cs391R - Physical Simulation Environment Tutorial

Yifeng Zhu

Department of Computer Science
The University of Texas at Austin

September 28, 2020

The University of Texas at Austin
Computer Science

Overview

1. Pybullet - Robovat (Fang, Zhu, Garg, Savarese, et al., 2019)
   RPL robovat: **[Link]**
   Original version - Stanford robovat: **[Link]**
2. Mujoco - Robosuite (Zhu et al., 2020) **[Link]**

Pybullet vs Mujoco?

Comparison between Mujoco and Pybullet (Erez, Tassa, and Todorov, 2015)

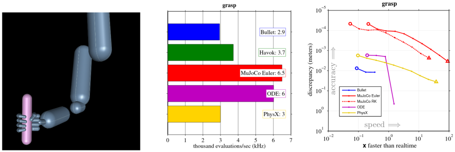

Figure: Grasping

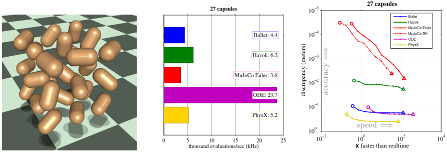

Figure: Number of bodies

Robovat vs Robosuite?

Robovat:

- Designed for grasping, manipulation research, better support of 3D objects.
- Slower than Mujoco, Fewer options of controllers.

Robosuite:

- Designed for Reinforcement Learning / Imitation Learning.
- Efficient simulation of objects with simple geometry.
- Easier to create procedurally generated scene.

Past research using robovat

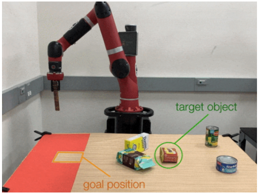

Figure: Hierarchical Planning (Fang, Zhu, Garg, Savarese, et al., 2019)

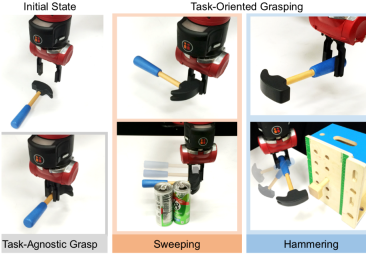

Figure: Grasping (Fang, Zhu, Garg, Kuryenkov, et al., 2018)

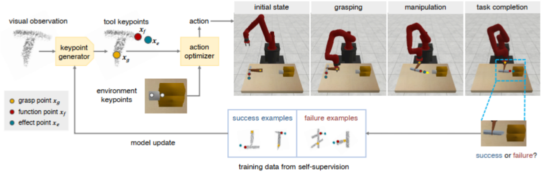

Figure: Grasping (Qin et al., 2019)

Past research using robosuite

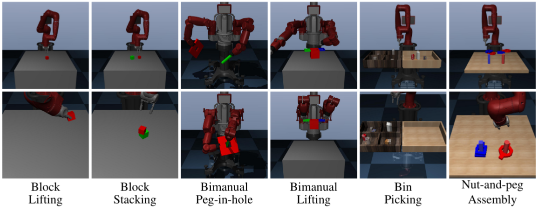

Figure: Reinforcement Learning (Fan et al., 2018)

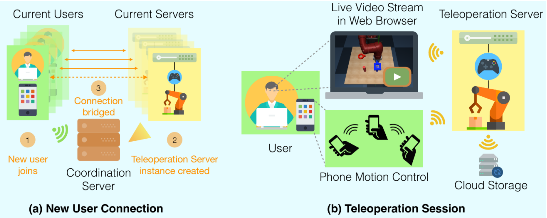

Figure: Teleoperation for data collection (Mandlekar et al., 2018)

Simulator

- Pros
  - Cost-efficient
  - Easy to prototype robot experiments
- Cons
  - Not perfect
  - Lots of artifacts
- How constraint is implemented?
  Impulse acts as a constraint.

Description File

- Pybullet - URDF file (Universal Robot Description File) http://wiki.ros.org/urdf/Tutorials
- Mujoco - MJCF file http://www.mujoco.org/book/XMLreference.html

Both of them are xml data files. Any parameters regarding robots / objects can be defined in URDF/MJCF files.

Pybullet

Documentation: [Link]
Pybullet is a python wrapper of Bullet physics.

Robovat

Robots:

1. Panda
2. Sawyer

Grippers:

1. Panda Gripper
2. Rethink Gripper

Controllers:

1. Joint Position Control
2. Joint Velocity Control
3. Joint Torque Control
4. Inverse Kinematics

A lot of object meshes (including YCB)

Robovat - Create Environment I

Create Simulator

```
simulator = Simulator(worker_id=args.worker_id,
                      use_visualizer=bool(args.debug),
                      assets_dir=args.assets_dir)
```

Parse configs for env and policy

```
env_config, policy_config =
             parse_config_files_and_bindings(args)
```

Robovat - Create Environment II

Create Environment

```
env = eval(env_name)(simulator=simulator,
                     config=env_config,
                     debug=args.debug)
obs = env.reset()
```

Plot visual observation

```
plt.imshow(np.squeeze(obs[env.config.OBSERVATION.TYPE]))
plt.show()
```

Robovat - Create Environment III

Move the gripper

```
# You need to deep copy pose objects
target_end_effector_pose = env.robot.end_effector.pose.copy()
target_end_effector_pose.x = 0.5
target_end_effector_pose.y = 0.0

# look at robovat/envs/franka_panda_grasp_env.py
env.execute_moving_action(target_end_effector_pose)
```

Robovat - Basic Functionality I

Add body from URDF

```
simulator.add_body(URDF_FILE_PATH,
                   pose, scale=scale, name=OBJECT_NAME)
```

Add body from obj file

```
simulator.add_body(OBJ_FILE_PATH,
                   pose, scale=scale, name=OBJECT_NAME,
                   collisionFrameOrientation=[0, 0, 0, 1],
                   visualFrameOrientation=[0, 0, 0, 1],
                   baseMass=0.1)
```

Change dynamics of an object

```
object.set_dynamics(lateral_friction=1.0,
                    contact_damping=1.0)
```

Robovat - Basic Functionality II

Control a robot (Franka example)

```
robot.move_to_joint_positions(position_sequence)
robot.move_to_gripper_pose(target_gripper_pose)
robot.move_along_gripper_path(gripper_pose_array)
robot.grip(grip_pos) # grip_pos \in [0, 1]
robot.stop_l_finger() # stop left finger
robot.stop_r_finger() # stop right finger
```

Robovat - Create an environment class

For more details on how to create an environment class, look at the example `robovat/envs/franka_panda_grasp_env.py`.

Robovat - Contact-rich tasks

By contact-rich, we mean motions that involve interaction with objects instead of purely robot arm movements. Contact-rich motions are not perfect in simulation, so we need to keep this fact in mind all the time. As a consequence, you need to tune some physical parameters in simulation in order to obtain a more realistic execution.
There are several important coefficients that you need to tune:

1. contact damping and contact stiffness
2. lateral friction
3. spinning friciton
4. rolling friction

Robovat - Config files

Config files in robovat are Yaml files. For franka panda robots, look at `configs/robots/franka_panda.yaml`. You can specify which URDF file to be loaded in `ARM_URDF` argument.

For environment configurations, please look at `configs/envs/franka_panda_envs/franka_panda_grasp_env.yaml`

Robovat Example - Grasping I

Before grasping - change parameters

```
robot.l_finger_tip.set_dynamics(
  lateral_friction=0.001,
  spinning_friction=0.001)
robot.r_finger_tip.set_dynamics(
  lateral_friction=0.001,
  spinning_friction=0.001)
table.set_dynamics(
  lateral_friction=100)
```

Robovat Example - Grasping II

After releasing - change parameters back

```
robot.l_finger_tip.set_dynamics(
  lateral_friction=100,
  rolling_friction=10,
  spinning_friction=10)
robot.r_finger_tip.set_dynamics(
  lateral_friction=100,
  rolling_friction=10,
  spinning_friction=10)
table.set_dynamics(
  lateral_friction=1)
```

Robovat Example - Grasping

Suppose we have object's pose information in a variable `object_pose`, if we want to move two tips of the gripper symmetrically, run the command:

```
env.execute_grasping_action(object_pose)
```

Or if you want to move fingers of a gripper asymmetrically (To create more stable grasp in simulation), run the command:

```
env.execute_gentle_grasping_action(object_pose)
```

Robovat Example - Pushing

Before pushing - change parameters

```
table.set_dynamics(
  lateral_friction=0.1)
```

After pushing - change parameters back

```
table.set_dynamics(
  lateral_friction=1.0)
```

And for more details of executing pushing, please look at the function `_execute_action` in `robovat/envs/franka_panda_push_env.py`.

Mujoco

Documentation: [Link]
What robosuite uses is the python wrapper of Mujoco.

Robosuite

Robots:

1. Panda
2. Jaco
3. Kinova3
4. IIWA
5. UR5e
6. Sawyer
7. Baxter

Grippers:

1. Panda Gripper
2. Jaco ThreeFinger
3. Wiping Gripper
4. Robotiq85
5. Rethink Gripper
6. ...

Controllers:

1. Joint Position Control
2. Joint Velocity Control
3. Joint Torque Control
4. Operational Space Control
5. Operational Space Control (Position only)
6. Inverse Kinematics

Robosuite - Create a robot

Create the world

```
from robosuite.models import MujocoWorldBase
world = MujocoWorldBase()
```

Create a robot

```
from robosuite.models.robots import Panda
mujoco_robot = Panda()
```

Add a gripper

```
from robosuite.models.grippers import gripper_factory
gripper = gripper_factory('PandaGripper')
mujoco_robot.add_gripper(gripper)
```

Add the robot to the world

```
mujoco_robot.set_base_xpos([0, 0, 0])
world.merge(mujoco_robot)
```

Robosuite - Create a tabletop environment

Create a table

```
from robosuite.models.arenas import TableArena
mujoco_arena = TableArena()
mujoco_arena.set_origin([0.8, 0, 0])
world.merge(mujoco_arena)
```

Robosuite - Create a tabletop environment

Add an object which can move around (thus needs a free joint)

```
from robosuite.models.objects import BallObject
from robosuite.utils.mjcf_utils import new_joint

sphere = BallObject(
    name="sphere",
    size=[0.04],
    rgba=[0, 0.5, 0.5, 1]).get_collision()
sphere.append(new_joint(
     name='sphere_free_joint',
     type='free'))
sphere.set('pos', '1.0 0 1.0')
world.worldbody.append(sphere)
```

Robsuite - Run simulation

Obtain a **MjModel** instance which can be used for Mujoco Simulation.

```
model = world.get_model(mode="mujoco_py")

from mujoco_py import MjSim, MjViewer

sim = MjSim(model)
viewer = MjViewer(sim)

# disable visualization of collision mesh
viewer.vopt.geomgroup[0] = 0

for i in range(10000):
  sim.data.ctrl[:] = 0
  sim.step()
  viewer.render()
```

Robsuite - Miscellaneous I

Get link / joint

```
# Get Joint indices
jnt_idx = env.robots[0]._ref_joint_indexes
n = len(jnt_idx) # Number of dof
# Get a body's name
env.sim.model.body_id2name(idx)
# Set to zero pose
env.robots[0].set_robot_joint_positions(np.zeros(n))
```

Robsuite - Miscellaneous II

Get screw axis

Unit Screw axis: $\mathcal{S} = [\omega, v]$

```
w0 = np.array(env.sim.data.xaxis)[jnt_idx]
p0 = np.array(env.sim.data.xaxis)[jnt_idx]
v0 = -np.cross(w0, p0)
S = np.hstack([w0, v0])
```

Robsuite - Miscellaneous III

Copmute Spatical Jacobian (a.k.a. Fixed frame) Jacobian: $\mathcal{J} = \text{Ad}_T(\mathcal{S})$

```
J = []
Ts = np.eye(4)
for i in range(n):
   row = adjoint(Ts).dot(S[i])
   J.append(row)
   T = exp2mat(w0[i], v0[i], theta[i])
```

Twist of end-effector $\mathcal{V} = \mathcal{J}\dot{\theta}$

```
V = J.dot(theta_dot)
```

Robsuite - Miscellaneous IV

Apply torque (Gravity compensation)

```
joint_names = ["gripper_z_joint"]
indices = [sim.model.get_joint_qvel_addr(x)
    for x in joint_names]
sim.data.qfrc_applied[indices] = sim.data.qfrc_bias[
     indices]
```

Other simulation options

1. OpenAI Gym
2. AI2-THOR
3. RLBench
4. CARLA
5. AirSim
6. Interactive Gibson
7. AI Habitat
8. ...

For more details and links, please at the course webpage:
https://www.cs.utexas.edu/~yukez/cs391r_fall2020/project.html.

References I

Erez, Tom, Yuval Tassa, and Emanuel Todorov (2015). "Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx". In: *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE, pp. 4397–4404.

Fan, Linxi et al. (2018). "SURREAL: Open-Source Reinforcement Learning Framework and Robot Manipulation Benchmark". In: *Conference on Robot Learning*.

Fang, Kuan, Yuke Zhu, Animesh Garg, Andrey Kuryenkov, et al. (2018). "Learning Task-Oriented Grasping for Tool Manipulation from Simulated Self-Supervision". In: *Robotics: Science and Systems (RSS)*.

Fang, Kuan, Yuke Zhu, Animesh Garg, Silvio Savarese, et al. (2019). "Dynamics Learning with Cascaded Variational Inference for Multi-Step Manipulation". In: *Conference on Robot Learning (CoRL)*.

References II

Mandlekar, Ajay et al. (2018). "Roboturk: A crowdsourcing platform for robotic skill learning through imitation". In: *arXiv preprint arXiv:1811.02790.*

Qin, Zengyi et al. (2019). "Keto: Learning keypoint representations for tool manipulation". In: *arXiv preprint arXiv:1910.11977.*

Zhu, Yuke et al. (2020). "robosuite: A Modular Simulation Framework and Benchmark for Robot Learning". In: *arXiv preprint arXiv:2009.12293.*