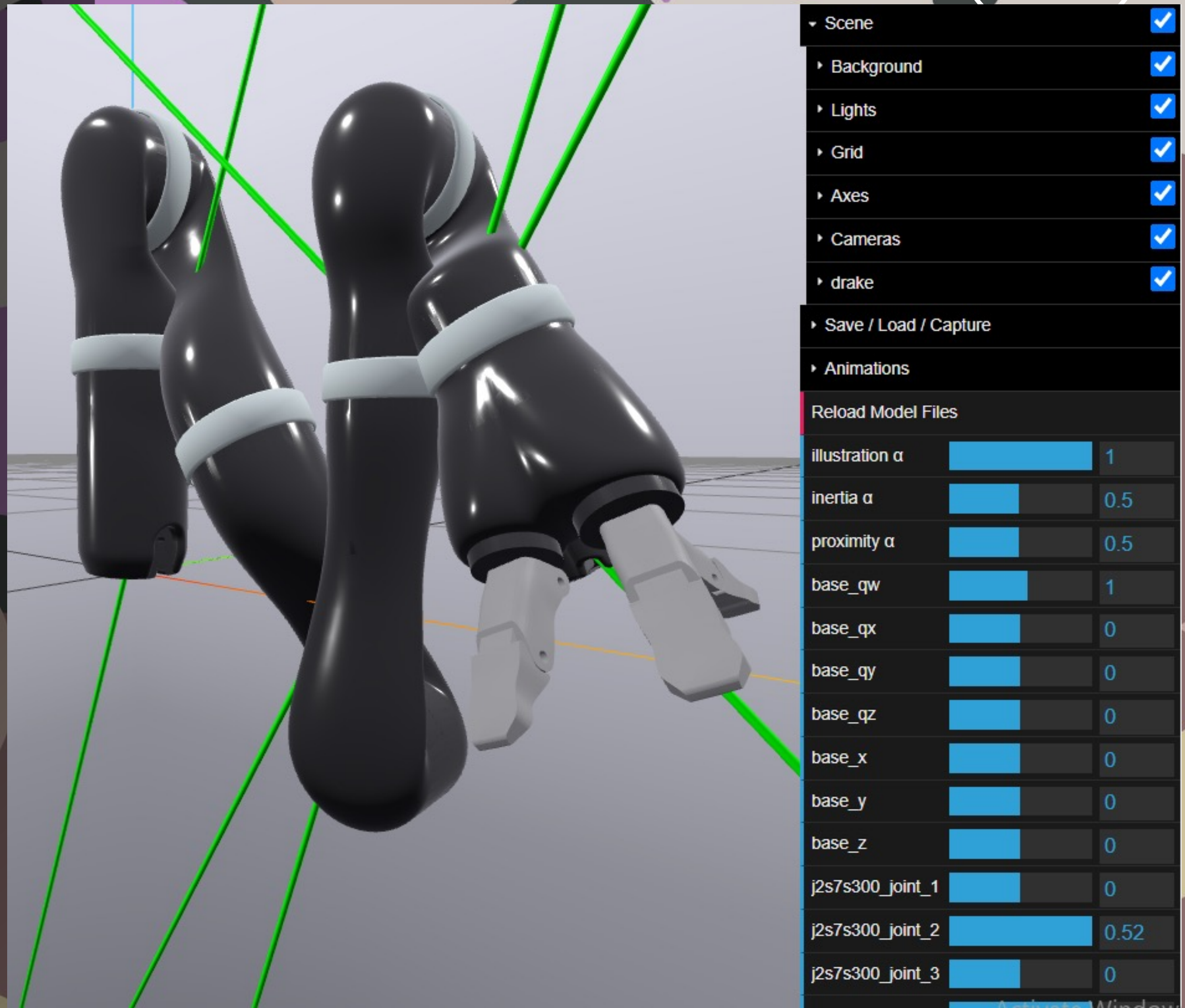
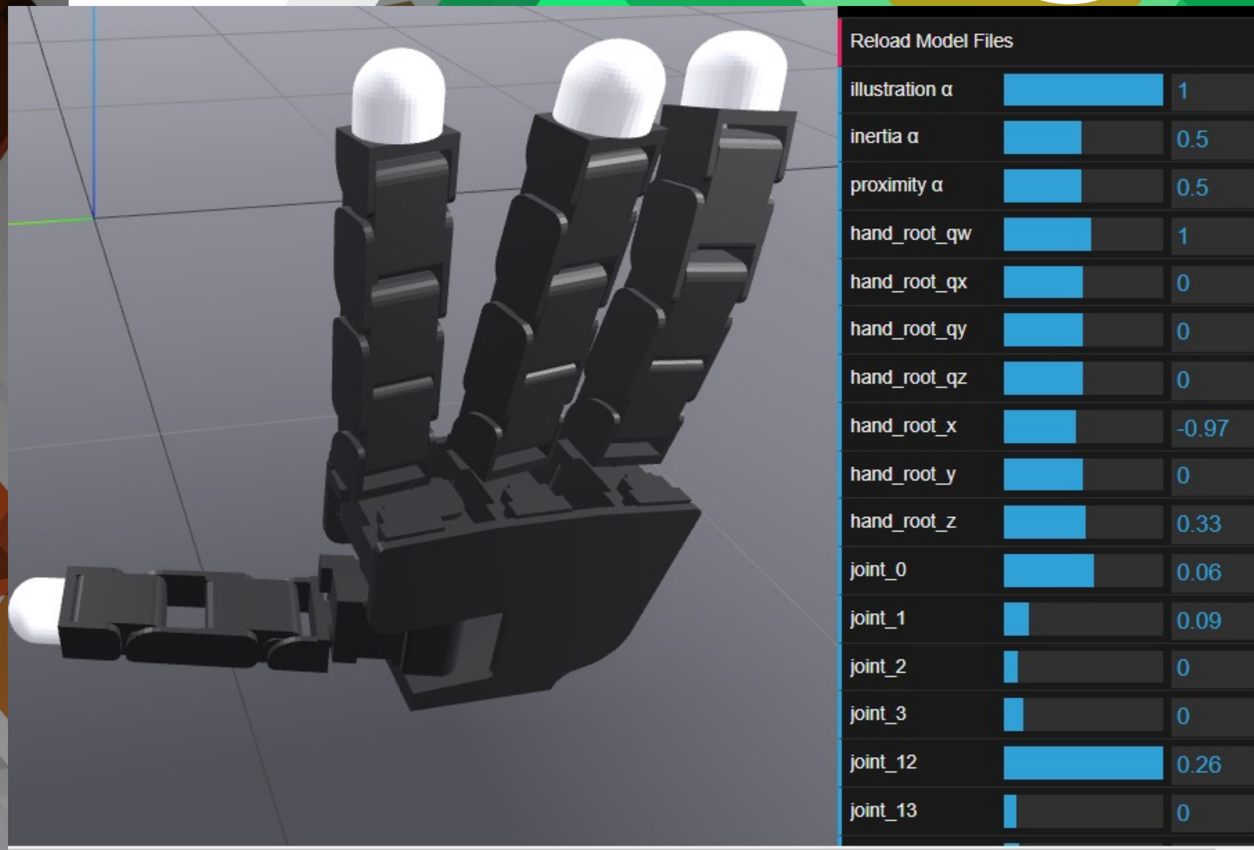
Essence of MIT – 6.421 Robotic Manipulation Chapter 1-3

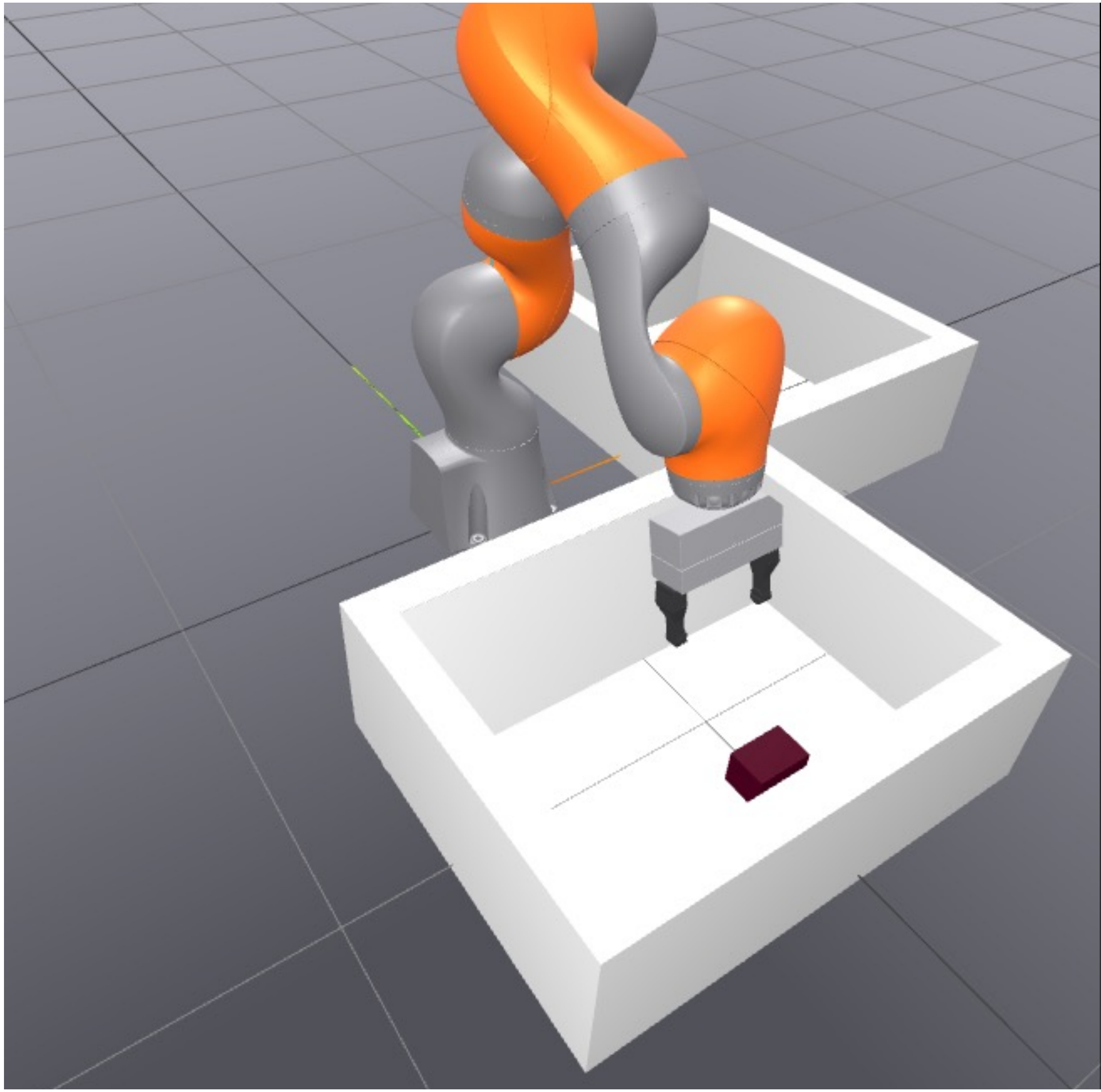


Using Drake software for robotics

Compiled by dew.ninja

Chapter 1
Introduction

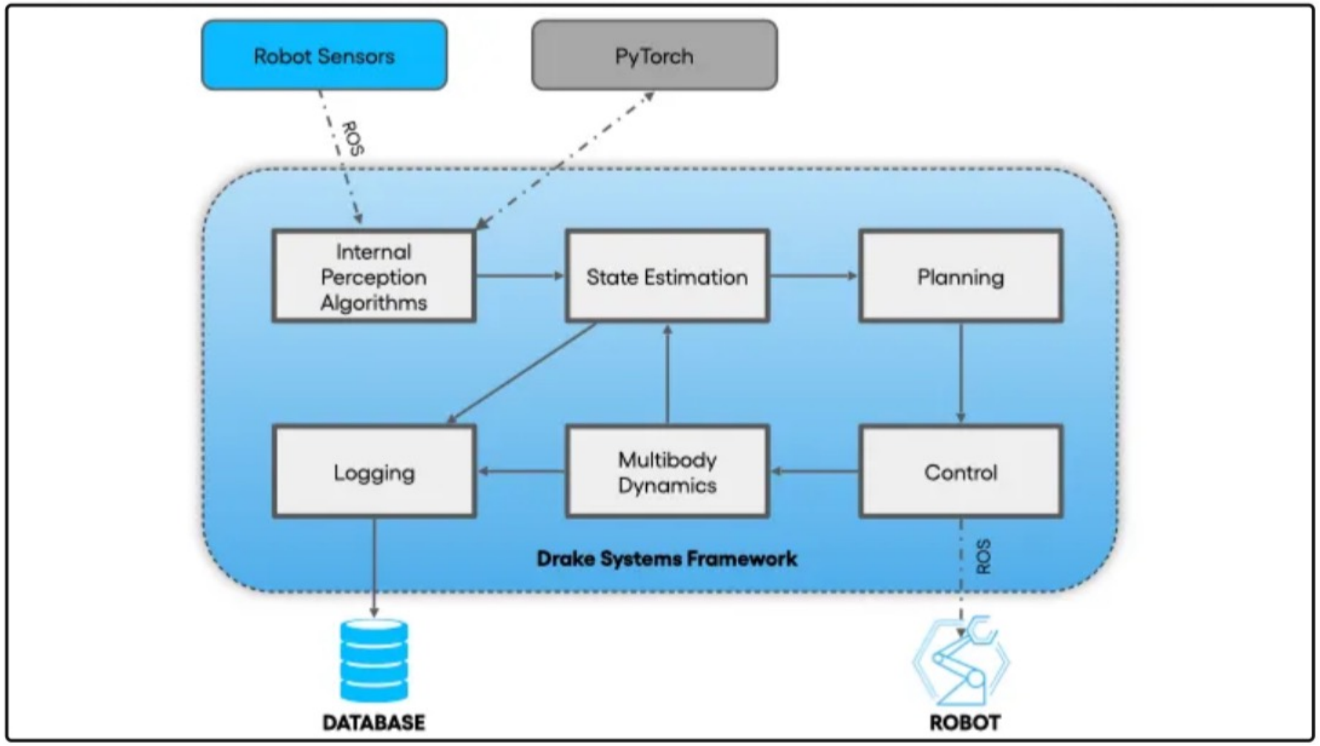




Ex 1.1-1.2 Teleoperation in 2D/3D

intro.ipynb in /book/intro

Drake Systems Framework

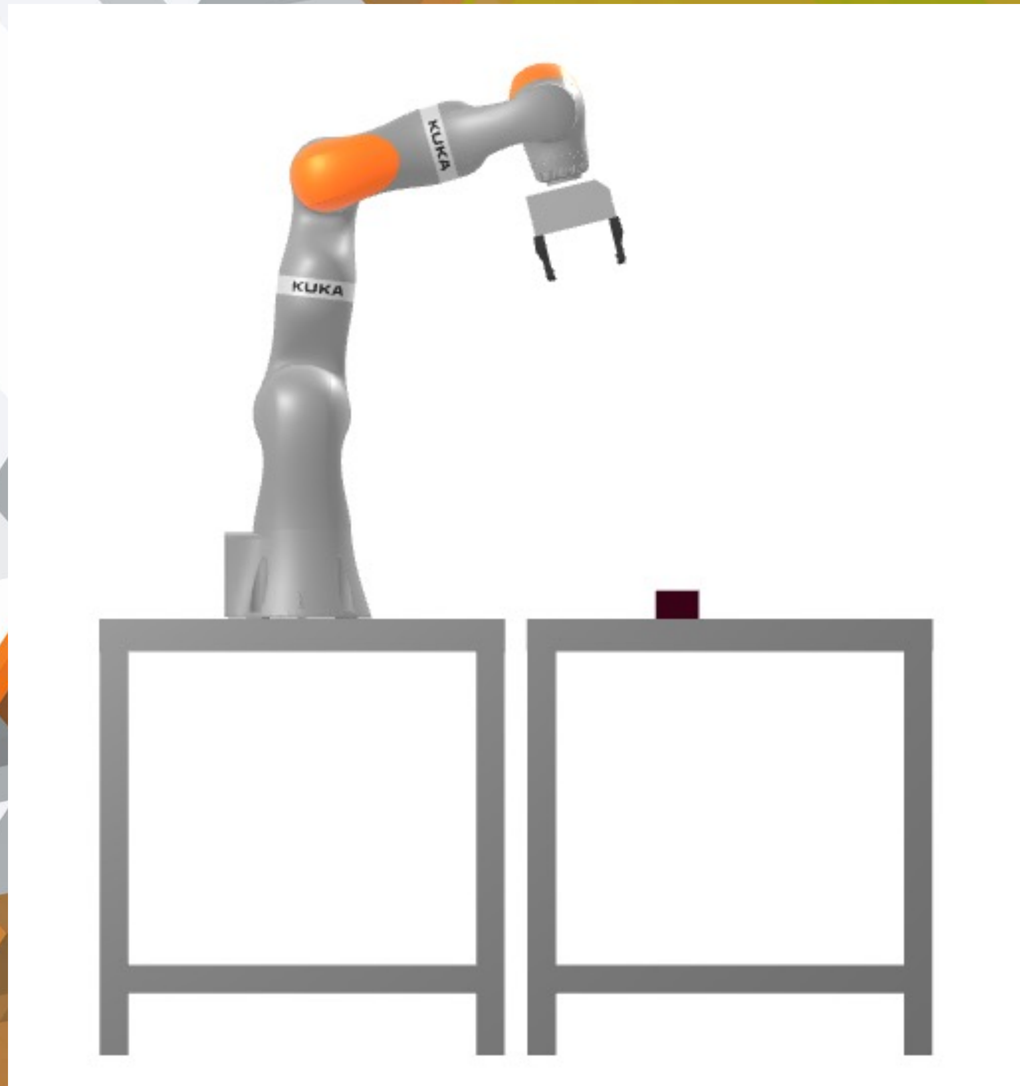


https://medium.com/toyotaresearch/drake-model-based-design-in-the-age-of-robotics-and-machine-learning-59938c985515

Typical Drake hierarchy

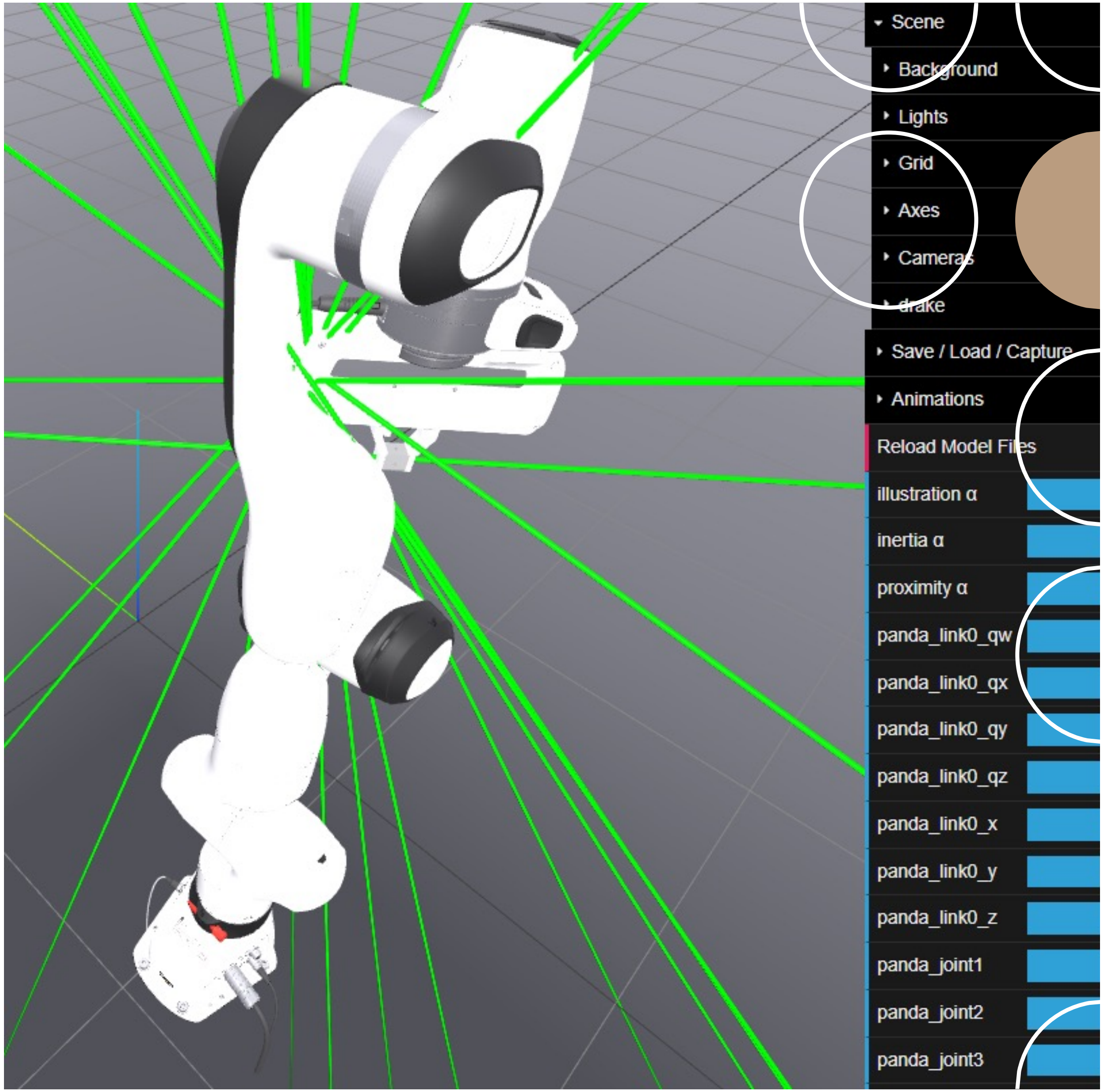
- Simulation
    - Diagrams
        - Systems
            - Static
            - Continuous
            - Discrete
            - Multibody
        - Other frameworks such as optimization, etc.

Chapter 2
Let’s get you a robot



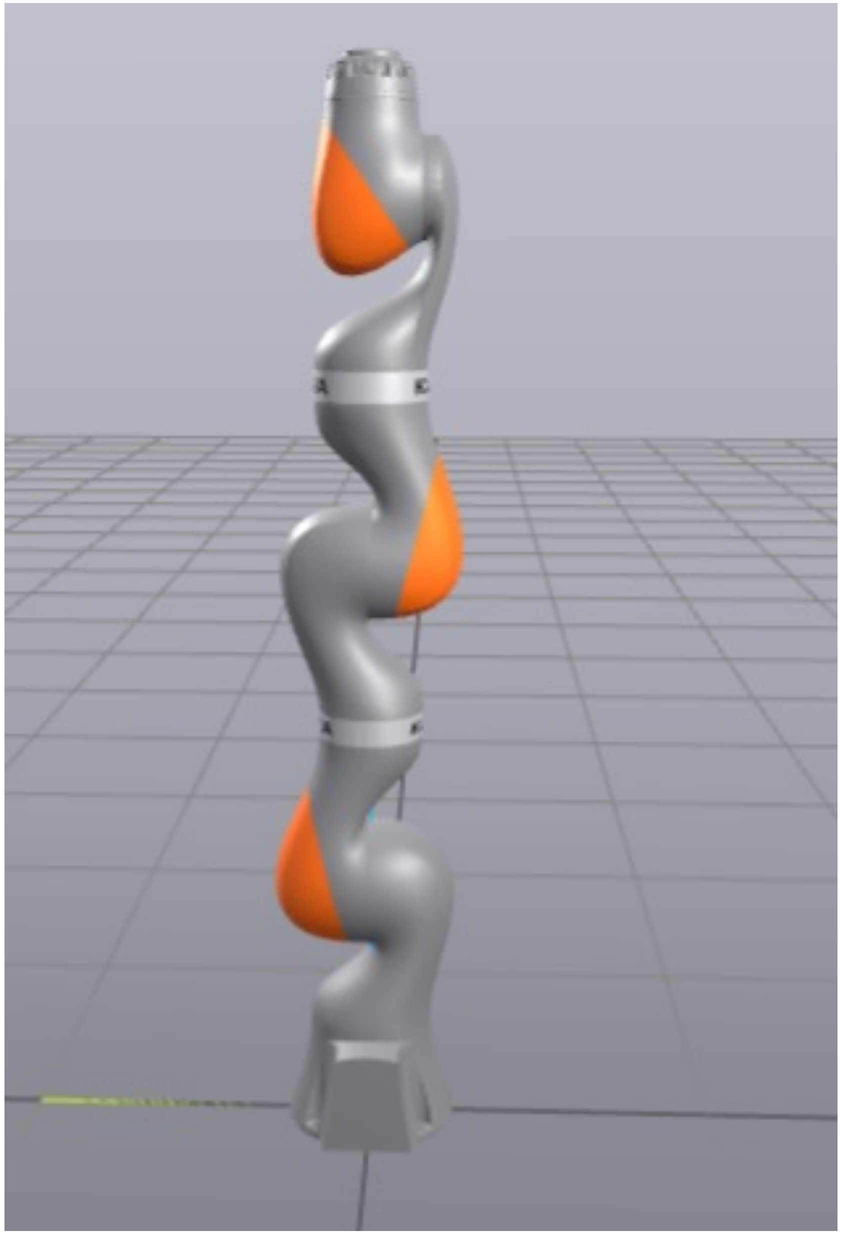
Robot examples

- /robot/inspector.ipynb
- Data from manufacturers
  - Universal Robot Description Format (URDF)
  - Simulation Description Format (SDF)
  - MuJoCo format (MJCF) (limited support)
- Drake Model Directives
  - YAML to load different file formats to one simulation



Robot classification

- Position-controlled
  - Most industrial robot arms are position-controlled using control scheme such as PID
  - electric motors are efficient at high speed
  - Torque-current relationship breaks down with substantial gear reductions, ex. 100:1
- Torque-controlled
  - Motor-side v.s. joint-side torque measurement
  - Make robot “huggable”



Simulating the Kuka iiwa

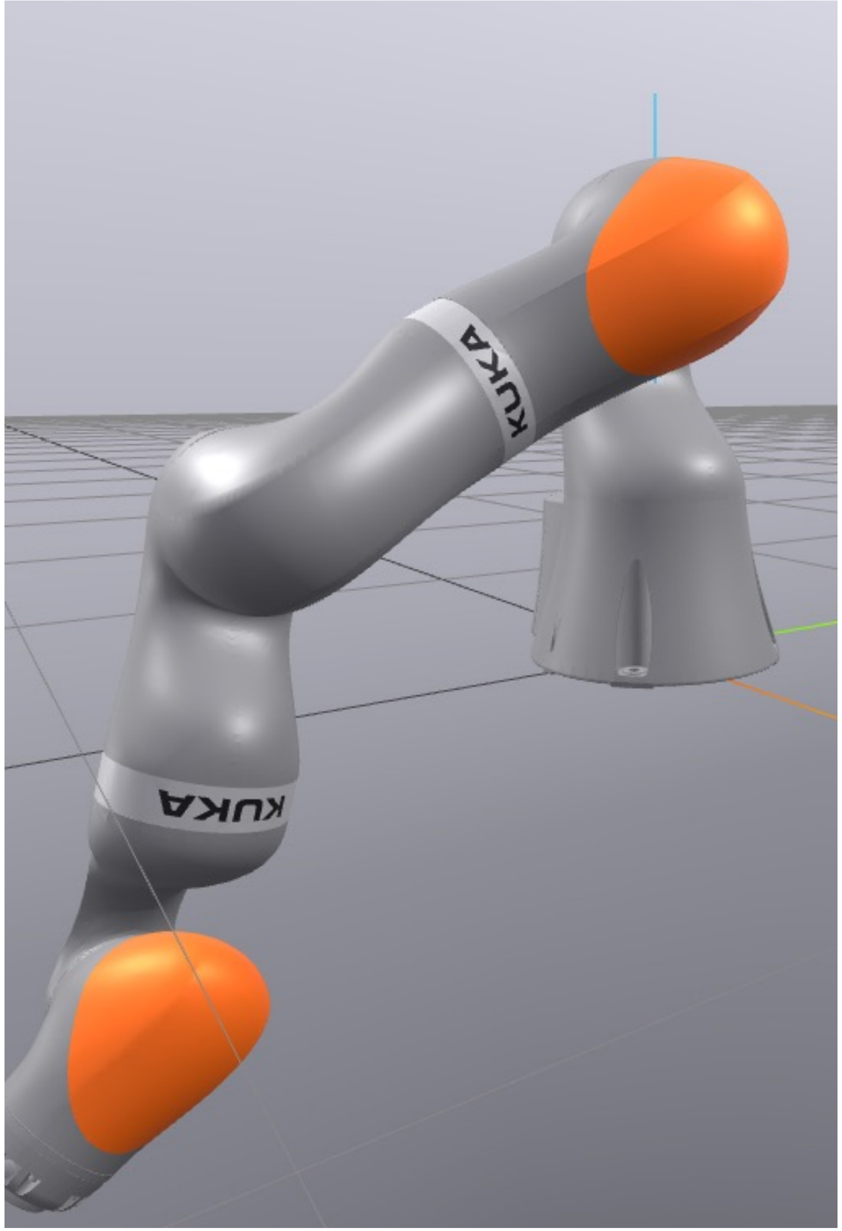
- Physics engine : MultibodyPlant
- Visualization : SceneGraph
- Simulation:
  - Passive iiwa (no control)
  - Add low-level control
- Run /robot/simulation.ipynb

Key points in simulation.ipynb (1)

- Load Kuka iiwa into physics engine as a MultibodyPlant
  - Specify time_step argument to make a discrete system
- Use context structure to pass values during simulation
- Fix 7 actuators value to zero
  - plant.get_actuation_input_port().FixValue(context, np.zeros(7))
- Simulate for 5 seconds to see changes in context structure

Key points in simulation.ipynb (2)

- SceneGraph is a system to creates geometry of screen
- MeshcatVisualizer is just convenient visualizer on Jupyter notebook
- Assemble MultibodyPlant, SceneGraph, MescatVisualizer into Diagram using DiagramBuilder.
- Notebook shows a shortcut from creating each system individually and connect ports together
- Create a diagram context and force publish to Meshcat Screen

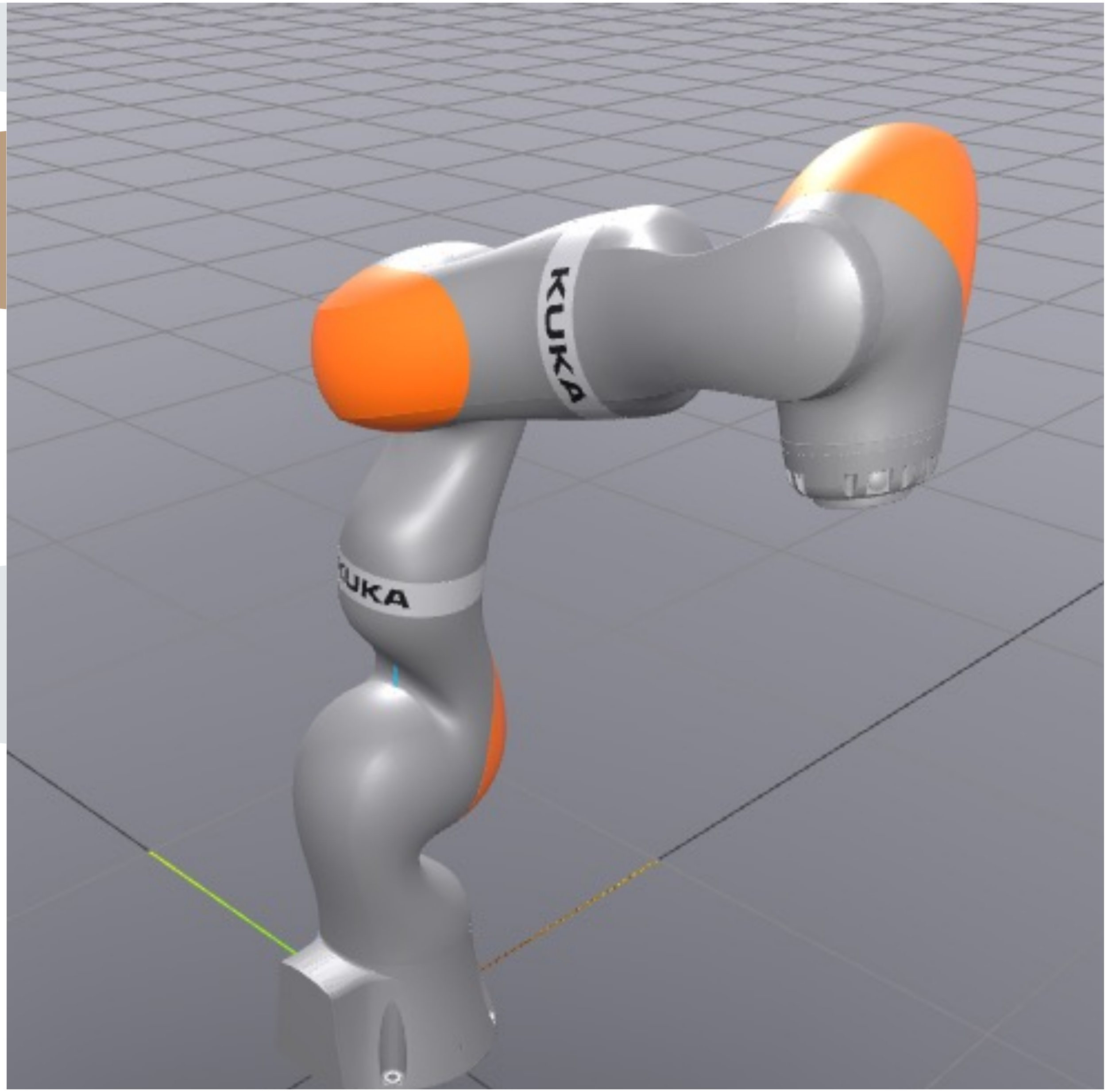


Key points in simulation.ipynb (3)

- To see a graphical block diagram, use RenderDiagram(diagram)
  - If you get error, perhaps graphviz is not installed. Fix by this command on terminal
    - Linux/wsl : sudo apt install graphviz
    - Mac-OSX : brew install graphviz
- Diagram context is just like a superset of plant context. The latter can be extracted to set the initial condition of robot.
- Simulate the robot with no control to see the arm falling down. (In the real robot, brakes are applied at joints to prevent this even when there is no control.)
- In order to run animation again, we must rerun code from the start. To make it convenient, all steps are written to a function animation_demo():

Key points in simulation.ipynb (4)

- Add controller to iiwa
- Set initial states in context. Desired position = current position. Desired velocicy = 0
- With control, the robot is now able to stay still amid the gravity.



HardwareStation

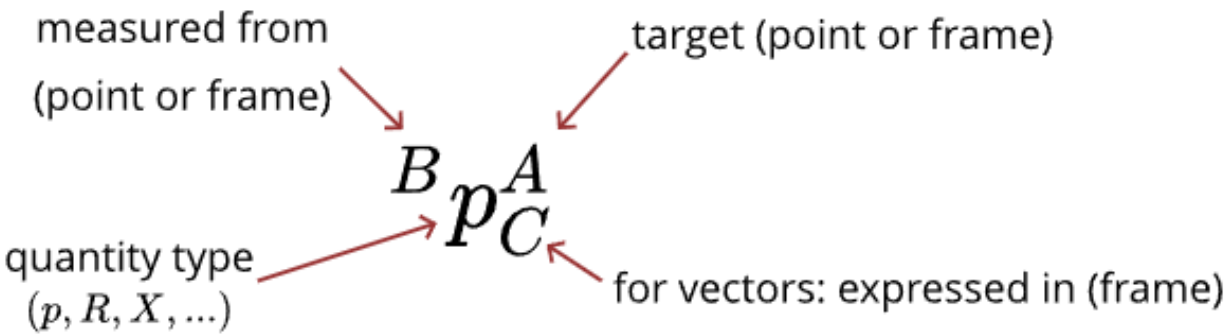
- MakeHardwareStation takes YAML description of the scene and robot hardware. See /intro/intro.ipynb and /robot/bimanual.ipynb
- Ports in orange do not exist in actual hardware platform. (They require ground-truth data)
- Switch to real hardware by passing hardware=True to MakeHardwareStation to create HardwareStationInterface in place of HardwareStation

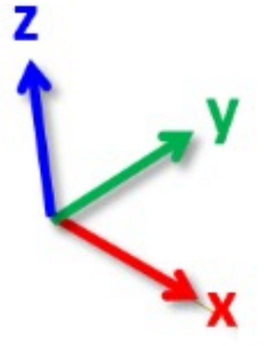
Chapter 2 exercises

- Ex 2.1 (role of reflected inertia)
  - notebook /robot/exercises/01_reflected_inertia.ipynb
  - read An aside: link dynamics with a transmission
  - implement pendulum_with_motor_dynamics() correctly
  - answer qualitative problem
- Ex 2.2 (input and output ports on manipulation station)
  - notebook /robot/exercises/02_hardware_station_io.ipynb
- Ex 2.3 (direct joint teleop in Drake)
  - notebook /robot/03_direct_joint_control.ipynb

Chapter 3 Basic pick and place

Notation used in book





XYZ = RGB

Spatial algebra example

- Addition $$ {}^{A}p_{F}^{B} + {}^{B}p_{F}^{C} = {}^{A}p_{F}^{C} $$ $$ {}^{A}p_{F}^{B} = -{}^{B}p_{F}^{A} $$
- Multiplication by rotation $$ {}^{A}p_{G}^{B} = {}^{G}R^{F}\,{}^{A}p_{F}^{B} $$
- Transforms $$ {}^{A}X^{B}\,{}^{B}X^{C} = {}^{A}X^{C} $$

See book and https://drake.mit.edu/doxygen_cxx/group__multibody__spatial__pose.html

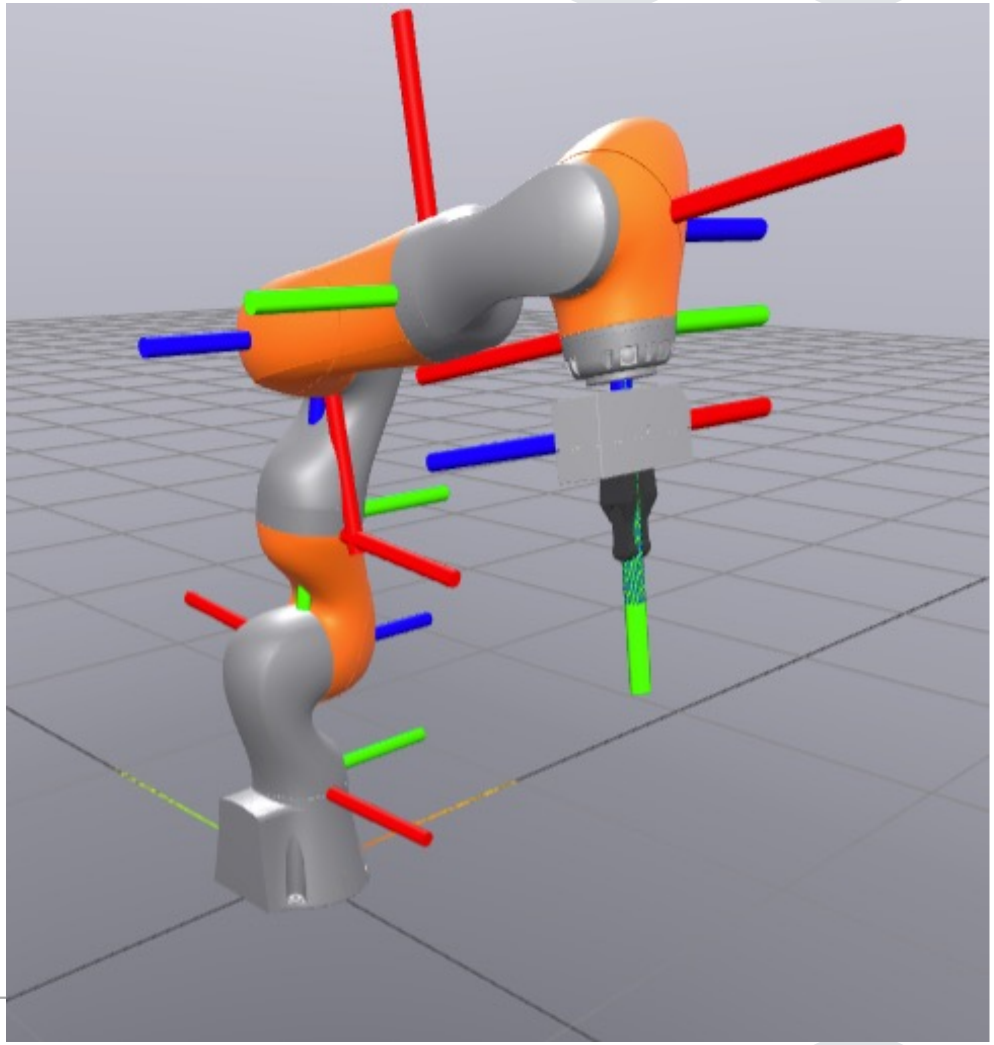
Representations for 3D rotation

- 3x3 rotation matrices
- Euler angles (e.g. roll-pitch-yaw)
- Axis angle
- Unit quaternions

Forward kinematics

$$X^G = f_{kin}^G(q)$$

- Ex 3.2 : Inspecting kinematic tree : /pick/kinematic_tree.ipynb
- Ex 3.3 : FK for gripper : /pick/forward_kinematics.ipynb

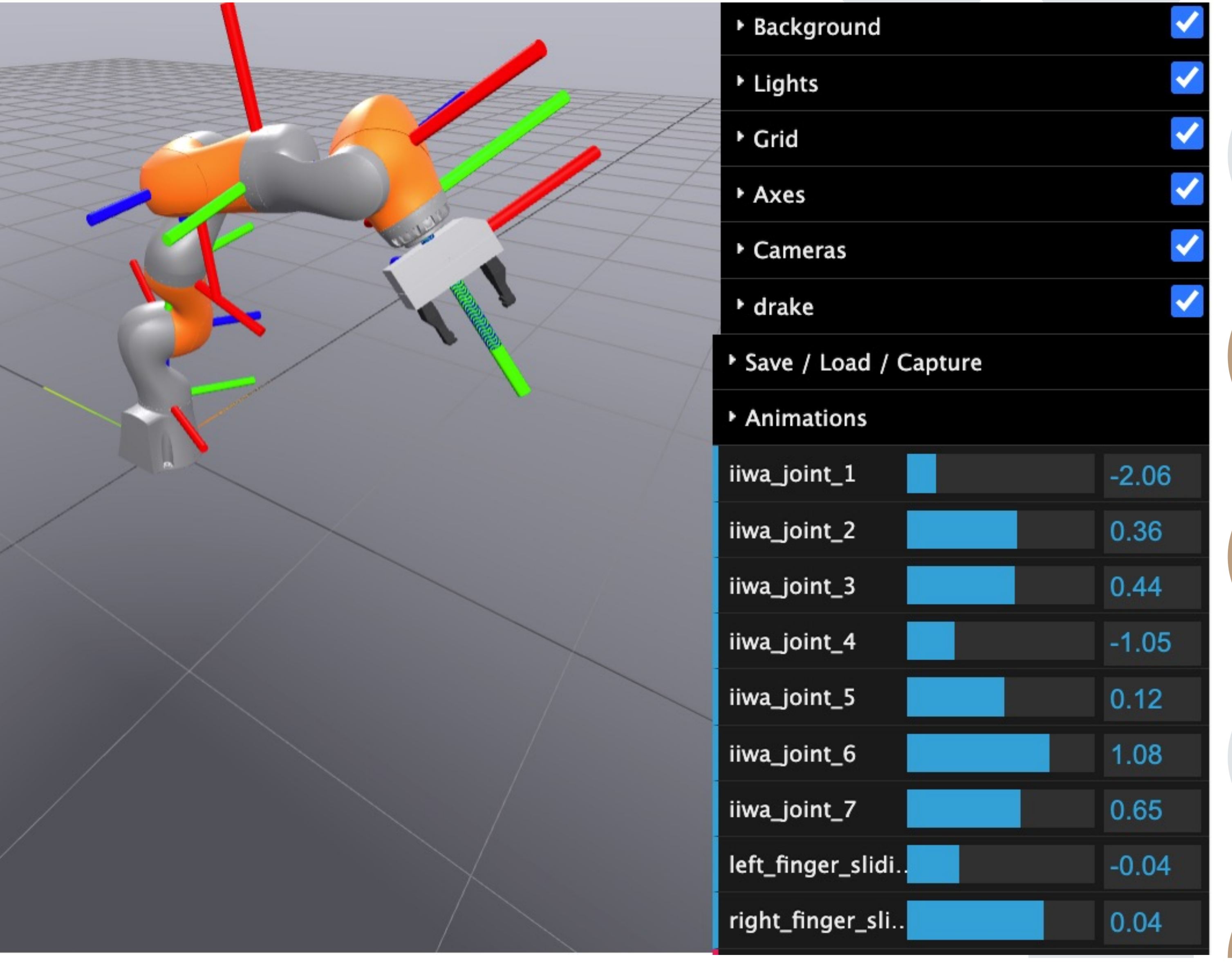


Kinematic tree

- /pick/kinematic_tree.ipynb
- Adding a body adds many degrees of freedom
- Adding a joint remove degrees of freedom. For example, adding a rotational joint between a child body and a parent body removes all but one degree of freedom.

$$\boxed{{}^{P}X^{C}(q) = {}^{P}X^{J_P}\,{}^{J_P}X^{J_C}(q)\,{}^{J_C}X^{C}}$$

adjust iiwa joints in Ex 3.3



Differential kinematics (Jacobian)

$$\boxed{dX^B = \frac{\partial f_{kin}^B(q)}{\partial q} dq = J^B(q) dq} \qquad (10)$$

- describes how changes in the joint angles relate to changes in the gripper pose
- Spatial velocity

$$\boxed{{}^{A}V_{C}^{B} = \begin{bmatrix} {}^{A}\omega_{C}^{B} \\ {}^{A}v_{C}^{B} \end{bmatrix}}$$

angular velocity

translational velocity

Differential kinematics (Jacobian)

- Ex 3.5 : (qdot not equal v) /pick/qdot_vs_v.ipynb
- explicit partial derivative of FK in (10) is called analytic Jacobian
- Drake uses geometric Jacobian, which replaces 3D rotations on left-hand side with spatial velocities
- Ex 3.6: (kinetic Jacobian for pick and place) /pick/jacobian.ipynb

Differential inverse kinematics

From the geometric Jaobian $\boxed{V^G = J^G(q)v}$ that describes linear relationship between generalized velocity and spatial velocity

Given $\boxed{V^{G_d}}$ want to compute joint velocity $\boxed{v = (J^G(q))^{-1} V^{G_d}}$

One problem is the inverse may not exist. ex., for the iiwa $\boxed{J^G(q) \in \mathrm{R}^{6\times 7}}$

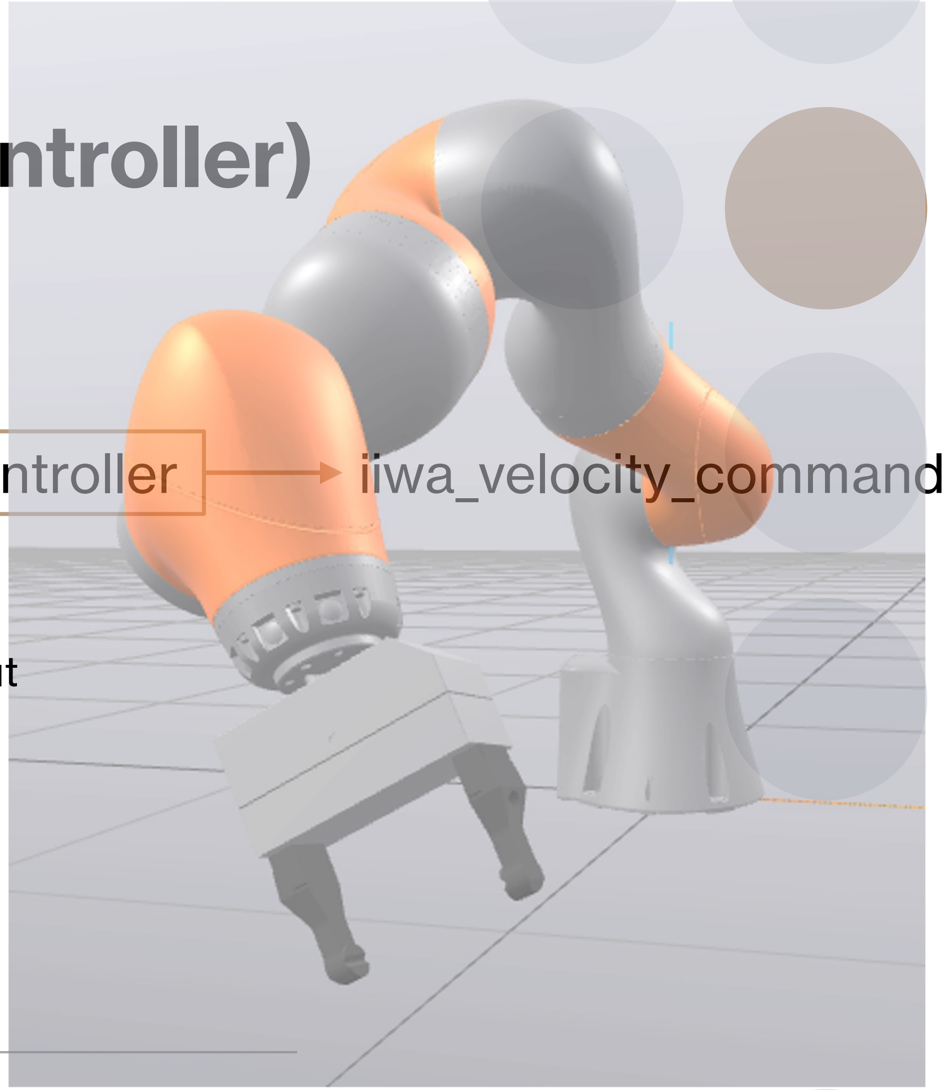
Moore-Penrose pseudo-inverse

$$\boxed{v = (J^G(q))^{+} V^{G_d}}$$

Ex 3.7 (PseudoInverseController)

/pick/pseudoinverse.ipynb

iiwa_position → PseudoInverseController → iiwa_velocity_command



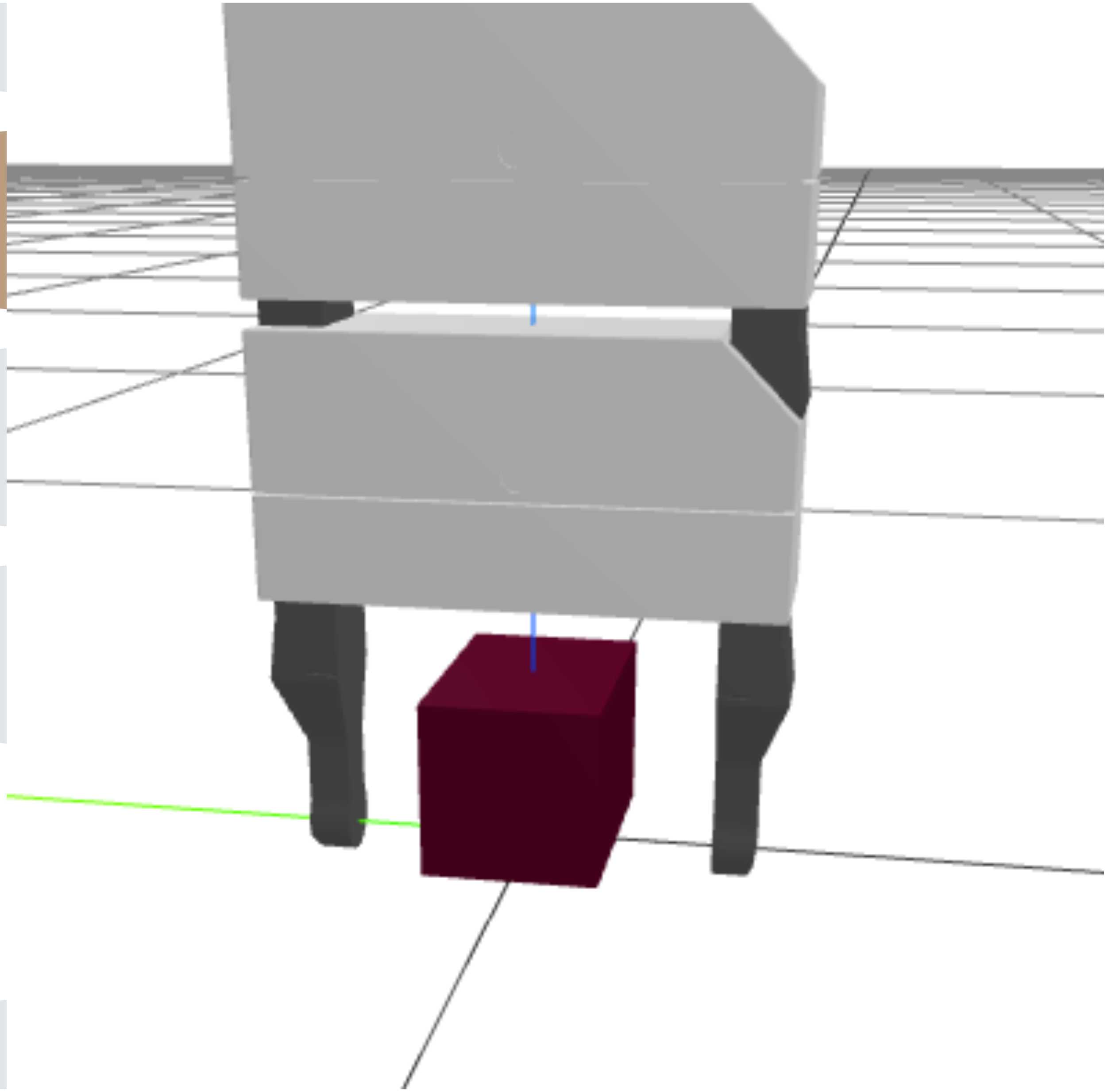
An integrator is needed to convert joint velocity output
To joint position.

Invertibility of Jacobian

- Check rank(J) = number of rows (full row rank)
- Problematic for real robot. J gets close to losing rank under noisy joint position.
- Better check smallest singular value. As it approaches zero, norm of pseudo-inverse approach infinity.
- Ex 3.8 : /pick/jacobian.ipynb. Find a singularity configuration, ex. Joint 2 and 4 set to zero, and observer smallest singular value.

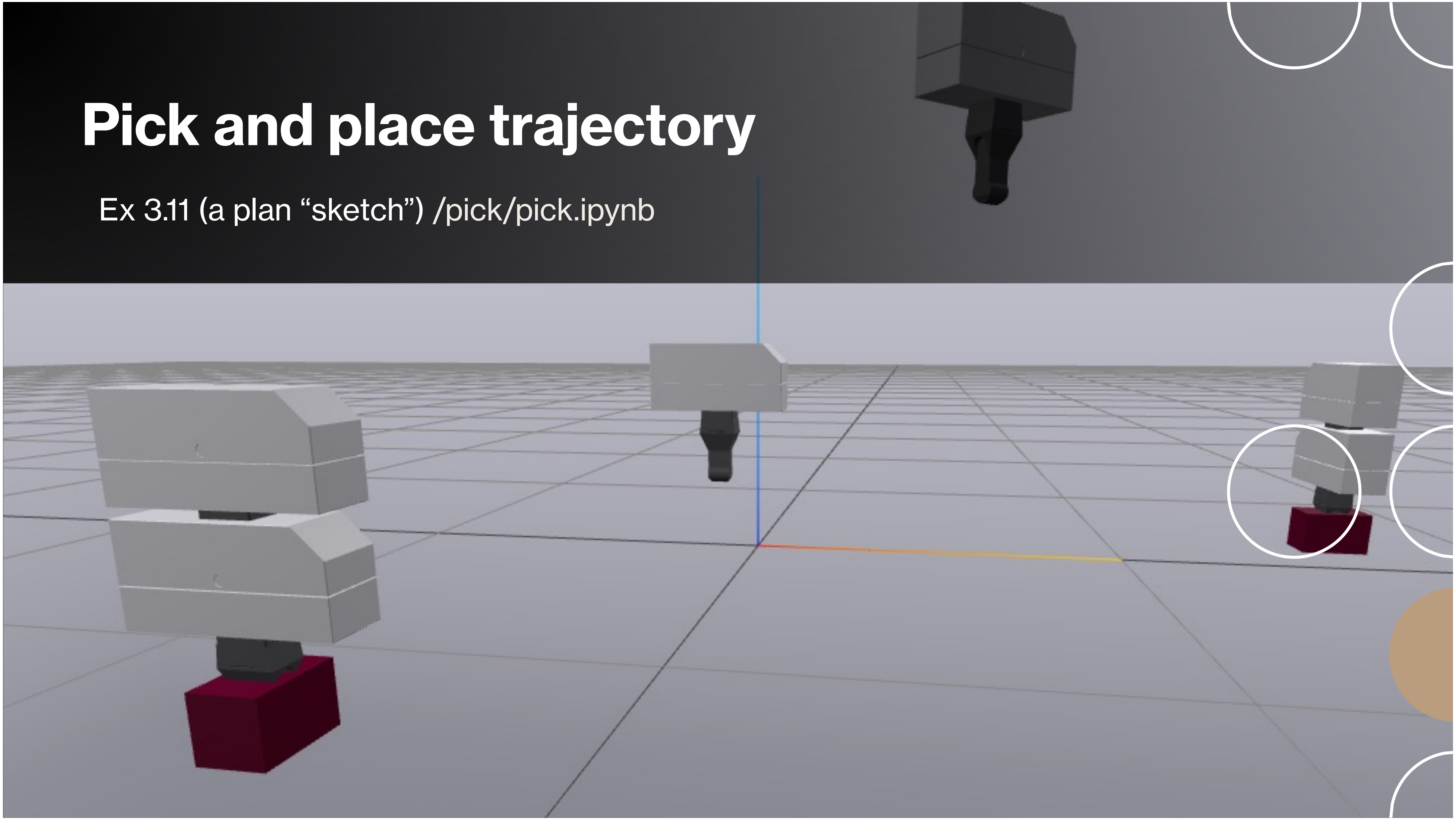
Defining grasp and pre-grasp poses

Ex 3.10 : /pick/grasp.ipynb

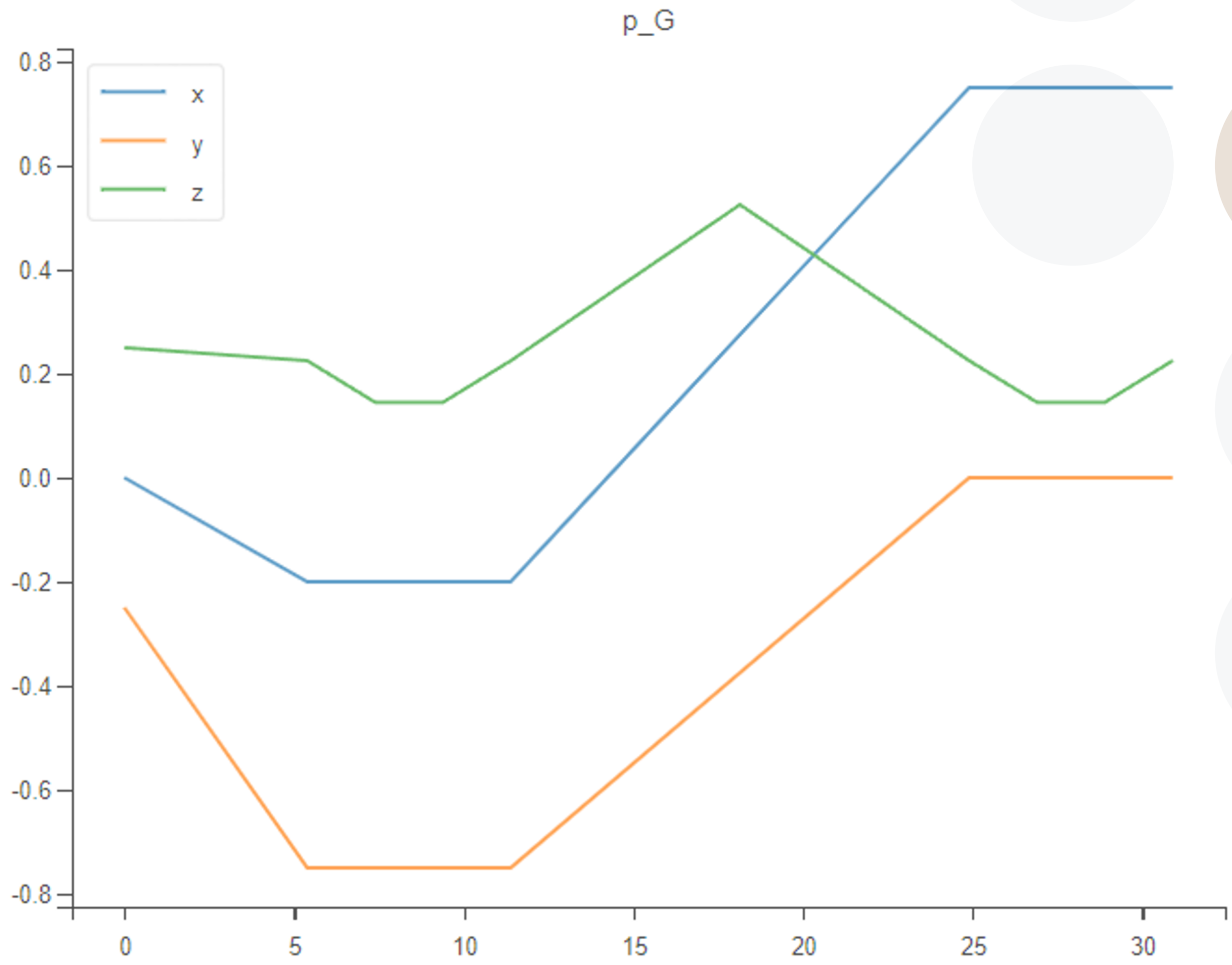


Pick and place trajectory

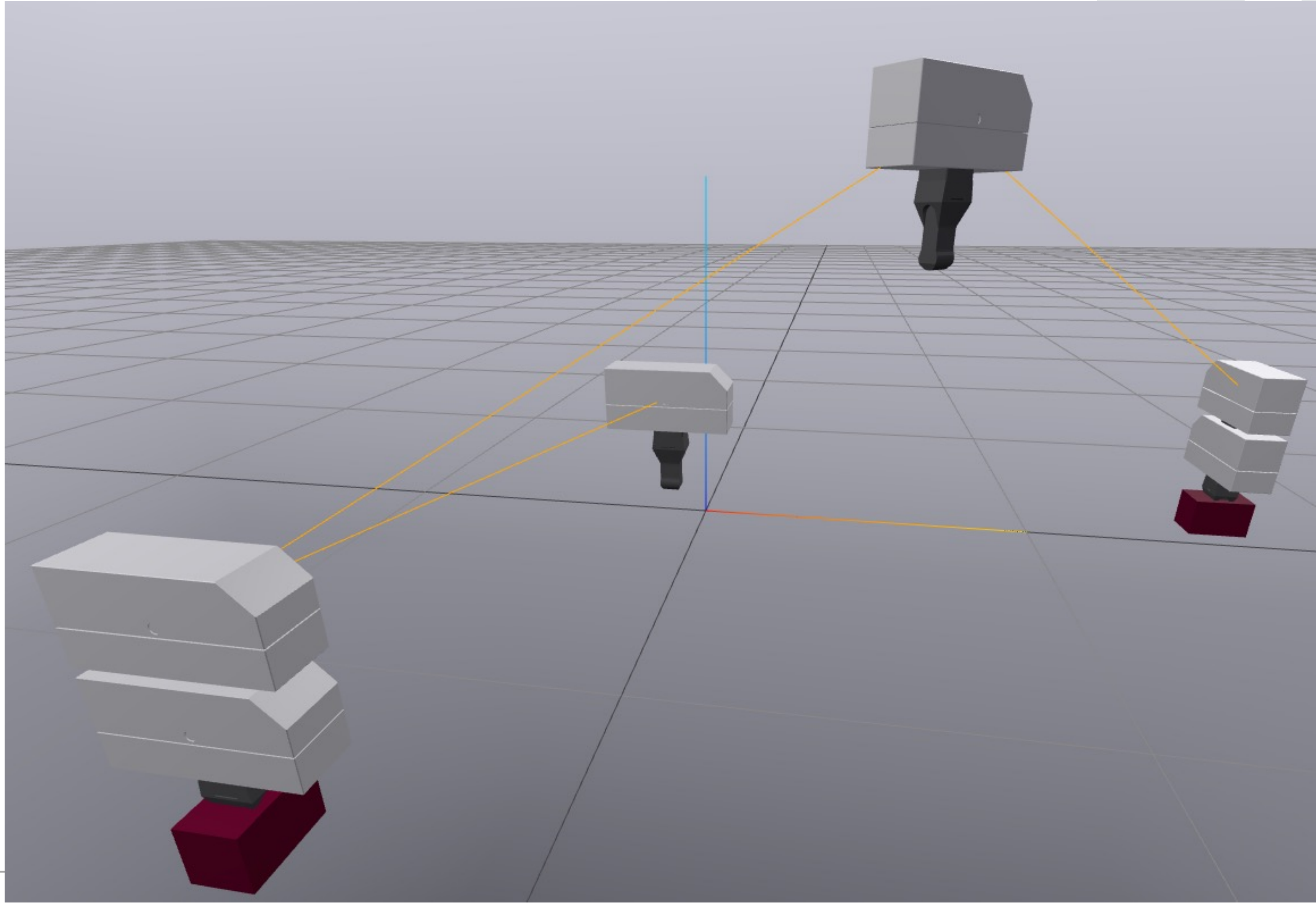
Ex 3.11 (a plan "sketch") /pick/pick.ipynb



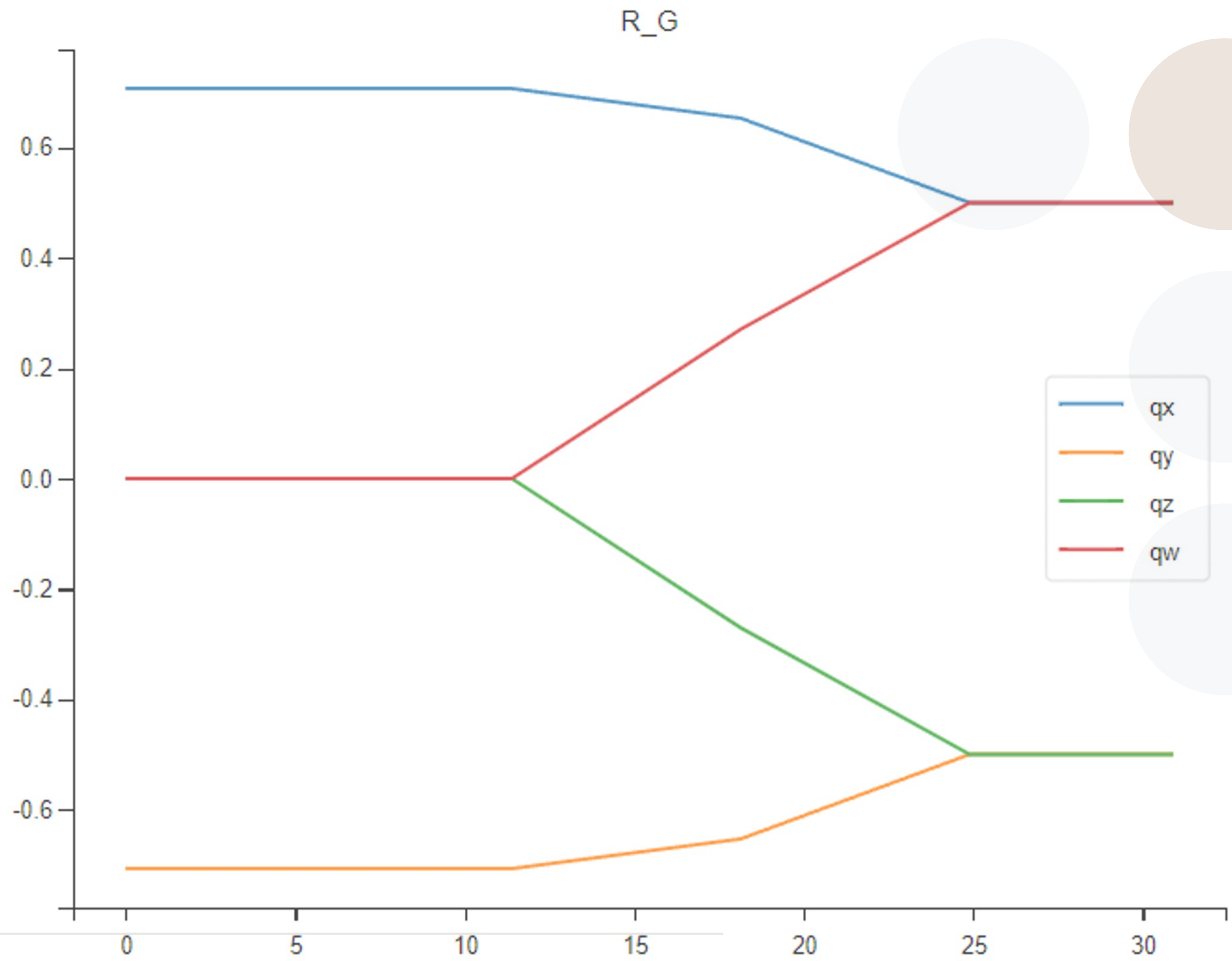
Pick and place trajectory



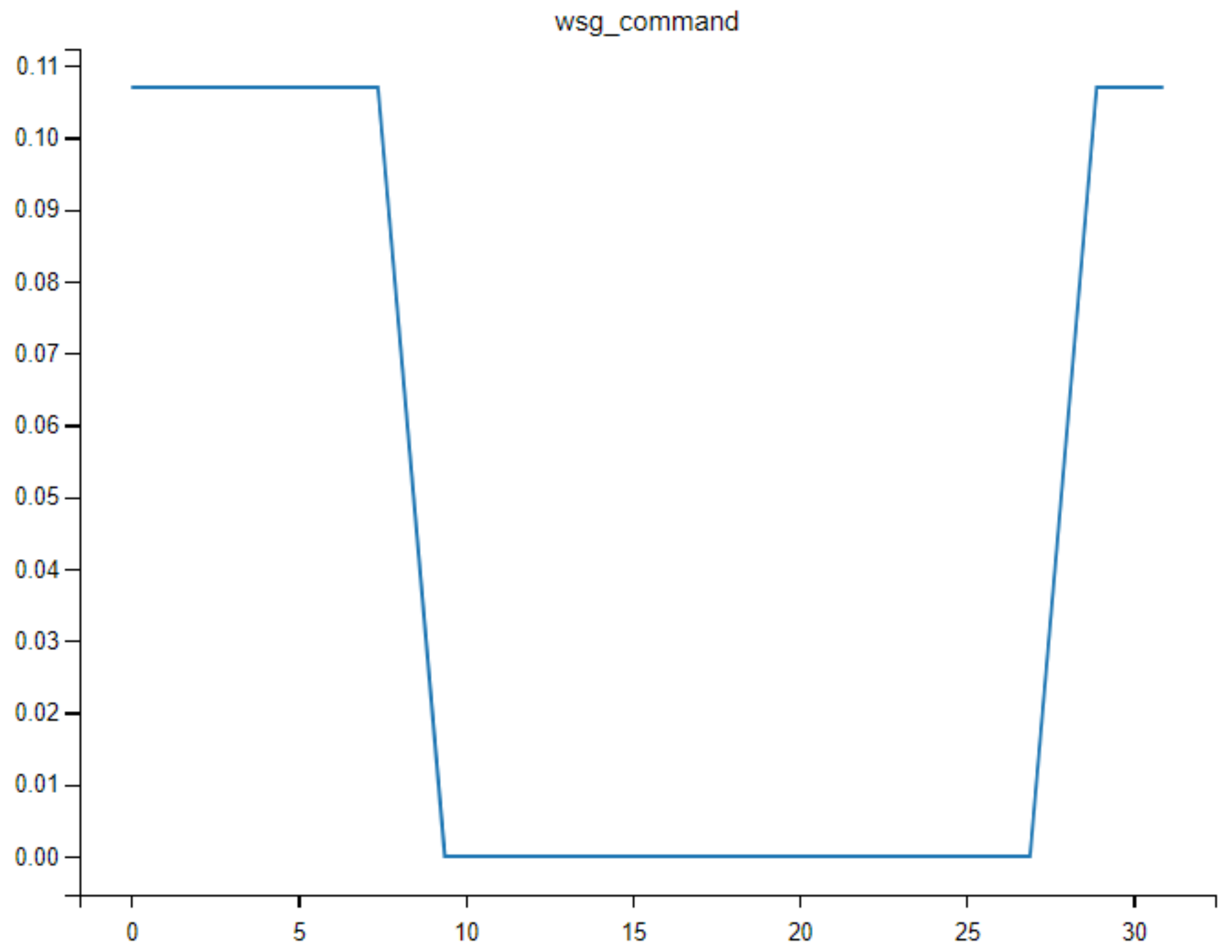
Pick and place trajectory



Pick and place orientation trajectory

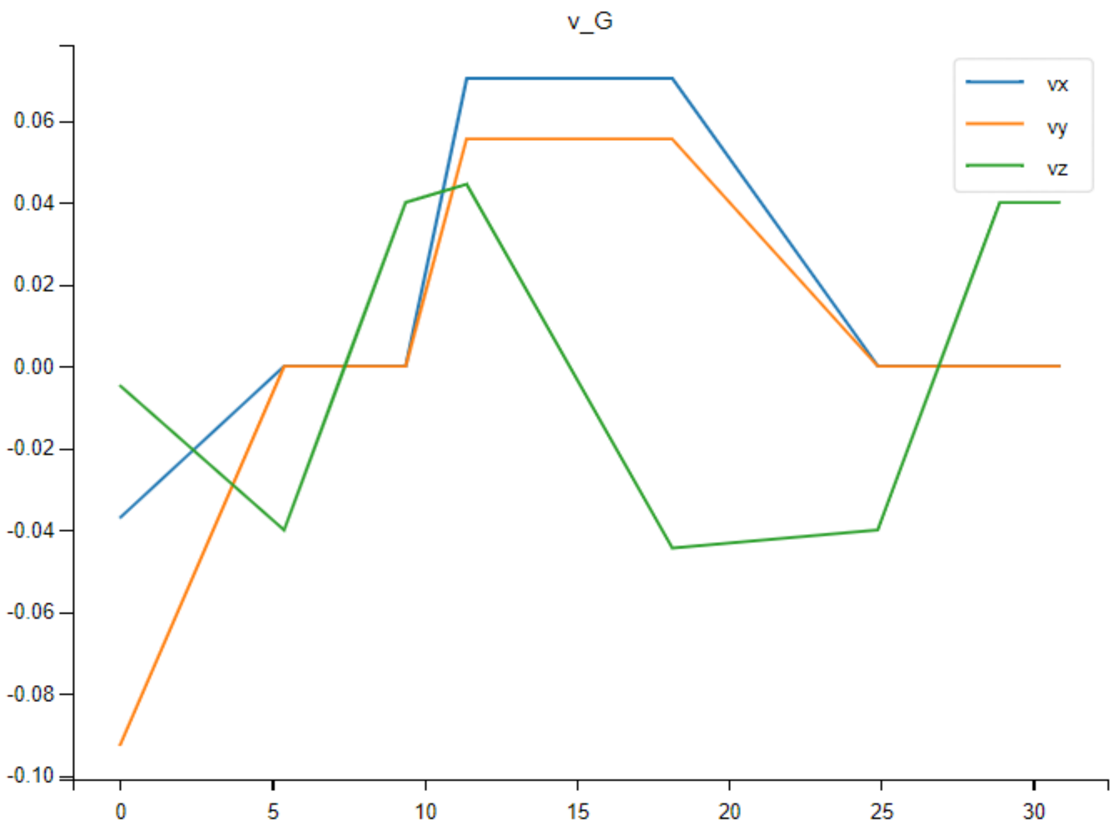


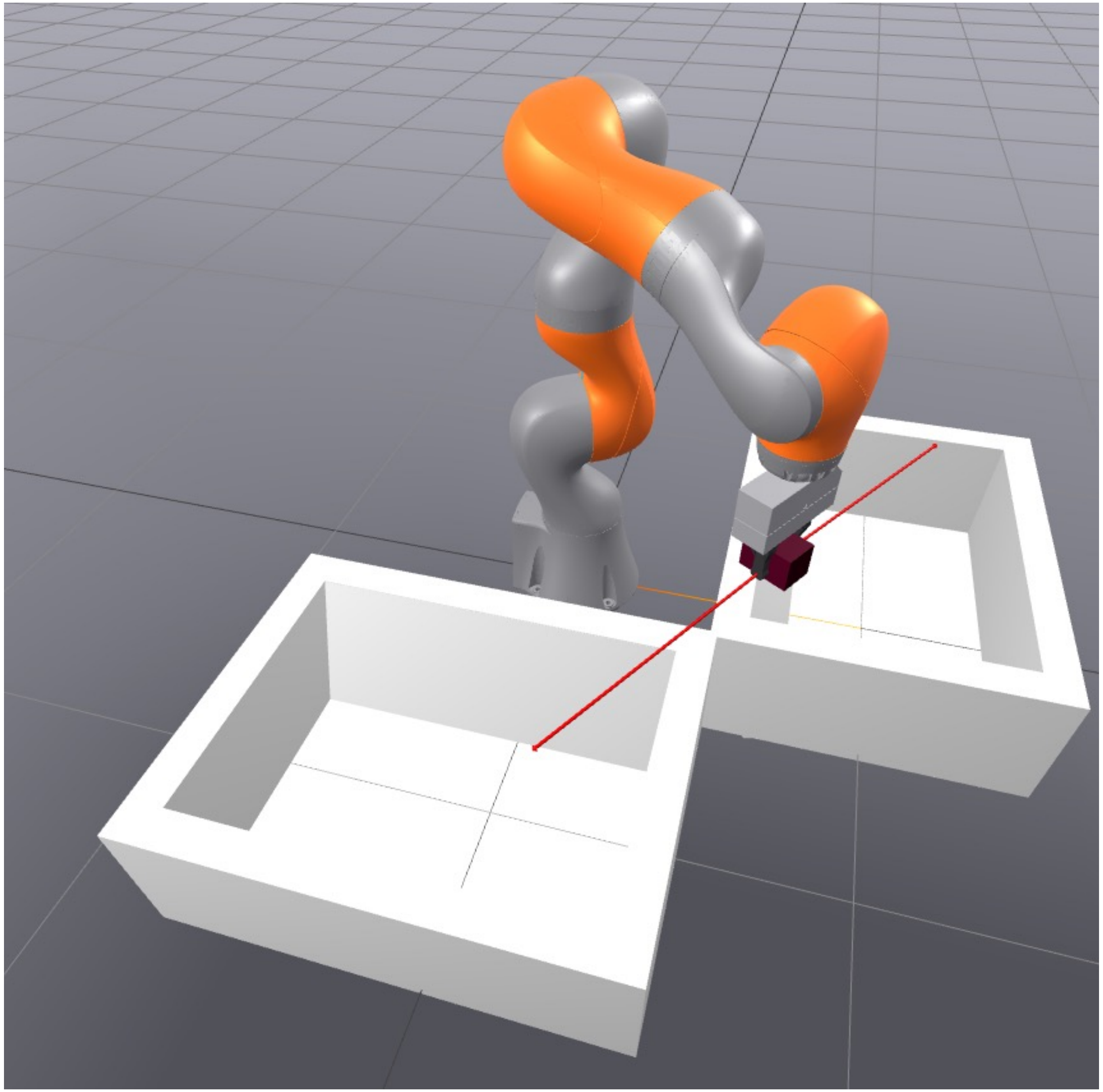
finger trajectory



Spatial velocities

- The Jacobian controller takes spatial velocities as input so we need to differentiate the positions and orientation





Simulation result

This covers up to Example 3.13 (the full pick and place demo)

Take about 30 seconds to finish

Point to consider: integration could accumulate errors.

Differential inverse kinematics with constraints

- Problems with pseudo-inverse controller
  - Does not perform well around singularities. (Result in large velocity commands)
  - Clipping of large velocity command by the controller could send robot trajectory off course.
- Solution:
  - Take constraints into account by the controller.

Pseudo-inverse as an optimization

- Pseudo-inverse is just solution to least-square optimization problem

$$\min_{v} \left\| J^{G}(q)v - V^{G_d} \right\|_2^2$$

- Denote the optimal solution as v^*

$$v^* = (J^{G}(q))^{+} V^{G_d}$$

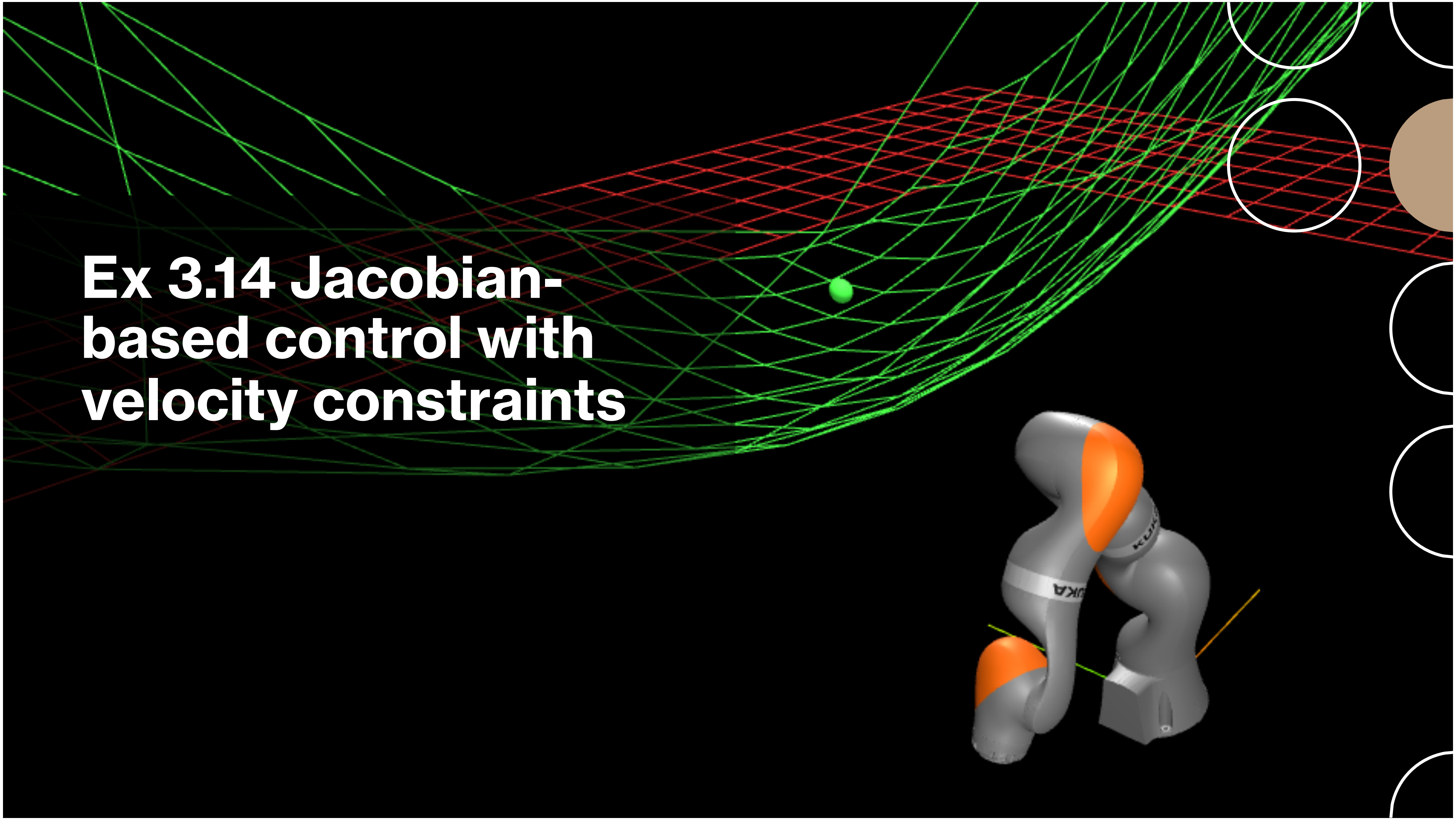
Adding velocity constraints

$$\boxed{\min_{v} \left\| J^{G}(q)v - V^{G_d} \right\|_2^2}$$

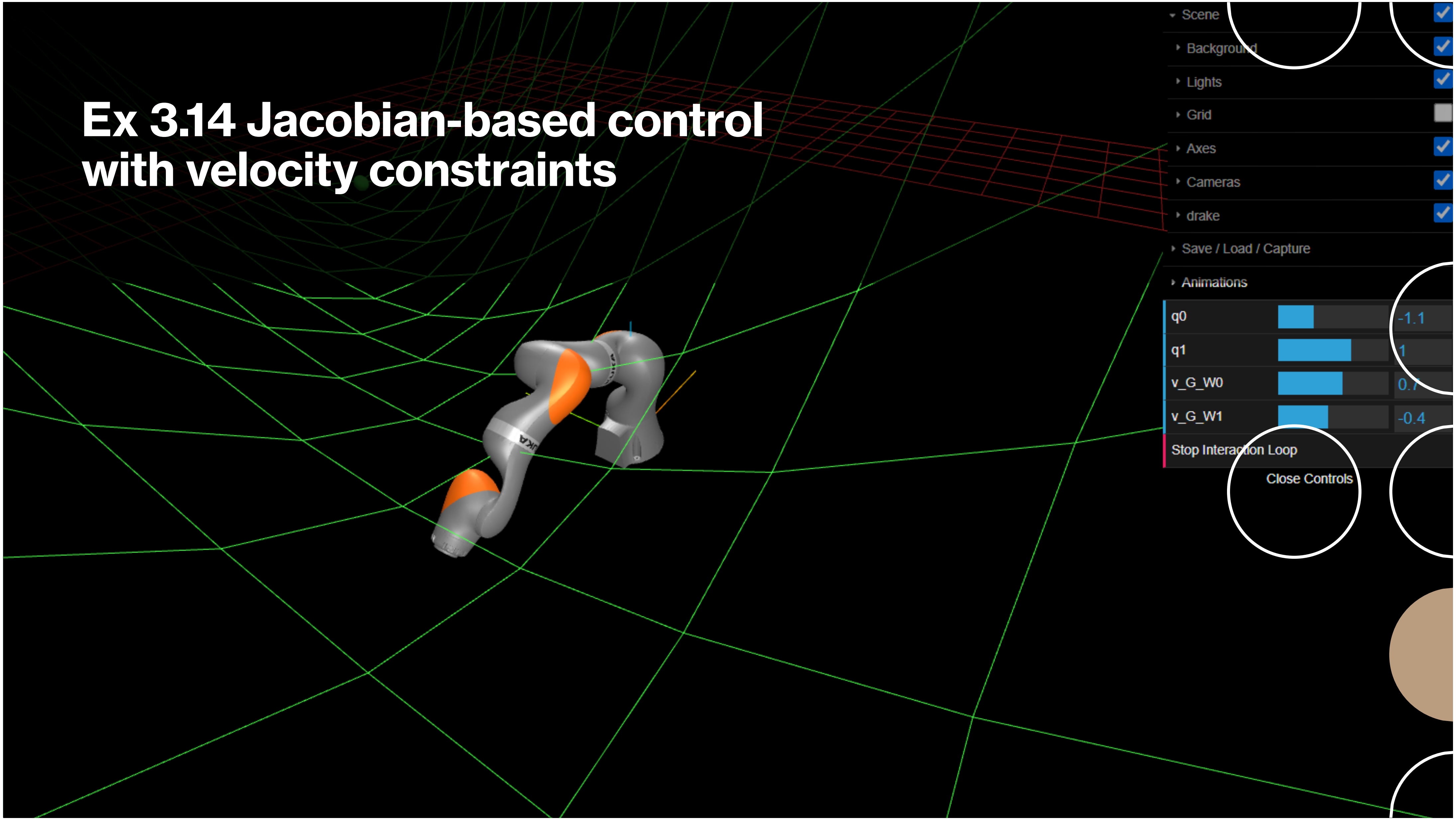
Subject to $v_{min} \leq v \leq v_{max}$

- Convex quadratic objective with linear constraints ->convex Quadratic programming (QP)
- To be solved by controller at each time interval
- Ex 3.14 : /pick/qp_diff_ik.ipynb

Ex 3.14 Jacobian-based control with velocity constraints



Ex 3.14 Jacobian-based control with velocity constraints



Adding position and acceleration constraints

$$\boxed{\min_{v} \left\| J^{G}(q) v_n - V^{G_d} \right\|_2^2}$$

Subject to $v_{\min} \le v \le v_{\max}$

$$q_{\min} \le q + h v_n \le q_{\max}$$

$$\dot{v}_{\min} \le \frac{v_n - v}{h} \le \dot{v}_{\max}$$

Exercises with notebooks /pick/exercises/*

- 3.5 – 3.6 (Planar manipulator) : 05_planar_manipulator.ipynb
- 3.7 (spatial transform and grasp pose) : 07_rigid_transforms.ipynb
- 3.8 (robot painter) : 08_robot_painter.ipynb
- 3.9 (introduction to QP) : 09_intro_to_qp.ipynb
- 3.10 (virtual wall) : 10_differential_ik_optimization.ipynb

References

- Drake website : https://drake.mit.edu/
- Tedrake, Russ. Robotic Manipulation : Perception, Planning, and Control, Course Notes for MIT 6.421, 2023 URL : http://manipulation.mit.edu