PyRep

Aug 22, 2023

Contents:

1 pyrep package **3**
- 1.1 Subpackages . . . 3
  - 1.1.1 pyrep.objects package . . . 3
    - 1.1.1.1 Submodules . . . 3
    - 1.1.1.2 pyrep.objects.cartesian_path module . . . 3
    - 1.1.1.3 pyrep.objects.dummy module . . . 4
    - 1.1.1.4 pyrep.objects.force_sensor module . . . 4
    - 1.1.1.5 pyrep.objects.joint module . . . 5
    - 1.1.1.6 pyrep.objects.object module . . . 7
    - 1.1.1.7 pyrep.objects.proximity_sensor module . . . 14
    - 1.1.1.8 pyrep.objects.shape module . . . 14
    - 1.1.1.9 pyrep.objects.vision_sensor module . . . 20
    - 1.1.1.10 Module contents . . . 23
  - 1.1.2 pyrep.robots package . . . 23
    - 1.1.2.1 Subpackages . . . 23
    - 1.1.2.2 Submodules . . . 29
    - 1.1.2.3 pyrep.robots.robot_component module . . . 29
    - 1.1.2.4 Module contents . . . 32
- 1.2 Submodules . . . 32
- 1.3 pyrep.const module . . . 32
- 1.4 pyrep.errors module . . . 35
- 1.5 pyrep.pyrep module . . . 35
- 1.6 Module contents . . . 38

2 Indices and tables **39**

Python Module Index **41**

Index **43**

PyRep is a toolkit for robot learning research, built on top of the virtual robotics experimentation platform (V-REP).

See the Github page for install instructions and examples.

CHAPTER 1

pyrep package

1.1 Subpackages

1.1.1 pyrep.objects package

1.1.1.1 Submodules

1.1.1.2 pyrep.objects.cartesian_path module

class pyrep.objects.cartesian_path.**CartesianPath**(*name_or_handle: Union[str, int]*)
Bases: *pyrep.objects.object.Object*

An object that defines a cartesian path or trajectory in space.

static create(*show_line: bool = True*, *show_orientation: bool = True*, *show_position: bool = True*, *closed_path: bool = False*, *automatic_orientation: bool = True*, *flat_path: bool = False*, *keep_x_up: bool = False*, *line_size: int = 1*, *length_calculation_method: int = 5*, *control_point_size: float = 0.01*, *ang_to_lin_conv_coeff: float = 1.0*, *virt_dist_scale_factor: float = 1.0*, *path_color: tuple = (0.1, 0.75, 1.0)*) → pyrep.objects.cartesian_path.CartesianPath

Creates a cartesian path and inserts in the scene.

Parameters

- **show_line** – Shows line in UI.
- **show_position** – Shows line in UI.
- **show_orientation** – Shows orientation in UI.
- **closed_path** – If set, then a path's last control point will be linked to its first control point to close the path and make its operation cyclic. A minimum of 3 control points are required for a path to be closed.
- **automatic_orientation** – If set, then all control points and Bezier point's orientation will automatically be calculated in order to have a point's z-axis along the path, and

its y-axis pointing outwards its curvature (if keep x up is enabled, the y-axis is not particularly constrained). If disabled, the user determines the control point's orientation and the Bezier points' orientation will be interpolated from the path's control points' orientation.

- **flat_path** – If set, then all control points (and subsequently all Bezier points) will be constraint to the z=0 plane of the path object's local reference frame.
- **keep_x_up** – If set, then the automatic orientation functionality will align each Bezier point's z-axis along the path and keep its x-axis pointing along the path object's z-axis.
- **line_size** – Size of the line in pixels.
- **length_calculation_method** – Method for calculating the path length. See https://www.coppeliarobotics.com/helpFiles/en/apiConstants.htm#distanceCalculationMethods
- **control_point_size** – Size of the control points in the path.
- **ang_to_lin_conv_coeff** – The angular to linear conversion coefficient.
- **virt_dist_scale_factor** – The virtual distance scaling factor.
- **path_color** – Ambient diffuse rgb color of the path.

Returns The newly created cartesian path.

get_pose_on_path(*relative_distance: float*) → Tuple[List[float], List[float]]
Retrieves the absolute interpolated pose of a point along the path.

Parameters **relative_distance** – A value between 0 and 1, where 0 is the beginning of the path, and 1 the end of the path.

Returns A tuple containing the x, y, z position, and the x, y, z orientation of the point on the path (in radians).

insert_control_points(*poses: List[List[float]]*) → None
Inserts one or several control points into the path.

Parameters **poses** – A list of lists containing 6 values representing the pose of each of the new control points. Orientation in radians.

1.1.1.3 pyrep.objects.dummy module

class pyrep.objects.dummy.**Dummy**(*name_or_handle: Union[str, int]*)
Bases: *pyrep.objects.object.Object*

A point with orientation.

Dummies are multipurpose objects that can have many different applications.

static create(*size=0.01*) → pyrep.objects.dummy.Dummy
Creates a dummy object and inserts in the scene.

Parameters **size** – The size of the dummy object.

Returns The newly created Dummy.

1.1.1.4 pyrep.objects.force_sensor module

class pyrep.objects.force_sensor.**ForceSensor**(*name_or_handle: Union[str, int]*)
Bases: *pyrep.objects.object.Object*

An object able to measure forces and torques that are applied to it.

classmethod create(*sensor_size=0.01*) → pyrep.objects.force_sensor.ForceSensor

read() → Tuple[List[float], List[float]]
Reads the force and torque applied to a force sensor.

Returns A tuple containing the applied forces along the sensor's x, y and z-axes, and the torques along the sensor's x, y and z-axes.

1.1.1.5 pyrep.objects.joint module

class pyrep.objects.joint.**Joint**(*name_or_handle: Union[str, int]*)
Bases: *pyrep.objects.object.Object*

A joint or actuator.

Four types are supported: revolute joints, prismatic joints, screws and spherical joints.

get_joint_force() → float
Retrieves the force or torque applied along/about its active axis.

This function retrieves meaningful information only if the joint is prismatic or revolute, and is dynamically enabled. With the Bullet engine, this function returns the force or torque applied to the joint motor (torques from joint limits are not taken into account). With the ODE and Vortex engine, this function returns the total force or torque applied to a joint along/about its z-axis.

Returns The force or the torque applied to the joint along/about its z-axis.

get_joint_interval() → Tuple[bool, List[float]]
Retrieves the interval parameters of a joint.

Returns A tuple containing a bool indicates whether the joint is cyclic (the joint varies between -pi and +pi in a cyclic manner), and a list containing the interval of the joint. interval[0] is the joint minimum allowed value, interval[1] is the joint range (the maximum allowed value is interval[0]+interval[1]). When the joint is "cyclic", then the interval parameters don't have any meaning.

get_joint_mode() → pyrep.const.JointMode
Retrieves the operation mode of the joint.

Returns The joint mode.

get_joint_position() → float
Retrieves the intrinsic position of a joint.

This function cannot be used with spherical joints.

Returns Intrinsic position of the joint. This is a one-dimensional value: if the joint is revolute, the rotation angle is returned, if the joint is prismatic, the translation amount is returned, etc.

get_joint_target_position() → float
Retrieves the target position of a joint.

Returns Target position of the joint (angular or linear value depending on the joint type).

get_joint_target_velocity() → float
Retrieves the intrinsic target velocity of a non-spherical joint.

Returns Target velocity of the joint (linear or angular velocity depending on the joint-type).

get_joint_type() → pyrep.const.JointType
Retrieves the type of a joint.

Returns The type of the joint.

get_joint_upper_velocity_limit() → float
Gets the joints upper velocity limit.

Returns The upper velocity limit.

get_joint_velocity() → float
Get the current joint velocity.

Returns Velocity of the joint (linear or angular velocity depending on the joint-type).

is_control_loop_enabled() → bool
Gets whether the control loop is enable.

Returns True if the control loop is enabled.

is_motor_enabled() → bool
Gets whether the motor is enable.

Returns True if the motor is enabled.

is_motor_locked_at_zero_velocity() → bool
Gets if the motor is locked when target velocity is zero.

When enabled in velocity mode and its target velocity is zero, then the joint is locked in place.

Returns If the motor will be locked at zero velocity.

set_control_loop_enabled(*value: bool*) → None
Sets whether the control loop is enable.

Parameters **value** – The new value for the control loop state.

set_joint_force(*force: float*) → None
Sets the maximum force or torque that a joint can exert.

The joint will apply that force/torque until the joint target velocity has been reached. To apply a negative force/torque, set a negative target velocity. This function has no effect when the joint is not dynamically enabled, or when it is a spherical joint.

Parameters **force** – The maximum force or torque that the joint can exert. This cannot be a negative value.

set_joint_interval(*cyclic: bool, interval: List[float]*) → None
Sets the interval parameters of a joint (i.e. range values).

The attributes or interval parameters might have no effect, depending on the joint-type.

Parameters

- **cyclic** – Indicates whether the joint is cyclic. Only revolute joints with a pitch of 0 can be cyclic.
- **interval** – Interval of the joint. interval[0] is the joint minimum allowed value, interval[1] is the joint range (i.e. the maximum allowed value is interval[0]+interval[1]).

set_joint_mode(*value: pyrep.const.JointMode*) → None
Sets the operation mode of the joint.

Parameters **value** – The new joint mode value.

set_joint_position(*position: float*, *disable_dynamics: bool = False*) → None
Sets the intrinsic position of the joint.

Parameters

- **disable_dynamics** – If True, then the position can be set even when the joint mode is in Force mode. It will disable dynamics, move the joint, and then re-enable dynamics.
- **position** – Position of a joint (angular or linear values depending on the joint type).

set_joint_target_position(*position: float*) → None
Sets the target position of a joint.

This command makes only sense when the joint is in torque/force mode (also make sure that the joint's motor and position control are enabled).

Parameters **position** – Target position of the joint (angular or linear value depending on the joint type).

set_joint_target_velocity(*velocity: float*) → None
Sets the intrinsic target velocity of a non-spherical joint.

This command makes only sense when the joint mode is torque/force mode: the dynamics functionality and the joint motor have to be enabled (position control should however be disabled).

Parameters **velocity** – Target velocity of the joint (linear or angular velocity depending on the joint-type).

set_motor_enabled(*value: bool*) → None
Sets whether the motor is enable.

Parameters **value** – The new value for the motor state.

set_motor_locked_at_zero_velocity(*value: bool*) → None
Set if the motor is locked when target velocity is zero.

When enabled in velocity mode and its target velocity is zero, then the joint is locked in place.

Parameters **value** – If the motor should be locked at zero velocity.

1.1.1.6 pyrep.objects.object module

class pyrep.objects.object.**Object**(*name_or_handle: Union[str, int]*)
Bases: object

Base class for V-REP scene objects that are used for building a scene.

Objects are visible in the scene hierarchy and in the scene view.

check_collision(*obj: Optional[pyrep.objects.object.Object] = None*) → bool
Checks whether two entities are colliding.

Parameters **obj** – The other collidable object to check collision against, or None to check against all collidable objects. Note that objects must be marked as collidable!

Returns If the object is colliding.

check_distance(*other: pyrep.objects.object.Object*) → float
Checks the minimum distance between two objects.

Parameters **other** – The other object to check distance against.

Returns The distance between the objects.

copy() → pyrep.objects.object.Object
Copy and pastes object in the scene.

The object is copied together with all its associated calculation objects and associated scripts.

Returns The new pasted object.

static exists(*name: str*) → bool
Checks if the given object is in the scene.

Parameters **id** – name/id of object. If the name is appended by a "@alt" suffix, then the object handle based on the object's alternative name will be retrieved.

Returns True of the object exists.

get_bounding_box() → List[float]
Gets the bounding box (relative to the object reference frame).

Returns A list containing the min x, max x, min y, max y, min z, max z positions.

get_bullet_friction() → float
Get bullet friction parameter.

Returns The friction.

get_configuration_tree() → bytes
Retrieves configuration information for a hierarchy tree.

Configuration includes object relative positions/orientations, joint/path values. Calling `PyRep.set_configuration_tree()` at a later time, will restore the object configuration (use this function to temporarily save object positions/orientations/joint/path values).

Returns The configuration tree.

get_contact(*contact_obj=None*, *get_contact_normal: bool = True*) → List[T]
Get the contact point and force with other object

Parameters

- **contact_obj** – The object want to check contact info with, set to None to get contact with all objects
- **get_contact_normal** – Weather get the force and direction

Returns a list of all the contact info

get_explicit_handling() → int
Get explicit handling flags.

Returns The flag: enabled(1) or disabled(0).

get_extension_string() → str
A string that describes additional environment/object properties.

Returns The extension string.

get_handle() → int
Gets the internal handle of this object.

Returns The internal handle.

get_matrix(*relative_to=None*) → numpy.ndarray
Retrieves the transformation matrix of this object.

Parameters **relative_to** – Indicates relative to which reference frame we want the matrix. Specify None to retrieve the absolute transformation matrix, or an Object relative to whose reference frame we want the transformation matrix.

Returns A 4x4 transformation matrix.

get_model_bounding_box() → List[float]
Gets the models bounding box (relative to models reference frame).

Raises ObjectIsNotModel if the object is not a model.

Returns A list containing the min x, max x, min y, max y, min z, max z positions.

get_name() → str
Gets the objects name in the scene.

Returns The objects name.

static get_object(*name_or_handle: str*) → pyrep.objects.object.Object
Gets object retrieved by name.

Returns The object.

static get_object_name(*name_or_handle: Union[str, int]*) → str
Gets object name.

Returns Object name.

static get_object_type(*name: str*) → pyrep.const.ObjectType
Gets the type of the object.

Returns Type of the object.

get_objects_in_tree(**args*, ***kwargs*) → List[pyrep.objects.object.Object]
Retrieves the objects in a given hierarchy tree.

Parameters

- **object_type** – The object type to retrieve. One of *ObjectType*.
- **exclude_base** – Exclude the tree base from the returned list.
- **first_generation_only** – Include in the returned list only the object's first children. Otherwise, entire hierarchy is returned.

Returns A list of objects in the hierarchy tree.

get_orientation(*relative_to=None*) → numpy.ndarray
Gets the orientation of this object.

Parameters **relative_to** – Indicates relative to which reference frame we want the orientation. Specify None to retrieve the absolute orientation, or an Object relative to whose reference frame we want the orientation.

Returns A list containing the x, y, z orientation of the object (in radians).

get_parent() → Optional[pyrep.objects.object.Object]
Gets the parent of this object in the scene hierarchy.

Returns The parent of this object, or None if it doesn't have a parent.

get_pose(*relative_to=None*) → numpy.ndarray
Retrieves the position and quaternion of an object

Parameters **relative_to** – Indicates relative to which reference frame we want the pose. Specify None to retrieve the absolute pose, or an Object relative to whose reference frame we want the pose.

Returns An array containing the (X,Y,Z,Qx,Qy,Qz,Qw) pose of the object.

get_position(*relative_to=None*) → numpy.ndarray
Gets the position of this object.

Parameters **relative_to** – Indicates relative to which reference frame we want the position. Specify None to retrieve the absolute position, or an Object relative to whose reference frame we want the position.

Returns An array containing the x, y, z position of the object.

get_quaternion(*relative_to=None*) → numpy.ndarray
Retrieves the quaternion (x,y,z,w) of an object.

Parameters **relative_to** – Indicates relative to which reference frame we want the orientation. Specify None to retrieve the absolute orientation, or an Object relative to whose reference frame we want the orientation.

Returns A list containing the quaternion (x,y,z,w).

get_type() → pyrep.const.ObjectType
Gets the type of the object.

Returns Type of the object.

get_velocity() → Tuple[numpy.ndarray, numpy.ndarray]
Get the velocity of this object.

Returns A pair of linear and angular velocity.

is_collidable() → bool
Whether the object is collidable or not.

Returns If the object is collidable.

is_detectable() → bool
Whether the object is detectable or not.

Returns If the object is detectable.

is_measurable() → bool
Whether the object is measurable or not.

Returns If the object is measurable.

is_model() → bool
Whether the object is a model or not.

Returns If the object is a model.

is_model_collidable() → bool
Whether the model is collidable or not.

Raises ObjectIsNotModel if the object is not a model.

Returns If the model is collidable.

is_model_detectable() → bool
Whether the model is detectable or not.

Raises ObjectIsNotModel if the object is not a model.

Returns If the model is detectable.

is_model_dynamic() → bool
Whether the model is dynamic or not.

Raises ObjectIsNotModel if the object is not a model.

Returns If the model is dynamic.

is_model_measurable() → bool
Whether the model is measurable or not.

Raises ObjectIsNotModel if the object is not a model.

Returns If the model is measurable.

is_model_renderable() → bool
Whether the model is renderable or not.

Raises ObjectIsNotModel if the object is not a model.

Returns If the model is renderable.

is_model_respondable() → bool
Whether the model is respondable or not.

Raises ObjectIsNotModel if the object is not a model.

Returns If the model is respondable.

is_renderable() → bool
Whether the object is renderable or not.

Returns If the object is renderable.

remove() → None
Removes this object/model from the scene.

Raises ObjectAlreadyRemoved if the object is no longer on the scene.

reset_dynamic_object() → None
Dynamically resets an object that is dynamically simulated.

This means that the object representation in the dynamics engine is removed, and added again. This can be useful when the set-up of a dynamically simulated chain needs to be modified during simulation (e.g. joint or shape attachement position/orientation changed). It should be noted that calling this on a dynamically simulated object might slightly change its position/orientation relative to its parent (since the object will be disconnected from the dynamics world in its current position/orientation), so the user is in charge of rectifying for that.

rotate(*rotation: List[float]*) → None
Rotates a transformation matrix.

Parameters **rotation** – The x, y, z rotation to perform (in radians).

save_model(*path: str*) → None
Saves a model.

Object can be turned to models via *Object.set_model()*. Any existing file with same name will be overwritten.

Parameters **path** – model filename. The filename extension is required ("ttm").

Raises ObjectIsNotModel if the object is not a model.

set_bullet_friction(*friction*) → None
Set bullet friction parameter.

Parameters **friction** – The friction to set.

set_collidable(*value: bool*) → None
Set whether the object is collidable or not.

Parameters **value** – The new value of the collidable state.

set_detectable(*value: bool*)
Set whether the object is detectable or not.

Parameters **value** – The new value of the detectable state.

set_explicit_handling(*value: int*) → None
Set explicit handling flags.

Parameters **value** – A flag to enable(1) or disable(0) explicit handling.

set_matrix(*matrix: numpy.ndarray*, *relative_to=None*) → None
Sets the transformation matrix of this object.

Parameters

- **relative_to** – Indicates relative to which reference frame the matrix is specified. Specify None to set the absolute transformation matrix, or an Object relative to whose reference frame the transformation matrix is specified.
- **matrix** – A 4x4 transformation matrix.

set_measurable(*value: bool*)
Set whether the object is measurable or not.

Parameters **value** – The new value of the measurable state.

set_model(*value: bool*)
Set whether the object is a model or not.

Parameters **value** – True to set as a model.

set_model_collidable(*value: bool*)
Set whether the model is collidable or not.

Parameters **value** – The new value of the collidable state of the model.

Raises ObjectIsNotModel if the object is not a model.

set_model_detectable(*value: bool*)
Set whether the model is detectable or not.

Parameters **value** – The new value of the detectable state of the model.

Raises ObjectIsNotModel if the object is not a model.

set_model_dynamic(*value: bool*)
Set whether the model is dynamic or not.

Parameters **value** – The new value of the dynamic state of the model.

Raises ObjectIsNotModel if the object is not a model.

set_model_measurable(*value: bool*)
Set whether the model is measurable or not.

Parameters **value** – The new value of the measurable state of the model.

Raises ObjectIsNotModel if the object is not a model.

set_model_renderable(*value: bool*)
Set whether the model is renderable or not.

Parameters **value** – The new value of the renderable state of the model.

Raises ObjectIsNotModel if the object is not a model.

set_model_respondable(*value: bool*)
Set whether the model is respondable or not.

Parameters **value** – The new value of the respondable state of the model.

Raises ObjectIsNotModel if the object is not a model.

set_name(*name: str*) → None
Sets the objects name in the scene.

set_orientation(*orientation: Union[list, numpy.ndarray], relative_to=None, reset_dynamics=True*) → None
Sets the orientation of this object.

Parameters

- **orientation** – An array containing the x, y, z orientation of the object (in radians).
- **relative_to** – Indicates relative to which reference frame the the orientation is specified. Specify None to set the absolute orientation, or an Object relative to whose reference frame the orientation is specified.
- **reset_dynamics** – If we want to reset the dynamics when rotating an object instantaneously.

set_parent(*parent_object: Optional[Object], keep_in_place=True*) → None
Sets this objects parent object in the scene hierarchy.

Parameters

- **parent_object** – The object that will become parent, or None if the object should become parentless.
- **keep_in_place** – Indicates whether the object's absolute position and orientation should stay same

set_pose(*pose: Union[list, numpy.ndarray], relative_to=None, reset_dynamics=True*) → None
Sets the position and quaternion of an object.

Parameters

- **pose** – An array containing the (X,Y,Z,Qx,Qy,Qz,Qw) pose of the object.
- **relative_to** – Indicates relative to which reference frame the the pose is specified. Specify None to set the absolute pose, or an Object relative to whose reference frame the pose is specified.
- **reset_dynamics** – If we want to reset the dynamics when rotating an object instantaneously.

set_position(*position: Union[list, numpy.ndarray], relative_to=None, reset_dynamics=True*) → None
Sets the position of this object.

Parameters

- **position** – A list containing the x, y, z position of the object.
- **relative_to** – Indicates relative to which reference frame the the position is specified. Specify None to set the absolute position, or an Object relative to whose reference frame the position is specified.
- **reset_dynamics** – If we want to reset the dynamics when moving an object instantaneously.

set_quaternion(*quaternion: Union[list, numpy.ndarray], relative_to=None, reset_dynamics=True*) → None
Sets the orientation of this object.

If the quaternion is not normalised, it will be normalised for you.

Parameters

- **quaternion** – An array containing the quaternion (x,y,z,w).
- **relative_to** – Indicates relative to which reference frame the the orientation is specified. Specify None to set the absolute orientation, or an Object relative to whose reference frame the orientation is specified.
- **reset_dynamics** – If we want to reset the dynamics when rotating an object instantaneously.

set_renderable(*value: bool*)
Set whether the object is renderable or not.

Parameters **value** – The new value of the renderable state.

still_exists() → bool
Gets whether this object is still in the scene or not.

Returns Whether the object exists or not.

1.1.1.7 pyrep.objects.proximity_sensor module

class pyrep.objects.proximity_sensor.**ProximitySensor**(*name_or_handle: Union[str, int]*)
Bases: *pyrep.objects.object.Object*

Detects objects within a detection volume.

V-REP supports pyramid-, cylinder-, disk-, cone- and ray-type proximity sensors.

is_detected(*obj: pyrep.objects.object.Object*) → bool
Checks whether the proximity sensor detects the indicated object.

Parameters **obj** – The object to detect.

Returns Bool indicating if the object was detected.

read() → float
Read the distance between sensor and first detected object. If there is no detected object returns -1.0. It can be considered as maximum measurable distance of the sensor.

Returns Float distance to the first detected object

1.1.1.8 pyrep.objects.shape module

class pyrep.objects.shape.**SShapeVizInfo**(*vertices, indices, normals, shading_angle, colors, texture, texture_id, texture_coords, texture_apply_mode, texture_options*)
Bases: tuple

colors
Alias for field number 4

indices
Alias for field number 1

normals
Alias for field number 2

shading_angle
Alias for field number 3

texture
Alias for field number 5

texture_apply_mode
Alias for field number 8

texture_coords
Alias for field number 7

texture_id
Alias for field number 6

texture_options
Alias for field number 9

vertices
Alias for field number 0

class pyrep.objects.shape.**Shape**(*name_or_handle: Union[str, int]*)
Bases: *pyrep.objects.object.Object*

Shapes are rigid mesh objects that are composed of triangular faces.

add_force(*position: numpy.ndarray*, *force: numpy.ndarray*, *reset_force_torque: bool = False*) → None
Adds a non-central force to a shape object that is dynamically enabled. Added forces are cumulative.

Parameters

- **position** – Relative position where the force should be applied.
- **force** – The force (in relative coordinates) to add.
- **reset_force_torque** – Clears the accumulated force and torque.

add_force_and_torque(*force: numpy.ndarray*, *torque: numpy.ndarray*, *reset_force: bool = False*, *reset_torque: bool = False*) → None
Adds a force and/or torque to a shape object that is dynamically enabled. Forces are applied at the center of mass. Added forces and torques are cumulative.

Parameters

- **force** – The force (in absolute coordinates) to add.
- **torque** – The torque (in absolute coordinates) to add.
- **reset_force** – Clears the accumulated force.
- **reset_torque** – Clears the accumulated torque.

apply_texture(*texture_coords: numpy.ndarray*, *texture: numpy.ndarray*, *interpolate: bool = True*, *decal_mode: bool = False*, *is_rgba: bool = False*, *fliph: bool = False*, *flipv: bool = False*) → None
Apply texture to the shape.

Parameters

- **texture_coords** – A list of (u, v) values that indicate the vertex position on the shape. For each of the shape's triangle, there should be exactly 3 UV texture coordinate pairs
- **texture** – The RGB or RGBA texture.

- **interpolate** – A flag to interpolate adjacent texture pixels.
- **decal_mode** – Texture is applied as a decal (its appearance won't be influenced by light conditions).
- **is_rgba** – A flag to use RGBA texture.
- **fliph** – A flag to flip texture horizontally.
- **flipv** – A flag to flip texture vertically. Note that CoppeliaSim texture coordinates are flipped vertically compared with Pillow and OpenCV and this flag must be true in general.

compute_mass_and_inertia(*density: float*) → None

Computes and applies the mass and inertia properties for a convex shape (or convex compound shape), based on a density value.

Parameters **density** – The density expressed in kg/m^3

static create(*type: pyrep.const.PrimitiveShape, size: List[float], mass=1.0, backface_culling=False, visible_edges=False, smooth=False, respondable=True, static=False, renderable=True, position=None, orientation=None, color=None*) → pyrep.objects.shape.Shape

Creates a primitive shape in the scene.

Parameters

- **type** – The type of primitive to shape. One of: PrimitiveShape.CUBOID PrimitiveShape.SPHERE PrimitiveShape.CYLINDER PrimitiveShape.CONE
- **size** – A list of the x, y, z dimensions.
- **mass** – A float representing the mass of the object.
- **backface_culling** – If backface culling is enabled.
- **visible_edges** – If the object will have visible edges.
- **smooth** – If the shape appears smooth.
- **respondable** – Shape is responsible.
- **static** – If the shape is static.
- **renderable** – If the shape is renderable.
- **position** – The x, y, z position.
- **orientation** – The x, y, z orientation (in radians).
- **color** – The r, g, b values of the shape.

Returns The created Shape object.

static create_mesh(*vertices: List[float], indices: List[int], shading_angle=None, backface_culling=False, visible_edges=False*) → pyrep.objects.shape.Shape

Creates a mesh shape.

Parameters

- **vertices** – A list of vertices.
- **indices** – A list of indices.
- **shading_angle** – The shading angle (in radians).
- **backface_culling** – To enable backface culling.
- **visible_edges** – To enable visible edges.

Returns The newly created mesh.

decimate_mesh(*percentage: float*) → pyrep.objects.shape.Shape
Retrieves a shape's mesh information.

Parameters **percentage** – The percentage of the desired decimation (0.1-0.9).

Returns A new shape that has a decimated mesh.

get_color() → List[float]
Gets the shape color.

Returns The r, g, b values of the shape.

get_convex_decomposition(*morph=False*, *same=False*, *use_vhacd=False*, *individual_meshes=False*, *hacd_extra_points=True*, *hacd_face_points=True*, *hacd_min_clusters=1*, *hacd_tri_target=500*, *hacd_max_vertex=200*, *hacd_max_iter=4*, *hacd_max_concavity=100*, *hacd_max_dist=30*, *hacd_cluster_thresh=0.25*, *vhacd_pca=False*, *vhacd_tetrahedron=False*, *vhacd_res=100000*, *vhacd_depth=20*, *vhacd_plane_downsample=4*, *vhacd_hull_downsample=4*, *vhacd_max_vertex=64*, *vhacd_concavity=0.0025*, *vhacd_alpha=0.05*, *vhacd_beta=0.05*, *vhacd_gamma=0.00125*, *vhacd_min_vol=0.0001*) → pyrep.objects.shape.Shape

Compute the convex decomposition of the shape using HACD or V-HACD algorithms

Parameters

- **morph** – The shape will be morphed into its convex decomposition. Otherwise a new shape will be created.
- **same** – Use the same parameters as the last call to the function.
- **use_vhacd** – Use V-HACD algorithm.
- **individual_meshes** – Each individual mesh of a compound shape will be handled on its own during decomposition, otherwise the compound shape is considered as a single mesh.
- **hacd_extra_points** – HACD: Extra points will be added when computing the concavity.
- **hacd_face_points** – HACD: Faces points will be added when computing the concavity.
- **hacd_min_clusters** – HACD: Minimum number of clusters to generate.
- **hacd_tri_target** – HACD: Targeted number of triangles of the decimated mesh.
- **hacd_max_vertex** – HACD: Maximum number of vertices for each generated convex hull.
- **hacd_max_iter** – HACD: Maximum number of iterations.
- **hacd_max_concavity** – HACD: The maximum allowed concavity.
- **hacd_max_dist** – HACD: The maximum allowed distance to get convex clusters connected.
- **hacd_cluster_thresh** – HACD: The threshold to detect small clusters, expressed as a fraction of the total mesh surface.
- **vhacd_pca** – V-HACD: Enable PCA.

- **vhacd_tetrahedron** – V-HACD: Tetrahedron-based approximate convex decomposition. Otherwise, voxel-based decomposition is used.
- **vhacd_res** – V-HACD: Resolution (10000-64000000)
- **vhacd_depth** – V-HACD: Depth (1-32)
- **vhacd_plane_downsample** – V-HACD: Plane downsampling (1-16)
- **vhacd_hull_downsample** – V-HACD: Convex hull downsampling (1-16)
- **vhacd_max_vertex** – V-HACD: Maximum number of vertices per convex hull (4-1024)
- **vhacd_concavity** – V-HACD: Concavity (0.0-1.0)
- **vhacd_alpha** – V-HACD: Alpha (0.0-1.0)
- **vhacd_beta** – V-HACD: Beta (0.0-1.0)
- **vhacd_gamma** – V-HACD: Gamma (0.0-1.0)
- **vhacd_min_vol** – V-HACD: Minimum volume per convex hull (0.0-0.01)

Returns Convex Decomposition of the shape.

get_mass() → float
Gets the mass of the shape.

Returns A float representing the mass.

get_mesh_data() → Tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray]
Retrieves a shape's mesh information.

Parameters **asnumpy** – A flag to cast vertices as numpy array with reshape.

Returns A tuple containing a list of vertices, indices, and normals.

get_shape_viz(*index*)
Retrieves a shape's visual information.

Parameters **index** – 0-based index of the shape element to retrieve (compound shapes contain more than one shape element)

Returns SShapeVizInfo.

get_texture()
Retrieves the texture from the shape. :return: The texture associated with this object.

get_transparency() → float
Sets the transparency of the shape.

Returns The transparency values of the shape.

static import_mesh(*filename: str*, *scaling_factor=1.0*, *keep_identical_vertices=False*, *ignore_up_vector=False*) → pyrep.objects.shape.Shape
Imports a mesh from a file.

Parameters

- **filename** – The location of the file to import.
- **scaling_factor** – The scaling factor to apply to the imported vertices
- **keep_identical_vertices** – Keep identical vertices.
- **ignore_up_vector** – Ignore up-vector coded in file.

Returns The grouped Shape object.

classmethod import_shape(*filename: str*, *scaling_factor=1.0*, *keep_identical_vertices=False*, *ignore_color=False*, *ignore_texture=False*, *reorient_bounding_box=False*, *ignore_up_vector=False*) → pyrep.objects.shape.Shape

Imports a shape with visuals from a file.

Parameters

- **filename** – The location of the file to import.
- **scaling_factor** – The scaling factor to apply to the imported vertices
- **keep_identical_vertices** – Keep identical vertices.
- **ignore_color** – Do not preserve colors.
- **ignore_texture** – Do not preserve texture.
- **reorient_bounding_box** – Reorient the shape's bounding box with the world.
- **ignore_up_vector** – Ignore up-vector coded in file.

Returns The Shape object.

is_dynamic() → bool

Whether the shape is dynamic or not.

Returns If the shape is dynamic.

is_respondable() → bool

Whether the shape is respondable or not.

Returns If the shape is respondable.

remove_texture()

Removes the texture from the shape.

reorient_bounding_box(*relative_to=None*) → None

set_color(*color: List[float]*) → None

Sets the color of the shape.

Parameters **color** – The r, g, b values of the shape.

set_dynamic(*value: bool*) → None

Set whether the shape is dynamic or not.

Parameters **value** – The new value of the dynamic state of the shape.

set_mass(*mass: float*) → None

Sets the mass of the shape.

Parameters **mass** – The new mass value.

set_respondable(*value: bool*) → None

Set whether the shape is respondable or not.

Parameters **value** – The new value of the respondable state of the shape.

set_texture(*texture: pyrep.textures.texture.Texture*, *mapping_mode: pyrep.const.TextureMappingMode*, *interpolate=True*, *decal_mode=False*, *repeat_along_u=False*, *repeat_along_v=False*, *uv_scaling=[1.0, 1.0]*, *position: List[float] = None*, *orientation: List[float] = None*)

Applies a texture to a shape

Parameters

- **texture** – The texture to add.
- **mapping_mode** – The texture mapping mode. One of: TextureMappingMode.PLANE TextureMappingMode.CYLINDER TextureMappingMode.SPHERE TextureMappingMode.CUBE
- **interpolate** – Adjacent texture pixels are not interpolated.
- **decal_mode** – Texture is applied as a decal (its appearance won't be influenced by light conditions).
- **repeat_along_u** – Texture will be repeated along the U direction.
- **repeat_along_v** – Texture will be repeated along the V direction.
- **uv_scaling** – A list of 2 values containig the texture scaling factors along the U and V directions.
- **position** – A list of (x,y,z) values that indicate the texture position on the shape. Can be None for default.
- **orientation** – A list of 3 Euler angles that indicate the texture orientation on the shape. Can be None for default.

set_transparency(*value: float*) → None
Sets the transparency of the shape.

Parameters **value** – Value between 0 and 1.

ungroup() → List[pyrep.objects.shape.Shape]
Ungroups a compound shape into several simple shapes.

Returns A list of shapes.

1.1.1.9 pyrep.objects.vision_sensor module

class pyrep.objects.vision_sensor.**VisionSensor**(*name_or_handle: Union[str, int]*)
Bases: *pyrep.objects.object.Object*

A camera-type sensor, reacting to light, colors and images.

capture_depth(*in_meters=False*) → numpy.ndarray
Retrieves the depth-image of a vision sensor.

Parameters **in_meters** – Whether the depth should be returned in meters.

Returns A numpy array of size (width, height)

capture_pointcloud() → numpy.ndarray
Retrieves point cloud in word frame.

Returns A numpy array of size (width, height, 3)

capture_rgb() → numpy.ndarray
Retrieves the rgb-image of a vision sensor.

Returns A numpy array of size (width, height, 3)

static create(*resolution: List[int], explicit_handling=False, perspective_mode=True, show_volume_not_detecting=True, show_volume_detecting=True, passive=False, use_local_lights=False, show_fog=True, near_clipping_plane=0.01, far_clipping_plane=10.0, view_angle=60.0, ortho_size=1.0, sensor_size=None, render_mode=<RenderMode.OPENGL3: 7>, position=None, orientation=None*) → pyrep.objects.vision_sensor.VisionSensor

Create a Vision Sensor

Parameters

- **resolution** – List of the [x, y] resolution.
- **explicit_handling** – Sensor will be explicitly handled.
- **perspective_mode** – Sensor will be operated in Perspective Mode. Orthographic mode if False.
- **show_volume_not_detecting** – Sensor volume will be shown when not detecting anything.
- **show_volume_detecting** – Sensor will be shown when detecting.
- **passive** – Sensor will be passive (use an external image).
- **use_local_lights** – Sensor will use local lights.
- **show_fog** – Sensor will show fog (if enabled).
- **near_clipping_plane** – Near clipping plane.
- **far_clipping_plane** – Far clipping plane.
- **view_angle** – Perspective angle (in degrees) if in Perspective Mode.
- **ortho_size** – Orthographic projection size [m] if in Orthographic Mode.
- **sensor_size** – Size [x, y, z] of the Vision Sensor object.
- **render_mode** – Sensor rendering mode, one of: RenderMode.OPENGL RenderMode.OPENGL_AUXILIARY RenderMode.OPENGL_COLOR_CODED RenderMode.POV_RAY RenderMode.EXTERNAL RenderMode.EXTERNAL_WINDOWED RenderMode.OPENGL3 RenderMode.OPENGL3_WINDOWED
- **position** – The [x, y, z] position, if specified.
- **orientation** – The [x, y, z] orientation in radians, if specified.

Returns The created Vision Sensor.

get_entity_to_render() → None

Get the entity to render to the Sensor, this can be an object or more usefully a collection. -1 if all objects in scene are rendered.

Returns Handle of the entity to render

get_far_clipping_plane() → float

Get the Sensor's far clipping plane.

Returns Near clipping plane (metres)

get_intrinsic_matrix()

get_near_clipping_plane() → float

Get the Sensor's near clipping plane.

Returns Near clipping plane (metres)

get_orthographic_size() → float
Get the Sensor's orthographic size.

Returns The sensor's orthographic size (in metres).

get_perspective_angle() → float
Get the Sensor's perspective angle.

Returns The sensor's perspective angle (in degrees).

get_perspective_mode() → pyrep.const.PerspectiveMode
Retrieve the Sensor's perspective mode.

Returns The current PerspectiveMode.

get_render_mode() → pyrep.const.RenderMode
Retrieves the Sensor's rendering mode

Returns RenderMode for the current rendering mode.

get_resolution() → List[int]
Return the Sensor's resolution.

Returns Resolution [x, y]

get_windowed_size() → Sequence[int]
Get the size of windowed rendering.

Returns The (x, y) resolution of the window. 0 for full-screen.

handle_explicitly() → None
Handle sensor explicitly.

This enables capturing image (e.g., capture_rgb()) without PyRep.step().

pointcloud_from_depth(*depth: numpy.ndarray*) → numpy.ndarray
Converts depth (in meters) to point cloud in word frame.

Returns A numpy array of size (width, height, 3)

static pointcloud_from_depth_and_camera_params(*depth: numpy.ndarray*, *extrinsics: numpy.ndarray*, *intrinsics: numpy.ndarray*) → numpy.ndarray
Converts depth (in meters) to point cloud in word frame. :return: A numpy array of size (width, height, 3)

set_entity_to_render(*entity_to_render: int*) → None
Set the entity to render to the Sensor, this can be an object or more usefully a collection. -1 to render all objects in scene.

Parameters **entity_to_render** – Handle of the entity to render

set_far_clipping_plane(*far_clipping: float*) → None
Set the Sensor's far clipping plane.

Parameters **far_clipping** – New far clipping plane (in metres)

set_near_clipping_plane(*near_clipping: float*) → None
Set the Sensor's near clipping plane.

Parameters **near_clipping** – New near clipping plane (in metres)

set_orthographic_size(*ortho_size: float*) → None
Set the Sensor's orthographic size.

Parameters **angle** – New orthographic size (in metres)

set_perspective_angle(*angle: float*) → None
Set the Sensor's perspective angle.

Parameters **angle** – New perspective angle (in degrees)

set_perspective_mode(*perspective_mode: pyrep.const.PerspectiveMode*) → None
Set the Sensor's perspective mode.

Parameters **perspective_mode** – The new perspective mode, one of: PerspectiveMode.ORTHOGRAPHIC PerspectiveMode.PERSPECTIVE

set_render_mode(*render_mode: pyrep.const.RenderMode*) → None
Set the Sensor's rendering mode

Parameters **render_mode** – The new sensor rendering mode, one of: RenderMode.OPENGL RenderMode.OPENGL_AUXILIARY RenderMode.OPENGL_COLOR_CODED RenderMode.POV_RAY RenderMode.EXTERNAL RenderMode.EXTERNAL_WINDOWED RenderMode.OPENGL3 RenderMode.OPENGL3_WINDOWED

set_resolution(*resolution: List[int]*) → None
Set the Sensor's resolution.

Parameters **resolution** – New resolution [x, y]

set_windowed_size(*resolution: Sequence[int] = (0, 0)*) → None
Set the size of windowed rendering.

Parameters **resolution** – The (x, y) resolution of the window. 0 for full-screen.

1.1.1.10 Module contents

1.1.2 pyrep.robots package

1.1.2.1 Subpackages

pyrep.robots.arms package

Submodules

pyrep.robots.arms.arm module

class pyrep.robots.arms.arm.**Arm**(*count: int*, *name: str*, *num_joints: int*, *base_name: str = None*, *max_velocity=1.0*, *max_acceleration=4.0*, *max_jerk=1000*)
Bases: *pyrep.robots.robot_component.RobotComponent*

Base class representing a robot arm with path planning support.

check_arm_collision(*obj: Optional[pyrep.objects.object.Object] = None*) → bool
Checks whether two entities are colliding.

Parameters **obj** – The other collidable object to check collision against, or None to check against all collidable objects. Note that objects must be marked as collidable!

Returns If the object is colliding.

get_configs_for_tip_pose(*position: Union[List[float], numpy.ndarray], euler: Union[List[float], numpy.ndarray] = None, quaternion: Union[List[float], numpy.ndarray] = None, ignore_collisions=False, trials=300, max_configs=60, relative_to: pyrep.objects.object.Object = None*) → List[List[float]]

Gets a valid joint configuration for a desired end effector pose. Must specify either rotation in euler or quaternions, but not both! :param position: The x, y, z position of the target. :param euler: The x, y, z orientation of the target (in radians). :param quaternion: A list containing the quaternion (x,y,z,w). :param ignore_collisions: If collision checking should be disabled. :param trials: The maximum number of attempts to reach max_configs :param max_configs: The maximum number of configurations we want to

generate before ranking them.

Parameters **relative_to** – Indicates relative to which reference frame we want

the target pose. Specify None to retrieve the absolute pose, or an Object relative to whose reference frame we want the pose. :raises: ConfigurationError if no joint configuration could be found. :return: A list of valid joint configurations for the desired end effector pose.

get_jacobian()

Calculates the Jacobian.

Returns the row-major Jacobian matix.

get_linear_path(*position: Union[List[float], numpy.ndarray], euler: Union[List[float], numpy.ndarray] = None, quaternion: Union[List[float], numpy.ndarray] = None, steps=50, ignore_collisions=False, relative_to: pyrep.objects.object.Object = None*) → pyrep.robots.configuration_paths.arm_configuration_path.ArmConfigurationPath

Gets a linear configuration path given a target pose.

Generates a path that drives a robot from its current configuration to its target dummy in a straight line (i.e. shortest path in Cartesian space).

Must specify either rotation in euler or quaternions, but not both!

Parameters

- **position** – The x, y, z position of the target.
- **euler** – The x, y, z orientation of the target (in radians).
- **quaternion** – A list containing the quaternion (x,y,z,w).
- **steps** – The desired number of path points. Each path point contains a robot configuration. A minimum of two path points is required. If the target pose distance is large, a larger number of steps leads to better results for this function.
- **ignore_collisions** – If collision checking should be disabled.
- **relative_to** – Indicates relative to which reference frame we want

the target pose. Specify None to retrieve the absolute pose, or an Object relative to whose reference frame we want the pose. :raises: ConfigurationPathError if no path could be created.

Returns A linear path in the arm configuration space.

get_nonlinear_path(*position: Union[List[float], numpy.ndarray], euler: Union[List[float], numpy.ndarray] = None, quaternion: Union[List[float], numpy.ndarray] = None, ignore_collisions=False, trials=300, max_configs=1, distance_threshold: float = 0.65, max_time_ms: int = 10, trials_per_goal=1, algorithm=<ConfigurationPathAlgorithms.SBL: 'SBL'>, relative_to: pyrep.objects.object.Object = None*) → pyrep.robots.configuration_paths.arm_configuration_path.ArmConfigurationPath

Gets a non-linear (planned) configuration path given a target pose.

A path is generated by finding several configs for a pose, and ranking them according to the distance in configuration space (smaller is better).

Must specify either rotation in euler or quaternions, but not both!

Parameters

- **position** – The x, y, z position of the target.
- **euler** – The x, y, z orientation of the target (in radians).
- **quaternion** – A list containing the quaternion (x,y,z,w).
- **ignore_collisions** – If collision checking should be disabled.
- **trials** – The maximum number of attempts to reach max_configs. See 'solve_ik_via_sampling'.
- **max_configs** – The maximum number of configurations we want to generate before sorting them. See 'solve_ik_via_sampling'.
- **distance_threshold** – Distance indicating when IK should be computed in order to try to bring the tip onto the target. See 'solve_ik_via_sampling'.
- **max_time_ms** – Maximum time in ms spend searching for each configuation. See 'solve_ik_via_sampling'.
- **trials_per_goal** – The number of paths per config we want to trial.
- **algorithm** – The algorithm for path planning to use.
- **relative_to** – Indicates relative to which reference frame we want

the target pose. Specify None to retrieve the absolute pose, or an Object relative to whose reference frame we want the pose. :raises: ConfigurationPathError if no path could be created.

Returns A non-linear path in the arm configuration space.

get_path(*position: Union[List[float], numpy.ndarray], euler: Union[List[float], numpy.ndarray] = None, quaternion: Union[List[float], numpy.ndarray] = None, ignore_collisions=False, trials=300, max_configs=1, distance_threshold: float = 0.65, max_time_ms: int = 10, trials_per_goal=1, algorithm=<ConfigurationPathAlgorithms.SBL: 'SBL'>, relative_to: pyrep.objects.object.Object = None*) → pyrep.robots.configuration_paths.arm_configuration_path.ArmConfigurationPath

Tries to get a linear path, failing that tries a non-linear path.

Must specify either rotation in euler or quaternions, but not both!

Parameters

- **position** – The x, y, z position of the target.
- **euler** – The x, y, z orientation of the target (in radians).
- **quaternion** – A list containing the quaternion (x,y,z,w).
- **ignore_collisions** – If collision checking should be disabled.

- **trials** – The maximum number of attempts to reach max_configs. See 'solve_ik_via_sampling'.
- **max_configs** – The maximum number of configurations we want to generate before sorting them. See 'solve_ik_via_sampling'.
- **distance_threshold** – Distance indicating when IK should be computed in order to try to bring the tip onto the target. See 'solve_ik_via_sampling'.
- **max_time_ms** – Maximum time in ms spend searching for each configuation. See 'solve_ik_via_sampling'.
- **trials_per_goal** – The number of paths per config we want to trial.
- **algorithm** – The algorithm for path planning to use.
- **relative_to** – Indicates relative to which reference frame we want

the target pose. Specify None to retrieve the absolute pose, or an Object relative to whose reference frame we want the pose.

Raises ConfigurationPathError if neither a linear or non-linear path can be created.

Returns A linear or non-linear path in the arm configuration space.

get_path_from_cartesian_path(*path: pyrep.objects.cartesian_path.CartesianPath*) → pyrep.robots.configuration_paths.arm_configuration_path.ArmConfigurationPath

Translate a path from cartesian space, to arm configuration space.

Note: It must be possible to reach the start of the path via a linear path, otherwise an error will be raised.

Parameters **path** – A `CartesianPath` instance to be translated to a configuration-space path.

Raises ConfigurationPathError if no path could be created.

Returns A path in the arm configuration space.

get_tip() → pyrep.objects.dummy.Dummy

Gets the tip of the arm.

Each arm is required to have a tip for path planning.

Returns The tip of the arm.

set_ik_element_properties(*constraint_x=True*, *constraint_y=True*, *constraint_z=True*, *constraint_alpha_beta=True*, *constraint_gamma=True*) → None

set_ik_group_properties(*resolution_method='pseudo_inverse'*, *max_iterations=6*, *dls_damping=0.1*) → None

solve_ik(*position: Union[List[float], numpy.ndarray], euler: Union[List[float], numpy.ndarray] = None, quaternion: Union[List[float], numpy.ndarray] = None, relative_to: pyrep.objects.object.Object = None*) → List[float]

Solves an IK group and returns the calculated joint values.

Must specify either rotation in euler or quaternions, but not both!

Parameters

- **position** – The x, y, z position of the target.
- **euler** – The x, y, z orientation of the target (in radians).
- **quaternion** – A list containing the quaternion (x,y,z,w).
- **relative_to** – Indicates relative to which reference frame we want

the target pose. Specify None to retrieve the absolute pose, or an Object relative to whose reference frame we want the pose. :return: A list containing the calculated joint values.

solve_ik_via_jacobian(*position: Union[List[float], numpy.ndarray], euler: Union[List[float], numpy.ndarray] = None, quaternion: Union[List[float], numpy.ndarray] = None, relative_to: pyrep.objects.object.Object = None*) → List[float]

Solves an IK group and returns the calculated joint values.

This IK method performs a linearisation around the current robot configuration via the Jacobian. The linearisation is valid when the start and goal pose are not too far away, but after a certain point, linearisation will no longer be valid. In that case, the user is better off using 'solve_ik_via_sampling'.

Must specify either rotation in euler or quaternions, but not both!

Parameters

- **position** – The x, y, z position of the target.
- **euler** – The x, y, z orientation of the target (in radians).
- **quaternion** – A list containing the quaternion (x,y,z,w).
- **relative_to** – Indicates relative to which reference frame we want

the target pose. Specify None to retrieve the absolute pose, or an Object relative to whose reference frame we want the pose. :return: A list containing the calculated joint values.

solve_ik_via_sampling(*position: Union[List[float], numpy.ndarray], euler: Union[List[float], numpy.ndarray] = None, quaternion: Union[List[float], numpy.ndarray] = None, ignore_collisions: bool = False, trials: int = 300, max_configs: int = 1, distance_threshold: float = 0.65, max_time_ms: int = 10, relative_to: pyrep.objects.object.Object = None*) → numpy.ndarray

Solves an IK group and returns the calculated joint values.

This IK method performs a random searches for manipulator configurations that matches the given end-effector pose in space. When the tip pose is close enough then IK is computed in order to try to bring the tip onto the target. This is the method that should be used when the start pose is far from the end pose.

We generate 'max_configs' number of samples within X number of 'trials', before ranking them according to angular distance.

Must specify either rotation in euler or quaternions, but not both!

Parameters

- **position** – The x, y, z position of the target.
- **euler** – The x, y, z orientation of the target (in radians).
- **quaternion** – A list containing the quaternion (x,y,z,w).
- **ignore_collisions** – If collision checking should be disabled.
- **trials** – The maximum number of attempts to reach max_configs.
- **max_configs** – The maximum number of configurations we want to generate before sorting them.
- **distance_threshold** – Distance indicating when IK should be computed in order to try to bring the tip onto the target.
- **max_time_ms** – Maximum time in ms spend searching for each configuation.
- **relative_to** – Indicates relative to which reference frame we want the target pose. Specify None to retrieve the absolute pose, or an Object relative to whose reference frame we want the pose.

Raises ConfigurationError if no joint configuration could be found.

Returns 'max_configs' number of joint configurations, ranked according to angular distance.

pyrep.robots.arms.configuration_path module

pyrep.robots.arms.mico module

class pyrep.robots.arms.mico.**Mico**(*count: int = 0*)
Bases: *pyrep.robots.arms.arm.Arm*

pyrep.robots.arms.panda module

class pyrep.robots.arms.panda.**Panda**(*count: int = 0*)
Bases: *pyrep.robots.arms.arm.Arm*

Module contents

pyrep.robots.end_effectors package

Submodules

pyrep.robots.end_effectors.gripper module

class pyrep.robots.end_effectors.gripper.**Gripper**(*count: int, name: str, joint_names: List[str]*)
Bases: *pyrep.robots.robot_component.RobotComponent*

Represents all types of end-effectors, e.g. grippers.

actuate(*amount: float*, *velocity: float*) → bool
Actuate the gripper, but return after each simulation step.

The functions attempts to open/close the gripper according to 'amount', where 1 represents open, and 0 represents close. The user should iteratively call this function until it returns True.

This is a convenience method. If you would like direct control of the gripper, use the `RobotComponent` methods instead.

For some grippers, this method will need to be overridden.

Parameters

- **amount** – A float between 0 and 1 representing the gripper open state. 1 means open, whilst 0 means closed.
- **velocity** – The velocity to apply to the gripper joints.

Raises ValueError if 'amount' is not between 0 and 1.

Returns True if the gripper has reached its open/closed limits, or if the 'max_force' has been exerted.

get_grasped_objects() → List[pyrep.objects.object.Object]
Gets the objects that are currently grasped.

Returns A list of grasped objects.

get_open_amount() → List[float]
Gets the gripper open state. 1 means open, whilst 0 means closed.

Returns A list of floats between 0 and 1 representing the gripper open state for each joint. 1 means open, whilst 0 means closed.

get_touch_sensor_forces() → List[List[float]]

grasp(*obj: pyrep.objects.object.Object*) → bool
Grasp object if it is detected.

Note: The does not actuate the gripper, but instead simply attaches the detected object to the gripper to 'fake' a grasp.

Parameters **obj** – The object to grasp if detected.

Returns True if the object was detected/grasped.

release() → None
Release any grasped objects.

Note: The does not actuate the gripper, but instead simply detaches any grasped objects.

pyrep.robots.end_effectors.mico_gripper module

class pyrep.robots.end_effectors.mico_gripper.**MicoGripper**(*count: int = 0*)
Bases: *pyrep.robots.end_effectors.gripper.Gripper*

pyrep.robots.end_effectors.panda_gripper module

class pyrep.robots.end_effectors.panda_gripper.**PandaGripper**(*count: int = 0*)
Bases: *pyrep.robots.end_effectors.gripper.Gripper*

Module contents

1.1.2.2 Submodules

1.1.2.3 pyrep.robots.robot_component module

class pyrep.robots.robot_component.**RobotComponent**(*count: int, name: str, joint_names: List[str], base_name: str = None*)
Bases: *pyrep.objects.object.Object*

Collection of joints representing arms, end effectors, mobile bases, etc.

copy() → pyrep.robots.robot_component.RobotComponent
Copy and pastes the arm in the scene.

The arm is copied together with all its associated calculation objects and associated scripts.

Returns The new pasted arm.

get_joint_count() → int
Gets the number of joints in this component.

Returns The number of joints.

get_joint_forces() → List[float]
Retrieves the forces or torques of the joints.

See `Joint.get_joint_force()` for more information.

Returns A list of the forces or the torques applied to the joints along/about their z-axis.

get_joint_intervals() → Tuple[List[bool], List[List[float]]]
Retrieves the interval parameters of the joints.

See `Joint.get_joint_interval()` for more information.

Returns A tuple containing a list of bools indicates whether the joint is cyclic (the joint varies between -pi and +pi in a cyclic manner), and a 2D list containing the interval of the joints.

get_joint_modes() → List[pyrep.const.JointMode]
Gets the operation mode of the joint group.

Returns A list of joint modes.

get_joint_positions() → List[float]
Retrieves the intrinsic position of the joints.

See `Joint.get_joint_position()` for more information.

Returns A list of intrinsic position of the joints.

get_joint_target_positions() → List[float]
Retrieves the target positions of the joints.

Returns A list of target position of the joints (angular or linear values depending on the joint type).

get_joint_target_velocities() → List[float]
Retrieves the intrinsic target velocities of the joints.

Returns List of the target velocity of the joints (linear or angular velocity depending on the joint-type).

get_joint_types() → List[pyrep.const.JointType]
Retrieves the type of the joints in this component.

Returns A list containing the types of the joints.

get_joint_upper_velocity_limits() → List[float]
Gets upper velocity limits of the joints.

Returns List of the upper velocity limits.

get_joint_velocities() → List[float]
Get the current joint velocities.

Returns List containing the velocities of the joints (linear or angular velocities depending on the joint-type).

get_visuals() → List[pyrep.objects.object.Object]
Gets a list of the visual elements of this component.

Can be useful for methods such as domain randomization. Should ideally be overridden for each robot.

Returns A list of visual shapes.

set_control_loop_enabled(*value: bool*) → None
Sets whether the control loop is enable for all joints.

Parameters **value** – The new value for the control loop state.

set_joint_forces(*forces: List[float]*) → None
Sets the maximum force or torque that the joints can exert.

See `Joint.set_joint_force()` for more information.

Parameters **forces** – The maximum force or torque that the joints can exert. These cannot be negative values.

set_joint_intervals(*cyclic: List[bool], intervals: List[List[float]]*) → None
Sets the interval parameters of the joints (i.e. range values).

See `Joint.set_joint_interval()` for more information.

Parameters

- **cyclic** – List of bools indicates whether the joint is cyclic. Only revolute joints with a pitch of 0 can be cyclic.
- **intervals** – 2D list containing the intervals of the joints.

set_joint_mode(*value: pyrep.const.JointMode*) → None
Sets the operation mode of the joint group.

Parameters **value** – The new joint mode value.

set_joint_positions(*positions: List[float], disable_dynamics: bool = False*) → None
Sets the intrinsic position of the joints.

See `Joint.set_joint_position()` for more information.

Parameters

- **disable_dynamics** – If True, then the position can be set even when the joint mode is in Force mode. It will disable dynamics, move the joint, and then re-enable dynamics.
- **positions** – A list of positions of the joints (angular or linear values depending on the joint type).

set_joint_target_positions(*positions: List[float]*) → None
Sets the target positions of the joints.

See `Joint.set_joint_target_position()` for more information.

Parameters **positions** – List of target position of the joints (angular or linear values depending on the joint type).

set_joint_target_velocities(*velocities: List[float]*) → None
Sets the intrinsic target velocities of the joints.

Parameters **velocities** – List of the target velocity of the joints (linear or angular velocities depending on the joint-type).

set_motor_locked_at_zero_velocity(*value: bool*) → None
Sets if motor is locked when target velocity is zero for all joints.

When enabled in velocity mode and its target velocity is zero, then the joint is locked in place.

Parameters **value** – If the motors should be locked at zero velocity.

1.1.2.4 Module contents

1.2 Submodules

1.3 pyrep.const module

class pyrep.const.**ConfigurationPathAlgorithms**
Bases: enum.Enum

An enumeration.

BITstar = 'BITstar'

BKPIECE1 = 'BKPIECE1'

BiTRRT = 'BiTRRT'

CForest = 'CForest'

EST = 'EST'

FMT = 'FMT'

KPIECE1 = 'KPIECE1'

LBKPIECE1 = 'LBKPIECE1'

LBTRRT = 'LBTRRT'

LazyPRM = 'LazyPRM'

LazyPRMstar = 'LazyPRMstar'

LazyRRT = 'LazyRRT'

PDST = 'PDST'

PRM = 'PRM'

PRMstar = 'PRMstar'

RRT = 'RRT'

RRTConnect = 'RRTConnect'

RRTstar = 'RRTstar'

SBL = 'SBL'

SPARS = 'SPARS'

SPARStwo = 'SPARStwo'

STRIDE = 'STRIDE'

TRRT = 'TRRT'

pRRT = 'pRRT'

pSBL = 'pSBL'

class pyrep.const.**JointMode**
Bases: enum.Enum

An enumeration.

DEPENDENT = 4

FORCE = 5

IK = 2

IK_DEPENDENT = 3

PASSIVE = 0

class pyrep.const.**JointType**
Bases: enum.Enum

An enumeration.

PRISMATIC = 11

REVOLUTE = 10

SPHERICAL = 12

class pyrep.const.**ObjectType**
Bases: enum.Enum

An enumeration.

ALL = -2

CAMERA = 3

DUMMY = 4

FORCE_SENSOR = 12

GRAPH = 2

JOINT = 1

LIGHT = 13

MILl = 11

MIRROR = 14

OCTREE = 15

PATH = 8

PROXIMITY_SENSOR = 5

SHAPE = 0

VISION_SENSOR = 9

VOLUME = 10

class pyrep.const.**PerspectiveMode**
Bases: enum.Enum

An enumeration.

ORTHOGRAPHIC = 0

PERSPECTIVE = 1

class pyrep.const.**PrimitiveShape**
Bases: enum.Enum

An enumeration.

CONE = 3

CUBOID = 0

CYLINDER = 2

SPHERE = 1

class pyrep.const.**RenderMode**
Bases: enum.Enum

An enumeration.

EXTERNAL = 5

EXTERNAL_WINDOWED = 6

OPENGL = 0

OPENGL3 = 7

OPENGL3_WINDOWED = 8

OPENGL_AUXILIARY = 1

OPENGL_COLOR_CODED = 2

POV_RAY = 3

class pyrep.const.**TextureMappingMode**
Bases: enum.Enum

An enumeration.

CUBE = 3

CYLINDER = 1

PLANE = 0

SPHERE = 2

class pyrep.const.**Verbosity**
Bases: enum.Enum

An enumeration.

DEBUG = 'debug'

ERRORS = 'errors'

INFOS = 'infos'

LOAD_INFOS = 'loadinfos'

NONE = 'none'

SCRIPT_ERRORS = 'scripterrors'

SCRIPT_INFOS = 'scriptinfos'

SCRIPT_WARNINGS = 'scriptwarnings'

TRACE = 'trace'

TRACE_LUA = 'tracelua'

TYRACE_ALL = 'traceall'

WARNINGS = 'warnings'

1.4 pyrep.errors module

exception pyrep.errors.**ConfigurationError**
Bases: Exception

exception pyrep.errors.**ConfigurationPathError**
Bases: Exception

exception pyrep.errors.**GripperError**
Bases: Exception

exception pyrep.errors.**IKError**
Bases: Exception

exception pyrep.errors.**ObjectAlreadyRemovedError**
Bases: Exception

exception pyrep.errors.**ObjectIsNotModelError**
Bases: Exception

exception pyrep.errors.**PyRepError**
Bases: Exception

exception pyrep.errors.**WrongObjectTypeError**
Bases: Exception

1.5 pyrep.pyrep module

class pyrep.pyrep.**PyRep**
Bases: object

Used for interfacing with the CoppeliaSim simulation.

Can be used for starting, stopping, and stepping the simulation. As well as getting, and creating scene objects and robots.

create_texture(*filename: str*, *interpolate=True*, *decal_mode=False*, *repeat_along_u=False*, *repeat_along_v=False*) → Tuple[pyrep.objects.shape.Shape, pyrep.textures.texture.Texture]
Creates a planar shape that is textured.

Parameters

- **filename** – Path to the texture to load.
- **interpolate** – Adjacent texture pixels are not interpolated.
- **decal_mode** – Texture is applied as a decal (its appearance won't be influenced by light conditions).
- **repeat_along_u** – Texture will be repeated along the U direction.
- **repeat_along_v** – Texture will be repeated along the V direction.

Returns A tuple containing the textured plane and the texture.

export_scene(*filename: str*) → None
Saves the current scene.

Parameters **filename** – scene filename. The filename extension is required ("ttt").

get_collection_handle_by_name(*collection_name: str*) → int
Retrieves the integer handle for a given collection.

Parameters **collection_name** – Name of the collection to retrieve the integer handle for

Returns An integer handle for the collection

get_objects_in_tree(*root_object=None*, **args*, ***kwargs*) → List[pyrep.objects.object.Object]
Retrieves the objects in a given hierarchy tree.

Parameters

- **root_object** – The root object in the tree. Pass None to retrieve all objects in the configuration tree. `Object` or *int*.
- **object_type** – The object type to retrieve. One of *ObjectType*.
- **exclude_base** – Exclude the tree base from the returned list.
- **first_generation_only** – Include in the returned list only the object's first children. Otherwise, entire hierarchy is returned.

Returns A list of objects in the hierarchy tree.

get_simulation_timestep() → float
Gets the simulation time step.

Returns The time step value in seconds.

group_objects(*objects: List[pyrep.objects.shape.Shape]*) → pyrep.objects.shape.Shape
Groups several shapes into a compound shape (or simple shape).

Parameters **objects** – The list of shapes to group.

Returns A single grouped shape.

import_model(*filename: str*) → pyrep.objects.object.Object
Loads a previously saved model.

Parameters **filename** – model filename. The filename extension is required ("ttm"). An optional "@copy" can be appended to the filename, in which case the model's objects will be named/renamed as if an associated script was attached to the model.

Returns The imported model.

launch(*scene_file: str = ''*, *headless: bool = False*, *responsive_ui: bool = False*, *blocking: bool = False*, *verbosity: pyrep.const.Verbosity = <Verbosity.NONE: 'none'>*) → None
Launches CoppeliaSim.

Launches the UI thread, waits until the UI thread has finished, this results in the current thread becoming the simulation thread.

Parameters

- **scene_file** – The scene file to load. Empty string for empty scene.
- **headless** – Run CoppeliaSim in simulation mode.
- **responsive_ui** – If True, then a separate thread will be created to asynchronously step the UI of CoppeliaSim. Note, that will reduce the responsiveness of the simulation thread.
- **blocking** – Causes CoppeliaSim to launch as if running the default c++ client application. This is causes the function to block. For most users, this will be set to False.
- **verbosity** – The verbosity level for CoppeliaSim. Usually Verbosity.NONE or Verbosity.LOAD_INFOS.

merge_objects(*objects: List[pyrep.objects.shape.Shape]*) → pyrep.objects.shape.Shape
Merges several shapes into a compound shape (or simple shape).

Parameters **objects** – The list of shapes to group.

Returns A single merged shape.

script_call(*function_name_at_script_name: str*, *script_handle_or_type: int*, *ints=()*, *floats=()*, *strings=()*, *bytes=''*) → Tuple[List[int], List[float], List[str], str]
Calls a script function (from a plugin, the main client application, or from another script). This represents a callback inside of a script.

Parameters

- **function_name_at_script_name** – A string representing the function name and script name, e.g. myFunctionName@theScriptName. When the script is not associated with an object, then just specify the function name.
- **script_handle_or_type** – The handle of the script, otherwise the type of the script.
- **ints** – The input ints to the script.
- **floats** – The input floats to the script.
- **strings** – The input strings to the script.
- **bytes** – The input bytes to the script (as a string).

Returns Any number of return values from the called Lua function.

set_configuration_tree(*config_tree: bytes*) → None
Restores configuration information previously retrieved.

Configuration information (object relative positions/orientations, joint/path values) can be retrieved with `Object.get_configuration_tree()`. Dynamically simulated objects will implicitly be reset before the command is applied (i.e. similar to calling `Object.reset_dynamic_object()` just before).

Parameters **config_tree** – The configuration tree to restore.

set_simulation_timestep(*dt: float*) → None
Sets the simulation time step. Default is 0.05.

Parameters **dt** – The time step value in seconds.

shutdown() → None
Shuts down the CoppeliaSim simulation.

start() → None
Starts the physics simulation if it is not already running.

step() → None
Execute the next simulation step.

If the physics simulation is not running, then this will only update the UI.

step_ui() → None
Update the UI.

This will not execute the next simulation step, even if the physics simulation is running. This is only applicable when PyRep was launched without a responsive UI.

stop() → None
Stops the physics simulation if it is running.

1.6 Module contents

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

pyrep, 38
pyrep.const, 32
pyrep.errors, 35
pyrep.objects, 23
pyrep.objects.cartesian_path, 3
pyrep.objects.dummy, 4
pyrep.objects.force_sensor, 4
pyrep.objects.joint, 5
pyrep.objects.object, 7
pyrep.objects.proximity_sensor, 14
pyrep.objects.shape, 14
pyrep.objects.vision_sensor, 20
pyrep.pyrep, 35
pyrep.robots, 32
pyrep.robots.arms, 28
pyrep.robots.arms.arm, 23
pyrep.robots.arms.mico, 28
pyrep.robots.arms.panda, 28
pyrep.robots.end_effectors, 29
pyrep.robots.end_effectors.gripper, 28
pyrep.robots.end_effectors.mico_gripper, 29
pyrep.robots.end_effectors.panda_gripper, 29
pyrep.robots.robot_component, 29

Index

A

actuate() (*pyrep.robots.end_effectors.gripper.Gripper method*), 28
add_force() (*pyrep.objects.shape.Shape method*), 15
add_force_and_torque() (*pyrep.objects.shape.Shape method*), 15
ALL (*pyrep.const.ObjectType attribute*), 33
apply_texture() (*pyrep.objects.shape.Shape method*), 15
Arm (*class in pyrep.robots.arms.arm*), 23

B

BiTRRT (*pyrep.const.ConfigurationPathAlgorithms attribute*), 32
BITstar (*pyrep.const.ConfigurationPathAlgorithms attribute*), 32
BKPIECE1 (*pyrep.const.ConfigurationPathAlgorithms attribute*), 32

C

CAMERA (*pyrep.const.ObjectType attribute*), 33
capture_depth() (*pyrep.objects.vision_sensor.VisionSensor method*), 20
capture_pointcloud() (*pyrep.objects.vision_sensor.VisionSensor method*), 20
capture_rgb() (*pyrep.objects.vision_sensor.VisionSensor method*), 20
CartesianPath (*class in pyrep.objects.cartesian_path*), 3
CForest (*pyrep.const.ConfigurationPathAlgorithms attribute*), 32
check_arm_collision() (*pyrep.robots.arms.arm.Arm method*), 23
check_collision() (*pyrep.objects.object.Object method*), 7
check_distance() (*pyrep.objects.object.Object method*), 7
colors (*pyrep.objects.shape.SShapeVizInfo attribute*), 14
compute_mass_and_inertia() (*pyrep.objects.shape.Shape method*), 16
CONE (*pyrep.const.PrimitiveShape attribute*), 33
ConfigurationError, 35
ConfigurationPathAlgorithms (*class in pyrep.const*), 32
ConfigurationPathError, 35
copy() (*pyrep.objects.object.Object method*), 7
copy() (*pyrep.robots.robot_component.RobotComponent method*), 29
create() (*pyrep.objects.cartesian_path.CartesianPath static method*), 3
create() (*pyrep.objects.dummy.Dummy static method*), 4
create() (*pyrep.objects.force_sensor.ForceSensor class method*), 4
create() (*pyrep.objects.shape.Shape static method*), 16
create() (*pyrep.objects.vision_sensor.VisionSensor static method*), 20
create_mesh() (*pyrep.objects.shape.Shape static method*), 16
create_texture() (*pyrep.pyrep.PyRep method*), 35
CUBE (*pyrep.const.TextureMappingMode attribute*), 34
CUBOID (*pyrep.const.PrimitiveShape attribute*), 33
CYLINDER (*pyrep.const.PrimitiveShape attribute*), 34
CYLINDER (*pyrep.const.TextureMappingMode attribute*), 34

D

DEBUG (*pyrep.const.Verbosity attribute*), 34
decimate_mesh() (*pyrep.objects.shape.Shape method*), 17
DEPENDENT (*pyrep.const.JointMode attribute*), 32
Dummy (*class in pyrep.objects.dummy*), 4
DUMMY (*pyrep.const.ObjectType attribute*), 33

E

ERRORS (*pyrep.const.Verbosity attribute*), 34
EST (*pyrep.const.ConfigurationPathAlgorithms attribute*), 32
exists() (*pyrep.objects.object.Object static method*), 8
export_scene() (*pyrep.pyrep.PyRep method*), 35
EXTERNAL (*pyrep.const.RenderMode attribute*), 34
EXTERNAL_WINDOWED (*pyrep.const.RenderMode attribute*), 34

F

FMT (*pyrep.const.ConfigurationPathAlgorithms attribute*), 32
FORCE (*pyrep.const.JointMode attribute*), 32
FORCE_SENSOR (*pyrep.const.ObjectType attribute*), 33
ForceSensor (*class in pyrep.objects.force_sensor*), 4

G

get_bounding_box() (*pyrep.objects.object.Object method*), 8
get_bullet_friction() (*pyrep.objects.object.Object method*), 8
get_collection_handle_by_name() (*pyrep.pyrep.PyRep method*), 35
get_color() (*pyrep.objects.shape.Shape method*), 17
get_configs_for_tip_pose() (*pyrep.robots.arms.arm.Arm method*), 23
get_configuration_tree() (*pyrep.objects.object.Object method*), 8
get_contact() (*pyrep.objects.object.Object method*), 8
get_convex_decomposition() (*pyrep.objects.shape.Shape method*), 17
get_entity_to_render() (*pyrep.objects.vision_sensor.VisionSensor method*), 21
get_explicit_handling() (*pyrep.objects.object.Object method*), 8
get_extension_string() (*pyrep.objects.object.Object method*), 8
get_far_clipping_plane() (*pyrep.objects.vision_sensor.VisionSensor method*), 21
get_grasped_objects() (*pyrep.robots.end_effectors.gripper.Gripper method*), 28
get_handle() (*pyrep.objects.object.Object method*), 8
get_intrinsic_matrix() (*pyrep.objects.vision_sensor.VisionSensor method*), 21
get_jacobian() (*pyrep.robots.arms.arm.Arm method*), 24
get_joint_count() (*pyrep.robots.robot_component.RobotComponent method*), 29
get_joint_force() (*pyrep.objects.joint.Joint method*), 5
get_joint_forces() (*pyrep.robots.robot_component.RobotComponent method*), 29
get_joint_interval() (*pyrep.objects.joint.Joint method*), 5
get_joint_intervals() (*pyrep.robots.robot_component.RobotComponent method*), 30
get_joint_mode() (*pyrep.objects.joint.Joint method*), 5
get_joint_modes() (*pyrep.robots.robot_component.RobotComponent method*), 30
get_joint_position() (*pyrep.objects.joint.Joint method*), 5
get_joint_positions() (*pyrep.robots.robot_component.RobotComponent method*), 30
get_joint_target_position() (*pyrep.objects.joint.Joint method*), 5
get_joint_target_positions() (*pyrep.robots.robot_component.RobotComponent method*), 30
get_joint_target_velocities() (*pyrep.robots.robot_component.RobotComponent method*), 30
get_joint_target_velocity() (*pyrep.objects.joint.Joint method*), 5
get_joint_type() (*pyrep.objects.joint.Joint method*), 5
get_joint_types() (*pyrep.robots.robot_component.RobotComponent method*), 30
get_joint_upper_velocity_limit() (*pyrep.objects.joint.Joint method*), 5
get_joint_upper_velocity_limits() (*pyrep.robots.robot_component.RobotComponent method*), 30
get_joint_velocities() (*pyrep.robots.robot_component.RobotComponent method*), 30
get_joint_velocity() (*pyrep.objects.joint.Joint method*), 6
get_linear_path() (*pyrep.robots.arms.arm.Arm method*), 24
get_mass() (*pyrep.objects.shape.Shape method*), 18
get_matrix() (*pyrep.objects.object.Object method*), 8
get_mesh_data() (*pyrep.objects.shape.Shape*

method), 18
get_model_bounding_box() (pyrep.objects.object.Object method), 8
get_name() (pyrep.objects.object.Object method), 9
get_near_clipping_plane() (pyrep.objects.vision_sensor.VisionSensor method), 21
get_nonlinear_path() (pyrep.robots.arms.arm.Arm method), 24
get_object() (pyrep.objects.object.Object static method), 9
get_object_name() (pyrep.objects.object.Object static method), 9
get_object_type() (pyrep.objects.object.Object static method), 9
get_objects_in_tree() (pyrep.objects.object.Object method), 9
get_objects_in_tree() (pyrep.pyrep.PyRep method), 36
get_open_amount() (pyrep.robots.end_effectors.gripper.Gripper method), 29
get_orientation() (pyrep.objects.object.Object method), 9
get_orthographic_size() (pyrep.objects.vision_sensor.VisionSensor method), 21
get_parent() (pyrep.objects.object.Object method), 9
get_path() (pyrep.robots.arms.arm.Arm method), 25
get_path_from_cartesian_path() (pyrep.robots.arms.arm.Arm method), 26
get_perspective_angle() (pyrep.objects.vision_sensor.VisionSensor method), 22
get_perspective_mode() (pyrep.objects.vision_sensor.VisionSensor method), 22
get_pose() (pyrep.objects.object.Object method), 9
get_pose_on_path() (pyrep.objects.cartesian_path.CartesianPath method), 4
get_position() (pyrep.objects.object.Object method), 9
get_quaternion() (pyrep.objects.object.Object method), 10
get_render_mode() (pyrep.objects.vision_sensor.VisionSensor method), 22
get_resolution() (pyrep.objects.vision_sensor.VisionSensor method), 22
get_shape_viz() (pyrep.objects.shape.Shape method), 18
get_simulation_timestep() (pyrep.pyrep.PyRep method), 36
get_texture() (pyrep.objects.shape.Shape method), 18
get_tip() (pyrep.robots.arms.arm.Arm method), 26
get_touch_sensor_forces() (pyrep.robots.end_effectors.gripper.Gripper method), 29
get_transparency() (pyrep.objects.shape.Shape method), 18
get_type() (pyrep.objects.object.Object method), 10
get_velocity() (pyrep.objects.object.Object method), 10
get_visuals() (pyrep.robots.robot_component.RobotComponent method), 30
get_windowed_size() (pyrep.objects.vision_sensor.VisionSensor method), 22
GRAPH (pyrep.const.ObjectType attribute), 33
grasp() (pyrep.robots.end_effectors.gripper.Gripper method), 29
Gripper (class in pyrep.robots.end_effectors.gripper), 28
GripperError, 35
group_objects() (pyrep.pyrep.PyRep method), 36

H

handle_explicitly() (pyrep.objects.vision_sensor.VisionSensor method), 22

I

IK (pyrep.const.JointMode attribute), 33
IK_DEPENDENT (pyrep.const.JointMode attribute), 33
IKError, 35
import_mesh() (pyrep.objects.shape.Shape static method), 18
import_model() (pyrep.pyrep.PyRep method), 36
import_shape() (pyrep.objects.shape.Shape class method), 19
indices (pyrep.objects.shape.SShapeVizInfo attribute), 14
INFOS (pyrep.const.Verbosity attribute), 34
insert_control_points() (pyrep.objects.cartesian_path.CartesianPath method), 4
is_collidable() (pyrep.objects.object.Object method), 10
is_control_loop_enabled() (pyrep.objects.joint.Joint method), 6
is_detectable() (pyrep.objects.object.Object method), 10
is_detected() (pyrep.objects.proximity_sensor.ProximitySensor method), 14

is_dynamic() (*pyrep.objects.shape.Shape method*), 19
is_measurable() (*pyrep.objects.object.Object method*), 10
is_model() (*pyrep.objects.object.Object method*), 10
is_model_collidable() (*pyrep.objects.object.Object method*), 10
is_model_detectable() (*pyrep.objects.object.Object method*), 10
is_model_dynamic() (*pyrep.objects.object.Object method*), 10
is_model_measurable() (*pyrep.objects.object.Object method*), 10
is_model_renderable() (*pyrep.objects.object.Object method*), 11
is_model_respondable() (*pyrep.objects.object.Object method*), 11
is_motor_enabled() (*pyrep.objects.joint.Joint method*), 6
is_motor_locked_at_zero_velocity() (*pyrep.objects.joint.Joint method*), 6
is_renderable() (*pyrep.objects.object.Object method*), 11
is_respondable() (*pyrep.objects.shape.Shape method*), 19

J

Joint (*class in pyrep.objects.joint*), 5
JOINT (*pyrep.const.ObjectType attribute*), 33
JointMode (*class in pyrep.const*), 32
JointType (*class in pyrep.const*), 33

K

KPIECE1 (*pyrep.const.ConfigurationPathAlgorithms attribute*), 32

L

launch() (*pyrep.pyrep.PyRep method*), 36
LazyPRM (*pyrep.const.ConfigurationPathAlgorithms attribute*), 32
LazyPRMstar (*pyrep.const.ConfigurationPathAlgorithms attribute*), 32
LazyRRT (*pyrep.const.ConfigurationPathAlgorithms attribute*), 32
LBKPIECE1 (*pyrep.const.ConfigurationPathAlgorithms attribute*), 32
LBTRRT (*pyrep.const.ConfigurationPathAlgorithms attribute*), 32
LIGHT (*pyrep.const.ObjectType attribute*), 33
LOAD_INFOS (*pyrep.const.Verbosity attribute*), 34

M

merge_objects() (*pyrep.pyrep.PyRep method*), 36
Mico (*class in pyrep.robots.arms.mico*), 28
MicoGripper (*class in pyrep.robots.end_effectors.mico_gripper*), 29
MILl (*pyrep.const.ObjectType attribute*), 33
MIRROR (*pyrep.const.ObjectType attribute*), 33

N

NONE (*pyrep.const.Verbosity attribute*), 34
normals (*pyrep.objects.shape.SShapeVizInfo attribute*), 14

O

Object (*class in pyrep.objects.object*), 7
ObjectAlreadyRemovedError, 35
ObjectIsNotModelError, 35
ObjectType (*class in pyrep.const*), 33
OCTREE (*pyrep.const.ObjectType attribute*), 33
OPENGL (*pyrep.const.RenderMode attribute*), 34
OPENGL3 (*pyrep.const.RenderMode attribute*), 34
OPENGL3_WINDOWED (*pyrep.const.RenderMode attribute*), 34
OPENGL_AUXILIARY (*pyrep.const.RenderMode attribute*), 34
OPENGL_COLOR_CODED (*pyrep.const.RenderMode attribute*), 34
ORTHOGRAPHIC (*pyrep.const.PerspectiveMode attribute*), 33

P

Panda (*class in pyrep.robots.arms.panda*), 28
PandaGripper (*class in pyrep.robots.end_effectors.panda_gripper*), 29
PASSIVE (*pyrep.const.JointMode attribute*), 33
PATH (*pyrep.const.ObjectType attribute*), 33
PDST (*pyrep.const.ConfigurationPathAlgorithms attribute*), 32
PERSPECTIVE (*pyrep.const.PerspectiveMode attribute*), 33
PerspectiveMode (*class in pyrep.const*), 33
PLANE (*pyrep.const.TextureMappingMode attribute*), 34
pointcloud_from_depth() (*pyrep.objects.vision_sensor.VisionSensor method*), 22
pointcloud_from_depth_and_camera_params() (*pyrep.objects.vision_sensor.VisionSensor static method*), 22
POV_RAY (*pyrep.const.RenderMode attribute*), 34
PrimitiveShape (*class in pyrep.const*), 33
PRISMATIC (*pyrep.const.JointType attribute*), 33
PRM (*pyrep.const.ConfigurationPathAlgorithms attribute*), 32

PRMstar (*pyrep.const.ConfigurationPathAlgorithms attribute*), 32
PROXIMITY_SENSOR (*pyrep.const.ObjectType attribute*), 33
ProximitySensor (*class in pyrep.objects.proximity_sensor*), 14
pRRT (*pyrep.const.ConfigurationPathAlgorithms attribute*), 32
pSBL (*pyrep.const.ConfigurationPathAlgorithms attribute*), 32
PyRep (*class in pyrep.pyrep*), 35
pyrep (*module*), 38
pyrep.const (*module*), 32
pyrep.errors (*module*), 35
pyrep.objects (*module*), 23
pyrep.objects.cartesian_path (*module*), 3
pyrep.objects.dummy (*module*), 4
pyrep.objects.force_sensor (*module*), 4
pyrep.objects.joint (*module*), 5
pyrep.objects.object (*module*), 7
pyrep.objects.proximity_sensor (*module*), 14
pyrep.objects.shape (*module*), 14
pyrep.objects.vision_sensor (*module*), 20
pyrep.pyrep (*module*), 35
pyrep.robots (*module*), 32
pyrep.robots.arms (*module*), 28
pyrep.robots.arms.arm (*module*), 23
pyrep.robots.arms.mico (*module*), 28
pyrep.robots.arms.panda (*module*), 28
pyrep.robots.end_effectors (*module*), 29
pyrep.robots.end_effectors.gripper (*module*), 28
pyrep.robots.end_effectors.mico_gripper (*module*), 29
pyrep.robots.end_effectors.panda_gripper (*module*), 29
pyrep.robots.robot_component (*module*), 29
PyRepError, 35

R

read() (*pyrep.objects.force_sensor.ForceSensor method*), 5
read() (*pyrep.objects.proximity_sensor.ProximitySensor method*), 14
release() (*pyrep.robots.end_effectors.gripper.Gripper method*), 29
remove() (*pyrep.objects.object.Object method*), 11
remove_texture() (*pyrep.objects.shape.Shape method*), 19
RenderMode (*class in pyrep.const*), 34
reorient_bounding_box() (*pyrep.objects.shape.Shape method*), 19
reset_dynamic_object() (*pyrep.objects.object.Object method*), 11
REVOLUTE (*pyrep.const.JointType attribute*), 33
RobotComponent (*class in pyrep.robots.robot_component*), 29
rotate() (*pyrep.objects.object.Object method*), 11
RRT (*pyrep.const.ConfigurationPathAlgorithms attribute*), 32
RRTConnect (*pyrep.const.ConfigurationPathAlgorithms attribute*), 32
RRTstar (*pyrep.const.ConfigurationPathAlgorithms attribute*), 32

S

save_model() (*pyrep.objects.object.Object method*), 11
SBL (*pyrep.const.ConfigurationPathAlgorithms attribute*), 32
script_call() (*pyrep.pyrep.PyRep method*), 37
SCRIPT_ERRORS (*pyrep.const.Verbosity attribute*), 34
SCRIPT_INFOS (*pyrep.const.Verbosity attribute*), 34
SCRIPT_WARNINGS (*pyrep.const.Verbosity attribute*), 34
set_bullet_friction() (*pyrep.objects.object.Object method*), 11
set_collidable() (*pyrep.objects.object.Object method*), 11
set_color() (*pyrep.objects.shape.Shape method*), 19
set_configuration_tree() (*pyrep.pyrep.PyRep method*), 37
set_control_loop_enabled() (*pyrep.objects.joint.Joint method*), 6
set_control_loop_enabled() (*pyrep.robots.robot_component.RobotComponent method*), 30
set_detectable() (*pyrep.objects.object.Object method*), 11
set_dynamic() (*pyrep.objects.shape.Shape method*), 19
set_entity_to_render() (*pyrep.objects.vision_sensor.VisionSensor method*), 22
set_explicit_handling() (*pyrep.objects.object.Object method*), 12
set_far_clipping_plane() (*pyrep.objects.vision_sensor.VisionSensor method*), 22
set_ik_element_properties() (*pyrep.robots.arms.arm.Arm method*), 26
set_ik_group_properties() (*pyrep.robots.arms.arm.Arm method*), 26
set_joint_force() (*pyrep.objects.joint.Joint method*), 6

set_joint_forces() (pyrep.robots.robot_component.RobotComponent method), 30
set_joint_interval() (pyrep.objects.joint.Joint method), 6
set_joint_intervals() (pyrep.robots.robot_component.RobotComponent method), 31
set_joint_mode() (pyrep.objects.joint.Joint method), 6
set_joint_mode() (pyrep.robots.robot_component.RobotComponent method), 31
set_joint_position() (pyrep.objects.joint.Joint method), 6
set_joint_positions() (pyrep.robots.robot_component.RobotComponent method), 31
set_joint_target_position() (pyrep.objects.joint.Joint method), 7
set_joint_target_positions() (pyrep.robots.robot_component.RobotComponent method), 31
set_joint_target_velocities() (pyrep.robots.robot_component.RobotComponent method), 31
set_joint_target_velocity() (pyrep.objects.joint.Joint method), 7
set_mass() (pyrep.objects.shape.Shape method), 19
set_matrix() (pyrep.objects.object.Object method), 12
set_measurable() (pyrep.objects.object.Object method), 12
set_model() (pyrep.objects.object.Object method), 12
set_model_collidable() (pyrep.objects.object.Object method), 12
set_model_detectable() (pyrep.objects.object.Object method), 12
set_model_dynamic() (pyrep.objects.object.Object method), 12
set_model_measurable() (pyrep.objects.object.Object method), 12
set_model_renderable() (pyrep.objects.object.Object method), 12
set_model_respondable() (pyrep.objects.object.Object method), 12
set_motor_enabled() (pyrep.objects.joint.Joint method), 7
set_motor_locked_at_zero_velocity() (pyrep.objects.joint.Joint method), 7
set_motor_locked_at_zero_velocity() (pyrep.robots.robot_component.RobotComponent method), 31
set_name() (pyrep.objects.object.Object method), 13
set_near_clipping_plane() (pyrep.objects.vision_sensor.VisionSensor method), 22
set_orientation() (pyrep.objects.object.Object method), 13
set_orthographic_size() (pyrep.objects.vision_sensor.VisionSensor method), 22
set_parent() (pyrep.objects.object.Object method), 13
set_perspective_angle() (pyrep.objects.vision_sensor.VisionSensor method), 22
set_perspective_mode() (pyrep.objects.vision_sensor.VisionSensor method), 23
set_pose() (pyrep.objects.object.Object method), 13
set_position() (pyrep.objects.object.Object method), 13
set_quaternion() (pyrep.objects.object.Object method), 13
set_render_mode() (pyrep.objects.vision_sensor.VisionSensor method), 23
set_renderable() (pyrep.objects.object.Object method), 14
set_resolution() (pyrep.objects.vision_sensor.VisionSensor method), 23
set_respondable() (pyrep.objects.shape.Shape method), 19
set_simulation_timestep() (pyrep.pyrep.PyRep method), 37
set_texture() (pyrep.objects.shape.Shape method), 19
set_transparency() (pyrep.objects.shape.Shape method), 20
set_windowed_size() (pyrep.objects.vision_sensor.VisionSensor method), 23
shading_angle (pyrep.objects.shape.SShapeVizInfo attribute), 15
Shape (class in pyrep.objects.shape), 15
SHAPE (pyrep.const.ObjectType attribute), 33
shutdown() (pyrep.pyrep.PyRep method), 37
solve_ik() (pyrep.robots.arms.arm.Arm method), 26
solve_ik_via_jacobian() (pyrep.robots.arms.arm.Arm method), 27
solve_ik_via_sampling() (pyrep.robots.arms.arm.Arm method), 27
SPARS (pyrep.const.ConfigurationPathAlgorithms attribute), 32
SPARStwo (pyrep.const.ConfigurationPathAlgorithms attribute), 32
SPHERE (pyrep.const.PrimitiveShape attribute), 34

SPHERE (*pyrep.const.TextureMappingMode attribute*), 34
SPHERICAL (*pyrep.const.JointType attribute*), 33
SShapeVizInfo (*class in pyrep.objects.shape*), 14
start() (*pyrep.pyrep.PyRep method*), 37
step() (*pyrep.pyrep.PyRep method*), 37
step_ui() (*pyrep.pyrep.PyRep method*), 37
still_exists() (*pyrep.objects.object.Object method*), 14
stop() (*pyrep.pyrep.PyRep method*), 37
STRIDE (*pyrep.const.ConfigurationPathAlgorithms attribute*), 32

T

texture (*pyrep.objects.shape.SShapeVizInfo attribute*), 15
texture_apply_mode (*pyrep.objects.shape.SShapeVizInfo attribute*), 15
texture_coords (*pyrep.objects.shape.SShapeVizInfo attribute*), 15
texture_id (*pyrep.objects.shape.SShapeVizInfo attribute*), 15
texture_options (*pyrep.objects.shape.SShapeVizInfo attribute*), 15
TextureMappingMode (*class in pyrep.const*), 34
TRACE (*pyrep.const.Verbosity attribute*), 34
TRACE_LUA (*pyrep.const.Verbosity attribute*), 34
TRRT (*pyrep.const.ConfigurationPathAlgorithms attribute*), 32
TYRACE_ALL (*pyrep.const.Verbosity attribute*), 34

U

ungroup() (*pyrep.objects.shape.Shape method*), 20

V

Verbosity (*class in pyrep.const*), 34
vertices (*pyrep.objects.shape.SShapeVizInfo attribute*), 15
VISION_SENSOR (*pyrep.const.ObjectType attribute*), 33
VisionSensor (*class in pyrep.objects.vision_sensor*), 20
VOLUME (*pyrep.const.ObjectType attribute*), 33

W

WARNINGS (*pyrep.const.Verbosity attribute*), 34
WrongObjectTypeError, 35