locomotion

Main question: if we want to follow a motion pattern, how should we actuate the motor to do that?

Traditional model-based methods

MPC(Model Predictive Control)

Model Predictive Control (MPC)

- Definition: At each time step t_s, solve the following optimization problem and apply $u(t_s)$ as action.

$$
\begin{aligned}
\underset{\mathbf{u}(\cdot)}{\text{minimize}} \quad & \int_{t_s}^{t_f} l(\mathbf{x}(t), \mathbf{u}(t), t)\, \mathrm{d}t, && \text{(1a) Cost} \\
\text{subject to} \quad & \mathbf{x}(t_s) = \mathbf{x}_s, && \text{(1b) Initial condition} \\
& \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}, t), && \text{(1c) System dynamic} \\
& \mathbf{g}(\mathbf{x}, \mathbf{u}, t) = \mathbf{0} && \text{(1d) Equality constraints} \\
& \mathbf{h}(\mathbf{x}, \mathbf{u}, t) \geq \mathbf{0}, && \text{(1e) Inequality Constraints}
\end{aligned}
$$

where $x(t)$ is the state and $u(t)$ is the control input.

An Example: Inverted Pendulum

Control input $\mathbf{u}(t) = \left[F(t)\right]$

State $\mathbf{x}(t) = \begin{bmatrix} y(t) \\ \theta(t) \\ \dot{y}(t) \\ \dot{\theta}(t) \end{bmatrix}$

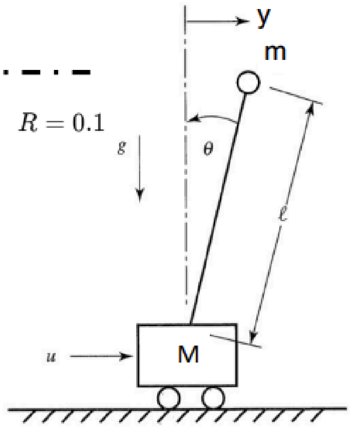
- Cost $\min_{\mathbf{u}(\cdot)} \int_{t_s}^{t_f} \left[\mathbf{x}(t)^T Q \mathbf{x}(t) + \mathbf{u}(t)^T R \mathbf{u}(t)\right] dt \qquad Q = \mathrm{diag}(10, 100, 1, 10), \quad R = 0.1$

- System dynamic

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{y} \\ \dot{\theta} \\ \frac{F + ml\dot{\theta}^2 \sin\theta - mg\sin\theta\cos\theta}{M + m\sin^2\theta} \\ \frac{-F\cos\theta - ml\dot{\theta}^2 \sin\theta\cos\theta + (M+m)g\sin\theta}{l(M + m\sin^2\theta)} \end{bmatrix}$$

- Constraints

$$\mathbf{h}(\mathbf{x}, \mathbf{u}, t) = \begin{bmatrix} y(t) + y_{\max} \\ y_{\max} - y(t) \\ F(t) + F_{\max} \\ F_{\max} - F(t) \end{bmatrix} \geq 0$$



Refer to this note for more details.

- intuition: the control is actually lag behind the observation. When the robot get the observation, it calculates the control signal(force, torque), during the calculation, the robot still moves, so the control signal is not accurate. That's why MPC predicts and optimize a period of cost, we can assume it optimize the cost over a span of calculation time, than when it get the answer, it can be apply to the real robot.
- problem:
  - contact constraints and dynamics are extremely hard to model.
  - need precise CAD model of real robot to make sure the calculation is right

RL

Why Use RL in Locomotion

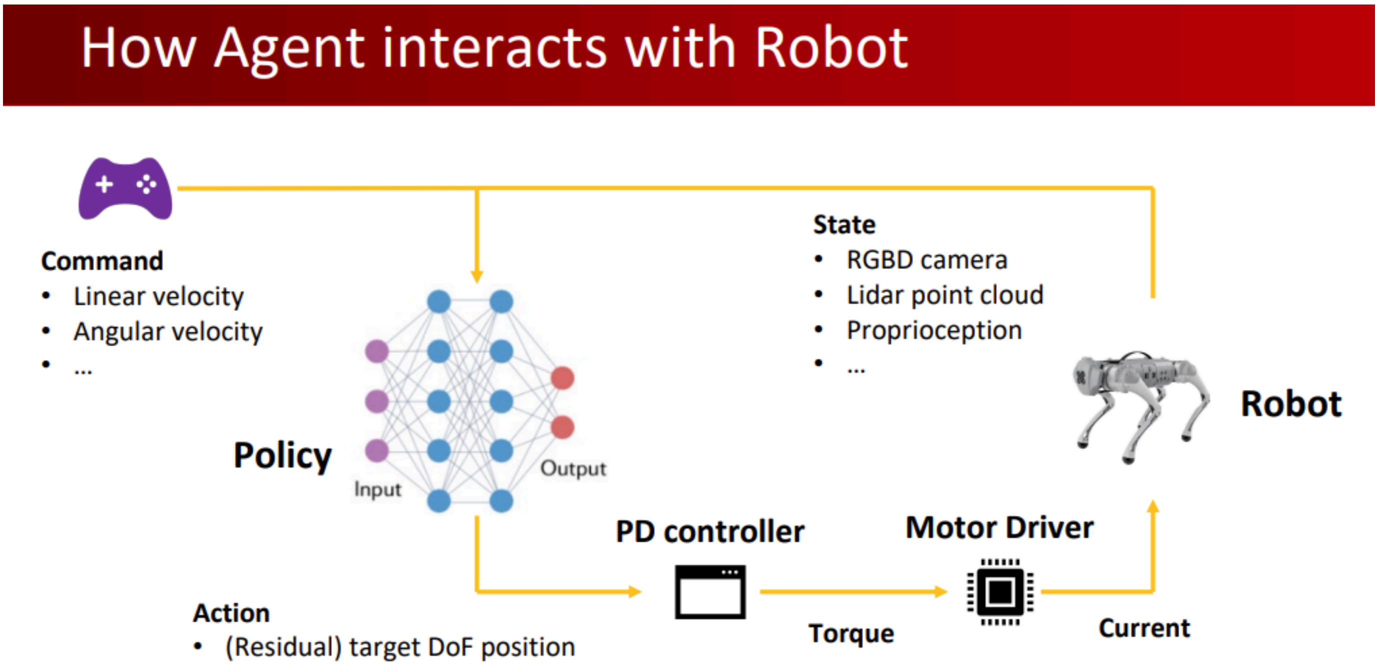
- deal with extremely complex dynamics: MPC(x)
  - Contacts and other constraints are hard to model.
  - MPC-based locomotion lacks robustness to disturbances.
- High difficulty in data acquisition: Supervised learning(x)
  - Strong coupling between data and configuration.
  - Teleoperation + imitation is a common practice in manipulation. However, it’s almost impossible to do teleoperation in locomotion.
  - Naturally integrates perception and decision-making

MPC vs RL

Aspect	MPC	RL
Model Requirement	Yes	No
Constraint Handling	Naturally handles constraints	Handle constraints through reward
Real-Time Optimization	Solves optimization when deploying	Requires a learning stage
Environment Adaptability	Suitable for known and stable environments	Capable of adapting to complex and unknown environments
Handling High-Dimension	Optimization can be complex	Performs well with high-dimensional state and action spaces
Long-Term Decision Making	Optimizes a fixed prediction horizon	Naturally handles long-term decision

Model-free RL

1. overview

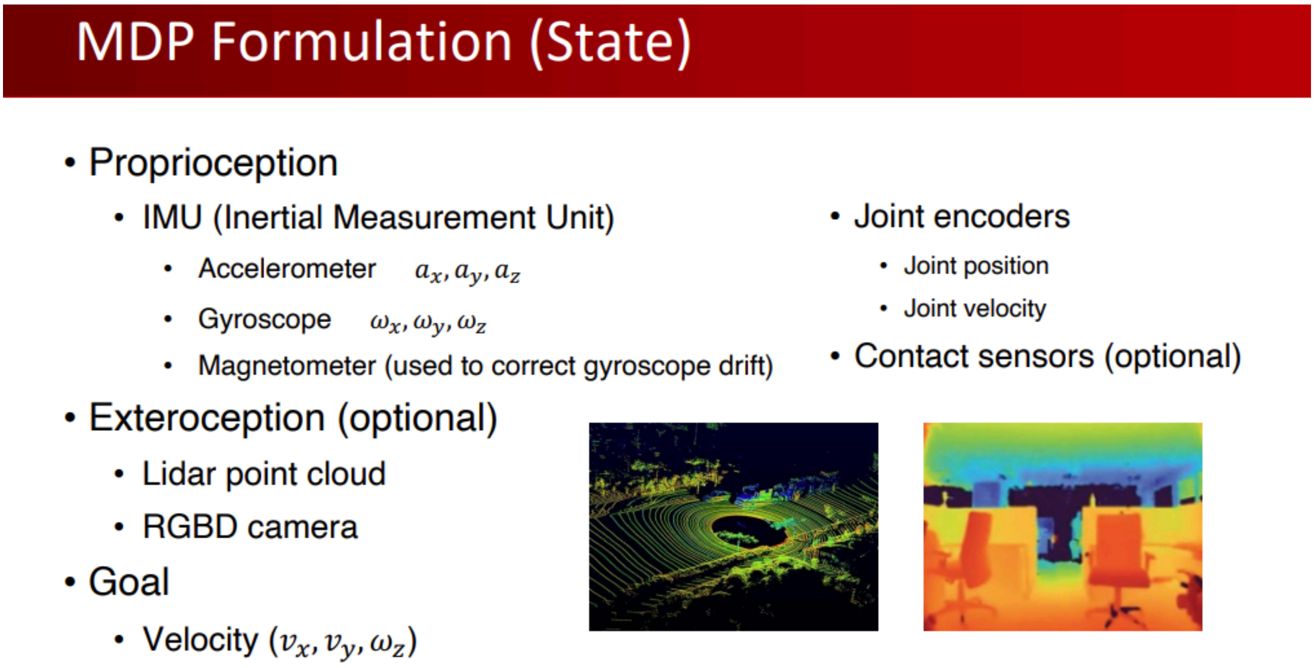


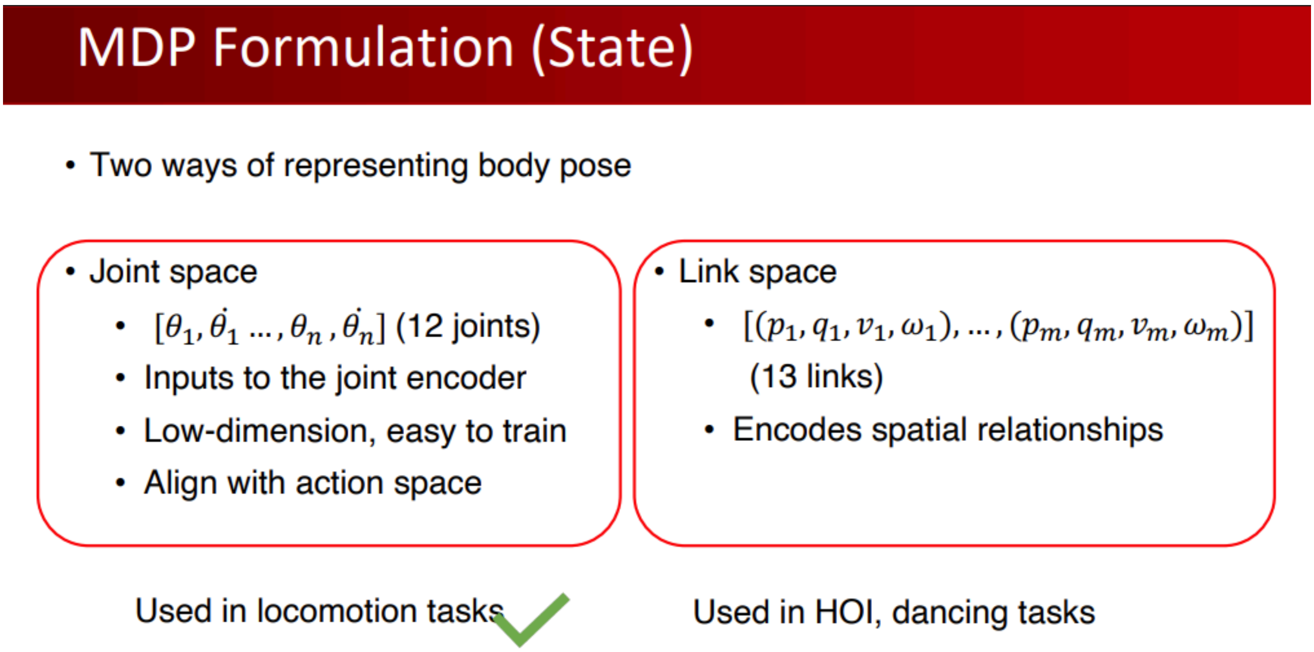
- first, we need a command input, which is the desired motion pattern. This can be generated by a motion planner or a human operator.
- second, the policy network takes in command and state to predict the action, which is usually the target position(target qpos).
- third, use PD to generate control signal(torque).
- last, the motor will apply current to actuate the robot.

This is not exactly an end2end system, because the policy network does not directly predict the torque.

2. details

- state: what should be taken to predict action(and calculate reward)?





- action: Predict Position + PD control
  - Position PD control is the most widely used in locomotion tasks
    - vs. torque control: PD policy can be updated very slowly (e.g. 50 Hz), but still output torque with high frequency (e.g. 1000 Hz).
    - vs. velocity PD control: Position PD control performs better in sim-to-real.
  - Why PID is not used?
    - In locomotion and other non-stationary motions, the integral term continuously accumulates errors, leading to oscillations in the control signal.
    - RL inherently possesses compensation capabilities similar to the integral term.

        - Most importantly, RL+PD performs well.
  - reward: just tune it.

other methods

1. curriculum learning: learn a sequence of skills and gradually increase the difficulty of the task.
    - example: quadrupedal learn to traverse different terrain with one policy.
    - disadvantage: the robot trys to learn a conservative policy, which helps it to adapt to diverse envs. This usually introduce some non-optimal behavior when deploys in real envs. Like hit the floor heavily. Fine-tuning on real envs can mitigate this issue, but every single robot needs different fine-tuning, so the cost is high.
2. Hierarchical learning: seperate perception(vision), navigation and locomotion.
    - example:
        a. ANYmal: learn seperate locomotion patterns, like walk, climb up, and jump. Then a decision-making policy can be trained to choose which locomotion pattern to use.
        b. QuadWBG: predict the base pose so that the robot can be able to grasp -> predict the linear and angular velocity -> do locomotion to follow that linear and angular velocity.
3. Privileged learning (teacher-student method): the teacher have access to oracle information, the student is what we use to make decision and student learns from the teacher. For example, the student only use vision and priprioception to make decision, but the teacher has access to the state of the robot and env.
    - Policy Distill:
    - Asymmetric Actor-Critic: critic has oracle information, but actor has no access to it.

Sim-to-real

- Ways to mitigate sim-to-real gap
    - Reward design
        - Action smoothness, action rate
- Domain Randomization
    - Dynamics: center of mass, mass, friction
    - sensors: camera, IMU, force torque sensor
    - Other: latency, image augmentation
- Domain Adaptation
- System Identification
    - Model actuator dynamics with real data
    - Better contact model in simulation

Difficulty

- Open-source is a problem

    - Many companies don’t open algorithm
    - Hard to reimplement even for open-sourced algorithm
- Both MPC and RL needs careful tuning
    - Each gait needs to tune a model (especially for MPC)
- Robustness issue and sim-to-real gap

Future Directions

- Combining control and learning
    - Using trajectory planner to guide the low-level RL-based controller
    - Using RL to calculate reference key points (e.g. foothold location) for MPC
- Better Training Strategy
    - Make training robust to the coefficients of reward terms
- Safety
    - Safe RL
- Mobile-manipulation
    - Make upper-body better coordinate with lower-body