Simulation, Control, and Replay in Python

Jan Margeta

Hi, I am Jan 👋

Building AI and computer vision tools

- 🐍 Pythonista since 2008
- 👨‍🎓 MSc in Computer Vision and Robotics
- 👨‍🔬 PhD in Medical imaging
- 🫀 Founder of KardioMe building medical imaging software
- 🔬 Researching new technologies

Reach out to jan@kardio.me or see more on 𝕏 @jmargeta

Three areas we will talk about today

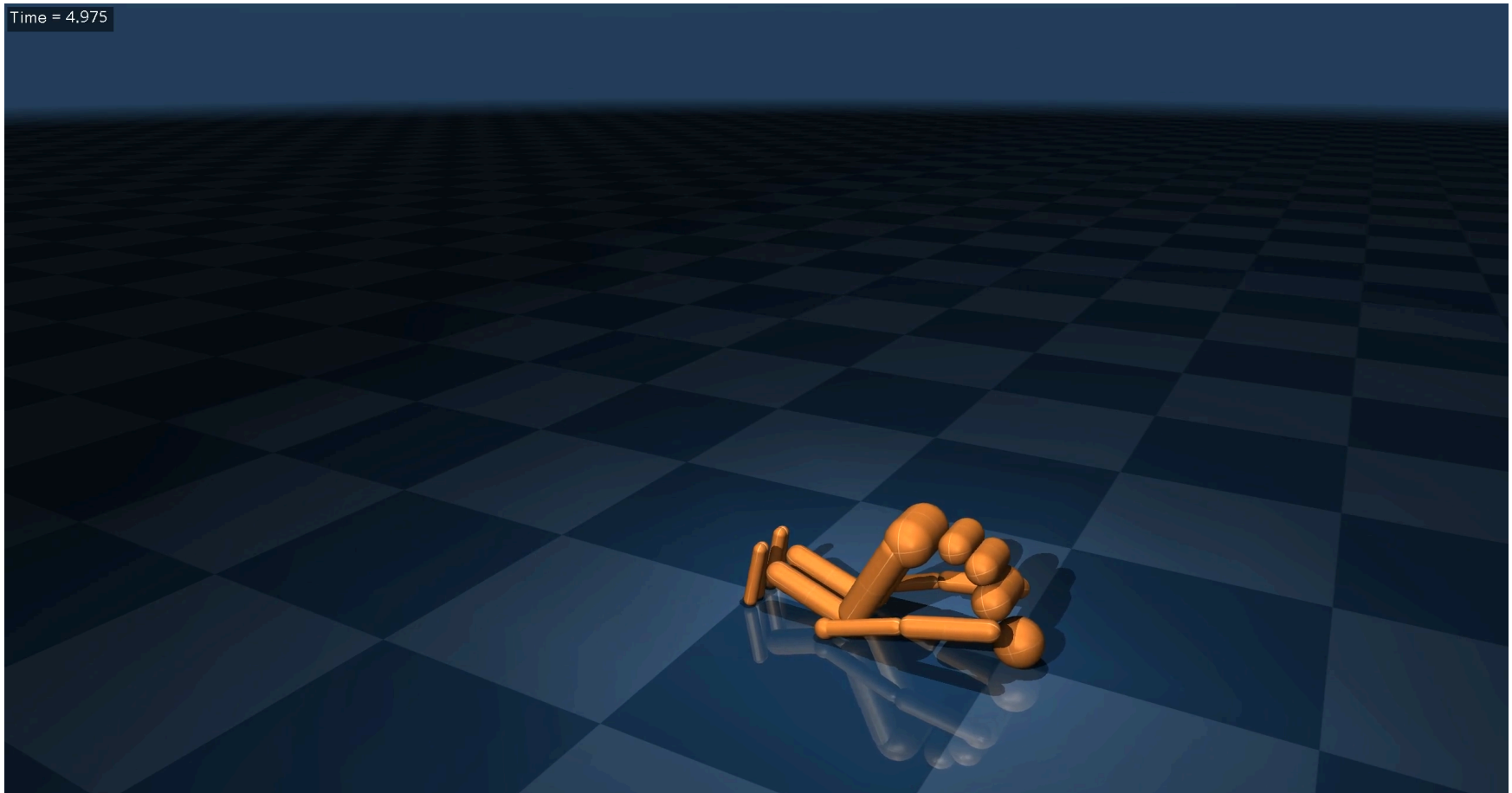
- Simulation
- Control
- Replay

Part I: Simulate

🍎 Which physics to simulate?

- Classical Mechanics
- Quantum Mechanics
- Astrophysics and Cosmology
- Fluid dynamics
- Thermodynamics
- Electromagnetism
- Condensed matter physics
- Nuclear and particle physics
- Acoustics and wave phenomena
- Biophysics
- …

In Python? Surely You're Joking, Mr. van Rossum!



Physics simulators in Python

Multiphysics

- SOFA Framework - FEM, real-time, haptics, VR, plugin system
- PyChrono

Mostly rigid body dynamics

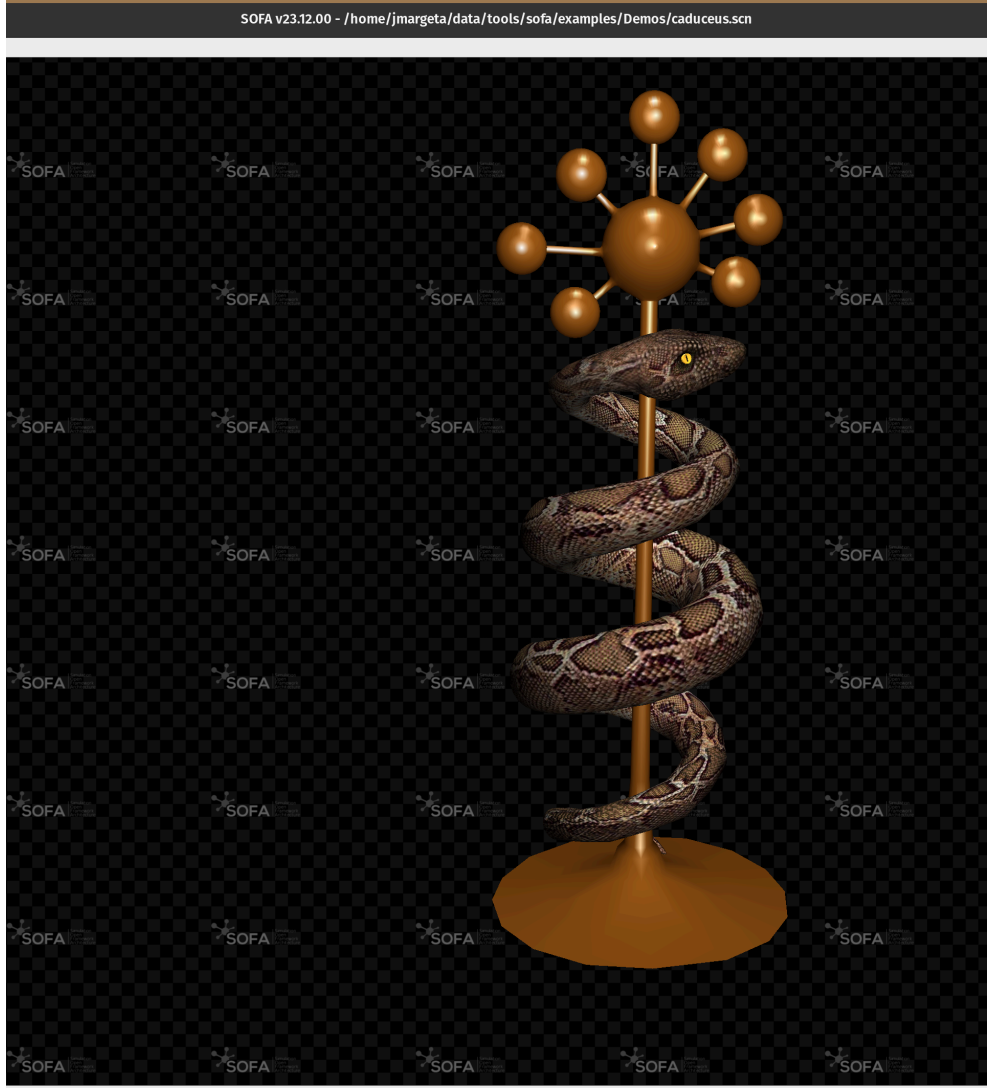
- MuJoCo - dynamics, soft contacts, soft objects, getting more multipurpose
- PyBullet - game engine, soft and rigid body dynamics

Soft robots

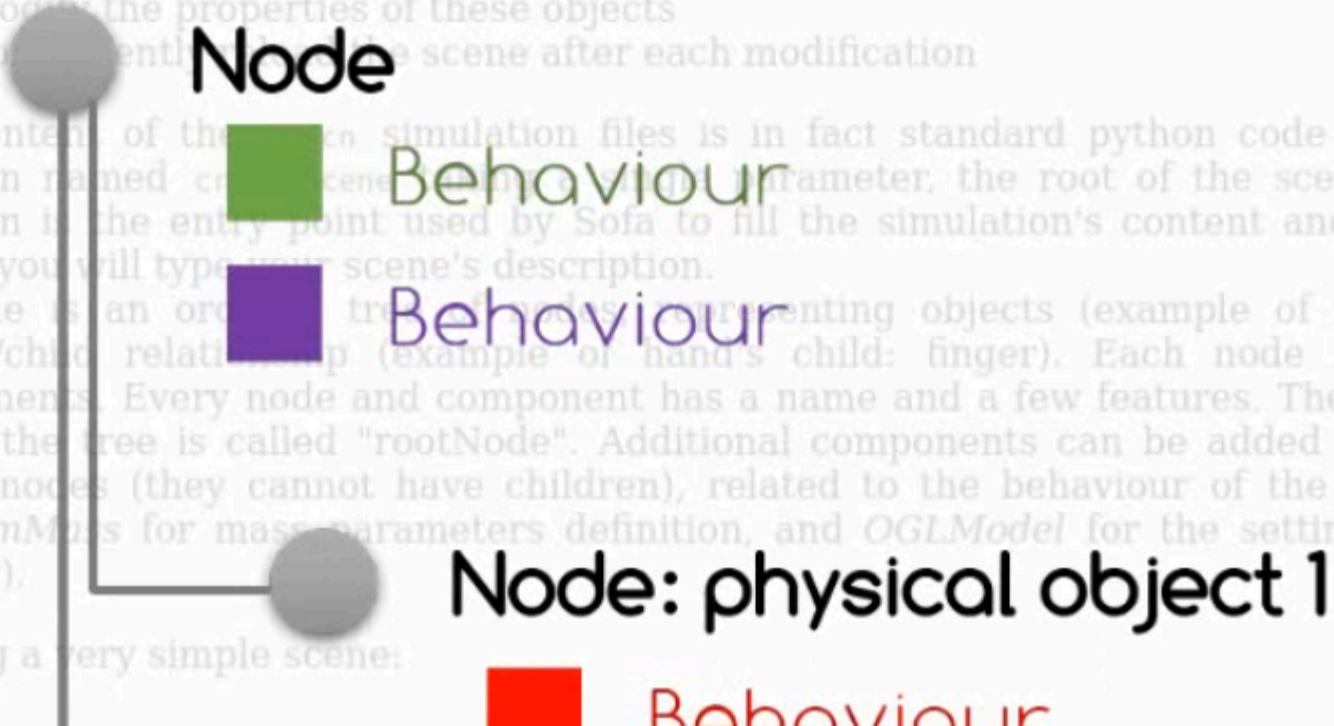
- SoftRobots - soft-robots and inverse kinematics (SOFA plugin)

Modelling with SOFA Framework

- Open source built by Inria in France
- Multiphysics
- Extensive plugin system
- Simulation through a visual tool
- XML and Python models
- Has a snake as the main example



Step 1: Setting up a simple scene



Define the scene graph in an XML file

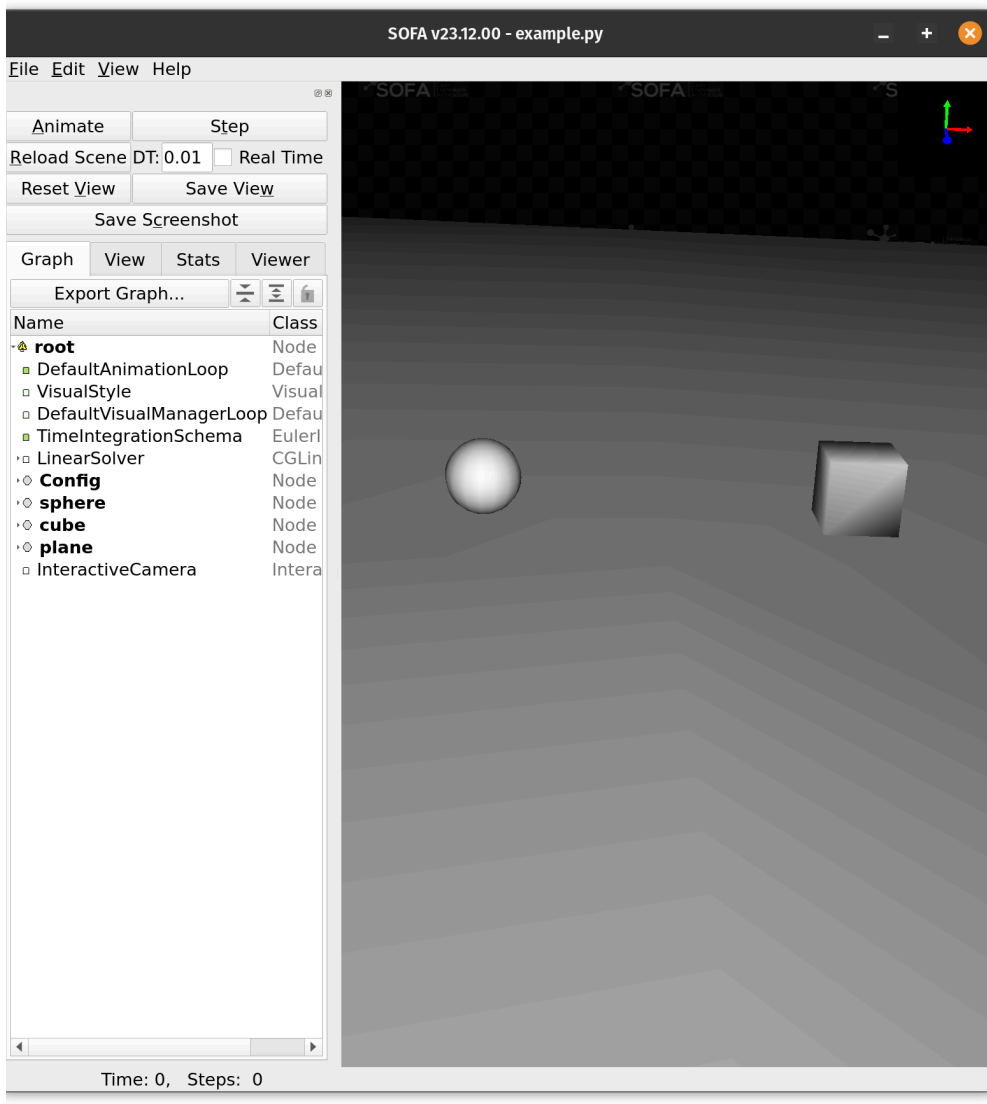
- Root node and animation loop (timestep, gravity)
- State vector (degrees of freedom)
- Solver (solve the mathematics at each step)
  - integration scheme (Euler, Runge Kutta, …)
  - how to solve (conjugate gradient, …)
- Physics (soft body mechanics, thermodynamics, fluid dynamics)

```
<Node name="root" dt="0.01" gravity="0 -9.81 0">
  <DefaultAnimationLoop />

  <EulerImplicitSolver rayleighStiffness="0.01"/>
  <CGLinearSolver iterations="100" tolerance="1e-06" />

  <MeshSTLLoader name="io" filename="liver.stl"/>

  <MechanicalObject name="dof" template="Vec3d" src="@i

  <DiagonalMass densitymass="1.0"/>
  <TetrahedronFEMForceField
    name="field"
    youngModulus="5000"
    poissonRatio="0.45"
  />
</Node>
```

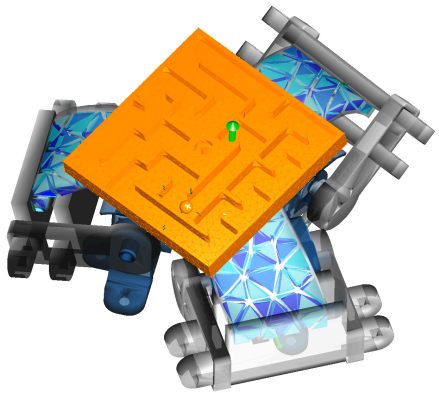
Define SOFA scene graphs in Python

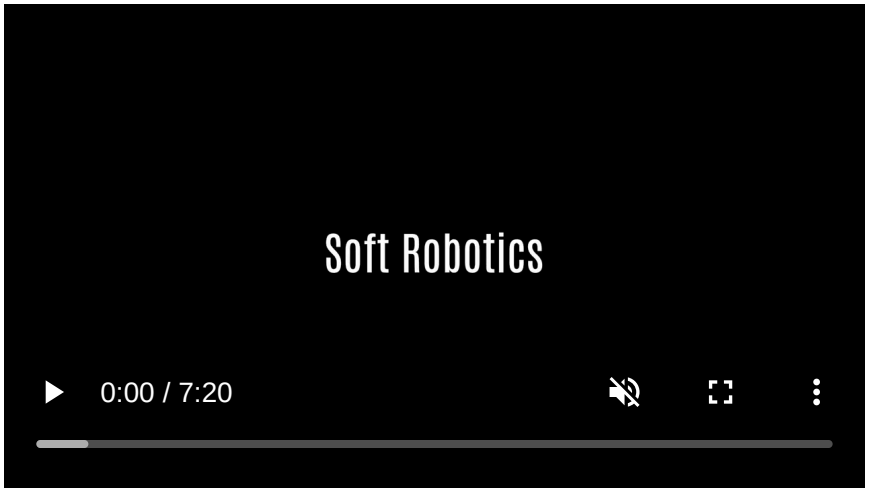
- SofaPython3 - low level API
- SOFA template library
  - prefabricated elements (floor, cube, ellastic objects, collision pipeline,...)

```python
from stlib3.scene import MainHeader
from stlib3.solver import DefaultSolver
from stlib3.physics.rigid import (
    Cube, Sphere, Floor
)

def createScene(root):
    MainHeader(root)
    DefaultSolver(root)

    Sphere(root,name="sphere",translation=[-5,0,0])
    Cube(root,name="cube",translation=[5,0,0])
    Floor(root,name="plane",translation=[0,-1,0])
```



Soft robots with SoftRobots and SoftRobots.Inverse





See the tutorial of Adriana Cabrera and build your own soft robots

Start from pre-existing simulations in SoftRobotics

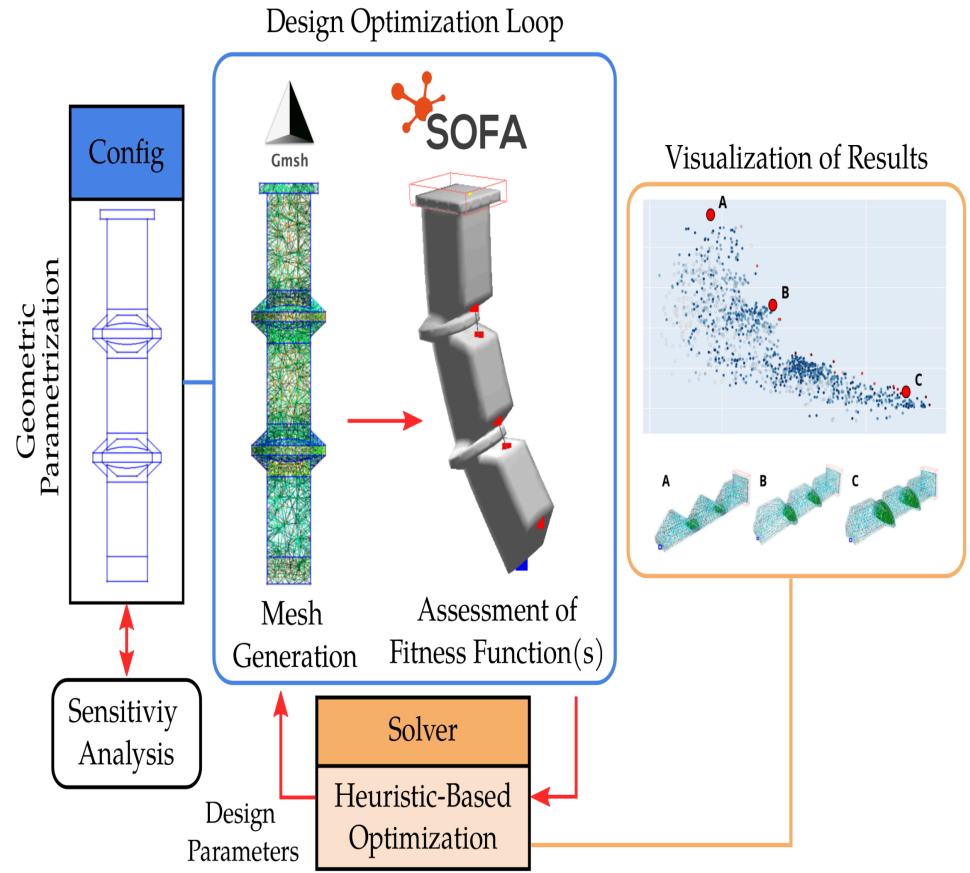
Design optimization

Explore parametric design of any SOFA scenes with Softrobots.DesignOptimization

With Optuna under the hood

```python
import optuna

def objective(trial):
    x = trial.suggest_float('x', -10, 10)
    y = trial.suggest_float('y', -10, 10)
    return (x - 2) ** 2 + (y - 4) ** 2

study = optuna.create_study()
study.optimize(objective, n_trials=100)
```

Alternative: Google vizier



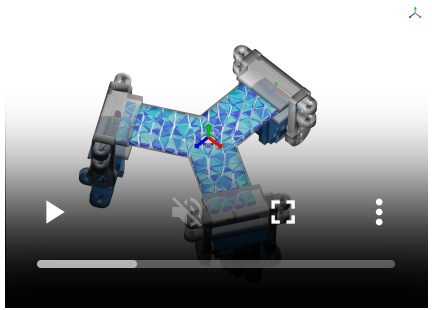
Vizier as another flexible black box optimization

```
def evaluate(w: float, x: int, y: float, z: str) -> float:
  return w**2 - y**2 + x * ord(z)

# Algorithm, search space, and metrics
study = vz.StudyConfig(algorithm='DEFAULT')
study.search_space.root.add_float_param('w', 0.0, 5.0)
study.search_space.root.add_int_param('x', -2, 2)
study.search_space.root.add_discrete_param('y', [0.3, 7.2])
study.search_space.root.add_categorical_param('z', ['a', 'g', 'k'])
study.metric_information.append(vz.MetricInformation('metric_name', goal=vz.ObjectiveMetricGoal.MAXIMIZE))

# Setup client and begin optimization. Vizier Service will be implicitly created.
study = clients.Study.from_study(study, owner='my_name', study_id='example')
for i in range(10):
  suggestions = study.suggest(count=2)
  for suggestion in suggestions:
    params = suggestion.parameters
    objective = evaluate(params['w'], params['x'], params['y'], params['z'])
    suggestion.complete(vz.Measurement({'metric_name': objective}))
```

SOFA Framework

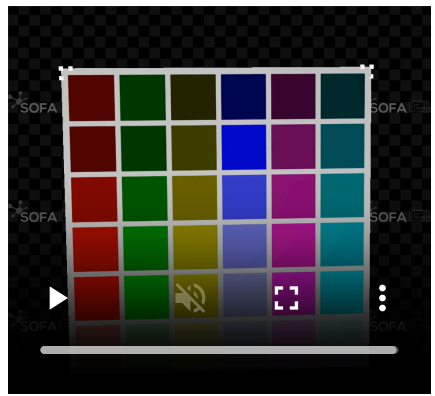
Likely the most complete open-source physics

Soft robot control

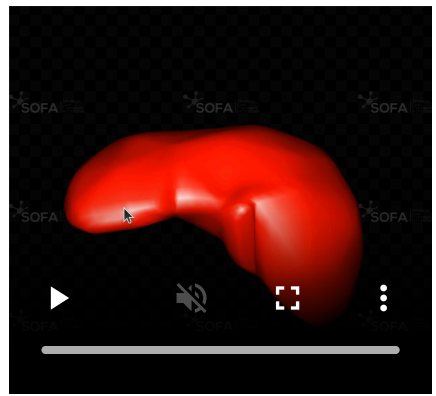


Diffusion process

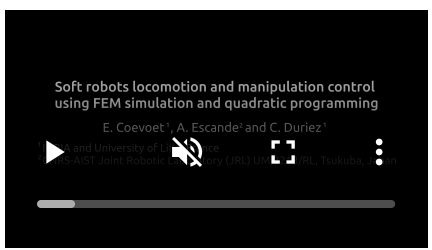
Cloth simulation



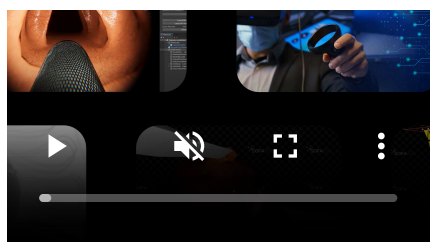
Liver model



Inria DEFROST reel



InfiniTech3D reel



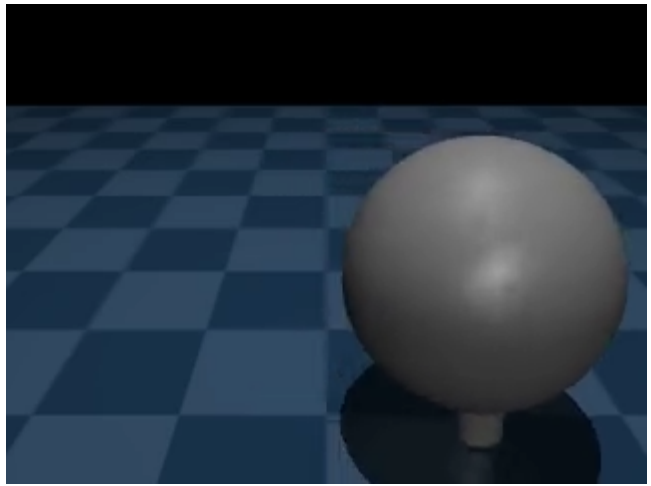
Université de Lille

Inria

MuJoCo

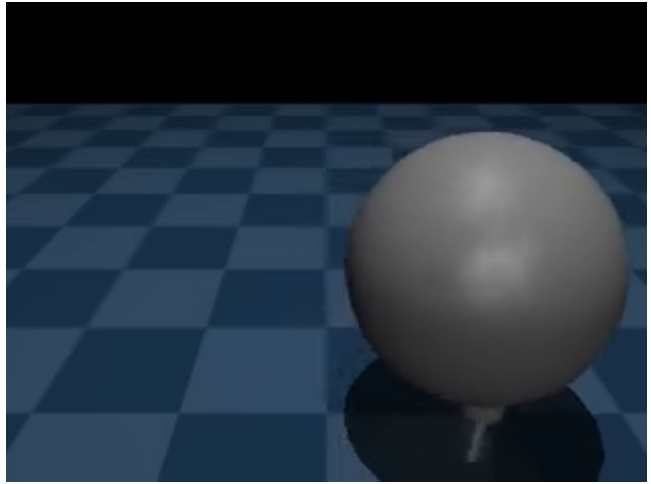
DeepMind acquired MuJoCo from Roboti LLC until 2021 and made it open-source.

- Simulation in generalized coordinates
- Avoiding joint violations
- Well-defined inverse dynamics even in the presence of contacts
- Constraints include soft contacts, limits, dry friction, equality constraints



Construct a kinematic tree

- Coordinate frames (positions and orientations)
- Bodies, deformables, rope, cloth
- Actuators, joints, tendons
- Cameras, lightning
- Solver



Run the simulation

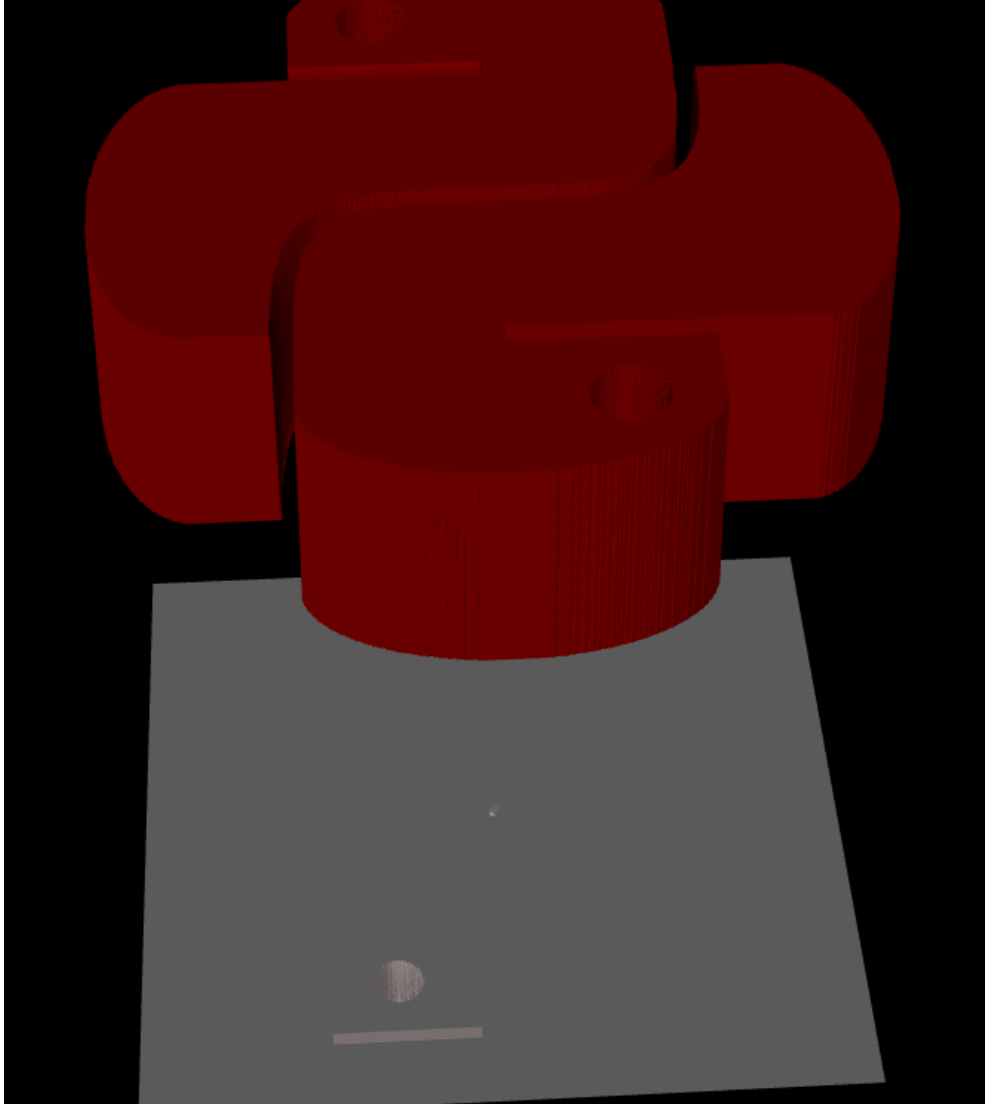
```
simulate tippe-top.xml
```

How a simple model looks like

```
<?xml-model href="xml-schema-mjcf/mujoco.xsd"?>
<mujoco>
  <asset>
    <mesh name="python" file="python.stl"/>
    <material name="logo" reflectance=".5" rgba="1 (
    <material name="floor" reflectance=".1"/>
  </asset>
  <worldbody>
    <light pos="-.2 -.4 .3" dir="1 2 -1.5" diffuse="
    <geom type="plane" pos="0 0 0" size="50 50 .01"
    <body>
      <freejoint/>
      <geom type="mesh" pos="0 0 100" name="python"
    </body>
  </worldbody>
</mujoco>
```

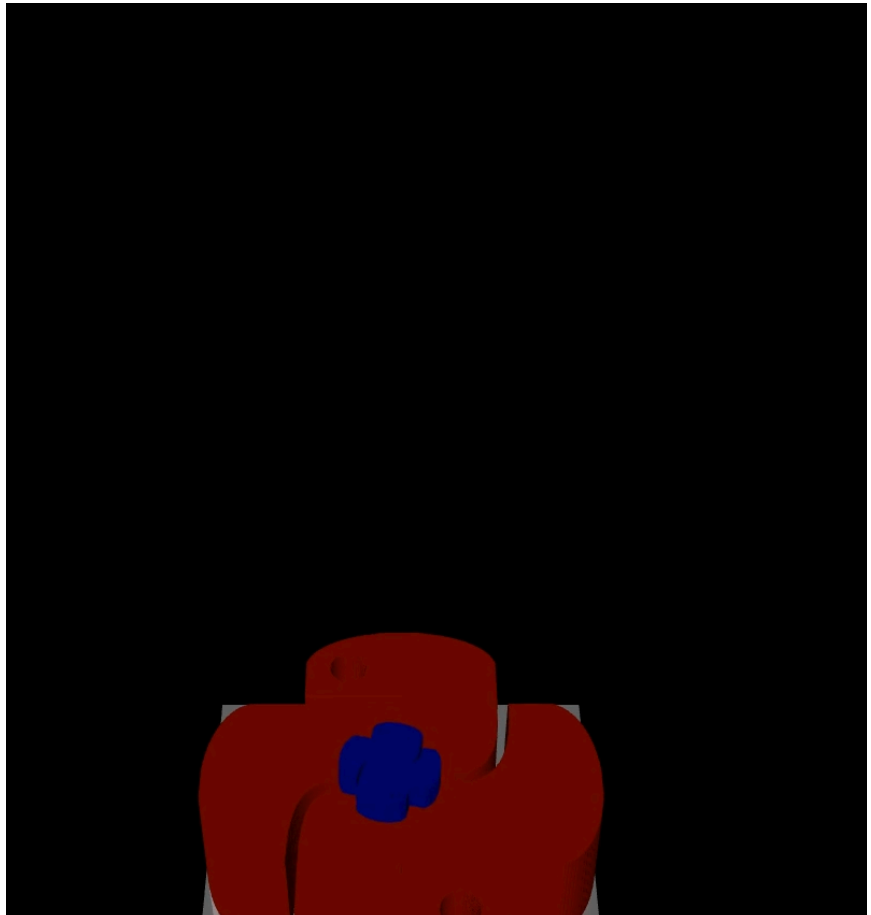
View the scene

python -m mujoco.viewer --mjcf=scene.xml



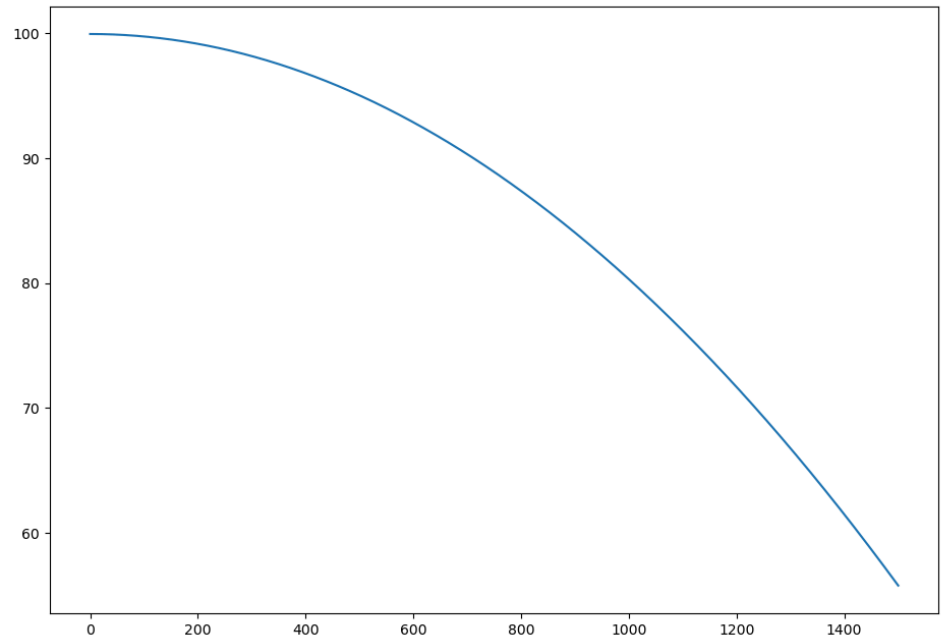
Load the scene in Python

```
import mujoco as mj
model = mj.MjModel.from_xml_string("""
<mujoco>
  <asset>
    <mesh name="python" file="python.stl"/>
    <mesh name="python_small" scale="0.25 0.25 0.25" file="
    <material name="logo" reflectance=".5" rgba="1 0 0 1"/>
    <material name="floor" reflectance=".1"/>
  </asset>
  <worldbody>
    <light pos="-.2 -.4 .3" dir="1 2 -1.5" diffuse=".6 .6 .
    <geom type="plane" pos="0 0 0" size="50 50 .01" rgba="1
    <body>
      <freejoint/>
      <geom type="mesh" pos="0 0 100" name="python" mesh="p
    </body>
    <body>
      <freejoint/>
      <geom type="mesh" pos="0 0 200" name="python_small" m
    </body>
  </worldbody>
</mujoco>
""")
data = mj.MjData(model)
```



Run the simulation loop and grab measurements

```
# Example measurements
measurements = []

# Run 3 seconds of the simulation
while data.time < 3:
  # Do a step in the simulation
  mj.mj_step(model, data)

  # Get the node and the measurements
  measurements.append(data.geom('python').xpos.copy(

import numpy as np
plt.plot(np.array(measurements[:, 2]))
plt.show()
```

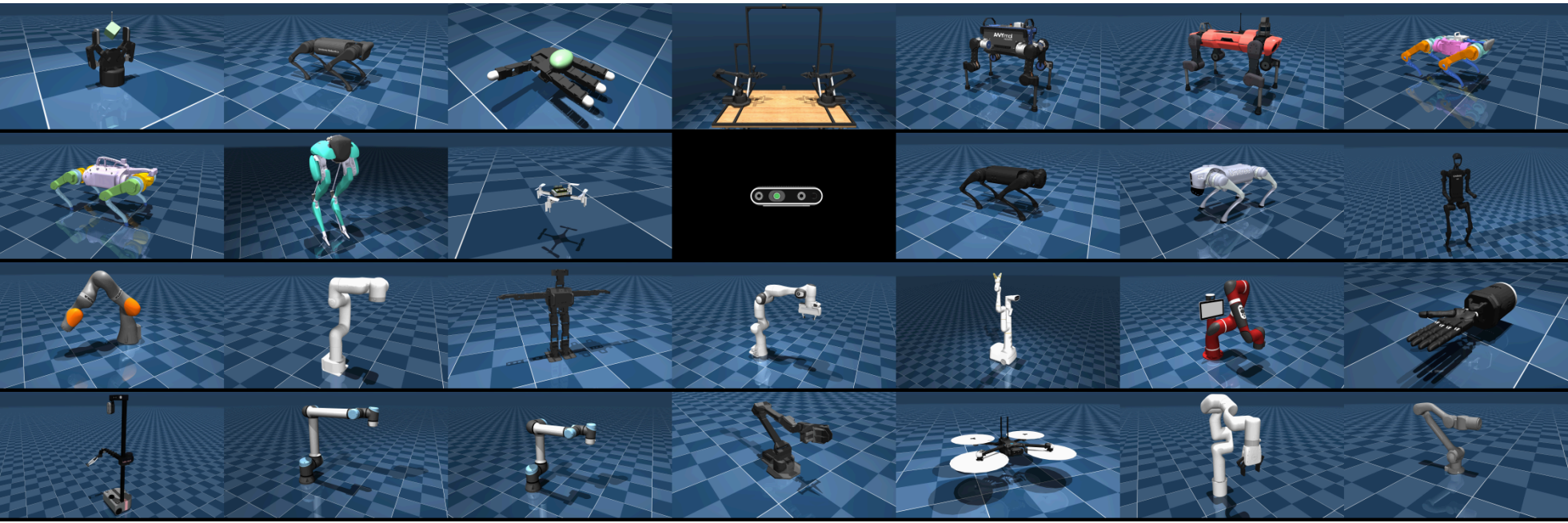


Defining MuJoCo models in Python

dm_control: Google DeepMind Infrastructure for Physics-Based Simulation

```
from dm_control import mjcf

model = mjcf.RootElement() # <mujoco>

# Adding new elements
my_box = model.worldbody.add('geom', name='my_box', type='box', pos=[0, .1, 0])
```

Mujoco menagerie - a collection of pre-built models



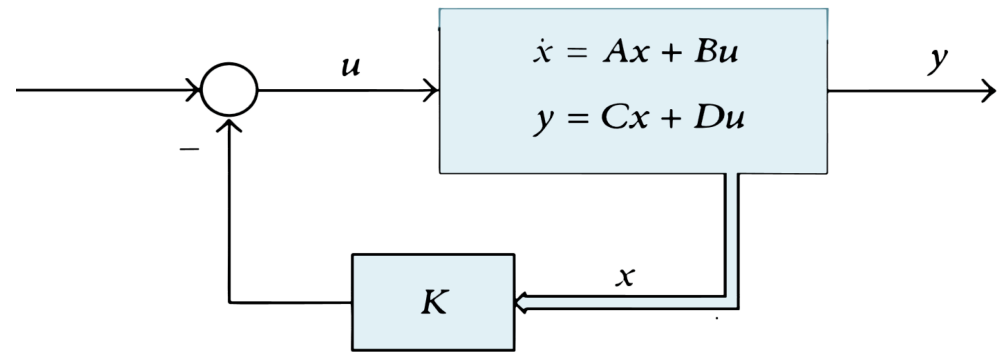
Part II: Control

🎮 Classical control theory?

State-transition matrix

$$x_{t+1} = Ax_t + By_t$$

- x_t - current state of the system
- y_t - sensor values
- u - control vector



What is feedback gain matrix K?

$$u = -Kx$$

Linear quadratic regulator

Optimal control over a time horizon

$$u = \underset{\pi}{\operatorname{argmin}} \sum_{t=0}^{\infty} J(x_t, \pi(x_t))$$

Compute A, B with finite differences

```
A, B = np.zeros(...), np.zeros(...)
mujoco.mjd_transitionFD(model, data, 1e-6, True, A, B, None, None)
```

Now compute the gain, K

=> Solve algebraic Riccatti equation (ARE)

Compute K - the feedback gain matrix

With scipy and numpy

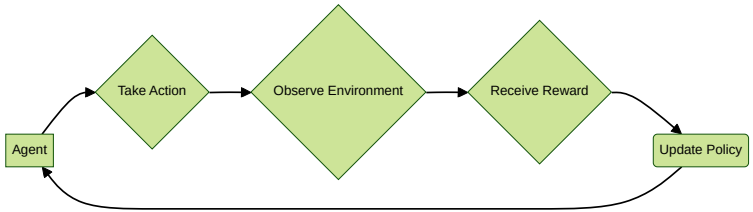
```
# Solve discrete Riccati equation.
P = scipy.linalg.solve_discrete_are(A, B, Q, R)

# Compute the feedback gain matrix K.
K = np.linalg.inv(R + B.T @ P @ B) @ B.T @ P @ A
```

With Python Control Systems Library

```
K, S, E = control.lqr(A, B, Q, R)
```

See LQR Tutorial in MuJoCoFor MPC, see mujoco_mpc

Reinforcement learning



Great tools

- Stable Baselines3 - reliable implementations of RL in PyTorch (aka scikit-learn for RL)
- CleanRL - Single-file implementations of RL algorithms

Name	Refactored [1]	Recurrent	`Box`	`Discrete`
A2C	✓	✓	✓	✓
ACER	✓	✓	✗ [4]	✓
ACKTR	✓	✓	✓	✓
DDPG	✓	✗	✓	✗
DQN	✓	✗	✗	✓
HER	✓	✗	✓	✓
GAIL [2]	✓	✓	✓	✓
PPO1	✓	✗	✓	✓
PPO2	✓	✓	✓	✓
SAC	✓	✗	✓	✗
TD3	✓	✗	✓	✗
TRPO	✓	✗	✓	✓

] Whether or not the algorithm has be refactored to fit the `BaseRLModel` class.

] Only implemented for TRPO.

] *(1, 2, 3, 4)* Multi Processing with MPI.

] TODO, in project scope.

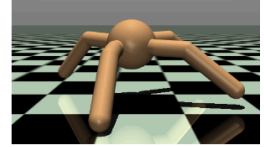
Reinforcement learning environment 👨‍🎓🤖

Unified structure Encapsulates arbitrary dynamics behind the step() and reset() functions.

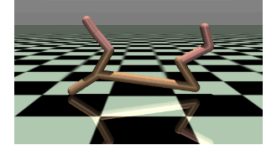
```
class Env:
    # Run one timestep of the environment's dynamics
    def step(self, action):
      ...

    # Resets the environment to an initial state
    def reset(self, **kwargs):
       ...

    # Render the environment to help visualize
    def render(self):
      ...

    # Clean-up
    def close(self):
      ...
```

Environments

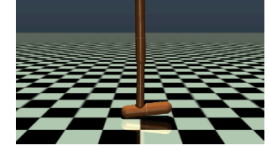
- Gymnasium - fork of OpenAI's Gym
- Gymnasium robotics
- SofaGym
- gym-chrono
- LocoMuJoCo - imitation learning benchmark for locomotion
- HumanoidGym
- Shimmy - conversion tool to Gymnasium and PettingZoo bindings



Ant



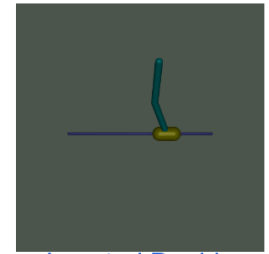
Half Cheetah



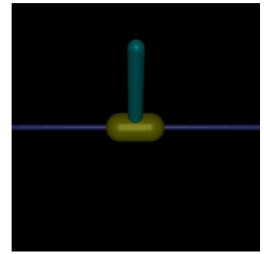
Hopper

Humanoid Standup

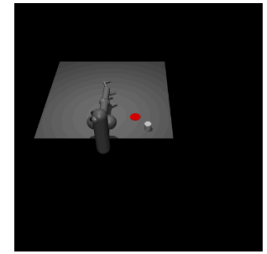
Humanoid



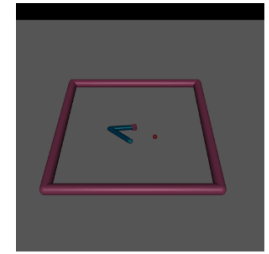
Inverted Double Pendulum



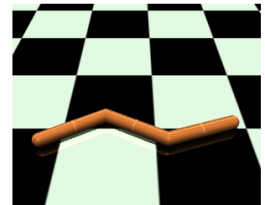
Inverted Pendulum

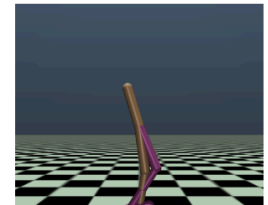


Pusher

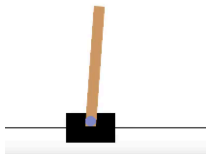


Reacher





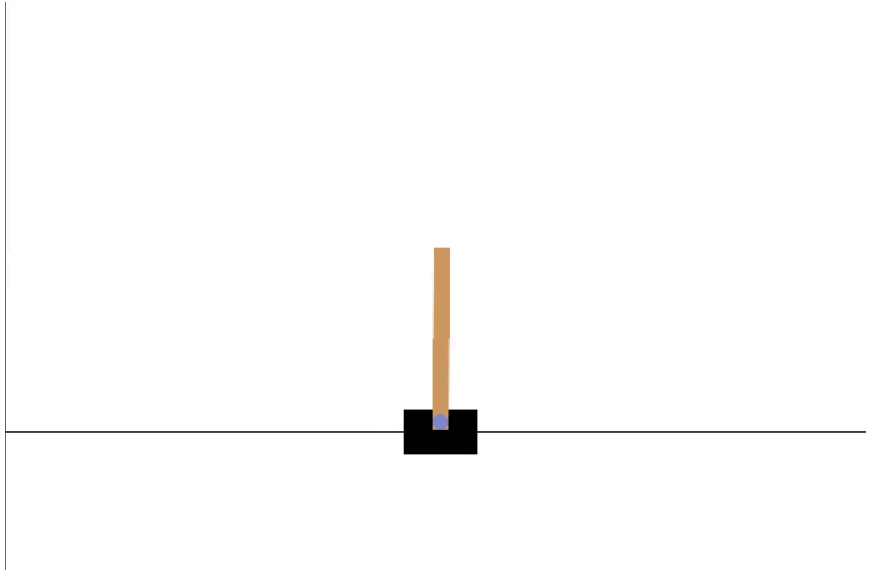
Training a Cart Pole controller



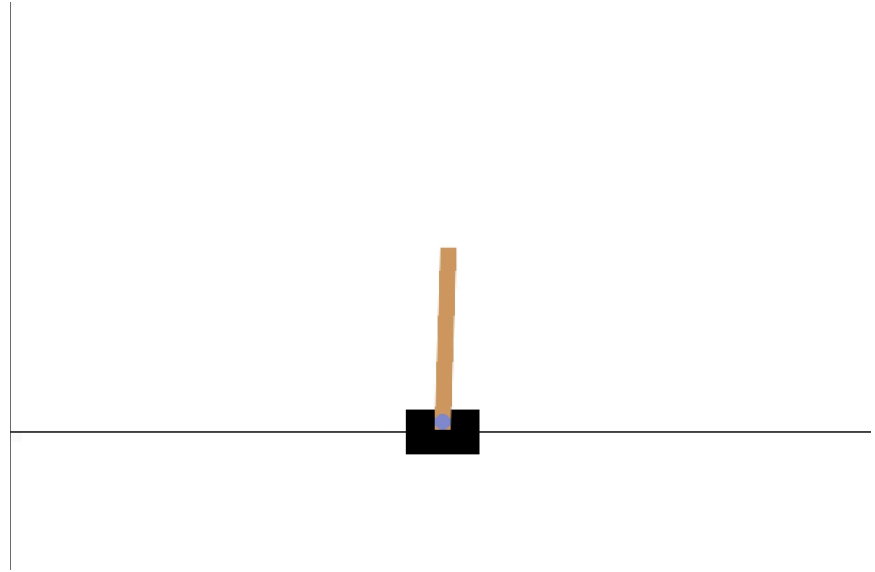
- Action space - Force applied on the cart [-1, 1] in Newton

- Observation space
  - Position and velocity of the cart
  - Angle and angular velocity of the pole
- Reward
  - Alive bonus (+10 per timestep)
  - Distance penalty (penalize travelling of the tip)

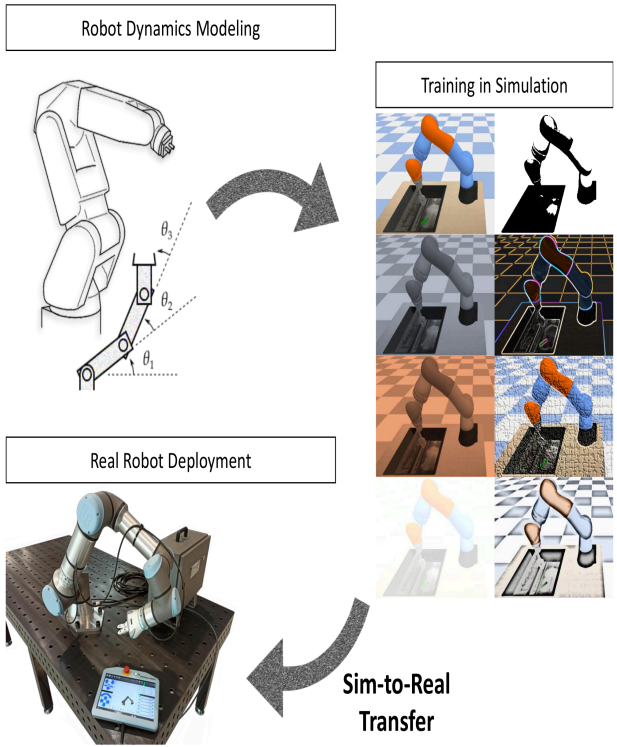
```
import gymnasium as gym
from stable_baselines3 import A2C

# create environment
env = gym.make("CartPole-v1", render_mode="rgb_array")

# RL model (Advantage Actor Critic)
model = A2C("MlpPolicy", env, verbose=1)

# train
model.learn(total_timesteps=50000)

# predict and render
vec_env = model.get_env()
obs = vec_env.reset()
for i in range(1000):
    action, _state = model.predict(obs, deterministic=T
    obs, reward, done, info = vec_env.step(action)
    vec_env.render("human")
```



Before training



After 15000 epochs

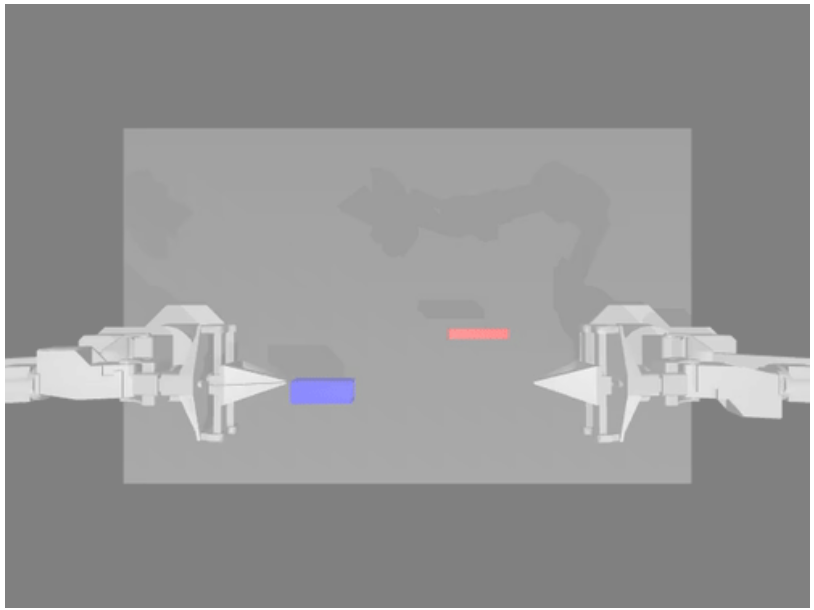


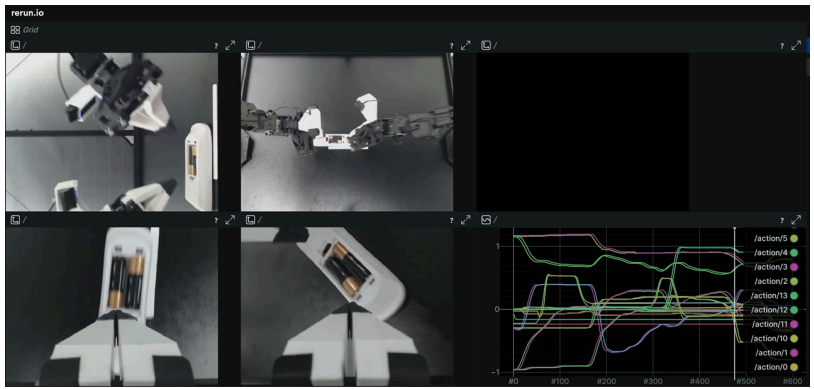
Closing the gap

- Transfer expertise from simulation to reality

Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: a Survey Wenshuai Zhao, Jorge Peña Queralta, Tomi Westerlund

LeRobot from 🤗

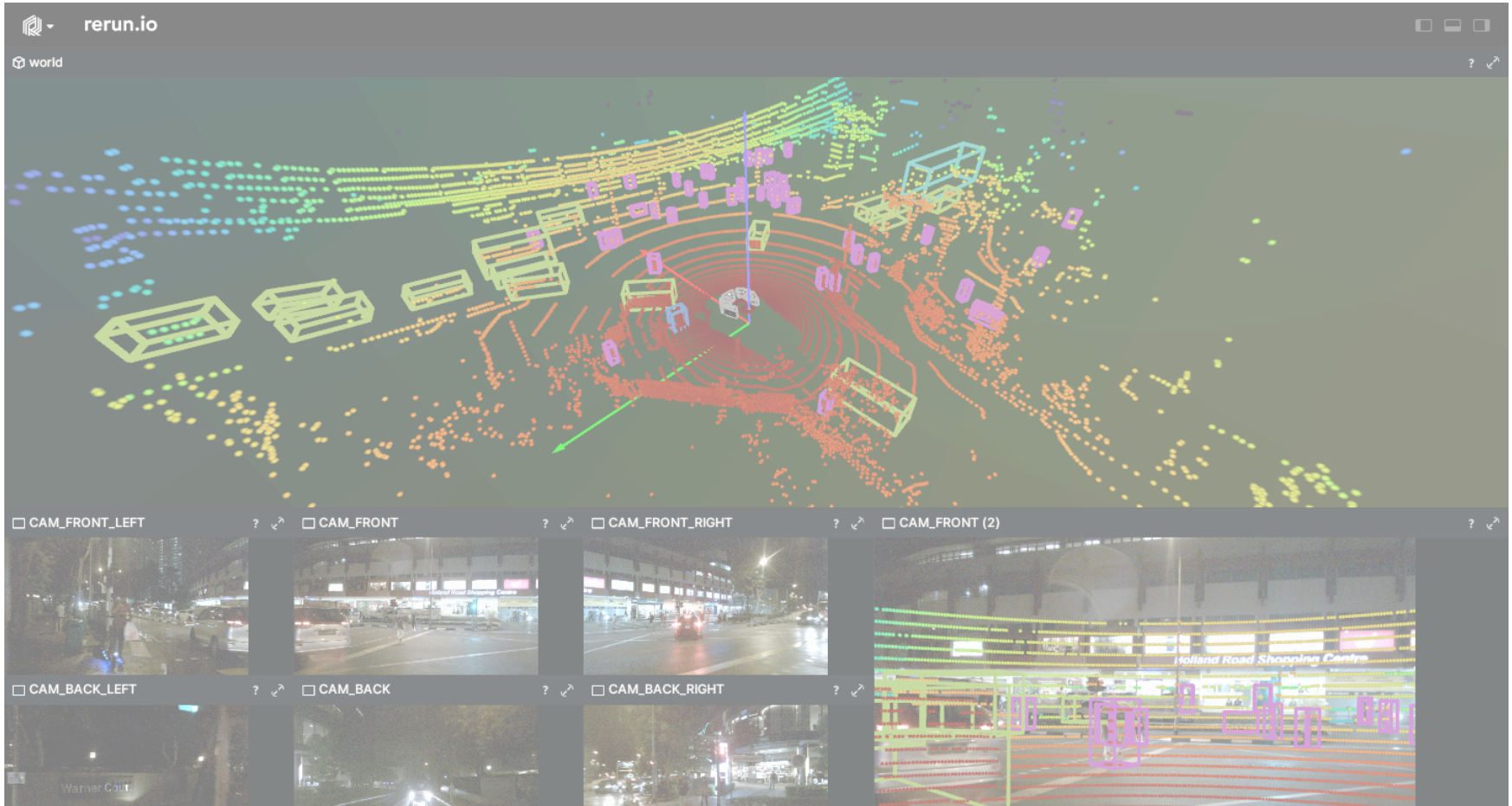




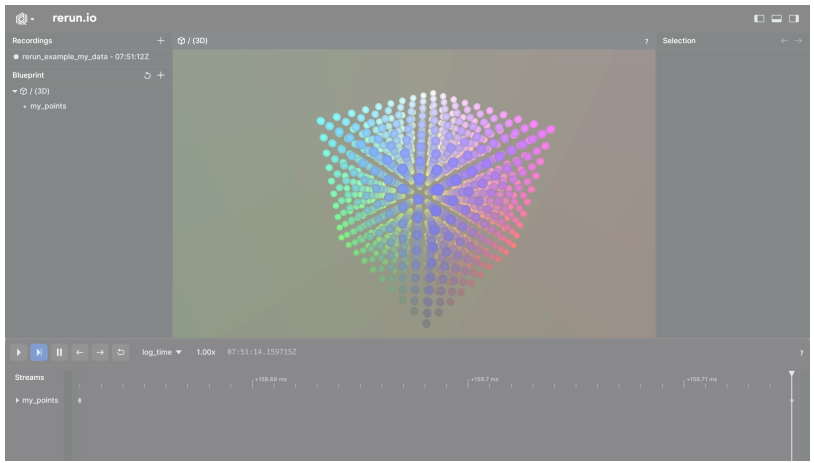

```
python lerobot/scripts/train.py \
    policy=act \
    env=aloha \
    env.task=AlohaInsertion-v0 \
    dataset_repo_id=lerobot/aloha_sim_insertion_human \
```

Part III: Replay

Rerun - let’s get Rusty

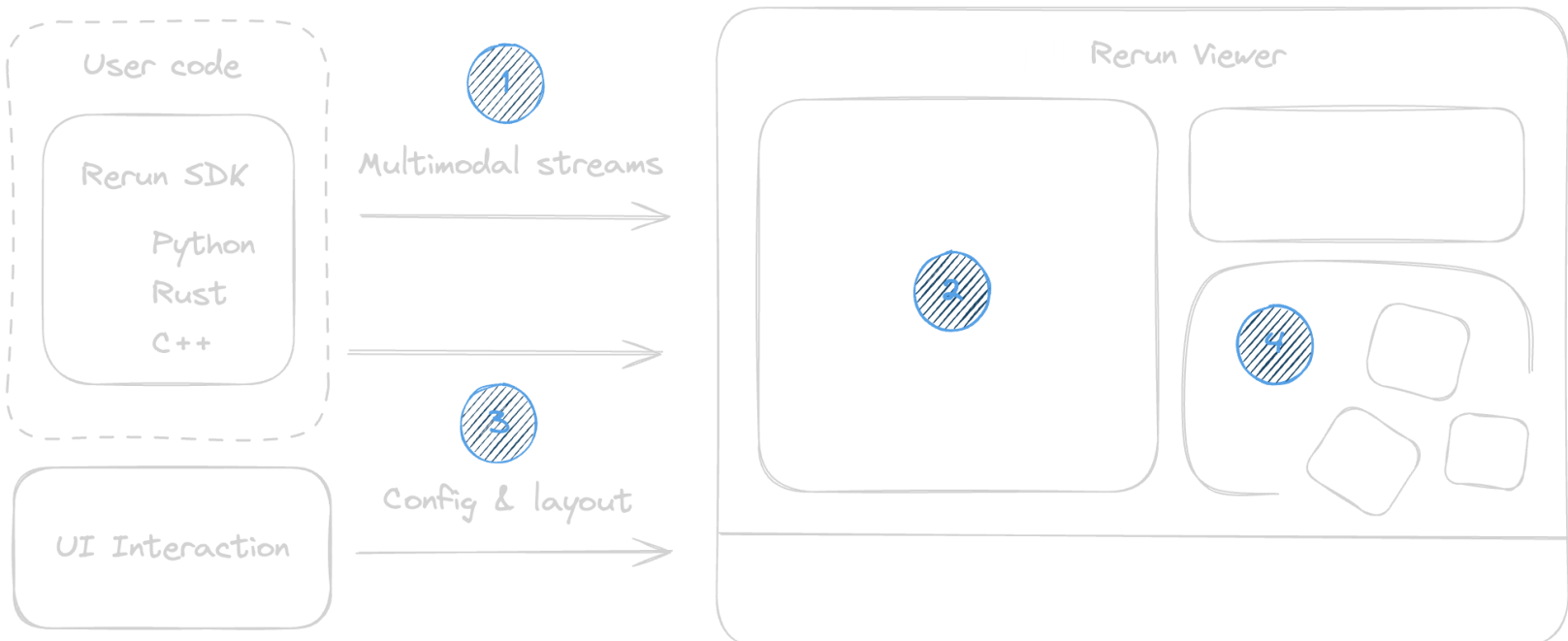


A better logging for your data apps



points 2D, points 3D, measurements, arrows, meshes, images, volumes, cameras, blueprints…

```python
import rerun as rr
import numpy as np

rr.init("rerun_example_my_data", spawn=True)

SIZE = 10

rr.set_time_seconds("stable_time", time)

pos_grid = np.meshgrid(*[np.linspace(-10, 10, SIZE)]*3)
positions = np.vstack([d.reshape(-1) for d in pos_grid]

col_grid = np.meshgrid(*[np.linspace(0, 255, SIZE)]*3)
colors = np.vstack([c.reshape(-1) for c in col_grid]).a

rr.log(
    "my_points",
    rr.Points3D(positions, colors=colors, radii=0.5)
)
```


User code
Rerun SDK
Python
Rust
C++
UI Interaction
1
Multimodal streams
3
Config & layout
Rerun Viewer
2
4

Extra Tools

- Robotics toolbox
- ROS - robot operating system
- Comparison of Popular Simulation Environments in the Scope of Robotics and RL
- MoCapAct - A Multi-Task Dataset for Simulated Humanoid Control
- Kubric: a scalable dataset generator
- MushroomRL: Reinforcement Learning Python library.

Is this a real life or is this some...

Amazing ecosystem of tools

- Simulate (Sofa, MuJoCo)
- Control (Python Control, dm_control, LeRobot)
- Replay (rerun)

Make your own rules and start breaking things with no consequences

Reach out to jan@kardio.me or see more on 𝕏 @jmargeta