mitsubishi_robots

index

```
# This module was made for students of Robotics to use
# so they don't need to care about ROS nodes and stuff
# and also to force them to practice stuff from lectures,
# so something may seem odd.
# It mostly wraps moveit commander, but also adds some
# custom functions, such as trajectory given by joint
# values and calling Kinematics over KIN_6DOF object.
```

Modules

copy	moveit_commander	numpy	sys
geometry_msgs	moveit_msgs	rospy	tf.transformations

Classes

__builtin__.object

Mitsubishi_robot

class **Mitsubishi_robot**(__builtin__.object)

```
Move group python interface object
```

Methods defined here:

__init__(self)

attach_object_to_gripper(self, name)

```
Attaches body to gripper
returns True if succedded, else returns False
```

deattach_object_from_gripper(self, name)

```
Deattaches body from gripper
returns True if succedded, else returns False
```

dkt(self, J)

```
Returns cartesian position of end-point of the robot for given joint values in form J = [J1,J2,J3,J4,J5,J6]
output is in form in form X = numpy.array([[X],[Y],[Z],[Rz1],[Rx],[Rz2]])
```

execute_cart_trajectory(self, waypoints)

```
Executes trajectory given by waypoints in cartesian coordinates
waypoints is list of numpy array of coordinates if form P = numpy.array([[X],[Y],[Z],[Rz1],[Rx],[Rz2]]) (Euler angles rotation)
After execution waits 0.4s to make sure the robot arrived to destination pose.
returns True if succeeded, else returns False
```

execute_joint_trajectory(self, waypoints)

```
Executes trajectory given by waypoints in joint coordinates
waypoints is a list of joint values if form waipoints = [[J1,J2,J3,J4,J5,J6],...]
After execution waits 0.4s to make sure the robot arrived to destination pose.
returns True if succeeded, else returns False
```

get_joint_values(self)

```
Returns complete list of current joint values in form J=[J1,J2,J3,J4,J5,J6]
In case gripper is attached, the form is J=[J1,J2,J3,J4,J5,J6,gripper]
```

get_position(self)

```
Returns current position of end point of the robot in Cartesian coordinates as a slope vector in form P = [X,Y,Z,Rz1,Rx,Rz2]
```

ikt(self, X)

```
Returns list all joint coordinates to reach given position in form X = numpy.array([[X],[Y],[Z],[Rz1],[Rx],[Rz2]]) (Euler angles rotation)
if soluion exists the output has form J = [[J1,J2,J3,J4,J5,J6],...]
if infinity solution exists, the joint value, for which infinity solutions exists is marked with char 'i'
(f.e.: J = [[J1,J2,J3,'i',J5,'i'],...])
if no solution exists returns empty list
this function ignores collisions!
```

inf_ikt(self, X, J)

```
Returns one solution in case of infinity solutions for position X in fom X = numpy.array([[X],[Y],[Z],[Rz1],[Rx],[Rz2]]) (Euler angles rotation)
input J is list of joint values in form J=[J1,J2,J3,J4,J5,J6], where one of joint variables is marked with 'i' for which solution is to be found
in list J can be only one 'i'
if no solution is found or the input is incorrect returns empty list, else returns list of joint values in form J=[J1,J2,J3,J4,J5,J6]
```

remove_object(self, name)

```
removes spawned body of given name
returns True if succedded, else returns False
```

set_gripper(self, position)

```
Sets gripper open, clossed or something between.
as position input can be used strings 'open', 'close' or float
returns True if succeded, else returns False
```

set_joint_values(self, joints)

```
Sets joit values to robot
as joints input must be a list. If list of length n<=7 (7 in case of gripper attached, 6 otherwise)
is provided only n joints are set. If gripper is attached and joints length is 7, gripper is set too,
else only arm joints are set.
returns True if succeeded, else returns False
```

set_max_speed(self, speed)

```
Sets maximal percentage of speed to execute motion with
maximal speed cannot be higher then maximal allowed speed
returns True if succeeded, else returns False
```

spawn_box(self, name, size, position)

```
Spawns box of given size and name to given position
size is a list of three ints or three floats
position is in form P = numpy.array([[X],[Y],[Z],[Rz1],[Rx],[Rz2]]) (Euler angles rotation)
```

spawn_sphere(self, name, radius, position)

```
Spawns sphere of given name and radius to given position
radius is float or int
position is in form P = numpy.array([[X],[Y],[Z],[Rz1],[Rx],[Rz2]]) (Euler angles rotation)
```

Data

pi = 3.141592653589793