as a (n, dof) numpy array
required	a_1	numpy	Trajectory coefficient as a (n, dof) numpy array
required	a_2	numpy	Trajectory coefficient as a (n, dof) numpy array
required	a_3	numpy	Trajectory coefficient as a (n, dof) numpy array
required	a_4	numpy	Trajectory coefficient as a (n, dof) numpy array
required	ti	numpy	Time steps for the trajectory as a flatten numpy array of the lenght of n

5.3.4 *qdd_t*

Calculates the trajectory of the joint position for every time step and return the joint position for a trajectory as a (n, dof) numpy array.

Input Parameters:

	Parameters	Type	Description
required	a_0	numpy	Trajectory coefficient as a (n, dof) numpy array
required	a_1	numpy	Trajectory coefficient as a (n, dof) numpy array
required	a_2	numpy	Trajectory coefficient as a (n, dof) numpy array
required	a_3	numpy	Trajectory coefficient as a (n, dof) numpy array
required	ti	numpy	Time steps for the trajectory as a flatten numpy array of the lenght of n

5.4 Exercise 5 - Movement planning

5.4.1 *unrestrained_movement*

Calculates the trajectory of a given trajectory with an euler step and return new trajectory for joint position, joint velocity, joint acceleration and the forces as a numpy array with a shape of (n, dof).

Input Parameters:

	Parameters	Type	Description
required	dyn	Dynamic	Object description of the dynamic robot model
required	robot	RobotSetup	Object description of the robot model with all important information
required	q	numpy	Trajectory for the joint position as a (n, dof) numpy array
required	qd	numpy	Trajectory for the joint velocity as a (n, dof) numpy array
required	qdd	numpy	Trajectory for the joint acceleration as a (n, dof) numpy array
required	h	numpy	Time step

5.4.2 *restrained_movement*

Calculates the trajectory of a given trajectory with an euler step and return new trajectory for joint position, joint velocity, joint acceleration and the forces as a numpy array with a shape of (n, dof).

Input Parameters:

	Parameters	Type	Description
required	dyn	Dynamic	Object description of the dynamic robot model
required	robot	RobotSetup	Object description of the robot model with all important information
required	q	numpy	Trajectory for the joint position as a (n, dof) numpy array
required	qd	numpy	Trajectory for the joint velocity as a (n, dof) numpy array
required	qdd	numpy	Trajectory for the joint acceleration as a (n, dof) numpy array
required	h	numpy	Time step

6 Robots

This is the most important class. Needed for all subroutines, like, *InverseKinematics* (see Sec.), *Dynamics* (see Sec.), or *PathPlanning* (see Sec.)

Pybullet does not always find the ID numbers from the scratch, for example for the gripper joint id's. The ID number is addressable in the simulation, but cannot be queried by pybullet. Somethimes it has to be hardcoded. For this use *ROBOT_GROUPS_ID* or *ROBOT_GROUPS* in Simulation/Robots/robot_utils.py

6.1 RobotSetup and RobotConfig

Define and crate an object of the robot model. Initialize all important robot information.

6.1.1 __*init*__

Input Parameters:

	Parameters	**Type**	**Description**
required	body	int	body unique id, as returned by loadURDF etc
required	name	str	Name of the robot
required	dof	int	Degrees of freedom of the robot
required	file_name	str	Path of the file name. Loads URDF file into RTB (robotics toolbox) object.
optional	custom_limits	list/numpy	Set custom limits to the robot model, instead of using true limits. Input shape for costum limits needs to be (dof, 2)

Initialized Parameters:

Parameters	**Type**	**Description**
model	ERobot	Robotics-Toolbox ERobot object
body	int	body unique id, as returned by loadURDF etc
name	str	Name of the robot
dof	int	Degrees of freedom of the robot
joint_limits	numpy	Joint limits of the robot. Shape of the joint limits is (n, 2) with n = dof + 1. We only need one additional limit for the gripper joints. Use the last row for all other fingers on the gripper.
joints	numpy	All joints (arm and gripper) of the robot. Attention: If finger1 is changed, then all following gripper joints should get the same values. It is not included in this list. use movable_gripper for setting the same values. In this list, there is only one gripper joint.
movable_joints	numpy	Arm joints id's. Uses the method *get_joints* (see Sec. TODO).
movable_gripper	numpy	Gripper joint id's. Define ID's of the grippes in ROBOT_GROUPS_ID['gripper']. Needs to be hardcoded. See explanation above in Sec. 6
max_velocity	numpy	Get all maximum velocity of every joint as a flatten numpy array. Shape is (dof,)
max_forces	numpy	Get all maximum forces of every joint as a flatten numpy array. Shape is (dof,)

7 Robot Primitives

7.1 Kinematic

7.1.1 *__init__*

Input Parameters:

	Parameters	Type	Description
required	robot	RobotSetup	Object dscription of the robot model with all important information

Initialized Parameters:

Parameters	Type	Description
model	robot	Object dscription of the robot model with all important information

7.1.2 *compute_tcp_position*

Compute the tcp position of the end-effector with a given joint configuration. Returns [X, Y, Z] position as a flatten numpy array

Input Parameters:

	Parameters	Type	Description
required	q	numpy	Vector of the joint configuration as a flatten numpy array
required	to_global	bool	Return end-effector position in local coordinate system or global coordinate system

7.1.3 *get_current_tcp_values*

Compute tcp position and the tcp orientation of the current joint position in the pybullet simulation.

Input Parameters:

	Parameters	Type	Description
required	to_global	bool	Return end-effector position in local coordinate system or global coordinate system

7.1.4 *compute_tcp_orientation*

Compute the tcp orientation of the end-effector with a given joint configuration and return it as euler angles and a flatten numpy array.

Input Parameters:

	Parameters	Type	Description
required	q	numpy	Vector of the joint configuration as a flatten numpy array

7.1.5 *compute_transformation_matrix*

Compute the transformation matrix for the target end-effector position as a SE3 object.

Input Parameters:

	Parameters	Type	Description
required	target_position	list/numpy	Vector of the target position. Can be a 3-dimensional list or a flatten 3-dimensional array
required	target_orientation	list/numpy	Vector of euler angles. Can be a 3-dimensional list or a flatten 3-dimensional array

7.1.6 *ikine_rtb*

Compute the inverse kinematic with the robotics-toolbox-library.

Input Parameters:

	Parameters	Type	Description
required	T	SE3	The desired end-effector pose or pose trajectory
required	ilimit	int	maximum number of iterations (default 500)
required	rlimit	int	maximum number of consecutive step rejections (default 100)
required	tol	float	final error tolerance (default 1e-10)
required	search	bool	if True, then global search, else local search
required	slimit	int	maximum number of search attempts
optional	q0	numpy	initial joint configuration (default all zeros) as a flatten numpy array

7.2 Inverse Kinematics

Calculates inverse kinematics. Class can do collision check in pybullet simulation and searching all joint configuration of a given trajectory

7.2.1 *__init__*

Input Parameters:

	Parameters	Type	Description
required	robot	RobotSetup	Object dscription of the robot model with all important information
required	all_bodies	list	list of all bodies in the simulation
required	num_attempts	RobotSetup	maximum number of attempts for searching an inverse kinematic
required	collision	bool	Do collision check in simulation
required	ilimit	int	maximum number of consecutive step rejections (default 100)
required	rlimit	int	Object description of the robot model with all important information
required	tol	int	final error tolerance (default 1e-10)
required	search	bool	if True, then global search, else local search
required	slimit	int	maximum number of search attempts

Initialized Parameters:

Parameters	Type	Description
robot	RobotSetup	Object description of the robot model with all important information
all_bodies	list	list of all bodies in the simulation
movable_joints	list	List of id numbers of the movable joints
num_attempts	int	maximum number of attempts for searching an inverse kinematic
collision	bool	do collision check in simulation
ilimit	int	maximum number of iterations (default 500)
rlimit	int	maximum number of consecutive step rejections (default 100)
tol	float	final error tolerance (default 1e-10)
search	bool	if True, then global search, else local search
slimit	int	maximum number of search attempts
history	list	List of all found configuration
test_successful	bool	Boolean for checking if exercise 2 task of the transformation-matrix is correct implemented
T	SE3	Default transformation matrix

7.2.2 *collision_check*

Check collision with a specific joint configuration. Return boolean for collision. If true, robot has a collision with a different body in simulation

Input Parameters:

	Parameters	Type	Description
required	q	numpy	Vector of the joint configuration as a flatten numpy array

7.2.3 *search_ik*

Search inverse kinematic. Return a list of the found tcp position, the joint position and a boolean value if a configuration is found.

Input Parameters:

	Parameters	Type	Description
required	input_tuple	tuple	Requires a 3D flatten numpy vector of the target tcp position, tcp orientation (euler angles). Requires a target body id for ignoring a collision in the pybullet simulation (int or None)
optional	search_type	bool	Search new inverse kinematic joint position if tcp position is already in tolerance, else do nothing
optional	q0	numpy	Start position for searching a new inverse kinematic
optional	local_tuple	bool	if True, then local inverse kinematic search, else global inverse kinematic search

7.2.4 *joint_space_search*

Method for searching a trajectory in the joint space. Return a boolean value for a found path. Trajectory is saved in history.

Input Parameters:

	Parameters	Type	Description
required	tcp_position	numpy	Vector of the target position. Can be a 3-dimensional list or a flatten 3-dimensional
optional	tcp_orientation	numpy	Vector of euler angles. Can be a 3-dimensional list or a flatten 3-dimensional array
optional	attempts	int	Attempts for trying to search a solution

7.2.5 *cartesian_search*

Method for searching a trajectory in the cartesian space. Return a boolean value for a found path. Result is saved in history as a list

Input Parameters:

	Parameters	Type	Description
required	tcp_position	numpy	Matrix of the target position. Is a 3-dimensional matrix with a shape of (n, 3)
optional	tcp_orientation	numpy	Matrix of the target orientations. Is a 3-dimensional matrix with a shape of (n, 3)
optional	attempts	int	Attempts for trying to search a solution

7.2.6 *search_ik_plan*

Search inverse kinematic for a target positions and target orientation with multiple

Input Parameters:

	Parameters	Type	Description
required	input_tuple	numpy	Gets list of tcp positions and tcp orientations (see cartesian search or joint space search)
optional	mode	bool	Mode for space search. if true, do joint space search, else cartesian search
optional	attempts	int	Attempts for trying to search a solution

7.3 PolynomialTrajectory

Class for computing a polynomial trajectory in the joint space.

7.3.1 *__init__*

Input Parameters:

	Parameters	Type	Description
required	robot	RobotSetup	Object dscription of the robot model with all important information

Initialized Parameters:

Parameters	Type	Description
robot	RobotSetup	Object description of the robot model with all important information

7.3.2 *vec_2_matrix*

Converts a vector to a matrix and return a numpy matrix of a shape of (n, len(vec))

Input Parameters:

	Parameters	Type	Description
required	vec	numpy	Flatten numpy array that should be n-times copy
required	n	int	Number of rows that should be copied

7.3.3 *linear_interpolation*

Build a straight line by using linear interpolation form one vector to another vector. Return a (n, 3) numpy array for the linear interpolated

Input Parameters:

	Parameters	Type	Description
required	target	numpy	Vector of the current point as a flatten numpy array
required	current	numpy	Vector of the current point as a flatten numpy array
required	time_steps	int	Execution time form the current point to the target point in seconds
required	step	int	step size of the pybullet execution time

7.3.4 *compute_trajectory*

Compute the joint position, joint velocity and joint acceleration for a trajectory. Return joint position, joint velocity and joint acceleration as a numpy array as a shape of (n, dof)

Input Parameters:

	Parameters	Type	Description
required	tcp_joint_position	tuple	Start and end tcp joint position as a numpy flatten array
required	qd_tuple	tuple	Start and end tcp joint velocity as a numpy flatten array
required	qdd_tuple	tuple	Start and end tcp joint acceleration as a numpy flatten array
required	t	tuple	Execution time range. Fist element has to be ALWAYS zero. Execution stop has to be an integer.
required	steps	int	step size of the pybullet execution time

7.4 PathPlanning

Class for planning a trajectory in cartesian space and joint space.

7.4.1 __*init*__

Input Parameters:

	Parameters	Type	Description
required	robot	RobotSetup	Object dscription of the robot model with all important information
required	all_bodies	list	list of all bodies in the simulation
required	movable_bodies	list	maximum number of consecutive step rejections (default 100)
required	collision	bool	Boolean for collision checking in the pybullet simulation
required	steps	float	Simulation step size
required	visualization	bool	Boolean for visualize tcp position the pybullet simulation

Initialized Parameters:

Parameters	Type	Description
robot	RobotSetup	Object description of the robot model with all important information
all_bodies	list	list of all bodies in the simulation
movable_joints	list	List of id numbers of the movable joints
num_attempts	int	maximum number of attempts for searching an inverse kinematic solution
collision	bool	Boolean for collision checking in the pybullet simulation
steps	int	Simulation step size. By default 60 steps is one seconds in the pybullet simulation.
visualization	bool	Boolean for visualize tcp position the pybullet simulation
dyn	Dynamic	Inizialise the dynanic model of the robot
q	numpy	Add current joint positions in the pybullet simulation as a (1, dof) numpy array. This variable is for saving all joint position in the trajectory and save it as a (n, dof)
qd	numpy	Add current velocity of the joints in the pybullet simulation as a (1, dof) numpy array. This variable is for saving all velocities in the trajectory and save it as a (n, dof)
qdd	numpy	Inizialize a numpy zero array with a shape of (1, dof) . This variable is for saving all acceleration in the trajectory and save it as a (n, dof)
eef_positions	list	Stores all tcp poisition of the end-effector position in global coordiante systems in a list
draw_trajectory_lines	list	Stores all id numbers from the draw line in pybullet simulation
successful_test	bool	Default transformation matrix

7.4.2 *draw_trajectory*

Draws a given trajectory into the pybullet simulation

Input Parameters:

	Parameters	Type	Description
optional	q	numpy	joint configuration of the trajectory with a shape of (n, dof)
optional	local	bool	If input of trajectory is in local or global space
optional	color	tuple	Tuple of RGB color representation
optional	width	float	Width of the line

7.4.3 *plan_execution_path*

Plans a trajectory in joint space and cartesian space. Return True, if a plan is found for a given path, else None

Input Parameters:

	Parameters	Type	Description
optional	tcp_positions	numpy	Numpy array of tcp target position in global coordiante system of a shape of (n, 3)
optional	tcp_orientations	bool	Numpy array of tcp target orientations in euler angles in a shape of (n, 3)
optional	execution_pattern	list	Execution time of the duration in the trajectory sections as integers
optional	execution_time	float	Width of the line
optional	velocities	numpy	Start and end velocity of every trajectory section. Needs shape of (2, dof) and set on every section
optional	accelerations	numpy	Start and end acceleration of every trajectory section. Needs shape of (2, dof) and set on every section

7.4.4 *clear_trajectory_lines*

Clears and delete all lines in the pybullet simulation

7.4.5 *clear_variables*

Set all important variables for creating a trajectory to __*init*__ state and clears all variables

7.5 Dynamic

Class for calculating the dynamics of a model.

7.5.1 __*init*__

Input Parameters:

	Parameters	Type	Description
required	robot	RobotSetup	Object dscription of the robot model with all important information
required	gravity	list/numpy	Set individual gravity to dynamic model, else loading normal earth gravity
optional	brake_task	bool	Boolean for exercise 5. Set all joints > 1 to zero. These joint are now connected

Initialized Parameters:

Parameters	Type	Description
robot	RobotSetup	Object description of the robot model with all important information
dof	int	Number of degrees of freedoms
R	list	List of rotations matrix
w	list	angular velocity vector as a flatten 3-dimensioanl numpy array and stored for all joints in a list
wd	list	angular acceleration vector as a flatten 3-dimensional numpy array and stored for all joints in a list
v	list	linear translation velocity vector as a flatten 3-dimensional numpy array and stored for all joints in a list
vd	list	linear translation acceleratioan vector as a flatten 3-dimensional numpy array and stored for all joints in a list
vdc	list	linear accceleartion of the center of mass of all joints as a flatten 3-dimenional numpy array and stored for all joints in a list
F	list	Acting forces in the center of mass of all joints as a flatten 3-dimensional numpy array and stored for all joints in a list
N	list	Acting torques in the center of mass of all joints as a flatten 3-dimensional numpy array and stored for all joints in a list
I	list	Inertia matrix of the shape of 3x3. For all joints in the body saved in that list
e	list	orthogonal unit vector as a flatten 3-dimensional numpy array and stored for all joints in a list
t	list	translation vector (sometimes r is used) of every joint as a flatten 3-dimensional numpy array stored in a list
G	list	gears of the joints as floats and stored in a list
Jm	list	motor inertia of all joints as floats stored in a list
B	list	motor viscous friction of all joints as floats stored in a list
Tc	list	motor Coulomb friction (1x2 or 2x1) as a flatten numpy array stored in a list
center_of_mass	list	center of mass of all joints as a flatten 3-dimensional numpy array stored in a list
masses	list	masses of the joints as floats and stored for all joints in a list
a_grav	float	gravity that should be used for the dynamics

7.5.2 *friction*

Calculate friction and feturn frictions as a numpy array as a flatten numpy array (dof)

Input Parameters:

	Parameters	Type	Description
required	qd	numpy	Joint velocity as a flatten numpy array
required	j	bool	Index of the joints

7.5.3 *forward_recursion*

Computes the forward recursion and return the center of gravity attacked on torque N and the center of gravity of the joint F as a 3x3 numpy array

Input Parameters:

	Parameters	Type	Description
required	qd	numpy	joint velocity as a flatten numpy array
required	qdd	numpy	joint acceleration as a flatten numpy array
required	gravity	list	Set individual gravity to dynamics, else loading normal earth gravity

7.5.4 *backward_recursion*

Compute the forecs for all joints as a flatten numpy array

Input Parameters:

	Parameters	Type	Description
required	qd	numpy	joint velocity as a flatten numpy array
required	qdd	nunpy	joint acceleration as a flatten numpy array
required	N	list	Center of gravity attacked on torque as list full of 3x3 numpy array
required	F	list	Center of gravity of the joint as a list full of 3x3 numpy array

7.6 Grasp

Class for grasping an object in the pybullet simulation. Depends on the URDF file.

7.6.1 *__init__*

Input Parameters:

	Parameters	Type	Description
required	robot	RobotSetup	Object description of the robot model with all important information
required	target_object	int	ID of the pybullet object
optional	kp	list/numpy	Positioning gain as a numpy flattan array as a shape of (number of fingers,)
optional	kv	list/numpy	Velocity gain as a numpy flattan array as a shape of (number of fingers,)
optional	control_mode	bool	Pybullet control mode like POSITION_CONTROL, VELOCITY_CONTROL and TORQUE_CONTROL

Initialized Parameters:

Parameters	Type	Description
gripper_limits	numpy	Gripper limits of a shape of (number of fingres, 2)
cid	int	unique id returned by createConstraint (pybullet)

7.6.2 *motion_control_grasp*

Grasp object in pybullet simulation with motion control

7.6.3 *motion_control_open_gripper*

Open gripper in pybullet simulation with motion control

7.7 MotionControl

Class for calculating the dynamics of a model.

7.7.1 *__init__*

Input Parameters:

	Parameters	Type	Description
required	robot	RobotSetup	Tuple of joint position, joint velocity and joint acceleration as a (n, dof) numpy array
required	trajectory	tuple	Set individual gravity to dynamic model, else loading normal earth gravity
optional	all_bodies	bool	list of all bodies in the simulation
optional	check_collision	bool	Boolean for checking collision in pybullet simulation
optional	target	int	Pybullet object ID that should be grasp in the simulation
optional	execution_times	list	Execution time for grasping and dropping an object in the pybullet simulation
optional	control_mode	int	Pybullet control mode like POSITION_CONTROL, VELOCITY_CONTROL and TORQUE_CONTROL
optional	kp	numpy	Positioning gain as a numpy flattan array as a shape of (dof,)
optional	kv	numpy	Velocity gain as a numpy flattan array as a shape of (dof,)
optional	dt	float	Sleeping time in the pybullet simulation for the next iteration in pybullet
optional	buffer_size	int	Size of maximum values that should be tracked and saved (i.e. position, velocity, acceleration and forces)

Initialized Parameters:

Parameters	Type	Description
joint_states	RobotSetup	Object description of the robot model with all important information
track_position	ReplayBuffer	Number of degrees of freedoms
track_velocity	ReplayBuffer	List of rotations matrix
track_acceleration	ReplayBuffer	angular velocity
track_forces	ReplayBuffer	angular acceleration
t	int	time-step for dynamic exercise task

7.7.2 *check_motion_collision*

Check if a collision appears in a trajectory

7.7.3 *joint_tracking*

Method for tracking position and velocity and add it to the queue

7.7.4 *execute_motion_control*

Execution a trajectory in the pybullet simulation

Input Parameters:

	Parameters	Type	Description
required	input_tuple	tuple	Tuple with a size of 2. Get row index of the grasp and drop position in the trajectory
required	tracking	bool	Boolean state for tracking joint position, joint velocity, joint acceleration and joint forces

7.7.5 *execute_dynamic_motion_control*

Execute dynamic motion control in pybullet simulation

Input Parameters:

	Parameters	Type	Description
required	q_t	numpy	joint position step as a flatten numpy array with a shape of (dof,)
required	qd_t	numpy	joint velocity step as a flatten numpy array with a shape of (dof,)
required	qdd_t	numpy	joint acceleration step as a flatten numpy array with a shape of (dof,)
required	tau	numpy	forces step as a flatten numpy array with a shape of (dof,)

8 Robot Utils

8.1 Setup Robot in Simulation

8.1.1 $get\_initial\_grasp\_type$

Load predefined joint position from a dictionary and return the joint configuration of the type of grasp position as a list. Uses the dictionary $INITIAL\_GRASP\_POSITIONS$ for setting a robot to a specific position. If parameter is $None$, then its load the QR list. $INITIAL\_GRASP\_POSITIONS$ and QR are defined in Simulation/Robots/robots_utils.py

Input Parameters:

	Parameters	Type	Description
optional	type_of_grasp	str	Key name in the dictionary . $INITIAL\_GRASP\_POSITIONS$

8.1.2 $get\_joint\_from\_model$

Get ID numbers from a given robot limb. Data are a list of the robot joint names. Pybullet searches the name in the URDF file and return an individual ID for this joint.

Attentions: Sometimes pybullet does not find the name in the URDF file. But the ID of the joint is still controlable in the simulation.

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc.
required	limbs	str	Key name of the dictionary $ROBOT\_GROUPS$. Key name can be arm or $gripper$

8.1.3 set_arm_config

Set joint configuration into the simulation environment with a given robot limb .

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc
required	limbs	str	Key name of the dictionary $ROBOT\_GROUPS$. Key name can be arm or $gripper$
required	config	list/numpy	Joint configuration, that should be set. Limbs have to define in Simulation/Robots/robot_utils.py in the dictionary ROBOT_GROUPS.

8.1.4 $open\_gripper$

Open gripper in the simulation environment. Define $ROBOT\_GROUPS\_ID$ first, if you are using a different Robot. You can fine the dictionary ROBOT_GROUPS_ID in file Simulation/Robots/robot_utils.py

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc

8.1.5 $close\_gripper$

Close gripper in the simulation environment. Define $ROBOT\_GROUPS\_ID$ first, if you are using a different Robot. You can fine the dictionary ROBOT_GROUPS_ID in file Simulation/Robots/robot_utils.py

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc

8.2 Random arm setup

8.2.1 $random\_arm\_config$

Add Variance to the current joint configurtion of a robot arm by using attemps. Used for searching all joint position in a trajectory. If robot reached a joint limit or is in singularity, start from new random arm position by adding different strength of variance. Return new joint configuration as a flatten numpy array

Input Parameters:

	Parameters	Type	Description
required	robot	RobotSetup	Object description of the robot model with all important information
required	curr_joints	numpy	Current joint position as a flatten numpy array
required	i	int	Current step
required	attemps	bool	Maximum attempts

8.2.2 $get\_sample\_arm\_config$

Sample current joint position with no, low and medium variance. Clip new joint configuration to limits. Return new joint configuration as a flatten numpy array

Input Parameters:

	Parameters	Type	Description
required	robot	RobotSetup	Object description of the robot model with all important information
required	joint_positions	numpy	Current joint position as a flatten numpy array
required	limits	numpy	Limit of the robot model which clips the new joint configuration to the limits. Shape is a (dof, 2) numpy array
required	random_state	int	Value for no, low and medium variance to the current joint position

8.3 Math

8.3.1 *transform_reference_systems*

Transform the reference system form the global world coordinate system to local robot coordinate system or from the local robot coordinate system into the global world coordinate system.

Input Parameters:

	Parameters	Type	Description
required	base_position	numpy	Base position in the global pybullet simulation coordinate system
required	tcp_position	numpy	Current [X, Y, Z] local or global coordinte position
required	to_world	bool	If true, transform position in global coordinate system, else into local robot coordinate system

8.3.2 *calculate_distance*

Measures the distances between two vectors. Usecase: Measures the distance of the TCP position and the TCP goal position.

Input Parameters:

	Parameters	Type	Description
required	vector1	numpy	flatten numpy array
required	vector1	numpy	flatten numpy array

8.3.3 *check_costum_limits_trajectory*

Check limits of a given trajectory with costum limits.

Input Parameters:

	Parameters	Type	Description
required	values	numpy	Shape of a (n, dof)-numpy array of the trajectory
required	min_values	numpy	Minimum values of the costum limits as a flatten numpy array
required	max_values	numpy	Maximum values of the costum limits as a flatten numpy array

8.4 CoordinateSystem

8.4.1 *__init__*

Class for constructing and drawing coordinate systems into the pybullet simulation environment. Saves all new coordinate systems into a dictionary and also can delete coordinate systems in the pybullet simulation.

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc

8.4.2 *get_z_point, get_x_point, get_y_point*

Return a three-dimensional flatten numpy vector. Its the direction of the axis.

8.4.3 *transform_points*

Calculates the end-coordinate system of the direction of the axis.

Input Parameters:

	Parameters	Type	Description
required	rotation_matrix	numpy	3x3 rotation matrix
required	point	numpy	direction of the axis as a flatten numpy vector.

8.4.4 *delete_debug_line*

Delete coordinate system in pybullet simulation.

Input Parameters:

	Parameters	Type	Description
required	idx/str	numpy	Index/name of the debug line that should be deleted

8.4.5 *clear_all*

Delete all coordinate systems in pybullet simulation and clears the dictionary.

8.4.6 *draw_coordinate_system*

Draw coordinate system into pybullet simulation and store pybullet ids in dictionaries.

Input Parameters:

	Parameters	Type	Description
required	position	numpy	Global [X,Y,Z] of the start position for the coordinate system
required	rotation_matrix	numpy	3x3 rotation matrix for the coordinate system orientation
name	name	str/int	Name of the coordinate system
length	length	float	Length of the line that should be drawn into the simulation

8.4.7 *to_rotation_matrix*

Calculate the rotation matrix from a joint id and return rotation matrix as a 3x3 numpy matrix.

Input Parameters:

	Parameters	Type	Description
required	joint	int	ID number of the joint

8.4.8 *convert_orientation_2_euler*

Convert rotation matrix to euler angles and return euler angles as a list

Input Parameters:

	Parameters	Type	Description
required	rotation_matrix	numpy	3x3 rotation matrix

8.4.9 *convert_orientation_2_matrix*

Convert euler angles into rotation matrix and return rotation matrix as a 3x3 numpy array.

Input Parameters:

	Parameters	Type	Description
required	euler	list/numpy	Euler angles, flatten numpy array or 3x1 numpy array

8.5 CoordinateSystemControl

8.5.1 *__init__*

Control TCP coordinate system with app and set/update joint coordinate systems. Uses threading for updating joints faster in the pybullet simulation. If a $Joint_i$ needs a new coordinate system, every $Joint_{i:n}$ also gets a new coordinate systems.

Input Parameters:

	Parameters	Type	Description
required	robot	RobotSetup	Object description of the robot model with all important information
required	visualization	bool	Boolean for updating/setting joint coordinate system into the pybullet simulation

8.5.2 *update_coordinate_system*

Delete last coordinate system of this current joint and draw new coordinate system.

Input Parameters:

	Parameters	Type	Description
required	joint_position	numpy	Global XYZ of the start position for the coordinate system
required	rotation_matrix	numpy	3x3 rotation matrix for the coordinate system orientation
required	joint	int	ID of the current joint

8.5.3 *multi_draw_coordinate_systems*

Draw list of coordinate systems into pybullet simulation.

Input Parameters:

	Parameters	Type	Description
required	values	list	list of tuples of joint position (numpy/flatten), rotation matrix (3x3 numpy array) and the joint id (int)

8.5.4 *multi_update_coordinate_systems*

Update list of coordinate system. Delete the current joint coordinate system and draw new one.

Input Parameters:

	Parameters	Type	Description
required	values	list	list of tuples of joint position (numpy/flatten), rotation matrix (3x3 numpy array) and the joint id (int)

8.5.5 *divide_coordinate_tasks*

Separate joints that should be updated. Used for updating coordinate systems in pybullet faster with threading and return separated list of updating coordinate system tasks. Right now, we are using only two treads for updating coordinate systems, because of the hardware limitation of the students.

Attention: Avoid using more threads for this task, because for the exercises we need more threads and processes, like, app, pybullet simulation, plot-app, etc. For now, we assume that a normal laptop has 4 cores. The limitation of maximum used threads are 8 at the same time.

Input Parameters:

	Parameters	Type	Description
required	joints_to_update	list	List of joint ids, that should be updated

8.5.6 *update*

Update coordinate systems in the pybullet simulation. It configures the list of updated joints, seperates the list in two threads and updates the coordiantes systems.

Input Parameters:

	Parameters	Type	Description
required	new_joint_position	list	List of all new joint positions as flatten numpy array

8.6 RobotTCPControl

8.6.1 *__init__*

Control TCP position in pybullet simulation with app and uses the configured workspace from exercise 2 and draw the start position of the tcp position.

Input Parameters:

	Parameters	Type	Description
required	robot	RobotSetup	Object description of the robot model with all important information
required	workspace	list	List of the workspace parameters of the robot, that should be controlled

8.6.2 *update*

Update TCP position in pybullet simulation

Input Parameters:

	Parameters	Type	Description
required	new_position	list	Position and orientation as euler angle in one list.

8.6.3 *add_new_tcp_target*

Add new tcp coordinate system in pybullet simulation.

Input Parameters:

	Parameters	Type	Description
required	parameters	list	Position and orientation as euler angle in one list
required	adding_tcp	bool	if true, adding new tcp position for a trajectory that should be followed. Needed for exercise 4

8.6.4 *swap_tcp_target*

Swap tcp targets and order the new trajectory in pybullet simulation.

Attention: In pybullet it is better to delete all coordinate systems and draw new ones. Errors can be easy avoided

Input Parameters:

	Parameters	Type	Description
required	tcp_targets	list	List of the index of the tcp targets that should be swapped

8.6.5 *delete_single_tcp_target*

Delete tcp target in a trajectory in pybullet simulation and draw new TCP sequence.

Input Parameters:

	Parameters	Type	Description
required	delete_targets	list	List of indexes that should be deleted in the trajectory

8.6.6 *delete_tcp_targets*

Delete all tcp targets or/and reset trajectory target points.

Input Parameters:

	Parameters	Type	Description
required	reset_tcp	bool	Boolean for resetting a trajectory

8.6.7 *save_path_plan*

Save trajectory into a file.

Input Parameters:

	Parameters	Type	Description
required	path	str	Path were the file should be saved

8.7 UserDebugControl

Class for setting joints in pybullet simulation. Used for exercise 2 forward kinematic

8.7.1 __*init*__

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc
required	robot_joints	list	list of the joint ids

Initialized Parameters:

Parameters	Type	Description
body	int	body unique id, as returned by loadURDF etc
robot_joints	list	list of the joint ids

8.7.2 *update_robot_joints*

Method for new debug joint parameters and set joint posiiton in simulation

Input Parameters:

	Parameters	Type	Description
required	values	list	New values that should be set in the pybullet simulation

9 PyBullet

9.1 Client

9.1.1 *set_client*

Set new client for the pybullet simulation if multiply simulation is started. Standard client is 0.

Input Parameters:

	Parameters	Type	Description
required	client	int	pybullet simulation client

9.1.2 *get_client*

Returns the pybullet simulation client as an integer value. By default with only one pybullet simulation is the client 0.

9.2 Simulation Connection

9.2.1 *is_connected*

Check if simulation is still active and returns a boolean value. If true, simulation is still active.

9.2.2 *get_connection*

Get connection of a specific simulation. Check connection of the client

Input Parameters:

	Parameters	Type	Description
optional	client	int	pybullet simulation client

9.2.3 *Connect/Disconnect*

PyBullet has to connect to the physics simulation, by sending different commands to the client-server API. The client connects to the physics server and the server retruns the status. You can read more about it in the Pybullet Documentation: `https://docs.google.com/document/d/10sXEhzFRSnvFcl3XxNGhnD4N2SedqwdAvK3dsihxVUA/edit#heading=h.2ye70wns7io3`

The connect method will automatically set the debug visualizer to False and set the time step to t = 1 / 60 (See Sec. 4.4.1). This is the update cycle of the pybullet simulation. Any trajection should be built according to this cycle.

Input Parameters:

	Parameters	Type	Description
optional	c	int	PyBullet has some built-in physics servers: DIRECT and GUI. Both GUI and DIRECT connections will execute the physics simulation and rendering in the same process as PyBullet.
optional	dt	float	Each time you call 'stepSimulation' the timeStep will proceed with 'timeStep'.

The disconnect method closes the pybullet simulation.

9.2.4 *has_active_gui*

Check if pybullet simulation is active. Returns True, if simulation is active.

9.3 Simulation Setup

9.3.1 *step_simulation*

Set step next in simulation for $MotionControl$ (see Sec. 2 TODO). By Default, running one second in MotionControl is equal to 60 pybullet steps with a $sleep\_time$ of 1./240.

Input Parameters:

	Parameters	Type	Description
optional	sleep_time	float	Sleep step in simulation

9.3.2 *real_simulation*

Set pybullet simulation to realtime. Calling stepSimulation is needed now for $MotionControl$ see sec 2 TODO.

Input Parameters:

	Parameters	Type	Description
optional	enable	bool	Activates realtime simulation in pybullet

9.3.3 *set_time_step*

Set step time for one simulation cyclus. it set the physics engine timestep

Input Parameters:

	Parameters	Type	Description
required	t	float	each time you call $stepSimulation$ the timestep will proceed with timestep

9.3.4 *activate_gravity*

For more details see Sec. 9.2.3 TODO

9.3.5 *debug_visualizer*

Pybullet debug visualizer option. Deactivates unnecessary GUI options in the simulation.

9.4 Load Body in Simulation

9.4.1 *load_pybullet_model*

?? Load model into simulation. Return an integer value for calling the object in the simulation. The loadURDF will send a command to the physics server to load a physics model from a Universal Robot Description File (URDF). The URDF file is used by the ROS project (Robot Operating System) to describe robots and other objects.

Input Parameters:

	Parameters	Type	Description
optional	filename	str	a relative path to the URDF file on the filesystem of the physics server
optional	fixed_base	bool	Set model or model base as fixed (cannot be moved)
optional	scale	float	scaling will apply a scale factor to the URDF model

Attention: Do not load a model with a start position and start orientation with this function. The internal pybullet method has already a method that can load a model with a given start position and orientation, but this causes troubles in the simulation. First load a model with $load\_pybullet\_model$ and then change the orientation and start position with $set\_position$ (see Sec. 4.5), $set\_orientation$ (see Sec. 4.5) or $set\_position\_and\_orientation$ (see Sec. 4.5).

9.4.2 $load\_model$

?? Load body in simulation with a specific position and orientation. Return individual simulation ID for the model.

Input Parameters:

	Parameters	Type	Description
optional	filename	str	Path of the body that should be loaded into the simulati
optional	position	list/numpy	3x1 vector for [x, y, z]-position
optional	orientation	list/numpy	3x1 vector for [x, y, z]-orientation as euler angles
optional	pose	tuple	position and orientation of the body
optional	fixed_base	list/numpy	3x1 vector for [x, y, z]-orientation as euler angles

9.4.3 get_URDF_flags

Flags that should be set by loading a body into the pybullet simulation

Input Parameters:

	Parameters	Type	Description
optional	cache	bool	Path were the file is saved

9.4.4 HideOutput, Saver and LockRenderer

Used load model in pybullet simulation and is for disabling rendering temporary makes adding objects faster, saving simulaton state, etc.

9.5 Angles

9.5.1 $quaternion\_from\_euler$

Transform euler angles to quaternion angles

Input Parameters:

	Parameters	Type	Description
optional	orientation	list/tuple/numpy	Euler angles as a [3x1] or a flatten numpy array

9.5.2 *euler_from_quaternion*

Transform quaternion angles to euler angles

Input Parameters:

	Parameters	Type	Description
optional	orientation	list/tuple/numpy	Quaternion angles as a [4x1] list or a flatten numpy array

9.5.3 *matrix_from_quaternion*

Transform quaternion angles to rotation matrix

Input Parameters:

	Parameters	Type	Description
optional	orientation	list/tuple/numpy	Quaternion angles as a [4x1] list or a flatten numpy array

9.5.4 *quaternion_from_matrix*

Transform quaternion angles to rotation matrix

Input Parameters:

	Parameters	Type	Description
optional	matrix	numpy	3x3 rotation matrix

9.5.5 *quaternion_from_euler*

Transform euler angles to quaternion angles

Input Parameters:

	Parameters	Type	Description
optional	orientation	list/tuple/numpy	Relative path of project head

9.6 Position and Orienation of bodies

9.6.1 *set_position*

Get global pybullet position of a pybullet body

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc
required	position	tuple/list/numpy	[x, y, z]-position of the global pybullet simulation

9.6.2 *get_position*

Get global pybullet position of a pybullet body. Retuns the [X, Y, Z] position of the body as a list.

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc
required	orientation	tuple/list/numpy	[x, y, z]-position of the global pybullet simulation

9.6.3 *set_orientation*

Set orientation of a body in the pybullet simulation.

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc
required	orientation	tuple/list/numpy	Euler angle in respect of the pybullet simulation coordinate systems

9.6.4 *get_orientation*

Get orientation of a pybullet body. Returns the orientation as a [4x1] quaternion angle as a list

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc

9.6.5 *set_position_and_orientation*

Set position and orientation of a pybullet body

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc
required	orientation	list/tuple/numpy	Orientation in euler angles

9.7 Joint Info

9.7.1 *get_joint_info*

Return information of a specific joint as a dictionary, like, JointIndex, JointName, JointType, uIndex, flags, jointDamping, jointFriction, jointLowerLimits, jointUpperLimits, jointMaxForce, jointMaxVelocity, linkName, jointAxis, parentFramePos, parentFrameOrn, parentIndex

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc
required	joint	int	joint id of the body

9.7.2 *get_joint_name*

Return joint name from joint info.

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc
required	joint	int	joint id of the body

9.7.3 *get_joint_name*

Return link name from joint info as a string.

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc
required	joint	int	link id of the body

9.7.4 *get_joints*

Get all joints as a list.

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc

9.7.5 *get_joint_config*

Return all information from a joint (joint-limit, current joint-velocity, ect). See pybullet documentation

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc
required	joint	int	joint id of the body

9.7.6 *get_all_joint_config*

Return all information as a dicitonary from all joints (joint-limit, current joint-velocity, ect). See pybullet documentaiton

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc

9.7.7 *get_all_joint_config*

Return current joint velocity. Return velocity of a joint as a float

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc
required	joint	int	joint id of the body

9.7.8 *get_all_joint_velocity*

Return all velocities of all joints in the body. Return velocities as a list

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc

9.7.9 *get_all_joint_forces*

Return all forces of all joints of the body. Return forces as a list

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc

9.7.10 *get_joint_position*

Return the current joint position/angle of the given joint. Return the current joint position as a float

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc
required	joint	int	joint id of the body

9.7.11 *save_all_joint_names_and_links*

Save all joint and link names in a CSV file

Input Parameters:

	Parameters	Type	Description
required	dict_joints	dict	Discription of the robot model

9.8 Get Joint Limits

9.8.1 *get_limits_of_joint_info*

Return joint limits of a body. Returns joint minimum and maximum limit as a list

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc
required	joint	int	joint id of the body

9.8.2 *get_all_joint_limits*

Return all joint limits of a body. Return all joint limits as a numpy array with a shape of [dof, 2]

Attention: The difference between *get_all_joint_limits* and *get_all_limbs_limits* is, that in *get_all_limbs_limit*we are only taking joint, that can be set. I. e. *joint_hand* limits is sometimes not needed for the description

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc

9.8.3 *get_joint_limits_from_name*

Return all joint limits of a body. Robot can be described as a dictionary, Arm and Gripper (see *robot_utils.py* for an example)

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc
required	joint_name_dict	int	Dictionary of the joint-names

9.8.4 *get_all_limbs_limits*

Return only limbs joint limits of a body.

Attention: The difference between *get_all_joint_limits* and *get_all_limbs_limits* is, that in *get_all_limbs_limit* we are only taking joint, that can be set. I. e. *joint_hand* limits is sometimes not needed for the description

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc

9.8.5 *get_max_velocity*

Return the maximum velocity of a joint. Return velocity of a joint as a float

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc
required	joint	int	joint id of the body

9.8.6 $get\_all\_max\_velocities$

Return all maximum velocities of the body as a list

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc
required	joints	list	list of joint ids

9.8.7 get_max_force

Return the maximum force of a joint

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc
required	joint	list	joint id of the body

9.8.8 $get\_all\_max\_forces$

Return all maximum forces of the body as a list

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc
required	joints	list	list of joint ids

9.9 Robot

9.9.1 get_eef_index

Return TCP joint index. Need right urdf description. Return TCP joint index as an integer.

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc

9.9.2 $get\_number\_of\_joints$

Return number of joints of a body. Return number of joints as an integer

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc

9.9.3 *get_number_of_links*

Return number of links of a body. Return number of links as an integer

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc

9.9.4 *all_joints*

Return all joints as a list

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc

9.9.5 *all_links*

Return all links as a list

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc

9.9.6 *get_arm_gripper_joints*

Return joint ids of a pybullet body form a specified dictionary $ROBOT\_GROUPS$ defined in $Simulation/Robot/robot\_primitives.py$. Return gripper joints as a list

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc

9.9.7 *get_all_joint_position*

Return the joint position/angle of all joints in the body. Return the current joint positions as a list

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc

9.9.8 *get_limb_joints*

Return all joint and gripper ids of a predefined dictionary $ROBOT\_GROUPS\_ID$ in $Simulaiton/Robots/robot\_utils.py$. All fingers in the Gripper are defined as one id. Return all joint position as a list

9.9.9 *get_limb_positions*

Return all joint and gripper position of a predefined dictionary $ROBOT\_GROUPS\_ID$ in $Simulaiton/Robots/robot\_utils$. All fingers in the Gripper are defined as one id. Return all joint position as a list

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc

9.9.10 *joint_from_name*

Return ID of a joint-name. Return the ID of the joint-name as an int

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc
required	name	str	Name of the joint defined in the URDF file
required	save_values	bool	Boolean for saving all joint ids in a file

9.9.11 *joint_from_name*

Return ID of a link-name. Return the ID of the link-name as an int

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc
required	name	str	Name of the joint defined in the URDF file
required	save_values	bool	Boolean for saving all joint ids in a file

9.9.12 *set_joint_position*

Set single joint of the body in the pybullet simulation

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc
required	joint	int	joint id of the body
required	config	float	Value/Position of the joint ID in radians

9.9.13 *set_joint_positions*

Set multiple joint positions/angles in pybullet simulation

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc
required	joint	int	joint id of the body
required	config	list	Values/Positions of the joint ID in radians

9.10 Limits and Collision

9.10.1 *check_joint_limits*

Load limits of a given joint from URDF and check if joint is in limits. Return True if its in joint-limits, else False

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc
required	joint	int	joint id of the body

9.10.2 *is_in_joint_limits*

Check if current robot configuration of all the joints is in limits. Return True if its in joint-limits, else False

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc
required	joints	list	List of all the joint IDs, that should be checked

9.10.3 *check_limits*

Check if values is in costum limits and raise error otherwise

Input Parameters:

	Parameters	Type	Description
required	joints	list	List of all the joint IDs, that should be checked
required	values	list	List of all joint values, that should be checked
required	limits	list	One dimensional limit list

9.10.4 *check_velocity*

Check if values is in costum limits and raise error otherwise

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc
required	joints	list	List of all the joint IDs, that should be checked

9.10.5 *check_forces*

Check if values is in costum limits and raise error otherwise

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc
required	joints	list	List of all the joint IDs, that should be checked

9.10.6 *is_in_never_collision*

Check if a collision appears, but can be negligible. In Pybullet also appears when two links are connected. These can be ignored. See *Simulation.Robots.never_collision.py*. If link1 and link 2 is in *never_collision*, return true, else false.

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc
required	link1	int	Link ID of the model
required	link2	int	Link ID of the model
required	**kwargs	list	list of never kollision

9.10.7 *pairwise_link_collision*

Check if a collision appears of two objects in the simulation. Return True, when collision

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc
required	link1	int	Link ID of the model
required	body	int	body unique id, as returned by loadURDF etc
required	link1	int	Link ID of the model
required	**kwargs	list	list of never kollision

9.10.8 *link_collision*

Check link collision of two objects in the pybullet simulation with individual links. If collision, return true

Input Parameters:

	Parameters	Type	Description
required	body1	int	body unique id, as returned by loadURDF etc
required	link1	int	Link ID of the model
required	body2	int	body unique id, as returned by loadURDF etc
required	link1	int	Link ID of the model
required	distance	float	Maximum distance of a link collision

9.10.9 *object_collision*

Check collision of two objects in the pybullet simulation and visualize it in the simulation

Input Parameters:

	Parameters	Type	Description
required	body1	int	body unique id, as returned by loadURDF etc
required	body2	int	body unique id, as returned by loadURDF etc
optional	visualization	bool	Visualize collision and draw collision line in pybullet simulation
optional	distance	float	Maximum distance of a link collision

9.11 Visualization

9.11.1 *collision_line_size*

Compute end position of the collision line in the pybullet simulation. Return start- and end-position of the collision line in a list

Input Parameters:

	Parameters	Type	Description
required	collision	numpy	start value of the [X, Y, Z] Position of the collision as a flatten numpy array
optional	size	float	line size of the collision

9.11.2 *draw_collision_line*

Draw collision line in the pybullet simulation.

Input Parameters:

	Parameters	Type	Description
required	positions	list	start- and end-position of the collision line. Position values are in [X, Y, Z]-coordinate systems as flatten numpy arrays.
required	color	tuple	RGB color value
required	width	float	RGB color value
optional	lifetime	int	Duration of collision line, set in simulation

9.11.3 *draw_text_to_collision_line*

Write text to a collision line

Input Parameters:

	Parameters	Type	Description
required	collision_text	str	Text of the collision
required	text_position	list	End-position of the collision text
required	color	list/tuple	RGB color value
optional	text_size	float	Text size of the collision
optional	lifetime	int	Duration of collision line, set in simulation

9.11.4 *draw_line*

Draw line in Simulation and return individual user debug line id of in the simulation. With this id, the line can be deleted. Return user debug ID of the line and can be removed from the pybullet simulation

Input Parameters:

	Parameters	Type	Description
required	positions	list	start- and end-position of the collision line. Position values are in [X, Y, Z]-coordinate systems as flatten numpy arrays
required	text_position	list	End-position of the collision text
optional	color	list/tuple	RGB color value
optional	width	float	RGB color value
optional	lifetime	int	Duration of collision line, set in simulation

9.11.5 *draw_text*

Draw line in Simulation and return individual user debug line id of in the simulation. With this id, the line can be deleted and removed from the pybullet simulation. Return user debug ID of the text

Input Parameters:

	Parameters	Type	Description
required	text	str	Text, that should be draw in the simulation
required	text_position	list	End-position of the collision text
required	color	list/tuple	RGB color value
optional	text_size	float	Text size of the collision
optional	lifetime	int	Duration of collision line, set in simulation

9.11.6 *remove_debug_item*

Remove/delete user debug item in simulation

Input Parameters:

	Parameters	Type	Description
required	debug_item	int	Simulation user debug ID of the text or line

9.11.7 *remove_debug_items*

Remove and delete all debug items in simulation

9.12 Motion

9.12.1 *motor_control_individual*

Set stepwise motor control of a list of joints. Used for grasping an object in pybullet simulation.

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc
required	joints	list/numpy	List of joints ids
required	q_pos_desired	list/numpy	target joint position
required	control_mode	int	Type of control mode: Position Control, Velcoity Control, Torque Control
required	position_gain	list/numpy	Gain of the position
required	velocity_gain	list/numpy	Gain of the velocity
required	dt	float	timestep of the simulation that should be paused

9.12.2 *motor_control*

Stepwise motor control in pybullet simulation with q, qd, or forces are required. Depends on *control_mode*. Joints and q, qd or forces has to be equal in shape

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc
required	joints	list/numpy	List of joints ids
required	control_mode	int	Type of control mode: Position Control, Velcoity Control, Torque Control
required	q	numpy	Target joint position as a flatten numpy array
required	qd	numpy	Target joint velocity as a flatten numpy array
required	forces	numpy	target joint force as a flatten numpy array
required	position_gain	list/numpy	Gain of the position
required	velocity_gain	list/numpy	Gain of the velocity
required	dt	float	timestep of the simulation that should be paused

9.12.3 *get_current_motor_joint_state*

Get current motor joints states, like position, velocity and forces. Return current joint, velocity and torques of the joints as a numpy array

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc

9.12.4 *change_dynamics_gripper*

Change dynamics of the gripper

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc
required	link_list	int	List of the links ids
optional	lateral_friction	float	lateral (linear) contact friction
optional	spinning_friction	float	torsional friction around the contact normal
optional	rolling_friction	float	torsional friction orthogonal to contact normal
optional	friction_anchor	float	enable or disable a friction anchor: positional friction correction (disabled by default, unless set in the URDF contact section)

9.12.5 *create_constraint*

createConstraint allows you to connect specific links of bodies to close those loops (see pybullet documentation). Return an unique id integer, that can be used to change or remove the constraint.

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc
optional	link1	int	parent body unique id
optional	link2	int	parent link index (or -1 for the base)
optional	joint_type	int	joint type: JOINT_PRISMATIC, JOINT_FIXED, JOINT_POINT2POINT, JOINT_GEAR
optional	joint_axis	tuple	joint axis, in child link frame - vec3
optional	parent_frame_position	tuple	position of the joint frame relative to parent center of mass frame.
optional	child_frame_position	tuple	position of the joint frame relative to a given child center of mass frame (or world origin if no child specified)

9.12.6 *create_constraint*

changeConstraint allows you to change parameters of an existing constraint

Input Parameters:

	Parameters	Type	Description
required	cid	int	unique id returned by createConstraint
optional	gear_ratio	float	the ratio between the rates at which the two gears rotate
optional	erp	float	constraint error reduction parameter
optional	max_forces	float	maximum force that constraint can apply

9.12.7 *create_constraint*

remove constraint allows you to remove parameters of an existing constraint

Input Parameters:

	Parameters	Type	Description
required	cid	int	unique id returned by createConstraint

9.13 Kinematic

9.13.1 *get_fkine_position*

Return the position of a specific link

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc
required	link_idx	int	link id of a joint
optional	compute_forward_kinematics	bool	if set to 1 (or True), the Cartesian world position/orientation will be recomputed using forward kinematics

9.13.2 *get_fkine_orientation*

Return the orientation of a specific link in quaternion angles

Input Parameters:

	Parameters	Type	Description
required	body	int	body unique id, as returned by loadURDF etc
required	link_idx	int	link id of a joint
optional	compute_forward_kinematics	bool	if set to 1 (or True), the Cartesian world position/orientation will be recomputed using forward kinematics

10 Utils

10.1 Saving to file and Loading from file

10.1.1 *save_list*

Save list in a CSV file

Input Parameters:

	Parameters	Type	Description
required	data	list	Data as a list
required	file_name	str	Path of the file

10.1.2 *save_numpy_array*

Save numpy array

Input Parameters:

	Parameters	Type	Description
required	data	list	Data as a list
required	file_name	str	Path of the file

10.1.3 *load_numpy_array*

Load numpy array

Input Parameters:

	Parameters	Type	Description
required	file_name	str	Path of the file

10.2 Converting Units

10.2.1 *convert_list_rad_2_grad*

Convert a list of radian values to grad

Input Parameters:

	Parameters	Type	Description
required	input_list	str	List of radians

10.2.2 *convert_list_grad_2_rad*

Convert a list of grad values to rad

Input Parameters:

	Parameters	Type	Description
required	input_list	str	list of grad

10.2.3 *rad_2_grad*

Convert rad to grad with a numpy array

Input Parameters:

	Parameters	Type	Description
required	value	str	Vector or matrix of a numpy array

10.2.4 *grad_2_rad*

Convert grad to rad with a numpy array

Input Parameters:

	Parameters	Type	Description
required	value	str	Vector or matrix of a numpy array

10.3 Others

10.3.1 *check_custom_limits*

Check limits of a given matrix/vector with costum limits.

Input Parameters:

	Parameters	Type	Description
required	values	numpy	Shape of a (n, dof)-numpy array of the trajectory
required	min_values	numpy	Minimum values of the costum limits as a flatten numpy array
required	max_values	numpy	Maximum values of the costum limits as a flatten numpy array

10.3.2 *clip_to_limits*

Clip numpy array values to a minimum or maximum

Input Parameters:

	Parameters	Type	Description
required	values	numpy	Shape of a (n, dof)-numpy array of the trajectory
required	min_values	numpy	Minimum values of the costum limits as a flatten numpy array
required	max_values	numpy	Maximum values of the costum limits as a flatten numpy array

10.3.3 *vec_2_matrix*

Convert vector to matrix by coping n-times a row vector

Input Parameters:

	Parameters	Type	Description
required	vec	numpy	Row numpy array with a shape of (1, m)
required	n	int	Number of rows that should be copied

10.4 ReplayBuffer

10.4.1 *__init__*

Buffer for storing data

Input Parameters:

	Parameters	Type	Description
required	buffer_size	int	Maximum buffer size of the stored data

Initialized Parameters:

Parameters	Type	Description
count	int	Variable for checking buffer_size
final_count	int	Number of elements that is added to the deque since initialization
buffer_size	int	Maximum buffer size of the stored data
buffer	deque	Buffer list

10.4.2 *add*

Add new values to buffer.

Input Parameters:

	Parameters	Type	Description
required	values	list/numpy	List of new values that should be put into the buffer .

10.4.3 *clear*

Clear queue.

10.5 AppProcess

10.5.1 *__init__*

Start new process for the User Interface

Input Parameters:

	Parameters	Type	Description
required	function	int	Function that should be paralized
required	tuple	int	All variables in a tuple handed over to the method function

Initialized Parameters:

Parameters	Type	Description
method	function	Function that should be paralized
input_values	tuple	Number of elements that is added to the deque since initialization
num_workers	int	Maximum number of workers
process	Process	Process of the application
app_values	Queue	Communication queue from app to simulation
simulation_values	Queue	Communication queue from simulation to app

10.6 MultiPlot

10.6.1 *__init__*

Start plot as a new process. Creates a (1, n)-multiplot.

Input Parameters:

	Parameters	Type	Description
required	titles	list	List of titles strings of the different column plots
required	suptitle	int	Name of the suptitle
required	supylabel	int	Name of the y axis

Initialized Parameters:

Parameters	Type	Description
fig	figure	Create a new figure, or activate an existing figure
axs	subplots	create number of single subplots

10.6.2 *show_plot*

Show multi-plot

10.6.3 *add_plot*

Add new values to all n-subtitles. Iterates new values through a tuple and add those values to single subplots with a specific color line.

Input Parameters:

	Parameters	Type	Description
required	input_value	list	List of new input data that should be added to the single subplots
required	label	str	label name of the new data that will be added to all subplots
required	color	tuple	Color of the line that will be added to all subplots.

10.7 MultiThreading

10.7.1 __*init*__

Start new threads.

Input Parameters:

	Parameters	Type	Description
required	num_workers	int	Maximum number of workers

Initialized Parameters:

Parameters	Type	Description
num_workers	int	Maximum number of workers

10.7.2 *start_threads*

Start threads.

Input Parameters:

	Parameters	Type	Description
required	threads	List	List of initialized thread that should be started

10.7.3 *join_threads*

Join threads.

Input Parameters:

	Parameters	Type	Description
required	threads	List	List of initialized thread that should be joined

10.7.4 *configure_threads*

Configures inizialised threads and starts threads.

Input Parameters:

	Parameters	Type	Description
required	methods	list	list of functions that should be started
required	args	list	list of tuples of arguments for the called method function

10.8 LivePlotDynamic

10.8.1 *run_plot*

Run robot dynamic plot in a tkinter app.

Input Parameters:

	Parameters	Type	Description
required	dof	int	Number of degrees of freedom of the robot
required	limits	numpy	Limits of the robot with the shape of (dof, 2)
required	value_queue	list	Queue for data transfer and communication with pybullet and tkinter app

10.8.2 *__init__*

Plot for robot dynamics integraded in a tkinter app. Initialize tkinter window

Input Parameters:

	Parameters	Type	Description
required	root	tk	TKinker master root object
required	dof	int	Number of degrees of freedom of the robot
required	limits	numpy	Limits of the robot with the shape of (dof, 2)
required	data	Queue	All data are stored in a queue
required	plot_title	list	list of plot titles

10.8.3 *update*

Update plot every 500 ms.

10.8.4 *update_plot*

Update plot with new values and call threads for parallelization

10.8.5 *sub_plot*

Update suplots.

Input Parameters:

	Parameters	Type	Description
required	args	tuple	Tuple of current subplot (int), data (deque), the limits (numpy), buffer size (int) and a counter (int) for all elements that already been added to the deque

10.8.6 *run_plot*

Start plot app.

Input Parameters:

	Parameters	Type	Description
required	dof	int	Number of degrees of freedom of the robot
required	limits	numpy	Limits of the robot with the shape of (dof, 2)
required	value_queue	list	Queue for data transfer and communication with pybullet and tkinter app

11 OS Utils

11.0.1 *get_operating_system*

Return the operating system that are currently used for the program.

11.0.2 *up_directory*

Return path and go one directory up.

Input Parameters:

	Parameters	Type	Description
required	path	str	Full path of file/directory

11.0.3 *get_real_path*

Return the full path of a file.

Input Parameters:

	Parameters	Type	Description
required	path	str	Path of a file or directory