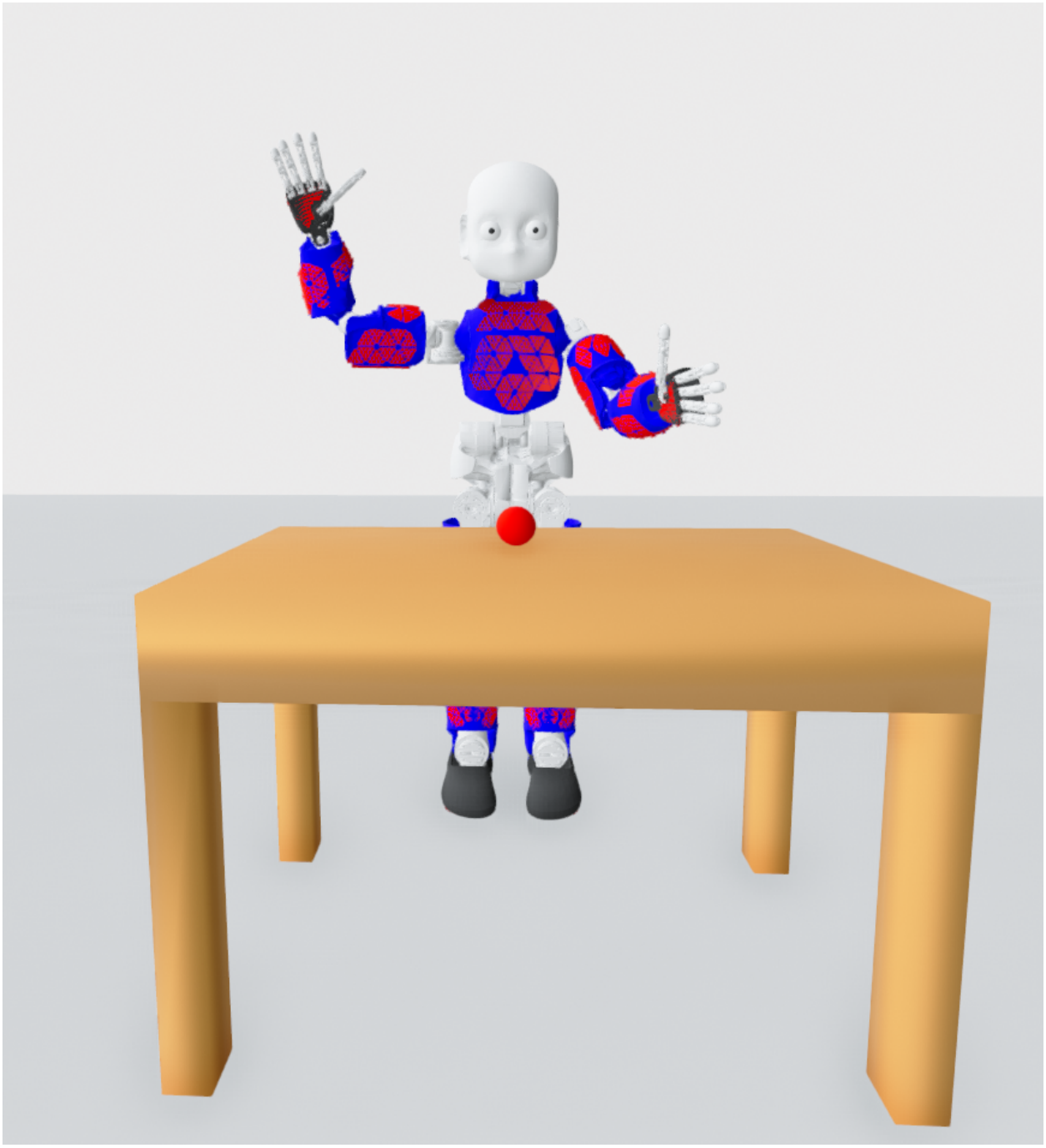


Příručka pyCub Studia

Lukas Rustler

26.11.2026

Obsah

1 Použití Studia **3**
- 1.1 Začínáme . . . 3
- 1.2 Dětské ovládání (Tlačítka) . . . 3
- 1.3 Dětské „Scratch“ (Bloky) . . . 3
- 1.4 Dětské programování (Python) . . . 4
- 1.5 Normální režim (Plné IDE & .py) . . . 4
- 1.6 Webový server pro třídu (Multi-instance) . . . 4
- 1.7 Složka uživatele a balíčky . . . 4

2 Programování robota **5**
- 2.1 Dětské režimy (`icub_pybullet.kid_api`) . . . 5
- 2.2 Plný režim Studia (Normální režim IDE) . . . 7
- 2.3 Normální / samostatný Python (mimo Studio) . . . 9
- 2.4 Zobrazení nápovědy a definice funkce v editoru . . . 10
- 2.5 Přehled vybraných metod a senzorů . . . 10

3 Ukázkové programy **13**
- 3.1 Kloubové řízení . . . 13
- 3.2 Kartézské řízení . . . 14
- 3.3 Dotyková kůže . . . 15
- 3.4 Minimální smyčka simulace . . . 15

4 Kurzovní cvičení **17**
- 4.1 Odtlačení míčku (Push the Ball) . . . 17
- 4.2 Plynulé pohyby (Smooth movements) . . . 19
- 4.3 Řízení pohledu (Gaze control) . . . 20
- 4.4 Řízení pomocí jakobiánu (Resolved-Rate Motion Control) . . . 22
- 4.5 Uchopení míčku (Grasp It!) . . . 24

5 Dokumentace projektu (README) **27**
- 5.1 pyCub Studio (Standalone Desktop Application) . . . 28
- 5.2 Building `pyCub Studio` (Native Installers & Bundles) . . . 30
- 5.3 Python Package Installation (`icub_pybullet` & `pycub-studio`) . . . 32
- 5.4 Documentation & Building the Handbook . . . 33
- 5.5 FAQ & Known Issues . . . 33
- 5.6 Docker . . . 34
- 5.7 Citing . . . 35

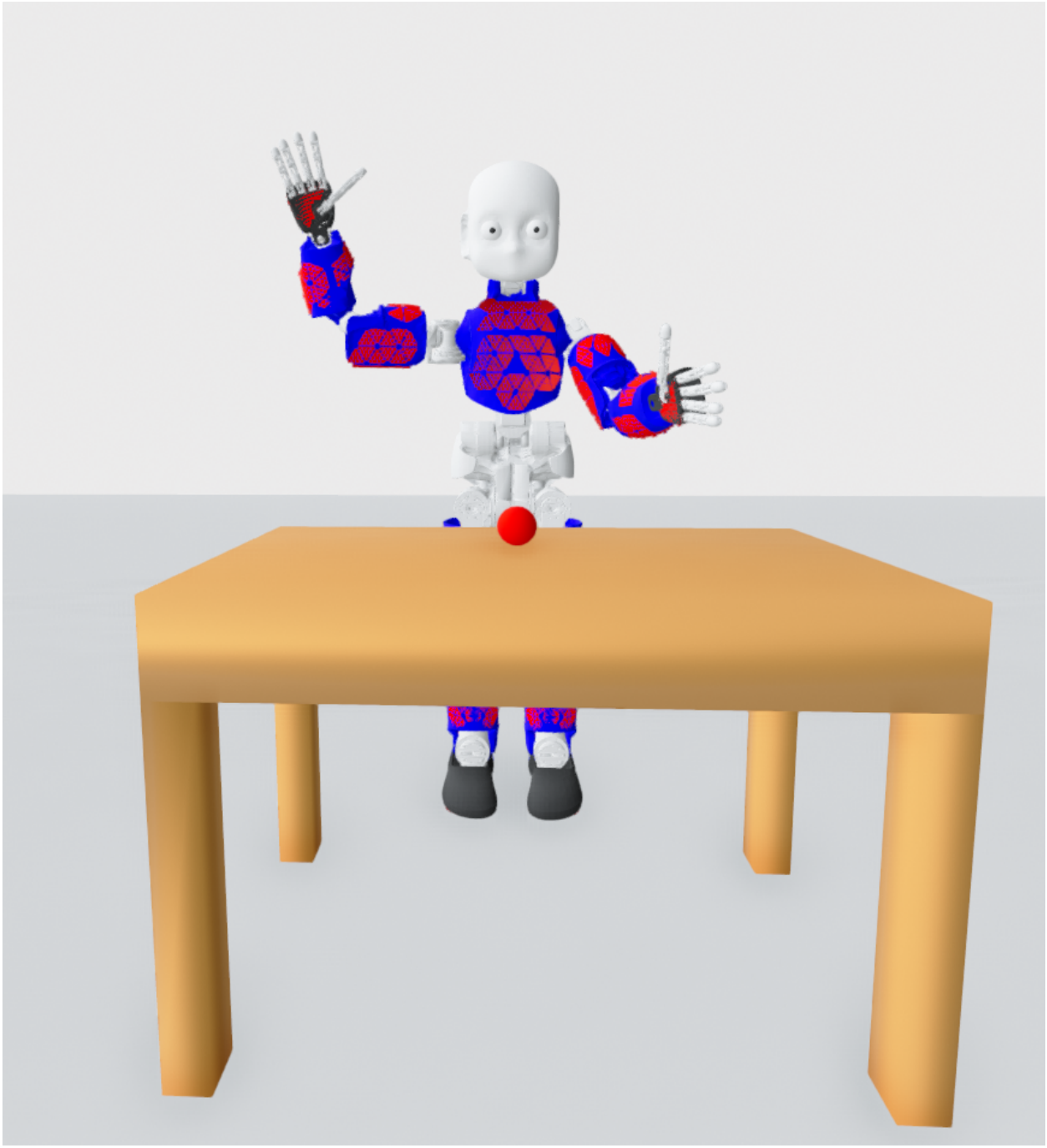
5.8 License 35

6 API Reference **37**
6.1 skin_test 37
6.2 icub_pybullet 37
6.3 push_the_ball_cartesian 56
6.4 push_the_ball_pure_joints 57

Python Module Index **59**

Rejstřík **61**

Naučte se ovládat humanoidního robota iCub v simulaci. Začněte jednoduchým zamáváním, napište vlastní řídicí program a otestujte jej v kurzovních cvičeních.



- *Použití Studia*

  Vyberte režim, nastavte pracovní složku a spusťte simulaci.

- *Programování robota*

  Pohybujte klouby, čtěte senzory a prohlížejte nápovědu k funkcím v Pythonu.

- *Ukázkové programy*

Prozkoumejte hotové ukázky kloubového, kartézského i dotykového řízení.

- *Kurzovní cvičení*

  Procvičte si plánování pohybu, řízení pohledu a uchopení objektů.

- *Dokumentace projektu (README)*

  Instalace, Docker, přehled funkcí a citace (v angličtině).

- *API reference*

  Přehled tříd, funkcí a jejich parametrů (v angličtině podle zdrojového kódu).

KAPITOLA 1

Použití Studia

Spusťte pyCub Studio a na domovské obrazovce (**Domů**) vyberte jeden z režimů. Přepínač režimů v horní liště umožňuje kdykoli přecházet mezi aktivitami. Tlačítko jazyka přepíná rozhraní i příručku mezi češtinou a angličtinou.

1.1 Začínáme

Otevřete desktopovou aplikaci z instalátoru nebo přenosné (portable) verze, nebo ji nainstalujte přes `pip` do virtuálního prostředí Pythonu 3.10–3.12 a spusťte příkazem `pycub-studio` odkudkoli:

```
# Z PyPI (nainstaluje simulátor, Studio, příručku, cvičení i ukázky):
python -m pip install pycub-studio
# nebo: python -m pip install "icub_pybullet[studio]"

# Nebo z lokálně naklonovaného repozitáře:
python -m pip install ./icub_pybullet ./studio

pycub-studio
```

1.2 Dětské ovládání (Tlačítka)

Vyberte svět s míčkem a vyzkoušejte některý z přednastavených pohybů. Pomocí tlačítek kloubů můžete pohybovat jednotlivými částmi robota. Změnou velikosti kroku (°) a rychlosti upravíte jemnost pohybu. Ukazatel míčku zobrazuje, zda se robot míčku dotkl a jak daleko se míček odkutálel.

1.3 Dětské „Scratch“ (Bloky)

Skládejte barevné bloky pro pohyb, senzory a řízení programu (cykly, podmínky) a spusťte program. Během běhu se v editoru zvýrazňuje právě vykonávaný blok. V náhledu Python kódu uvidíte, jak se bloky překládají do Pythonu. Rozpracované projekty lze uložit jako soubory `.pycub_blocks` nebo jedním kliknutím převést do režimu Dětského programování.

1.4 Dětské programování (Python)

Začněte s připravenou šablonou nebo klikněte na příkaz v paletě vlevo pro vložení ukázky do kódu. Spusťte program a sledujte výstup v konzoli. Tlačítkem **Zastavit** přerušíte běžící smyčku. Případná chyba zvýrazní příslušný řádek v editoru. Podrobnosti k nápovědě a definicím funkcí najdete v části *Programování robota*.

1.5 Normální režim (Plné IDE & .py)

Otevřete vlastní `.py` soubor nebo upravte výchozí skript v první záložce. Klávesou **F5** nebo tlačítkem **Spustit skript** spustíte aktuální program. V záložkách lze upravovat i `.yaml` konfigurace světů. Pokud píšete kód v externím editoru (např. VS Code nebo PyCharm), použijte **Spustit externí .py…** a volbu automatického znovuspuštění při uložení souboru.

Pomocí tlačítek pro levé a pravé oko (`l_eye`, `r_eye`) zobrazíte náhled RGB a hloubkových kamer. V nabídce **Příklady & Cvičení** můžete otevřít ukázkové skripty nebo nakopírovat kurzovní cvičení do své pracovní složky a rovnou spouštět jejich automatický tester s vyhodnocením výsledků.

1.6 Webový server pro třídu (Multi-instance)

Nastavte počet samostatných instancí simulátoru (`N`) a spusťte server. Studenti ve stejné síti se mohou připojit přes zobrazenou adresu nebo naskenováním QR kódu z tabletu či jiného počítače. Přehled zobrazuje připojené uživatele a stav každé instance; v případě potřeby můžete zobrazit logy nebo jednotlivou instanci restartovat.

1.7 Složka uživatele a balíčky

Tlačítkem **Složka uživatele** v horní liště otevřete adresář `~/Documents/pyCub` s vašimi skripty, Scratch projekty, ukázkami, cvičeními a uloženými obrázky. Vlastní `.yaml` světy ukládejte do podsložky `worlds/` (mohou dědit výchozí svět pomocí `parent: with_ball.yaml`) a vlastní 3D modely do `meshes/`. Přes **Balíčky / Python** lze doinstalovat další Python knihovny nebo vybrat externí interpret Pythonu pro Normální režim.

KAPITOLA 2

Programování robota

pyCub nabízí tři hlavní způsoby programování robota podle úrovně zkušeností a podle toho, zda chcete pracovat v integrovaném prostředí Studia, nebo ze samostatného Pythonu:

Režim / přístup	Hlavní API	Jednotky	Simulační smyčka	Vhodné pro
Dětské režimy (Scratch a Dětské programování ve Studiu)	`icub_pybull kid_api`	**Stupně**	Řízeno automaticky	Začátečníky, výuku ve třídě, rychlé pokusy
Plný režim Studia (Normální režim IDE ve Studiu)	`icub_pybull pycub. pyCub`	**Radiány**	Explicitní (`wait=True` nebo `update_simulation(`	Plné řízení robota, kamery, kůže, kurzovní cvičení
Normální / samostatný Python (Terminál / PyCharm / VS Code)	`pyCub` nebo `kid_api`	Radiány (`pyCub`) / Stupně (`kid_api`)	Explicitní (`pyCub`) / Automatická (`kid_api`)	Automatické testy, headless běh, samostatné Open3D/Web okno

*(Poznámka: v konfiguračních souborech `.yaml` se `initial_joint_angles` zadávají vždy ve **stupních** bez ohledu na režim.)*

2.1 Dětské režimy (`icub_pybullet.kid_api`)

Dětské režimy v **pyCub Studiu** — **Dětské „Scratch" (Bloky)** i **Dětské programování (Python)** — používají stejné vysokoúrovňové rozhraní `icub_pybullet.kid_api`.

2.1.1 Jak dětské režimy fungují

- Studio udržuje na pozadí běžící interaktivní fyzikální proces pro právě vybraný svět (`with_ball.yaml`, `default.yaml`, `skin_test.yaml` nebo vlastní `.yaml` svět ze složky `~/Documents/pyCub/worlds/`).
- V kódu **nevytváříte** instanci `pyCub(...)` ani **nepíšete** smyčku `while client.is_alive(): client.update_simulation()`.

- Každý pohybový příkaz počká na dokončení pohybu (nebo vypršení limitu), plynule krokuje fyziku a průběžně kontroluje tlačítko **Zastavit** (`KidScriptStopped`), takže i nekonečnou smyčku lze okamžitě přerušit.

2.1.2 Od bloků Scratch k Pythonu

V režimu **Dětské „Scratch“** skládáte bloky pro pohyb, cykly, podmínky a senzory. Pomocí **Náhled Python kódu** vidíte živý překlad bloků do Pythonu, projekt si můžete uložit jako `.pycub_blocks` nebo jej jedním kliknutím převést tlačítkem **Otevřít jako Python v Dětském programování** do textového editoru.

2.1.3 První program v Dětském programování

Vyberte svět s míčkem (*Stůl s míčkem* / `with_ball.yaml`) a spusťte ukázku (lze použít české i anglické názvy příkazů):

```
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())
```

Ekvivalentní zápis v angličtině:

```
from icub_pybullet.kid_api import look_down, wave, shoulder_up, 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 automaticky zpřístupňuje všechny české i anglické příkazy pro našeptávač a nápovědu; obě varianty sdílejí tentýž řadič robota. Pojmenované argumenty vždy používají názvy z Python signatury (`side`, `degrees`, `speed`, `target_degrees`, `delta_degrees`, `seconds`).

2.1.4 Přehled skupin dětských příkazů

- **Hotové pohyby:** `zamávej(side="right")` / `wave`, `zamávej_levou()` / `left_wave`, `zamávej_pravou()` / `right_wave`, `zvedni_obě_ruce()` / `both_arms_up`, `spusť_obě_ruce()` / `both_arms_down`, `podívej_se_před_sebe()` / `look_center`, `narovnej_se()` / `stand_straight`, `obnov_polohu()` / `reset_pose`, `zastav()` / `stop`.
- **Relativní pohyby paží (ve stupních):** `zvedni_rameno` / `shoulder_up`, `spusť_rameno` / `shoulder_down`, `upaž` / `shoulder_out`, `připaž` / `shoulder_in`, `otoč_paži` / `rotate_arm`, `ohni_loket` / `bend_elbow`, `narovnej_loket` / `straighten_elbow`, `otoč_zápěstí` / `rotate_wrist`. Přijímají `side` (`"pravá"` / `"levá"` nebo `"right"` / `"left"`), `degrees=15.0` a `speed=1.5` (rad/s).
- **Relativní pohyby hlavy a trupu (ve stupních):** `otoč_hlavu` / `turn_head`, `podívej_se_nahoru` / `look_up`, `podívej_se_dolů` / `look_down`, `nakloň_hlavu` / `tilt_head`, `otoč_trup` / `turn_body`, `předkloň_se` / `bend_forward`, `zakloň_se` / `bend_back`, `ukloň_se` / `bend_sideways`.
- **Přímá práce s klouby a čekání:** `nastav_kloub(joint_name, target_degrees, speed=1.5)` / `move_joint` pro absolutní úhel, `posuň_kloub(joint_name, delta_degrees, speed=1.5)` / `jog_joint` pro relativní posun, `zjisti_úhel_kloubu(joint_name)` / `get_joint` pro přečtení aktuálního úhlu ve stupních a `počkej(seconds=1.0)` / `wait` pro pozastavení za běhu fyziky.

- **Senzory a míček:** `vzdálenost_míčku()` / `ball_distance()` (vzdálenost míčku od stolu v cm), `dotkl_se_míčku()` / `ball_touched()` (`bool`), `počet_zásahů()` / `ball_hits()` (`int`) a `dotyk_kůže()` / `skin_touched()` (`bool`, vyžaduje `skin.use: True`).

Kliknutím na položku v **Paletě příkazů** vlevo vložíte ukázkový příkaz přímo na pozici kurzoru, případně můžete začít s připravenou šablonou. Při chybě se problémový řádek zvýrazní červeně a v konzoli se zobrazí srozumitelná nápověda v češtině.

2.2 Plný režim Studia (Normální režim IDE)

Normální režim v **pyCub Studiu** poskytuje kompletní rozhraní `icub_pybullet.pycub.pyCub` společně s vestavěným hardwarově akcelerovaným 3D zobrazením, živou konzolí, náhledy očních kamer a nástroji pro kurzovní cvičení.

2.2.1 Spouštění skriptů v Normálním režimu Studia

- Upravujte `.py` soubory i `.yaml` konfigurace světů v záložkách editoru a stiskněte **F5** (nebo klikněte na **Spustit skript**).
- Každý běh se spouští v izolovaném podprocesu (`SimProcessSupervisor`), jehož 3D obraz se vykresluje přímo do vestavěného OpenGL okna Studia — neotevírá se žádné samostatné okno Open3D.
- Pokud píšete kód v externím editoru (PyCharm, VS Code) a chcete používat 3D náhled a kamery Studia, použijte **Spustit externí .py…** se zaškrtnutou volbou **Sledovat změny externího souboru**, nebo Studio spusťte z terminálu:

```
python -m studio.main --run-script cesta/k/vasemu_skriptu.py
```

2.2.2 Základní kostra skriptu pro pyCub ve Studiu

V plném / Normálním režimu vytvoříte instanci `pyCub`, předáte jí `.yaml` konfiguraci světa, zadáváte polohy kloubů v **radiánech** a na konci udržíte okno živé smyčkou do stisknutí tlačítka **Zastavit**:

```
from icub_pybullet.pycub import pyCub

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

# Blokující pohyb (počká na dosažení cíle, kolizi nebo vypršení limitu)
client.move_position("neck_pitch", -0.25)

# Neblokující pohyb: simulaci krokujete ve vlastní smyčce
client.move_position(["torso_pitch", "torso_yaw"], [0.30, -0.60], wait=False, velocity=2.
↪0)
while not client.motion_done():
    client.update_simulation()

# Po dokončení pohybu ponechte simulaci aktivní, dokud uživatel neklikne na Zastavit
while client.is_alive():
    client.update_simulation()
```

2.2.3 Kloubové, rychlostní a kartézské řízení

- **Polohové řízení (`move_position`):** Přijímá název nebo ID jednoho kloubu, případně stejně dlouhé seznamy kloubů a cílových úhlů v **radiánech**. Volitelné parametry: `wait=True`, `velocity=1` (rad/s), `set_col_state=True`, `check_collision=True` a `timeout=None` (v sekundách). Při `wait=False` volejte v cyklu `client.update_simulation()`, dokud `client.motion_done()` nevrátí `True`, nebo zavolejte `client.wait_motion_done()`.
- **Rychlostní řízení (`move_velocity`):** Nastaví cílové úhlové rychlosti vybraných kloubů v **radiánech za sekundu**.
- **Čtení stavu a zastavení:** `client.get_joint_state(joints=None)` vrátí aktuální polohy kloubů v radiánech (bez argumentu pro všechny pohyblivé klouby); `client.stop_robot(joints=None)` zastaví běžící pohyby.
- **Kartézské řízení (`move_cartesian` a `Pose`):** Získejte aktuální polohu koncového efektoru voláním `pose = client.end_effector.get_position()`, upravte souřadnice `pose.pos` (`[x, y, z]` v metrech) nebo orientaci `pose.ori` (kvaternion `[x, y, z, w]`) a zavolejte `client.move_cartesian(pose, wait=True, velocity=1, check_collision=True, timeout=None)`. Inverzní kinematika vypočítá cílové úhly kloubů automaticky.

```
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)
```

- **Jakobián (`compute_jacobian`):** Výpočet geometrického jakobiánu v bázovém souřadném systému a seznamu aktivních kloubů pro vybraný kinematický řetězec (`"left_arm"`, `"right_arm"`, `"left_leg"`, `"right_leg"`, `"torso"`, `"head"`):

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

2.2.4 Oční kamery, dotyková kůže a pracovní složka

- **Oční kamery (`l_eye`, `r_eye`):** Zapněte v horní liště Normálního režimu tlačítky **Levé oko (`l_eye`) / Pravé oko (`r_eye`)** nebo v `.yaml` konfiguraci (`eyes.l_eye: True`, `eyes.r_eye: True`). Ve skriptu získáte obrazy voláním `client.get_camera_images(eyes)` (RGB) a `client.get_camera_depth_images(eyes)` (hloubka). Tlačítky **Uložit RGB / Uložit hloubku** uložíte snímky do `~/Documents/pyCub/images/`.
- **Dotyková kůže:** Nastavte v konfiguraci `skin.use: True` (viz `skin_test.yaml`), krokujte simulaci pomocí `client.update_simulation()` a čtěte `client.skin_activations` a `client.activated_skin_points`.
- **Příklady a kurzovní cvičení:** Nabídka **Příklady & Cvičení** otevře ukázkové skripty nebo nakopíruje cvičení `1-5` do `~/Documents/pyCub/exercises/`. Pokud je otevřený soubor cvičení, tlačítkem **Spustit tester cvičení** spustíte automatické vyhodnocení a zobrazíte HTML protokol.
- **Balíčky a vlastní Python:** Přes **Balíčky / Python** v horní liště můžete doinstalovat knihovny z PyPI do `~/Documents/pyCub/extra_packages/`, připojit `site-packages` existujícího virtuálního prostředí nebo vybrat externí interpret Pythonu pro skripty Normálního režimu.

2.3 Normální / samostatný Python (mimo Studio)

Skripty pro pyCub lze spouštět i přímo z terminálu, PyCharmu nebo VS Code v jakémkoli prostředí s nainstalovanou knihovnou `icub_pybullet` (`python3 -m pip install icub_pybullet` nebo `python3 -m pip install ./icub_pybullet`), bez spouštění GUI Studia.

2.3.1 Výběr vizualizace v `.yaml` konfiguraci (`gui`)

Při běhu mimo Studio se `pyCub` řídí sekcí `gui` v konfiguraci `.yaml`:

```
gui:
  standard: True   # Samostatné Open3D 3D okno
  web: False       # Náhradní webový prohlížeč (pomalejší fallback)
```

- **Samostatné okno Open3D (`standard: True, web: False`):** Výchozí lokální 3D okno.
- **Webová vizualizace (`standard: False, web: True`):** Zobrazí scénu v prohlížeči — hodí se při vzdáleném připojení nebo při potížích s OpenGL ovladači.
- **Bez grafického okna / headless (`standard: False, web: False`):** Nevytváří žádné 3D okno. Nejrychlejší varianta pro automatické testy, zpětnovazební učení (RL), dávkové vyhodnocování a CI.

(Uvnitř Normálního režimu Studia aplikace tyto volby automaticky přesměruje do svého vestavěného OpenGL okna.)

2.3.2 Samostatné spuštění skriptu s `kid_api`

Skripty napsané pomocí `icub_pybullet.kid_api` lze spouštět i jako běžné Python soubory mimo Studio:

```
python muj_detsky_skript.py
```

Při prvním volání příkazu funkce `get_robot()` automaticky vytvoří klienta `pyCub(config="with_ball.yaml")` a postará se o krokování simulace.

2.3.3 Vyhledávání konfigurací a spouštění testerů z příkazové řádky

Předáte-li relativní název konfigurace (např. `pyCub(config="with_ball.yaml")`), pyCub hledá v tomto pořadí:

1. Přímá / absolutní cesta nebo cesta vůči aktuálnímu pracovnímu adresáři.
2. Cesta vůči adresáři volajícího skriptu (a podsložce `configs/` vedle něj).
3. Uživatelská složka (`~/Documents/pyCub/worlds/`, `~/Documents/pyCub/configs/`, `~/Documents/pyCub/`).
4. Vestavěné konfigurace `icub_pybullet/configs/` (`default.yaml`, `with_ball.yaml`, `skin_test.yaml`).

Vlastní `.yaml` světy mohou dědit výchozí nastavení přes `parent: with_ball.yaml` a přepsat jen vybrané položky (`urdfs`, `eyes`, `skin`, `end_effector`).

Tester cvičení lze spustit také přímo z příkazové řádky mimo Studio:

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

2.4 Zobrazení nápovědy a definice funkce v editoru

Editor kódu v režimech **Dětské programování** i **Normální režim** využívá statickou analýzu Jedi:

- **Nápověda po najetí myší:** Najeďte kurzorem myši na název funkce nebo třídy — zobrazí se její signatura a docstring bez přesunu textového kurzoru. Nápověda převádí pole `:param:` i koncové komentáře u parametrů na přehledné řádky.
- **Aktivní parametr a signatura:** Ponechte textový kurzor uvnitř závorek volané funkce (nebo stiskněte **Ctrl+Shift+Space**) pro zobrazení signatury a výchozích hodnot parametrů.
- **Přejít na definici (F12):** Umístěte kurzor na název funkce či třídy a stiskněte **F12** — otevře se definice včetně dokumentace, a to jak pro neuložené funkce v aktuálním souboru, tak pro zdrojový kód `icub_pybullet`.

2.5 Přehled vybraných metod a senzorů

2.5.1 Rychlé srovnání: Kid API vs. plné pyCub API

Úloha	Kid API (`icub_pybullet.kid_api`)	Plné pyCub API (`client = pyCub(...)`)
Cílový úhel kloubu	`move_joint("neck_pitch", -15)` *(stupně)*	`client.move_position("neck_pitch", -0.26)` *(radiány)*
Pohyb více kloubů	Postupná volání nebo cyklus	`client.move_position(["torso_pitch", "torso_yaw"], [0.3, -0.5])`
Relativní posun kloubu	`jog_joint("r_elbow", 20)` *(stupně)*	Přečíst `get_joint_state(...)` + `move_position(...)`
Rychlost kloubu	—	`client.move_velocity(joints, velocities)` *(rad/s)*
Kartézská IK poloha	—	`client.move_cartesian(pose)` *(Pose(pos, ori))*
Čtení úhlu kloubu	`get_joint("neck_pitch")` *(stupně)*	`client.get_joint_state("neck_pitch")` *(radiány)*
Obrazy z kamer	—	`client.get_camera_images(eyes)` / `client.get_camera_depth_images(eyes)`
Senzory míčku	`ball_distance()`, `ball_touched()`, `ball_hits()`	Přes `client.other_objects` + `getClosestPoints` / `getContactPoints`
Dotyková kůže	`skin_touched()`	`client.skin_activations`, `client.activated_skin_points`
Krokování simulace	Řešeno uvnitř každého příkazu	`client.update_simulation()` / `client.wait_motion_done()`

2.5.2 Hlavní klouby a kinematické řetězce iCuba

- **Trup:** `torso_pitch`, `torso_roll`, `torso_yaw`
- **Hlava a oči:** `neck_pitch`, `neck_roll`, `neck_yaw`, `eyes_tilt`, `l_eye_pan_joint`, `r_eye_pan_joint`
- **Levá / pravá paže (`l_` / `r_`):** `*_shoulder_pitch`, `*_shoulder_roll`, `*_shoulder_yaw`, `*_elbow`, `*_wrist_prosup`, `*_wrist_pitch`, `*_wrist_yaw`, plus klouby prstů `*_hand_*`
- **Levá / pravá noha (`l_` / `r_`):** `*_hip_pitch`, `*_hip_roll`, `*_hip_yaw`, `*_knee`, `*_ankle_pitch`, `*_ankle_roll`

- **Kinematické řetězce (`CHAINS`):** `"left_arm"`, `"right_arm"`, `"left_leg"`, `"right_leg"`, `"torso"`, `"head"`

Další podrobnosti, výchozí parametry a hotové řídicí programy najdete v sekcích *Ukázkové programy* a *API reference*.

KAPITOLA 3

Ukázkové programy

Tyto příklady můžete otevřít přímo v Normálním režimu přes nabídku **Příklady & Cvičení**, spustit je a libovolně upravovat. Každý skript si vytváří vlastního klienta a vybírá svět.

3.1 Kloubové řízení

Nastavení cílových úhlů jednotlivých kloubů tak, aby robot rukou posunul míček.

```
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:
```

(continues on next page)

(pokračujte na předchozí stránce)

```
    client.wait_motion_done()
    """

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

3.2 Kartézské řízení

Posun koncového efektoru (ruky) v prostoru vzhledem k jeho aktuální pozici.

```
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 Dotyková kůže

Svět *skin_test.yaml* ukazuje, které senzory na umělé kůži se aktivují při dopadu míčků na robota.

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

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

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

3.4 Minimální smyčka simulace

`Stáhněte si startovací skript` a otevřete jej v Normálním režimu. Vlastní příkazy pro robota přidejte za vytvoření klienta.

```
from icub_pybullet.pycub import pyCub

if __name__ == "__main__":
    # Načtení robota se zvolenou konfigurací světa
    client = pyCub(config="with_ball.yaml")

    # Smyčka simulace běží, dokud není okno/proces ukončen
    while client.is_alive():
        client.update_simulation()
```

KAPITOLA 4

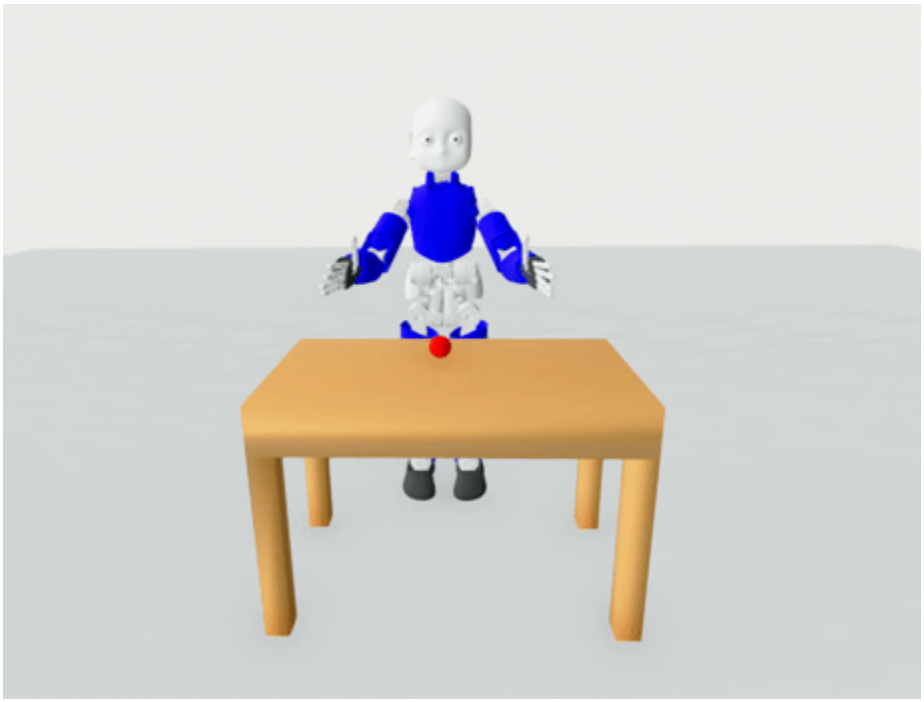
Kurzovní cvičení

Vyberte cvičení z nabídky **Příklady & Cvičení** v Normálním režimu, doplňte řešení do vyznačené funkce či třídy a spusťte tlačítkem **Spustit tester cvičení**. Jednotlivé návody níže popisují požadované rozhraní a bodování.

4.1 Odtlačení míčku (Push the Ball)

Cílem úlohy je udeřit do míčku a odtlačit jej co nejdále od **jakékoli části** stolu. Míček se vždy objeví na stejném místě. Robota lze ovládat pomocí pozičního nebo kartézského řízení. Trajektorie musí být bezkolizní (vůči stolu) a maximální povolená rychlost je 10 rad/s.

Výsledná vzdálenost se vyhodnocuje po 2 sekundách.



4.1.1 Úkol

Implementujte funkci `push_the_ball` v souboru exercise_1.py, která odtlačí míček co nejdále od stolu.

- Funkce přijímá jeden argument:
  - `client` – instance třídy `pyCub` ovládající simulaci
  - nevytvářejte ve svém kódu novou instanci klienta!

4.1.2 Bodování

- Body za vzdálenost se počítají jako: `min(3, distance * 2)`
  - tj. maximálně 3 body získáte při odtlačení míčku alespoň 1,5 m od stolu.

4.1.3 Požadavky

Požadavky nutné pro správnou funkci testeru exercise_1_tester.py:

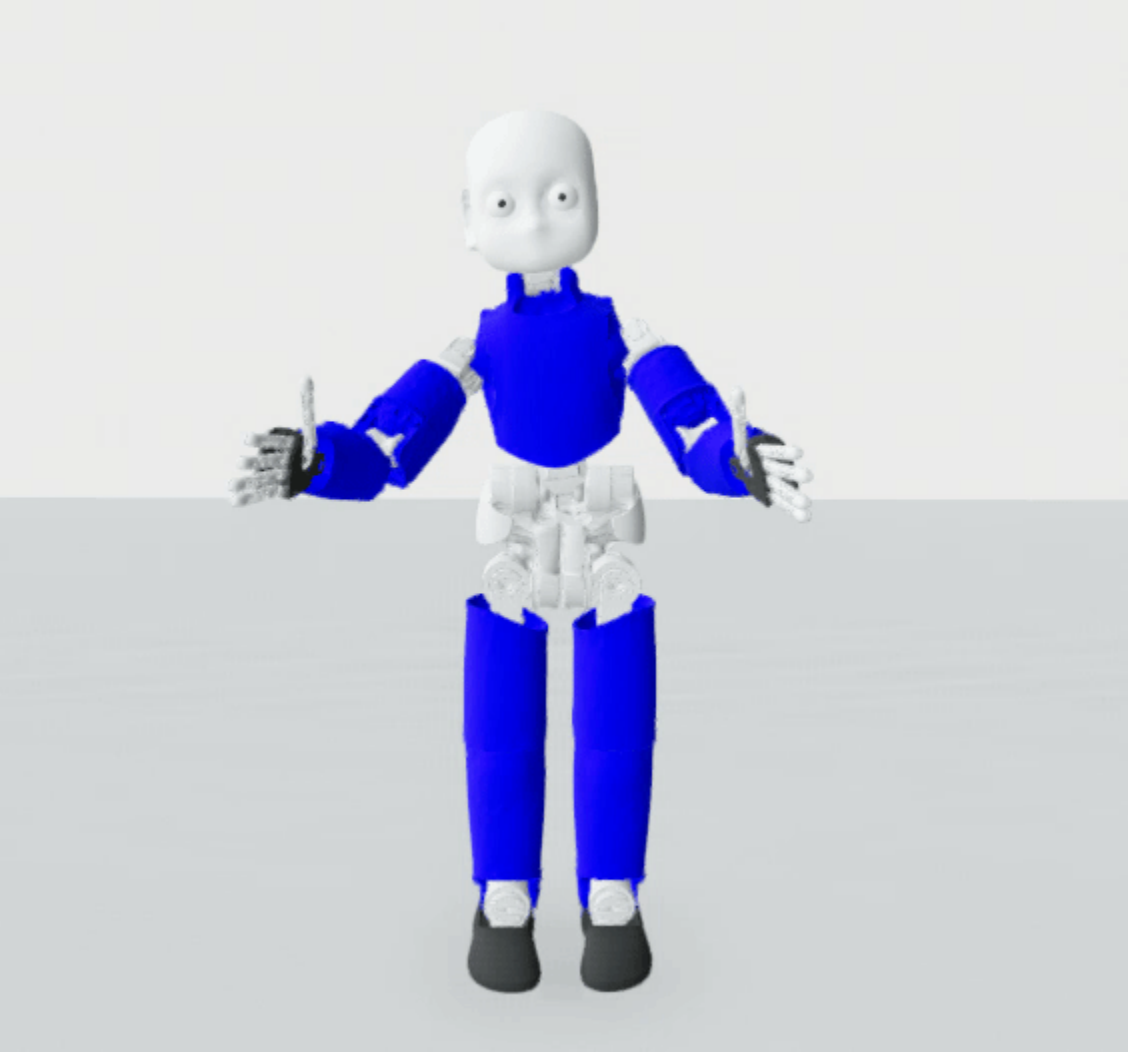
- nevytvářejte nového klienta, použijte předaný argument `client`
- nepřejmenovávejte funkci ani soubor
- **nevkládejte** umělé prodlevy, např. `sleep()` nebo volání `update_simulation()` po dokončení pohybu

Požadavky na správné splnění úlohy:

- Trajektorie nesmí kolidovat se stolem (vlastní kolize robota jsou povoleny)
- Maximální povolená rychlost je 10 rad/s
- **Nevypínejte** gravitaci

4.2 Plynulé pohyby (Smooth movements)

Úkolem je provádět plynulé pohyby rukou robota. Robot má být schopen pohybovat se po přímce nebo po kružnici s plynulou trajektorií — pohyb tedy musí být spojitý a bez trhání.



4.2.1 Úkol

Implementujte funkci `move()` v souboru exercise_2.py, která bude pohybovat rukou robota po přímce nebo kružnici.

- Funkce přijímá čtyři argumenty:
  - `client` – instance třídy `pyCub` ovládající simulaci
  - `action` – řetězec `"line"` (přímka) nebo `"circle"` (kružnice)
  - `axis` – seznam celých čísel. Pro `"circle"` má vždy délku 1. Pro `"line"` může mít délku 1–3. Čísla udávají osy, podél kterých se má robot pohybovat (např. `[0, 1]` znamená pohyb v osách x a y).
  - `r` – seznam desetinných čísel (stejné délky jako `axis`), v metrech. Pro `"circle"` jde o poloměr kružnice, pro `"line"` o délku úsečky v dané ose.
- Ukázkové vstupy:

  - `action="circle", axis=[0], r=[0.01]` – koncový efektor opíše kružnici kolem osy x s poloměrem 0,01 m
  - `action="line", axis=[1], r=[-0.05]` – koncový efektor se posune po přímce v ose y o -0,05 m
  - `action="line", axis=[0, 1], r=[0.05, -0.05]` – koncový efektor se posune současně v ose x o 0,05 m a v ose y o -0,05 m (výsledkem je jedna šikmá přímka)
- Funkce **musí** vrátit dvě hodnoty: počáteční a koncovou pózu trajektorie
  - obě získejte pomocí `client.end_effector.get_position()`

4.2.2 Bodování

- Získáte 10 náhodných testovacích případů (po 1 bodu).
- Kruhové i přímkové pohyby se vyhodnocují podle tří dílčích kritérií:
  - **kružnice**:
    - směrodatná odchylka vzdálenosti bodů kružnice od středu musí být < `r * 0.125`
    - rozdíl očekávaného a skutečného poloměru musí být < `r * 0.1`
    - průměrná vzdálenost bodů od očekávané roviny musí být < `0.1`
  - **přímka**:
    - rozdíl skutečné a očekávané délky musí být < `0.0075`
    - úhel mezi očekávanou a skutečnou přímkou musí být < `0.1`
    - průměrná vzdálenost všech bodů od očekávané přímky musí být < `0.01`
- **Plynulost pohybu není automatickým testerem kontrolována!**
- **Kód by měl být co nejobecnější**, aby fungoval pro jakýkoli povolený vstup (nepište např. dlouhé `if/else` větvení pro každou kombinaci os).

4.2.3 Požadavky

Pro správnou funkci testeru exercise_2_tester.py:

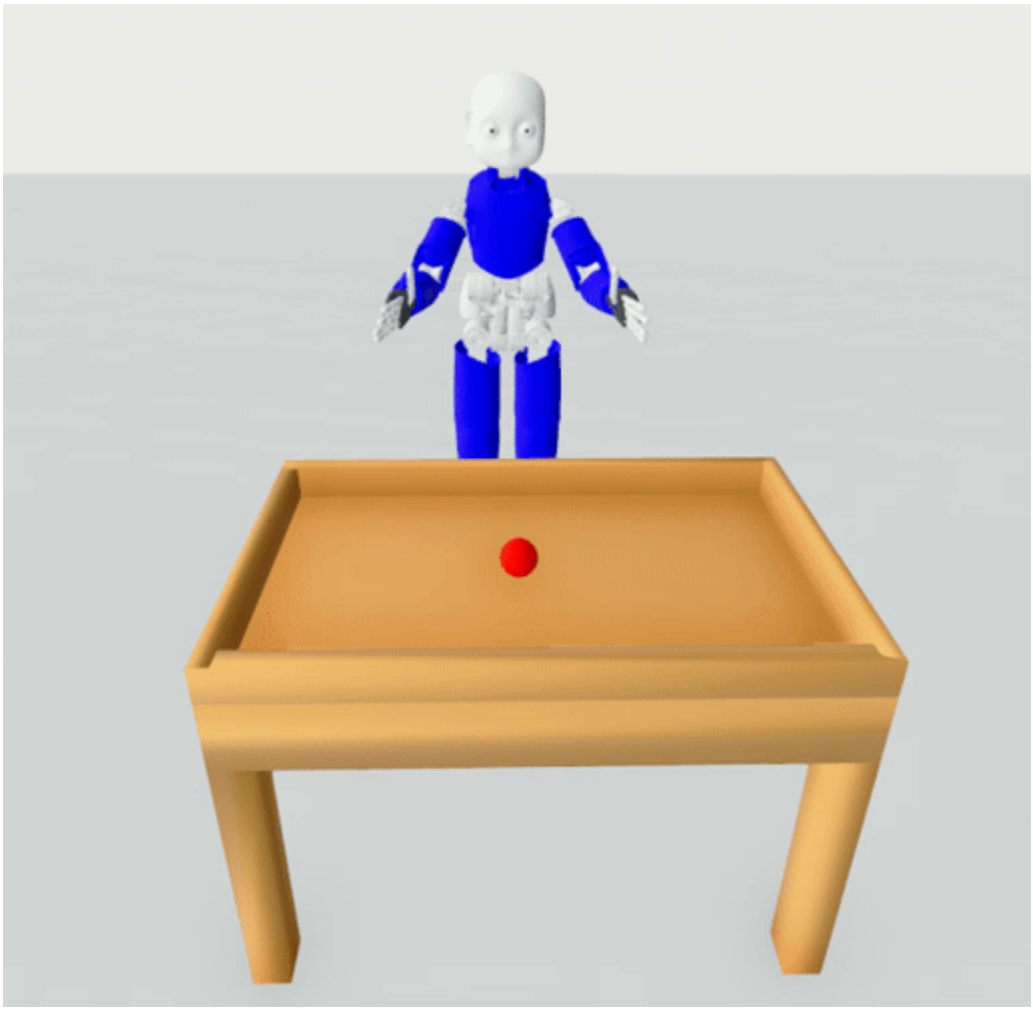
- nevytvářejte nového klienta, použijte předaný argument `client`
- nepřejmenovávejte funkci ani soubor
- **vraťte dvě hodnoty:** počáteční a koncovou pózu trajektorie typu `utils.Pose`

Pro správné splnění úlohy:

- **Nevypínejte** gravitaci
- Udělejte pohyb co nejplynulejší

4.3 Řízení pohledu (Gaze control)

Cílem této úlohy je implementovat řízení pohledu (gaze controller) pro robota iCub, které dokáže sledovat míček pohybující se po stole. Úloha je zjednodušena na 2D případ (míček se pohybuje v osách x a y). K dispozici máte vektor z hlavy k míčku (kam se má robot dívat) a vektor směru, kam se robot dívá právě teď.



4.3.1 Úkol

Implementujte funkci `gaze()` v souboru exercise_3.py, která bude řídit pohled robota při sledování míčku.

- Funkce přijímá tři argumenty:
  - `client` – instance třídy `pyCub` ovládající simulaci
  - `head_direction` – normalizovaný vektor představující aktuální směr pohledu robota
  - `head_ball_direction` – normalizovaný vektor z hlavy robota směrem k míčku (cílový směr pohledu)
- Funkce má ovládat krční klouby robota tak, aby sledoval míček:
  - pohyb **musí** být neblokující, tj. s parametrem `wait=False`
  - v této funkci **nevolejte** `update_simulation()`

4.3.2 Bodování

- Míček se pohybuje po dobu 10 s (výchozí nastavení) a v každém kroku se měří chyba ve stupních.
- Maximální počet bodů je 10 (10 běhů); body můžete **ztratit** za jednotlivé běhy podle:
  - průměrná absolutní chyba:
    - $< 0.5^\circ$ – ztráta 0 % bodů

        * `0.5°` – `1°` – ztráta 50 % bodů
        * `1°` – `5°` – ztráta 75 % bodů
        * `> 5°` – ztráta 100 % bodů
    - maximální chyba:
        * `< 2°` – ztráta 0 % bodů
        * `2°` – `5°` – ztráta 25 % bodů
        * `5°` – `10°` – ztráta 50 % bodů
        * `> 10°` – ztráta 100 % bodů
- Ztráty z průměrné a maximální chyby se sčítají (např. 75 % za průměr a 25 % za maximum znamená 100 % ztrátu a 0 bodů).

4.3.3 Požadavky

Pro správnou funkci testeru exercise_3_tester.py:

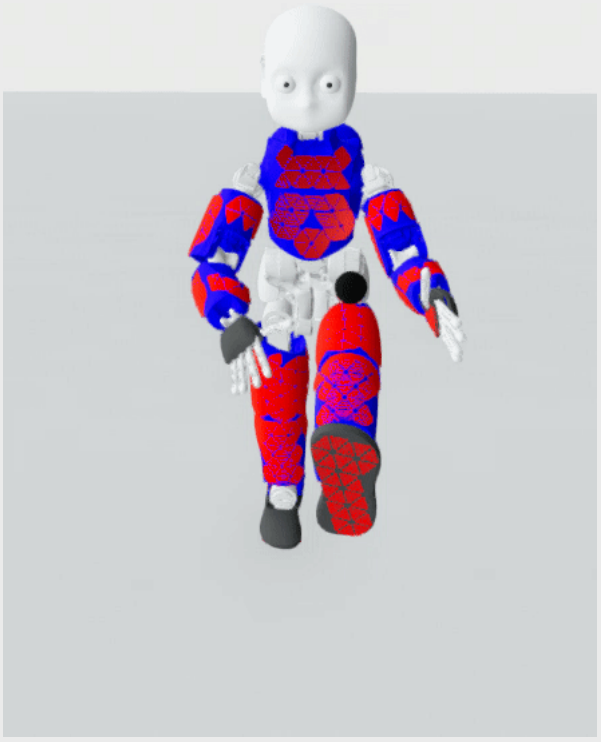
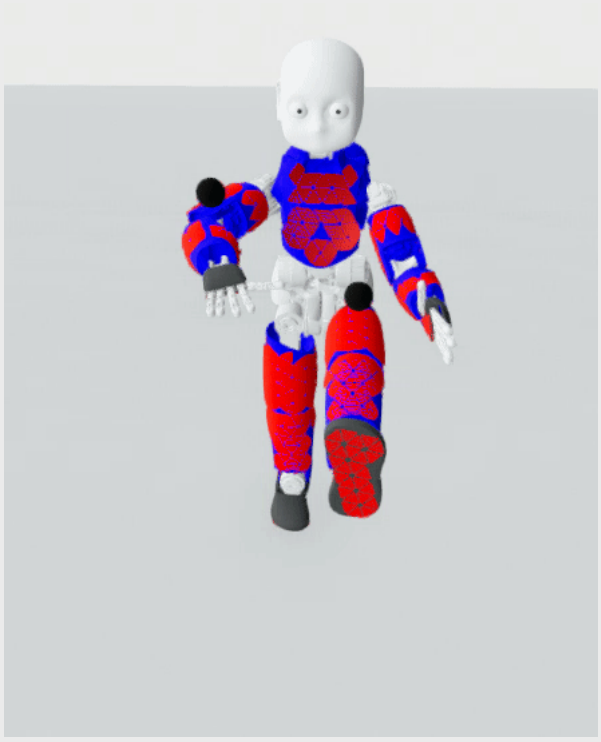
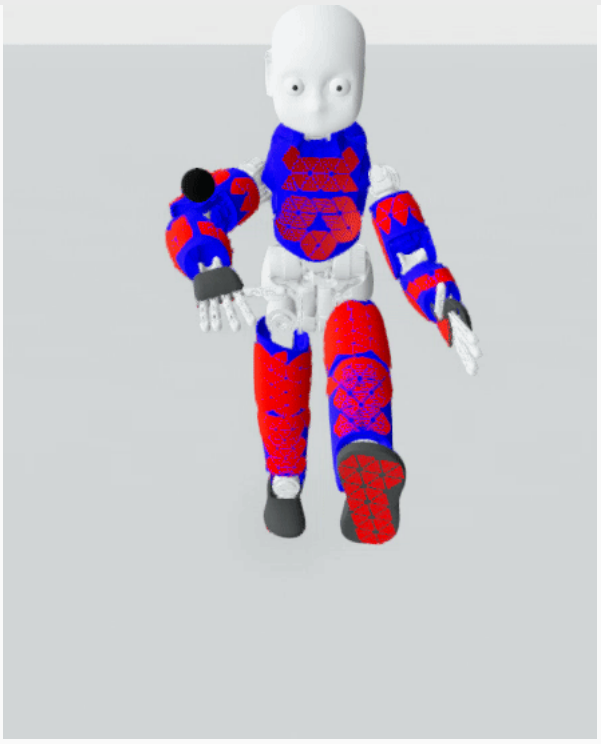
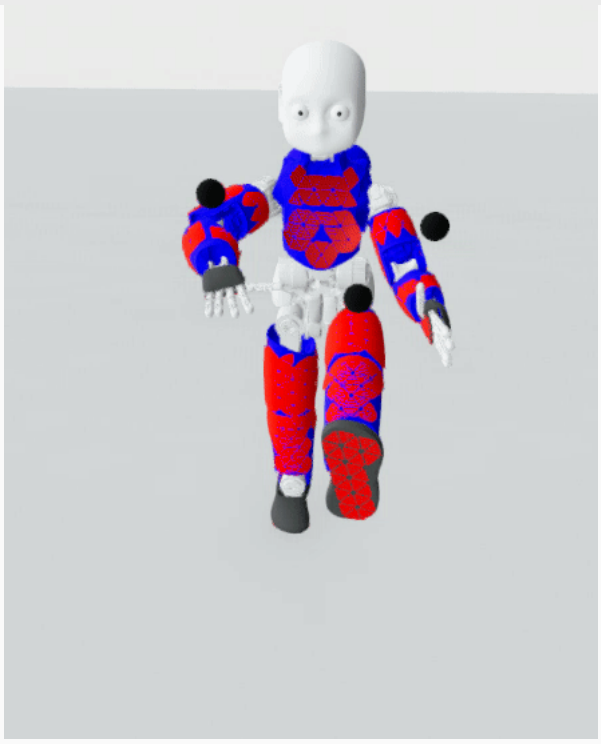
- nevytvářejte nového klienta, použijte předaný argument `client`
- nepřejmenovávejte funkci ani soubor
- **používejte neblokující pohyby**, tj. parametr `wait=False` nebo rychlostní řízení
- v žádném místě kódu **nevolejte** `update_simulation()`

Pro správné splnění úlohy:

- **Nevypínejte** gravitaci

4.4 Řízení pomocí jakobiánu (Resolved-Rate Motion Control)

Cílem úlohy je implementovat regulátor RRMC (Resolved-Rate Motion Control), který uhýbá končetinami před kolizemi na základě zpětné vazby z umělé kůže robota. Na robota budou padat míčky a příslušnou část těla je třeba odsunout od kontaktu — tj. pohybovat jí proti normále dotyku.

Jedna končetina	Více končetin
	
	

4.4.1 Úkol

Implementujte metodu `process()` v souboru exercise_4.py, která bude ovládat robota pomocí RRMC. Funkce je tentokrát metodou třídy, abyste si mohli uchovávat vlastní stavové proměnné.

- Třída přijímá jeden argument:
  - `client` – instance třídy `pyCub` ovládající simulaci
- Funkce má klouby ovládat pomocí RRMC, tedy rychlostním řízením (`move_velocity`)
- V kódu **nesmíte** volat `update_simulation()`

Postupně dostanete čtyři scénáře s různými částmi těla zasaženými různým počtem míčků. Uhýbejte každou částí kůže, která je v kolizi s libovolným míčkem, ale pro každou část kůže použijte vždy jeden kontakt — najděte největší shluk (klastr) aktivovaných taxelů na dané části kůže.

Testovací skript exercise_4_tester.py spustí všechny čtyři testy postupně od nejjednoduššího. Po každém testu zavřete okno vizualizace (případně ukončete běh ve Studiu).

4.4.2 Bodování

Pro tuto úlohu není automatické číselné hodnocení. Úloha je splněna tehdy, když robot spolehlivě uhýbá před padajícími míčky. Ukázkové výsledky vidíte v animacích výše; nejde však o jediné správné řešení, chování závisí na zvolených parametrech RRMC.

4.4.3 Požadavky

Pro správnou funkci testeru exercise_4_tester.py:

- nevytvářejte nového klienta, použijte argument `client` předaný třídě RRMC
- nepřejmenovávejte funkci, soubor ani třídu
- v kódu **nevolejte** `update_simulation()`

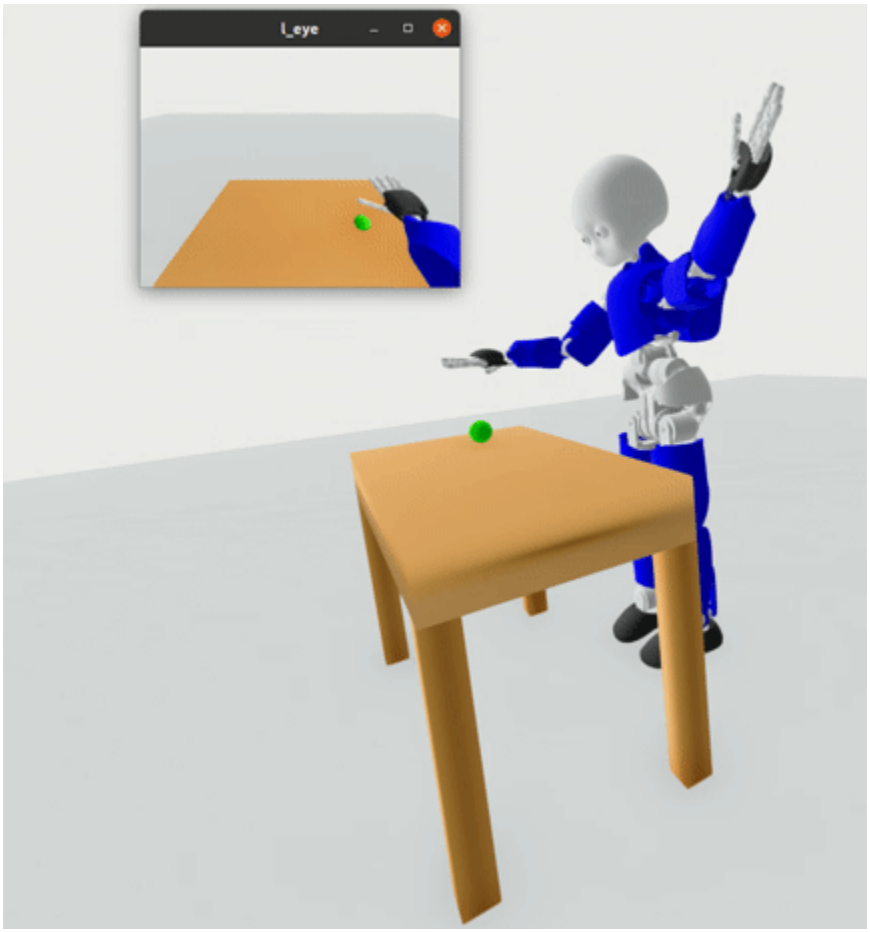
Pro správné splnění úlohy:

- **Nevypínejte** gravitaci

4.5 Uchopení míčku (Grasp It!)

Cílem úlohy je uchopit míček ze stolu. Úloha je rozdělena na dvě části:

- najít míček na stole v obrazové rovině kamery (míček je v zorném poli robota a je zelený)
- uchopit míček: pomocí polohy míčku v obrazu a hloubkové mapy spočítat jeho 3D polohu a uchopit jej rukou



4.5.1 Úkol

Implementujte metody `find_the_ball()` a `grasp()` v souboru exercise_5.py. Obě jsou metodami třídy `Grasper`, která již obsahuje několik užitečných pomocných metod. Po uchopení zvedněte míček tak, aby byl alespoň 5 cm nad stolem.

Třída přijímá dva argumenty:

- `client` – instance třídy `pyCub` ovládající simulaci
- `eye` – název oka s kamerou (`'l_eye'` nebo `'r_eye'`)

Předepsané rozhraní obou metod:

- `find_the_ball()` – bez argumentů, vrací `(u, v)` (střed míčku v obrazové rovině)
- `grasp(center)` – přijímá jeden argument `center` (2D střed z `find_the_ball()`) a při úspěšném uchopení vrací `0`

Do třídy můžete libovolně přidávat vlastní metody a atributy, tester však volá pouze `find_the_ball()` a `grasp()`. Třída `Grasper` obsahuje pomocné metody pro získání RGB a hloubkového obrazu, zpětnou projekci 2D bodu do 3D i zavření prstů.

Míček je vždy zelený `(0, 255, 0)` a má poloměr 2,5 cm.

4.5.2 Bodování

- Pokud je míček po návratu z `grasp()` alespoň 5 cm nad stolem a v ruce robota, testem projdete.

4.5.3 Požadavky

Pro správnou funkci testeru exercise_5_tester.py:

- nevytvářejte nového klienta, použijte předaný argument `client`
- nepřejmenovávejte `find_the_ball()`, `grasp()` ani soubor
- po návratu z `grasp()` musí být míček alespoň 5 cm nad stolem a držen v ruce robota

Pro správné splnění úlohy:

- **Nevypínejte** gravitaci

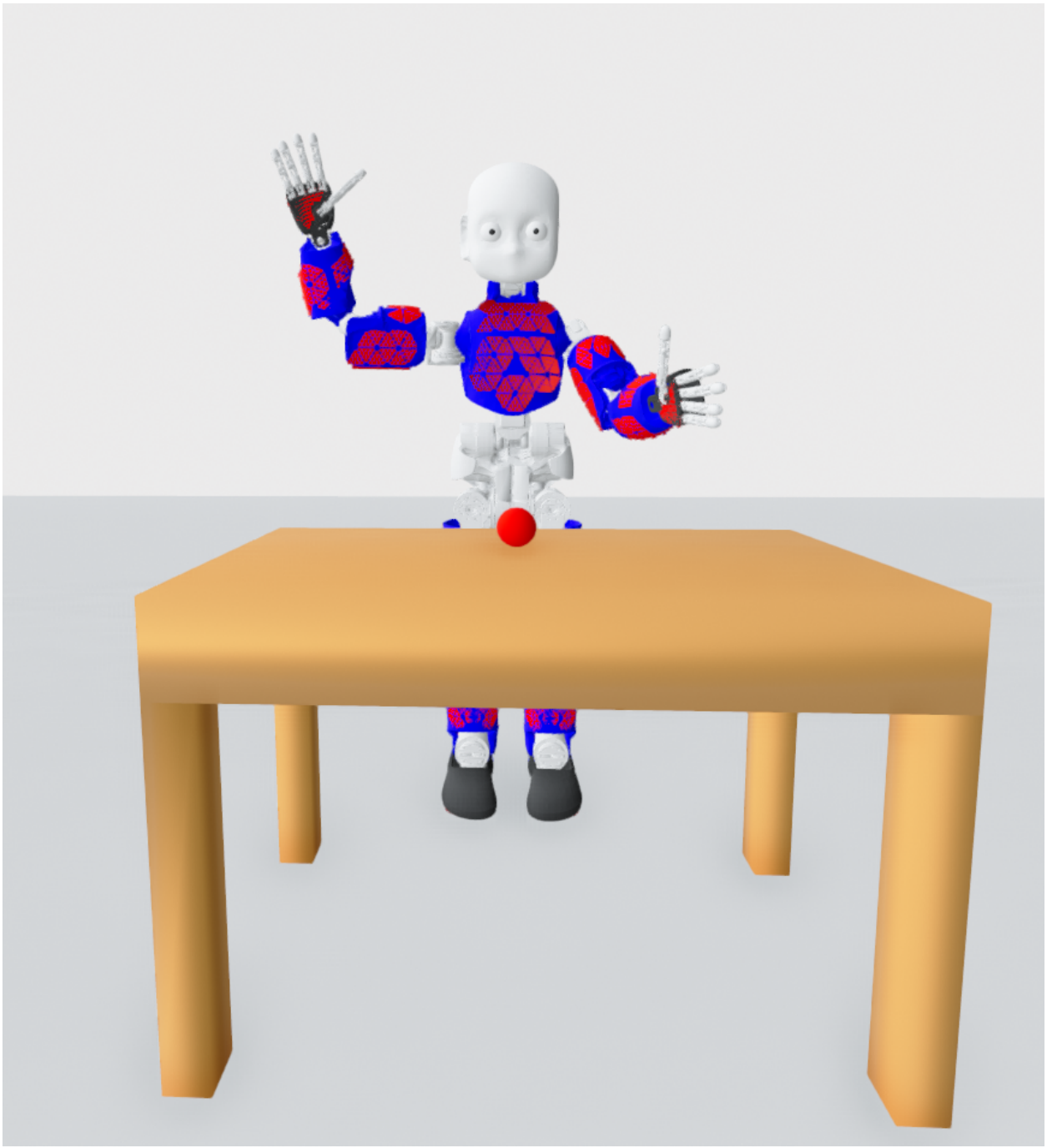
KAPITOLA 5

Dokumentace projektu (README)

Kompletní úvodní dokumentace repozitáře (`README.md`, v angličtině):

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.

KAPITOLA 6

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

[1] Created with sphinx-autoapi

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.

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.

continues on next page

Tabulka 4 – pokračujte na předchozí stránce

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.
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
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*).

continues on next page

Tabulka 8 – pokračujte na předchozí stránce

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).
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.

continues on next page

Tabulka 12 – pokračujte na předchozí stránce

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, 37
icub_pybullet.kid_api, 38
icub_pybullet.pycub, 44
icub_pybullet.runtime_paths, 49
icub_pybullet.utils, 51
icub_pybullet.visualizer, 54

p

push_the_ball_cartesian, 56
push_the_ball_pure_joints, 57

s

skin_test, 37

Rejstřík

B

ball_distance() (*metoda icub_pybullet.kid_api.KidRobot*), 40
ball_distance() (*v modulu icub_pybullet.kid_api*), 44
ball_hits() (*metoda icub_pybullet.kid_api.KidRobot*), 41
ball_hits() (*v modulu icub_pybullet.kid_api*), 44
ball_touched() (*metoda icub_pybullet.kid_api.KidRobot*), 41
ball_touched() (*v modulu icub_pybullet.kid_api*), 44
bbox_overlap() (*statická metoda icub_pybullet.pycub.pyCub*), 47
bend_back() (*metoda icub_pybullet.kid_api.KidRobot*), 40
bend_back() (*v modulu icub_pybullet.kid_api*), 43
bend_elbow() (*metoda icub_pybullet.kid_api.KidRobot*), 40
bend_elbow() (*v modulu icub_pybullet.kid_api*), 42
bend_forward() (*metoda icub_pybullet.kid_api.KidRobot*), 40
bend_forward() (*v modulu icub_pybullet.kid_api*), 43
bend_sideways() (*metoda icub_pybullet.kid_api.KidRobot*), 40
bend_sideways() (*v modulu icub_pybullet.kid_api*), 44
blue (*atribut icub_pybullet.visualizer.Visualizer*), 54
bold_red (*atribut icub_pybullet.utils.CustomFormatter*), 53
both_arms_down() (*metoda icub_pybullet.kid_api.KidRobot*), 39
both_arms_down() (*v modulu icub_pybullet.kid_api*), 41
both_arms_up() (*metoda icub_pybullet.kid_api.KidRobot*), 39
both_arms_up() (*v modulu icub_pybullet.kid_api*), 41

C

center (*atribut icub_pybullet.visualizer.Visualizer.EyeWindow*), 56
CHAINS (*v modulu icub_pybullet.utils*), 54
check_stop() (*metoda icub_pybullet.kid_api.KidRobot*), 39
client (*atribut icub_pybullet.kid_api.KidRobot*), 39
client (*atribut icub_pybullet.pycub.EndEffector*), 49
client (*atribut icub_pybullet.visualizer.Visualizer*), 54
collision_during_motion (*atribut icub_pybullet.pycub.pyCub*), 46
colors (*atribut icub_pybullet.visualizer.Visualizer*), 54
compute_jacobian() (*metoda icub_pybullet.pycub.pyCub*), 47
compute_skin() (*metoda icub_pybullet.pycub.pyCub*), 47
config (*atribut icub_pybullet.pycub.pyCub*), 46
Config (*třída v icub_pybullet.utils*), 53
config_dir (*atribut icub_pybullet.pycub.pyCub*), 46
config_path (*atribut icub_pybullet.pycub.pyCub*), 46
contactPoints (*atribut icub_pybullet.pycub.pyCub*), 46
create_urdf() (*metoda icub_pybullet.pycub.pyCub*), 49
CustomFormatter (*třída v icub_pybullet.utils*), 53

D

default_center (*atribut icub_pybullet.visualizer.Visualizer*), 55
dereference() (*metoda icub_pybullet.utils.URDF*), 52
dialog_opened (*atribut icub_pybullet.visualizer.Visualizer.MenuCallback*), 55
dir (*atribut icub_pybullet.visualizer.Visualizer.EyeWindow*), 56
dynamicsInfo (*atribut icub_pybullet.pycub.pyCub*), 46

E

end_effector (*atribut icub_pybullet.pycub.pyCub*), 46
EndEffector (*třída v icub_pybullet.pycub*), 49
ensure_user_workspace() (*v modulu icub_pybullet.runtime_paths*), 50
expand_package_uris() (*metoda icub_pybullet.utils.URDF*), 52

eye (*atribut icub_pybullet.visualizer.Visualizer.EyeWindow*), 55
eye_windows (*atribut icub_pybullet.visualizer.Visualizer*), 55

F

file_dir (*atribut icub_pybullet.pycub.pyCub*), 46
file_dir (*atribut icub_pybullet.visualizer.Visualizer*), 54
file_fingerprint() (*v modulu icub_pybullet.runtime_paths*), 51
file_menu (*atribut icub_pybullet.visualizer.Visualizer*), 54
find_joint_id() (*metoda icub_pybullet.pycub.pyCub*), 48
find_link_id() (*metoda icub_pybullet.pycub.pyCub*), 48
find_processes_by_name() (*metoda icub_pybullet.pycub.pyCub*), 47
find_root_tags() (*metoda icub_pybullet.utils.URDF*), 52
find_xyz_rpy() (*metoda icub_pybullet.visualizer.Visualizer*), 55
fix_urdf() (*metoda icub_pybullet.utils.URDF*), 52
format (*atribut icub_pybullet.utils.CustomFormatter*), 53
FORMATS (*atribut icub_pybullet.utils.CustomFormatter*), 53
free_objects (*atribut icub_pybullet.pycub.pyCub*), 46

G

get_cached_urdf_path() (*v modulu icub_pybullet.runtime_paths*), 51
get_camera_depth_images() (*metoda icub_pybullet.pycub.pyCub*), 47
get_camera_images() (*metoda icub_pybullet.pycub.pyCub*), 47
get_chains() (*statická metoda icub_pybullet.pycub.pyCub*), 47
get_depth_image() (*metoda icub_pybullet.visualizer.Visualizer.EyeWindow*), 56
get_extra_packages_dir() (*v modulu icub_pybullet.runtime_paths*), 50
get_image() (*metoda icub_pybullet.visualizer.Visualizer.EyeWindow*), 56
get_joint() (*metoda icub_pybullet.kid_api.KidRobot*), 39
get_joint() (*v modulu icub_pybullet.kid_api*), 44
get_joint_state() (*metoda icub_pybullet.pycub.pyCub*), 48
get_position() (*metoda icub_pybullet.pycub.EndEffector*), 49
get_robot() (*v modulu icub_pybullet.kid_api*), 41
get_user_cache_dir() (*v modulu icub_pybullet.runtime_paths*), 50
get_user_log_dir() (*v modulu icub_pybullet.runtime_paths*), 50
get_user_workspace_dir() (*v modulu icub_pybullet.runtime_paths*), 50
get_vhacd_cache_path() (*v modulu icub_pybullet.runtime_paths*), 51
get_visualizer_factory() (*v modulu icub_pybullet.pycub*), 45
gravity (*atribut icub_pybullet.pycub.pyCub*), 46
grey (*atribut icub_pybullet.utils.CustomFormatter*), 53
gui (*atribut icub_pybullet.visualizer.Visualizer*), 54

H

highlight_block() (*metoda icub_pybullet.kid_api.KidRobot*), 39

I

icub_pybullet
    module, 37
icub_pybullet.kid_api
    module, 38
icub_pybullet.pycub
    module, 44
icub_pybullet.runtime_paths
    module, 49
icub_pybullet.utils
    module, 51
icub_pybullet.visualizer
    module, 54
id (*atribut icub_pybullet.visualizer.Visualizer.MenuCallback*), 55
IK_config (*atribut icub_pybullet.pycub.pyCub*), 46
img_counter (*atribut icub_pybullet.visualizer.Visualizer.EyeWindow*), 56
init_robot() (*metoda icub_pybullet.pycub.pyCub*), 47
init_urdfs() (*metoda icub_pybullet.pycub.pyCub*), 47
input_completed() (*metoda icub_pybullet.visualizer.Visualizer.MenuCallback*), 55
is_alive (*atribut icub_pybullet.visualizer.Visualizer*), 55
is_alive() (*metoda icub_pybullet.pycub.pyCub*), 47
is_frozen() (*v modulu icub_pybullet.runtime_paths*), 50
is_writable_dir() (*v modulu icub_pybullet.runtime_paths*), 50

J

jog_joint() (*metoda icub_pybullet.kid_api.KidRobot*), 39
jog_joint() (*v modulu icub_pybullet.kid_api*), 44

Joint (*třída v icub_pybullet.pycub*), 45
joint_tolerance (*atribut icub_pybullet.pycub.pyCub*), 46
jointInfo (*atribut icub_pybullet.pycub.pyCub*), 46
joints (*atribut icub_pybullet.utils.URDF*), 52
JOINTS (*v modulu icub_pybullet.utils*), 53
joints_id (*atribut icub_pybullet.pycub.Joint*), 45
JOINTS_IDS (*v modulu icub_pybullet.utils*), 54
jointStates (*atribut icub_pybullet.pycub.pyCub*), 46

K

KidRobot (*třída v icub_pybullet.kid_api*), 39
KidScriptStopped, 39
kill_open3d() (*metoda icub_pybullet.pycub.pyCub*), 47

L

last_depth_image (*atribut icub_pybullet.visualizer.Visualizer.EyeWindow*), 56
last_image (*atribut icub_pybullet.visualizer.Visualizer.EyeWindow*), 56
last_log (*atribut icub_pybullet.pycub.pyCub*), 46
last_step (*atribut icub_pybullet.pycub.pyCub*), 46
left_wave() (*v modulu icub_pybullet.kid_api*), 41
Link (*třída v icub_pybullet.pycub*), 45
link_name (*atribut icub_pybullet.visualizer.Visualizer.EyeWindow*), 55
linkInfo (*atribut icub_pybullet.pycub.pyCub*), 46
links (*atribut icub_pybullet.utils.URDF*), 52
local_ipv4_addresses() (*v modulu icub_pybullet.runtime_paths*), 51
logger (*atribut icub_pybullet.pycub.pyCub*), 46
look_center() (*metoda icub_pybullet.kid_api.KidRobot*), 39
look_center() (*v modulu icub_pybullet.kid_api*), 41
look_down() (*metoda icub_pybullet.kid_api.KidRobot*), 40
look_down() (*v modulu icub_pybullet.kid_api*), 43
look_up() (*metoda icub_pybullet.kid_api.KidRobot*), 40
look_up() (*v modulu icub_pybullet.kid_api*), 43
lower_limit (*atribut icub_pybullet.pycub.Joint*), 45

M

main() (*v modulu push_the_ball_cartesian*), 57
main() (*v modulu push_the_ball_pure_joints*), 57
main() (*v modulu skin_test*), 37
make_references() (*metoda icub_pybullet.utils.URDF*), 52
mat (*atribut icub_pybullet.visualizer.Visualizer*), 54
mattr (*atribut icub_pybullet.visualizer.Visualizer*), 54
max_force (*atribut icub_pybullet.pycub.Joint*), 45
max_velocity (*atribut icub_pybullet.pycub.Joint*), 45
menu (*atribut icub_pybullet.visualizer.Visualizer*), 54
MENU_IDS (*atribut icub_pybullet.visualizer.Visualizer.EyeWindow*), 55
meshes (*atribut icub_pybullet.visualizer.Visualizer*), 54
module
    icub_pybullet, 37
    icub_pybullet.kid_api, 38
    icub_pybullet.pycub, 44
    icub_pybullet.runtime_paths, 49
    icub_pybullet.utils, 51
    icub_pybullet.visualizer, 54
    push_the_ball_cartesian, 56
    push_the_ball_pure_joints, 57
    skin_test, 37
motion_done() (*metoda icub_pybullet.pycub.pyCub*), 48
move_cartesian() (*metoda icub_pybullet.pycub.pyCub*), 48
move_joint() (*metoda icub_pybullet.kid_api.KidRobot*), 39
move_joint() (*v modulu icub_pybullet.kid_api*), 44
move_position() (*metoda icub_pybullet.pycub.pyCub*), 47
move_velocity() (*metoda icub_pybullet.pycub.pyCub*), 48

N

name (*atribut icub_pybullet.pycub.EndEffector*), 49
name (*atribut icub_pybullet.pycub.Joint*), 45
name (*atribut icub_pybullet.pycub.Link*), 46
neighbour_links (*atribut icub_pybullet.pycub.pyCub*), 46
new_urdf (*atribut icub_pybullet.utils.URDF*), 52

O

on_close() (*metoda icub_pybullet.visualizer.Visualizer.EyeWindow*), 56
on_mouse() (*metoda icub_pybullet.visualizer.Visualizer.EyeWindow*), 56
ori (*atribut icub_pybullet.utils.Pose*), 53
orientations (*atribut icub_pybullet.visualizer.Visualizer*), 54
other_objects (*atribut icub_pybullet.pycub.pyCub*), 46

P

package_dir (*atribut icub_pybullet.utils.URDF*), 52
parent (*atribut icub_pybullet.visualizer.Visualizer.EyeWindow*), 55
parent (*atribut icub_pybullet.visualizer.Visualizer.MenuCallback*), 55
parent_name (*atribut icub_pybullet.pycub.pyCub*), 46
path (*atribut icub_pybullet.utils.URDF*), 52
path (*atribut icub_pybullet.visualizer.Visualizer.MenuCallback*), 55
paths (*atribut icub_pybullet.visualizer.Visualizer*), 55

pos (*atribut icub_pybullet.utils.Pose*), 53
Pose (*třída v icub_pybullet.utils*), 53
positions (*atribut icub_pybullet.visualizer.Visualizer*), 54
POSITIONS (*atribut icub_pybullet.visualizer.Visualizer.EyeWindow*), 55
precomputed_link_ids (*atribut icub_pybullet.pycub.pyCub*), 46
prepare_log() (*metoda icub_pybullet.pycub.pyCub*), 47
print_collision_info() (*metoda icub_pybullet.pycub.pyCub*), 49
push_the_ball() (*v modulu push_the_ball_cartesian*), 57
push_the_ball() (*v modulu push_the_ball_pure_joints*), 57
push_the_ball_cartesian module, 56
push_the_ball_pure_joints module, 57
pyCub (*třída v icub_pybullet.pycub*), 46

R

R_urdf (*atribut icub_pybullet.visualizer.Visualizer*), 54
read() (*metoda icub_pybullet.utils.URDF*), 52
read_info() (*metoda icub_pybullet.visualizer.Visualizer*), 55
red (*atribut icub_pybullet.utils.CustomFormatter*), 53
red (*atribut icub_pybullet.visualizer.Visualizer*), 54
render() (*metoda icub_pybullet.visualizer.Visualizer*), 55
reset (*atribut icub_pybullet.utils.CustomFormatter*), 53
reset_pose() (*metoda icub_pybullet.kid_api.KidRobot*), 39
reset_pose() (*v modulu icub_pybullet.kid_api*), 41
reset_view() (*metoda icub_pybullet.visualizer.Visualizer*), 55
resolve_config_path() (*v modulu icub_pybullet.runtime_paths*), 51
resolve_mesh_or_urdf_path() (*v modulu icub_pybullet.runtime_paths*), 51
resolve_robot_urdf_path() (*v modulu icub_pybullet.runtime_paths*), 51
right_wave() (*v modulu icub_pybullet.kid_api*), 41
robot_joint_id (*atribut icub_pybullet.pycub.Joint*), 45
robot_link_id (*atribut icub_pybullet.pycub.Link*), 46
robot_name (*atribut icub_pybullet.utils.URDF*), 52
ROOT_TAGS (*atribut icub_pybullet.utils.URDF*), 52
rotate_arm() (*metoda icub_pybullet.kid_api.KidRobot*), 40
rotate_arm() (*v modulu icub_pybullet.kid_api*), 42
rotate_wrist() (*metoda icub_pybullet.kid_api.KidRobot*), 40
rotate_wrist() (*v modulu icub_pybullet.kid_api*), 42
rtb_robot (*icub_pybullet.pycub.pyCub property*), 47
run_vhacd() (*metoda icub_pybullet.pycub.pyCub*), 49

S

save_depth_image() (*metoda icub_pybullet.visualizer.Visualizer.EyeWindow*), 56
save_image() (*metoda icub_pybullet.visualizer.Visualizer.EyeWindow*), 56
save_image() (*metoda icub_pybullet.visualizer.Visualizer.MenuCallback*), 55
save_images() (*metoda icub_pybullet.visualizer.Visualizer.EyeWindow*), 56
save_images_bool (*atribut icub_pybullet.visualizer.Visualizer.EyeWindow*), 56
scale_bbox() (*statická metoda icub_pybullet.pycub.pyCub*), 47
scene (*atribut icub_pybullet.visualizer.Visualizer*), 54
scene (*atribut icub_pybullet.visualizer.Visualizer.EyeWindow*), 56
set_attribute() (*metoda icub_pybullet.utils.Config*), 53
set_point (*atribut icub_pybullet.pycub.Joint*), 45
set_status_callback() (*v modulu icub_pybullet.pycub*), 45
set_visualizer_factory() (*v modulu icub_pybullet.pycub*), 45
shoulder_down() (*metoda icub_pybullet.kid_api.KidRobot*), 40
shoulder_down() (*v modulu icub_pybullet.kid_api*), 41
shoulder_in() (*metoda icub_pybullet.kid_api.KidRobot*), 40
shoulder_in() (*v modulu icub_pybullet.kid_api*), 42
shoulder_out() (*metoda icub_pybullet.kid_api.KidRobot*), 40
shoulder_out() (*v modulu icub_pybullet.kid_api*), 42
shoulder_up() (*metoda icub_pybullet.kid_api.KidRobot*), 40
shoulder_up() (*v modulu icub_pybullet.kid_api*), 41
show_first() (*metoda icub_pybullet.visualizer.Visualizer*), 55
show_menu (*atribut icub_pybullet.visualizer.Visualizer*), 54
show_mesh() (*metoda icub_pybullet.visualizer.Visualizer*), 55
skin_activated_last_turn (*atribut icub_pybullet.visualizer.Visualizer*), 54
skin_test module, 37

skin_touched() (*metoda icub_pybullet.kid_api.KidRobot*), 41
skin_touched() (*v modulu icub_pybullet.kid_api*), 44
SKIP_IFACE_PREFIXES (*v modulu icub_pybullet.runtime_paths*), 50
stand_straight() (*metoda icub_pybullet.kid_api.KidRobot*), 39
stand_straight() (*v modulu icub_pybullet.kid_api*), 41
start_time (*atribut icub_pybullet.pycub.Joint*), 45
steps_done (*atribut icub_pybullet.pycub.pyCub*), 46
stop() (*metoda icub_pybullet.kid_api.KidRobot*), 40
stop() (*v modulu icub_pybullet.kid_api*), 41
stop_robot() (*metoda icub_pybullet.pycub.pyCub*), 48
straighten_elbow() (*metoda icub_pybullet.kid_api.KidRobot*), 40
straighten_elbow() (*v modulu icub_pybullet.kid_api*), 42

T

terminate_process_tree() (*v modulu icub_pybullet.runtime_paths*), 51
tilt_head() (*metoda icub_pybullet.kid_api.KidRobot*), 40
tilt_head() (*v modulu icub_pybullet.kid_api*), 43
timeout (*atribut icub_pybullet.pycub.Joint*), 45
to_posix() (*v modulu icub_pybullet.runtime_paths*), 50
to_string() (*metoda icub_pybullet.utils.Pose*), 53
toggle_gravity() (*metoda icub_pybullet.pycub.pyCub*), 47
turn_body() (*metoda icub_pybullet.kid_api.KidRobot*), 40
turn_body() (*v modulu icub_pybullet.kid_api*), 43
turn_head() (*metoda icub_pybullet.kid_api.KidRobot*), 40
turn_head() (*v modulu icub_pybullet.kid_api*), 43

U

unproject() (*metoda icub_pybullet.visualizer.Visualizer.EyeWindow*), 56
update_simulation() (*metoda icub_pybullet.pycub.pyCub*), 47
upper_limit (*atribut icub_pybullet.pycub.Joint*), 45
URDF (*třída v icub_pybullet.utils*), 52
urdf_link (*atribut icub_pybullet.pycub.Link*), 46
urdf_path (*atribut icub_pybullet.pycub.pyCub*), 46
urdfs (*atribut icub_pybullet.pycub.pyCub*), 46
user_documents_dir() (*v modulu icub_pybullet.runtime_paths*), 50

V

vis (*atribut icub_pybullet.visualizer.Visualizer*), 54
vis (*atribut icub_pybullet.visualizer.Visualizer.EyeWindow*), 56
Visualizer (*třída v icub_pybullet.visualizer*), 54
Visualizer.EyeWindow (*třída v icub_pybullet.visualizer*), 55
Visualizer.MenuCallback (*třída v icub_pybullet.visualizer*), 55
visualShapeData (*atribut icub_pybullet.pycub.pyCub*), 46

W

wait() (*metoda icub_pybullet.kid_api.KidRobot*), 40
wait() (*v modulu icub_pybullet.kid_api*), 44
wait_for_dialog_completion() (*metoda icub_pybullet.visualizer.Visualizer.MenuCallback*), 55
wait_motion_done() (*metoda icub_pybullet.pycub.pyCub*), 48
wave() (*metoda icub_pybullet.kid_api.KidRobot*), 39
wave() (*v modulu icub_pybullet.kid_api*), 41
window (*atribut icub_pybullet.visualizer.Visualizer*), 54
window (*atribut icub_pybullet.visualizer.Visualizer.EyeWindow*), 56
write_attr() (*metoda icub_pybullet.utils.URDF*), 53
write_text_atomic() (*v modulu icub_pybullet.runtime_paths*), 50
write_urdf() (*metoda icub_pybullet.utils.URDF*), 53

Y

yellow (*atribut icub_pybullet.utils.CustomFormatter*), 53