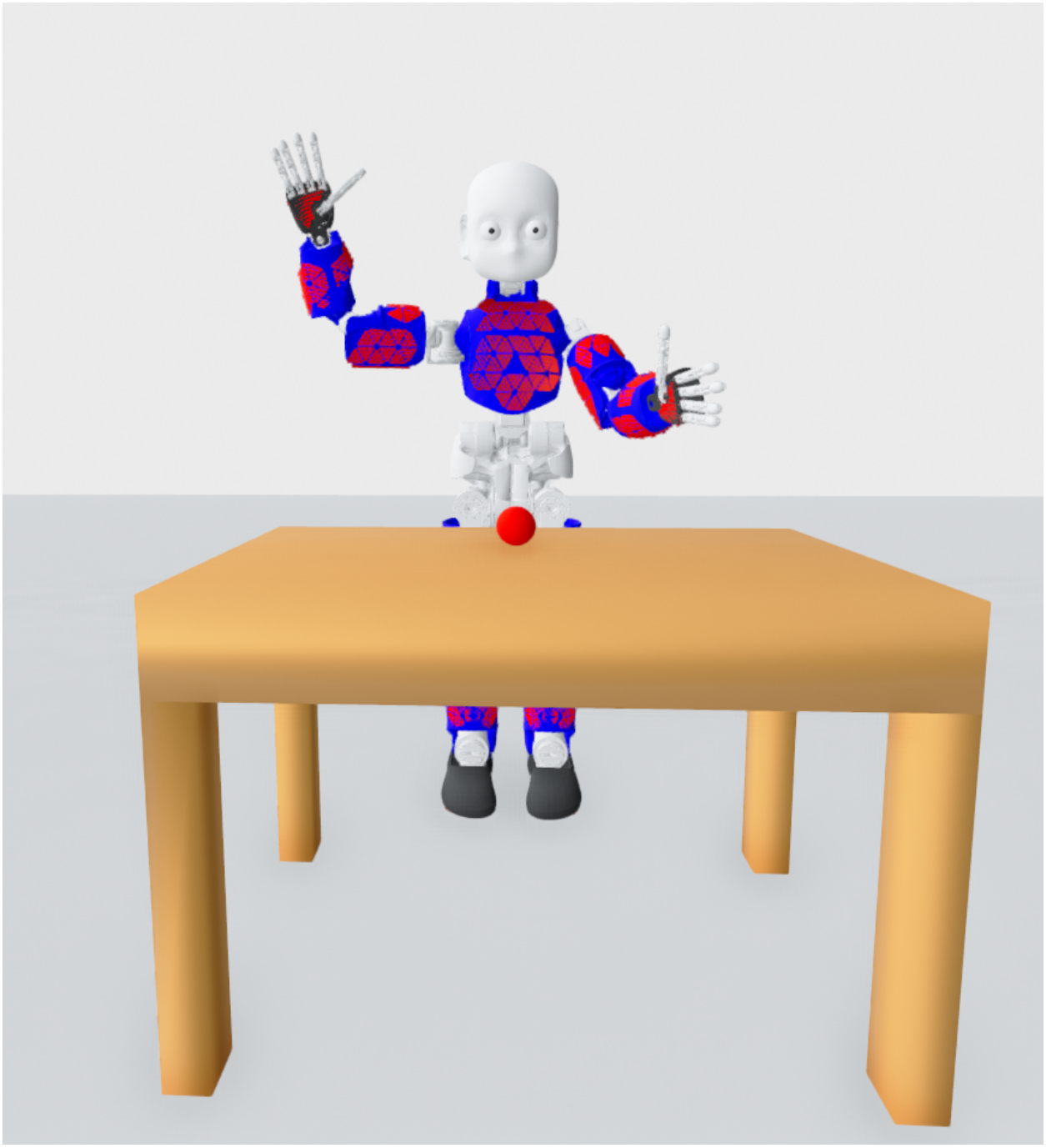


pyCub Studio Handbook

Lukas Rustler

Nov 26, 2026

CONTENTS

1 Using Studio **3**
- 1.1 Getting started . . . 3
- 1.2 Kid GUI . . . 3
- 1.3 Scratch blocks . . . 3
- 1.4 Kid Programming . . . 3
- 1.5 Normal Mode . . . 4
- 1.6 Classroom web sessions . . . 4
- 1.7 Workspace and packages . . . 4

2 Programming the robot **5**
- 2.1 Kid modes (`icub_pybullet.kid_api`) . . . 5
- 2.2 Studio Full Mode (Normal Mode IDE) . . . 7
- 2.3 Normal / Standalone Python mode (outside Studio) . . . 8
- 2.4 Inspecting calls in the editor . . . 9
- 2.5 Motion, kinematics, and sensor reference . . . 10

3 Example controllers **11**
- 3.1 Joint control . . . 11
- 3.2 Cartesian control . . . 12
- 3.3 Skin contacts . . . 12
- 3.4 Minimal simulation loop . . . 13

4 Course exercises **15**
- 4.1 Push the Ball . . . 15
- 4.2 Smooth movements . . . 16
- 4.3 Gaze control . . . 18
- 4.4 Resolved-Rate Motion Control . . . 20
- 4.5 Grasp It! . . . 22

5 Project README **25**
- 5.1 pyCub Studio (Standalone Desktop Application) . . . 26
- 5.2 Building `pyCub Studio` (Native Installers & Bundles) . . . 28
- 5.3 Python Package Installation (`icub_pybullet` & `pycub-studio`) . . . 30
- 5.4 Documentation & Building the Handbook . . . 31
- 5.5 FAQ & Known Issues . . . 31
- 5.6 Docker . . . 32
- 5.7 Citing . . . 33
- 5.8 License . . . 33

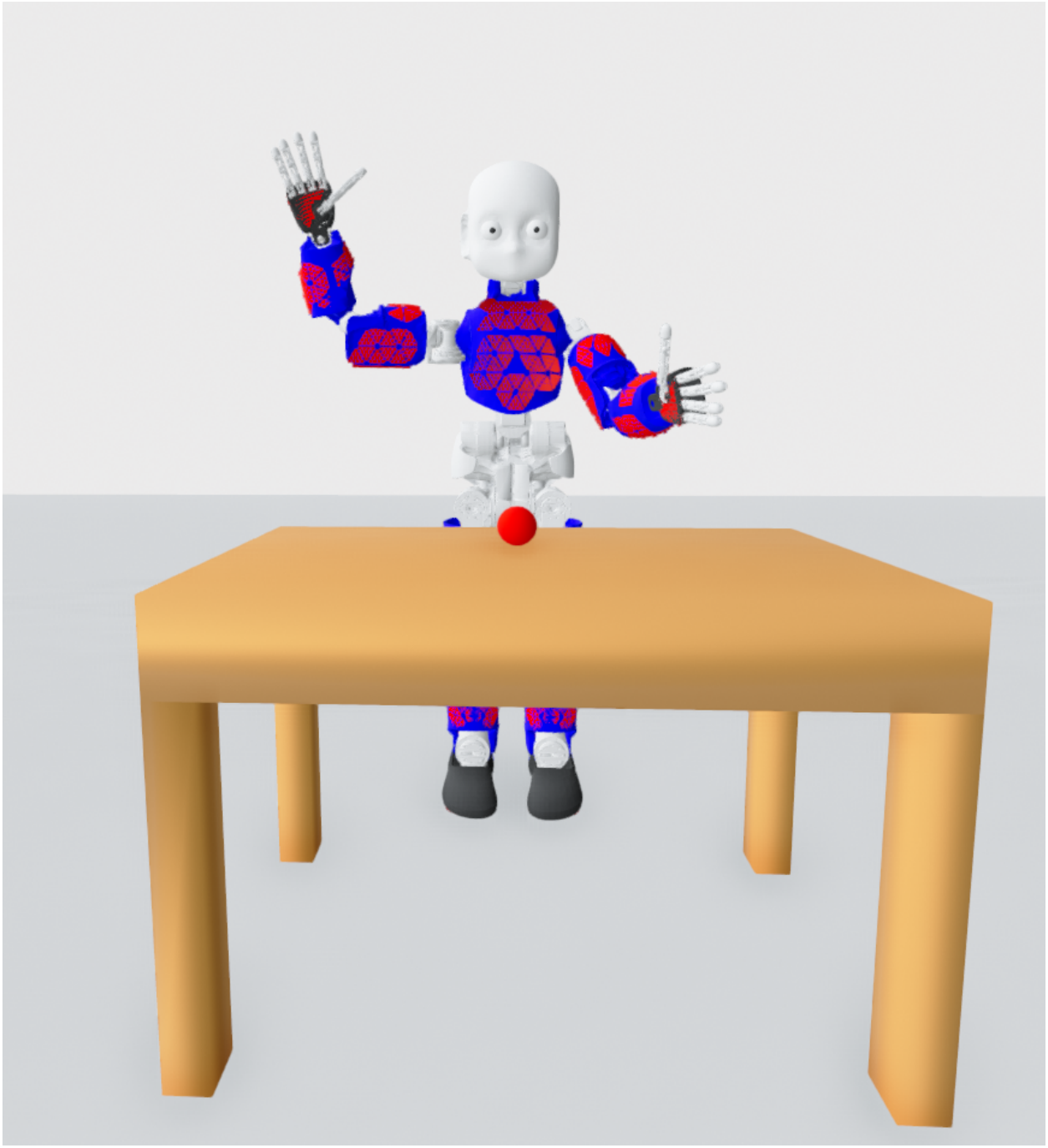
6 API Reference **35**
- 6.1 skin_test . . . 35

6.2 icub_pybullet 35
6.3 push_the_ball_cartesian 54
6.4 push_the_ball_pure_joints 55

Python Module Index **57**

Index **59**

Learn to control the iCub humanoid in simulation. Start with a wave, write your own controller, then test it against the course exercises.



- *Using Studio*

  Choose a mode, set up your workspace and run a simulation.

- *Programming the robot*

  Move joints, read sensors and inspect Python calls.

- *Example controllers*

Follow working joint, Cartesian and skin examples.

- *Course exercises*

  Practice motion control, gaze and grasping.

- *Project README*

  Installation, Docker setup, features overview and citation.

- *API reference*

  Look up classes, functions and their arguments.

CHAPTER

ONE

USING STUDIO

Launch pyCub Studio and choose a mode from Home. The mode selector in the header lets you switch activities. The language button switches between English and Czech.

1.1 Getting started

Open the desktop application from its installer or portable distribution, or install the Python packages into a Python 3.10–3.12 virtual environment and launch `pycub-studio` from anywhere:

```
# From PyPI (either command installs simulator + Studio + handbook + exercises):
python -m pip install pycub-studio
# or: python -m pip install "icub_pybullet[studio]"

# Or from a local repository checkout:
python -m pip install ./icub_pybullet ./studio

pycub-studio
```

1.2 Kid GUI

Choose a world with a ball, then try a preset movement. Use the joint buttons to move one joint at a time. Adjust the step size for finer motion. The ball readout shows whether the robot touched the ball and how far it moved.

1.3 Scratch blocks

Snap movement, sensor and control blocks together and run the program. Watch the highlighted block as the simulator steps through it. Use the Python preview to see how a block translates to code. Save editable projects as `.pycub_blocks`, or open the generated Python in Kid Programming.

1.4 Kid Programming

Start with a template or click a command in the palette to insert an example. Run the program and read its output in the console. Click Stop to interrupt a loop. A reported error highlights the line in the editor. See the *Python guide* for call hints and definitions.

1.5 Normal Mode

Open a Python file, or start with the script in the first tab. Press F5 to run the current Python script. Use YAML tabs to edit world configurations. When working in another editor, open an external script and enable run-on-save to rerun it after changes. Console output belongs to the script worker.

Use the left and right eye views to inspect RGB and depth images. Use the examples menu for a working controller, or select a course exercise to copy its files to your workspace. Run the tester from the exercise tab and inspect the results report.

1.6 Classroom web sessions

Select the number of simulator instances and start the server in Classroom Web GUI. Students on the same network can open the displayed landing URL or scan the QR code. The dashboard shows connected users and instance status. Open the logs to investigate a failed instance or restart it individually.

1.7 Workspace and packages

Open **Workspace** from the header to locate saved scripts, blocks, examples, exercises and images. Store custom YAML files in `worlds/`. A world can inherit built-in settings with `parent: with_ball.yaml`. Put custom geometry in `meshes/`. Use **Packages / Python** to install extra packages or select an external interpreter for Normal Mode scripts.

CHAPTER

TWO

PROGRAMMING THE ROBOT

pyCub supports three main programming workflows depending on your experience level and whether you want Studio's integrated 3D interface or a standalone Python environment:

Workflow	Entry API	Units	Simulation loop	Best for
Kid modes (Scratch & Kid Programming in Studio)	`icub_pybull kid_api`	**Degrees**	Managed automatically	Beginners, classrooms, quick experiments
Studio Full Mode (Normal Mode IDE in Studio)	`icub_pybull pycub. pyCub`	**Radians**	Explicit (`wait=True` or `update_simulation`	Full robot control, cameras, skin, course exercises
Normal / Standalone Python (Terminal / PyCharm / VS Code)	`pyCub` or `kid_api`	Radians (`pyCub`) / Degrees (`kid_api`)	Explicit (`pyCub`) / Managed (`kid_api`)	Automated evaluation, headless runs, external Open3D/Web GUI

*(Note: inside `.yaml` world files, `initial_joint_angles` are always written in **degrees**, regardless of mode.)*

2.1 Kid modes (`icub_pybullet.kid_api`)

The Kid modes inside **pyCub Studio** — **Kid "Scratch" (Blocks)** and **Kid Programming (Python)** — both run on top of `icub_pybullet.kid_api`.

2.1.1 How Kid execution works

- Studio keeps an interactive physics worker running continuously in the background for the currently selected world (`with_ball.yaml`, `default.yaml`, `skin_test.yaml`, or your custom `.yaml` world in `~/Documents/pyCub/worlds/`).
- You do **not** instantiate `pyCub(...)` and do **not** write a `while client.is_alive(): client.update_simulation()` loop.
- Every movement command blocks until the motion finishes (or reaches its timeout), stepping the simulation smoothly and checking the **Stop** button cooperatively so infinite loops can always be interrupted immediately.

2.1.2 From Scratch blocks to Python

In **Kid "Scratch"**, snap movement, loop, logic, and sensor blocks together. Toggle **Python Preview** to watch the live Python translation as you edit blocks, save reusable projects as `.pycub_blocks`, or click **Open as Python in Kid Programming** to export the generated script directly into the text Python editor.

2.1.3 First program in Kid Programming

Select a world containing a ball (such as *Table with a ball* / `with_ball.yaml`) and run:

```
from icub_pybullet.kid_api import wave, shoulder_up, look_down, ball_touched, ball_
↪distance

look_down(20)
wave("right")
shoulder_up("left", degrees=20, speed=1.5)
print("Touched:", ball_touched(), "Distance (cm):", ball_distance())
```

Studio automatically exposes all English and Czech `kid_api` commands for autocomplete and inline help, and both spellings share the exact same underlying controller. The equivalent Czech script is:

```
from icub_pybullet.kid_api import podívej_se_dolů, zamávej, zvedni_rameno, dotkl_se_
↪míčku, vzdálenost_míčku

podívej_se_dolů(20)
zamávej("pravá")
zvedni_rameno("levá", degrees=20, speed=1.5)
print("Dotyk:", dotkl_se_míčku(), "Vzdálenost (cm):", vzdálenost_míčku())
```

Keyword arguments always use the Python parameter names (`side`, `degrees`, `speed`, `target_degrees`, `delta_degrees`, `seconds`).

2.1.4 Kid command groups

- **Preset actions:** `wave(side="right")`, `left_wave()`, `right_wave()`, `both_arms_up()`, `both_arms_down()`, `look_center()`, `stand_straight()`, `reset_pose()`, `stop()`.
- **Relative arm moves (degrees):** `shoulder_up`, `shoulder_down`, `shoulder_out`, `shoulder_in`, `rotate_arm`, `bend_elbow`, `straighten_elbow`, `rotate_wrist`. All accept `side` (`"left"` / `"right"` or `"levá"` / `"pravá"`), `degrees=15.0`, and `speed=1.5` (rad/s).
- **Relative head & torso moves (degrees):** `turn_head`, `look_up`, `look_down`, `tilt_head`, `turn_body`, `bend_forward`, `bend_back`, `bend_sideways`.
- **Direct joint access & timing:** `move_joint(joint_name, target_degrees, speed=1.5)` for an absolute target angle, `jog_joint(joint_name, delta_degrees, speed=1.5)` for a signed relative offset, `get_joint(joint_name)` to read the current angle in degrees, and `wait(seconds=1.0)` to pause while physics keeps running.
- **Sensors & telemetry:** `ball_distance()` (distance between ball and table in centimetres), `ball_touched()` (`bool`), `ball_hits()` (`int`), and `skin_touched()` (`bool`, active when `skin.use: True`).

Click any entry in the **Command Palette** on the left to insert a working snippet at the cursor, or load a starter template from the toolbar. Syntax and runtime errors highlight the offending line in red and display a beginner-friendly hint in the console.

2.2 Studio Full Mode (Normal Mode IDE)

Normal Mode in **pyCub Studio** gives you the complete `icub_pybullet.pycub.pyCub` simulator API together with an embedded hardware-accelerated OpenGL 3D viewport, live console, eye cameras, and course exercise tools.

2.2.1 Running scripts in Studio Normal Mode

- Edit `.py` controllers and `.yaml` world configs across multiple tabs and press **F5** (or click **Run Script**) to execute the active tab.
- Each run starts in a clean isolated worker subprocess (`SimProcessSupervisor`) whose rendering streams directly into Studio's embedded 3D viewport—no separate Open3D window is opened.
- To code in PyCharm or VS Code while using Studio's 3D viewport and camera panels, click **Run External .py…** and enable **Auto-run external file on save**, or launch Studio directly from the command line:

```
python -m studio.main --run-script path/to/your_controller.py
```

2.2.2 Minimal pyCub controller structure

In Full / Normal mode, your script creates the `pyCub` client, selects a world `.yaml` config, commands motions in **radians**, and keeps the simulation alive at the end until you click **Stop**:

```
from icub_pybullet.pycub import pyCub

client = pyCub(config="with_ball.yaml")

# Blocking joint motion (waits until target is reached or collision/timeout occurs)
client.move_position("neck_pitch", -0.25)

# Non-blocking joint motion: advance simulation manually in your own loop
client.move_position(["torso_pitch", "torso_yaw"], [0.30, -0.60], wait=False, velocity=2.
↪0)
while not client.motion_done():
    client.update_simulation()

# Keep the 3D view responsive after your routine finishes
while client.is_alive():
    client.update_simulation()
```

2.2.3 Joint, velocity, and Cartesian control

- **Position control (`move_position`):** Pass a single joint name/ID or equal-length lists of joints and target angles in **radians**. Optional parameters: `wait=True`, `velocity=1` (rad/s), `set_col_state=True`, `check_collision=True`, and `timeout=None` (seconds). With `wait=False`, drive the simulation yourself with `client.update_simulation()` until `client.motion_done()` returns `True`, or call `client.wait_motion_done()`.
- **Velocity control (`move_velocity`):** Command target angular speeds in **radians per second** for one joint or a list of joints.
- **Reading joint state & stopping:** `client.get_joint_state(joints=None)` returns current joint positions in radians (defaulting to all movable joints); `client.stop_robot(joints=None)` halts active position or velocity targets.

- **Cartesian control (`move_cartesian` & `Pose`):** Read the current end-effector pose with `pose = client.end_effector.get_position()`, adjust `pose.pos` (`[x, y, z]` in metres) or `pose.ori` (quaternion `[x, y, z, w]`), and call `client.move_cartesian(pose, wait=True, velocity=1, check_collision=True, timeout=None)`. PyBullet computes the inverse kinematics solution and commands the movable joints automatically.

```
from icub_pybullet.pycub import pyCub, Pose

client = pyCub(config="with_ball.yaml")
cur = client.end_effector.get_position()
target = Pose(list(cur.pos), list(cur.ori))
target.pos[2] -= 0.15
target.pos[0] -= 0.10
client.move_cartesian(target, velocity=1.5)
```

- **Jacobians (`compute_jacobian`):** Compute the base-frame geometric Jacobian and active joint names for any kinematic chain (`"left_arm"`, `"right_arm"`, `"left_leg"`, `"right_leg"`, `"torso"`, `"head"`):

```
J, active_joints = client.compute_jacobian("right_arm", end="r_hand")
```

2.2.4 Eye cameras, tactile skin, and workspace tools

- **Eye cameras (`l_eye`, `r_eye`):** Toggle **Left Eye (`l_eye`)** or **Right Eye (`r_eye`)** in the Normal Mode toolbar, or set `eyes.l_eye: True` / `eyes.r_eye: True` in your `.yaml` config. In Python, call `client.get_camera_images(eyes)` for RGB arrays and `client.get_camera_depth_images(eyes)` for depth arrays. Use **Save RGB / Save Depth** to export PNGs to `~/Documents/pyCub/images/`.
- **Tactile skin:** Set `skin.use: True` in your config (see `skin_test.yaml`), step the simulation with `client.update_simulation()`, and inspect `client.skin_activations` and `client.activated_skin_points`.
- **Examples & Course Exercises:** Open the **Examples & Exercises** menu to load bundled scripts or copy Exercises `1-5` into `~/Documents/pyCub/exercises/`. When an exercise file is open, click **Run Exercise Tester** to run its automated verification suite and view the HTML report.
- **Extra packages & custom Python:** Use **Packages / Python** in the top bar to install PyPI wheels into `~/Documents/pyCub/extra_packages/`, attach `site-packages` from an existing virtual environment, or select an external Python executable for Normal Mode scripts.

2.3 Normal / Standalone Python mode (outside Studio)

You can also run any pyCub script directly from a terminal, PyCharm, or VS Code using an environment where `icub_pybullet` is installed (`python3 -m pip install icub_pybullet` or `python3 -m pip install ./icub_pybullet`), without launching Studio at all.

2.3.1 Visualizer backends (`gui` in `.yaml`)

When running outside Studio, `pyCub` reads the `gui` block of your `.yaml` config:

```
gui:
  standard: True   # Native Open3D 3D window
  web: False       # Browser-based visualizer (slower fallback)
```

- **Open3D desktop window (`standard: True, web: False`):** Default interactive 3D window when running locally.

- **Browser visualizer (`standard: False, web: True`):** Serves an interactive view in your web browser—useful over remote sessions or if host OpenGL drivers conflict with Open3D.
- **Headless mode (`standard: False, web: False`):** No 3D GUI is created. Best for automated tests, reinforcement learning, batch evaluation, and CI.

(Inside Studio Normal Mode, Studio automatically overrides these flags to route rendering into its embedded OpenGL viewport.)

2.3.2 Running `kid_api` scripts standalone

Scripts written for Kid Programming using `icub_pybullet.kid_api` also run directly as standard Python files:

```
python my_kid_script.py
```

On the first command call, `get_robot()` lazily creates `pyCub(config="with_ball.yaml")` and steps the simulation automatically.

2.3.3 Config resolution and running exercises from CLI

When you pass a relative config name such as `pyCub(config="with_ball.yaml")`, pyCub searches in order:

1. Direct / absolute path or relative to the current working directory.
2. Relative to the calling script's directory (and a `configs/` subfolder next to it).
3. Your user workspace (`~/Documents/pyCub/worlds/`, `~/Documents/pyCub/configs/`, `~/Documents/pyCub/`).
4. Built-in `icub_pybullet/configs/` (`default.yaml`, `with_ball.yaml`, `skin_test.yaml`).

Custom `.yaml` configs can inherit built-in settings via `parent: with_ball.yaml` and override only the fields you need (such as `urdfs`, `eyes`, `skin`, or `end_effector`).

To run an exercise tester directly from the command line outside Studio:

```
python exercises/exercise_1/exercise_1_tester.py
```

2.4 Inspecting calls in the editor

Both **Kid Programming** and **Normal Mode** in Studio include live Jedi-powered code inspection:

- **Hover documentation:** Hover the mouse over a function or class name to read its signature and docstring without moving the cursor. Hover help formats `:param:` entries and trailing inline parameter comments on separate lines.
- **Signature & parameter hints:** Pause with the cursor inside parentheses of a function call (or press **Ctrl+Shift+Space**) to view the active parameter and default values.
- **Go to Definition (`F12`):** Place the cursor on any symbol and press **F12** to view its source definition and docstring—working for both unsaved functions in your current buffer and `icub_pybullet` library code.

2.5 Motion, kinematics, and sensor reference

2.5.1 Quick comparison: Kid API vs Full pyCub API

Task	Kid API (`icub_pybullet.kid_api`)	Full pyCub API (`client = pyCub(...)`)
Target joint angle	`move_joint("neck_pitch", -15)` *(degrees)*	`client.move_position("neck_pitch", -0.26)` *(radians)*
Multi-joint move	Call sequentially or loop	`client.move_position(["torso_pitch", "torso_yaw"], [0.3, -0.5])`
Relative joint offset	`jog_joint("r_elbow", 20)` *(degrees)*	Read `get_joint_state(...)` + `move_position(...)`
Joint velocity	—	`client.move_velocity(joints, velocities)` *(rad/s)*
Cartesian IK move	—	`client.move_cartesian(pose)` *(Pose(pos, ori))*
Read joint angle	`get_joint("neck_pitch")` *(degrees)*	`client.get_joint_state("neck_pitch")` *(radians)*
Camera RGB / Depth	—	`client.get_camera_images(eyes)` / `client.get_camera_depth_images(eyes)`
Ball telemetry	`ball_distance()`, `ball_touched()`, `ball_hits()`	Inspect `client.other_objects` + `getClosestPoints` / `getContactPoints`
Tactile skin	`skin_touched()`	`client.skin_activations`, `client.activated_skin_points`
Step simulation	Handled internally inside each command	`client.update_simulation()` / `client.wait_motion_done()`

2.5.2 Key iCub joints and kinematic chains

- **Torso:** `torso_pitch`, `torso_roll`, `torso_yaw`
- **Head & eyes:** `neck_pitch`, `neck_roll`, `neck_yaw`, `eyes_tilt`, `l_eye_pan_joint`, `r_eye_pan_joint`
- **Left / right arms (`l_` / `r_`):** `*_shoulder_pitch`, `*_shoulder_roll`, `*_shoulder_yaw`, `*_elbow`, `*_wrist_prosup`, `*_wrist_pitch`, `*_wrist_yaw`, plus finger joints `*_hand_*`
- **Left / right legs (`l_` / `r_`):** `*_hip_pitch`, `*_hip_roll`, `*_hip_yaw`, `*_knee`, `*_ankle_pitch`, `*_ankle_roll`
- **Kinematic chains (`CHAINS`):** `"left_arm"`, `"right_arm"`, `"left_leg"`, `"right_leg"`, `"torso"`, `"head"`

The bundled *Example controllers* and *API reference* document all classes, method signatures, and default parameters in detail.

CHAPTER

THREE

EXAMPLE CONTROLLERS

Open these examples in Normal Mode to run or modify them. Each example creates its own client and selects a world. The excerpts below come from the scripts installed with Studio.

3.1 Joint control

Set individual joint targets to position the arm and push the ball.

```
def push_the_ball(client: pyCub) -> None:
    """
    Example function to push the ball from the table with joint control.

    :param client: instance of pyCub
    :type client: pyCub
    :return:
    :rtype:
    """
    """
    """


    # move torso pitch so the robot bends over the table
    client.move_position("torso_pitch", 0.325)

    # move torso to push the ball, but do now wait for completion. Hit it a bit faster
    client.move_position("torso_yaw", -0.7, wait=False, velocity=5)

    # wait manually
    while not client.motion_done():
        client.update_simulation()

    """
    Wait could be also achieved with:
    client.wait_motion_done()
    """

    client.logger.info("Moved the ball!")
```

3.2 Cartesian control

Move the end effector relative to its current pose.

```
def push_the_ball(client: pyCub) -> None:
    """
    Example function to move the ball with cartesian control.

    :param client: instance of pyCub
    :type client: pyCub
    :return:
    :rtype:
    """

    # Get current pose
    cur_pose = client.end_effector.get_position()

    # Copy, the pose and go 15cm lower and 10cm closer to the robot
    new_pose = Pose(cur_pose.pos, cur_pose.ori)
    new_pose.pos[2] -= 0.15
    new_pose.pos[0] -= 0.1

    # move
    client.move_cartesian(new_pose)

    # get current pose
    pose = client.end_effector.get_position()

    # assign straight to it to move hand left
    pose.pos[1] -= 0.2

    # move; do not wait for completion; and move a bit faster
    client.move_cartesian(pose, wait=False, velocity=2)

    # wait manually
    while not client.motion_done():
        client.update_simulation()

    """
    Wait could be also achieved with:
    client.wait_motion_done()
    """

    client.logger.info("Moved the ball!")
```

3.3 Skin contacts

Use the skin-test world to see which sensors activate as balls fall on the robot.

```
def main() -> NoReturn:
    """
    Main function to run the example
```

(continues on next page)

(continued from previous page)

```
    :return:
    :rtype:
    """
    client = pyCub(config="skin_test.yaml")

    while client.is_alive():
        client.update_simulation(None)
```

3.4 Minimal simulation loop

Download the starter script and open it in Normal Mode. Add your robot commands after creating the client.

```
from icub_pybullet.pycub import pyCub

if __name__ == "__main__":
    # load the robot with correct world/config
    client = pyCub(config="with_ball.yaml")

    # wait until the gui is closed
    while client.is_alive():
        client.update_simulation()
```

CHAPTER

FOUR

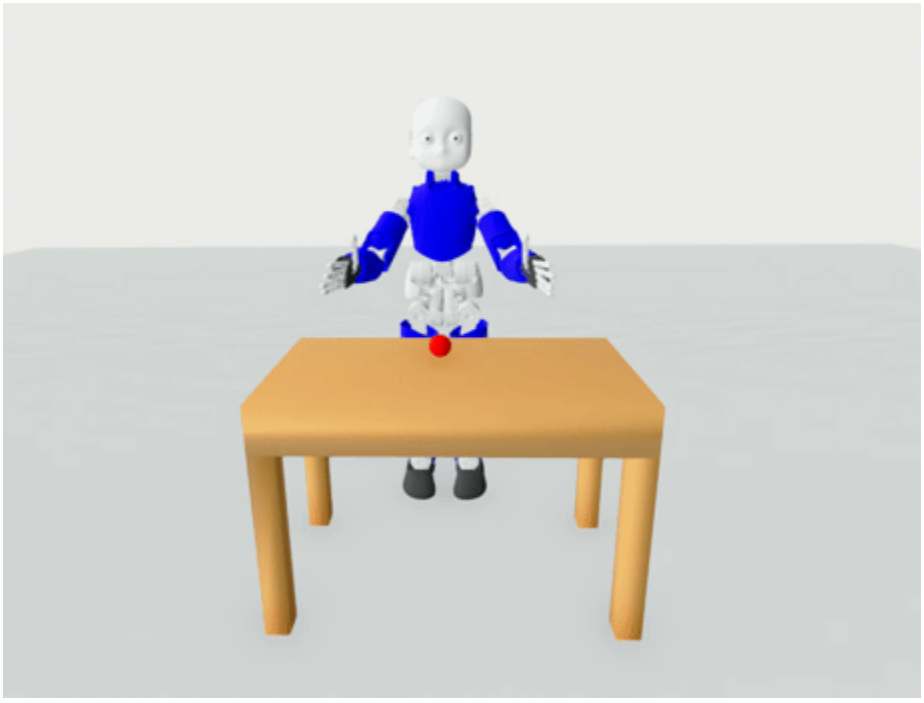
COURSE EXERCISES

Open an exercise from Normal Mode, edit its controller and run the exercise tester. Each task below describes the controller interface and scoring.

4.1 Push the Ball

The goal is to hit the ball to push it as far as possible from **any part** of the table. The ball will always spawn at the same place. The robot can be moved with position or cartesian control. The trajectories should be collision free and max allowed velocity is 10 (rad/s).

The resulting distance is checked after 2 seconds.



4.1.1 Task

Implement `push_the_ball` function in exercise_1.py that will push the ball as far as possible from the table.

- the function takes one argument:
  - client - instance of pycub class that controls the simulation
    - do not create new one in your code!

4.1.2 Scoring

- points for distance are computed as: min(3, distance*2)
  - i.e., you get max 3 points if you push the ball 1.5m away

4.1.3 Requirements:

Those apply mainly for exercise_1_tester.py **to work correctly**:

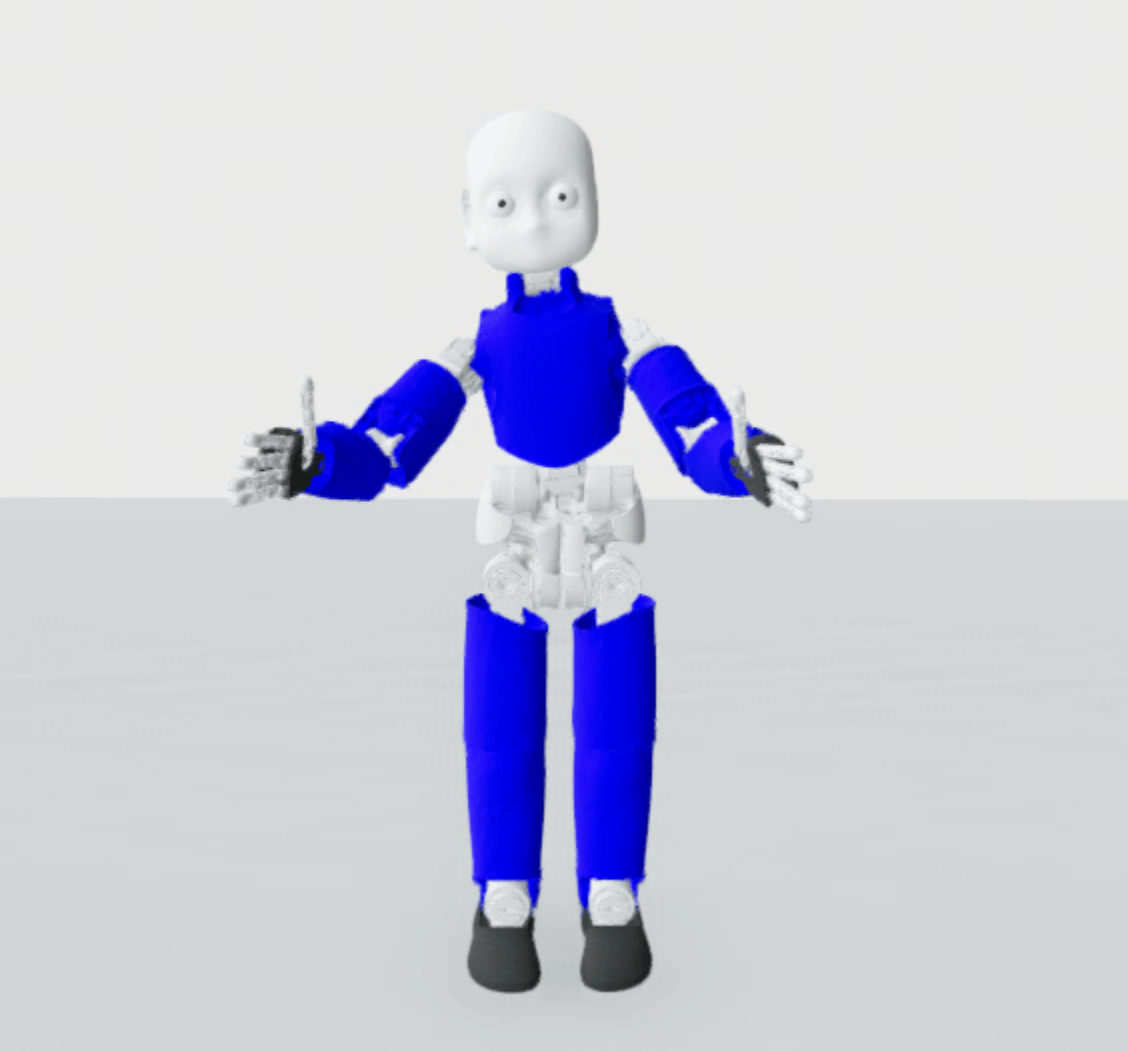
- do not create new client instance, use the one that is passed as an argument
- do not rename the function or file
- **Do not** introduce artificial delays in the code, e.g., sleep() or using update_simulation() after the movement is done

Those apply so that you fulfill the exercise as intended:

- Trajectories must be collision free with the table (self-collisions of the robot are allowed)
- Max allowed velocity is 10 (rad/s)
- **Do not** turn of gravity

4.2 Smooth movements

The task is to perform smooth movements with the arms of the robot. The robot should be able to move in a straight line or a circle with a smooth trajectory. In other words, the movement should be continuous and without jerks.



4.2.1 Task

Implement function move() in exercise_2.py that will move the robot's arm in a straight line or a circle.

- the function takes four arguments:
  - client - instance of pycub class that controls the simulation
  - action - string "line" or "circle"
  - axis - list of ints. For "circle" it the length is always 1. For "line" the length can be from 1-3. Individual numbers on the list are axes along which the robot should move. For example, [0, 1] means that the robot should move in x- and y-axis.
  - r - list of floats. The same length as axis. Number in metres. For "circle" it is the radius of the circle. For "line" it is the length of the line in the given axis.
- example arguments:
  - action="circle", axis=[0], r=[0.01] - the end-effector should move in a circle around the x-axis with a radius of 0.01m
  - action="line", axis=[1], r=[-0.05] - the end-effector should follow a line in the y-axis with a distance of -0.05m

  - action="line", axis=[0, 1], r=[0.05, -0.05] - the end-effector should follow a line in the x-axis for 0.05cm and y-axis for -0.05m
    * it should be simultaneous movement in both axes, i.e., it will be one line
- the function **have to** return two arguments: start and end pose of the trajectory
  - both should be obtained with `client.end_effector.get_position()`

4.2.2 Scoring

- you will be given 10 random test cases
  - each will be awarded with 1 points
- both circle and line movements will be evaluated with three sub-criteria:
  - circle:
    * standard deviation of each point of the circle to center; must be < r*0.125
    * difference between expected and real radius; must be < r*0.1
    * mean distance of all points from the expected plane; must be < 0.1
  - line:
    * difference from real length to expected length; must be < 0.0075
    * angle between the expected and real line; must be <0.1
    * mean distance of all points from the expected line; must be < 0.01
- **the smoothness of the movements will is not checked by the automatic evaluation tool!**
- **the code should be as general as possible to work for any allowed input.** Meaning that it should not be, for example, a long if/else (switch) statement for each axis etc.

4.2.3 Requirements:

Those apply mainly for exercise_2_tester.py **to work correctly**:

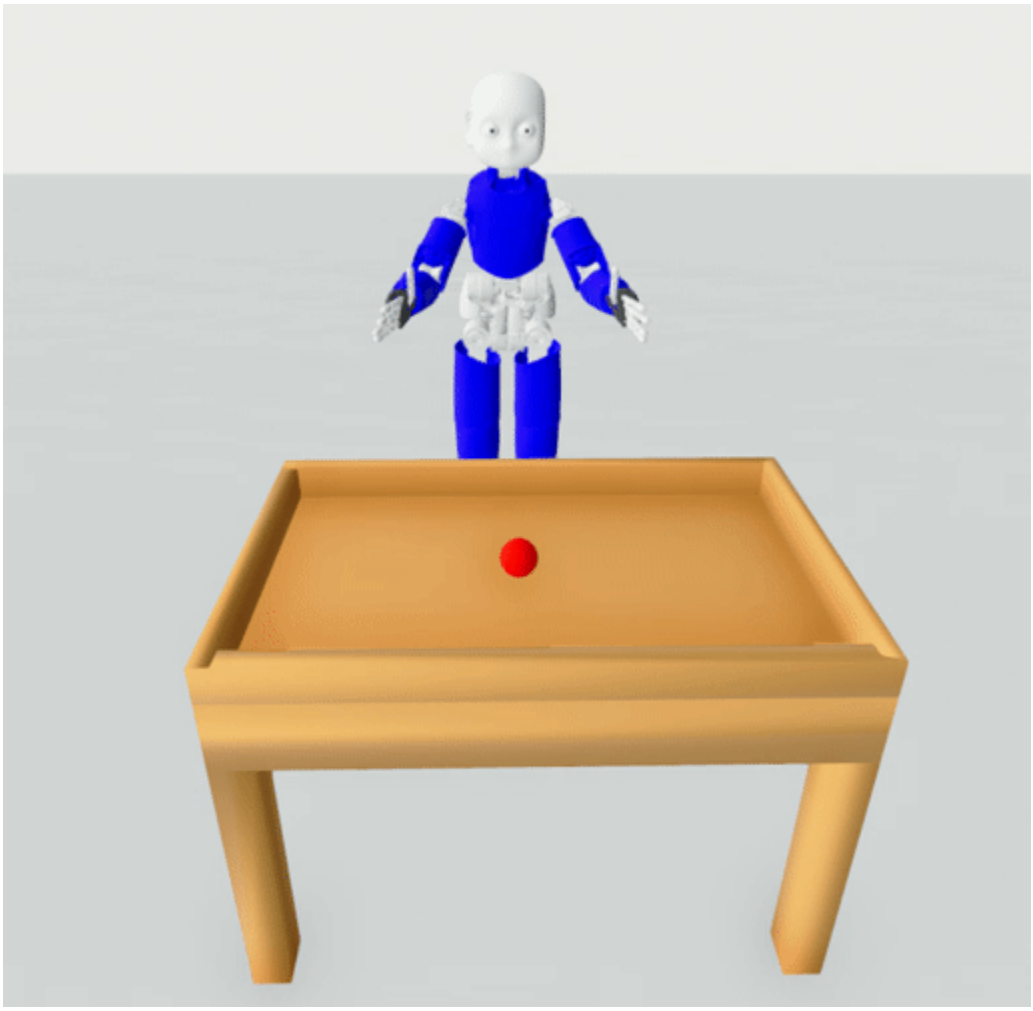
- do not create new client instance, use the one that is passed as an argument
- do not rename the function or file
- **return two arguments:** start and end pose of the trajectory of type utils.Pose

Those apply so that you fulfill the exercise as intended:

- **Do not** turn of gravity
- Make the movement as smooth as possible

4.3 Gaze control

The goal of this task is to implement gaze controller for the iCub robot that is able to follow a ball moving on a table. The task is simplified to 2D case, i.e., the ball can move in x and y. The user is given the vector from the head to a ball (the where the robot should look) and the vector where the robot is looking right now.



4.3.1 Task

Implement gaze() function in exercise_3.py that will control the gaze of the robot to follow the ball.

- the function takes three arguments:
  - client - instance of pycub class that controls the simulation
  - head_direction - normalized vector representing the view of the robot, i.e., where the robot is looking
  - head_ball_direction - normalized vector representing the direction from the robot to the ball, i.e., where the robot should be looking
- the function should control joints in the necks of the robot to follow the ball
  - the move **must** non-blocking, i.e., parameter wait=False
- you **should not** call `update_simulation()` in this function

4.3.2 Scoring

- the ball will be moving for 10 (default) seconds and each step the error in degrees will be calculated
- maximum number of points is 10 (default; 10 runs of the code) and you can **lose** points for individual runs based on the following:

  - if the mean absolute error is:
    * less than 0.5 degree - 0% of points
    * more than 0.5 and less than 1 degree - 50% of points
    * more than 1 and less than 5 degrees - 75% of points
    * more than 5 - 100% of points
  - if the max error is:
    * less than 2 degrees - 0% of points
    * more than 2 and less than 5 degrees - 25% of points
    * more than 5 and less than 10 degrees - 50% of points
    * more than 10 - 100% of points
  - the loss is accumulated for both mean and max error, e.g., 75% loss in mean and 25% in max means 100% and 0 points

4.3.3 Requirements

Those apply mainly for exercise_3_tester.py **to work correctly**:

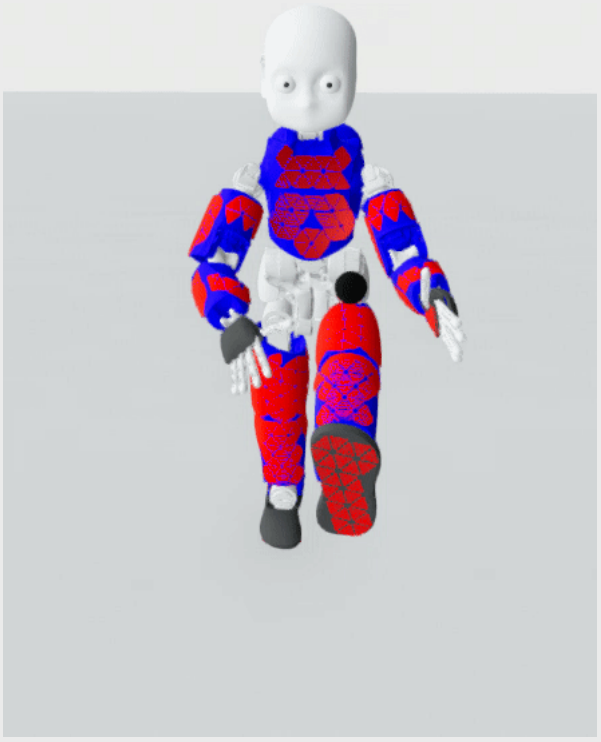
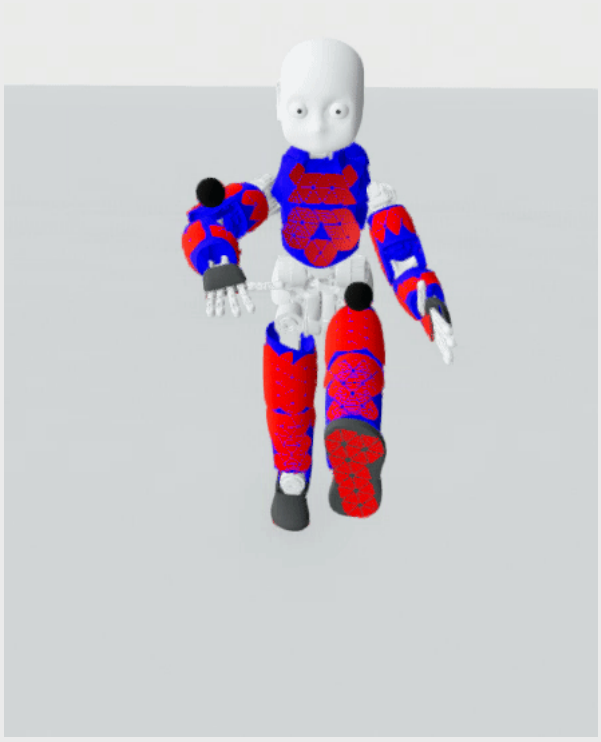
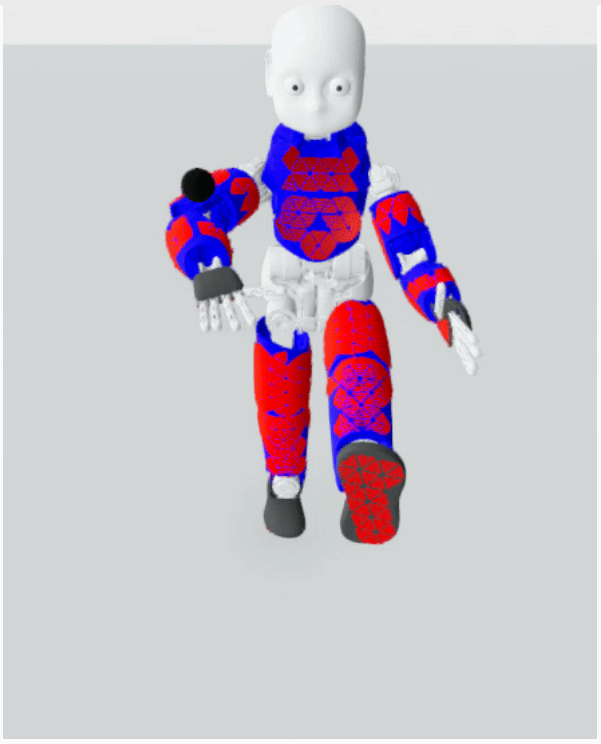
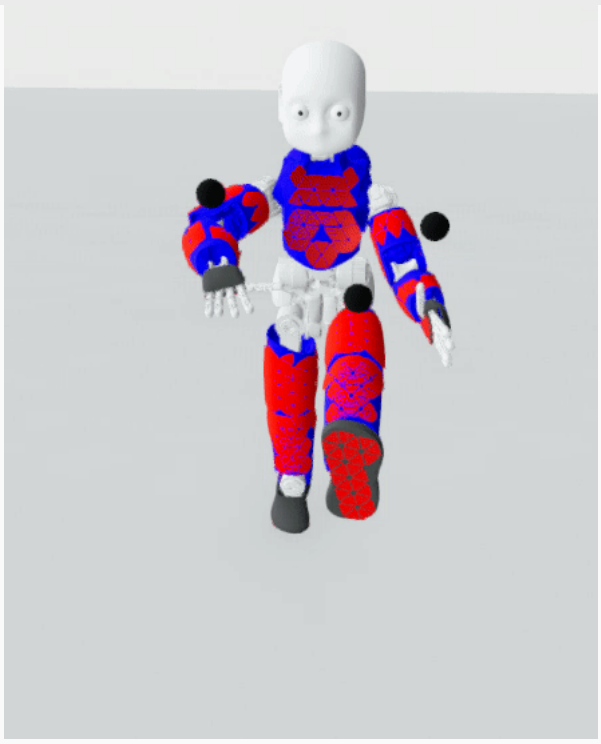
- do not create new client instance, use the one that is passed as an argument
- do not rename the function or file
- **use non-blocking movements**, i.e., use parameter wait=False or use velocity control
- **do not call** `update_simulation()` in any of your code

Those apply so that you fulfill the exercise as intended:

- **Do not** turn of gravity

4.4 Resolved-Rate Motion Control

The goal of this task is to implement a Resolved-Rate Motion Control (RRMC) controller that moves the limbs away from collision using feedback from the artificial skin. There will be ball falling on the robot and you should move the correct body part away from it, i.e., move it against the normal of the contact.

Single limb	Multiple limbs
	
	

4.4.1 Task

Implement process() function in exercise_4.py that will control the robot using RRMC. This time, the function is a method of a class. The reason is the option to store you own variables in the class.

- the class takes one argument:
  - client - instance of pycub class that controls the simulation
- the function should control joints using RRMC -> using velocity control
- you **should not** call `update_simulation()` in any of your code

You will be given four different scenarios with different body_parts being hit by the ball and with different number of balls. You should move every skin part that is in collision with any ball! But always use one contact per skin part, i.e., you should find the biggest cluster of activated taxels (skin points) on each skin part.

The tester script exercise_4_tester.py will run all four tests in sequence from the easier one. You need to manually close the visualization window after each test.

4.4.2 Scoring

There is no automatic evaluation for this task. But basically the task will be correctly fulfilled if the robot moves away from the ball. You can consult the video on top of this README to see possible outcomes. However, keep in mind that those are not the only correct solutions as the movements depends on the parameters selected in RRMC.

4.4.3 Requirements

Those apply mainly for exercise_4_tester.py **to work correctly**:

- do not create new client instance, use the one that is passed as an argument to the RRMC class
- do not rename the function, file or class
- **do not call** `update_simulation()` in any of your code

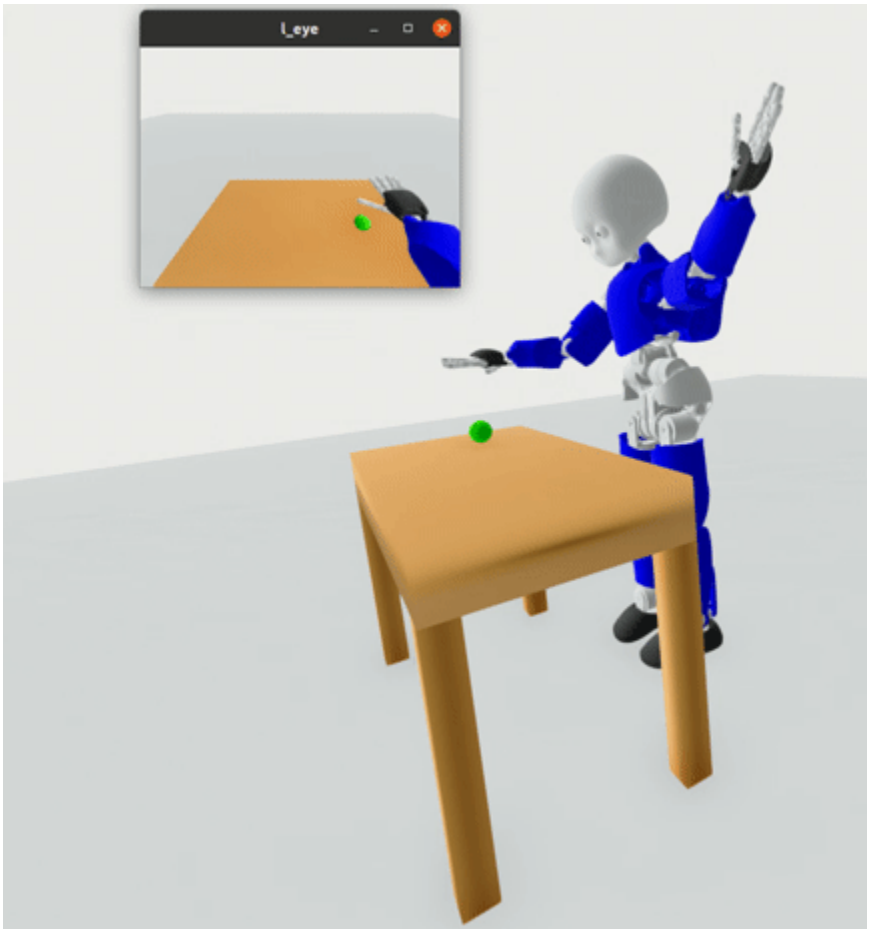
Those apply so that you fulfill the exercise as intended:

- **Do not** turn of gravity

4.5 Grasp It!

The goal of the task is to grasp a ball from the table. The exercise is divided into two parts:

- find the ball on a table in the image plane. The ball is in the visual field of the robot and is green.
- grasp the ball. Use the position of ball in image plane to compote its 3D position and grasp it.



4.5.1 Task

Implement find_the_ball() and grasp() functions in exercise_5.py that will find and grasp the ball. The function are methods of a class Grasper, that contains several useful methods. After grasp, you should move the ball up such that it is at least 5cm above the table.

The class takes two arguments

- client - instance of pyCub class that controls the simulation
- eye - name of the eye in which the camera is placed; 'l_eye' or 'r_eye' The two function have predefined arguments and returns:
- find_the_ball() - takes no arguments and should return (u, v) - center of the ball in image plane
- grasp() - takes one argument `center` (the 2D center returned by find_the_ball()) and should return 0 in case of successful grasp You are free to add any method or variable to the class, but only find_the_ball() and grasp() will be called.

The class Grasper contains several helpful methods, e.g., to get RGB and Depth image; to deproject the 2D point; to close fingers.

The ball is always green (0, 255, 0) and it's radius is 2.5cm.

4.5.2 Scoring

- If the ball is 5cm above the table and in the hand of the robot after you return from grasp(), you passed the test

4.5.3 Requirements

Those apply mainly for exercise_5_tester.py **to work correctly**:

- do not create new client instance, use the one that is passed as an argument to Grasper class
- do not rename the functions find_the_ball(), grasp() or the file
- The ball must be 5cm above the table and in the hand of the robot after grasp() returns

Those apply so that you fulfill the exercise as intended:

- **Do not** turn of gravity

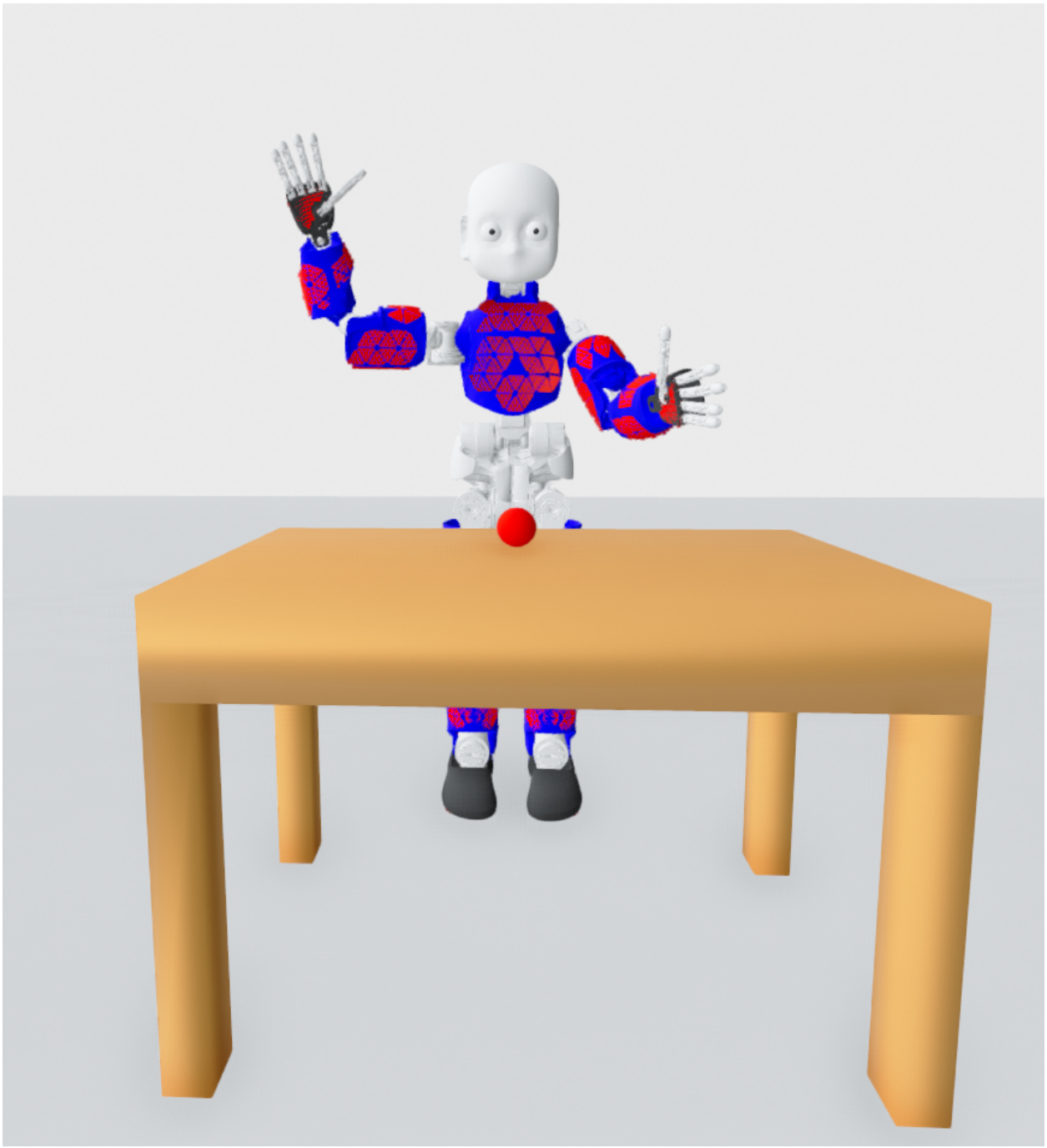
CHAPTER

FIVE

PROJECT README

pyCub is a PyBullet-based simulator and educational robotics framework for the **iCub** humanoid robot. It provides both:

- **pyCub Studio** (`pycub-studio`) — a cross-platform desktop application with an embedded native OpenGL 3.3 / 2.1 viewport, five integrated learning/development modes, offline bilingual handbook, course exercises, live **English / Čeština** switching, and standalone OS installers.
- **`icub_pybullet`** — a lightweight Python package for joint, velocity, and Cartesian control, tactile skin simulation, stereo eye cameras, and example controllers (without Studio GUI dependencies or handbook assets).



Quick Links: Project Website | *Local HTML Handbook* | Online Handbook | PDF Handbook (EN) | PDF příručka (CZ)

5.1 pyCub Studio (Standalone Desktop Application)

pyCub Studio packages the simulator into native installers and portable executables for **Windows (x64)**, **macOS (Apple Silicon `arm64` & Intel `x86_64`)**, and **Linux (`x86_64` & `arm64` `.AppImage` / `.deb`)**. All desktop modes render through a hardware-accelerated native OpenGL 3.3 / 2.1 viewport embedded directly inside the application window (no separate Open3D GUI window or browser needed).

5.1.1 Five Integrated Modes

1. **Kid GUI (Native Buttons)** — Tactile playground with large preset actions, single-joint jog buttons, adjustable step size and speed, custom `.yaml` world loading, live ball distance/hit telemetry, and confetti celebration.
2. **Kid GUI "Scratch" (Visual Blocks)** — Offline Google Blockly visual programming with bilingual EN/CZ blocks, real-time active-block highlighting, `.pycub_blocks` save/load, live Python code preview, and one-click export to Kid Programming.
3. **Kid GUI "Programming" (High-Level Python)** — Beginner-friendly Python editor powered by `icub_pybullet.kid_api` (`wave("right")`, `shoulder_up("left", 20)`, `look_down(15)`, `ball_touched()`, `ball_distance()`, etc., plus Czech aliases) with clickable Command Palette, hover/signature help, starter templates, and friendly error hints.
4. **Normal Mode (Full pyCub IDE & Script Runner)** — Multi-tab Python/YAML IDE with autocomplete, hover docs & Go to Definition (`F12`), external `.py` script runner (`--run-script` + auto-run on save), live stdout/stderr console, dockable `l_eye`/`r_eye` RGB & Depth camera views, coordinate axes toggle, RGB/Depth export, and one-click Course Exercises (`1-5`) installer + automated Exercise Tester.
5. **Classroom Web GUI Selector (Multi-Instance Server)** — Native dashboard to launch and supervise `N` independent browser-based `kid_ui` simulator instances behind a load-balanced landing page (`kid_ui_landing.py`), with LAN URLs, a scannable Wi-Fi QR code, live per-instance status/user counts, individual restart, and log inspection.

5.1.2 Installing & Running `pyCub Studio` via `pip`

Requires **Python 3.10–3.12**. You can install `pycub-studio` as a Python package so it launches from any working directory (no cloned repository needed):

```
python3 -m venv .venv
source .venv/bin/activate
python3 -m pip install --upgrade pip

# From PyPI (installs icub_pybullet + Studio GUI + handbook + exercises + examples):
python3 -m pip install pycub-studio
# Equivalently: python3 -m pip install "icub_pybullet[studio]"

# Or from a local repository clone:
python3 -m pip install ./icub_pybullet ./studio
# (For editable development installs: python3 -m pip install -e ./icub_pybullet -e ./
↪studio)

# Launch Home Mode Selector
pycub-studio
# (or: python3 -m studio.main)

# Launch directly into a specific mode (home | kid_native | kid_scratch | kid_code |␣
↪normal | web)
pycub-studio --mode kid_scratch --lang cs

# Open Normal Mode and immediately run an external Python script
pycub-studio --run-script /path/to/your_script.py
```

On Windows PowerShell, activate the virtual environment with `.venv\Scripts\Activate.ps1`.

5.1.3 User Workspace & Extra Python Packages (~/Documents/pyCub/)

On first launch, pyCub Studio creates a writable workspace in ~/Documents/pyCub/:

- worlds/ — Custom .yaml world configs (automatically listed in the world selector across Modes 1–3 & 5). Custom worlds can inherit built-in configs via parent: default.yaml or parent: with_ball.yaml.
- meshes/ — Custom .obj or .urdf scene models. Any .obj without a pre-baked *_vhacd.obj automatically runs pybullet.vhacd on first load and caches the convex decomposition in <user_cache_dir>/pycub/vhacd/.
- extra_packages/ — Automatically added to sys.path in simulation and script workers for third-party libraries not bundled in the installer.
- scripts/, projects/, exercises/, examples/, images/ — Saved Python scripts, Scratch .pycub_blocks projects, unpacked course exercises, and exported RGB/Depth screenshots.

External scripts & packages without system Python: Scripts in Mode 4 or --run-script run in an isolated worker using Studio's bundled Python runtime (pybullet, numpy, scipy, open3d, roboticstoolbox, spatialmath, matplotlib, etc.). Click **Packages / Python** in the top bar to install PyPI packages into extra_packages/, attach site-packages from an existing virtual environment, or select an external Python executable for Mode 4 scripts.

5.2 Building pyCub Studio (Native Installers & Bundles)

5.2.1 Option 1: Cross-Platform CI Build (Recommended)

Trigger .github/workflows/release-standalone.yml manually (workflow_dispatch, with optional version input) or push a v* tag to build all five targets in parallel and attach them to a GitHub Release:

- **Linux x86_64 & arm64**: .AppImage and .deb (pycub-studio)
- **Windows x64**: pyCub-Setup-<version>-Windows-x64.exe and pyCub-<version>-Portable-Windows-x64.zip
- **macOS arm64 (Apple Silicon) & x86_64 (Intel)**: .dmg and .app.zip

5.2.2 Option 2: Local Builds

Common prerequisites:

- **Python 3.10–3.12** (Python 3.13+ is not yet supported by all robotics dependencies).
- *(Optional)* If you edited documentation in docs/source/, docs/source_cs/, or exercises/, rebuild the offline handbook first (python docs/build.py) so packaging/pycub.spec bundles the updated docs/handbook/.

Linux (x86_64 and arm64)

1. Install required system libraries (Debian/Ubuntu):

```
sudo apt-get update
sudo apt-get install -y --no-install-recommends \
  libegl1 libgl1 libxcb-cursor0 libnss3 libasound2 libfuse2 dpkg-dev curl
# On Linux arm64, also install build tools because pybullet compiles from source:
# sudo apt-get install -y build-essential python3-dev
```

2. Run the packaging script from the repository root (automatically creates an isolated packaging/work/build_venv unless PYTHON_BIN is provided):

```
bash packaging/linux/build_linux.sh
```

Optional environment variables:

- `PYCUB_VERSION=1.4.0` — override artifact version string.
- `REBUILD_VENV=1` — recreate `packaging/work/build_venv` from scratch.
- `PYTHON_BIN=/path/to/python` — use a pre-configured interpreter that already has `icub_pybullet`, `studio/requirements.txt`, and `pyinstaller>=6.10`.

3. Outputs in `packaging/out/`:
   - `pycub-studio_<version>_<amd64|arm64>.deb`
   - `pyCub-<version>-Linux-<x86_64|arm64>.AppImage`

Windows (x64)

1. Activate a Python 3.10–3.12 environment with Studio and PyInstaller dependencies installed:

```
python -m pip install --upgrade pip setuptools wheel
python -m pip install .\icub_pybullet
python -m pip install -r .\studio\requirements.txt
python -m pip install "pyinstaller>=6.10"
```

2. *(Optional)* Install Inno Setup 6 (e.g. `choco install innosetup -y`) to generate the `Setup.exe` installer (if omitted, the script still builds the Portable `.zip`).
3. Run the Windows build script:

```
$env:PYCUB_VERSION = "1.4.0"   # optional
powershell -ExecutionPolicy Bypass -File packaging\windows\build_windows.ps1
```

4. Outputs in `packaging/out/`:
   - `pyCub-Setup-<version>-Windows-x64.exe`
   - `pyCub-<version>-Portable-Windows-x64.zip`

macOS (arm64 Apple Silicon & x86_64 Intel)

1. Install dependencies in a Python 3.10–3.12 environment:

```
python3 -m pip install --upgrade pip setuptools wheel
# On Intel macOS (x86_64), pin Open3D 0.19.0 (last wheel supporting Intel macOS):
# python3 -m pip install "open3d==0.19.0"
python3 -m pip install ./icub_pybullet
python3 -m pip install -r studio/requirements.txt
python3 -m pip install "pyinstaller>=6.10"
```

2. Build `.app` bundle (ad-hoc signed) and `.dmg`:

```
PYCUB_VERSION=1.4.0 bash packaging/macos/build_dmg.sh
```

3. Outputs in `packaging/out/`:
   - `pyCub-<version>-macOS-<arm64|x86_64>.dmg`
   - `pyCub-<version>-macOS-<arm64|x86_64>.zip`

Raw PyInstaller Bundle (Any OS)

To build only the unpacked PyInstaller application directory (`dist/pyCub` or `dist/pyCub.app`) without creating OS installers:

```
pyinstaller --noconfirm packaging/pycub.spec
```

5.3 Python Package Installation (`icub_pybullet` & `pycub-studio`)

- **`icub_pybullet`** installs the simulator, robot meshes/URDFs, YAML configs, `kid_api`, and example scripts cleanly on their own — without Qt/PySide6, Studio UI, or the offline handbook.
- **`pycub-studio`** (or `icub_pybullet[studio]`) depends on `icub_pybullet` and adds the full desktop application, Blockly assets, offline HTML/PDF handbook, and course exercises (1–5) with the `pycub-studio` launcher command.
- Requires **Python 3.10 to 3.12** (newer Python versions are not currently supported due to dependency compatibility).
- Recommended: create a virtual environment first:

```
python3 -m venv pycub_venv
source pycub_venv/bin/activate
```

1. **Install from PyPI (recommended):**

```
python3 -m pip install icub_pybullet          # simulator library only
python3 -m pip install pycub-studio           # or "icub_pybullet[studio]" for full␣
↪Studio
```

2. **Install from source:**

```
git clone https://github.com/rustlluk/pyCub.git
cd pyCub
python3 -m pip install --upgrade pip
python3 -m pip install ./icub_pybullet        # simulator library only
python3 -m pip install ./icub_pybullet ./studio   # simulator + pycub-studio command
```

3. **Docker / Gitpod:**
   - Native Docker (GNU/Linux): see *Native Version*
   - VNC Docker (Windows/macOS/Linux): see *VNC Version*
   - Gitpod: open https://app.gitpod.io/#https://github.com/rustlluk/pycub, log in with GitHub, open port `6080`, and run `start-vnc-session.sh` in the terminal.

5.3.1 Examples & Core Modules

- push_the_ball_pure_joints.py — controlling the robot in joint space.
- push_the_ball_cartesian.py — controlling the robot in Cartesian space (`Pose` + IK).
- skin_test.py — tactile skin contacts with balls falling onto the robot (slow down simulation or check `skin_activations`).
- Simulator core: pycub.py

- Beginner API: kid_api.py
- Visualization backend: visualizer.py (uses Open3D for standalone scripts, web fallback, or the embedded OpenGL viewport inside `pyCub Studio`).

5.4 Documentation & Building the Handbook

Open **Handbook** in `pyCub Studio` or browse the *local HTML handbook* (online at pyCub Handbook; PDF: English, Čeština). The handbook covers Studio modes, Python programming across Kid/Studio/Normal modes, examples, course exercises, and the auto-generated API reference.

To rebuild the English and Czech HTML + PDF handbooks from the repository root (Python 3.10–3.12):

```
python3 -m pip install -r docs/requirements.txt
python3 docs/build.py          # builds HTML + PDF (pass --no-pdf to skip LaTeX PDF)
```

The build writes `docs/handbook/index.html`, `docs/handbook/cs/index.html`, and (when `latexmk` + `pdflatex` + `make` are installed) `docs/handbook/pycub.pdf` and `docs/handbook/cs/pycub.pdf`. Commit the rebuilt `docs/handbook/` alongside documentation changes so GitHub Pages and offline Studio installers pick them up. To preview the full Jekyll site locally, run `jekyll serve` inside `docs/`.

For additional Sphinx targets, run `make -C docs help` or `docs\make.bat help`.

5.5 FAQ & Known Issues

1. **Visualization errors or segmentation faults (e.g. with Open3D):**
    1. Run `pyCub Studio` (`pycub-studio` or `python3 -m studio.main`), which uses an embedded native OpenGL 3.3 / 2.1 viewport instead of a separate Open3D GUI window.
    2. Check your graphics drivers (Mesa / OpenGL / NVIDIA).
    3. Use *Native Docker* (Linux) or *VNC Docker*.
    4. In your `.yaml` config, set `gui.standard: False` and `gui.web: True` to use the browser visualizer.
2. **Import errors (e.g. `No module named icub_pybullet`):**
    1. Install from PyPI (`python3 -m pip install icub_pybullet` or `pycub-studio`) or from source (`python3 -m pip install ./icub_pybullet`).
    2. Or add the repository root directory to your `PYTHONPATH`.
3. **Known bug with Open3D skin visualization:**
    - In standalone Open3D mode, visualization with tactile skin active can crash after ~65k steps (see Open3D#4992). `pyCub Studio`'s embedded OpenGL viewport is not affected.

5.6 Docker

5.6.1 Native Version

- Native GUI inside Docker (GNU/Linux only — Ubuntu, Mint, Arch, etc.); useful for avoiding host OpenGL driver issues.

1. Install docker-engine (**do not install Docker Desktop** on Linux) and perform the Linux post-installation steps.
2. Clone the repository and pull or build the image:

```
git clone https://github.com/rustlluk/pyCub.git
cd pyCub/Docker
# Pull prebuilt image from Docker Hub:
./deploy.py -p /path/to/pyCub -c pycub -pu
# Or build locally:
./deploy.py -p /path/to/pyCub -c pycub -b
```

3. Run an existing container or open a new terminal in it:

```
./deploy.py -c pycub -e   # start/attach
./deploy.py -c pycub -t   # open another terminal
```

5.6.2 VNC Version

- Works on Windows, macOS, and Linux; displays the GUI in your browser at http://localhost:6080.

1. Install docker-engine (Linux) or docker-desktop (all systems) and complete any post-install steps.
2. Pull or build with `-vnc`:

```
cd /path/to/pyCub/Docker
./deploy.py -p /path/to/pyCub -c pycub -pu -vnc
```

3. Inside the container, run `start-vnc-session.sh` and open http://localhost:6080.

5.6.3 Docker + PyCharm

- **Native Version:** Either launch PyCharm inside Docker, or configure an SSH interpreter in host PyCharm (`user: docker`, host `localhost` or server IP, port `2222`, disable automatic upload, map remote path to `/home/docker/pycub_ws`).
- **VNC / Gitpod Version:** Open PyCharm directly inside the container.

5.6.4 Deploy Parameters

Run `./deploy.py` inside `Docker/`:

- `-b` / `--build`: Build the Docker image (default: `False`).
- `-e`: Run an existing container without rebuilding (default: `False`).
- `-p` / `--path`: Path to the repository folder (default: `""`).
- `-pu` / `--pull`: Pull image from Docker Hub (default: `False`).
- `-c` / `--container`: Container name (default: `my_new_docker`).
- `-t` / `--terminal`: Open a new terminal in a running container (default: `False`).
- `-pv` / `--python-version`: Python version to install (default: `3.11`).

- `-pcv` / `--pycharm-version`: PyCharm version to install (default: `2023.2.3`).
- `-bi` / `--base-image`: Base Docker image (default: `ubuntu:20.04`).

Run `deploy.py` on the machine where you run the code (over SSH if using a remote server).

5.6.5 Docker FAQ

- **Not in sudo group when running the pulled image:** Check `id -u`. If your UID is not `1000`, build the image locally instead of pulling it.
- **`sudo apt install ...` fails inside the container:** Run `sudo apt update` first (apt caches are cleaned in the `Dockerfile` to save space).

5.7 Citing

When used in academic work, please cite this repository as:

```
@inproceedings{rustler2026pycub,
      title={Learning with pyCub: A Simulation and Exercise Framework for Humanoid␣
↪Robotics},
      author={Lukas Rustler and Matej Hoffmann},
      year={2026},
      booktitle={17th International Conference on Robotics in Education (RiE 2026)},
      organization={Springer}}
```

5.8 License

This work is licensed under a Creative Commons Attribution 4.0 International License.

CHAPTER

SIX

API REFERENCE

This page contains auto-generated API reference documentation[1].

6.1 skin_test

Script to the test the skin sensors. Balls falling to the skin and turning activated point to green should be seen.

Author
Lukas Rustler

6.1.1 Functions

main() → NoReturn	Main function to run the example

6.1.2 Module Contents

skin_test.**main**() → NoReturn
Main function to run the example

Returns

Return type

6.2 icub_pybullet

PyBullet-based simulator package for the iCub humanoid robot.

Author
Lukas Rustler

6.2.1 Submodules

icub_pybullet.kid_api

High-level beginner-friendly Python API for pyCub (“Kid Programming” & “Scratch” Blockly runtime).

Allows students to control iCub with intuitive one-line commands without writing simulation loops or radian conversions, both inside the standalone desktop app (Modes 2 & 3) and when executed directly as a standard *.py* script.

[1] Created with sphinx-autoapi

Exceptions

KidScriptStopped	Raised cooperatively to stop a running Kid Scratch / Kid Programming script immediately.

Classes

KidRobot	Controller binding *kid_api* functions either to a managed step-callback (inside the

Functions

get_robot() → KidRobot	Return the active KidRobot controller (lazily starting *with_ball.yaml* if run standalone).
wave([side]) → None	Wave using the left or right arm.
left_wave() → None	Wave using the left arm.
right_wave() → None	Wave using the right arm.
both_arms_up() → None	Raise both arms to their preset positions.
both_arms_down() → None	Lower both arms to their preset positions.
look_center() → None	Center the neck joints.
stand_straight() → None	Return the torso to its neutral position.
reset_pose() → None	Return the robot to its starting pose.
stop() → None	Stop current joint movements.
shoulder_up([side, degrees, speed]) → None	Raise the selected shoulder.
shoulder_down([side, degrees, speed]) → None	Lower the selected shoulder.
shoulder_out([side, degrees, speed]) → None	Move the selected arm away from the torso.
shoulder_in([side, degrees, speed]) → None	Move the selected arm toward the torso.
rotate_arm([side, degrees, speed]) → None	Rotate the selected upper arm.
bend_elbow([side, degrees, speed]) → None	Bend the selected elbow.
straighten_elbow([side, degrees, speed]) → None	Extend the selected elbow.
rotate_wrist([side, degrees, speed]) → None	Rotate the selected wrist.
turn_head([degrees, speed]) → None	Turn the head about the neck yaw axis.
look_up([degrees, speed]) → None	Tilt the head upward.
look_down([degrees, speed]) → None	Tilt the head downward.
tilt_head([degrees, speed]) → None	Tilt the head to the side.
turn_body([degrees, speed]) → None	Rotate the torso about its yaw axis.
bend_forward([degrees, speed]) → None	Bend the torso forward.
bend_back([degrees, speed]) → None	Bend the torso backward.
bend_sideways([degrees, speed]) → None	Tilt the torso to the side.
move_joint(joint_name, target_degrees[, speed]) → None	Move a joint to an absolute angle.
jog_joint(joint_name, delta_degrees[, speed]) → None	Move a joint relative to its current angle.
get_joint(joint_name) → float	Read the current joint angle in degrees.
wait([seconds]) → None	Pause the program for the given number of seconds while the simulation runs.
ball_distance() → float	Read the distance between the ball and the table in centimetres.
ball_touched() → bool	Read whether the robot has touched the ball.

continues on next page

Table 4 – continued from previous page

ball_hits() → int	Read the ball hit count from the active controller.
skin_touched() → bool	Read whether any enabled skin sensor is registering contact.

Module Contents

exception icub_pybullet.kid_api.**KidScriptStopped**

Raised cooperatively to stop a running Kid Scratch / Kid Programming script immediately.

class icub_pybullet.kid_api.**KidRobot**(*client*, *step_fn: Callable[[], None] | None = None*, *stop_event: threading.Event | None = None*, *highlight_cb: Callable[[str], None] | None = None*, *telemetry_getter: Callable[[], Dict[str, object]] | None = None*)

Controller binding *kid_api* functions either to a managed step-callback (inside the interactive simulation worker) or to a standalone *pyCub* client instance.

client

check_stop() → None

Raise `KidScriptStopped` when cancellation is requested or the simulator closes.

highlight_block(*block_id: str*) → None

Notify the Blockly editor which block is executing and yield briefly.

move_joint(*joint_name: str*, *target_degrees: float*, *speed: float = 1.5*) → None

Move a named joint to an absolute angle in degrees.

jog_joint(*joint_name: str*, *delta_degrees: float*, *speed: float = 1.5*) → None

Move a named joint relatively by *delta_degrees*.

get_joint(*joint_name: str*) → float

Return the current angle of *joint_name* in degrees.

wave(*side: str = 'right'*) → None

Wave with the *'right'* or *'left'* arm.

both_arms_up() → None

Raise both arms to their preset positions.

both_arms_down() → None

Lower both arms to their preset positions.

look_center() → None

Center the neck joints.

stand_straight() → None

Return the torso to its neutral upright posture.

reset_pose() → None

Return all body joints to their configured initial angles.

stop() → None

Stop all current joint movements.

shoulder_up(*side: str = 'right'*, *degrees: float = 15.0*, *speed: float = 1.5*) → None

Raise the selected shoulder by `degrees`.

shoulder_down(*side: str = 'right'*, *degrees: float = 15.0*, *speed: float = 1.5*) → None
Lower the selected shoulder by `degrees`.

shoulder_out(*side: str = 'right'*, *degrees: float = 15.0*, *speed: float = 1.5*) → None
Move the selected arm away from the torso by `degrees`.

shoulder_in(*side: str = 'right'*, *degrees: float = 15.0*, *speed: float = 1.5*) → None
Move the selected arm toward the torso by `degrees`.

rotate_arm(*side: str = 'right'*, *degrees: float = 15.0*, *speed: float = 1.5*) → None
Rotate the selected upper arm by `degrees`.

bend_elbow(*side: str = 'right'*, *degrees: float = 15.0*, *speed: float = 1.5*) → None
Bend the selected elbow by `degrees`.

straighten_elbow(*side: str = 'right'*, *degrees: float = 15.0*, *speed: float = 1.5*) → None
Extend the selected elbow by `degrees`.

rotate_wrist(*side: str = 'right'*, *degrees: float = 15.0*, *speed: float = 1.5*) → None
Rotate the selected wrist by `degrees`.

turn_head(*degrees: float = 15.0*, *speed: float = 1.5*) → None
Turn the head about the neck yaw axis by `degrees`.

look_up(*degrees: float = 15.0*, *speed: float = 1.5*) → None
Tilt the head upward by `degrees`.

look_down(*degrees: float = 15.0*, *speed: float = 1.5*) → None
Tilt the head downward by `degrees`.

tilt_head(*degrees: float = 15.0*, *speed: float = 1.5*) → None
Tilt the head sideways by `degrees`.

turn_body(*degrees: float = 15.0*, *speed: float = 1.5*) → None
Rotate the torso about its yaw axis by `degrees`.

bend_forward(*degrees: float = 15.0*, *speed: float = 1.5*) → None
Bend the torso forward by `degrees`.

bend_back(*degrees: float = 15.0*, *speed: float = 1.5*) → None
Bend the torso backward by `degrees`.

bend_sideways(*degrees: float = 15.0*, *speed: float = 1.5*) → None
Tilt the torso sideways by `degrees`.

wait(*seconds: float = 1.0*) → None
Pause script execution for *seconds* while keeping physics running.

ball_distance() → float
Distance of the ball from the table in cm (0.0 if on table or no ball).

ball_touched() → bool
True if the robot has touched the ball in the current run.

ball_hits() → int
Number of times the robot hit the ball in the current run.

skin_touched() → bool

True if tactile skin is enabled and currently registering contact.

icub_pybullet.kid_api.**get_robot**() → *KidRobot*

Return the active KidRobot controller (lazily starting *with_ball.yaml* if run standalone).

icub_pybullet.kid_api.**wave**(*side: str = 'right'*) → None

Wave using the left or right arm.

Parameters

side – Arm to use (“left” or “right”).

icub_pybullet.kid_api.**left_wave**() → None

Wave using the left arm.

icub_pybullet.kid_api.**right_wave**() → None

Wave using the right arm.

icub_pybullet.kid_api.**both_arms_up**() → None

Raise both arms to their preset positions.

icub_pybullet.kid_api.**both_arms_down**() → None

Lower both arms to their preset positions.

icub_pybullet.kid_api.**look_center**() → None

Center the neck joints.

icub_pybullet.kid_api.**stand_straight**() → None

Return the torso to its neutral position.

icub_pybullet.kid_api.**reset_pose**() → None

Return the robot to its starting pose.

icub_pybullet.kid_api.**stop**() → None

Stop current joint movements.

icub_pybullet.kid_api.**shoulder_up**(*side: str = 'right'*, *degrees: float = 15.0*, *speed: float = 1.5*) → None

Raise the selected shoulder.

Parameters

- **side** – Arm to use (“left” or “right”).
- **degrees** – Change from the current angle, in degrees.
- **speed** – Maximum joint speed in radians per second.

icub_pybullet.kid_api.**shoulder_down**(*side: str = 'right'*, *degrees: float = 15.0*, *speed: float = 1.5*) → None

Lower the selected shoulder.

Parameters

- **side** – Arm to use (“left” or “right”).
- **degrees** – Change from the current angle, in degrees.
- **speed** – Maximum joint speed in radians per second.

icub_pybullet.kid_api.**shoulder_out**(*side: str = 'right'*, *degrees: float = 15.0*, *speed: float = 1.5*) → None

Move the selected arm away from the torso.

Parameters

- **side** – Arm to use (“left” or “right”).
- **degrees** – Change from the current angle, in degrees.
- **speed** – Maximum joint speed in radians per second.

icub_pybullet.kid_api.**shoulder_in**(*side: str = 'right'*, *degrees: float = 15.0*, *speed: float = 1.5*) → None

Move the selected arm toward the torso.

Parameters

- **side** – Arm to use (“left” or “right”).
- **degrees** – Change from the current angle, in degrees.
- **speed** – Maximum joint speed in radians per second.

icub_pybullet.kid_api.**rotate_arm**(*side: str = 'right'*, *degrees: float = 15.0*, *speed: float = 1.5*) → None

Rotate the selected upper arm.

Parameters

- **side** – Arm to use (“left” or “right”).
- **degrees** – Change from the current angle, in degrees.
- **speed** – Maximum joint speed in radians per second.

icub_pybullet.kid_api.**bend_elbow**(*side: str = 'right'*, *degrees: float = 15.0*, *speed: float = 1.5*) → None

Bend the selected elbow.

Parameters

- **side** – Arm to use (“left” or “right”).
- **degrees** – Change from the current angle, in degrees.
- **speed** – Maximum joint speed in radians per second.

icub_pybullet.kid_api.**straighten_elbow**(*side: str = 'right'*, *degrees: float = 15.0*, *speed: float = 1.5*) → None

Extend the selected elbow.

Parameters

- **side** – Arm to use (“left” or “right”).
- **degrees** – Change from the current angle, in degrees.
- **speed** – Maximum joint speed in radians per second.

icub_pybullet.kid_api.**rotate_wrist**(*side: str = 'right'*, *degrees: float = 15.0*, *speed: float = 1.5*) → None

Rotate the selected wrist.

Parameters

- **side** – Arm to use (“left” or “right”).
- **degrees** – Change from the current angle, in degrees.
- **speed** – Maximum joint speed in radians per second.

icub_pybullet.kid_api.**turn_head**(*degrees: float = 15.0*, *speed: float = 1.5*) → None

Turn the head about the neck yaw axis.

Parameters

- **degrees** – Change from the current angle, in degrees.
- **speed** – Maximum joint speed in radians per second.

icub_pybullet.kid_api.**look_up**(*degrees: float = 15.0*, *speed: float = 1.5*) → None

Tilt the head upward.

Parameters

- **degrees** – Change from the current angle, in degrees.
- **speed** – Maximum joint speed in radians per second.

icub_pybullet.kid_api.**look_down**(*degrees: float = 15.0*, *speed: float = 1.5*) → None

Tilt the head downward.

Parameters

- **degrees** – Change from the current angle, in degrees.
- **speed** – Maximum joint speed in radians per second.

icub_pybullet.kid_api.**tilt_head**(*degrees: float = 15.0*, *speed: float = 1.5*) → None

Tilt the head to the side.

Parameters

- **degrees** – Change from the current angle, in degrees.
- **speed** – Maximum joint speed in radians per second.

icub_pybullet.kid_api.**turn_body**(*degrees: float = 15.0*, *speed: float = 1.5*) → None

Rotate the torso about its yaw axis.

Parameters

- **degrees** – Change from the current angle, in degrees.
- **speed** – Maximum joint speed in radians per second.

icub_pybullet.kid_api.**bend_forward**(*degrees: float = 15.0*, *speed: float = 1.5*) → None

Bend the torso forward.

Parameters

- **degrees** – Change from the current angle, in degrees.
- **speed** – Maximum joint speed in radians per second.

icub_pybullet.kid_api.**bend_back**(*degrees: float = 15.0*, *speed: float = 1.5*) → None

Bend the torso backward.

Parameters

- **degrees** – Change from the current angle, in degrees.
- **speed** – Maximum joint speed in radians per second.

icub_pybullet.kid_api.**bend_sideways**(*degrees: float = 15.0*, *speed: float = 1.5*) → None

Tilt the torso to the side.

Parameters

- **degrees** – Change from the current angle, in degrees.
- **speed** – Maximum joint speed in radians per second.

icub_pybullet.kid_api.**move_joint**(*joint_name: str*, *target_degrees: float*, *speed: float = 1.5*) → None

Move a joint to an absolute angle.

Parameters

- **joint_name** – Simulator joint name, such as neck_pitch.
- **target_degrees** – Absolute target angle in degrees.
- **speed** – Maximum joint speed in radians per second.

icub_pybullet.kid_api.**jog_joint**(*joint_name: str*, *delta_degrees: float*, *speed: float = 1.5*) → None

Move a joint relative to its current angle.

Parameters

- **joint_name** – Simulator joint name, such as neck_pitch.
- **delta_degrees** – Signed change in degrees.
- **speed** – Maximum joint speed in radians per second.

icub_pybullet.kid_api.**get_joint**(*joint_name: str*) → float

Read the current joint angle in degrees.

Parameters

joint_name – Simulator joint name, such as neck_pitch.

icub_pybullet.kid_api.**wait**(*seconds: float = 1.0*) → None

Pause the program for the given number of seconds while the simulation runs.

icub_pybullet.kid_api.**ball_distance**() → float

Read the distance between the ball and the table in centimetres.

icub_pybullet.kid_api.**ball_touched**() → bool

Read whether the robot has touched the ball.

icub_pybullet.kid_api.**ball_hits**() → int

Read the ball hit count from the active controller.

icub_pybullet.kid_api.**skin_touched**() → bool

Read whether any enabled skin sensor is registering contact.

icub_pybullet.pycub

The main class and utils for pyCub simulator

Author

Lukas Rustler

Classes

Joint	Container describing a movable PyBullet robot joint and its active control target.
Link	Container mapping a robot link name to its PyBullet and URDF indices.
pyCub	Client class which inherits from BulletClient and contains the whole simulation functionality

continues on next page

Table 5 – continued from previous page

EndEffector	Helper representing the configured end-effector link of the robot.

Functions

set_visualizer_factory(factory) → None	Register a custom visualizer factory (e.g. EmbeddedVisualizerProxy in the standalone desktop app).
get_visualizer_factory() → Optional[Callable[[pyCub], Any]]	Return the currently registered custom visualizer factory, or `None`.
set_status_callback(callback) → None	Register an optional status callback receiving (stage, message) updates (e.g. during dynamic VHACD).

Module Contents

icub_pybullet.pycub.**set_visualizer_factory**(*factory: Callable[[pyCub], Any] | None*) → None

Register a custom visualizer factory (e.g. EmbeddedVisualizerProxy in the standalone desktop app).

icub_pybullet.pycub.**get_visualizer_factory**() → Callable[[*pyCub*], Any] | None

Return the currently registered custom visualizer factory, or `None`.

icub_pybullet.pycub.**set_status_callback**(*callback: Callable[[str, str], None] | None*) → None

Register an optional status callback receiving (stage, message) updates (e.g. during dynamic VHACD).

class icub_pybullet.pycub.**Joint**(*name: str*, *robot_joint_id: int*, *joints_id: int*, *lower_limit: float*, *upper_limit: float*, *max_force: float*, *max_velocity: float*)

Container describing a movable PyBullet robot joint and its active control target.

name

robot_joint_id

joints_id

lower_limit

upper_limit

max_force

max_velocity

set_point = None

start_time = None

timeout = None

class icub_pybullet.pycub.**Link**(*name: str*, *robot_link_id: int*, *urdf_link: int*)

Container mapping a robot link name to its PyBullet and URDF indices.

name

robot_link_id

urdf_link

class icub_pybullet.pycub.**pyCub**(*config: str | None = 'default.yaml'*)

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

jointInfo

jointStates

linkInfo

contactPoints

dynamicsInfo

visualShapeData

parent_name

file_dir

config_path = b'.'

config_dir

config

logger

gravity = False

urdf_path = b'.'

urdfs

other_objects = []

free_objects = []

IK_config

end_effector

last_step

last_log

joint_tolerance

neighbour_links

collision_during_motion = False

steps_done = 0

precomputed_link_ids

property rtb_robot

Lazily constructed Robotics Toolbox `Robot` model used for Jacobian queries.

static get_chains() → Tuple[dict, dict]

Return link and joint definitions for each kinematic chain on iCub.

compute_jacobian(*chain: icub_pybullet.utils.CHAINS*, *start: str | None = None*, *end: str | None = None*) → Tuple[numpy.array, numpy.array]

Compute the base-frame geometric Jacobian and active joint names for a kinematic chain.

Parameters

- **chain** – Kinematic chain identifier (e.g. `"right_arm"`).
- **start** – Optional base link name; defaults to the first link in `chain`.
- **end** – Optional tip link name; defaults to the last link in `chain`.

get_camera_images(*eyes: str | List[str] | None = None*) → dict

Return the latest RGB arrays keyed by eye name. Omit eyes for all open eye views.

get_camera_depth_images(*eyes: str | List[str] | None = None*) → dict

Return the latest depth arrays keyed by eye name. Omit eyes for all open eye views.

find_processes_by_name() → List[int]

Return the current process ID (legacy helper preserved for compatibility).

kill_open3d() → None

Cleanly terminate only the current process if Open3D hangs on exit, never parent/sibling processes.

init_robot() → Tuple[int, List[*Joint*], List[*Link*]]

Load the iCub URDF into PyBullet and build `Joint` and `Link` lists.

init_urdfs() → None

Load URDFs / .obj files of scene objects (supports both built-in meshes and user custom meshes).

is_alive() → bool

Return True while the PyBullet client (and GUI, if enabled) remain active.

update_simulation(*sleep_duration: float | None | None = -1*) → None

Advance physics when the interval in seconds has elapsed and refresh the visualizer.

Use -1 for the configured step interval or None to step on every call.

toggle_gravity() → None

Toggle simulated gravity between Earth standard (-9.81 m/s^2) and zero-G.

static scale_bbox(*bbox: list*, *scale: float*) → Tuple[numpy.array, numpy.array]

Scale an axis-aligned bounding box `(min, max)` about its center.

static bbox_overlap(*b1_min: list*, *b1_max: list*, *b2_min: list*, *b2_max: list*) → bool

Return True when two axis-aligned 3D bounding boxes overlap.

compute_skin() → None

Update simulated tactile skin taxel activations from current PyBullet contacts.

prepare_log() → str

Serialize current timestamp, step count, joint angles, and skin state into a CSV row.

move_position(*joints: icub_pybullet.utils.JOINTS | List[icub_pybullet.utils.JOINTS] | icub_pybullet.utils.JOINTS_IDS | List[icub_pybullet.utils.JOINTS_IDS]*, *positions: float | List[float]*, *wait: bool | None = True*, *velocity: float | None = 1*, *set_col_state: bool | None = True*, *check_collision: bool | None = True*, *timeout: float | None = None*) → None

Move one joint or a list of joints to target angles in radians.

Parameters

- **joints** – Joint name or ID, or a list of joints to move together.
- **positions** – Target angle in radians for each joint, in the same order.
- **wait** – Step the simulation until the movement finishes.
- **velocity** – Maximum joint speed in radians per second.
- **set_col_state** – Clear the collision flag at the start of this move.
- **check_collision** – Check for collisions while waiting for the move.
- **timeout** – Maximum motion time in seconds, or None for no time limit.

move_velocity(*joints: icub_pybullet.utils.JOINTS | List[icub_pybullet.utils.JOINTS] | icub_pybullet.utils.JOINTS_IDS | List[icub_pybullet.utils.JOINTS_IDS]*, *velocities: float | List[float]*) → None

Command one or more joint velocities in radians per second, in joint order.

get_joint_state(*joints: icub_pybullet.utils.JOINTS | List[icub_pybullet.utils.JOINTS] | icub_pybullet.utils.JOINTS_IDS | List[icub_pybullet.utils.JOINTS_IDS] | None = None*, *allow_error: bool | None = False*) → List[list]

Read positions in radians in the requested order, defaulting to all joints.

motion_done(*joints: icub_pybullet.utils.JOINTS | List[icub_pybullet.utils.JOINTS] | icub_pybullet.utils.JOINTS_IDS | List[icub_pybullet.utils.JOINTS_IDS] | None = None*, *check_collision=True*) → bool

Return True when all specified joints reached their targets, timed out, or hit a collision.

Parameters

- **joints** – Joint name/ID or list of joints to check; defaults to all robot joints.
- **check_collision** – Abort and return True if an unexpected contact occurs.

wait_motion_done(*sleep_duration: float | None = 0.01*, *check_collision: bool | None = True*) → None

Block while stepping the simulation until `motion_done` returns True.

stop_robot(*joints: icub_pybullet.utils.JOINTS | List[icub_pybullet.utils.JOINTS] | icub_pybullet.utils.JOINTS_IDS | List[icub_pybullet.utils.JOINTS_IDS] | None = None*) → None

Halt active position or velocity commands for `joints` (or all joints).

move_cartesian(*pose:* icub_pybullet.utils.Pose, *wait: bool | None = True*, *velocity: float | None = 1*, *check_collision: bool | None = True*, *timeout: float | None = None*) → None

Move the end effector to a pose using inverse kinematics.

Velocity limits joint speed in radians per second. With wait=False, call update_simulation to advance the motion.

find_joint_id(*joint_name: icub_pybullet.utils.JOINTS | icub_pybullet.utils.JOINTS_IDS*) → Tuple[int, int]

Resolve a joint name or index to `(pybullet_joint_id, joints_list_index)`.

find_link_id(*mesh_name: str*, *robot: int | None = None*, *urdf_name: str | None = 'robot'*) → int

Find the PyBullet link index whose collision shape uses `mesh_name`.

run_vhacd(*robot: bool | None = True*) → None

Run VHACD on loaded URDF objects using pre-baked meshes when present, or writing newly generated collision meshes and patched URDFs into the writable user cache directory.

create_urdf(*object_path: str*, *fixed: bool*, *color: List[float]*, *suffix: str | None = ''*, *obj_name: str | None = None*) → str

Create a URDF for a given .obj file inside the writable user cache directory (safe on read-only installs and safe under concurrent multi-instance execution).

print_collision_info(*c: list | None = None*)

Log details for a single PyBullet contact record or all active robot contacts.

class icub_pybullet.pycub.**EndEffector**(*name: str*, *client:* pyCub)

Helper representing the configured end-effector link of the robot.

name

client

get_position() → *icub_pybullet.utils.Pose*

Return the current world-frame `Pose` of the end-effector link.

icub_pybullet.runtime_paths

Cross-platform runtime paths, user workspace/cache helpers, config/mesh resolution, and OS-agnostic process/network utilities for pyCub.

Attributes

SKIP_IFACE_PREFIXES

Functions

user_documents_dir() → str	Return the fallback path to the user's Documents directory.
is_frozen() → bool	Return True when running inside a frozen bundle (e.g. PyInstaller).
to_posix(path) → str	Normalize a path and convert separators to forward slashes (safe inside URDF XML on Windows).
is_writable_dir(directory) → bool	Return True only if directory is writable and NOT inside a frozen bundle directory.
get_user_workspace_dir() → pathlib.Path	Return the user's writable pyCub workspace directory (default: ~/Documents/pyCub).
ensure_user_workspace() → pathlib.Path	Ensure standard user workspace subdirectories exist and return root path.
get_extra_packages_dir() → pathlib.Path	Return the writable extra Python packages directory (*~/Documents/pyCub/extra_packages*).
get_user_cache_dir() → pathlib.Path	Return writable cache directory for generated URDFs and dynamic VHACD collision meshes.
get_user_log_dir([package_dir]) → pathlib.Path	Return writable log directory (falls back to user data dir when package_dir/logs is read-only).

continues on next page

Table 8 – continued from previous page

write_text_atomic(target_path, content) → str	Write text atomically via temp file + os.replace so concurrent instances never read partial URDFs.
resolve_config_path(config, package_dir[, caller_file]) → str	Resolve a YAML config path across:
resolve_robot_urdf_path(robot_urdf_rel, package_dir[, ...]) → str	Resolve robot_urdf_path across absolute path, config dir, user workspace, and bundled package dir.
resolve_mesh_or_urdf_path(obj_path, package_dir[, ...]) → str	Resolve an object .obj or .urdf path referenced in config.urdfs.paths across:
file_fingerprint(path) → str	Fast fingerprint combining canonical path, size, and mtime_ns.
get_vhacd_cache_path(col_path[, force_vhacd]) → Tuple[str, bool]	Locate an existing pre-baked `*_vhacd.obj` or return a writable cache path.
get_cached_urdf_path(stem, key[, subdir]) → str	Return a deterministic path in the user cache for a generated/patched URDF.
local_ipv4_addresses() → List[str]	Cross-platform IPv4 LAN addresses discovery (Windows, macOS, Linux).
terminate_process_tree(procs[, timeout]) → None	Cross-platform termination of child processes and all their descendants using psutil.

Module Contents

icub_pybullet.runtime_paths.**user_documents_dir**() → str

Return the fallback path to the user's Documents directory.

icub_pybullet.runtime_paths.**SKIP_IFACE_PREFIXES = ('lo', 'docker', 'br-', 'veth', 'virbr', 'lxc', 'tailscale', 'utun', 'awdl', 'llw')**

icub_pybullet.runtime_paths.**is_frozen**() → bool

Return True when running inside a frozen bundle (e.g. PyInstaller).

icub_pybullet.runtime_paths.**to_posix**(*path: str | os.PathLike*) → str

Normalize a path and convert separators to forward slashes (safe inside URDF XML on Windows).

icub_pybullet.runtime_paths.**is_writable_dir**(*directory: str | os.PathLike*) → bool

Return True only if directory is writable and NOT inside a frozen bundle directory.

icub_pybullet.runtime_paths.**get_user_workspace_dir**() → pathlib.Path

Return the user's writable pyCub workspace directory (default: ~/Documents/pyCub).

icub_pybullet.runtime_paths.**ensure_user_workspace**() → pathlib.Path

Ensure standard user workspace subdirectories exist and return root path.

icub_pybullet.runtime_paths.**get_extra_packages_dir**() → pathlib.Path

Return the writable extra Python packages directory (*~/Documents/pyCub/extra_packages*).

icub_pybullet.runtime_paths.**get_user_cache_dir**() → pathlib.Path

Return writable cache directory for generated URDFs and dynamic VHACD collision meshes.

icub_pybullet.runtime_paths.**get_user_log_dir**(*package_dir: str | None = None*) → pathlib.Path

Return writable log directory (falls back to user data dir when package_dir/logs is read-only).

icub_pybullet.runtime_paths.**write_text_atomic**(*target_path: str | os.PathLike*, *content: str*) → str

Write text atomically via temp file + os.replace so concurrent instances never read partial URDFs.

icub_pybullet.runtime_paths.**resolve_config_path**(*config: str*, *package_dir: str*, *caller_file: str | None = None*) → str

Resolve a YAML config path across:

1. Direct / absolute path or relative to CWD
2. Relative to caller script directory
3. User workspace (~/Documents/pyCub/worlds, configs, root)
4. Bundled icub_pybullet/configs

icub_pybullet.runtime_paths.**resolve_robot_urdf_path**(*robot_urdf_rel: str*, *package_dir: str*, *config_dir: str | None = None*) → str

Resolve robot_urdf_path across absolute path, config dir, user workspace, and bundled package dir.

icub_pybullet.runtime_paths.**resolve_mesh_or_urdf_path**(*obj_path: str*, *package_dir: str*, *config_dir: str | None = None*) → str

Resolve an object .obj or .urdf path referenced in config.urdfs.paths across:

1. Absolute path
2. Relative to the loaded config file's directory
3. Relative to CWD
4. User workspace (~/Documents/pyCub/meshes and root)
5. Bundled icub_pybullet/other_meshes and package root

icub_pybullet.runtime_paths.**file_fingerprint**(*path: str*) → str

Fast fingerprint combining canonical path, size, and mtime_ns.

icub_pybullet.runtime_paths.**get_vhacd_cache_path**(*col_path: str*, *force_vhacd: bool = False*) → Tuple[str, bool]

Locate an existing pre-baked `*_vhacd.obj` or return a writable cache path. Returns (vhacd_mesh_path, needs_compute).

icub_pybullet.runtime_paths.**get_cached_urdf_path**(*stem: str*, *key: str*, *subdir: str = 'generated_urdf'*) → str

Return a deterministic path in the user cache for a generated/patched URDF.

icub_pybullet.runtime_paths.**local_ipv4_addresses**() → List[str]

Cross-platform IPv4 LAN addresses discovery (Windows, macOS, Linux).

icub_pybullet.runtime_paths.**terminate_process_tree**(*procs: Iterable*, *timeout: float = 5.0*) → None

Cross-platform termination of child processes and all their descendants using psutil.

icub_pybullet.utils

Utils for pyCub simulator

Author
Lukas Rustler

Attributes

JOINTS	
JOINTS_IDS	
CHAINS	

Classes

URDF	Class to parse URDF file
Config	Class to parse and keep the config loaded from yaml file.
Pose	Mini help class for Pose representation
CustomFormatter	Custom formatter that assigns colors to logs

Module Contents

class icub_pybullet.utils.**URDF**(*path: str*, *package_dir: str | None = None*)

Class to parse URDF file

ROOT_TAGS = []

path = b'.'

package_dir

links = []

joints = []

robot_name

new_urdf = ''

find_root_tags() → None

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

read(*el: xml.etree.ElementTree*, *parent: xml.etree.ElementTree*) → None

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.

dereference() → None

Make parent/child again as names to allow urdf write

make_references() → None

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

expand_package_uris() → None

Expand package:// mesh paths to absolute POSIX paths so cached URDFs load from any directory.

fix_urdf() → None

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 writable directory or user cache.

write_attr(*attr_name: str*, *attr: Any*, *level: int | None = 1*, *skip_header: bool | None = False*) → None

Write an attribute to the new URDF string.

write_urdf() → None

Write the URDF object to a string.

class icub_pybullet.utils.**Config**(*config_path: str*)

Class to parse and keep the config loaded from yaml file. Supports optional *parent: <base.yaml>* inheritance so user-defined custom worlds can override only what they need.

set_attribute(*attr: str*, *value: Any*, *reference: int*) → int

Recursively assign dictionary entries as named attributes on `reference`.

Parameters

- **attr** – Attribute name to assign.
- **value** – Scalar or nested dictionary value.
- **reference** – Target object or namedtuple receiving the attribute.

class icub_pybullet.utils.**Pose**(*pos: list*, *ori: list*)

Mini help class for Pose representation

pos

ori

to_string() → str

Serialize position and orientation components as a semicolon-delimited string.

class icub_pybullet.utils.**CustomFormatter**(*fmt=None*, *datefmt=None*, *style='%'*, *validate=True*, ** (Keyword-only parameters separator (PEP 3102))*, *defaults=None*)

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

grey = '\x1b[97;10m'

yellow = '\x1b[33;20m'

red = '\x1b[31;20m'

bold_red = '\x1b[31;1m'

reset = '\x1b[0m'

format = '%(module)s %(levelname)s: %(message)s'

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.

FORMATS

icub_pybullet.utils.**JOINTS**

icub_pybullet.utils.**JOINTS_IDS**

icub_pybullet.utils.**CHAINS**

icub_pybullet.visualizer

Visualization utils for pyCub simulator

Author
Lukas Rustler

Classes

Visualizer	Class to help with custom rendering

Module Contents

class icub_pybullet.visualizer.**Visualizer**(*client:* pyCub)

Class to help with custom rendering

client

meshes

R_urdf

skin_activated_last_turn

blue

red

gui

window

menu

scene

vis

mattr

mat

file_menu

show_menu

file_dir

orientations = []

positions = []

colors = []

paths = []

eye_windows

default_center

is_alive = True

reset_view() → None

Reset the camera view to the initial look-at pose.

show_first(*urdf_name: str | None = 'robot'*) → None

Load and register initial visual meshes and skin clouds for `urdf_name`.

show_mesh() → None

Update mesh transforms, eye camera poses, and skin activation colors.

read_info(*obj_id: int*) → int

Fetch current link poses, colors, and mesh file paths for body `obj_id`.

render() → None

Synchronize the 3D scene and eye windows with PyBullet and advance one GUI tick.

class MenuCallback(*menu_id: int*, *parent:* Visualizer)

Callable handler bound to Open3D Save/Show menu items.

id

parent

path = None

dialog_opened = False

save_image(*im: open3d.geometry.Image*, *mode: int*) → None

Prompt for an output path and write an RGB (mode 0) or depth (mode 1) image.

wait_for_dialog_completion() → None

Pump Open3D GUI events until the save-file dialog closes.

input_completed(*text: str | None = None*)

Store the selected file path and mark the file dialog complete.

find_xyz_rpy(*mesh_name: str*, *urdf_name: str | None = 'robot'*) → Tuple[list, list, float, str]

Look up visual origin offset, rotation, scale, and link name for `mesh_name`.

class EyeWindow(*eye: str*, *parent:* Visualizer)

Secondary Open3D viewport rendering the view from `l_eye` or `r_eye`.

MENU_IDS

POSITIONS

eye

link_name

parent

window

scene

vis

center = None

dir = None

img_counter = 0

save_images_bool = False

last_image = None

last_depth_image = None

on_close() → bool

Unregister this eye window and disable its stream menu entries.

on_mouse(*event: open3d.visualization.gui.MouseEvent*) → int

Consume mouse events so users cannot orbit the fixed eye camera view.

get_image() → None

Capture an asynchronous RGB snapshot from the eye scene.

get_depth_image() → None

Capture an asynchronous depth snapshot from the eye scene.

save_images() → None

Save the latest captured RGB frame to disk and increment the frame counter.

save_image(*im: open3d.geometry.Image*) → None

Callback storing the latest rendered RGB image.

save_depth_image(*im: open3d.geometry.Image*) → None

Callback storing the latest rendered depth image.

unproject(*u*, *v*, *d*)

Unproject pixel `(u, v)` at depth `d` into 3D world coordinates.

6.3 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

6.3.1 Functions

push_the_ball(client) → None	Example function to move the ball with cartesian control.
main() → NoReturn	Main function to run the example

6.3.2 Module Contents

push_the_ball_cartesian.**push_the_ball**(*client:* icub_pybullet.pycub.pyCub) → None

Example function to move the ball with cartesian control.

Parameters
client (pyCub) – instance of pyCub

Returns

Return type

push_the_ball_cartesian.**main**() → NoReturn

Main function to run the example

Returns

Return type

6.4 push_the_ball_pure_joints

Example of how to push the ball from the table using only pure joint control. It works without planner of collisions 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

6.4.1 Functions

push_the_ball(client) → None	Example function to push the ball from the table with joint control.
main() → NoReturn	Main function to run the example

6.4.2 Module Contents

push_the_ball_pure_joints.**push_the_ball**(*client:* icub_pybullet.pycub.pyCub) → None

Example function to push the ball from the table with joint control.

Parameters
client (pyCub) – instance of pyCub

Returns

Return type

push_the_ball_pure_joints.**main**() → NoReturn

Main function to run the example

Returns

Return type

PYTHON MODULE INDEX

i

icub_pybullet, 35
icub_pybullet.kid_api, 35
icub_pybullet.pycub, 42
icub_pybullet.runtime_paths, 47
icub_pybullet.utils, 49
icub_pybullet.visualizer, 52

p

push_the_ball_cartesian, 54
push_the_ball_pure_joints, 55

s

skin_test, 35

INDEX

B

ball_distance() (*icub_pybullet.kid_api.KidRobot method*), 38
ball_distance() (*in module icub_pybullet.kid_api*), 42
ball_hits() (*icub_pybullet.kid_api.KidRobot method*), 38
ball_hits() (*in module icub_pybullet.kid_api*), 42
ball_touched() (*icub_pybullet.kid_api.KidRobot method*), 38
ball_touched() (*in module icub_pybullet.kid_api*), 42
bbox_overlap() (*icub_pybullet.pycub.pyCub static method*), 45
bend_back() (*icub_pybullet.kid_api.KidRobot method*), 38
bend_back() (*in module icub_pybullet.kid_api*), 41
bend_elbow() (*icub_pybullet.kid_api.KidRobot method*), 38
bend_elbow() (*in module icub_pybullet.kid_api*), 40
bend_forward() (*icub_pybullet.kid_api.KidRobot method*), 38
bend_forward() (*in module icub_pybullet.kid_api*), 41
bend_sideways() (*icub_pybullet.kid_api.KidRobot method*), 38
bend_sideways() (*in module icub_pybullet.kid_api*), 41
blue (*icub_pybullet.visualizer.Visualizer attribute*), 52
bold_red (*icub_pybullet.utils.CustomFormatter attribute*), 51
both_arms_down() (*icub_pybullet.kid_api.KidRobot method*), 37
both_arms_down() (*in module icub_pybullet.kid_api*), 39
both_arms_up() (*icub_pybullet.kid_api.KidRobot method*), 37
both_arms_up() (*in module icub_pybullet.kid_api*), 39

C

center (*icub_pybullet.visualizer.Visualizer.EyeWindow attribute*), 54
CHAINS (*in module icub_pybullet.utils*), 52
check_stop() (*icub_pybullet.kid_api.KidRobot method*), 37
client (*icub_pybullet.kid_api.KidRobot attribute*), 37
client (*icub_pybullet.pycub.EndEffector attribute*), 47
client (*icub_pybullet.visualizer.Visualizer attribute*), 52
collision_during_motion (*icub_pybullet.pycub.pyCub attribute*), 44
colors (*icub_pybullet.visualizer.Visualizer attribute*), 53
compute_jacobian() (*icub_pybullet.pycub.pyCub method*), 45
compute_skin() (*icub_pybullet.pycub.pyCub method*), 45
Config (*class in icub_pybullet.utils*), 51
config (*icub_pybullet.pycub.pyCub attribute*), 44
config_dir (*icub_pybullet.pycub.pyCub attribute*), 44
config_path (*icub_pybullet.pycub.pyCub attribute*), 44
contactPoints (*icub_pybullet.pycub.pyCub attribute*), 44
create_urdf() (*icub_pybullet.pycub.pyCub method*), 47
CustomFormatter (*class in icub_pybullet.utils*), 51

D

default_center (*icub_pybullet.visualizer.Visualizer attribute*), 53
dereference() (*icub_pybullet.utils.URDF method*), 50
dialog_opened (*icub_pybullet.visualizer.Visualizer.MenuCallback attribute*), 53
dir (*icub_pybullet.visualizer.Visualizer.EyeWindow attribute*), 54
dynamicsInfo (*icub_pybullet.pycub.pyCub attribute*), 44

E

end_effector (*icub_pybullet.pycub.pyCub attribute*), 44
EndEffector (*class in icub_pybullet.pycub*), 47
ensure_user_workspace() (*in module icub_pybullet.runtime_paths*), 48
expand_package_uris() (*icub_pybullet.utils.URDF method*), 50
eye (*icub_pybullet.visualizer.Visualizer.EyeWindow attribute*), 53
eye_windows (*icub_pybullet.visualizer.Visualizer attribute*), 53

F

file_dir (*icub_pybullet.pycub.pyCub attribute*), 44
file_dir (*icub_pybullet.visualizer.Visualizer attribute*), 52
file_fingerprint() (*in module icub_pybullet.runtime_paths*), 49
file_menu (*icub_pybullet.visualizer.Visualizer attribute*), 52
find_joint_id() (*icub_pybullet.pycub.pyCub method*), 46
find_link_id() (*icub_pybullet.pycub.pyCub method*), 46
find_processes_by_name() (*icub_pybullet.pycub.pyCub method*), 45
find_root_tags() (*icub_pybullet.utils.URDF method*), 50
find_xyz_rpy() (*icub_pybullet.visualizer.Visualizer method*), 53
fix_urdf() (*icub_pybullet.utils.URDF method*), 51
format (*icub_pybullet.utils.CustomFormatter attribute*), 51
FORMATS (*icub_pybullet.utils.CustomFormatter attribute*), 52
free_objects (*icub_pybullet.pycub.pyCub attribute*), 44

G

get_cached_urdf_path() (*in module icub_pybullet.runtime_paths*), 49
get_camera_depth_images() (*icub_pybullet.pycub.pyCub method*), 45
get_camera_images() (*icub_pybullet.pycub.pyCub method*), 45
get_chains() (*icub_pybullet.pycub.pyCub static method*), 44
get_depth_image() (*icub_pybullet.visualizer.Visualizer.EyeWindow method*), 54
get_extra_packages_dir() (*in module icub_pybullet.runtime_paths*), 48
get_image() (*icub_pybullet.visualizer.Visualizer.EyeWindow method*), 54
get_joint() (*icub_pybullet.kid_api.KidRobot method*), 37
get_joint() (*in module icub_pybullet.kid_api*), 42
get_joint_state() (*icub_pybullet.pycub.pyCub method*), 46
get_position() (*icub_pybullet.pycub.EndEffector method*), 47
get_robot() (*in module icub_pybullet.kid_api*), 39
get_user_cache_dir() (*in module icub_pybullet.runtime_paths*), 48
get_user_log_dir() (*in module icub_pybullet.runtime_paths*), 48
get_user_workspace_dir() (*in module icub_pybullet.runtime_paths*), 48
get_vhacd_cache_path() (*in module icub_pybullet.runtime_paths*), 49
get_visualizer_factory() (*in module icub_pybullet.pycub*), 43
gravity (*icub_pybullet.pycub.pyCub attribute*), 44
grey (*icub_pybullet.utils.CustomFormatter attribute*), 51
gui (*icub_pybullet.visualizer.Visualizer attribute*), 52

H

highlight_block() (*icub_pybullet.kid_api.KidRobot method*), 37

I

icub_pybullet
    module, 35
icub_pybullet.kid_api
    module, 35
icub_pybullet.pycub
    module, 42
icub_pybullet.runtime_paths
    module, 47
icub_pybullet.utils
    module, 49
icub_pybullet.visualizer
    module, 52
id (*icub_pybullet.visualizer.Visualizer.MenuCallback attribute*), 53
IK_config (*icub_pybullet.pycub.pyCub attribute*), 44
img_counter (*icub_pybullet.visualizer.Visualizer.EyeWindow attribute*), 54
init_robot() (*icub_pybullet.pycub.pyCub method*), 45
init_urdfs() (*icub_pybullet.pycub.pyCub method*), 45
input_completed() (*icub_pybullet.visualizer.Visualizer.MenuCallback method*), 53
is_alive (*icub_pybullet.visualizer.Visualizer attribute*), 53
is_alive() (*icub_pybullet.pycub.pyCub method*), 45
is_frozen() (*in module icub_pybullet.runtime_paths*), 48
is_writable_dir() (*in module icub_pybullet.runtime_paths*), 48

J

jog_joint() (*icub_pybullet.kid_api.KidRobot method*), 37
jog_joint() (*in module icub_pybullet.kid_api*), 42
Joint (*class in icub_pybullet.pycub*), 43
joint_tolerance (*icub_pybullet.pycub.pyCub attribute*), 44
jointInfo (*icub_pybullet.pycub.pyCub attribute*), 44
joints (*icub_pybullet.utils.URDF attribute*), 50
JOINTS (*in module icub_pybullet.utils*), 52

joints_id (*icub_pybullet.pycub.Joint attribute*), 43
JOINTS_IDS (*in module icub_pybullet.utils*), 52
jointStates (*icub_pybullet.pycub.pyCub attribute*), 44

K

KidRobot (*class in icub_pybullet.kid_api*), 37
KidScriptStopped, 37
kill_open3d() (*icub_pybullet.pycub.pyCub method*), 45

L

last_depth_image (*icub_pybullet.visualizer.Visualizer.EyeWindow attribute*), 54
last_image (*icub_pybullet.visualizer.Visualizer.EyeWindow attribute*), 54
last_log (*icub_pybullet.pycub.pyCub attribute*), 44
last_step (*icub_pybullet.pycub.pyCub attribute*), 44
left_wave() (*in module icub_pybullet.kid_api*), 39
Link (*class in icub_pybullet.pycub*), 43
link_name (*icub_pybullet.visualizer.Visualizer.EyeWindow attribute*), 53
linkInfo (*icub_pybullet.pycub.pyCub attribute*), 44
links (*icub_pybullet.utils.URDF attribute*), 50
local_ipv4_addresses() (*in module icub_pybullet.runtime_paths*), 49
logger (*icub_pybullet.pycub.pyCub attribute*), 44
look_center() (*icub_pybullet.kid_api.KidRobot method*), 37
look_center() (*in module icub_pybullet.kid_api*), 39
look_down() (*icub_pybullet.kid_api.KidRobot method*), 38
look_down() (*in module icub_pybullet.kid_api*), 41
look_up() (*icub_pybullet.kid_api.KidRobot method*), 38
look_up() (*in module icub_pybullet.kid_api*), 41
lower_limit (*icub_pybullet.pycub.Joint attribute*), 43

M

main() (*in module push_the_ball_cartesian*), 55
main() (*in module push_the_ball_pure_joints*), 55
main() (*in module skin_test*), 35
make_references() (*icub_pybullet.utils.URDF method*), 50
mat (*icub_pybullet.visualizer.Visualizer attribute*), 52
mattr (*icub_pybullet.visualizer.Visualizer attribute*), 52
max_force (*icub_pybullet.pycub.Joint attribute*), 43
max_velocity (*icub_pybullet.pycub.Joint attribute*), 43
menu (*icub_pybullet.visualizer.Visualizer attribute*), 52
MENU_IDS (*icub_pybullet.visualizer.Visualizer.EyeWindow attribute*), 53
meshes (*icub_pybullet.visualizer.Visualizer attribute*), 52
module
    icub_pybullet, 35
    icub_pybullet.kid_api, 35
    icub_pybullet.pycub, 42
    icub_pybullet.runtime_paths, 47
    icub_pybullet.utils, 49
    icub_pybullet.visualizer, 52
    push_the_ball_cartesian, 54
    push_the_ball_pure_joints, 55
    skin_test, 35
motion_done() (*icub_pybullet.pycub.pyCub method*), 46
move_cartesian() (*icub_pybullet.pycub.pyCub method*), 46
move_joint() (*icub_pybullet.kid_api.KidRobot method*), 37
move_joint() (*in module icub_pybullet.kid_api*), 41
move_position() (*icub_pybullet.pycub.pyCub method*), 45
move_velocity() (*icub_pybullet.pycub.pyCub method*), 46

N

name (*icub_pybullet.pycub.EndEffector attribute*), 47
name (*icub_pybullet.pycub.Joint attribute*), 43
name (*icub_pybullet.pycub.Link attribute*), 43
neighbour_links (*icub_pybullet.pycub.pyCub attribute*), 44
new_urdf (*icub_pybullet.utils.URDF attribute*), 50

O

on_close() (*icub_pybullet.visualizer.Visualizer.EyeWindow method*), 54
on_mouse() (*icub_pybullet.visualizer.Visualizer.EyeWindow method*), 54
ori (*icub_pybullet.utils.Pose attribute*), 51
orientations (*icub_pybullet.visualizer.Visualizer attribute*), 52
other_objects (*icub_pybullet.pycub.pyCub attribute*), 44

P

package_dir (*icub_pybullet.utils.URDF attribute*), 50
parent (*icub_pybullet.visualizer.Visualizer.EyeWindow attribute*), 54
parent (*icub_pybullet.visualizer.Visualizer.MenuCallback attribute*), 53
parent_name (*icub_pybullet.pycub.pyCub attribute*), 44
path (*icub_pybullet.utils.URDF attribute*), 50
path (*icub_pybullet.visualizer.Visualizer.MenuCallback attribute*), 53
paths (*icub_pybullet.visualizer.Visualizer attribute*), 53
pos (*icub_pybullet.utils.Pose attribute*), 51
Pose (*class in icub_pybullet.utils*), 51
positions (*icub_pybullet.visualizer.Visualizer attribute*), 52
POSITIONS (*icub_pybullet.visualizer.Visualizer.EyeWindow attribute*), 53

precomputed_link_ids (*icub_pybullet.pycub.pyCub attribute*), 44
prepare_log() (*icub_pybullet.pycub.pyCub method*), 45
print_collision_info() (*icub_pybullet.pycub.pyCub method*), 47
push_the_ball() (*in module push_the_ball_cartesian*), 55
push_the_ball() (*in module push_the_ball_pure_joints*), 55
push_the_ball_cartesian module, 54
push_the_ball_pure_joints module, 55
pyCub (*class in icub_pybullet.pycub*), 44

R

R_urdf (*icub_pybullet.visualizer.Visualizer attribute*), 52
read() (*icub_pybullet.utils.URDF method*), 50
read_info() (*icub_pybullet.visualizer.Visualizer method*), 53
red (*icub_pybullet.utils.CustomFormatter attribute*), 51
red (*icub_pybullet.visualizer.Visualizer attribute*), 52
render() (*icub_pybullet.visualizer.Visualizer method*), 53
reset (*icub_pybullet.utils.CustomFormatter attribute*), 51
reset_pose() (*icub_pybullet.kid_api.KidRobot method*), 37
reset_pose() (*in module icub_pybullet.kid_api*), 39
reset_view() (*icub_pybullet.visualizer.Visualizer method*), 53
resolve_config_path() (*in module icub_pybullet.runtime_paths*), 48
resolve_mesh_or_urdf_path() (*in module icub_pybullet.runtime_paths*), 49
resolve_robot_urdf_path() (*in module icub_pybullet.runtime_paths*), 49
right_wave() (*in module icub_pybullet.kid_api*), 39
robot_joint_id (*icub_pybullet.pycub.Joint attribute*), 43
robot_link_id (*icub_pybullet.pycub.Link attribute*), 43
robot_name (*icub_pybullet.utils.URDF attribute*), 50
ROOT_TAGS (*icub_pybullet.utils.URDF attribute*), 50
rotate_arm() (*icub_pybullet.kid_api.KidRobot method*), 38
rotate_arm() (*in module icub_pybullet.kid_api*), 40
rotate_wrist() (*icub_pybullet.kid_api.KidRobot method*), 38
rotate_wrist() (*in module icub_pybullet.kid_api*), 40
rtb_robot (*icub_pybullet.pycub.pyCub property*), 44
run_vhacd() (*icub_pybullet.pycub.pyCub method*), 46

S

save_depth_image() (*icub_pybullet.visualizer.Visualizer.EyeWindow method*), 54
save_image() (*icub_pybullet.visualizer.Visualizer.EyeWindow method*), 54
save_image() (*icub_pybullet.visualizer.Visualizer.MenuCallback method*), 53
save_images() (*icub_pybullet.visualizer.Visualizer.EyeWindow method*), 54
save_images_bool (*icub_pybullet.visualizer.Visualizer.EyeWindow attribute*), 54
scale_bbox() (*icub_pybullet.pycub.pyCub static method*), 45
scene (*icub_pybullet.visualizer.Visualizer attribute*), 52
scene (*icub_pybullet.visualizer.Visualizer.EyeWindow attribute*), 54
set_attribute() (*icub_pybullet.utils.Config method*), 51
set_point (*icub_pybullet.pycub.Joint attribute*), 43
set_status_callback() (*in module icub_pybullet.pycub*), 43
set_visualizer_factory() (*in module icub_pybullet.pycub*), 43
shoulder_down() (*icub_pybullet.kid_api.KidRobot method*), 37
shoulder_down() (*in module icub_pybullet.kid_api*), 39
shoulder_in() (*icub_pybullet.kid_api.KidRobot method*), 38
shoulder_in() (*in module icub_pybullet.kid_api*), 40
shoulder_out() (*icub_pybullet.kid_api.KidRobot method*), 38
shoulder_out() (*in module icub_pybullet.kid_api*), 39
shoulder_up() (*icub_pybullet.kid_api.KidRobot method*), 37
shoulder_up() (*in module icub_pybullet.kid_api*), 39
show_first() (*icub_pybullet.visualizer.Visualizer method*), 53
show_menu (*icub_pybullet.visualizer.Visualizer attribute*), 52
show_mesh() (*icub_pybullet.visualizer.Visualizer method*), 53
skin_activated_last_turn (*icub_pybullet.visualizer.Visualizer attribute*), 52
skin_test module, 35
skin_touched() (*icub_pybullet.kid_api.KidRobot method*), 38
skin_touched() (*in module icub_pybullet.kid_api*), 42
SKIP_IFACE_PREFIXES (*in module icub_pybullet.runtime_paths*), 48
stand_straight() (*icub_pybullet.kid_api.KidRobot method*), 37

stand_straight() (*in module icub_pybullet.kid_api*), 39
start_time (*icub_pybullet.pycub.Joint attribute*), 43
steps_done (*icub_pybullet.pycub.pyCub attribute*), 44
stop() (*icub_pybullet.kid_api.KidRobot method*), 37
stop() (*in module icub_pybullet.kid_api*), 39
stop_robot() (*icub_pybullet.pycub.pyCub method*), 46
straighten_elbow() (*icub_pybullet.kid_api.KidRobot method*), 38
straighten_elbow() (*in module icub_pybullet.kid_api*), 40

T

terminate_process_tree() (*in module icub_pybullet.runtime_paths*), 49
tilt_head() (*icub_pybullet.kid_api.KidRobot method*), 38
tilt_head() (*in module icub_pybullet.kid_api*), 41
timeout (*icub_pybullet.pycub.Joint attribute*), 43
to_posix() (*in module icub_pybullet.runtime_paths*), 48
to_string() (*icub_pybullet.utils.Pose method*), 51
toggle_gravity() (*icub_pybullet.pycub.pyCub method*), 45
turn_body() (*icub_pybullet.kid_api.KidRobot method*), 38
turn_body() (*in module icub_pybullet.kid_api*), 41
turn_head() (*icub_pybullet.kid_api.KidRobot method*), 38
turn_head() (*in module icub_pybullet.kid_api*), 40

U

unproject() (*icub_pybullet.visualizer.Visualizer.EyeWindow method*), 54
update_simulation() (*icub_pybullet.pycub.pyCub method*), 45
upper_limit (*icub_pybullet.pycub.Joint attribute*), 43
URDF (*class in icub_pybullet.utils*), 50
urdf_link (*icub_pybullet.pycub.Link attribute*), 43
urdf_path (*icub_pybullet.pycub.pyCub attribute*), 44
urdfs (*icub_pybullet.pycub.pyCub attribute*), 44
user_documents_dir() (*in module icub_pybullet.runtime_paths*), 48

V

vis (*icub_pybullet.visualizer.Visualizer attribute*), 52
vis (*icub_pybullet.visualizer.Visualizer.EyeWindow attribute*), 54
Visualizer (*class in icub_pybullet.visualizer*), 52
Visualizer.EyeWindow (*class in icub_pybullet.visualizer*), 53
Visualizer.MenuCallback (*class in icub_pybullet.visualizer*), 53
visualShapeData (*icub_pybullet.pycub.pyCub attribute*), 44

W

wait() (*icub_pybullet.kid_api.KidRobot method*), 38
wait() (*in module icub_pybullet.kid_api*), 42
wait_for_dialog_completion() (*icub_pybullet.visualizer.Visualizer.MenuCallback method*), 53
wait_motion_done() (*icub_pybullet.pycub.pyCub method*), 46
wave() (*icub_pybullet.kid_api.KidRobot method*), 37
wave() (*in module icub_pybullet.kid_api*), 39
window (*icub_pybullet.visualizer.Visualizer attribute*), 52
window (*icub_pybullet.visualizer.Visualizer.EyeWindow attribute*), 54
write_attr() (*icub_pybullet.utils.URDF method*), 51
write_text_atomic() (*in module icub_pybullet.runtime_paths*), 48
write_urdf() (*icub_pybullet.utils.URDF method*), 51

Y

yellow (*icub_pybullet.utils.CustomFormatter attribute*), 51