

---

# **robotics\_sem**

***Release 3.1.2025***

**Ondrej Vita, Josef Kahoun, Simon Pilc**

**Jan 03, 2025**



**CONTENTS:**

<b>1</b>	<b>robotics_sem</b>	<b>3</b>
1.1	lib package . . . . .	3
1.2	src package . . . . .	31
1.3	tests package . . . . .	34
	<b>Python Module Index</b>	<b>37</b>



Documentation for semestral work from Robotics course on FEE CTU.



## ROBOTICS\_SEM

## 1.1 lib package

### 1.1.1 Subpackages

**lib.robotics\_toolbox package**

**Subpackages**

**lib.robotics\_toolbox.core package**

**Submodules**

**lib.robotics\_toolbox.core.se2 module**

Module for representing 2D transformation.

```
class lib.robotics_toolbox.core.se2.SE2(translation: _SupportsArray[dtype[Any]] |
                                         _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int |
                                         float | complex | str | bytes | _NestedSequence[bool | int | float |
                                         complex | str | bytes] | None = None, rotation: SO2 | float | None
                                         = None)
```

Bases: object

Transformation in 2D that is composed of rotation and translation.

```
act(vector: _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float |
      complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes]) → ndarray
```

Transform given 2D vector by this SE2 transformation.

```
homogeneous() → ndarray
```

Return homogeneous transformation matrix.

```
inverse() → SE2
```

Compute inverse of the transformation. Do not use np.linalg.inv.

```
set_from(other: SE2)
```

Copy the properties into current instance.

**lib.robotics\_toolbox.core.se3 module**

Module for representing 3D transformation.

```
class lib.robotics_toolbox.core.se3.SE3(translation: _SupportsArray[dtype[Any]] |  
                                         _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int |  
                                         float | complex | str | bytes | _NestedSequence[bool | int | float |  
                                         complex | str | bytes] | None = None, rotation: SO3 | None =  
                                         None)
```

Bases: object

Transformation in 2D that is composed of rotation and translation.

```
act(vector: _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float |  
      complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes]) → ndarray
```

Rotate given 3D vector by this transformation.

```
homogeneous() → ndarray
```

Return homogeneous matrix representation of the transformation.

```
inverse() → SE3
```

Compute inverse of the transformation

```
set_from(other: SE3)
```

Copy the properties into current instance.

### lib.robotics\_toolbox.core.so2 module

Module for representing 2D rotation.

```
class lib.robotics_toolbox.core.so2.SO2(angle: float = 0.0)
```

Bases: object

This class represents an SO2 rotations internally represented by rotation matrix.

```
act(vector: _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float |  
      complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes]) → ndarray
```

Rotate given vector by this transformation.

```
property angle: float
```

Return angle [rad] from the internal rotation matrix representation.

```
inverse() → SO2
```

Return inverse of the transformation. Do not change internal property of the object.

### lib.robotics\_toolbox.core.so3 module

Module for representing 3D rotation.

```
class lib.robotics_toolbox.core.so3.SO3(rotation_matrix: _SupportsArray[dtype[Any]] |  
                                         _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int |  
                                         float | complex | str | bytes | _NestedSequence[bool | int | float |  
                                         complex | str | bytes] | None = None)
```

Bases: object

This class represents an SO3 rotations internally represented by rotation matrix.

```
act(vector: _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float |  
      complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes]) → ndarray
```

Rotate given vector by this transformation.



**static exp**(*rot\_vector*: *\_SupportsArray*[*dtype*[*Any*]] | *\_NestedSequence*[*\_SupportsArray*[*dtype*[*Any*]]] | *bool* | *int* | *float* | *complex* | *str* | *bytes* | *\_NestedSequence*[*bool* | *int* | *float* | *complex* | *str* | *bytes*])  $\rightarrow$  *SO3*

Compute SO3 transformation from a given rotation vector, i.e. exponential representation of the rotation.

**static from\_angle\_axis**(*angle*: *float*, *axis*: *\_SupportsArray*[*dtype*[*Any*]] | *\_NestedSequence*[*\_SupportsArray*[*dtype*[*Any*]]] | *bool* | *int* | *float* | *complex* | *str* | *bytes* | *\_NestedSequence*[*bool* | *int* | *float* | *complex* | *str* | *bytes*])  $\rightarrow$  *SO3*

Compute rotation from angle axis representation.

**static from\_euler\_angles**(*angles*: *\_SupportsArray*[*dtype*[*Any*]] | *\_NestedSequence*[*\_SupportsArray*[*dtype*[*Any*]]] | *bool* | *int* | *float* | *complex* | *str* | *bytes* | *\_NestedSequence*[*bool* | *int* | *float* | *complex* | *str* | *bytes*], *seq*: *list*[*str*])  $\rightarrow$  *SO3*

Compute rotation from euler angles defined by a given sequence. *angles*: is a three-dimensional array of angles *seq*: is a list of axis around which angles rotate, e.g. 'xyz', 'xzx', etc.

**static from\_quaternion**(*q*: *\_SupportsArray*[*dtype*[*Any*]] | *\_NestedSequence*[*\_SupportsArray*[*dtype*[*Any*]]] | *bool* | *int* | *float* | *complex* | *str* | *bytes* | *\_NestedSequence*[*bool* | *int* | *float* | *complex* | *str* | *bytes*])  $\rightarrow$  *SO3*

Compute rotation from quaternion in a form [qx, qy, qz, qw].

**inverse()**  $\rightarrow$  *SO3*

Return inverse of the transformation.

**log()**  $\rightarrow$  ndarray

Compute rotation vector from this SO3

**static rx**(*angle*: *float*)  $\rightarrow$  *SO3*

Return rotation matrix around x axis.

**static ry**(*angle*: *float*)  $\rightarrow$  *SO3*

Return rotation matrix around y axis.

**static rz**(*angle*: *float*)  $\rightarrow$  *SO3*

Return rotation matrix around z axis.

**to\_angle\_axis()**  $\rightarrow$  tuple[*float*, ndarray]

Compute angle axis representation from self.

**to\_quaternion()**  $\rightarrow$  ndarray

Compute quaternion from self.

## Module contents

Core functionality for the `robotics_toolbox` package is defined by transformations in 2D and 3D space. This module implements these transformations as classes that can be used to represent and manipulate rotations and translations in 2D and 3D space.

## lib.robotics\_toolbox.planning package

### Submodules

#### lib.robotics\_toolbox.planning.prm module

Module for path planning using Probabilistic Roadmap Method (PRM).

```
class lib.robotics_toolbox.planning.prm.GraphPlanner(graph_matrix: list)
    Bases: object
    get_path(i, j) → list
        will get path from node i to j as list of nodes IDs visited
class lib.robotics_toolbox.planning.prm.Node(id: int, config: ArrayLike | SE2 | SE3)
    Bases: object
    add_neighbour(neighbour: Node, path_to_neighbour: list[ArrayLike | SE2 | SE3])
class lib.robotics_toolbox.planning.prm.PRM(robot: RobotBase, delta_q=0.2)
    Bases: object
    closest_connect(q: ArrayLike | SE2 | SE3, q_to_graph: bool = True)
        connect the closest node to the given configuration
        return: path from the given config to the closest reachable node and closest reachable node id
    connect(q_init: ArrayLike | SE2 | SE3, q_goal: ArrayLike | SE2 | SE3, max_iter: int = 1000) →
        list[ArrayLike | SE2 | SE3] | None
        will find path between two given configurations
    explore(max_nodes: int = 500) → None
        PRM algorithm for motion planning.
    plan(q_start: ArrayLike | SE2 | SE3, q_goal: ArrayLike | SE2 | SE3, graph_planner: GraphPlanner = <class
        'lib.robotics_toolbox.planning.prm.GraphPlanner'>) → list[ArrayLike | SE2 | SE3]
        will plan path in the current graph
```

### lib.robotics\_toolbox.planning.rrt module

Module for RRT motion planning algorithm.

```
class lib.robotics_toolbox.planning.rrt.RRT(robot: RobotBase, delta_q=0.2, p_sample_goal=0.5)
    Bases: object
    plan(q_start: _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int |
        float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes] | SE2 | SE3,
        q_goal: _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int |
        float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes] | SE2 | SE3,
        max_iterations: int = 10000) → list[_SupportsArray[dtype[Any]] |
        _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | str | bytes |
        _NestedSequence[bool | int | float | complex | str | bytes] | SE2 | SE3]
        RRT algorithm for motion planning.
    random_shortcut(path: list[ndarray | SE2 | SE3], max_iterations=100) → list[ndarray | SE2 | SE3]
        Random shortcut algorithm that pick two points on the path randomly and tries to interpolate between them.
        If collision free interpolation exists, the path between selected points is replaced by the interpolation.
class lib.robotics_toolbox.planning.rrt.TreeList
    Bases: object
    add_node(node: TreeNode) → None
```

```
calculate_distance(q1: _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] |
    bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex |
    str | bytes] | SE2 | SE3, q2: _SupportsArray[dtype[Any]] |
    _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | str |
    bytes | _NestedSequence[bool | int | float | complex | str | bytes] | SE2 | SE3) → float
```

```
return_nearest_node(q: _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] |
    bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex |
    str | bytes] | SE2 | SE3) → TreeNode
```

```
class lib.robotics_toolbox.planning.rrt.TreeNode(q: _SupportsArray[dtype[Any]] |
    _NestedSequence[_SupportsArray[dtype[Any]]] |
    bool | int | float | complex | str | bytes |
    _NestedSequence[bool | int | float | complex | str |
    bytes] | SE2 | SE3, parent: TreeNode | None =
    None)
```

Bases: object

**return\_parent**()

**return\_path\_from\_start**()

## Module contents

Module for path planning algorithms.

## lib.robotics\_toolbox.render package

### Submodules

#### lib.robotics\_toolbox.render.mobile\_robot\_renderer module

```
class lib.robotics_toolbox.render.mobile_robot_renderer.MobileRobotRenderer(ax: Axes, robot:
    MobileRobot)
```

Bases: object

**update**()

#### lib.robotics\_toolbox.render.planar\_manipulator\_renderer module

```
class lib.robotics_toolbox.render.planar_manipulator_renderer.PlanarManipulatorRenderer(ax:
    Axes,
    robot:
    Pla-
    n-
    ar-
    Ma-
    nip-
    u-
    la-
    tor,
    **kwargs)
```

Bases: object

**plot\_init**(color='k', lw=3, ms=7)

Plot the robot. It returns the plt\_objects, that you can use to update the robot pose in animation.

**plot\_line\_between\_points**(a: *SupportsArray*[dtype[Any]] | *NestedSequence*[*SupportsArray*[dtype[Any]]] | bool | int | float | complex | str | bytes | *NestedSequence*[bool | int | float | complex | str | bytes], b: *SupportsArray*[dtype[Any]] | *NestedSequence*[*SupportsArray*[dtype[Any]]] | bool | int | float | complex | str | bytes | *NestedSequence*[bool | int | float | complex | str | bytes], \*args, \*\*kwargs)

Plot line between two given 2D points a and b. Other arguments passed to ax.plot function.

**update**()

### lib.robotics\_toolbox.render.renderplanar module

**class** lib.robotics\_toolbox.render.renderplanar.**Renderplanar**(xlim: tuple[float, float] = (-1, 1), ylim: tuple[float, float] = (-1, 1), lim\_scale: float = 1.0)

Bases: object

**plot\_line\_between\_points**(a: *SupportsArray*[dtype[Any]] | *NestedSequence*[*SupportsArray*[dtype[Any]]] | bool | int | float | complex | str | bytes | *NestedSequence*[bool | int | float | complex | str | bytes], b: *SupportsArray*[dtype[Any]] | *NestedSequence*[*SupportsArray*[dtype[Any]]] | bool | int | float | complex | str | bytes | *NestedSequence*[bool | int | float | complex | str | bytes], \*args, \*\*kwargs)

Plot line between two given 2D points a and b. Other arguments passed to ax.plot function.

**plot\_manipulator**(robot: *PlanarManipulator*, \*\*kwargs)

**plot\_mobile\_robot**(robot: *MobileRobot*)

Plot mobile robot into the figure. If this function is called multiple times for the same robot, this function redraw the robot to the new pose instead of drawing a new one.

**plot\_se2**(t: *SE2*, length=0.1, \*args, \*\*kwargs)

Plot SE2 frame.

**plot\_so2**(t: *SO2*, length=0.1, \*args, \*\*kwargs)

Plot SO2 frame in the origin.

**redraw\_all**()

Redraw all the manipulators that has been plotted before.

**static wait\_for\_close**()

**static wait\_for\_enter**(msg: str | None = None)

### lib.robotics\_toolbox.render.renderspatial module

**class** lib.robotics\_toolbox.render.renderspatial.**RenderSpatial**

Bases: Scene

**plot\_drone**(robot: *Drone*, render=True)

**plot\_manipulator**(robot: [SpatialManipulator](#), render=True)

**plot\_se3**(t: [SE3](#), scale=1.0, render=True)

**wait\_at\_the\_end**()

A method that just sleep for a few seconds. Call it at the end to prevent interruption of the connection with the renderer.

**static wait\_for\_enter**(msg: str | None = None)

## lib.robotics\_toolbox.render.se2\_renderer module

**class** lib.robotics\_toolbox.render.se2\_renderer.**SE2Renderer**(ax: Axes, t: [SE2](#) | [SO2](#), length: float = 0.1, \*args, \*\*kwargs)

Bases: object

**plot\_line\_between\_points**(a: [\\_SupportsArray](#)[dtype[Any]] | [\\_NestedSequence](#)[[\\_SupportsArray](#)[dtype[Any]]] | bool | int | float | complex | str | bytes | [\\_NestedSequence](#)[bool | int | float | complex | str | bytes], b: [\\_SupportsArray](#)[dtype[Any]] | [\\_NestedSequence](#)[[\\_SupportsArray](#)[dtype[Any]]] | bool | int | float | complex | str | bytes | [\\_NestedSequence](#)[bool | int | float | complex | str | bytes], \*args, \*\*kwargs)

Plot line between two given 2D points a and b. Other arguments passed to ax.plot function.

**update**()

## Module contents

Module for rendering robot models in 2D (matplotlib) and 3D (robomeshcat).

**class** lib.robotics\_toolbox.render.**RendererPlanar**(xlim: tuple[float, float] = (-1, 1), ylim: tuple[float, float] = (-1, 1), lim\_scale: float = 1.0)

Bases: object

**plot\_line\_between\_points**(a: [\\_SupportsArray](#)[dtype[Any]] | [\\_NestedSequence](#)[[\\_SupportsArray](#)[dtype[Any]]] | bool | int | float | complex | str | bytes | [\\_NestedSequence](#)[bool | int | float | complex | str | bytes], b: [\\_SupportsArray](#)[dtype[Any]] | [\\_NestedSequence](#)[[\\_SupportsArray](#)[dtype[Any]]] | bool | int | float | complex | str | bytes | [\\_NestedSequence](#)[bool | int | float | complex | str | bytes], \*args, \*\*kwargs)

Plot line between two given 2D points a and b. Other arguments passed to ax.plot function.

**plot\_manipulator**(robot: [PlanarManipulator](#), \*\*kwargs)

**plot\_mobile\_robot**(robot: [MobileRobot](#))

Plot mobile robot into the figure. If this function is called multiple times for the same robot, this function redraw the robot to the new pose instead of drawing a new one.

**plot\_se2**(t: [SE2](#), length=0.1, \*args, \*\*kwargs)

Plot SE2 frame.

**plot\_so2**(t: [SO2](#), length=0.1, \*args, \*\*kwargs)

Plot SO2 frame in the origin.

**redraw\_all()**

Redraw all the manipulators that has been plotted before.

**static wait\_for\_close()**

**static wait\_for\_enter**(*msg: str | None = None*)

**class lib.robotics\_toolbox.render.RendererSpatial**

Bases: Scene

**plot\_drone**(*robot: Drone, render=True*)

**plot\_manipulator**(*robot: SpatialManipulator, render=True*)

**plot\_se3**(*t: SE3, scale=1.0, render=True*)

**wait\_at\_the\_end()**

A method that just sleep for a few seconds. Call it at the end to prevent interruption of the connection with the renderer.

**static wait\_for\_enter**(*msg: str | None = None*)

## lib.robotics\_toolbox.robots package

### Submodules

#### lib.robotics\_toolbox.robots.drone module

**class lib.robotics\_toolbox.robots.drone.Drone**

Bases: RobotBase

**configuration()** → ndarray | SE2 | SE3

Get the configuration of the robot, can be array, SE2, or SE3.

**in\_collision()** → bool

Check if robot is in collision.

**sample\_configuration()** → ndarray | SE2 | SE3

Sample robot configuration inside the configuration space.

**set\_configuration**(*configuration: ndarray | SE2 | SE3*)

Set internal configuration to @param configuration. Returns self.

#### lib.robotics\_toolbox.robots.mobile\_robot module

**class lib.robotics\_toolbox.robots.mobile\_robot.MobileRobot**(*size: float = 0.3*)

Bases: RobotBase

**configuration()** → ndarray | SE2 | SE3

Get the configuration of the robot, can be array, SE2, or SE3.

**in\_collision()** → bool

Check if robot is in collision.

**sample\_configuration()** → ndarray | SE2 | SE3

Sample robot configuration inside the configuration space.

**set\_configuration**(*configuration: ndarray | SE2 | SE3*)

Set internal configuration to @param configuration. Returns self.

**lib.robotics\_toolbox.robots.planar\_manipulator module**

Module for representing planar manipulator.

```
class lib.robotics_toolbox.robots.planar_manipulator.PlanarManipulator(link_parameters: _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes] | None = None, structure: list[str] | str | None = None, base_pose: SE2 | None = None, gripper_length: float = 0.2)
```

Bases: RobotBase

**configuration()** → ndarray | [SE2](#) | [SE3](#)

Get the robot configuration.

**property dof**

Return number of degrees of freedom.

**fk\_all\_links()** → list[[SE2](#)]

Compute FK for frames that are attached to the links of the robot. The first frame is base\_frame, the next frames are described in the constructor.

**flange\_pose()** → [SE2](#)

Return the pose of the flange in the reference frame.

**ik\_analytical**(flange\_pose\_desired: [SE2](#)) → list[ndarray]

Compute IK analytically, return all solutions for joint limits being from -pi to pi for revolute joints -inf to inf for prismatic joints.

**ik\_numerical**(flange\_pose\_desired: [SE2](#), max\_iterations=1000, acceptable\_err=0.0001) → bool

Compute IK numerically. Value self.q is used as an initial guess and updated to solution of IK. Returns True if converged, False otherwise.

**in\_collision()** → bool

Check if robot in its current pose is in collision.

**jacobian()** → ndarray

Computes jacobian of the manipulator for the given structure and configuration.

**jacobian\_finite\_difference**(delta=1e-05) → ndarray

**sample\_configuration()**

Sample robot configuration inside the configuration space. Will change internal state.

**set\_configuration**(configuration: ndarray | [SE2](#) | [SE3](#))

Set configuration of the robot, return self for chaining.

**lib.robotics\_toolbox.robots.planar\_manipulator\_dynamics module**

**class** lib.robotics\_toolbox.robots.planar\_manipulator\_dynamics.**PlanarManipulatorDynamics**(masses=None, \*args, \*\*kwargs)

Bases: PlanarManipulator

**constrained\_forward\_dynamics**(q: *\_SupportsArray[dtype[Any]]* | *\_NestedSequence[\_SupportsArray[dtype[Any]]]* | bool | int | float | complex | str | bytes | *\_NestedSequence[bool | int | float | complex | str | bytes]*, dq: *\_SupportsArray[dtype[Any]]* | *\_NestedSequence[\_SupportsArray[dtype[Any]]]* | bool | int | float | complex | str | bytes | *\_NestedSequence[bool | int | float | complex | str | bytes]*, tau: *\_SupportsArray[dtype[Any]]* | *\_NestedSequence[\_SupportsArray[dtype[Any]]]* | bool | int | float | complex | str | bytes | *\_NestedSequence[bool | int | float | complex | str | bytes]*, damping: float = 0.0) → ndarray

Implement constrained forward dynamics of the robot. I.e. compute ddq from q, dq, tau and damping. Use eq. of motion:  $\tau = M(q) \ddot{q} + h(q, \dot{q}) + \text{damping} * \dot{q} + A^T \lambda$ . The constraint is fixed, such that end effector moves along line with angle 45deg w.r.t. x-axis of reference frame.

**forward\_dynamics**(q: *\_SupportsArray[dtype[Any]]* | *\_NestedSequence[\_SupportsArray[dtype[Any]]]* | bool | int | float | complex | str | bytes | *\_NestedSequence[bool | int | float | complex | str | bytes]*, dq: *\_SupportsArray[dtype[Any]]* | *\_NestedSequence[\_SupportsArray[dtype[Any]]]* | bool | int | float | complex | str | bytes | *\_NestedSequence[bool | int | float | complex | str | bytes]*, tau: *\_SupportsArray[dtype[Any]]* | *\_NestedSequence[\_SupportsArray[dtype[Any]]]* | bool | int | float | complex | str | bytes | *\_NestedSequence[bool | int | float | complex | str | bytes]*, damping: float = 0.0) → ndarray

Implement forward dynamics of the robot. I.e. compute ddq from q, dq, tau and damping. Use eq. of motion:  $\tau = M(q) \ddot{q} + h(q, \dot{q}) + \text{damping} * \dot{q}$ .

**h**(q: *\_SupportsArray[dtype[Any]]* | *\_NestedSequence[\_SupportsArray[dtype[Any]]]* | bool | int | float | complex | str | bytes | *\_NestedSequence[bool | int | float | complex | str | bytes]*, dq: *\_SupportsArray[dtype[Any]]* | *\_NestedSequence[\_SupportsArray[dtype[Any]]]* | bool | int | float | complex | str | bytes | *\_NestedSequence[bool | int | float | complex | str | bytes]*) → ndarray

Coriolis and gravity terms at configuration q and velocity dq.

**inverse\_dynamics**(q: *\_SupportsArray[dtype[Any]]* | *\_NestedSequence[\_SupportsArray[dtype[Any]]]* | bool | int | float | complex | str | bytes | *\_NestedSequence[bool | int | float | complex | str | bytes]*, dq: *\_SupportsArray[dtype[Any]]* | *\_NestedSequence[\_SupportsArray[dtype[Any]]]* | bool | int | float | complex | str | bytes | *\_NestedSequence[bool | int | float | complex | str | bytes]*, ddq: *\_SupportsArray[dtype[Any]]* | *\_NestedSequence[\_SupportsArray[dtype[Any]]]* | bool | int | float | complex | str | bytes | *\_NestedSequence[bool | int | float | complex | str | bytes]*, damping: float = 0.0) → ndarray

Implement inverse dynamics of the robot. I.e. compute tau from q, dq, ddq, and damping. Use eq. of motion:  $\tau = M(q) \ddot{q} + h(q, \dot{q}) + \text{damping} * \dot{q}$ .

**mass\_matrix**(q: *\_SupportsArray[dtype[Any]]* | *\_NestedSequence[\_SupportsArray[dtype[Any]]]* | bool | int | float | complex | str | bytes | *\_NestedSequence[bool | int | float | complex | str | bytes]*) → ndarray

Mass matrix of the robot at configuration q.



**lib.robotics\_toolbox.robots.robot\_base module**

```
class lib.robotics_toolbox.robots.robot_base.RobotBase
```

Bases: object

**abstract configuration()** → ndarray | [SE2](#) | [SE3](#)

Get the configuration of the robot, can be array, SE2, or SE3.

**abstract in\_collision()** → bool

Check if robot is in collision.

**abstract sample\_configuration()** → ndarray | [SE2](#) | [SE3](#)

Sample robot configuration inside the configuration space.

**abstract set\_configuration(configuration: ndarray | [SE2](#) | [SE3](#))**

Set internal configuration to @param configuration. Returns self.

**lib.robotics\_toolbox.robots.spatial\_manipulator module**

```
class lib.robotics_toolbox.robots.spatial_manipulator.SpatialManipulator(robot_name: str |
                                                                    None = None,
                                                                    urdf_path: str | Path
                                                                    | None = None,
                                                                    mesh_folder_path:
                                                                    Path | str | list[Path] |
                                                                    list[str] | None =
                                                                    None, srdf_path: str |
                                                                    Path | None = None,
                                                                    base_pose: SE3 |
                                                                    None = None,
                                                                    **kwargs)
```

Bases: RobotBase

**configuration()** → ndarray | [SE2](#) | [SE3](#)

Get the configuration of the robot, can be array, SE2, or SE3.

**property dof: int**

Return number of degrees of freedom for the robot.

**flange\_pose(flange\_link\_name: str | None = None)** → [SE3](#)

Return a flange pose defined by the link name. Flange link name can be empty for Panda robot.

**in\_collision()** → bool

Check if robot is in collision.

**jacobian(flange\_link\_name: str | None = None)** → ndarray

Computes jacobian of the manipulator for the given structure and configuration.

**sample\_configuration()** → ndarray | [SE2](#) | [SE3](#)

Sample robot configuration inside the configuration space.

**set\_configuration(configuration: ndarray | [SE2](#) | [SE3](#))**

Set internal configuration to @param configuration. Returns self.

## Module contents

This module contains the robot classes for mobile robots, manipulators, and drones.

**class** lib.robotics\_toolbox.robots.**Drone**

Bases: RobotBase

**configuration()** → ndarray | [SE2](#) | [SE3](#)

Get the configuration of the robot, can be array, SE2, or SE3.

**in\_collision()** → bool

Check if robot is in collision.

**sample\_configuration()** → ndarray | [SE2](#) | [SE3](#)

Sample robot configuration inside the configuration space.

**set\_configuration(configuration: ndarray | [SE2](#) | [SE3](#))**

Set internal configuration to @param configuration. Returns self.

**class** lib.robotics\_toolbox.robots.**MobileRobot**(size: float = 0.3)

Bases: RobotBase

**configuration()** → ndarray | [SE2](#) | [SE3](#)

Get the configuration of the robot, can be array, SE2, or SE3.

**in\_collision()** → bool

Check if robot is in collision.

**sample\_configuration()** → ndarray | [SE2](#) | [SE3](#)

Sample robot configuration inside the configuration space.

**set\_configuration(configuration: ndarray | [SE2](#) | [SE3](#))**

Set internal configuration to @param configuration. Returns self.

**class** lib.robotics\_toolbox.robots.**PlanarManipulator**(link\_parameters: *SupportsArray[dtype[Any]] |  
\_NestedSequence[\_SupportsArray[dtype[Any]]]  
| bool | int | float | complex | str | bytes |  
\_NestedSequence[bool | int | float | complex | str  
| bytes] | None = None, structure: list[str] | str |  
None = None, base\_pose: [SE2](#) | None = None,  
gripper\_length: float = 0.2)*

Bases: RobotBase

**configuration()** → ndarray | [SE2](#) | [SE3](#)

Get the robot configuration.

**property dof**

Return number of degrees of freedom.

**fk\_all\_links()** → list[[SE2](#)]

Compute FK for frames that are attached to the links of the robot. The first frame is base\_frame, the next frames are described in the constructor.

**flange\_pose()** → [SE2](#)

Return the pose of the flange in the reference frame.

**ik\_analytical**(*flange\_pose\_desired*: [SE2](#)) → list[ndarray]

Compute IK analytically, return all solutions for joint limits being from  $-\pi$  to  $\pi$  for revolute joints  $-\infty$  to  $\infty$  for prismatic joints.

**ik\_numerical**(*flange\_pose\_desired*: [SE2](#), *max\_iterations*=1000, *acceptable\_err*=0.0001) → bool

Compute IK numerically. Value `self.q` is used as an initial guess and updated to solution of IK. Returns True if converged, False otherwise.

**in\_collision**() → bool

Check if robot in its current pose is in collision.

**jacobian**() → ndarray

Computes jacobian of the manipulator for the given structure and configuration.

**jacobian\_finite\_difference**(*delta*=1e-05) → ndarray

**sample\_configuration**()

Sample robot configuration inside the configuration space. Will change internal state.

**set\_configuration**(*configuration*: ndarray | [SE2](#) | [SE3](#))

Set configuration of the robot, return self for chaining.

**class** lib.robotics\_toolbox.robots.**SpatialManipulator**(*robot\_name*: str | None = None, *urdf\_path*: str | Path | None = None, *mesh\_folder\_path*: Path | str | list[Path] | list[str] | None = None, *srdf\_path*: str | Path | None = None, *base\_pose*: [SE3](#) | None = None, *\*\*kwargs*)

Bases: RobotBase

**configuration**() → ndarray | [SE2](#) | [SE3](#)

Get the configuration of the robot, can be array, SE2, or SE3.

**property dof**: int

Return number of degrees of freedom for the robot.

**flange\_pose**(*flange\_link\_name*: str | None = None) → [SE3](#)

Return a flange pose defined by the link name. Flange link name can be empty for Panda robot.

**in\_collision**() → bool

Check if robot is in collision.

**jacobian**(*flange\_link\_name*: str | None = None) → ndarray

Computes jacobian of the manipulator for the given structure and configuration.

**sample\_configuration**() → ndarray | [SE2](#) | [SE3](#)

Sample robot configuration inside the configuration space.

**set\_configuration**(*configuration*: ndarray | [SE2](#) | [SE3](#))

Set internal configuration to @param configuration. Returns self.

## lib.robotics\_toolbox.utils package

### Submodules

#### lib.robotics\_toolbox.utils.animation\_utils module

```
lib.robotics_toolbox.utils.animation_utils.create_gif_from_mp4(input_vid: Path | str, output: str |  
                                                             Path | None = None)
```

Convert input\_vid (mp4) to GIF by first generating the color pallet.

```
lib.robotics_toolbox.utils.animation_utils.create_mp4_from_folder(folder: Path | str =  
                                                                '/tmp/animation', output: str |  
                                                                Path | None = None, fps: int =  
                                                                10)
```

From the folder that contains images names img\_X.png, create mp4 animation.

```
lib.robotics_toolbox.utils.animation_utils.save_fig(output_folder: Path | str = '/tmp/animation',  
                                                    renderer=None)
```

Save fig of the renderer into the given output folder. Output folder is cleaned on the first run of this command.  
If renderer not provided use plt.savefig. This name of figures is img\_{id}.png.

### lib.robotics\_toolbox.utils.configuration\_utils module

```
lib.robotics_toolbox.utils.configuration_utils.distance_between_configurations(a:  
                                     _SupportsAr-  
                                     ray[dtype[Any]]  
                                     | _NestedSe-  
                                     quence[_SupportsArray[dtype[Any]]]  
                                     | bool | int |  
                                     float |  
                                     complex | str  
                                     | bytes |  
                                     _NestedSe-  
                                     quence[bool |  
                                     int | float |  
                                     complex | str  
                                     | bytes] | SE2  
                                     | SE3, b:  
                                     _SupportsAr-  
                                     ray[dtype[Any]]  
                                     | _NestedSe-  
                                     quence[_SupportsArray[dtype[Any]]]  
                                     | bool | int |  
                                     float |  
                                     complex | str  
                                     | bytes |  
                                     _NestedSe-  
                                     quence[bool |  
                                     int | float |  
                                     complex | str  
                                     | bytes] | SE2  
                                     | SE3) →  
                                     float
```

Compute distance between two configurations, expressed either in task-space SE2/SE3 or joint space np.ndarray

```
lib.robotics_toolbox.utils.configuration_utils.interpolate(a: _SupportsArray[dtype[Any]] |
    _NestedSequence[_SupportsArray[dtype[Any]]] |
    bool | int | float | complex | str | bytes |
    _NestedSequence[bool | int | float |
    complex | str | bytes] | SE2 | SE3, b:
    _SupportsArray[dtype[Any]] |
    _NestedSequence[_SupportsArray[dtype[Any]]] |
    bool | int | float | complex | str | bytes |
    _NestedSequence[bool | int | float |
    complex | str | bytes] | SE2 | SE3, d:
    float) → ndarray | SE2 | SE3
```

Interpolate between two configurations, s.t.  $\text{dist}(a,b) = d$

### lib.robotics\_toolbox.utils.geometry\_utils module

```
lib.robotics_toolbox.utils.geometry_utils.circle_circle_intersection(c0: _SupportsArray[dtype[Any]] |
    _NestedSequence[_SupportsArray[dtype[Any]]]
    | bool | int | float | complex
    | str | bytes |
    _NestedSequence[bool |
    int | float | complex | str |
    bytes], r0: float, c1: _SupportsArray[dtype[Any]] |
    _NestedSequence[_SupportsArray[dtype[Any]]]
    | bool | int | float | complex
    | str | bytes |
    _NestedSequence[bool |
    int | float | complex | str |
    bytes], r1: float) →
    list[ndarray]
```

Computes intersection of the circles defined by center  $c_i$  and radius  $r_i$ . Returns empty array if there is no solution, two solutions otherwise. If there are infinite number of solutions, select two (almost) randomly.

```
lib.robotics_toolbox.utils.geometry_utils.circle_line_intersection(c:
    _SupportsArray[dtype[Any]]
    | _NestedSequence[_SupportsArray[dtype[Any]]]
    | bool | int | float | complex |
    str | bytes |
    _NestedSequence[bool | int |
    float | complex | str | bytes],
    r: float, a:
    _SupportsArray[dtype[Any]]
    | _NestedSequence[_SupportsArray[dtype[Any]]]
    | bool | int | float | complex |
    str | bytes |
    _NestedSequence[bool | int |
    float | complex | str | bytes],
    b:
    _SupportsArray[dtype[Any]]
    | _NestedSequence[_SupportsArray[dtype[Any]]]
    | bool | int | float | complex |
    str | bytes |
    _NestedSequence[bool | int |
    float | complex | str | bytes])
    → list[ndarray]
```

Compute intersection of circle (c, r) with line defined by two points (a, b)

```
lib.robotics_toolbox.utils.geometry_utils.nullspace(A, atol=1e-13, rtol=0.0)
```

Compute kernel, i.e. nullspace of the given matrix A.

## Module contents

Module with utility functions for the robotics\_toolbox package.

```
lib.robotics_toolbox.utils.circle_circle_intersection(c0: _SupportsArray[dtype[Any]] | _NestedSe-
    quence[_SupportsArray[dtype[Any]]] | bool |
    int | float | complex | str | bytes |
    _NestedSequence[bool | int | float | complex |
    str | bytes], r0: float, c1:
    _SupportsArray[dtype[Any]] | _NestedSe-
    quence[_SupportsArray[dtype[Any]]] | bool |
    int | float | complex | str | bytes |
    _NestedSequence[bool | int | float | complex |
    str | bytes], r1: float) → list[ndarray]
```

Computes intersection of the circles defined by center `c_i` and radius `r_i`. Returns empty array if there is no solution, two solutions otherwise. If there are infinite number of solutions, select two (almost) randomly.

```
lib.robotics_toolbox.utils.circle_line_intersection(c: _SupportsArray[dtype[Any]] |
    _NestedSequence[_SupportsArray[dtype[Any]]] |
    bool | int | float | complex | str | bytes |
    _NestedSequence[bool | int | float | complex | str |
    bytes], r: float, a: _SupportsArray[dtype[Any]] |
    _NestedSequence[_SupportsArray[dtype[Any]]] |
    bool | int | float | complex | str | bytes |
    _NestedSequence[bool | int | float | complex | str |
    bytes], b: _SupportsArray[dtype[Any]] |
    _NestedSequence[_SupportsArray[dtype[Any]]] |
    bool | int | float | complex | str | bytes |
    _NestedSequence[bool | int | float | complex | str |
    bytes]) → list[ndarray]
```

Compute intersection of circle (c, r) with line defined by two points (a, b)

```
lib.robotics_toolbox.utils.create_gif_from_mp4(input_vid: Path | str, output: str | Path | None = None)
    Convert input_vid (mp4) to GIF by first generating the color pallet.
```

```
lib.robotics_toolbox.utils.create_mp4_from_folder(folder: Path | str = 'tmp/animation', output: str |
    Path | None = None, fps: int = 10)
```

From the folder that contains images names img\_X.png, create mp4 animation.

```
lib.robotics_toolbox.utils.distance_between_configurations(a: _SupportsArray[dtype[Any]] |
    _NestedSequence[_SupportsArray[dtype[Any]]] |
    bool | int | float | complex | str | bytes |
    _NestedSequence[bool | int | float |
    complex | str | bytes] | SE2 | SE3, b:
    _SupportsArray[dtype[Any]] |
    _NestedSequence[_SupportsArray[dtype[Any]]] |
    bool | int | float | complex | str | bytes |
    _NestedSequence[bool | int | float |
    complex | str | bytes] | SE2 | SE3) →
    float
```

Compute distance between two configurations, expressed either in task-space SE2/SE3 or joint space np.ndarray

```
lib.robotics_toolbox.utils.interpolate(a: _SupportsArray[dtype[Any]] |
    _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float |
    complex | str | bytes | _NestedSequence[bool | int | float |
    complex | str | bytes] | SE2 | SE3, b: _SupportsArray[dtype[Any]] |
    _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int |
    float | complex | str | bytes | _NestedSequence[bool | int | float |
    complex | str | bytes] | SE2 | SE3, d: float) → ndarray | SE2 | SE3
```

Interpolate between two configurations, s.t. dist(a,b) = d

```
lib.robotics_toolbox.utils.nullspace(A, atol=1e-13, rtol=0.0)
    Compute kernel, i.e. nullspace of the given matrix A.
```

```
lib.robotics_toolbox.utils.save_fig(output_folder: Path | str = 'tmp/animation', renderer=None)
    Save fig of the renderer into the given output folder. Output folder is cleaned on the first run of this command.
    If renderer not provided use plt.savefig. This name of figures is img_{id}.png.
```

## Module contents

Robotics Toolbox is a Python library for robot kinematics, dynamics, and control. However, it is not complete, some of the implementations are missing and your goal is to complete them. To verify the correctness of your implementation, you can run the tests in the tests folder.

### 1.1.2 Submodules

#### 1.1.3 lib.base module

Module for storing information about the base of the cubes.

**class** lib.base.Base

Bases: object

Class for storing information about the base of the cubes.

- **aruco\_ids**

List[int] - List of ArUco IDs

- **cube\_centers**

List[List[float]] - List of cube centers (list of [x, y])

- **position**

SE3 - Transformation matrix from base to the camera (T\_CB)

- **target\_transforms**

List[SE3] - List of target transformation matrices from the cube positions to the camera (T\_CT)

**csv2base**(filename: str) → None

Loads base data from a CSV file.

**Parameters**

**filename** (-) – str - path to the CSV file describing the base

**get\_target\_transforms\_from\_camera**() → List[SE3]

Return the list of target transformation matrices from the cube positions to camera (T\_CT).

**Returns**

- List[SE3] - List of target transformation matrices from cube positions to camera

**get\_target\_transforms\_from\_robot**(T\_RC: SE3) → List[SE3]

Return the list of target transformation matrices from cube positions to robot (T\_RT).

**Parameters**

**T\_RC** (-) – SE3 - Transformation matrix from camera to robot (T\_RC)

**Returns**

- List[SE3] - List of target transformation matrices from cube positions to robot

**set\_position**(position: SE3) → None

Sets the position of the base.

**Parameters**

**position** (-) – SE3 - Transformation matrix from base to the camera (T\_CB) (calculated using base\_pose\_estimator)



### 1.1.4 lib.base\_pose\_estimator module

Module for estimating the pose of the base of the cubes in relation to the camera.

**class** lib.base\_pose\_estimator.**BasePoseEstimator**(*camera\_matrix: ndarray | None = None,*  
*distortion\_coefficients: ndarray | None = None*)

Bases: object

This class serves for estimating the pose of the base of the cubes in relation to the camera.

- **camera\_matrix**

np.ndarray - camera matrix

- **distortion\_coefficients**

np.ndarray - distortion coefficients

- **aruco\_size**

float - size of the aruco markers

- **base**

Base - base object for which the pose is estimated

**calculate\_pose**(*camera*) → bool

**calculate\_pose\_multiple\_photos**(*images: list*) → bool

Calculates the pose of the base in relation to the camera using perspective-n-point algorithm.

**Parameters**

**img** (-) – list - list of images from the camera

**Returns**

- bool - True if the pose was calculated, False if the algorithm failed

**calculate\_pose\_normal**(*img: ndarray*) → bool

Calculates the pose of the base in relation to the camera using perspective-n-point algorithm.

**Parameters**

**img** (-) – np.ndarray - image from the camera

**Returns**

- bool - True if the pose was calculated, False if the algorithm failed

**get\_updated\_base**() → *Base*

Returns the base object with the updated pose. Can be called after calculate\_pose.

**Returns**

- Base - base object with the updated pose.

**set\_base**(*base: Base*) → None

Sets the base object.

**Parameters**

**base** (-) – Base - base object

**set\_camera\_matrix**(*camera\_matrix: ndarray*) → None

Sets the camera matrix.

**Parameters**

**camera\_matrix** (-) – np.ndarray - camera matrix

**set\_distortion\_coefficients**(*distortion\_coefficients: ndarray*) → None

Sets the distortion coefficients.

**Parameters**

**distortion\_coefficients** (-) – np.ndarray - distortion coefficients

**set\_rotation**(*base, tvec, rvec*)

## 1.1.5 lib.basler\_camera module

### 1.1.6 lib.camera\_calibration module

**class** lib.camera\_calibration.CameraCalib

Bases: object

class for calibrating the camera using a chessboard or a Charuco board.

**calibrate**(*json\_filename: str*) → None

Calibrate the camera. Save the calibration data to a json file.

**Parameters**

**json\_filename** (*str*) – Name of the json file to save the calibration data.

**calibrate\_using\_charuco**(*json\_filename: str*) → None

Calibrate the camera using a Charuco board. Save the calibration data to a json file.

**Parameters**

**json\_filename** (*str*) – Name of the json file to save the calibration data.

**get\_images**() → list

**set\_image\_path**(*img\_path: str*) → None

lib.camera\_calibration.**blur\_laplacian**(*image, threshold=250*)

Check if an image is blurry using Laplacian variance.

lib.camera\_calibration.**edge\_sharpness**(*image: ndarray, gradient\_threshold=300*) → tuple[bool, float]

Analyze edge sharpness based on gradient size, and disqualify blurry images. :param image: gradient\_threshold (float, threshold for gradient sharpness) :type image: np.ndarray :param gradient\_threshold: \_description\_. Defaults to GRADIENT\_THRESHOLD. :type gradient\_threshold: \_type\_, optional

**Returns**

avg\_gradient (float, average gradient)

**Return type**

tuple[bool, float]

### 1.1.7 lib.display module

**class** lib.display.Display

Bases: object

Class to display the positions of the robot in 3D space.

**add\_positions**(*positions: list*)

Add positions to the list of positions.

**Parameters**

**positions** (*list*) – list of positions

**display\_positions()**

Displace the positions in 3D space.

**setup\_positions**(*positions: list*)

Positions that will be displayed.

**Parameters**

**positions** (*list*) – list of positions

**1.1.8 lib.hand\_eye\_calib module**

Module for hand-eye calibration using images of a calibration target taken from the camera.

**class lib.hand\_eye\_calib.HandEyeCalib**

Bases: object

**load\_fk\_transformation**(*hand\_eye\_calib\_data: str*) → List[[SE3](#)]

Loads the forward kinematics data from a json file.

**Parameters**

**hand\_eye\_calib\_data** (*str*) – Path to the hand-eye calibration data

**Returns**

List of SE3 objects representing the forward kinematics data

**Return type**

List[[SE3](#)]

**load\_fk\_transformation\_and\_points**(*hand\_eye\_calib\_data: str*) → Tuple[List[[SE3](#)], List[array], List[array]]

Loads the forward kinematics data from a json file, and the corners of aruco markers.

**Parameters**

**hand\_eye\_calib\_data** (*str*) – Path to the hand-eye calibration data

**Returns**

List of SE3 objects representing the forward kinematics data, 3D points in target coordinate system and 2D points in camera image

**Return type**

Tuple[List[[SE3](#)], List[np.array], List[np.array]]

**load\_initial\_parameters**(*hand\_eye\_calib\_results: str*) → array

Loads the initial parameters from a json file.

**Parameters**

**hand\_eye\_calib\_results** (*str*) – Path to the initial hand-eye calibration results

**Returns**

Initial parameters [tvec\_RC, rvec\_RC, tvec\_GT, rvec\_GT]

**Return type**

np.array

**prepare\_points**(*calib\_imgs\_path: str*) → Tuple[List[array], List[array], List[[SE3](#)]]

Prepare the 3D and 2D points for the optimization.

**Parameters**

**calib\_imgs\_path** (-) – str - Path to the calibration images

**Returns**

- Tuple[List[np.array], List[np.array]] - 3D points in target coordinate system and 2D points in camera image

**reprojection\_error**(*params: array, K: array, dist\_coef: array, points\_3d: array, points\_2d: array, T\_RG: List[SE3]*) → float

Calculates the reprojection error.

Used as the optimization criterion to calculate the robot to camera (and gripper to target) calibration.

**Args:aruco\_ids:**

- *params*: np.array - Parameters to optimize (rvecs and tvecs of the transformations) [tvec\_RC, rvec\_RC, tvec\_GT, rvec\_GT] (tvecs in meters)
- *K*: np.array - Camera matrix
- *dist\_coef*: np.array - Distortion coefficients
- *points\_3d*: np.array - 3D points of the calibration target (in the target coordinate system)
- *points\_2d*: np.array - 2D points of the calibration target (the image points)
- *T\_RG*: List[SE3] - List of gripper to robot transformation matrices (from forward kinematics) (in meters)

**Returns**

- float - Reprojection error

**reprojection\_error\_ls**(*params: array, K: array, dist\_coef: array, points\_3d: array, points\_2d: array, T\_RG: List[SE3]*) → array

New and revised reprojection error function for least squares optimization.

Used as the optimization criterion to calculate the robot to camera (and gripper to target) calibration.

**Parameters**

- **params** (-) – np.array - Parameters to optimize (rvecs and tvecs of the transformations) [tvec\_RC, rvec\_RC, tvec\_GT, rvec\_GT] (tvecs in meters)
- **K** (-) – np.array - Camera matrix
- **dist\_coef** (-) – np.array - Distortion coefficients
- **points\_3d** (-) – np.array - 3D points of the calibration target (in the target coordinate system)
- **points\_2d** (-) – np.array - 2D points of the calibration target (the image points)
- **T\_RG** (-) – List[SE3] - List of gripper to robot transformation matrices (from forward kinematics) (in meters)

**Returns**

- np.array - Reprojection error residuals

**reprojection\_optimization**(*camera\_calib\_filename: str, hand\_eye\_calib\_data: str, hand\_eye\_calib\_results: str*) → bool

Optimizes the camera to robot and gripper to target calibration using reprojection error.

**Parameters**

- **camera\_calib\_filename** – (str) Camera calibration filename
- **hand\_eye\_calib\_data** – (str) Path to the hand-eye calibration data

- **hand\_eye\_calib\_results** – (str) Path to the initial hand-eye calibration results

**Returns**

True if the optimization was successful, False otherwise

**Return type**

bool

### 1.1.9 lib.robot\_wrapper module

**class** lib.robot\_wrapper.**Robot**(*robotHome: bool = True*)

Bases: CRS97

**get\_desired\_pos\_of\_custom\_gripper**(*pos: SE3*) → *SE3*

This function returns the desired position of the robot in the robot base coordinates, for the custom gripper.

**Parameters**

**pos** (*SE3*) – position of the robot in the camera coordinates.

**Returns**

Position of the custom gripper in the robot base coordinates, or None if the position is incorrect.

**Return type**

*SE3*

**get\_flange\_pos**() → ndarray

This function returns the position of the flange (not the gripper) in the robot base coordinates.

**Returns**

(x, y, z) coordinates of the flange.

**Return type**

np.ndarray

**get\_gripper\_pos**() → ndarray

This function returns the T transformation matrix of the gripper.

**Returns**

T matrix of the flange.

**Return type**

np.ndarray

**get\_name**() → str

returns name of the robot.

**Returns**

name of the robot

**Return type**

str

**gripper\_drop**()

Closes the gripper using the learned strength.

**gripper\_pick**()

Opens the gripper.

**move\_custom\_gripper**(*pos*: SE3)

**This function moves the robot to the given position in robot coordinates.**

moves the custom gripper to a given position.

**Parameters**

**pos** (SE3) – position to which the custom gripper will be moved

**Returns**

True if the robot moved to the position, False otherwise.

**Return type**

bool

**move\_to\_angles**(*q*: ndarray) → bool

moves the robot to the given position based on a given joint angles.

**Parameters**

**q** (np.ndarray) – joint angles of the robot

**Returns**

True if the robot moved to the position, False otherwise.

**Return type**

bool

**move\_to\_camera\_capture\_pos**() → bool

This function moves the robot to a position in which shadows are minimized in the camera view.

**Returns**

True if the robot moved to the gate position, False otherwise.

**Return type**

bool

**move\_to\_coordinates**(*pos*: ndarray) → bool

This function moves the robot to the given position.

**Parameters**

**pos** (np.ndarray) – Transformation matrix of the gripper. T matrix.

**Returns**

True if the robot moved to the position, False otherwise.

**Return type**

bool

**move\_to\_gate**() → bool

This function moves the robot to the gate position, which makes easier access to the gripper for making changes.

**Returns**

True if the robot moved to the gate position, False otherwise.

**Return type**

bool

**move\_under\_camera**() → bool

Moves the robot under a camera.

**Returns**

True if the robot moved under the camera, False otherwise.

**Return type**

bool

**pick\_drop\_part**(base: Base, index\_of\_target: int, pick\_part: bool)

This function picks or drops a part from a given base.

**Parameters**

- **base** (Base) – Base object from which the part will be picked or dropped.
- **index\_of\_target** (int) – Index of the target part.
- **pick\_part** (bool) – True if the part will be picked, False if the part will be dropped.

**pick\_drop\_part\_with\_wiggle**(base: Base, index\_of\_target: int, pick\_part: bool)

This function picks or drops a part from a given base and wiggles to make sure the part falls into the holder.

**Parameters**

- **base** (Base) – Base object from which the part will be picked or dropped.
- **index\_of\_target** (int) – Index of the target part.
- **pick\_part** (bool) – True if the part will be picked, False if the part will be dropped.

**position\_with\_adaptive\_offset**(pos: ndarray) → ndarray

Transforms the position using adaptive offset

**Parameters****pos** (np.ndarray) – Position to be transformed**Returns**

Transformed position

**Return type**

np.ndarray

**return\_best\_ik**(pos: ndarray) → ndarray

This function returns the best ik solution for the robot to reach the given position.

**Parameters****pos** (np.ndarray) – Transformation matrix of the gripper.**Returns**

Best ik solution for the robot.

**Return type**

np.ndarray

**setup**(gripper\_offset\_matrix: SE3 | None = None, cam\_to\_rob\_matrix: SE3 | None = None)

sets up transformation matrix from camera to robot base. also sets up the offset matrix

**Parameters**

- **cam\_to\_rob\_matrix** (SE3) – transformation matrix from camera to robot base.
- **gripper\_offset\_matrix** (SE3 | None, optional) – offset matrix to custom gripper. Defaults to None.

**wiggle\_move\_to\_coords**(pos: ndarray) → bool

This function moves the robot to the given position while wiggling.

**Parameters****pos** (np.ndarray) – Transformation matrix of the gripper. T matrix.

**Returns**

True if the robot moved to the position, False otherwise.

**Return type**

bool

**wiggle\_wiggle\_wiggle()**

This function moves the robot in a wiggle pattern.

Serves as a failsafe when the coordinates of the part are not precise enough. This wiggle position should make the part fall into the holder.

### 1.1.10 lib.se3 module

Module for representing 3D transformation.

```
class lib.se3.SE3(translation: _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] |  
                  bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str |  
                  bytes] | None = None, rotation: SO3 | None = None)
```

Bases: object

Transformation in 2D that is composed of rotation and translation.

```
act(vector: _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float |  
    complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes])  $\rightarrow$  ndarray
```

Rotate given 3D vector by this transformation.

```
classmethod from_dict(data)
```

```
homogeneous()  $\rightarrow$  ndarray
```

Return homogeneous matrix representation of the transformation.

```
inverse()  $\rightarrow$  SE3
```

Compute inverse of the transformation

```
set_from(other: SE3)
```

Copy the properties into current instance.

```
to_dict()
```

### 1.1.11 lib.so2 module

Module for representing 2D rotation.

```
class lib.so2.S02(angle: float = 0.0)
```

Bases: object

This class represents an SO2 rotations internally represented by rotation matrix.

```
act(vector: _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float |  
    complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes])  $\rightarrow$  ndarray
```

Rotate given vector by this transformation.

```
property angle: float
```

Return angle [rad] from the internal rotation matrix representation.

```
inverse()  $\rightarrow$  SO2
```

Return inverse of the transformation. Do not change internal property of the object.



### 1.1.12 lib.so3 module

Module for representing 3D rotation.

```
class lib.so3.SO3(rotation_matrix: _SupportsArray[dtype[Any]] |  

    _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | str | bytes |  

    _NestedSequence[bool | int | float | complex | str | bytes] | None = None)
```

Bases: object

This class represents an SO3 rotations internally represented by rotation matrix.

```
act(vector: _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float |  

    complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes]) → ndarray
```

Rotate given vector by this transformation.

```
static exp(rot_vector: _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] |  

    bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes])  

    → SO3
```

Compute SO3 transformation from a given rotation vector, i.e. exponential representation of the rotation.

```
static from_angle_axis(angle: float, axis: _SupportsArray[dtype[Any]] |  

    _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | str  

    | bytes | _NestedSequence[bool | int | float | complex | str | bytes]) → SO3
```

Compute rotation from angle axis representation.

```
static from_euler_angles(angles: _SupportsArray[dtype[Any]] |  

    _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex |  

    str | bytes | _NestedSequence[bool | int | float | complex | str | bytes], seq:  

    list[str]) → SO3
```

Compute rotation from euler angles defined by a given sequence. angles: is a three-dimensional array of angles seq: is a list of axis around which angles rotate, e.g. 'xyz', 'xzx', etc.

```
static from_quaternion(q: _SupportsArray[dtype[Any]] |  

    _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | str  

    | bytes | _NestedSequence[bool | int | float | complex | str | bytes]) → SO3
```

Compute rotation from quaternion in a form [qx, qy, qz, qw].

```
inverse() → SO3
```

Return inverse of the transformation.

```
log() → ndarray
```

Compute rotation vector from this SO3

```
static rx(angle: float) → SO3
```

Return rotation matrix around x axis.

```
static ry(angle: float) → SO3
```

Return rotation matrix around y axis.

```
static rz(angle: float) → SO3
```

Return rotation matrix around z axis.

```
to_angle_axis() → tuple[float, ndarray]
```

Compute angle axis representation from self.

```
to_quaternion() → ndarray
```

Compute quaternion from self.

### 1.1.13 lib.utils module

`lib.utils.get_R(roll, pitch, yaw) → ndarray`

Returns the rotation matrix from roll, pitch, yaw angles

**Parameters**

- **roll** – float - roll angle
- **pitch** – float - pitch angle
- **yaw** – float - yaw angle

**Returns**

rotation matrix

**Return type**

np.ndarray

`lib.utils.get_images_from_dir(img_path: str) → list`

Loads all images from a directory.

**Parameters**

**img\_path** (*str*) – The path to the directory containing the images.

**Returns**

A list of paths to the images.

**Return type**

list

`lib.utils.load_camera_calibration_data(filename: str) → Tuple[ndarray, ndarray, ndarray]`

Loads camera calibration data from a file

**Parameters**

**filename** – str - filename of the calibration data

**Returns**

camera matrix, distortion coefficients, new camera matrix

**Return type**

tuple[np.ndarray, np.ndarray, np.ndarray]

`lib.utils.read_data_from_json(file_path)`

Reads multiple lists or dictionaries from a JSON file.

**Parameters**

**file\_path** (*str*) – The path to the JSON file.

**Returns**

A dictionary containing the data.

**Return type**

dict

`lib.utils.robot_gripper_to_SE3(robot) → SE3`

Gets transformation from gripper to robot using robot joint angles

**Parameters**

**robot** – Robot - robot object

**Returns**

SE3 object representing transformation from gripper to robot

**Return type***SE3*`lib.utils.vector_to_se3(tvec, rvec) → SE3`

Converts translation and rotation (Euler) vectors to SE3 transformation class

**Parameters**

- **tvec** – ArrayLike - translation vector
- **rvec** – ArrayLike - rotation vector

**Returns**

SE3 object

**Return type***SE3*`lib.utils.write_data_to_json(file_path, **data)`

Writes multiple lists or dictionaries to a JSON file.

**Parameters**

- **file\_path** (*str*) – The path to the JSON file.
- **\*\*data** – Arbitrary keyword arguments representing the data to be saved.

## 1.1.14 Module contents

Library containing all the modules

## 1.2 src package

### 1.2.1 Subpackages

#### src.calibration package

##### Submodules

#### src.calibration.camera\_calib\_new module

Gets the camera image and detects the Aruco markers on it.

`src.calibration.camera_calib_new.main()`

#### src.calibration.camera\_charuco\_calibration\_routine module

`src.calibration.camera_charuco_calibration_routine.camera_take_image(camera: BaslerCamera,  
dirname: str)`

Takes images from the camera and saves them in the directory given, until the user presses ‘x’ to exit. Numbers the images sequentially, never overwriting an existing image. Only accepts detected chessboard images.

**Parameters**

- **camera** (*BaslerCamera*) – camera object
- **dirname** (*str*) – directory to save the images

`src.calibration.camera_charuco_calibration_routine.initialise_camera()` → BaslerCamera  
initialises the camera and returns the camera object

**Returns**

camera object

**Return type**

BaslerCamera

`src.calibration.camera_charuco_calibration_routine.main()`

**src.calibration.hand\_eye\_calib\_from\_data module**

`src.calibration.hand_eye_calib_from_data.load_json_data(filename: str)` → List[List[SE3]]

Load the json data from the file

**Parameters**

**filename** (*str*) – filename to load the data from

**Returns**

list of robot positions and eye positions

**Return type**

List[List[SE3]]

`src.calibration.hand_eye_calib_from_data.main()`

`src.calibration.hand_eye_calib_from_data.solve_AX_YB(a, b)`

Solve  $A^i X = Y B^i$ , return X, Y :param a: list of SE3 objects :param b: list of SE3 objects

**Returns**

SE3 objects

**Return type**

X, Y

**src.calibration.hand\_eye\_calib\_with\_moves module**

`src.calibration.hand_eye_calib_with_moves.find_aruco_id_vec(corners, ids, ARUCO_SIZE: int)` → SE3

Find the vectors of the wanted aruco id in the image.

**Parameters**

- **corners** (*list*) – list of aruco corners
- **ids** (*list*) – list of aruco ids
- **ARUCO\_SIZE** (*int*) – size of the aruco marker

**Returns**

SE3 object of the aruco marker

**Return type**

SE3

`src.calibration.hand_eye_calib_with_moves.generate_robot_positions(robot: Robot)`

Generate a list of robot positions (in)

**Parameters**

**robot** (*Robot*) – robot object

**Returns**

list of robot positions

**Return type**

list

`src.calibration.hand_eye_calib_with_moves.generate_robot_positions_new(robot: Robot) → List`

Generate a list of robot positions

**Parameters**

**robot** (*Robot*) – robot object

**Returns**

list of robot positions

**Return type**

List

`src.calibration.hand_eye_calib_with_moves.hand_eye_take_image(dirname: str) → List`

Cycles through robot positions and takes images at each position. Format of robot positions is a list of robot SE3s.

**Parameters**

**dirname** (*str*) – directory to save the images

**Returns**

list of SE3s of the robot

**Return type**

list

`src.calibration.hand_eye_calib_with_moves.initialise_camera() → BaslerCamera`

initialises the camera and returns the camera object

**Returns**

camera object

**Return type**

BaslerCamera

`src.calibration.hand_eye_calib_with_moves.main()`

`src.calibration.hand_eye_calib_with_moves.solve_AX_YB(a, b)`

Solve  $A^i X = Y B^i$ , return X, Y

**Parameters**

- **a** – list of SE3 objects
- **b** – list of SE3 objects

**Returns**

SE3 objects

**Return type**

X, Y

**Module contents**

calibration scripts for camera and hand-eye calibration.

## **src.move\_commands package**

### **Submodules**

**src.move\_commands.go\_home** module

**src.move\_commands.move** module

**src.move\_commands.move\_outside\_cam** module

**src.move\_commands.move\_to\_gate** module

**src.move\_commands.release** module

### **Module contents**

Programs for easier manipulation of the robot.

## **1.2.2 Module contents**

source file for the project.

## **1.3 tests package**

### **1.3.1 Submodules**

#### **1.3.2 tests.aruco\_test module**

Gets the camera image and detects the Aruco markers on it.

`tests.aruco_test.main()`

#### **1.3.3 tests.camera\_test module**

Program to display one photo of the camera

`tests.camera_test.main()`

#### **1.3.4 tests.conseq\_moves module**

First attempt at creating consequences of moves.

`tests.conseq_moves.main()`

#### **1.3.5 tests.conseq\_moves\_2 module**

First attempt at creating consequences of moves.

`tests.conseq_moves_2.main()`

#### **1.3.6 tests.conseq\_moves\_3 module**

First attempt at creating consequences of moves.

`tests.conseq_moves_3.main()`

### 1.3.7 tests.conseq\_moves\_4 module

First attempt at creating consequences of moves.

```
tests.conseq_moves_4.main()
```

### 1.3.8 tests.conseq\_moves\_5 module

First attempt at creating consequences of moves.

```
tests.conseq_moves_5.main()
```

### 1.3.9 tests.reproj\_error module

Test to check the reprojection error of the camera model.

```
tests.reproj_error.main()
```

### 1.3.10 tests.robot\_wrapper\_test module

```
tests.robot_wrapper_test.generate_robot_positions(robot: Robot)
```

Generate a list of robot positions (in)

```
tests.robot_wrapper_test.main()
```

### 1.3.11 tests.test\_robot2target module

Gets the camera image and detects the Aruco markers on it.

```
tests.test_robot2target.main()
```

### 1.3.12 tests.test\_robot\_camera\_calib module

```
tests.test_robot_camera_calib.find_aruco_id_vec(corners, ids, ARUCO_SIZE: int, wanted_id: int) →  
SE3
```

Find the vectors of the wanted aruco id in the image.

```
tests.test_robot_camera_calib.main()
```

### 1.3.13 tests.test\_show\_positions module

### 1.3.14 Module contents

Test files for the camera calibration module, hand-eye calibration module and robot movement modules.





## PYTHON MODULE INDEX

|  
lib, 31  
lib.base, 20  
lib.base\_pose\_estimator, 21  
lib.basler\_camera, 22  
lib.camera\_calibration, 22  
lib.display, 22  
lib.hand\_eye\_calib, 23  
lib.robot\_wrapper, 25  
lib.robotics\_toolbox, 20  
lib.robotics\_toolbox.core, 5  
lib.robotics\_toolbox.core.se2, 3  
lib.robotics\_toolbox.core.se3, 3  
lib.robotics\_toolbox.core.so2, 4  
lib.robotics\_toolbox.core.so3, 4  
lib.robotics\_toolbox.planning, 7  
lib.robotics\_toolbox.planning.prm, 5  
lib.robotics\_toolbox.planning.rrt, 6  
lib.robotics\_toolbox.render, 9  
lib.robotics\_toolbox.render.mobile\_robot\_renderer,  
7  
lib.robotics\_toolbox.render.planar\_manipulator\_renderer,  
7  
lib.robotics\_toolbox.render.renderer\_planar,  
8  
lib.robotics\_toolbox.render.renderer\_spatial,  
8  
lib.robotics\_toolbox.render.se2\_renderer, 9  
lib.robotics\_toolbox.robots, 14  
lib.robotics\_toolbox.robots.drone, 10  
lib.robotics\_toolbox.robots.mobile\_robot, 10  
lib.robotics\_toolbox.robots.planar\_manipulator,  
11  
lib.robotics\_toolbox.robots.planar\_manipulator\_dynamics,  
12  
lib.robotics\_toolbox.robots.robot\_base, 13  
lib.robotics\_toolbox.robots.spatial\_manipulator,  
13  
lib.robotics\_toolbox.utils, 18  
lib.robotics\_toolbox.utils.animation\_utils,  
15  
lib.robotics\_toolbox.utils.configuration\_utils,  
16  
lib.robotics\_toolbox.utils.geometry\_utils, 17  
lib.se3, 28  
lib.so2, 28  
lib.so3, 29  
lib.utils, 30  
  
S  
src, 34  
src.calibration, 33  
src.calibration.camera\_calib\_new, 31  
src.calibration.camera\_charuco\_calibration\_routine,  
31  
src.calibration.hand\_eye\_calib\_from\_data, 32  
src.calibration.hand\_eye\_calib\_with\_moves, 32  
src.move\_commands, 34  
  
t  
tests, 35  
tests.aruco\_test, 34  
tests.camera\_test, 34  
tests.conseq\_moves, 34  
tests.conseq\_moves\_2, 34  
tests.conseq\_moves\_3, 34  
tests.conseq\_moves\_4, 35  
tests.conseq\_moves\_5, 35  
tests.reproj\_error, 35  
tests.robot\_wrapper\_test, 35  
tests.test\_robot2target, 35  
tests.test\_robot\_camera\_calib, 35  
tests.test\_show\_positions, 35

## INDEX

`\spxentryact()\spxextralib.robotics_toolbox.core.se2.SE2` method, 3  
`\spxentryact()\spxextralib.robotics_toolbox.core.se3.SE3` method, 4  
`\spxentryact()\spxextralib.robotics_toolbox.core.so2.SO2` method, 4  
`\spxentryact()\spxextralib.robotics_toolbox.core.so3.SO3` method, 4  
`\spxentryact()\spxextralib.se3.SE3` method, 28  
`\spxentryact()\spxextralib.so2.SO2` method, 28  
`\spxentryact()\spxextralib.so3.SO3` method, 29  
`\spxentryadd_neighbour()\spxextralib.robotics_toolbox.planning.prm.PRM` method, 6  
`\spxentryadd_node()\spxextralib.robotics_toolbox.planning.rrt.TreeList` method, 6  
`\spxentryadd_node()\spxextralib.robotics_toolbox.planning.rrt.TreeList` method, 14  
`\spxentryadd_positions()\spxextralib.display.Display` method, 22  
`\spxentryangle\spxextralib.robotics_toolbox.core.so2.SO2` property, 4  
`\spxentryangle\spxextralib.so2.SO2` property, 28  
`\spxentryBase\spxextraclass` in lib.base, 20  
`\spxentryBasePoseEstimator\spxextraclass` in lib.base\_pose\_estimator, 21  
`\spxentryblur_laplacian()\spxextrain` module lib.camera\_calibration, 22  
`\spxentrycalculate_distance()\spxextralib.robotics_toolbox.planning.rrt.TreeList` method, 6  
`\spxentrycalculate_pose()\spxextralib.base_pose_estimator.BasePoseEstimator` method, 21  
`\spxentrycalculate_pose_multiple_photos()\spxextralib.base_pose_estimator.BasePoseEstimator` method, 21  
`\spxentrycalculate_pose_normal()\spxextralib.base_pose_estimator.BasePoseEstimator` method, 21  
`\spxentrycalibrate()\spxextralib.camera_calibration.CameraCalib` method, 22  
`\spxentrycalibrate_using_charuco()\spxextralib.camera_calibration.CameraCalib` method, 22  
`\spxentrycamera_take_image()\spxextrain` module src.calibration.camera\_charuco\_calibration\_routine, 31  
`\spxentryCameraCalib\spxextraclass` in lib.camera\_calibration, 22  
`\spxentrycircle_circle_intersection()\spxextrain` module lib.robotics\_toolbox.utils, 18  
`\spxentrycircle_circle_intersection()\spxextrain` module lib.robotics\_toolbox.utils.geometry\_utils, 17  
`\spxentrycircle_line_intersection()\spxextrain` module lib.robotics\_toolbox.utils, 18  
`\spxentrycircle_line_intersection()\spxextrain` module lib.robotics\_toolbox.utils.geometry\_utils, 17  
`\spxentryclosest_connect()\spxextralib.robotics_toolbox.planning.prm.PRM` method, 6  
`\spxentryconfiguration()\spxextralib.robotics_toolbox.robots.Drone` method, 10  
`\spxentryconfiguration()\spxextralib.robotics_toolbox.robots.drone.Drone` method, 10  
`\spxentryconfiguration()\spxextralib.robotics_toolbox.robots.mobile_robot.MobileRobot` method, 10  
`\spxentryconfiguration()\spxextralib.robotics_toolbox.robots.MobileRobot` method, 14  
`\spxentryconfiguration()\spxextralib.robotics_toolbox.robots.planar_manipulator.PlanarManipulator` method, 11  
`\spxentryconfiguration()\spxextralib.robotics_toolbox.robots.PlanarManipulator` method, 14  
`\spxentryconfiguration()\spxextralib.robotics_toolbox.robots.robot_base.RobotBase` method, 13  
`\spxentryconfiguration()\spxextralib.robotics_toolbox.robots.spatial_manipulator.SpatialManipulator` method, 13  
`\spxentryconfiguration()\spxextralib.robotics_toolbox.robots.SpatialManipulator` method, 15  
`\spxentryconnect()\spxextralib.robotics_toolbox.planning.prm.PRM` method, 6  
`\spxentryconstrained_forward_dynamics()\spxextralib.robotics_toolbox.robots.Drone` method, 12  
`\spxentrycreate_gif_from_mp4()\spxextrain` module lib.robotics\_toolbox.utils, 19  
`\spxentrycreate_gif_from_mp4()\spxextrain` module lib.robotics\_toolbox.utils.animation\_utils, 15  
`\spxentrycreate_mp4_from_folder()\spxextrain` module lib.robotics\_toolbox.utils, 19  
`\spxentrycreate_mp4_from_folder()\spxextrain` module lib.robotics\_toolbox.utils.animation\_utils, 16

`\spxentrycsv2base()\spxextralib.base.Base` method, 20  
`\spxentryDisplay\spxextra` class in `lib.display`, 22  
`\spxentrydisplay_positions()\spxextralib.display.Display` method, 22  
`\spxentrydistance_between_configurations()\spxextrain` module `lib.robotics_toolbox.utils`, 19  
`\spxentrydistance_between_configurations()\spxextrain` module `lib.robotics_toolbox.utils.configuration_utils`, 16  
`\spxentrydof\spxextralib.robotics_toolbox.robots.planar_manipulator.PlanarManipulator` property, 11  
`\spxentrydof\spxextralib.robotics_toolbox.robots.PlanarManipulator` tests.robot\_wrapper\_test, 35  
`\spxentrydof\spxextralib.robotics_toolbox.robots.PlanarManipulator` property, 14  
`\spxentrydof\spxextralib.robotics_toolbox.robots.spatial_manipulator.SpatialManipulator` property, 13  
`\spxentrydof\spxextralib.robotics_toolbox.robots.SpatialManipulator` property, 15  
`\spxentryDrone\spxextra` class in `lib.robotics_toolbox.robots`, 14  
`\spxentryDrone\spxextra` class in `lib.robotics_toolbox.robots.drone`, 10  
`\spxentryedge_sharpness()\spxextrain` module `lib.camera_calibration`, 22  
`\spxentryexp()\spxextralib.robotics_toolbox.core.so3.SO3` static method, 4  
`\spxentryexp()\spxextralib.so3.SO3` static method, 29  
`\spxentryexplore()\spxextralib.robotics_toolbox.planning.prm.PRM` method, 6  
`\spxentryfind_aruco_id_vec()\spxextrain` module `src.calibration.hand_eye_calib_with_moves`, 32  
`\spxentryfind_aruco_id_vec()\spxextrain` module `tests.test_robot_camera_calib`, 35  
`\spxentryfk_all_links()\spxextralib.robotics_toolbox.robots.planar_manipulator.PlanarManipulator` method, 11  
`\spxentryfk_all_links()\spxextralib.robotics_toolbox.robots.PlanarManipulator` method, 14  
`\spxentryflange_pose()\spxextralib.robotics_toolbox.robots.planar_manipulator.PlanarManipulator` method, 11  
`\spxentryflange_pose()\spxextralib.robotics_toolbox.robots.PlanarManipulator` method, 14  
`\spxentryflange_pose()\spxextralib.robotics_toolbox.robots.spatial_manipulator.SpatialManipulator` method, 13  
`\spxentryflange_pose()\spxextralib.robotics_toolbox.robots.SpatialManipulator` method, 15  
`\spxentryforward_dynamics()\spxextralib.robotics_toolbox.robots.planar_manipulator_dynamics.PlanarManipulatorDynamics` method, 12  
`\spxentryfrom_angle_axis()\spxextralib.robotics_toolbox.core.so3.SO3` static method, 5  
`\spxentryfrom_angle_axis()\spxextralib.so3.SO3` static method, 29  
`\spxentryfrom_dict()\spxextralib.se3.SE3` class method, 28  
`\spxentryfrom_euler_angles()\spxextralib.robotics_toolbox.core.so3.SO3` static method, 5  
`\spxentryfrom_euler_angles()\spxextralib.so3.SO3` static method, 29  
`\spxentryfrom_quaternion()\spxextralib.robotics_toolbox.core.so3.SO3` static method, 5  
`\spxentryfrom_quaternion()\spxextralib.so3.SO3` static method, 29  
`\spxentrygenerate_robot_positions()\spxextrain` module `src.calibration.hand_eye_calib_with_moves`, 32  
`\spxentrygenerate_robot_positions()\spxextrain` module `tests.robot_wrapper_test`, 35  
`\spxentrygenerate_robot_positions_new()\spxextrain` module `src.calibration.hand_eye_calib_with_moves`, 33  
`\spxentryget_desired_pos_of_custom_gripper()\spxextralib.robot_wrapper.RobotWrapper` method, 25  
`\spxentryget_flange_pos()\spxextralib.robot_wrapper.RobotWrapper` method, 25  
`\spxentryget_gripper_pos()\spxextralib.robot_wrapper.RobotWrapper` method, 25  
`\spxentryget_images()\spxextralib.camera_calibration.CameraCalibration` method, 22  
`\spxentryget_images_from_dir()\spxextrain` module `lib.utils`, 30  
`\spxentryget_name()\spxextralib.robot_wrapper.RobotWrapper` method, 25  
`\spxentryget_path()\spxextralib.robotics_toolbox.planning.prm.GraphPlanner` method, 6  
`\spxentryget_R()\spxextrain` module `lib.utils`, 30  
`\spxentryget_target_transforms_from_camera()\spxextralib.base.Base` method, 20  
`\spxentryget_target_transforms_from_robot()\spxextralib.base.Base` method, 20  
`\spxentryget_updated_base()\spxextralib.base_pose_estimator.BasePoseEstimator` method, 21  
`\spxentryGraphPlanner\spxextra` class in `lib.robotics_toolbox.planning.prm`, 5  
`\spxentrygripper_drop()\spxextralib.robot_wrapper.RobotWrapper` method, 25  
`\spxentrygripper_pick()\spxextralib.robot_wrapper.RobotWrapper` method, 25  
`\spxentryh()\spxextralib.robotics_toolbox.robots.planar_manipulator_dynamics.PlanarManipulatorDynamics` method, 12  
`\spxentryhand_eye_take_image()\spxextrain` module `src.calibration.hand_eye_calib_with_moves`, 33  
`\spxentryHandEyeCalib\spxextra` class in `lib.hand_eye_calib`, 23  
`\spxentryhomogeneous()\spxextralib.robotics_toolbox.core.se2.SE2` method, 3  
`\spxentryhomogeneous()\spxextralib.robotics_toolbox.core.se3.SE3` method, 4

\spxentryhomogeneous()\spxextralib.se3.SE3 method, 28	\spxentryjacobian()\spxextralib.robotics_toolbox.robots.PlanarManipulator method, 15
\spxentryik_analytical()\spxextralib.robotics_toolbox.robots.planar_manipulator.PlanarManipulator method, 11	\spxentryjacobian()\spxextralib.robotics_toolbox.robots.spatial_manipulator method, 13
\spxentryik_analytical()\spxextralib.robotics_toolbox.robots.PlanarManipulator method, 14	\spxentryjacobian()\spxextralib.robotics_toolbox.robots.SpatialManipulator method, 15
\spxentryik_numerical()\spxextralib.robotics_toolbox.robots.planar_manipulator.PlanarManipulator method, 11	\spxentryjacobian_finite_difference()\spxextralib.robotics_toolbox.robots.pl method, 11
\spxentryik_numerical()\spxextralib.robotics_toolbox.robots.PlanarManipulator method, 15	\spxentryjacobian_finite_difference()\spxextralib.robotics_toolbox.robots.Pl method, 15
\spxentryin_collision()\spxextralib.robotics_toolbox.robots.Drone method, 14	\spxentrylib
\spxentryin_collision()\spxextralib.robotics_toolbox.robots.drone.Drone method, 10	\spxentrymodule, 31
\spxentryin_collision()\spxextralib.robotics_toolbox.robots.mobile_robot.MobileRobot method, 10	\spxentrylib.base
\spxentryin_collision()\spxextralib.robotics_toolbox.robots.MobileRobot method, 14	\spxentrylib.base_pose_estimator
\spxentryin_collision()\spxextralib.robotics_toolbox.robots.planar_manipulator.PlanarManipulator method, 11	\spxentrymodule, 21
\spxentryin_collision()\spxextralib.robotics_toolbox.robots.PlanarManipulator method, 15	\spxentrylib.basler_camera
\spxentryin_collision()\spxextralib.robotics_toolbox.robots.PlanarManipulator method, 15	\spxentrylib.camera_calibration
\spxentryin_collision()\spxextralib.robotics_toolbox.robots.robot.Robot method, 13	\spxentrylib.display
\spxentryin_collision()\spxextralib.robotics_toolbox.robots.spatial_manipulator.SpatialManipulator method, 13	\spxentrylib.hand_eye_calib
\spxentryin_collision()\spxextralib.robotics_toolbox.robots.SpatialManipulator method, 15	\spxentrylib.robot_wrapper
\spxentryinitialise_camera()\spxextrain module	\spxentrylib.robotics_toolbox
src.calibration.camera_charuco_calibration_routines, 31	\spxentrylib.robotics_toolbox.core
\spxentryinitialise_camera()\spxextrain module	\spxentrymodule, 20
src.calibration.hand_eye_calib_with_moves, 33	\spxentrylib.robotics_toolbox.core
\spxentryinterpolate()\spxextrain module	\spxentrymodule, 5
lib.robotics_toolbox.utils, 19	\spxentrylib.robotics_toolbox.core.se2
\spxentryinterpolate()\spxextrain module	\spxentrymodule, 3
lib.robotics_toolbox.utils.configuration_utils, 16	\spxentrylib.robotics_toolbox.core.se3
\spxentryinverse()\spxextralib.robotics_toolbox.core.se2.SE2 method, 3	\spxentrymodule, 3
\spxentryinverse()\spxextralib.robotics_toolbox.core.se3.SE3 method, 4	\spxentrylib.robotics_toolbox.core.so2
\spxentryinverse()\spxextralib.robotics_toolbox.core.so2.SO2 method, 4	\spxentrymodule, 4
\spxentryinverse()\spxextralib.robotics_toolbox.core.so3.SO3 method, 5	\spxentrylib.robotics_toolbox.core.so3
\spxentryinverse()\spxextralib.se3.SE3 method, 28	\spxentrymodule, 4
\spxentryinverse()\spxextralib.so2.SO2 method, 28	\spxentrylib.robotics_toolbox.planning
\spxentryinverse()\spxextralib.so3.SO3 method, 29	\spxentrymodule, 7
\spxentryinverse_dynamics()\spxextralib.robotics_toolbox.robots.planar_manipulator.PlanarManipulator method, 12	\spxentrylib.robotics_toolbox.planning.prm
	\spxentrymodule, 5
	\spxentrylib.robotics_toolbox.planning.rrt
	\spxentrymodule, 6
	\spxentrylib.robotics_toolbox.render
	\spxentrymodule, 9
	\spxentrylib.robotics_toolbox.render.mobile_robot_renderer
	\spxentrymodule, 7
	\spxentrylib.robotics_toolbox.render.renderer_planar
	\spxentrymodule, 7
	\spxentrylib.robotics_toolbox.render.renderer_spatial

\spxentrymodule, 8	\spxentrymain()\spxextrain module
\spxentrylib.robotics_toolbox.render.se2_renderer	src.calibration.hand_eye_calib_from_data,
\spxentrymodule, 9	32
\spxentrylib.robotics_toolbox.robots	\spxentrymain()\spxextrain module
\spxentrymodule, 14	src.calibration.hand_eye_calib_with_moves, 33
\spxentrylib.robotics_toolbox.robots.drone	\spxentrymain()\spxextrain module tests.aruco_test, 34
\spxentrymodule, 10	\spxentrymain()\spxextrain module tests.camera_test, 34
\spxentrylib.robotics_toolbox.robots.mobile_robot	\spxentrymain()\spxextrain module tests.conseq_moves,
\spxentrymodule, 10	34
\spxentrylib.robotics_toolbox.robots.planar_manipulator	\spxentrymain()\spxextrain module
\spxentrymodule, 11	tests.conseq_moves_2, 34
\spxentrylib.robotics_toolbox.robots.planar_manipulator_dyn	\spxentrymain()\spxextrain module
\spxentrymodule, 12	tests.conseq_moves_3, 34
\spxentrylib.robotics_toolbox.robots.robot_base	\spxentrymain()\spxextrain module
\spxentrymodule, 13	tests.conseq_moves_4, 35
\spxentrylib.robotics_toolbox.robots.spatial_manipulator	\spxentrymain()\spxextrain module
\spxentrymodule, 13	tests.conseq_moves_5, 35
\spxentrylib.robotics_toolbox.utils	\spxentrymain()\spxextrain module tests.reproj_error, 35
\spxentrymodule, 18	\spxentrymain()\spxextrain module
\spxentrylib.robotics_toolbox.utils.animation_utils	tests.robot_wrapper_test, 35
\spxentrymodule, 15	\spxentrymain()\spxextrain module
\spxentrylib.robotics_toolbox.utils.configuration_utils	tests.test_robot2target, 35
\spxentrymodule, 16	\spxentrymain()\spxextrain module
\spxentrylib.robotics_toolbox.utils.geometry_utils	tests.test_robot_camera_calib, 35
\spxentrymodule, 17	\spxentrymass_matrix()\spxextralib.robotics_toolbox.robots.planar_manipu
\spxentrylib.se3	method, 12
\spxentrymodule, 28	\spxentryMobileRobot\spxextraclass in
\spxentrylib.so2	lib.robotics_toolbox.robots, 14
\spxentrymodule, 28	\spxentryMobileRobot\spxextraclass in
\spxentrylib.so3	lib.robotics_toolbox.robots.mobile_robot,
\spxentrymodule, 29	10
\spxentrylib.utils	\spxentryMobileRobotRenderer\spxextraclass in
\spxentrymodule, 30	lib.robotics_toolbox.render.mobile_robot_renderer,
\spxentryload_camera_calibration_data()\spxextrain	7
module lib.utils, 30	\spxentrymodule
\spxentryload_fk_transformation()\spxextralib.hand_eye_calib	\spxentrylib, 31
method, 23	\spxentrylib.base, 20
\spxentryload_fk_transformation_and_points()\spxextralib.hand_eye_calib	\spxentrylib.HandEyeCalibEstimator, 21
method, 23	\spxentrylib.basler_camera, 22
\spxentryload_initial_parameters()\spxextralib.hand_eye_calib	\spxentrylib.basler_camera_calibration, 22
method, 23	\spxentrylib.display, 22
\spxentryload_json_data()\spxextrain module	\spxentrylib.hand_eye_calib, 23
src.calibration.hand_eye_calib_from_data,	\spxentrylib.robot_wrapper, 25
32	\spxentrylib.robotics_toolbox, 20
\spxentrylog()\spxextralib.robotics_toolbox.core.so3.SO3	\spxentrylib.robotics_toolbox.core, 5
method, 5	\spxentrylib.robotics_toolbox.core.se2, 3
\spxentrylog()\spxextralib.so3.SO3 method, 29	\spxentrylib.robotics_toolbox.core.se3, 3
	\spxentrylib.robotics_toolbox.core.so2, 4
	\spxentrylib.robotics_toolbox.core.so3, 4
\spxentrymain()\spxextrain module	\spxentrylib.robotics_toolbox.planning, 7
src.calibration.camera_calib_new, 31	\spxentrylib.robotics_toolbox.planning.prm, 5
\spxentrymain()\spxextrain module	\spxentrylib.robotics_toolbox.planning.rrt, 6
src.calibration.camera_charuco_calibration_routine,	\spxentrylib.robotics_toolbox.render, 9
32	\spxentrylib.robotics_toolbox.render.mobile_robot_renderer,

7	\spxentrymove_custom_gripper()\spxextralib.robot_wrapper.Robot
\spxentrylib.robotics_toolbox.render.planar_manipulator_renderer, 25	method, 25
7	\spxentrymove_to_angles()\spxextralib.robot_wrapper.Robot
\spxentrylib.robotics_toolbox.render.renderer_planar,	method, 26
8	\spxentrymove_to_camera_capture_pos()\spxextralib.robot_wrapper.Robot
\spxentrylib.robotics_toolbox.render.renderer_spatial,	method, 26
8	\spxentrymove_to_coordinates()\spxextralib.robot_wrapper.Robot
\spxentrylib.robotics_toolbox.render.se2_renderer, 9	method, 26
\spxentrylib.robotics_toolbox.robots, 14	\spxentrymove_to_gate()\spxextralib.robot_wrapper.Robot
\spxentrylib.robotics_toolbox.robots.drone, 10	method, 26
\spxentrylib.robotics_toolbox.robots.mobile_robot,	\spxentrymove_under_camera()\spxextralib.robot_wrapper.Robot
10	method, 26
\spxentrylib.robotics_toolbox.robots.planar_manipulator,	
11	\spxentryNode\spxextraclass in
\spxentrylib.robotics_toolbox.robots.planar_manipulator_dynamics, 6	lib.robotics_toolbox.planning.prm, 6
12	\spxentrynullspace()\spxextrain module
\spxentrylib.robotics_toolbox.robots.robot_base, 13	lib.robotics_toolbox.utils, 19
\spxentrylib.robotics_toolbox.robots.spatial_manipulator,	\spxentrynullspace()\spxextrain module
13	lib.robotics_toolbox.utils.geometry_utils,
\spxentrylib.robotics_toolbox.utils, 18	18
\spxentrylib.robotics_toolbox.utils.animation_utils,	\spxentrypick_drop_part()\spxextralib.robot_wrapper.Robot
15	method, 27
\spxentrylib.robotics_toolbox.utils.configuration_utils,	\spxentrypick_drop_part_with_wiggle()\spxextralib.robot_wrapper.Robot
16	method, 27
\spxentrylib.robotics_toolbox.utils.geometry_utils,	\spxentryplan()\spxextralib.robotics_toolbox.planning.prm.PRM
17	method, 6
\spxentrylib.se3, 28	\spxentryplan()\spxextralib.robotics_toolbox.planning.rrt.RRT
\spxentrylib.so2, 28	method, 6
\spxentrylib.so3, 29	\spxentryPlanarManipulator\spxextraclass in
\spxentrylib.utils, 30	lib.robotics_toolbox.robots, 14
\spxentrysrc, 34	\spxentryPlanarManipulator\spxextraclass in
\spxentrysrc.calibration, 33	lib.robotics_toolbox.robots.planar_manipulator,
\spxentrysrc.calibration.camera_calib_new, 31	11
\spxentrysrc.calibration.camera_charuco_calibration_routine,	\spxentryPlanarManipulatorDynamics\spxextraclass in
31	lib.robotics_toolbox.robots.planar_manipulator_dynamics,
\spxentrysrc.calibration.hand_eye_calib_from_data,	12
32	\spxentryPlanarManipulatorRenderer\spxextraclass in
\spxentrysrc.calibration.hand_eye_calib_with_moves,	lib.robotics_toolbox.render.planar_manipulator_renderer,
32	7
\spxentrysrc.move_commands, 34	\spxentryplot_drone()\spxextralib.robotics_toolbox.render.renderer_spatial.L
\spxentrytests, 35	method, 8
\spxentrytests.aruco_test, 34	\spxentryplot_drone()\spxextralib.robotics_toolbox.render.RendererSpatial
\spxentrytests.camera_test, 34	method, 10
\spxentrytests.conseq_moves, 34	\spxentryplot_init()\spxextralib.robotics_toolbox.render.planar_manipulator
\spxentrytests.conseq_moves_2, 34	method, 7
\spxentrytests.conseq_moves_3, 34	\spxentryplot_line_between_points()\spxextralib.robotics_toolbox.render.pl
\spxentrytests.conseq_moves_4, 35	method, 8
\spxentrytests.conseq_moves_5, 35	\spxentryplot_line_between_points()\spxextralib.robotics_toolbox.render.re
\spxentrytests.reproj_error, 35	method, 8
\spxentrytests.robot_wrapper_test, 35	\spxentryplot_line_between_points()\spxextralib.robotics_toolbox.render.Re
\spxentrytests.test_robot2target, 35	method, 9
\spxentrytests.test_robot_camera_calib, 35	\spxentryplot_line_between_points()\spxextralib.robotics_toolbox.render.se
\spxentrytests.test_show_positions, 35	method, 9



\spxentryplot\_manipulator()\spxextralib.robotics\_toolbox.render.render\_method, 27  
 method, 8 \spxentryreturn\_nearest\_node()\spxextralib.robotics\_toolbox.planning.rrt.Tr  
 \spxentryplot\_manipulator()\spxextralib.robotics\_toolbox.render.render\_method, 11  
 method, 8 \spxentryreturn\_parent()\spxextralib.robotics\_toolbox.planning.rrt.TreeNod  
 \spxentryplot\_manipulator()\spxextralib.robotics\_toolbox.render.RendererPlanar  
 method, 9 \spxentryreturn\_path\_from\_start()\spxextralib.robotics\_toolbox.planning.rrt  
 \spxentryplot\_manipulator()\spxextralib.robotics\_toolbox.render.RendererSpatial  
 method, 10 \spxentryRobot\spxextraclass in lib.robot\_wrapper, 25  
 \spxentryplot\_mobile\_robot()\spxextralib.robotics\_toolbox.render.render\_method, 8  
 method, 8 \spxentryRobotBase\spxextraclass in lib.robot\_wrapper, 25  
 method, 9 lib.robotics\_toolbox.robots.robot\_base, 13  
 \spxentryplot\_se2()\spxextralib.robotics\_toolbox.render.render\_method, 8  
 method, 8 \spxentryRobotBase\spxextraclass in lib.robot\_wrapper, 25  
 \spxentryplot\_se2()\spxextralib.robotics\_toolbox.render.RendererPlanar()\spxextralib.robotics\_toolbox.core.so3.SO3  
 method, 9 static method, 5  
 \spxentryplot\_se3()\spxextralib.robotics\_toolbox.render.render\_method, 9  
 method, 9 \spxentryRobotBase\spxextraclass in lib.robot\_wrapper, 25  
 \spxentryplot\_se3()\spxextralib.robotics\_toolbox.render.RendererSpatial()\spxextralib.robotics\_toolbox.core.so3.SO3  
 method, 10 static method, 5  
 \spxentryplot\_se3()\spxextralib.robotics\_toolbox.render.RendererSpatial static method, 5  
 method, 10 \spxentryRobotBase\spxextraclass in lib.robot\_wrapper, 25  
 \spxentryplot\_so2()\spxextralib.robotics\_toolbox.render.render\_method, 8  
 method, 8 \spxentryRobotBase\spxextraclass in lib.robot\_wrapper, 25  
 \spxentryplot\_so2()\spxextralib.robotics\_toolbox.render.RendererPlanar()\spxextralib.robotics\_toolbox.core.so3.SO3  
 method, 9 static method, 5  
 \spxentryplot\_so2()\spxextralib.robotics\_toolbox.render.RendererPlanar static method, 29  
 method, 9 \spxentryRobotBase\spxextraclass in lib.robot\_wrapper, 25  
 \spxentryposition\_with\_adaptive\_offset()\spxextralib.robot\_wrapper.Robot  
 method, 27 \spxentrysample\_configuration()\spxextralib.robotics\_toolbox.robots.Drone  
 method, 23 \spxentrysample\_configuration()\spxextralib.robotics\_toolbox.robots.drone.  
 method, 23 \spxentrysample\_configuration()\spxextralib.robotics\_toolbox.robots.drone.  
 method, 23 \spxentrysample\_configuration()\spxextralib.robotics\_toolbox.robots.drone.  
 \spxentryPRM\spxextraclass in \spxentrysample\_configuration()\spxextralib.robotics\_toolbox.robots.mobile  
 lib.robotics\_toolbox.planning.prm, 6 method, 10  
 \spxentryrandom\_shortcut()\spxextralib.robotics\_toolbox.planning.rrt.RRT, 14  
 method, 6 \spxentrysample\_configuration()\spxextralib.robotics\_toolbox.robots.planar  
 \spxentryread\_data\_from\_json()\spxextralib.robotics\_toolbox.planning.rrt.RRT, 14  
 lib.utils, 30 method, 11  
 \spxentryredraw\_all()\spxextralib.robotics\_toolbox.render.render\_method, 8  
 method, 8 \spxentrysample\_configuration()\spxextralib.robotics\_toolbox.robots.planar  
 \spxentryredraw\_all()\spxextralib.robotics\_toolbox.render.RendererPlanar  
 method, 9 method, 13  
 \spxentryRendererPlanar\spxextraclass in \spxentrysample\_configuration()\spxextralib.robotics\_toolbox.robots.spatial  
 lib.robotics\_toolbox.render, 9 method, 13  
 \spxentryRendererPlanar\spxextraclass in \spxentrysample\_configuration()\spxextralib.robotics\_toolbox.robots.spatial  
 lib.robotics\_toolbox.render.render\_method, 8 method, 15  
 \spxentryRendererSpatial\spxextraclass in \spxentrysave\_fig()\spxextralib.robotics\_toolbox.robots.spatial  
 lib.robotics\_toolbox.render, 10 lib.robotics\_toolbox.utils, 19  
 \spxentryRendererSpatial\spxextraclass in \spxentrysave\_fig()\spxextralib.robotics\_toolbox.robots.spatial  
 lib.robotics\_toolbox.render.render\_method, 8 16  
 \spxentryreprojection\_error()\spxextralib.hand\_eye\_calib.HandEyeCalib\spxextraclass in  
 method, 24 lib.robotics\_toolbox.core.se2, 3  
 \spxentryreprojection\_error\_ls()\spxextralib.hand\_eye\_calib.HandEyeCalib\spxextraclass in  
 method, 24 lib.robotics\_toolbox.render.se2\_renderer,  
 \spxentryreprojection\_optimization()\spxextralib.hand\_eye\_calib.HandEyeCalib  
 method, 24 \spxentrySE3\spxextraclass in  
 \spxentryreturn\_best\_ik()\spxextralib.robot\_wrapper.Robot lib.robotics\_toolbox.core.se3, 3

\spxentrySE3\spxextraclass in lib.se3, 28	13
\spxentryset_base()\spxextralib.base_pose_estimator.BasePoseEstimator	\spxentrymodule, 34
method, 21	\spxentrymodule, 33
\spxentryset_camera_matrix()\spxextralib.base_pose_estimator.BasePoseEstimator	\spxentrymodule, 31
method, 21	\spxentrymodule, 31
\spxentryset_configuration()\spxextralib.robotics_toolbox.robots.Drone	\spxentrymodule, 32
method, 14	\spxentrymodule, 32
\spxentryset_configuration()\spxextralib.robotics_toolbox.robots.Drone	\spxentrymodule, 34
method, 10	\spxentrymodule, 35
\spxentryset_configuration()\spxextralib.robotics_toolbox.robots.MobileRobot	\spxentrymodule, 34
method, 10	\spxentrymodule, 34
\spxentryset_configuration()\spxextralib.robotics_toolbox.robots.MobileRobot	\spxentrymodule, 34
method, 14	\spxentrymodule, 34
\spxentryset_configuration()\spxextralib.robotics_toolbox.robots.PlanarManipulator	\spxentrymodule, 34
method, 11	\spxentrymodule, 34
\spxentryset_configuration()\spxextralib.robotics_toolbox.robots.PlanarManipulator	\spxentrymodule, 34
method, 15	\spxentrymodule, 34
\spxentryset_configuration()\spxextralib.robotics_toolbox.robots.RobotBase	\spxentrymodule, 34
method, 13	\spxentrymodule, 34
\spxentryset_configuration()\spxextralib.robotics_toolbox.robots.SpatialManipulator	\spxentrymodule, 34
method, 13	\spxentrymodule, 34
\spxentryset_configuration()\spxextralib.robotics_toolbox.robots.SpatialManipulator	\spxentrymodule, 34
method, 15	\spxentrymodule, 34
\spxentryset_distortion_coefficients()\spxextralib.base_pose_estimator.BasePoseEstimator	\spxentrymodule, 34
method, 21	\spxentrymodule, 34
\spxentryset_from()\spxextralib.robotics_toolbox.core.se2.SE2	\spxentrymodule, 34
method, 3	\spxentrymodule, 34
\spxentryset_from()\spxextralib.robotics_toolbox.core.se3.SE3	\spxentrymodule, 34
method, 4	\spxentrymodule, 34
\spxentryset_from()\spxextralib.se3.SE3 method, 28	\spxentrymodule, 34
\spxentryset_image_path()\spxextralib.camera_calibration.CameraCalib	\spxentrymodule, 34
method, 22	\spxentrymodule, 34
\spxentryset_position()\spxextralib.base.Base method, 20	\spxentrymodule, 34
\spxentryset_rotation()\spxextralib.base_pose_estimator.BasePoseEstimator	\spxentrymodule, 34
method, 22	\spxentrymodule, 34
\spxentrysetup()\spxextralib.robot_wrapper.Robot	\spxentrymodule, 34
method, 27	\spxentrymodule, 34
\spxentrysetup_positions()\spxextralib.display.Display	\spxentrymodule, 34
method, 23	\spxentrymodule, 34
\spxentrySO2\spxextraclass	\spxentrymodule, 34
lib.robotics_toolbox.core.so2, 4	\spxentrymodule, 34
\spxentrySO2\spxextraclass in lib.so2, 28	\spxentrymodule, 34
\spxentrySO3\spxextraclass	\spxentrymodule, 34
lib.robotics_toolbox.core.so3, 4	\spxentrymodule, 34
\spxentrySO3\spxextraclass in lib.so3, 29	\spxentrymodule, 34
\spxentrysolve_AX_YB()\spxextrain module	\spxentrymodule, 34
src.calibration.hand_eye_calib_from_data,	\spxentrymodule, 34
32	\spxentrymodule, 34
\spxentrysolve_AX_YB()\spxextrain module	\spxentrymodule, 34
src.calibration.hand_eye_calib_with_moves, 33	\spxentrymodule, 34
\spxentrySpatialManipulator\spxextraclass	\spxentrymodule, 34
lib.robotics_toolbox.robots, 15	\spxentrymodule, 34
\spxentrySpatialManipulator\spxextraclass	\spxentrymodule, 34
lib.robotics_toolbox.robots.spatial_manipulator,	\spxentrymodule, 34



`\spxentryupdate()\spxextralib.robotics_toolbox.render.mobile_robot_renderer.MobileRobotRenderer`  
method, 7

`\spxentryupdate()\spxextralib.robotics_toolbox.render.planar_manipulator_renderer.PlanarManipulatorRenderer`  
method, 8

`\spxentryupdate()\spxextralib.robotics_toolbox.render.se2_renderer.SE2Renderer`  
method, 9

`\spxentryvector_to_se3()\spxextrain module lib.utils`, 31

`\spxentrywait_at_the_end()\spxextralib.robotics_toolbox.render.renderer_spatial.RendererSpatial`  
method, 9

`\spxentrywait_at_the_end()\spxextralib.robotics_toolbox.render.RendererSpatial`  
method, 10

`\spxentrywait_for_close()\spxextralib.robotics_toolbox.render.renderer_planar.RendererPlanar`  
static method, 8

`\spxentrywait_for_close()\spxextralib.robotics_toolbox.render.RendererPlanar`  
static method, 10

`\spxentrywait_for_enter()\spxextralib.robotics_toolbox.render.renderer_planar.RendererPlanar`  
static method, 8

`\spxentrywait_for_enter()\spxextralib.robotics_toolbox.render.renderer_spatial.RendererSpatial`  
static method, 9

`\spxentrywait_for_enter()\spxextralib.robotics_toolbox.render.RendererPlanar`  
static method, 10

`\spxentrywait_for_enter()\spxextralib.robotics_toolbox.render.RendererSpatial`  
static method, 10

`\spxentrywiggle_move_to_coords()\spxextralib.robot_wrapper.Robot`  
method, 27

`\spxentrywiggle_wiggle_wiggle()\spxextralib.robot_wrapper.Robot`  
method, 28

`\spxentrywrite_data_to_json()\spxextrain` module  
lib.utils, 31