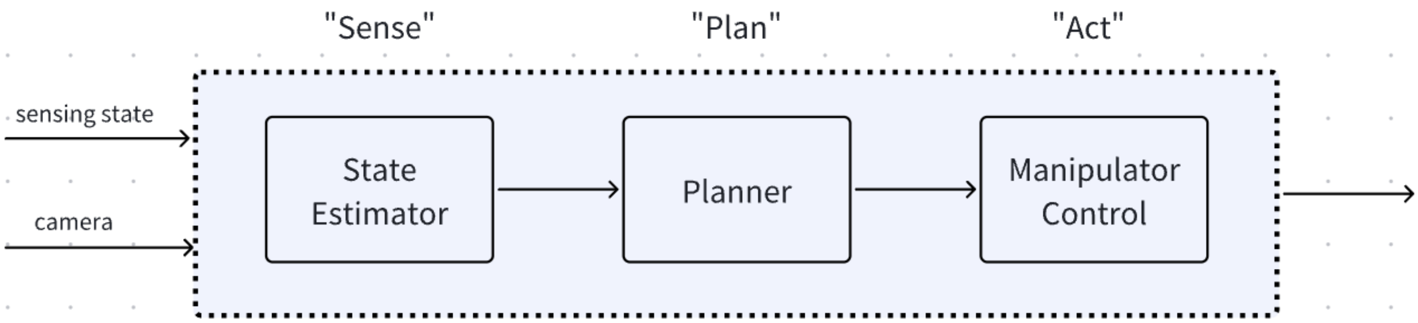
19. Reinforcement Learning 1

Key words

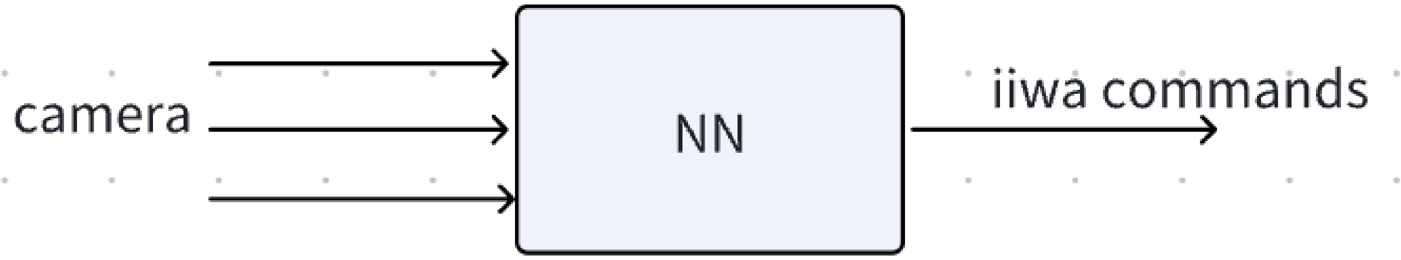
- Behavior cloning pipeline
- Foundation models
- Reinforcement Learning
    - model-free
    - model-based

Last Lecture, Visualmotor Policy

- How do we design controllers for visuomotor control?
- "model-based" control



- Behavior Cloning (Supervised sequence learning)



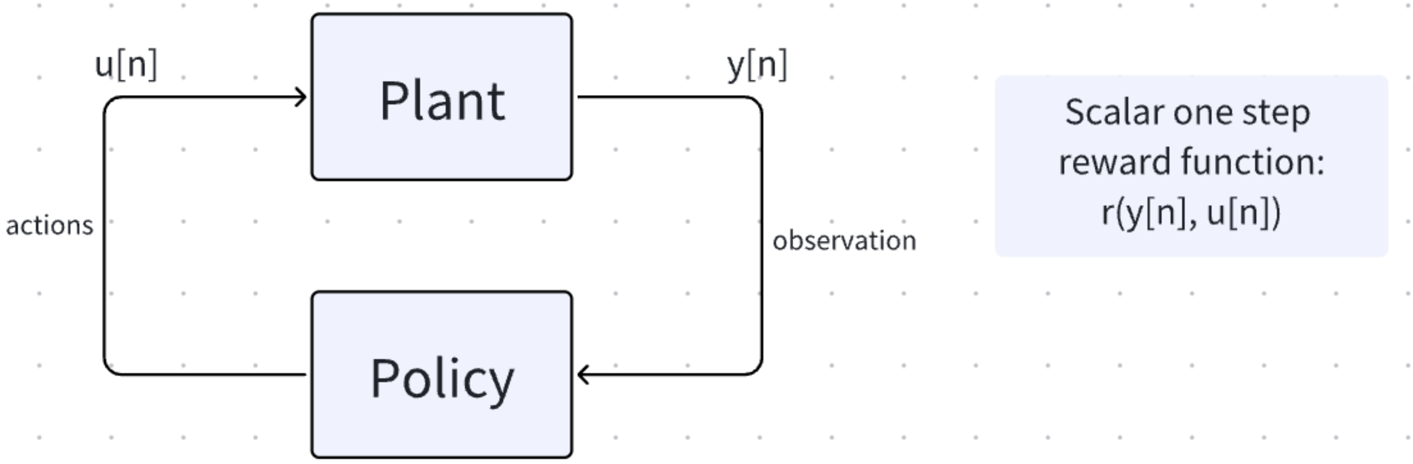
collect many demostrations (e.g. human teleoperation)

- Reinforcement learning lies in the middle, requiring less supervision, but a harder optimization problem

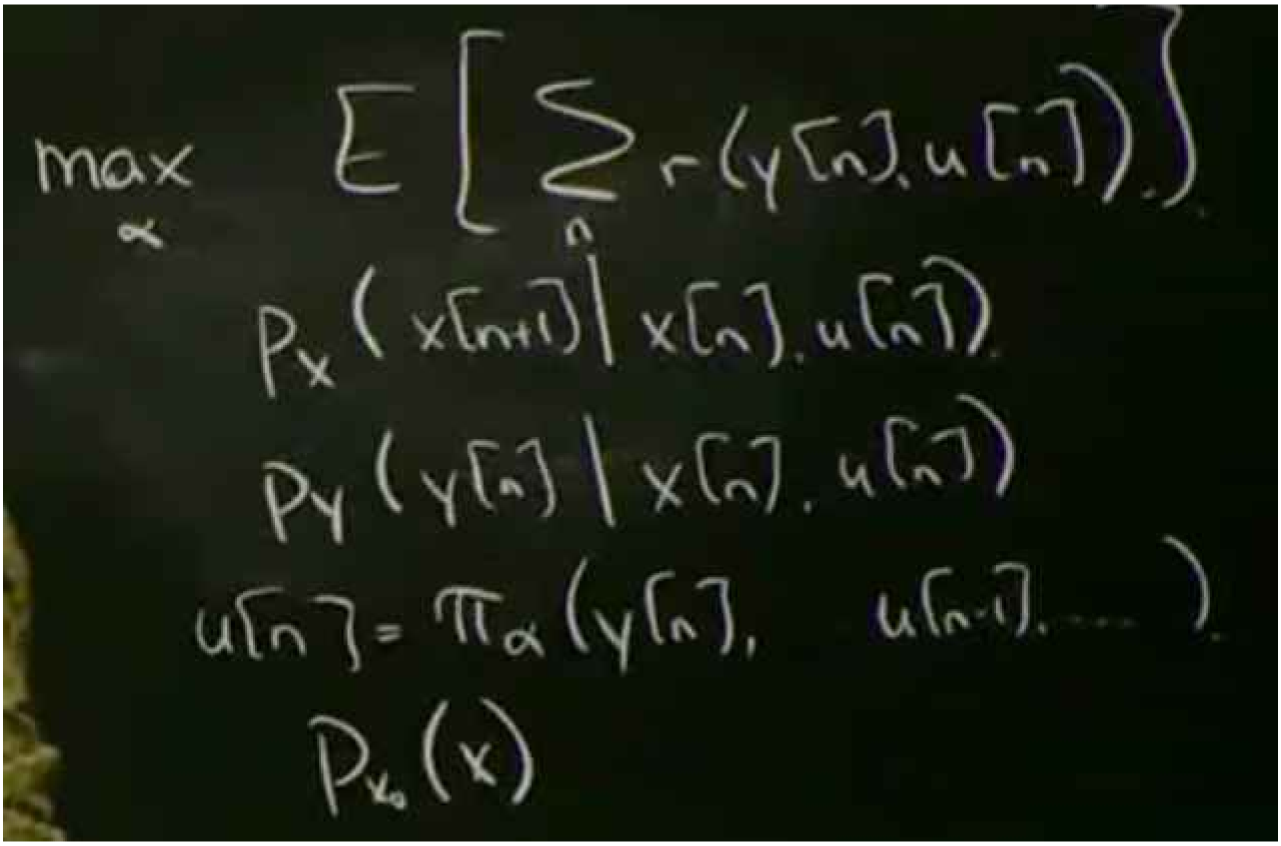
- These methods are more similar than they are different: people asked, should i use BC, RL or model-based, as a matter of fact, if u get it done using one, u can also make it using the other methods. There are a lot connections.
- $y[n] = \pi_\alpha(y[n], u[n-1], y[n-1], \ldots)$, α : NN parameters
-

Today's Topic outline

- Reinforcement Learing



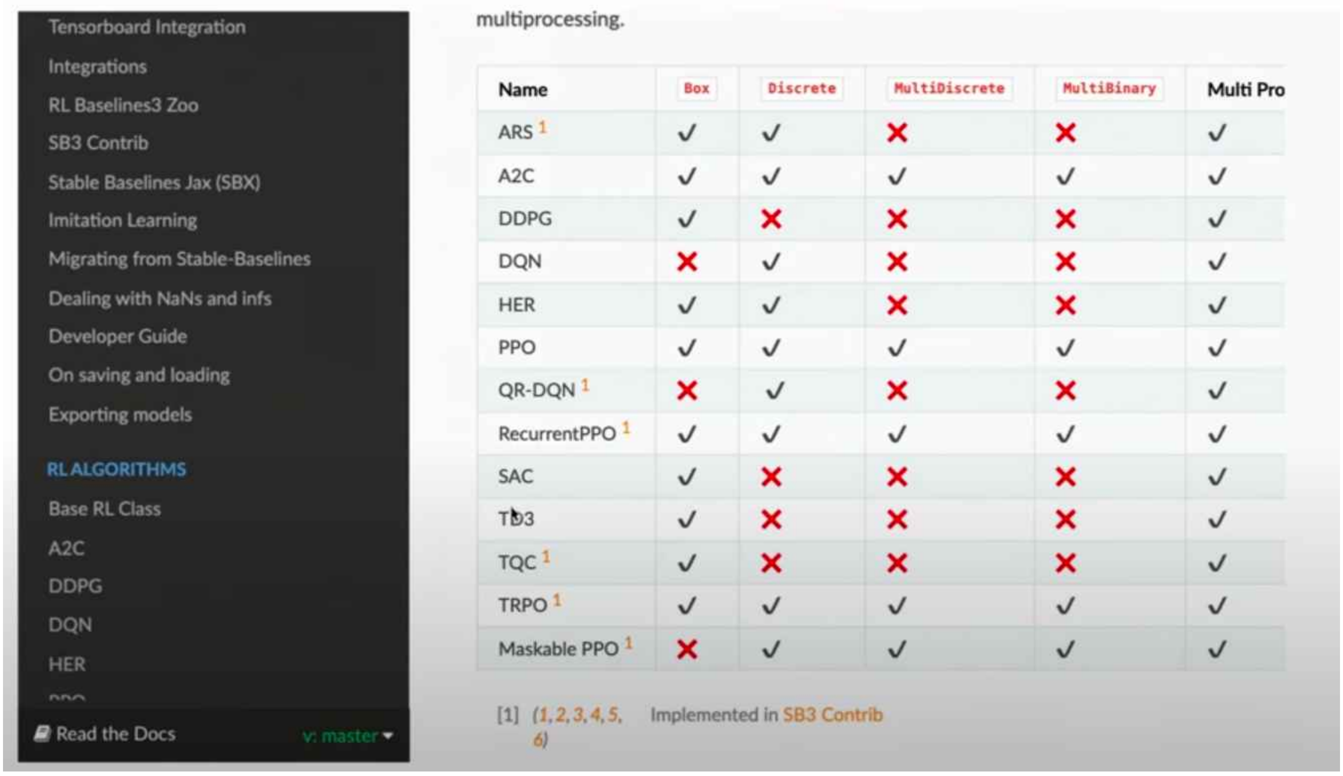
- RL problem formulation
  - $max_\alpha \sum_n r\left(y[n], u[n]\right)$
  - subject to:
  - $x[n+1] = f\left(x[n], u[n]\right)$, $x[0] = x_0$
  - $y[n] = g(x[n], u[n])$
  - $u[n] = \pi_\alpha(y[n], \ldots u[n-1] \ldots)$
- If I get a new robot, train from the beginning? Yes, most of the time, policy might be reused: Cross embodiment
- This is a general optimal control formulation, RL is a subset of Optimal Control, which emphasises
  - solution from trial & error
  - "black-box" optimization
  - stochosis updates, turn into probabilities, calculate the expectations



  - neutral net policies

1. RL is a harder optimization than BC

  - cost depends on feedback loop through the world
  - "delayed" reward
  - collections of RL algorithms
    - RL in simulation use PPO(proximal Policy Optimization)
    - RL without simulation -> off-policy RL methods are popular (SAC)
    - Drake is not using GPU, not best choice for RL, Issac is. Drake is more focus on close the sim2real gap so its more accurate, Nvidia is not. So training from zero Use Issac, and then use Drake to train the second round.



multiprocessing.

Name	Box	Discrete	MultiDiscrete	MultiBinary	Multi Pro
ARS [1]	✔	✔	❌	❌	✔
A2C	✔	✔	✔	✔	✔
DDPG	✔	❌	❌	❌	✔
DQN	❌	✔	❌	❌	✔
HER	✔	✔	❌	❌	✔
PPO	✔	✔	✔	✔	✔
QR-DQN [1]	❌	✔	❌	❌	✔
RecurrentPPO [1]	✔	✔	✔	✔	✔
SAC	✔	❌	❌	❌	✔
TD3	✔	❌	❌	❌	✔
TQC [1]	✔	❌	❌	❌	✔
TRPO [1]	✔	✔	✔	✔	✔
Maskable PPO [1]	❌	✔	✔	✔	✔

[1] (1, 2, 3, 4, 5, 6) Implemented in SB3 Contrib

2. Aside Parametrizing Dynamic Policies

- Linear dynamics example:

$x[n+1] = Ax[n] + Bu[n]$ $y[n] = Cx[n] + Dx[n]$	$y[n] = \sum_k \alpha_k y[n-k] + \sum_k \beta_k u[n-k]$
state space	ARX auto regressive model
recurrent neutral network (LSTM)	GPT
Multibody Plant	Diffusion Policy

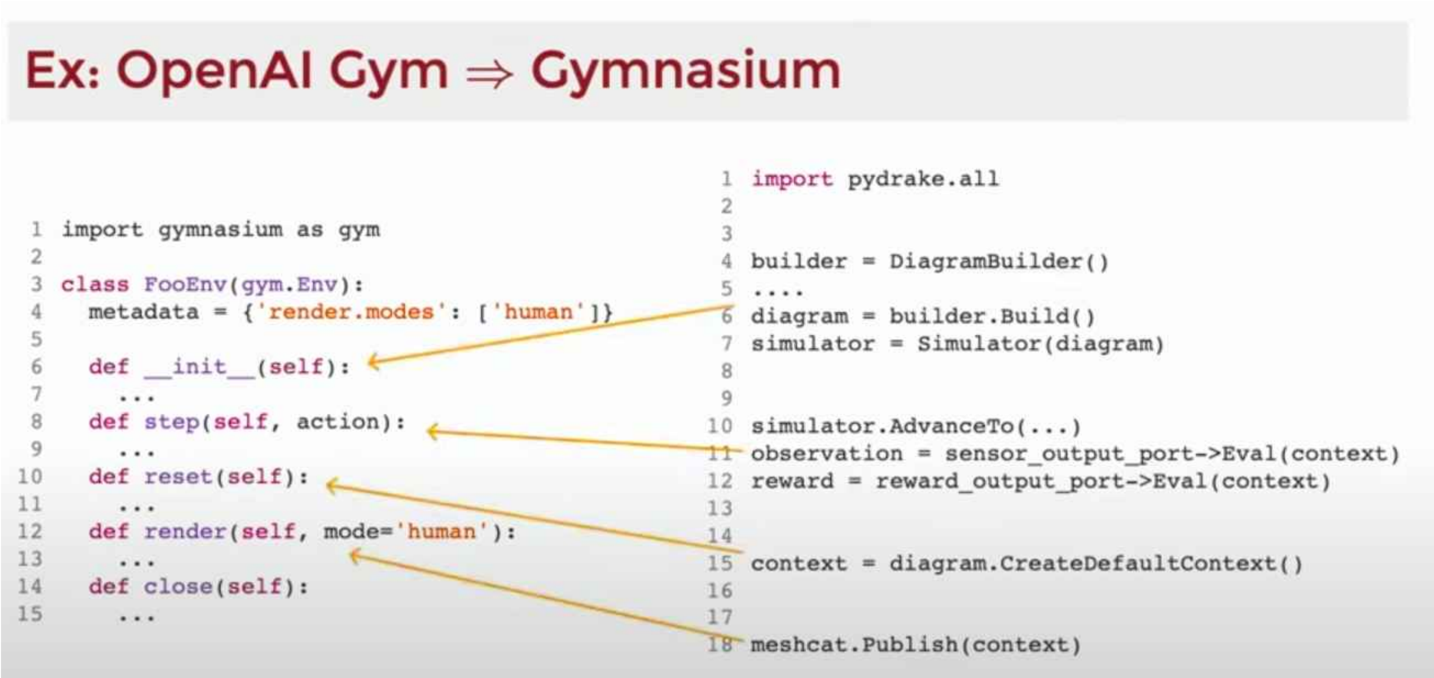
3. RL Recipe

1. Make the simulator
2. Write cost function, hardest part
3. Deep Policy gradient

4. Higher level messages

- Good software(gpu, pytorch etc)

- if you can write a simulation + elbow grease(cost funtion), most problems can be solved with RL
- Not RL vs BC vs Models, they all related



5. Gym vs Drake

- Gym
    - no Context (-> nointrospection, no deterministic playback)
    - floats only (no autodiff, no symbolic)
    - no access to multibody quantities (e.g., jacobians, inverse dynamics)
    - but can model anything
    - RL philosophy: "black box"; Drake: "glass box"

FINGER PIVOTING — SLIDING — FINGER GAITING

OpenAI - Learning Dexterity

"PPO has become the default reinforcement learning algorithm at OpenAI because of its ease of use and good performance."

https://openai.com/blog/openai-baselines-ppo/

Policy Architecture

Network

```
model = PPO('MlpPolicy', env, verbose=1, tensorboard_log=log)
```

```
stable_baselines3/common/policies.py#L435-L440

 # Default network architecture, from stable-baselines
net_arch = [dict(pi=[64, 64], vf=[64, 64])]
```

Actions

approximately:

```
builder.ExportOutput(inv_dynamics.get_desired_position(), "actio
```

Observations

```
builder.ExportOutput(plant.get_state_output_port(), "observations")
```

Cost Function

```
angle_from_vertical = (box_state[2] % np.pi) - np.pi / 2
cost = 2 * angle_from_vertical**2  # box angle
cost += 0.1 * box_state[5]**2  # box velocity
effort = actions - finger_state[:2]
cost += 0.1 * effort.dot(effort)  # effort
# finger velocity
cost += 0.1 * finger_state[2:].dot(finger_state[2:])
# Add 10 to make rewards positive (to avoid rewarding simulator
# crashes).
output[0] = 10 - cost
```

- in RL, if you have partial derivatives, do you use them? No
- Then how to minimize cost function without using gradients (python toolbox, never grad)
- Simplest form of RL, idea: weight perturbation
  - stochastic gradient descent
  - Eval f() twice:
  - $f(\alpha)$, $f(\alpha + w)$, w: small random noise
  - $\Delta\vec{\alpha} = -\eta[f(\alpha + w) - f(\alpha)]w$
  - learning rate: $E[\Delta\alpha] \propto -\frac{\partial f}{\partial \alpha}$
  - $var(\Delta(\alpha))$ might be bad, converging slowly, one thing you can do:
  - Change the place where you do the sampling:
  - adding noise to α, scale with number of params in network:
  - $u[n] = \pi_\alpha(y[n], u[n-1]..) + w[n]$, add noise to policy ouput
  - variance grows with # of outputs * # of timesteps