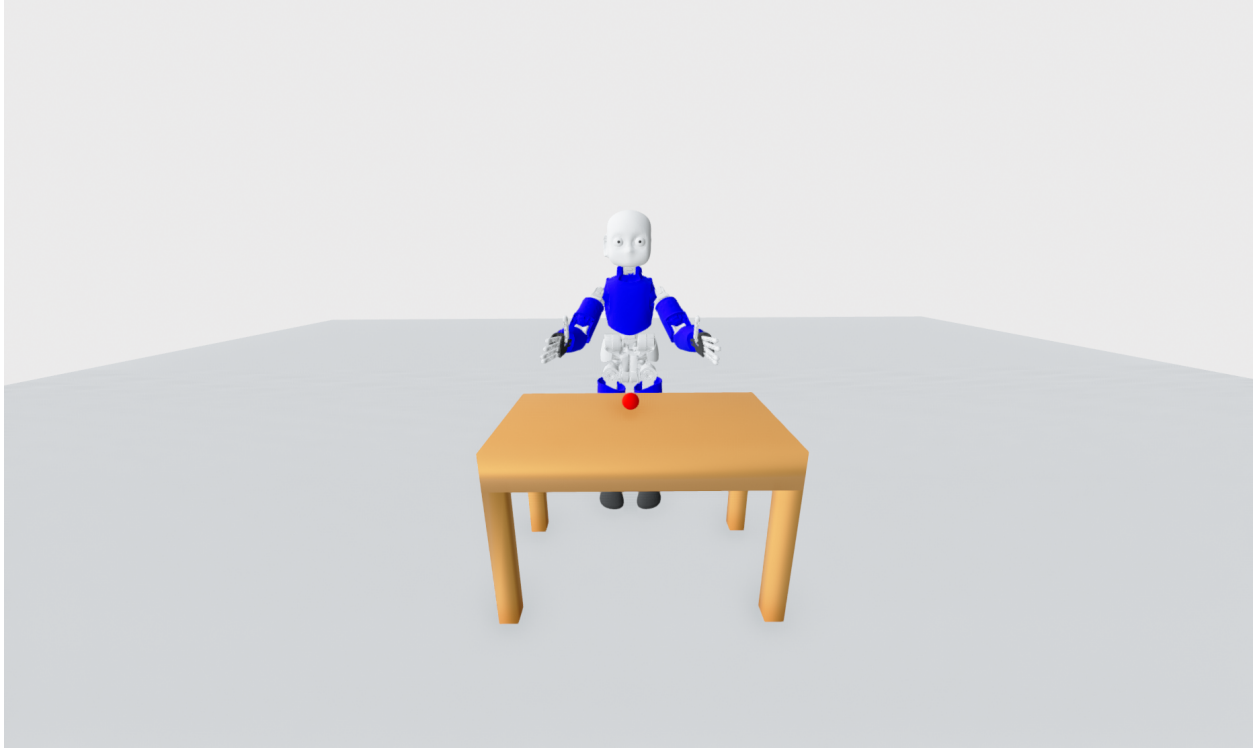


# pyCub

Author: Lukas Rustler



**CONTENTS:**

<b>1</b>	<b>README</b>	<b>1</b>
1.1	Known bugs . . . . .	1
1.2	Installation . . . . .	1
1.3	Examples . . . . .	1
1.4	Information . . . . .	2
1.5	Docker . . . . .	2
<b>2</b>	<b>icub_pybullet</b>	<b>7</b>
2.1	pyCub . . . . .	7
2.2	utils . . . . .	11
2.3	visualizer . . . . .	13
<b>3</b>	<b>examples</b>	<b>17</b>
3.1	push_the_ball_cartesian . . . . .	17
3.2	push_the_ball_pure_joints . . . . .	17
3.3	skin_test . . . . .	17
	<b>Python Module Index</b>	<b>19</b>
	<b>Index</b>	<b>21</b>



## README

- this repo contains code for iCub simulation in PyBullet
- it provides also dockerfile for easy setup of the environment

### 1.1 Known bugs

- visualization with skin dies after ~65k steps
  - e.g., <https://github.com/isl-org/Open3D/issues/4992>

### 1.2 Installation

1. Install dependencies
  - use Docker
    - see *Docker* section
  - or install python3 and pip3 `install pybullet numpy scipy open3d`
2. Pull this repository
3. try run some example, e.g., `python3 PATH_TO_THE_REPOSITORY/icub_pybullet/examples/push_the_ball_cartesian.py`

### 1.3 Examples

- `push_the_ball_pure_joints.py` contains an example that shows how to control the robot in joint space
- `push_the_ball_cartesian.py` contains an example that shows how to control the robot in Cartesian space
- `skin_test.py` contains an example with balls falling the robot and skin should turn green on the places where contact occurs. You may want to slow the simulation a little bit to see that :)

## 1.4 Information

- documentation can be found at [lukasrustler.cz/pycub](http://lukasrustler.cz/pycub) or in [pyCub.pdf](#)
- simulator code is in [pycub.py](#)
  - it uses PyBullet for simulation and provides high-level interface
- visualization code in [visualizer.py](#)
  - it uses Open3D for visualization as it is much more customizable than PyBullet default GUI
- movement is done using position control. You can either use position control directly (`pycub.move_position()`) or use cartesian control (`pycub.move_cartesian()`)
  - **Neither of these check for collision before movement!**
  - Function `pycub.motion_done()` check whether all joints reached the target or whether collision occurred. If collision, the variable `pycub.collision_during_movement` is set. You can also run `pycub.motion_done()` with `check_collision=False` to ignore collision checks, e.g., to get out of collision state
- until the repository is installed as a packages, examples contains line `sys.path.insert(0, os.path.dirname(os.path.dirname(os.path.abspath(__file__))))` so that python can find the necessary modules. To avoid that, you can run all codes from `icub_pybullet` folder or use PyCharm (or other IDE)

## 1.5 Docker

<https://github.com/rustlluk/easy-docker> is utilized to use Docker

### 1.5.1 Installation

- install [docker-engine](#) (**DO NOT INSTALL DOCKER DESKTOP**), do [post-installation steps](#) and (optional) install [nvidia-docker](#) for GPU support
- For ubuntu (and Mint, but you have) users:
  - if you are a mint user, change `VERSION_CODENAME` to `UBUNTU_CODENAME`

```
sudo apt-get update
sudo apt-get install ca-certificates curl gnupg
sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /
↳etc/apt/keyrings/docker.gpg
sudo chmod a+r /etc/apt/keyrings/docker.gpg

echo \
  "deb [arch="$(dpkg --print-architecture)" signed-by=/etc/apt/keyrings/docker.gpg]
↳https://download.docker.com/linux/ubuntu \
  "$(. /etc/os-release && echo "$VERSION_CODENAME")" stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin
↳docker-compose-plugin
```

- and post-installation to use docker without sudo:

```
sudo groupadd docker
sudo usermod -aG docker $USER
```

– and restart your computer

- clone this repository

```
cd SOME_PATH
git clone https://github.com/rustlluk/pyCub.git
```

- (optionally) rename it to be called the same as in docker

```
mv SOME_PATH/pycub SOME_PATH/pycub_ws
```

- run the docker (see [Parameters](#) for more parameters)

```
cd SOME_PATH/pycub_ws/Docker
```

– to build it on your computer:

```
./deploy.py -b -p SOME_PATH/pycub_ws -c pycub
```

## 1.5.2 Docker + PyCharm

1. Either run pycharm from docker
2. In your local docker:
  - add ssh interpreter
    - user docker
    - ip can be localhost or ip where you run the docker
    - port 2222
  - uncheck automatic upload to remote folder
  - change remote path to /home/docker/pycub\_ws

Common steps:

- mark all folder icub\_pybullet as Source Root
- for X11 forwarding:
  - Click on configurations drop menu -> Edit Configurations -> Edit configuration templates -> Python -> Edit environment variables -> add DISPLAY with the same value as in docker and uncheck 'Include system environment variables'. Every new configuration will have that settings from now
    - \* if you already have configuration created before doing the above -> delete it and create again, or change it manually

### 1.5.3 Deploy Parameters

- cd to folder with Dockerfile
- ./deploy.py
  - -b or --build when building
    - \* default: False
  - -nv or --nvidia when you want to use your Nvidia card
    - \* you have to use it when creating a new container
    - \* default: False
  - -e if you just want to run existing docker without building
    - \* default: False
  - -p or --path with path to current folder
    - \* default: ""
  - -c or --container with desired name of the new, created container
    - \* default: my\_new\_docker
  - -t or --terminal to run new terminal in running docker session
    - \* default: False
  - -pv or --python-version to specify addition python version to install
    - \* default: 3.11
  - -pcv or --pycharm-version to specify version of pycharm to use
    - \* default: 2023.2.3
  - -bi or --base-image to specify base image that will be used
    - \* default: nvidia/cuda:11.0.3-devel-ubuntu20.04
    - \* other can be found at [hub.docker.com](https://hub.docker.com)

Do this on computer where you will run the code. If you have a server you have to run it on the server over SSH to make things work properly.

### 1.5.4 FAQ

- **applications do not run on your screen** (or you have some strange screen related errors)
  - in another terminal run `xhost local:docker`
    - \* if it does not work, try `xhost +`
      - if this does not work, nothing can be done
- **you get error of not being in sudo group when running image**
  - check output of `id -u` command. If the output is not 1000 you have to build the image by yourself and can not pull it
    - \* this happens when your account is not the first one created on your computer
- **``sudo apt install something`` does not work**

- you need to run `sudo apt update` first after you run the container for the first time
  - \* apt things are removed in Dockerfile, so it does not take unnecessary space in the image





## ICUB\_PYBULLET

### 2.1 pyCub

**class** pycub.**EndEffector**(*name, client*)

Bases: object

Help function for end-effector encapsulation

**Parameters**

- **name** (*str*) – name of the end-effector
- **client** (*pointer to pyCub instance*) – parent client

**get\_position()**

Function to get current position of the end-effector

**class** pycub.**Joint**(*name, robot\_joint\_id, joints\_id, lower\_limit, upper\_limit, max\_force, max\_velocity*)

Bases: object

Help class to encapsulate joint information

**Parameters**

- **name** (*str*) – name of the joint
- **robot\_joint\_id** (*int*) – id of the joint in pybullet
- **joints\_id** (*int*) – id of the joint in pycub.joints
- **lower\_limit** (*float*) – lower limit of the joint
- **upper\_limit** (*float*) – upper limit of the joint
- **max\_force** (*float*) – max force of the joint
- **max\_velocity** (*float*) – max velocity of the joint

**class** pycub.**Link**(*name, robot\_joint\_id, urdf\_link*)

Bases: object

Help function to encapsulate link information

**Parameters**

- **name** (*str*) – name of the link
- **robot\_joint\_id** (*int*) – id of the link in pybullet
- **urdf\_link** (*int*) – id of the link in pycub.urdfs[“robot”].links

```
class pycub.pyCub(config='default.yaml')
```

Bases: BulletClient

Client class which inherits from BulletClient and contains the whole simulation functionality

**Parameters**

**config** (*str*, *optional*, *default*="default.yaml") – path to the config file

```
static bbox_overlap(b1_min, b1_max, b2_min, b2_max)
```

```
compute_skin()
```

Function to emulate skin activations using ray casting.

```
contactPoints = {'DISTANCE': 8, 'FLAG': 0, 'FORCE': 9, 'FRICTION1': 10,
'FRICTION2': 12, 'FRICTIONDIR1': 11, 'FRICTIONDIR2': 13, 'IDA': 1, 'IDB': 2,
'INDEXA': 3, 'INDEXB': 4, 'NORMAL': 7, 'POSITIONA': 5, 'POSITIONB': 6}
```

```
create_urdf(object_path, fixed, color, suffix="")
```

Creates a URDF for the given .obj file

**Parameters**

- **object\_path** (*str*) – path to the .obj
- **fixed** (*bool*) – whether the object is fixed in space
- **color** (*list of 3 floats*) – color of the object

```
dynamicsInfo = {'BODYTYPE': 10, 'DAMPING': 8, 'FRICTION': 1, 'INERTIAOR': 4,
'INERTIAPOS': 3, 'INTERTIADIAGONAL': 2, 'MARGIN': 11, 'MASS': 0, 'RESTITUTION': 5,
'ROLLINGFRICTION': 6, 'SPINNINGFRICTION': 7, 'STIFFNESS': 9}
```

```
find_joint_id(joint_name)
```

Help function to get indexes from joint name of joint index in self.joints list

**Parameters**

**joint\_name** (*str or int*) – name or index of the link

**Returns**

joint id in pybullet and pycub space

**Return type**

int, int

```
find_link_id(mesh_name, robot=None, urdf_name='robot')
```

Help function to find link id from mesh name

**Parameters**

- **mesh\_name** (*str*) – name of the mesh (only basename with extension)
- **robot** (*int, optional, default=None*) – robot pybullet id
- **urdf\_name** (*str, optional, default="robot"*) – name of the object in pycub.urdfs

**Returns**

id of the link in pybullet space

**Return type**

int

**get\_camera\_images()**

Gets the images from enabled eye cameras

**Returns**

list of numpy arrays

**Return type**

list

**get\_joint\_state(joints=None)**

Get the state of the specified joints

**Parameters**

**joints** (*int or list, optional, default=None*) – joint or list of joints to get the state of

**Returns**

list of states of the joints

**Return type**

list

**init\_robot()**

Load the robot URDF and get its joints' information

**Returns**

robot and its joints

**Return type**

int or list

**is\_alive()**

Checks whether the engine is still running

**Returns**

True when running

**Return type**

bool

```
jointInfo = {'AXIS': 13, 'DAMPING': 6, 'FLAGS': 5, 'FRICTION': 7, 'INDEX': 0,
'LINKNAME': 12, 'LOWERLIMIT': 8, 'MAXFORCE': 10, 'MAXVELOCITY': 11, 'NAME': 1,
'PARENTINDEX': 16, 'PARENTORN': 15, 'PARENTPOS': 14, 'QINDEX': 3, 'TYPE': 2,
'UINDEX': 4, 'UPPERLIMIT': 9}
```

```
jointStates = {'FORCES': 2, 'POSITION': 0, 'TORQUE': 3, 'VELOCITY': 1}
```

**kill\_open3d()**

```
linkInfo = {'ANGVEL': 7, 'INERTIAORI': 3, 'INERTIAPOS': 2, 'LINVEL': 6, 'URDFORI':
5, 'URDFPOS': 4, 'WORLDORI': 1, 'WORLDPOS': 0}
```

**motion\_done(joints=None, check\_collision=True)**

Checks whether the motion is done.

**Parameters**

- **joints** (*int or list, optional, default=None*) – joint or list of joints to get the state of
- **check\_collision** (*bool, optional, default=True*) – whether to check for collision during motion

**Returns**

True when motion is done, false otherwise

**Return type**

bool

**move\_cartesian**(*pose, wait=True, velocity=1, check\_collision=True*)

Move the robot in cartesian space by computing inverse kinematics and running position control

**Parameters**

- **pose** (*utils.Pose*) – desired pose of the end effector
- **wait** (*bool, optional, default=True*) – whether to wait for movement completion
- **velocity** (*float, optional, default=1*) – joint velocity to move with
- **check\_collision** (*bool, optional, default=True*) – whether to check for collisions during motion

**move\_position**(*joints, positions, wait=True, velocity=1, set\_col\_state=True, check\_collision=True*)

Move the specified joints to the given positions

**Parameters**

- **joints** (*int, list, str*) – joint or list of joints to move
- **positions** (*float or list*) – position or list of positions to move the joints to
- **wait** (*bool, optional, default=True*) – whether to wait until the motion is done
- **velocity** (*float, optional, default=1*) – velocity to move the joints with
- **set\_col\_state** (*bool, optional, default=True*) – whether to reset collision state
- **check\_collision** (*bool, optional, default=True*) – whether to check for collision during motion

**move\_velocity**(*joints, velocities*)

Move the specified joints with the specified velocity IT IS HERE, BUT NOT IN WORKING STATE

**Parameters**

- **joints** (*int or list*) – joint or list of joints to move
- **velocities** (*float or list*) – velocity or list of velocities to move the joints to

**prepare\_log**()

Prepares the log string

**Returns**

log string

**Return type**

str

**print\_collision\_info**(*c=None*)

Help function to print collision info

**Parameters**

**c** (*list, optional, default=None*) – one collision

**run\_vhacd**()

Function to run VHACD on all objects in loaded URDFs, and to create new URDFs with changed collision meshes

**static scale\_bbox**(*bbox*, *scale*)

**stop\_robot**()

Stops the robot

**toggle\_gravity**()

Toggles the gravity

**update\_simulation**(*sleep\_duration=0.01*)

Updates the simulation

**Parameters**

**sleep\_duration** (*float*, *optional*, *default=0.01*) – duration to sleep before the next simulation step

**visualShapeData** = {'COLOR': 7, 'DIMS': 3, 'FILE': 4, 'GEOMTYPE': 2, 'ID': 0, 'LINK': 1, 'ORI': 6, 'POS': 5, 'TEXTURE': 8}

**wait\_motion\_done**(*sleep\_duration=0.01*, *check\_collision=True*)

Help function to wait for motion to be done. Can sleep for a specific duration

**Parameters**

- **sleep\_duration** (*float*, *optional*, *default=0.01*) – how long to sleep before running simulation step
- **check\_collision** (*bool*, *optional*, *default=True*) – whether to check for collisions during motion

## 2.2 utils

**class** `utils.Config`(*config\_path*)

Bases: `object`

Class to parse and keep the config loaded from yaml file

**Parameters**

**config\_path** (*str*) – path to the config file

**set\_attribute**(*attr*, *value*, *reference*)

Function to recursively fill the instance variables from dictionary. When value is non-dict, it is directly assigned to a variable. Else, the dict is recursively parsed.

**Parameters**

- **attr** (*str*) – name of the attribute
- **value** (*str*, *float*, *int*, *dict*, *list*, ... - *and other that can be loaded from yaml*) – value of the attribute
- **reference** (*pointer or whatever it is called in Python*) – reference to the parent class. “self” for the upper attributes, pointer to namedtuple for inner attributes

**Returns**

0

**Return type**

int

```
class utils.CustomFormatter(fmt=None, datefmt=None, style='%', validate=True, *, defaults=None)
```

Bases: `Formatter`

Custom formatter that assigns colors to logs From <https://stackoverflow.com/a/56944256>

Initialize the formatter with specified format strings.

Initialize the formatter either with the specified format string, or a default as described above. Allow for specialized date formatting with the optional `datefmt` argument. If `datefmt` is omitted, you get an ISO8601-like (or RFC 3339-like) format.

Use a style parameter of `'%'`, `'{'` or `'$'` to specify that you want to use one of `%`-formatting, `str.format()` (`{}`) formatting or `string.Template` formatting in your format string.

Changed in version 3.2: Added the `style` parameter.

```
FORMATS = {10:  '\x1b[38;20m%(module)s %(levelname)s: %(message)s\x1b[0m', 20:
'\x1b[38;20m%(module)s %(levelname)s: %(message)s\x1b[0m', 30:
'\x1b[33;20m%(module)s %(levelname)s: %(message)s\x1b[0m', 40:
'\x1b[31;20m%(module)s %(levelname)s: %(message)s\x1b[0m', 50:
'\x1b[31;1m%(module)s %(levelname)s: %(message)s\x1b[0m'}
```

```
bold_red = '\x1b[31;1m'
```

```
format(record)
```

Format the specified record as text.

The record's attribute dictionary is used as the operand to a string formatting operation which yields the returned string. Before formatting the dictionary, a couple of preparatory steps are carried out. The message attribute of the record is computed using `LogRecord.getMessage()`. If the formatting string uses the time (as determined by a call to `usesTime()`, `formatTime()` is called to format the event time. If there is exception information, it is formatted using `formatException()` and appended to the message.

```
grey = '\x1b[38;20m'
```

```
red = '\x1b[31;20m'
```

```
reset = '\x1b[0m'
```

```
yellow = '\x1b[33;20m'
```

```
class utils.Pose(pos, ori)
```

Bases: `object`

Mini help class for Pose representation

Init function that takes position and orientation and saves them as attributes

#### Parameters

- **pos** (*list*) – x,y,z position
- **ori** (*list*) – rpy orientation

```
class utils.URDF(path)
```

Bases: `object`

Class to parse URDF file

#### Parameters

**path** (*str*) – path to the URDF file

**ROOT\_TAGS** = []

**dereference()**

Make parent/child again as names to allow urdf write

**find\_root\_tags()**

Finds tags that are 'root', i.e., they have child 'inside'

**fix\_urdf()**

Fix the URDF file by converting non-mesh geometries to mesh and saving them as .obj files. If changes were made, write the new URDF to a file.

**make\_references()**

Make parent/child in joint list as references to the given link

**read(*el*, *parent*)**

Recursive function to read the URDF file. When there are no children, it reads the attributes and saves them.

#### Parameters

- **el** (*xml.etree.ElementTree.Element*) – The current element in the XML tree.
- **parent** (*xml.etree.ElementTree.Element*) – The parent element in the XML tree.

**write\_attr(*attr\_name*, *attr*, *level*=1, *skip\_header*=False)**

Write an attribute to the new URDF string.

#### Parameters

- **attr\_name** (*str*) – The name of the attribute.
- **attr** (*any*) – The attribute value.
- **level** (*int*, *optional*, *default*=1) – The indentation level for the attribute.
- **skip\_header** (*bool*, *optional*, *default*=False) – Whether to skip writing the attribute header.

**write\_urdf()**

Write the URDF object to a string.

## 2.3 visualizer

**class visualizer.Visualizer(*client*=None)**

Bases: object

Class to help with custom rendering

#### Parameters

- **client** (*int*, *optional*, *default*=None) – The client to be used for the visualizer.

**class EyeWindow(*eye*, *parent*)**

Bases: object

Class to handle windows for eye rendering

#### Parameters

- **eye** (*str*) – name of the eye



- **parent** (*int*) – The parent class (Visualizer).

**MENU\_IDS** = {'l\_eye': [2, 3, 8], 'r\_eye': [4, 5, 9]}

**POSITIONS** = {'l\_eye': [320, 560], 'r\_eye': [0, 560]}

**get\_image()**

**on\_close()**  
Small function to delete the window from the parent class

**on\_mouse(event)**  
Small function to ignore mouse events

**Parameters**  
**event** (*gui.MouseEvent*) – Mouse event

**save\_image(im)**  
Callback to get images from open3d

**Parameters**  
**im** (*o3d.geometry.Image*) – the image to be saves

**save\_images()**  
Function to save stream of images to file

**class MenuCallback(menu\_id, parent)**  
Bases: object  
Class to handle menu callbacks.  
Initialize the MenuCallback class.

**Parameters**

- **menu\_id** (*int*) – The id of the menu.
- **parent** (*pointer to the class of visualizer.Visualizer type*) – The parent class (Visualizer).

**input\_completed(text=None)**

**save\_image(im, mode)**  
Save the image. It shows FileDialog to find path for image save. It saves it with the current resolution of the window.

**Parameters**

- **im** (*open3d.geometry.Image*) – The image to be saved.
- **mode** (*int*) – The mode of the image. 0 for RGB, 1 for depth.

**wait\_for\_dialog\_completion()**  
Help function to keep the gui loop running

**find\_xyz\_rpy(mesh\_name, urdf\_name='robot')**  
Find the xyz, rpy and scales values.

**Parameters**

- **mesh\_name** (*str*) – The name of the mesh.
- **urdf\_name** (*str, optional, default="robot"*) – The name of the urdf.

**Returns**  
The xyz, rpy, and scales, link\_name

**read\_info(*obj\_id*)**

Read info from PyBullet

**Parameters**

**obj\_id** (*int*) – id of the object; given by pybullet

**Returns**

0 for success

**Return type**

int

**render()**

Render all the things

**show\_first(*urdf\_name*='robot')**

Show the first batch of meshes in the visualizer. It loads the meshes and saves the to dict for quicker use later

**Parameters**

**urdf\_name** (*str, optional, default="robot"*) – The name of the urdf to be used.

**show\_mesh()**

Function to parse info about meshes from PyBullet



## EXAMPLES

### 3.1 push\_the\_ball\_cartesian

Example of moving the robot in cartesian space to push the ball. It is more robust than the pure joint control.

**Author**

Lukas Rustler

`push_the_ball_cartesian.push_the_ball()`

Function to move the ball with cartesian control. The robot is moved 15cm lower and 10cm closer and then moved left to push the ball.

### 3.2 push\_the\_ball\_pure\_joints

Example of how to push the ball from the table using only pure joint control. It works without planner or collision detection/avoidance. It is not very robust, and it is laborious, but it is a good starting point for your own experiments.

**Author**

Lukas Rustler

`push_the_ball_pure_joints.push_the_ball()`

Function to push the ball from the table

### 3.3 skin\_test

Script to test the skin sensors.

**Author**

Lukas Rustler



## PYTHON MODULE INDEX

### p

`push_the_ball_cartesian`, [17](#)  
`push_the_ball_pure_joints`, [17](#)  
`pycub`, [7](#)

### s

`skin_test`, [17](#)

### u

`utils`, [11](#)

### v

`visualizer`, [13](#)



## B

`bbox_overlap()` (*pycub.pyCub static method*), 8  
`bold_red` (*utils.CustomFormatter attribute*), 12

## C

`compute_skin()` (*pycub.pyCub method*), 8  
`Config` (*class in utils*), 11  
`contactPoints` (*pycub.pyCub attribute*), 8  
`create_urdf()` (*pycub.pyCub method*), 8  
`CustomFormatter` (*class in utils*), 11

## D

`dereference()` (*utils.URDF method*), 13  
`dynamicsInfo` (*pycub.pyCub attribute*), 8

## E

`EndEffector` (*class in pycub*), 7

## F

`find_joint_id()` (*pycub.pyCub method*), 8  
`find_link_id()` (*pycub.pyCub method*), 8  
`find_root_tags()` (*utils.URDF method*), 13  
`find_xyz_rpy()` (*visualizer.Visualizer method*), 14  
`fix_urdf()` (*utils.URDF method*), 13  
`format()` (*utils.CustomFormatter method*), 7  
`FORMATS` (*utils.CustomFormatter attribute*), 12

## G

`get_camera_images()` (*pycub.pyCub method*), 8  
`get_image()` (*visualizer.Visualizer.EyeWindow method*), 14  
`get_joint_state()` (*pycub.pyCub method*), 9  
`get_position()` (*pycub.EndEffector method*), 7  
`grey` (*utils.CustomFormatter attribute*), 12

## I

`init_robot()` (*pycub.pyCub method*), 9  
`input_completed()` (*visualizer.Visualizer.MenuCallback method*), 14  
`is_alive()` (*pycub.pyCub method*), 9

## J

`Joint` (*class in pycub*), 7  
`jointInfo` (*pycub.pyCub attribute*), 9  
`jointStates` (*pycub.pyCub attribute*), 9

## K

`kill_open3d()` (*pycub.pyCub method*), 9

## L

`Link` (*class in pycub*), 7  
`linkInfo` (*pycub.pyCub attribute*), 9

## M

`make_references()` (*utils.URDF method*), 13  
`MENU_IDS` (*visualizer.Visualizer.EyeWindow attribute*), 14  
`module`  
    `push_the_ball_cartesian`, 17  
    `push_the_ball_pure_joints`, 17  
    `pycub`, 7  
    `skin_test`, 17  
    `utils`, 11  
    `visualizer`, 13  
`motion_done()` (*pycub.pyCub method*), 9  
`move_cartesian()` (*pycub.pyCub method*), 10  
`move_position()` (*pycub.pyCub method*), 10  
`move_velocity()` (*pycub.pyCub method*), 10

## O

`on_close()` (*visualizer.Visualizer.EyeWindow method*), 14  
`on_mouse()` (*visualizer.Visualizer.EyeWindow method*), 14

## P

`Pose` (*class in utils*), 12  
`POSITIONS` (*visualizer.Visualizer.EyeWindow attribute*), 14  
`prepare_log()` (*pycub.pyCub method*), 10  
`print_collision_info()` (*pycub.pyCub method*), 10  
`push_the_ball()` (*in module push\_the\_ball\_cartesian*), 17



`push_the_ball()` (in `push_the_ball_pure_joints`), 17  
`push_the_ball_cartesian`  
    module, 17  
`push_the_ball_pure_joints`  
    module, 17  
`pycub`  
    module, 7  
`pyCub` (class in `pycub`), 7

## R

`read()` (*utils.URDF method*), 13  
`read_info()` (*visualizer.Visualizer method*), 14  
`red` (*utils.CustomFormatter attribute*), 12  
`render()` (*visualizer.Visualizer method*), 15  
`reset` (*utils.CustomFormatter attribute*), 12  
`ROOT_TAGS` (*utils.URDF attribute*), 12  
`run_vhacd()` (*pycub.pyCub method*), 10

## S

`save_image()` (*visualizer.Visualizer.EyeWindow method*), 14  
`save_image()` (*visualizer.Visualizer.MenuCallback method*), 14  
`save_images()` (*visualizer.Visualizer.EyeWindow method*), 14  
`scale_bbox()` (*pycub.pyCub static method*), 10  
`set_attribute()` (*utils.Config method*), 11  
`show_first()` (*visualizer.Visualizer method*), 15  
`show_mesh()` (*visualizer.Visualizer method*), 15  
`skin_test`  
    module, 17  
`stop_robot()` (*pycub.pyCub method*), 11

## T

`toggle_gravity()` (*pycub.pyCub method*), 11

## U

`update_simulation()` (*pycub.pyCub method*), 11  
`URDF` (class in *utils*), 12  
`utils`  
    module, 11

## V

`visualizer`  
    module, 13  
`Visualizer` (class in *visualizer*), 13  
`Visualizer.EyeWindow` (class in *visualizer*), 13  
`Visualizer.MenuCallback` (class in *visualizer*), 14  
`visualShapeData` (*pycub.pyCub attribute*), 11

## W

`wait_for_dialog_completion()` (*visualizer.Visualizer.MenuCallback method*), 14

## Y

`yellow` (*utils.CustomFormatter attribute*), 12