TriP Documentation

Release 0.9

Torben Miller

Aug 25, 2021

CONTENTS

CHAPTER

ONE

CODE DOCUMENTATION

class `trip_kinematics.HomogenTransformationMatrix.`**`TransformationMatrix`**(*qw=1, qx=0, qy=0, qz=0, tx=0, ty=0, tz=0, conv='quat', rx=0, ry=0, rz=0*)

Class Implementing and Building a simple homogenous transformation matrix.

Parameters

- **qw** (*float, optional*) – part of a quaternion [qw,qx,qy,qz]. Defaults to 1.
- **qx** (*float, optional*) – part of a quaternion [qw,qx,qy,qz]. Defaults to 0.
- **qy** (*float, optional*) – part of a quaternion [qw,qx,qy,qz]. Defaults to 0.
- **qz** (*float, optional*) – part of a quaternion [qw,qx,qy,qz]. Defaults to 0.
- **tx** (*float, optional*) – translation along x-axis. Defaults to 0.
- **ty** (*float, optional*) – translation along y-axis. Defaults to 0.
- **tz** (*float, optional*) – translation along z-axis. Defaults to 0.
- **conv** (*str, optional*) – convention used to build the transformation matrix. Defaults to 'quat'.
- **rx** (*float, optional*) – rotation around the x-axis. Defaults to 0.
- **ry** (*float, optional*) – rotation around the y-axis. Defaults to 0.
- **rz** (*float, optional*) – rotation around the z-axis. Defaults to 0.

`get_rotation`()

Returns the 3x3 rotation matrix of the :py:class`TransformationMatrix`

Returns The 3x3 rotation matrix

Return type numpy.array

`get_translation`()

Returns the translation of the :py:class`TransformationMatrix`

Returns The 3 dimensional translation

Return type numpy array

`trip_kinematics.HomogenTransformationMatrix.`**`quat_rotation_matrix`**(*qw*, *qx*, *qy*, *qz*) → numpy.ndarray

Generates a 3x3 rotation matrix from q quaternion

Parameters

- **qw** (*float*) – part of a quaternion [qw,qx,qy,qz]
- **qx** (*float*) – part of a quaternion [qw,qx,qy,qz]
- **qy** (*float*) – part of a quaternion [qw,qx,qy,qz]
- **qz** (*float*) – part of a quaternion [qw,qx,qy,qz]

Returns A 3x3 rotation matrix

Return type np.array

`trip_kinematics.HomogenTransformationMatrix.`**`x_axis_rotation_matrix`**(*theta*)

Generates a matrix rotating around the x axis

Parameters **theta** (*float*) – The angle of rotation in rad

Returns A 3x3 rotation matrix

Return type np.array

`trip_kinematics.HomogenTransformationMatrix.`**`y_axis_rotation_matrix`**(*theta*)

Generates a matrix rotating around the y axis

Parameters **theta** (*float*) – The angle of rotation in rad

Returns A 3x3 rotation matrix

Return type np.array

`trip_kinematics.HomogenTransformationMatrix.`**`z_axis_rotation_matrix`**(*theta*)

Generates a matrix rotating around the z axis

Parameters **theta** (*float*) – The angle of rotation in rad

Returns A 3x3 rotation matrix

Return type np.array

class `trip_kinematics.KinematicGroup.`**`KinematicGroup`**(*name: str*, *virtual_transformations: List[*trip_kinematics.KinematicGroup.Transformation*]*, *actuated_state: List[Dict[str, float]] = None*, *actuated_to_virtual: Callable = None*, *virtual_to_actuated: Callable = None*, *act_to_virt_args=None*, *virt_to_act_args=None*, *parent=None*)

Initializes a *KinematicGroup* object.

Parameters

- **name** (*str*) – The unique name identifying the group. No two *KinematicGroup* objects of a :py:class`Robot` should have the same name

- **virtual_transformations** (*List[*Transformation*]*) – A list of *Transformation* objects forming a serial Kinematic chain.
- **actuated_state** (*List[Dict[str, float]], optional*) – The State of the Groups actuated joints. Defaults to None.
- **actuated_to_virtual** (*Callable, optional*) – Maps the `actuated_state` to the `virtual_state` of the `virtual_transformations`. Defaults to None.
- **virtual_to_actuated** (*Callable, optional*) – Maps the `virtual_state` of the `virtual_transformations` to the `actuated_state`.
- **act_to_virt_args** (*[type], optional*) – Arguments that can be passed to `actuated_to_virtual` during the initial testing of the function. Defaults to None.
- **virt_to_act_args** (*[type], optional*) – Arguments that can be passed to `virtual_to_actuated` during the initial testing of the function. Defaults to None.
- **parent** (*[type], optional*) – [description]. Defaults to None.

Raises

- **ValueError** – 'Error: Actuated state is missing. You provided a mapping to actuate the group but no state to be actuated.' if there is no `actuated_state` despite a mapping being passed
- **ValueError** – 'Error: Only one mapping provided. You need mappings for both ways. Consider to pass a trivial mapping.' if either `actuated_to_virtual` or `virtual_to_actuated` was not set despite providing a `actuated_state`.
- **ValueError** – 'Error: Mappings missing. You provided an actuated state but no mappings. If you want a trivial mapping you don't need to pass an actuated state. Trip will generate one for you.' if both `actuated_to_virtual` and `virtual_to_actuated` were not set despite providing a `actuated_state`.
- **RuntimeError** – "actuated_to_virtual does not fit virtual state" if the `actuated_to_virtual` function does not return a valid `virtual_state` dictionary
- **RuntimeError** – "virtual_to_actuated does not fit actuated state" if the `virtual_to_actuated` function does not return a valid `actuated_state` dictionary

get_transformation_matrix() → *trip_kinematics.HomogenTransformationMatrix.TransformationMatrix*

Calculates the full transformationmatrix from the start of the virtual chain to its endeffector.

Returns The homogenous transformation matrix from the start of the virtual chain to its endeffector.

Return type *TransformationMatrix*

set_actuated_state(*state: Dict[str, float]*)

Sets the `__actuated_state` of the Group and automatically updates the corresponding `__virtual_state`

Parameters **state** (*Dict[str, float]*) – A dictionary containing the members of `__actuated_state` that should be set.

Raises

- **RuntimeError** – "This is a static group! There is no state to be set" if all *Transformation* objects of `__virtual_transformations` are static.

- **ValueError** – Error: State not set! Keys do not match! Make sure that your state includes the same keys as your intial virtual transformations." if the state to set is not part of keys of `__actuated_state`

set_virtual_state(*state: Dict[str, Dict[str, float]]*)

Sets the `__virtual_state` of the Group and automatically updates the corresponding `__actuated_state`

Parameters **state** (*Dict[str,Dict[str, float]]*) – A dictionary containing the members of `__virtual_state` that should be set. The new values need to be valid state for the state of the joint.

Raises

- **RuntimeError** – "This is a static group! There is no state to be set" if all *Transformation* objects of `__virtual_transformations` are static.
- **ValueError** – "Error: State not set! Keys do not match! Make sure that your state includes the same keys as your intial virtual transformations." if the state to set is not part of keys of `__virtual_state`

class trip_kinematics.KinematicGroup.**Transformation**(*name: str, values: Dict[str, float], state_variables: List[str] = []*)

Initializes the *Transformation* class.

Parameters

- **name** (*str*) – The unique name identifying the . No two *Transformation* objects of a :py:class`Robot` should have the same name
- **values** (*Dict[str, float]*) – A parametric description of the transformation.
- **state_variables** (*List[str], optional*) – This list describes which state variables are dynamically changable. This is the case if the *Transformation* represents a joint. Defaults to [].

Raises **ValueError** – A dynamic state was declared that does not correspond to a parameter declared in `values`.

static get_convention(*state: Dict[str, float]*)

Returns the connvention which describes how the matrix of a *Transformation* is build from its state.

Parameters **state** (*Dict[str, float]*) – :py:attr:'state'

Raises

- **ValueError** – "Invalid key." If the dictionary kontains keys that dont correspond to a parameter of the transformation.
- **ValueError** – "State can't have euler angles and quaternions!" If the dictionary contains keys correspondig to multiple mutually exclusive conventions.

Returns A string describing the convention

Return type [type]

get_transformation_matrix()

Returns a homogeneous transformation matrix build from the `state` and `constants`

Raises **RuntimeError** – If the convention used in `state` is not supported. Should normally be catched during initialization.

Returns A transformation matrix build using the parameters of the *Transformation* `state`

Return type [type]

class `trip_kinematics.Robot.`**Robot**(*kinematic_chain: List[*trip_kinematics.KinematicGroup.KinematicGroup*]*)

A class managing multiple :py:class`KinematicGroup` objects pable of building tree like kinematic topologies.

Parameters **kinematic_chain** (*List[*KinematicGroup*]*) – A list of Kinematic Groups with make up the robot.

Raises

- **KeyError** – "More than one robot actuator has the same name! Please give each actuator a unique name" if there are actuated states with the same names between the :py:class`KinematicGroup` objects of the :py:class`Robot`
- **KeyError** – "More than one robot virtual transformation has the same name! Please give each virtual transformation a unique name" if there are joints with the same names between the :py:class`KinematicGroup` objects of the :py:class`Robot`

get_actuated_state()

Returns the actuated state of the :py:class`Robot` comprised of the actuated states of the individual :py:class`KinematicGroup`.

Returns combined actuated state of all :py:class`KinematicGroup` objects.

Return type Dict[str, float]

get_groups()

Returns a dictionary of the py:class`KinematicGroup` managed by the :py:class`Robot`-

Returns The dictionary of py:class`KinematicGroup` objects.

Return type Dict[str, *KinematicGroup*]

get_symbolic_rep(*opti_obj*, *endeffector*)

This Function returnes a symbolic representation of the virtual chain.

Returns The `TransformationMatrix` containing symbolic objects

Return type *TransformationMatrix*

get_virtual_state()

Returns the virtual state of the :py:class`Robot` comprised of the virtual states of the individual :py:class`KinematicGroup`.

Returns combined virtual state of all :py:class`KinematicGroup` objects.

Return type Dict[str,Dict[str, float]]

set_actuated_state(*state: Dict[str, float]*)

Sets the virtual state of multiple actuated joints of the robot.

Parameters **state** (*Dict[str, float]*) – A dictionary containing the members of `__actuated_state` that should be set.

set_virtual_state(*state: Dict[str, Dict[str, float]]*)

Sets the virtual state of multiple virtual joints of the robot.

Parameters **state** (*Dict[str,Dict[str, float]]*) – A dictionary containing the members of `__virtual_state` that should be set. The new values need to be valid state for the state of the joint.

static solver_to_virtual_state(*sol*, *symbolic_state*)

This Function maps the solution of a opti solver to the virtual state of the robot

Parameters

- **sol** (*[type]*) – A opti solver object
- **symbolic_state** (*[type]*) – the description of the symbolic state that corresponds to the solver values

Returns a `virtual_state` of a robot.

Return type Dict[str,Dict[str, float]]

`trip_kinematics.Robot.`**`forward_kinematics`**(*robot:* trip_kinematics.Robot.Robot)

Calculates a robots transformation from base to endeffector using its current state

Parameters **robot** (`Robot`) – The robot for which the forward kinematics should be computed

Returns The Transformation from base to endeffector

Return type numpy.array

`trip_kinematics.Robot.`**`inverse_kinematics`**(*robot:* trip_kinematics.Robot.Robot, *end_effector_position*)

Simple Inverse kinematics algorithm that computes the actuated state necessairy for the endeffector to be at a specified position

Parameters

- **robot** (`Robot`) – The robot for which the inverse kinematics should be computed
- **end_effector_position** (*[type]*) – the desrired endeffector position

Returns combined actuated state of all :py:class`KinematicGroup` objects.

Return type Dict[str, float]

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

t

`trip_kinematics.HomogenTransformationMatrix`, ??
`trip_kinematics.KinematicGroup`, ??
`trip_kinematics.Robot`, ??