Robot Learning
Lecture Script

Marc Toussaint & Wolfgang Hönig

Summer 2024

This is a direct concatenation and reformatting of lecture slides and exercises.

Contents

1 Lectures

1.1 Introduction

(slides by Marc Toussaint)

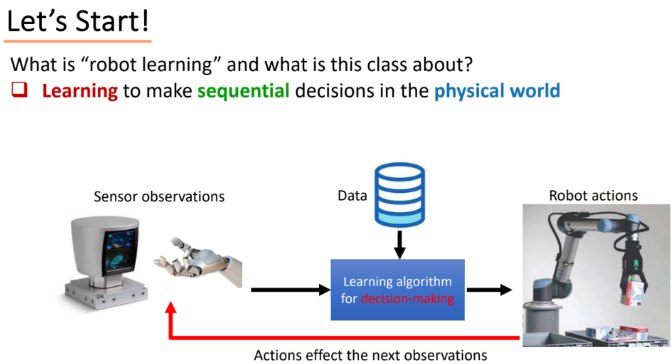
What is this lecture about?

- Related Lectures:
  - Guanya Shi (CMU): Robot Learning https://16-831-s24.github.io/lectures
  - Erdem Biyik (USC): https://liralab.usc.edu/csci699/
  - Jan Peters (TU Darmstadt): https://learn.ki-campus.org/courses/moocrobot-tud2021
  - Yisong Yue & Hoang M. Le (CalTech): https://sites.google.com/view/icml2018-imitation-learning/

1.1:1

What is this lecture about?

- Shi's lecture (referenced below):



1.1:2

What is this lecture about?

- Shi's lecture (referenced below):

What is “Robot Learning”?

- **Learning** to make **sequential** decisions in the **physical world**
- **Learning**: Data-driven and improve from data
  - W/o learning & data: search & planning, classic control, optimal control, ...
- **Sequential**: The current action/decision influences the next state therefore the next action/decision
  - W/o sequential: bandit, standard supervised learning, ...
- **Physical world**: The robot needs to interact with the physical world in the closed-loop
  - A.k.a. “embodied intelligence”
  - W/o physical world: RL for games, LLMs...

1.1:3

What is this lecture about?

- In Shi’s view:
  - Formalize the problem “making sequential decisions in a physical world” ($\rightarrow$ MDPs)
  - Focus on Learning in MDPs $\rightarrow$ Reinforcement Learning

1.1:4

What is this lecture about?

- However, the topic is much wider
- Robotics is a very wide field – **Learning can be applied almost anywhere**

1.1:5

What is this lecture about?

- Module description (Moses 41016) – Learning Outcomes
  - The students have a systematic understanding of the wide variety of contexts and problems settings in which machine learning methods can be applied within robotics.
  - They understand how the learning problems are mathematically formulated in these settings.
  - [They also learn about underlying ML methods to tackle these problems.]. . .
- Content
  - The term Robot Learning generally denotes the use of learning methods in the context of robotics, which is ubiquitous in modern robotics research. This course aims to provide a systematic introduction to the field, in particular to the various contexts and problem setting where machine learning can be applied and the specific learning methods themselves. This includes topics such as:
  - System identification, model learning, residual model learning
  - Imitation learning, behavior cloning, learning from demonstration
  - Reinforcement Learning (RL), skill learning, offline RL
  - Constraint learning, grasp learning, iterative learning control

- Learning to predict plans, learning to warmstart MPC or optimization
- Inverse RL
- ...

1.1:6

Motivation

- OpenAI / Figure robot: https://www.youtube.com/watch?v=Sq1QZB5baNw
- Boston Dynamics: https://www.youtube.com/watch?v=tF4DML7FIWk

- CoRL 2023 award/finalist papers:
  - https://hshi74.github.io/robocook/
  - https://mimic-play.github.io/
  - https://robot-parkour.github.io/

1.1:7

The State-of-the-Art in Robot Learning

- Conference on Robot Learning https://www.corl.org/
- Robotics: Science and Systems Conference https://roboticsconference.org/
- ICRA, IROS, L4C conferences
- NeurIPS, ICML conferences

1.1:8

- The meta-goal of this lecture:
  Enable you to read & understand papers at these conferences

- Some of the lectures will directly discuss essential research papers

1.1:9

Planned Lectures

- Taxonomy (today)
- Robotics Primer & Machine Learning Primer
- Dynamics Learning / System Identification
- Imitation Learning
- *Method Lecture:* Diffusion & other policy representations

- Reinforcement Learning & variants (several lectures)
- Safe Learning, Multi-Robot Learning
- Constraint Learning, Grasping/Manipulation Learning, Affordance Learning
- *Method Lecture:* Robotics/3D ML: Rotation encodings, PointNet, SE(3)-Equivariant
- *Method Lecture:* Black-Box Optimization, CMA, CEM
- Plan Prediction Learning (from MPC to Language Models)
- Online adaptation
- *Method Lecture:* Generative models (PCA, auto encoder, VAE, GANs, diffusion, stochastic outputs in transformers)

1.1:10

Organization

1.1:11

Organization

- 6 LPs (180h, 12h/w, 15 weeks)
- Lectures, weekly, in person
- Tutorials, weekly:
  – Weekly exercise sheets, mix of analytic/coding, to be discussed in the tutorials

- ISIS as central webpage
- Contact:
  – Office (grades/etc): Ilaria Cicchetti-Nilsson <office@lis.ut-berlin.de>

Ilaria Cicchetti-Nilsson

1.1:12

Assignments & Exam

- Tutorial exercises are a mix of analytic and coding problems. **Voting System:**
  – When attending a tutorial, students mark in an ISIS questionnaire which exercises they have worked on
  – Students are randomly selected to present their solutions (no need for correct solutions – just something to present and discuss)
  – When not attending: upload pdf notes/solutions on ISIS

- **Exam prerequisite:**
  – at least 50% votes in the exercises

- The written exam will be about analytical problems, determines final grade (no portfolio)

1.1:13

Prerequisites

- Module description:
  – Knowledge in Machine Learning
  – Fundamentals in AI (esp. Markov Decision Processes)
  – Foundations of robotics
  – Basic programming skills
- Self-Checks:
  – Maths, AI, ML & Robotics lectures:
    https://www.user.tu-berlin.de/mtoussai/teaching/Lecture-Maths.pdf
    https://www.user.tu-berlin.de/mtoussai/teaching/Lecture-AI.pdf
    https://www.user.tu-berlin.de/mtoussai/teaching/Lecture-MachineLearning.pdf
    https://www.user.tu-berlin.de/mtoussai/teaching/Lecture-Robotics.pdf
  – ML: not only pyTorch.. but also *Hastie et al: The Elements of Statistical Learning*?
    https://hastie.su.domains/Papers/ESLII.pdf
  – For reference:
    https://www.user.tu-berlin.de/mtoussai/teaching/#reference-material
- Numeric coding in Python (numpy)

1.1:14

Module description (Moses 41016)

- Grading
  – graded, written exam, English (90min)
- This module is used in the following module lists:
  – Automotive Systems (M. Sc.)
  – Computer Engineering (M. Sc.)
  – Computer Science (Informatik) (M. Sc.)
  – Elektrotechnik (M. Sc.)

1.1:15

1.2 Taxonomy

(slides by Marc Toussaint)

Robot Learning Taxonomy

I. What is learned?

– Which mapping between state, control, rewards/values/constraints, plan, observation is learned?

II. How is the data generated?

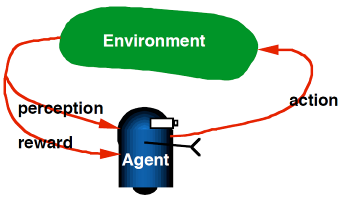
– By robot itself? (online?) By human demonstration? In simulation?

– Optimally? Safe?

– Are labels available? (Supervised vs. RL vs. un-/self-supervised)

1.2:1

I. What is learned?



[Satinder Singh, ∼2005]

1.2:2

I. What is learned?

- State, control → next state: **dynamics** – System identification
- State → control: **policy** – Optimal Control, iterative learning control, Reinforcement Learning
- State, control → **rewards** – Reward function. Model-based RL, InvRL
- Observations → control: **policy** (in partially observable case)
- State → plan: **plan prediction** – for MPC, but also language models
- Observations → state: **state estimation**
- State/Observations → value: **value function** – learnt, also planned/computed (DDP)

    - State/Observations $\to$ constraint: – constraint model, success model, affordance
    - ...

1.2:3

II. How is the data generated?

- By human demonstration
  - Imitation learning (behavior cloning)
  - Inverse Reinforcement Learning, human preference learning
- Online, by robot itself
  - on-policy/off-policy learning, RL vs. offline RL
- In simulation/domain transfer
  - sim2real gap, domain randomization, domain transfer
- "Optimally": e.g. maximizing information gain
  - Active Learning, intrinsic rewards, Bayesian RL & Exploration
  - Frequency excitation in system identification
  - Pink noise, structured RL exploration
- "Safely": e.g. subject to chance constraints
  - Safe RL, safe exploration, simultaneous risk learning

1.2:4

Robot Learning Taxonomy

- These two dimensions (*I. What is learned? II. How is the data generated?*) span a large space of robot learning approaches
  - Quite beyond focus on RL only
  - Across the fields of robotics and control theory
  - Learning is not necessarily *replacing* "search & planning, classical control, optimization"

- Other aspects:
  - *Direct/Indirect?* Is the mapping learned directly? Or are components/models learned that are input to a classical solver?
  - *Scenario specific* E.g. specific for grasping, or multi-robot systems

1.2:5

1.3 Robotics Essentials

(slides by Marc Toussaint)

Robotics Essentials Outline

- A robot is an articulated multi-body system: kinematics & dynamics

- Standard Control: IK, path finding & traj. opt, PD & MPC

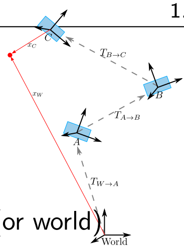
1.3:1

Robot as Articulated Multibody System

- A robot is a multibody system. Each body
  - has a pose $x_i \in \text{SE}(3)$
  - has inertia (m_i, I_i) with mass $m_i \in \mathbb{R}$ and inertia tensor $I_i \in \mathbb{R}^{3\times 3}$ sym.pos.def.
  - has a shape s_i (formally: any representation that defines a pairwise signed-distance $d(s_i, s_j)$)

[Useful: "multibody system" on Wikipedia]

1.3:2



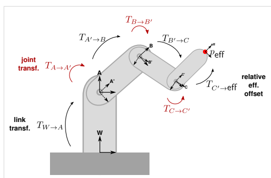
Robot as Articulated Multibody System

- **Tree structure:**
  - Every body is linked to a parent body or the world
  - We have relative transformations $Q_i \in \text{SE}(3)$ from parent (or world)

  [If not tree-structured, we only represent a tree and use additional constraints to describe loops → more involved, but doable]

- **Articulated Degrees of Freedom (dofs):**
  - Some of the relative transformations Q_i may have articulated (=motorized) **dofs** q so that $Q_i(q)$

    [Different types of joints (hinge, prismatic, universal, ball) have different # dofs and different mapping from dofs $q \mapsto Q_i(q)$]
  - We stack all dofs of all relative transformations into a single
    joint vector $q \in \mathbb{R}^n$



1.3:3

$$x \in \mathsf{SE}(3)^m: \text{ all body poses,} \qquad q \in \mathbb{R}^n: \text{ joint vector}$$

– Forward kinematics: $q \mapsto x$, $\dot q \mapsto \dot x$, $\ddot q \mapsto \ddot x$
– Forward dynamics: $u \mapsto \ddot q$, inverse dynamics: $\ddot q \mapsto u$ ($u \in \mathbb{R}^n$: joint torques)

1.3:4

Forward Kinematics $q \mapsto x$

- Given q, what is the pose of any body i?

$$q \;\mapsto \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{pmatrix} = \phi(q) \quad \in \mathsf{SE}(3)^m$$

– *Algorithm:* First determine all rel. trans. $Q_i(q)$, then forward chain them
– Often one cares only about position/orientation of one particular body x_i: the **"endeffector"**

1.3:5

Forward Velocities & Jacobian $\dot q \mapsto \dot x$

- Given $\dot q$, what is the linear and angular velocity (v_i, w_i) of any body i?

$$\dot q \mapsto \begin{pmatrix} v_1, w_1 \\ v_2, w_2 \\ \vdots \\ v_m, w_m \end{pmatrix} = J(q)\ \dot q \quad \in \mathbb{R}^{m \times 6}$$

– with **Jacobian** $J(q) = \partial_q \phi(q) \ \in \mathbb{R}^{m \times 6 \times n}$.
[Since, ϕ is SE(3)-valued, the Jacobian actually has output in its tangent space $se(3) \equiv \mathbb{R}^6$. In practise, code typically provides separate positional Jacobian $J^{\mathsf{pos}} \in \mathbb{R}^{m \times 3 \times n}$ and angular Jacobian $J^{\mathsf{ang}} \in \mathbb{R}^{m \times 3 \times n}$.]
– Since we know how to compute $\phi(q)$, we can think of $J(q)$ as the "autodiff" of it
– However, positional/angular Jacobians are really very easy to provide without expensive autodiff
[In practise, one only needs to figure out the J^{pos}, J^{ang} for a rotational and translational joint – all others follow from this.]

1.3:6

Forward Accelerations $\ddot{q} \mapsto \ddot{x}$

- Given $\ddot{q}$, what is the linear and angular acceleration $(\dot{v}_i, \dot{w}_i)$ of any body i?

$$\ddot{x} = \dot{J}(q)\ \dot{q} + J(q)\ \ddot{q}\ \approx\ J(q)\ \ddot{q}$$

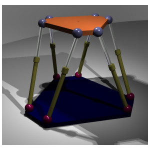
  – One typically approximates $\dot{J} = 0$

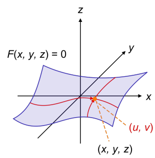
1.3:7

The word "kinematics"

[in parts from Wikipedia]

  – Mathematical description of possible motions of a (constrainted/multibody) system/mechanism *without considering the forces*
  – "geometry of [possible] motions"
  – Formally: Describe the space (manifold) of possible system poses and all possible paths in that space
  – Read **generalized coordinates** on wikipedia: Understanding motion in terms of coordinates and (non-)holonomic constraints:





1.3:8

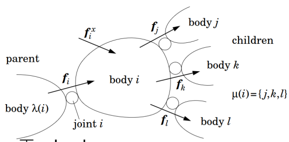
Inverse dynamics $\ddot{q} \mapsto u$

- Given $\ddot{q}$, what joint torques u do we need to generate this $\ddot{q}$ (accounting for gravity)?

- Coupled Newton-Euler equations: For each body:

$$F_i = \begin{pmatrix} f_i \\ \tau_i \end{pmatrix} = \begin{pmatrix} m_i \dot{v}_i \\ I_i \dot{w}_i + w_i \times I_i w_i \end{pmatrix}$$

$$F_i^{\text{back}} = F_i - F_i^{\text{ext}} + \sum_{j=\text{child(i)}} F_j^{\text{back}}\ , \quad u_i = h_i^\top F_i^{\text{back}}$$



from Featherstone'14

[where F_i^{ext} are external (e.g. gravity) forces; and F_i^{back} is the force "send back through the joint to the parent of i"; h_i is the joint axis (picking up the torque)]

[Can also be written as linear equation system between $\ddot{q}$, F, F^{back}, and u (with sparse matrices only) – and solved/inverted in $O(m)$.]

1.3:9

solved! We can accelerate the thing as we like

the rest is planning: How should I accelerate to reach some future goals?

1.3:10

Standard Template: Waypoint + Reference Motion + Controller

- Standard problem setting: Control motors, so that at $t = T$ seconds the endeffector x_i is at desired position $y^* \in \mathbb{R}^3$, i.e., $\phi(q_{t=T}) = y^*$
- Problem decomposition:
  - Find a final robot pose q_T that fulfills constraint $\phi(q_{t=T}) = y^*$ – **inverse kinematics**
  - Find a nice *reference* motion from current robot pose q_0 to q_T – **path finding, trajectory optimization, or trivial interpolation/PD**
  - Find a control policy $\pi : x_t \mapsto u_t$ that reactively sends motor commands to follow the reference motion – **inverse dynamics, PD control, Riccati**

  [You could think of this as three different time scales: rough future waypoint(s)/goal(s), continuous motion to next waypoint, short-term controls.]

  [There are other ways to approach this: You could remove step (1) and shift that issue into (2), or remove (1 & 2) and shift all issues into (3) - morphing this into other approaches. E.g. directly defining a desired force/acceleration behavior in "task space" (=operational space control).]

  [continuous replanning/re-estimation can also make (1) and (2) reactive.]

1.3:11

Inverse Kinematics

- Find q to fulfill $\phi(q) = y^*$ for differentiable fwd kinematics ϕ.

$$\min_{q \in \mathbb{R}^n} \|q - q_0\|^2 \quad \text{s.t.} \quad \phi(q) = y^*$$

$$\text{or} \quad \min_{q \in \mathbb{R}^n} \|q - q_0\|^2 + \mu \|\phi(q) - y^*\|^2 \quad \text{for large } \mu$$

- Solution for linearized ϕ:

$$q^* = q_0 + J^\top (J J^\top + \tfrac{1}{\mu}\mathbf{I})^{-1} (y^* - \phi(q_0))$$

Python Package: https://marctoussaint.github.io/robotic/

1.3:12

Path Finding & Trajectory Optimization

- Given current q_0 and future q^*, find a collision free **path**
  - Wolfgang Hönig's & Andreas Orthey's lecture
  - RRTs, PRMs, under constraints (kinodynamic)

- **Trajectory** opimization
  - Time continuous formulation:

$$\min_{q(t)} \int_0^T c(q(t), \dot q(t), \ddot q(t))\ dt \quad \text{s.t.} \quad q(0) = q_0,\ q(T) = q^*, \dot q(0) = \dot q(T) = 0\ , \forall_{t \in [0,T]} : \bar\phi(q(t$$

  - Time-discretized, assuming k-order Markov coupling terms (KOMO):

    A tutorial on Newton methods for constrained trajectory optimization and relations to SLAM, Gaussian Process smoothing, optimal control, and probabilistic inference: *Marc Toussaint*. Springer 2017

1.3:13

Control around a Reference

- Use **Inverse Dynamics** directly
  - We have $\ddot q^*(t) \to$ map it to controls u directly
  - But what if you're off the reference a bit? *How to steer back?*
- Use **PD law** to accelerate back to reference:
  - Define a PD law $\ddot q^{\text{desired}} = \ddot q^*(t) + k_p(q^*(t) - q) + k_d(\dot q^*(t) - \dot q)$ with desired PD behavior back to reference
  - Then use Inv dynamics $\ddot q^{\text{desired}} \mapsto u$
  - (Also ok, but needs severe tuning: directly define a PD controller $\ddot u = M\ddot q^*(t) + K_p(q^*(t) - q) + K_d(\dot q^*(t) - \dot q)$.)
- Use **Riccati** to get an **Optimal Linear Regulator** around reference
  - Define optimal control problem, e.g., $\min_{\pi : q, \dot q \mapsto u} \int_0^T c(q(t), \dot q(t), u(t))\ dt + \phi(x(T))$
  - We can linearize dynamics around reference $\to$ has an analytic solution (Algebraic Riccati eq.)
  - Resulting controller is a "linear regulator", i.e., a PD law where matrices K_p, K_d depend on t and are chosen optimally.

1.3:14

Model-Predictive Control (MPC)

- When getting far away from the reference, linearization of Riccati might break, and PD is too simple

- Continuously replan ($\sim$ 10-1000Hz): re-solve the optimal control problem
  - Optimal Control problem can also include task constraints directly, not only following a reference
  - As a compromise: typically limit horizon

This is a default way of "thinking control" in robotics

1.3:15

Summary

- A robot is an articulated multi-body system
  - Fwd kinematics: $q \mapsto x$, $\dot q \mapsto \dot x$, $\ddot q \mapsto \ddot x$
  - Fwd dynamics: $u \mapsto \ddot q$, inv dynamics: $\ddot q \mapsto u$

- Standard Control Template:
  - IK (or constraint solving) to estimate future goal/waypoints
  - Path Finding & Trajectory Optimization to estimate Reference Motion
  - PD, Linear Regulator, or MPC to control (around the reference)

1.3:16

How far can we get with this approach?

- What did we assume to *know*?
  - Structure of multi-body system, all shapes, inertias
  - All goals/objectives modelled (=programmed) as differentiable costs/constraints

1.3:17

Challenge 1: Interacting with the environment

- If we only care about the **robot itself** (all goals/objectives/models concern the robot directly) – the above it totally fine

- Things get challenging when we care about **interacting with the environment**
  - Models/goals/objectives of interaction (contact, grasp) are more complicated

1.3:18

Challenge 1: Interacting with the environment

- Example: Locomotion
  - Interaction: Making contact with the ground to generate ground forces
  - Robot root is not attached to world, but free floating (complicates dynamics a bit)
  - Dynamics heavily influenced by ground forces, which are *contact complementary* hard on-off switching of forces at contact $\to$ hybrid/discrete structure, makes dynamics and solvers much much more complicated (hybrid control)

... more complicated than "vanilla robot", but still doable

1.3:19

Challenge 1: Interacting with the environment

- Example: Manipulation
  - Objects in the environment (part of the "multibody system") have their own DOFs, but are NOT "articulated" with motors: if not grasped or touched, they cannot move $\to$ their Jacobian $\partial_q x_i = 0$
  - Hard on-off switching of manipulability; hybrid dynamics & problem
  - Dynamics of object motions can be much more complicated than (also free-floating) robot dynamics: friction, stiction, slip, non-point contacts
  - Waypoint constraints $\phi(x_t)$ much more complicated (correct grasping of complex shape, pushing, throwing)
  - If objects are deformable, their form becomes DOF (e.g. neural latent code) – becomes much much more complicated in above approach

- In essence, things become much more complicated, but one still *can* write down essential physics equations of object interaction, and use these equations in above approach

1.3:20

Challenge 2: State Estimation

- All of the above requires to estimate states
  - q_0 (includes pose of a mobile robot)
  - x_i (poses of objects in environment)
  - shapes and inertias in the environment, dynamics parameters (e.g. friction)

[Basic state estimation can often also be formulated as optimization problem (e.g. graph-SLAM) – similar to motion optimization: Find estimates (also of past motion) that is *most consistent* with sensor readings; minimze error between real readings and model-predicted readings. (Or as probabilistic inference.)]

1.3:21

Relation to Robot Learning

- On the formal/theory side, they share foundations:
  - Optimal Control formulation $\leftrightarrow$ Markov Decision Processes & Reinforcement Learning
  - More generally: optimality formulations $\to$ learning/black-box opt. approaches

- Components can be *replaced* or *shortcut* by learning:
  - Dynamic modelling $\leftrightarrow$ system identification
  - Optimal Control (e.g., MPC, Riccati) can be shortcut by learning V- or Q-function
  - Need of inverse dynamics can be shortcut by learning Q-function instead of V-function
  - Constraint solving (also IK) can be shortcut by directly learning a policy or sampler that fulfills constraint
  - **Shortcut state estimation:** Avoid all state-based models, learn direct sensor-based models (policies, value functions, planners, dynamics, etc)
  - **End-to-end:** Shortcut the whole approach by learning images $\mapsto$ torques

1.3:22

1.4 Machine Learning Essentials

(slides by Marc Toussaint)

Machine Learning Essentials

- Supervised ML $\quad f_\theta : x \mapsto y$

- Unsupervised ML $\quad p_\theta(x) \quad$ (and conditional $p_\theta(x|z)$)

  [Neglected here: Optimal embeddings, clustering]

1.4:1

Supervised ML

- Given data $D = \{(x_i, y_i)\}_{i=1}^n$ and a parameterized $f_\theta : x \mapsto y$, find θ

$$\min_\theta \underbrace{\sum_{i=1}^n \ell(y_i, f_\theta(x_i))}_{\text{(data) loss}} + \underbrace{R(\theta)}_{\text{regularization}}$$

done! That's (supervised) ML

1.4:2

Loss Functions

- Regularizations:
  - L_2 (Ridge): $R(\theta) = \|\theta\|_2^2$
  - L_1 (Lasso): $R(\theta) = \|\theta\|_1$

- Regression $y \in \mathbb{R}^m$: Squared error: $\ell(y, \hat{y}) = (y - \hat{y})^2$

  [Robust variants: Huber loss, Forsyth]

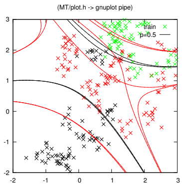
- Classification $y \in \{0, .., M\}$ (where $f : x \mapsto f(x) \in \mathbb{R}^M$ discriminative values)
  - Neg-Log-Likelihood: $\ell(y, f(x)) = -\log p(y|x)$ with $p(y|x) = \frac{e^{f_y(x)}}{\sum_{y'} e^{f_{y'}(x)}}$
  - Hinge: $\ell(y, f(x)) = \sum_{y' \neq y} [1 - (f_{y^*}(x) - f_{y'}(x))]_+$
  - Cross-Entropy: $\ell(y, f(x)) = -\sum_z h_y(z) \log p(z|x)$ *same as NLL for one-hot-encoding* $h_y(z) = [y = z]$

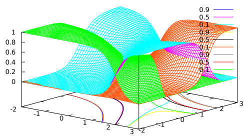
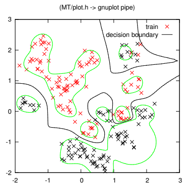
1.4:3

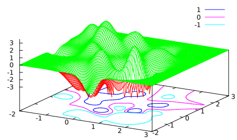
Parameterized Functions

- Linear $f_\theta(x) = \theta_0 + \sum_{j=1}^d \theta_j x_j = \bar{x}^\top \theta$

- Linear in features: $f_\theta(x) = \phi(x)^\top \theta$ (or Hilbert space..)
  - Linear: $\phi(x) = (1, x_1, .., x_d) \in \mathbb{R}^{1+d}$
  - Quadratic: $\phi(x) = (1, x_1, .., x_d, x_1^2, x_1 x_2, x_1 x_3, .., x_d^2) \in \mathbb{R}^{1+d+\frac{d(d+1)}{2}}$
  - Cubic: $\phi(x) = (.., x_1^3, x_1^2 x_2, x_1^2 x_3, .., x_d^3) \in \mathbb{R}^{1+d+\frac{d(d+1)}{2}+\frac{d(d+1)(d+2)}{6}}$
  - Also: Radial-Basis Functions (RBF), piece-wise linear





1.4:4

Parameterized Functions

- Neural Nets: Repeating non-linear and linear parts: (this is a 3-layer NN):

$$f_\theta(x) = \underset{\uparrow \text{lin}}{W_3}\ \underset{\uparrow \text{nlin}}{\phi}\Big[\ \underset{\uparrow \text{lin}}{W_2}\ \underset{\uparrow \text{nlin}}{\phi}[\ \underset{\uparrow \text{lin}}{W_1}\ x + b_1\ ] + b_2\ \Big] + b_3$$

  – Non-linear parts:
    – rectified linear unit (ReLU): $\phi(x) = [x]_+ = \max\{0, x\}$
    – leaky ReLU: $\phi(x) = \max\{0.01x, x\}$
    – sigmoid, logistic: $\phi(x) = 1/(1 + e^{-x})$
    – max-pooling, soft-max, layer-norm
  – Linear parts:
    – Fully connected (W_i is a full matrix)
    – Convolutional
    – Transformer-like (cross-attentions)

1.4:5

- In essense
  – You define the parameterized function f_θ
  – You define the loss ℓ and regularization R
  – You provide the data set D

  – An optimizer (analytic for linear models, stochastic gradient otherwise) finds good parameters θ

- And you cross-validate to check your hyper-parameter choices

1.4:6

Unsupervised ML

- Given data $D = \{x_i\}_{i=1}^n$, learn "something" about $p(x)$

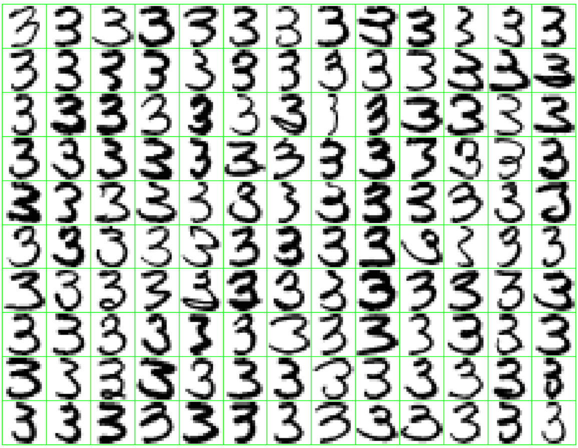
- Important setting: parameterized **autoencoder** $f_\theta : x \mapsto z \mapsto x'$, find θ

$$\min_\theta \underbrace{\sum_{i=1}^n \ell(x_i, f_\theta(x_i))}_{\text{autoencoding loss}} \;+\; \underbrace{R(\theta)}_{\text{regularization}}$$

  - You learn to reproduce x through a compact **latent code** $z \in \mathbb{R}^h$ (while $x \in \mathbb{R}^d$ is high-dimensional)
  - z has high entropy (typically Gaussian) distribution $\to$ you can **generate** $x' \sim p(x)$ by sampling z and decoding
  - If f is linear, this is called **Principle Component Analysis**
  - Better: Variational Autoencoder (VAC): Enforces $p(z)$ to have proper distribution.

1.4:7

Example: Digits



1.4:8

- There are other ideas in unsupervised learning, but the autoencoding objective is a major breakthrough

  - You "understand" the structure of data if you can compress and de-compress it
  - Autoencoders do this with powerful NN architectures

1.4:9

Diffusion Denoising Models

- Given data D, you want to learn a "system" that **generates** samples $x \sim p_\theta(x)$ where $p_\theta(x)$ models D

- Autoencoders are one approach, Diffusion Denoising Models another:
  - Train a stepwise stochastic process (Langevin dynamics) to generate samples $x \sim p_\theta(x)$
  - Has its origin in "energy-based models" and score matching
  - The step-wite sample generation process is very powerful

1.4:10

Conditional Generative Models

- Given data $D = \{(x_i, c_i)\}_{i=1}^n$ train a *conditional* distribution $p_\theta(x|c)$
  - We're actually back to Supervised ML $c \mapsto x$ (where c is the input)
  - But **if x is high-dimensional** (and c low-dim.), the generative model aspect is important:
  - The reconstruction objective enforces the system to find a good latent representation to generate high-dim. x
  - this is complemented by making conditional to c

$$f_\theta : \begin{array}{ccccc} x & \mapsto & z & \mapsto & x' \\ & & \uparrow & & \\ & & c & & \end{array}$$

A loss $\ell(x_i, f_\theta(x_i, c_i))$ jointly trains autoencoding $x \mapsto z \mapsto x'$ and conditional generation $c \mapsto z \mapsto x'$

1.4:11

1.5 Dynamics Learning

(slides by Marc Toussaint & Wolfgang Hönig)

Outline

- I. What is learned?
  - Incl. which mapping exactly, model assumption, parameterization, loss function

- II. How is the data generated?

- III. Multirotor Examples

1.5:1

I. What is learned?

environment/task parameters

instructions/lang./goal info g
physics parameters Θ

state evaluations

rewards r_t
value $V(x)$
Q-value $Q(x,u)$
constraint $\phi(x)$

state
x_t

controls
u_t

observations
y_t

plans/anticipation

waypoints/subgoals $x_{t_{1:K}}$
trajectory $x_{[t,t+H]}$
action plan $a_{1:K}$

1.5:2

Dynamics Learning – State-based view

- Learning the *state-based* dynamics:

$$x_t = f(x_{t\text{-}1}, u_{t\text{-}1}) \qquad \text{or} \qquad p(x_t \,|\, x_{t\text{-}1}, u_{t\text{-}1})$$

- Distinguish three cases:
  - **Parameter Estimation:** f is assumed physics with unknown physics parameters Θ
  - **Full Regression:** f is learned as regression model
  - **Residual Dynamics:** learn the difference to a nominal physics model

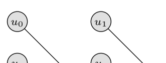
1.5:3

Dynamics Learning – Observation-based view

- x_t is the system *state*

  [Markov Property: We call a variable *state* if the future is conditionally independent on the past when conditioned on state; $I(\text{future}, \text{past} \,|\, \text{state}) = 0$.]

- Sometimes the true state is not observed (or unknown), only observations y_t are available (y_t: sensor readings, or *state estimates* from sensors)



- We need to use the **history** of observed y_t, u_t to predict next y_t!
- Distinguish three cases:
  - **Autoregression:** Learn a direct history-based model $y_t = f(y_{t-H:t}, u_{t-H:t})$
  - **Recurrent Model:** Learn a recurrent model with latent state h_t (e.g. LSTM)
  - **State-space Model:** Jointly learn embedding/decoding $x \mapsto y$ and latent dynamics $x, u \mapsto x'$ (is also a recurrent model)

1.5:4

- In summary, six cases we'll discuss more concretely:
  - state-based dynamics
    - physical parameter estimation
    - full regression
    - residual dynamics
  - observation-based dynamics
    - autoregression (NARX)
    - observation-based dynamics – recurrent model
    - observation-based dynamics – state-space model

1.5:5

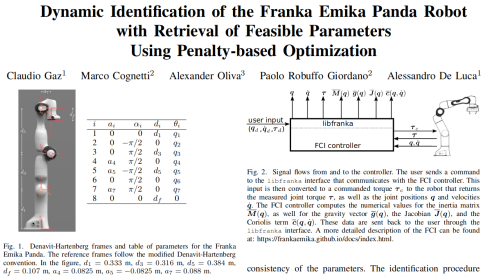
- Why learn the dynamics?
  - Given learned dynamics, we can use planning (MPC) or RL against the learned model to generate controllers
  - Examples in literature: Schaal'02, Deisenroth'15 (PILCO!), Finn'17, Driess'23, Schubert'23

- Quick terminology:
  - Dynamics Learning $\leftrightarrow$ System Identification (in control theory), Model Learning (in model-based RL)
  - In control theory u_t are called **inputs** and the *observations/measurements* y_t are called **outputs**

1.5:6

State Dynamics – Parameter Estimation

- Assume that dynamics $x_t = f_\Theta(x_{t\text{-}1}, u_{t\text{-}1})$ has unknown physical parameters Θ,e.g.:



Dynamic Identification of the Franka Emika Panda Robot with Retrieval of Feasible Parameters Using Penalty-based Optimization

Claudio Gaz[1] Marco Cognetti[2] Alexander Oliva[3] Paolo Robuffo Giordano[2] Alessandro De Luca[1]

i	a_i	α_i	d_i	θ_i
1	0	0	d_1	q_1
2	0	$-\pi/2$	0	q_2
3	0	$\pi/2$	d_3	q_3
4	a_4	$\pi/2$	0	q_4
5	a_5	$-\pi/2$	d_5	q_5
6	0	$\pi/2$	0	q_6
7	a_7	$\pi/2$	0	q_7
8	0	0	d_f	0

Fig. 1. Denavit-Hartenberg frames and table of parameters for the Franka Emika Panda. The reference frames follow the modified Denavit-Hartenberg convention. In the figure, $d_1 = 0.333$ m, $d_3 = 0.316$ m, $d_5 = 0.384$ m, $d_f = 0.107$ m, $a_4 = 0.0825$ m, $a_5 = -0.0825$ m, $a_7 = 0.088$ m.

Fig. 2. Signal flows from and to the controller. The user sends a command to the libfranka interface that communicates with the FCI controller. This input is then converted to a commanded torque $\boldsymbol{\tau}_c$ to the robot that returns the measured joint torque $\boldsymbol{\tau}$, as well as the joint positions $\boldsymbol{q}$ and velocities $\dot{\boldsymbol{q}}$. The FCI controller computes the numerical values for the inertia matrix $\overline{\boldsymbol{M}}(\boldsymbol{q})$, as well for the gravity vector $\overline{\boldsymbol{g}}(\boldsymbol{q})$, the Jacobian $\overline{\boldsymbol{J}}(\boldsymbol{q})$, and the Coriolis term $\overline{\boldsymbol{c}}(\boldsymbol{q}, \dot{\boldsymbol{q}})$. These data are sent back to the user through the libfranka interface. A more detailed description of the FCI can be found at: https://frankaemika.github.io/docs/index.html.

consistency of the parameters. The identification procedure

Claudio Gaz, Marco Cognetti, Alexander Oliva, Paolo Robuffo Giordano, and Alessandro De Luca, (2019). Dynamic identification of the franka emika panda robot with retrieval of feasible parameters using penalty-based optimization. *IEEE Robotics and Automation Letters*, 4(4):4147–4154

1.5:7

State Dynamics – Parameter Estimation

- Given data $D = \{(x_t, x_{t\text{-}1}, u_{t\text{-}1})\}_{t=1}^T$, find parameters

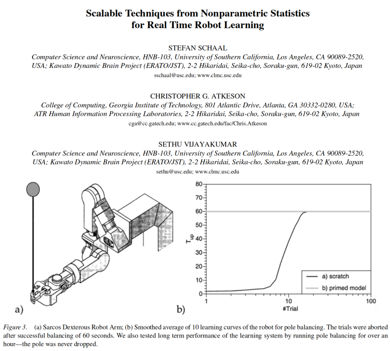
$$\min_\Theta \sum_t \|x_t - f_\Theta(x_{t\text{-}1}, u_{t\text{-}1})\|^2$$

- Sometimes, it is possible to describe f_Θ as linear in Θ. See Gaz'19!
  - Then finding optimal Θ leads to a linear least squares problem.
  - Otherwise: Black-box optimization (CMA-ES) or gradient-based (SGD, Gauss-Newton)

1.5:8

State Dynamics – Full Regression

- Learn f_θ directly, using some ML regression, e.g. (old-fashioned LWR):



Scalable Techniques from Nonparametric Statistics for Real Time Robot Learning

STEFAN SCHAAL
Computer Science and Neuroscience, HNB-103, University of Southern California, Los Angeles, CA 90089-2520, USA; Kawato Dynamic Brain Project (ERATO/JST), 2-2 Hikaridai, Seika-cho, Soraku-gun, 619-02 Kyoto, Japan
sschaal@usc.edu; www.clmc.usc.edu

CHRISTOPHER G. ATKESON
College of Computing, Georgia Institute of Technology, 801 Atlantic Drive, Atlanta, GA 30332-0280, USA; ATR Human Information Processing Laboratories, 2-2 Hikaridai, Seika-cho, Soraku-gun, 619-02 Kyoto, Japan
cga@cc.gatech.edu; www.cc.gatech.edu/fac/Chris.Atkeson

SETHU VIJAYAKUMAR
Computer Science and Neuroscience, HNB-103, University of Southern California, Los Angeles, CA 90089-2520, USA; Kawato Dynamic Brain Project (ERATO/JST), 2-2 Hikaridai, Seika-cho, Soraku-gun, 619-02 Kyoto, Japan
sethu@usc.edu; www.clmc.usc.edu

Figure 3. (a) Sarcos Dexterous Robot Arm; (b) Smoothed average of 10 learning curves of the robot for pole balancing. The trials were aborted after successful balancing of 60 seconds. We also tested long term performance of the learning system by running pole balancing for over an hour—the pole was never dropped.

Stefan Schaal, Christopher G. Atkeson, and Sethu Vijayakumar, (2002). Scalable techniques from nonparametric statistics for real time robot learning. *Applied Intelligence*, 17(1):49–60

1.5:9

State Dynamics – Full Regression

- Given data $D = \{(x_t, x_{t\text{-}1}, u_{t\text{-}1})\}_{i=1:n,t=1:T_i}$, find parameters

$$\min_\theta \sum_t \|x_t - f_\theta(x_{t\text{-}1}, u_{t\text{-}1})\|^2$$

  $\to$ same formulation as parameter estimation, really.

- Use supervised ML to minimize regression error

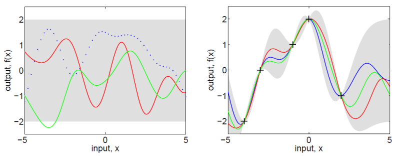
1.5:10

State Dynamics – Full Regression (probabilistic)

- Given data $D = \{(x_t, x_{t\text{-}1}, u_{t\text{-}1})\}_{i=1:n,t=1:T_i}$, find parameters

$$\min_\theta -\sum_t \log p_\theta(x_t \,|\, x_{t\text{-}1}, u_{t\text{-}1})$$

  where $p_t(x_t \,|\, x_{t\text{-}1}, u_{t\text{-}1})$ is a probabilistic regression, e.g. Gaussian Process:



(a), prior (b), posterior

(from Rasmussen & Williams)

[Marc Deisenroth's PICLO paper had huge impact: Using learned GP dynamics to derive optimal controls.]

1.5:11

State Dynamics – Residual Dynamics

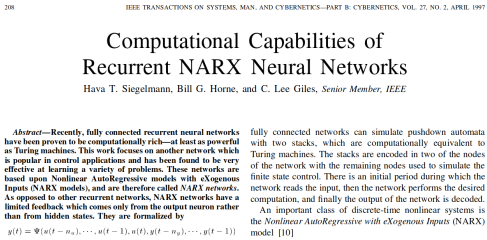
- Given a nominal dynamics f_M (e.g., assumed physics), learn a residual model f_θ to minimze

$$\min_\theta \sum_t \|x_t - [f_M(x_{t\text{-}1}, u_{t\text{-}1}) + f_\theta(x_{t\text{-}1}, u_{t\text{-}1})]\|^2$$

- Examples: Gaz'19, Multirotor Examples

1.5:12

Observation-based Dynamics – Autoregression (NARX)



208 IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART B: CYBERNETICS, VOL. 27, NO. 2, APRIL 1997

Computational Capabilities of
Recurrent NARX Neural Networks

Hava T. Siegelmann, Bill G. Horne, and C. Lee Giles, *Senior Member, IEEE*

***Abstract*—Recently, fully connected recurrent neural networks have been proven to be computationally rich—at least as powerful as Turing machines. This work focuses on another network which is popular in control applications and has been found to be very effective at learning a variety of problems. These networks are based upon Nonlinear AutoRegressive models with eXogenous Inputs (NARX models), and are therefore called *NARX networks*. As opposed to other recurrent networks, NARX networks have a limited feedback which comes only from the output neuron rather than from hidden states. They are formalized by**

$$y(t) = \Psi(u(t-n_u), \cdots, u(t-1), u(t), y(t-n_y), \cdots, y(t-1))$$

fully connected networks can simulate pushdown automata with two stacks, which are computationally equivalent to Turing machines. The stacks are encoded in two of the nodes of the network with the remaining nodes used to simulate the finite state control. There is an initial period during which the network reads the input, then the network performs the desired computation, and finally the output of the network is decoded.

An important class of discrete-time nonlinear systems is the *Nonlinear AutoRegressive with eXogenous Inputs* (NARX) model [10]

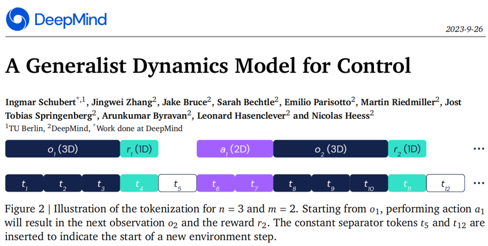
Hava T. Siegelmann, Bill G. Horne, and C. Lee Giles, (1997). Computational capabilities of recurrent NARX neural networks. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 27(2):208–215

- – NARX="Autoregression with controls" our notation: $y_t = f_\theta(y_{t\text{-}H:t\text{-}1}, u_{t\text{-}H:t\text{-}1})$
- – developed in time-series modelling, sequence modelling

- How long does the history H have to be?
- What's the modern version of autoregression?

1.5:13

Observation-based Dynamics – Autoregression (Transformers)



DeepMind 2023-9-26

A Generalist Dynamics Model for Control

Ingmar Schubert[*,1], Jingwei Zhang[2], Jake Bruce[2], Sarah Bechtle[2], Emilio Parisotto[2], Martin Riedmiller[2], Jost Tobias Springenberg[2], Arunkumar Byravan[2], Leonard Hasenclever[2] and Nicolas Heess[2]

[1]TU Berlin, [2]DeepMind, [*]Work done at DeepMind

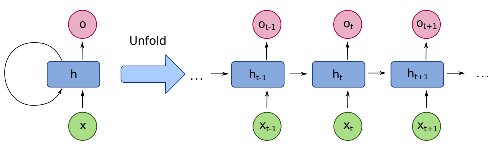
Figure 2 | Illustration of the tokenization for $n = 3$ and $m = 2$. Starting from o_1, performing action a_1 will result in the next observation o_2 and the reward r_2. The constant separator tokens t_5 and t_{12} are inserted to indicate the start of a new environment step.

Ingmar Schubert, Jingwei Zhang, Jake Bruce, Sarah Bechtle, Emilio Parisotto, Martin Riedmiller, Jost Tobias Springenberg, Arunkumar Byravan, Leonard Hasenclever, and Nicolas Heess, (2023). A generalist dynamics model for control

1.5:14

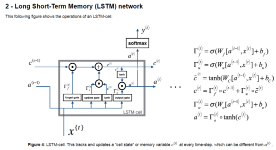
Observation-based Dynamics – Recurrent Model

- Rather than giving the model a history as input, it should *learn* to memorize relevant information, i.e., learn a latent representation for relevant information → recurrent NN
- Train a latent representation h_t to consume history information and predict y_t



(Wikipedia; change in notation: $x \rightsquigarrow (y, u), o \rightsquigarrow y$)

- The most common NN architecture is LSTM (better: Gated Recurrent Units):



(Hochreiter, Schmidthuber, 1997)

Observation-based Dynamics – State-Space Model

- Also a recurrent model, but explicitly assumes latent state $x_t \in \mathbb{R}^d$

Probabilistic Recurrent State-Space Models

Andreas Doerr [1 2] **Christian Daniel** [1] **Martin Schiegg** [1] **Duy Nguyen-Tuong** [1] **Stefan Schaal** [2 3] **Marc Toussaint** [4] **Sebastian Trimpe** [2]

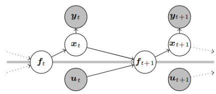


Figure 1. Graphical model of the PR-SSM. Gray nodes are observed variables in contrast to latent variables in white nodes. Thick lines indicate variables, which are jointly Gaussian under a GP prior.

Andreas Doerr, Christian Daniel, Martin Schiegg, Nguyen-Tuong Duy, Stefan Schaal, Marc Toussaint, and Trimpe Sebastian, (2018). Probabilistic recurrent state-space models. In *International conference on machine learning*, pages 1280–1289

Observation-based Dynamics – State-Space Model

- Jointly train an embedding/decoding $g : x \mapsto y$ and latent dynamics $f : x, u \mapsto x'$:

$$\begin{array}{ccc} x & , \mathbf{u} \overset{f}{\mapsto} & x' \\ g\downarrow & & g\downarrow \\ \mathbf{y} & & \mathbf{y}' \end{array}$$

- Only $u_{1:T}, y_{1:T}$ are observed! Train model to maximize data likelihood,

$$\log p(y_{1:T} \mid u_{1:T}) \geq \text{Evidence Lower Bound (ELBO)}$$

  - This method trains both, g and f, and implicitly *infers* a notion of state x_t
  - Technically, use SGD to maximize ELBO

- More Literature for the six cases provided at the end of these slides...

II. How is the data generated?

- Importance of data generation is (mostly) under-acknowledged in papers!

- Ideas to generate *good* data may be more important than ML method details

- What is *good* data?

1.5:19

Good Data – in Linear Regression

- Reconsider regression with linear model $f_\theta(x) = \bar{x}^\top\theta$, loss

$$L(\theta) = \sum_i (y_i - f_\theta(x_i))^2 \ + \ \lambda\|\theta\|^2$$

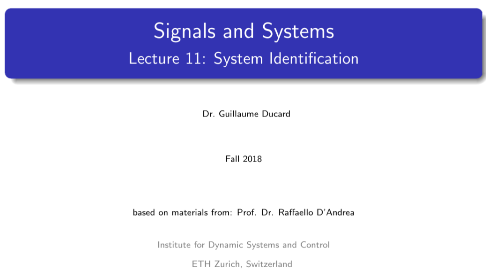
and solution

$$\theta^* = (X^\top X + \lambda\mathbf{I})^{-1} X^\top y \ .$$

- What is good data?
- What is the estimator variance Var$\{\theta^*\}$?
  - Assume data with variance Var$\{y\} = \sigma^2\mathbf{I}_n$
  - Then Var$\{\theta^*\} = (X^\top X + \lambda I)^{-1}\sigma^2$
  - Smaller variance via larger λ (but then larger bias), or **larger** $\det(X^\top X)$!
- Good data means reducing variance (=randomness) of estimated model!
  - large $\det(X^\top X) \leftrightarrow$ cover input space!
    [Large estimator variance $\leftrightarrow$ "Overfitting": Reducing variance prevents overfitting. Hastie has great section on *shrinkage* methods (=regularization)]

1.5:20

Good Data – in Linear System Identification



https://ethz.ch/content/dam/ethz/special-interest/mavt/dynamic-systems-n-control/idsc-dam/Lectures/Signals-and-System
Lectures/Fall2018/Lecture11_sigsys.pdf

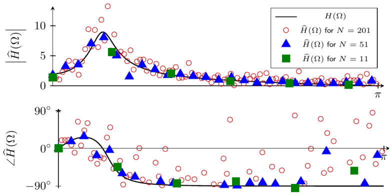
1.5:21

Good Data – in Linear System Identification

- Cover the input space $\to$ cover frequency space
  - Linear dynamics can be Laplace transformed into frequency domain:

$$Y(s) = H(s)\;U(s)$$

  - $U(s)$ are controls; Y observations; $H(s)$ is called **transfer function** (complex)
  - $H(s)$ can be probed by sending a single control frequence ($U(s) = \delta_{ss'}$)



- In essence: stimulate the system with control frequencies $u(t) = \cos(kt/\tau_0)$ for $k = 0, 1, ..$
- Franka SystemId paper [Gaz'19]: Sinusoidal reference motions (Eq. 31):

$$\dot{q}_{i,\text{des}(t)} = A_i \sin\left(\frac{2\pi}{T_i}\;t\right)\;,\quad i \in \{1, .., n\}$$

1.5:22

Good Data – in general

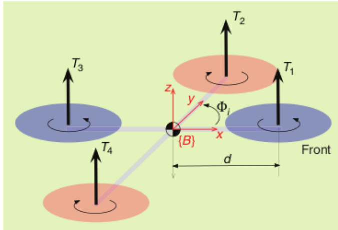
- Think about good state space coverage! (in all variants of Robot Learning)
  - Frequency coverage in control systems
  - Exploration in RL beyond ϵ-greedy
  - Long-term structured variation (at least pink noise, Ornstein-Uhlenbeck) instead of Brownian motion
  - Explicit exploration: Novelty seeking, information seeking, exploration bonus, Bayesian RL

1.5:23

III. Background: Multirotors

- State $\mathbf{x} = (\mathbf{p}, \mathbf{q}, \mathbf{v}, \omega)^\top$
- Control $\mathbf{u}_\Omega = (\Omega_1, \dots, \Omega_n)^\top$
- Forces $\mathbf{f} = \sum_i c_{f_i} \Omega_i \mathbf{z}_{\Omega_i} = \mathbf{F}\mathbf{u}_\Omega$,
- Torques $\boldsymbol{\tau} = \sum_i (c_{f_i} \mathbf{p}_{\Omega_i} \times \mathbf{z}_{\Omega_i} + c_{\tau_i} \mathbf{z}_{\Omega_i}) \Omega_i = \mathbf{M}\mathbf{u}_\Omega$
- Dynamics

$$\dot{\mathbf{p}} = \mathbf{v}, \qquad m\dot{\mathbf{v}} = m\mathbf{g} + \mathbf{R}(\mathbf{q})\mathbf{F}\mathbf{u}_\Omega + \mathbf{f}_a,$$
$$\dot{\mathbf{q}} = \frac{1}{2}\mathbf{q} \circ \begin{bmatrix} 0 \\ \boldsymbol{\omega} \end{bmatrix}, \mathbf{J}\dot{\boldsymbol{\omega}} = -\boldsymbol{\omega} \times \mathbf{J}\boldsymbol{\omega} + \mathbf{M}\mathbf{u}_\Omega + \boldsymbol{\tau}_a,$$



[Mahony, ∼2012]

[Propellers create forces and torques, rest is Newton-Euler]

[$\mathbf{f}_a$, $\boldsymbol{\tau}_a$ can model drag, wind, aerodynamic interactions etc.]

1.5:24

Multirotors: What is learned?

- Parameters that are hard to measure: inertia $\mathbf{J}$, motor params (c_{f_i}, c_{τ_i}, delay)

- Residuals $\mathbf{f}_a$, $\boldsymbol{\tau}_a$

  [potentially as a function of the state (e.g., drag) or environment (e.g., downwash)]

  [potentially non-Markovian, i.e., a function of a history of states]

- Full dynamics model not so much — Why?

  [Impossible to gather data from all states safely]

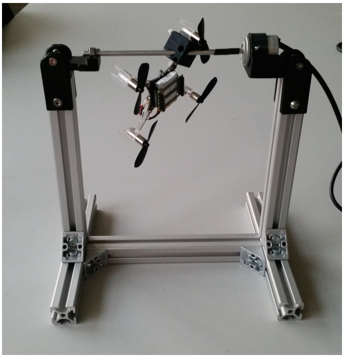
  [Rotational symmetries are surprisingly difficult to learn]

1.5:25

Multirotors: How is it "learned"? (Classic)

Estimate parameters with dedicated experiments

- Inertia: Swing body in different positions and record motion; solve an optimization problem



1.5:26

Multirotors: How is it “learned”? (Classic)

Estimate parameters with dedicated experiments

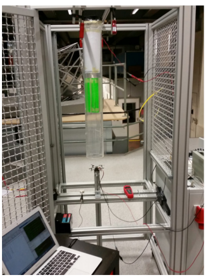
- Motors: Use thrust stand (often for a single motor + propeller) + curve fitting

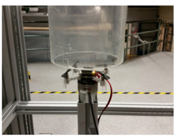
1.5:27

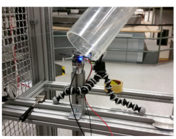
Multirotors: How is it “learned”? (Classic)

Estimate parameters with dedicated experiments

- Drag: Use wind tunnel + curve fitting with “guessed” models







Julian Förster, (2015). System identification of the crazyflie 2.0 nano quadrocopter

1.5:28

Multirotors: How is it "learned"? (Classic)

Estimate parameters with dedicated experiments

- Is this learning?

  [Yes, since curve fitting is extensively used]

- Advantages and Disadvantages?

  [Pros: Physics intuition (explainability); can improve "important" parameters if needed; no need to have a flying system]

  [Cons: Labor and equipment intensive; does not capture unmodeled terms; does not capture the robot as a system]

1.5:29

Multirotors: How is it learned? (Parameter Estimation)

- Assumption: we have a system that can already fly; Can we do better?

  [Strong assumption, since controllers need models, too]

- Direct (analytical) optimization

  Jonas Eschmann, Dario Albani, and Giuseppe Loianno, (2024). Data-driven system identification of quadrotors subject to motor delays

  [Will skip the discussion here]

- Probabilistic formulation (Gaussian noise)

  Michael Burri, Janosch Nikolic, Helen Oleynikova, Markus W. Achtelik, and Roland Siegwart, (2016). Maximum likelihood parameter identification for MAVs. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4297–4303

1.5:30

Multirotors: How is it learned? (Maximum Likelihood)

- Given: Dataset with trajectory (position, orientation, motor speed), $\mathbf{Z}$; measurements (IMU data, motor commands), $\mathbf{U}$
- Goal:

$$\hat{\mathbf{X}}_{ML}, \hat{\theta}_{ML} = \underset{\hat{\mathbf{X}}, \hat{\theta}}{\operatorname{argmax}}\, p(\mathbf{Z}, \mathbf{U}, \hat{\mathbf{X}}, \hat{\theta})$$

(parameters to estimate $\hat{\theta}$; state estimates $\hat{\mathbf{X}}$; probability p)

1.5:31

Multirotors: How is it learned? (Maximum Likelihood)

- Assumptions to simplify $p(\mathbf{Z}, \mathbf{U}, \hat{\mathbf{X}}, \hat{\theta})$
  - White noise (IMU, motors)

  - Access to a prior trajectory → linearize around it and reason about "residuals" instead
- $p(\cdot)$ becomes a mixture of Gaussians → can be maximized by minimizing the negative log-likelihood

  [essentially a least square problem]

1.5:32

Multirotors: How is it learned? (Maximum Likelihood)

$n := 0$
$\bar{\boldsymbol{y}} :=$ InitializeEstimator ()
% Solve ML problem
while $n < n_{max}$ **do**
  $\boldsymbol{b}, \boldsymbol{A} :=$ EvaluateResiduals $(\bar{\boldsymbol{y}})$
  $\delta\boldsymbol{y} :=$ SolveLeastSquaresProblem $(\boldsymbol{b}, \boldsymbol{A})$
  $\bar{\boldsymbol{y}} = \bar{\boldsymbol{y}} \boxplus \delta\boldsymbol{y}$
$\boldsymbol{\theta}^* :=$ ExtractParameters $(\bar{\boldsymbol{y}})$
$\boldsymbol{\Sigma}_\theta :=$ RecoverParameterCovariance $(\boldsymbol{A})$
return $\boldsymbol{\theta}^*, \boldsymbol{\Sigma}_\theta$

where $\bar{y} = (\hat{\mathbf{X}}, \hat{\theta})^\top$ from before

Michael Burri, Janosch Nikolic, Helen Oleynikova, Markus W. Achtelik, and Roland Siegwart, (2016). Maximum likelihood parameter identification for MAVs. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4297–4303

Michael Burri, Michael Bloesch, Zachary Taylor, Roland Siegwart, and Juan Nieto, (2018). A framework for maximum likelihood parameter identification applied on MAVs. *Journal of Field Robotics*, 35(1):5–22

1.5:33

Multirotors: How is it learned? (Supervised Deep NN)

- Basic models do not capture "complicated" aerodynamic effects
- Blade Element Momentum (BEM) work for single rotors (but high computational effort)
- Can we use (more) data to use function approximation instead?
  Challenges:
  - Training/Data efficiency
  - Inference speed

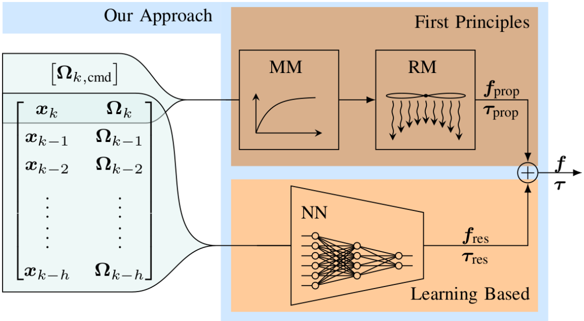
1.5:34

Multirotors: How is it learned? (Supervised Deep NN)

- Key idea: learn the "residual physics", only

  [Input: past h states and motor commands → not Markovian!]

  [Output: forces and torques that cannot be explained by the basic model(s) $(\mathbf{f}_a, \boldsymbol{\tau}_a)$]



1.5:35

Multirotors: How is it learned? (Supervised Deep NN)

- ML method: Supervised training — Where do the labels come frome?

  [Solve dynamics for $\mathbf{f}_a$, $\boldsymbol{\tau}_a$]

- Architecture
  - Input $h = 20$ (past 50 ms)
  - temporal convolutional (TCN) with 25k parameters (MLP and other parameters in ablation)

- Main takeaway: strong model/physics priors are better

  Leonard Bauersfeld, Elia Kaufmann, Philipp Foehn, Sihao Sun, and Davide Scaramuzza, (2021). NeuroBEM: Hybrid aerodynamic quadrotor model. In *Robotics: Science and Systems XVII*, volume 17

  [Video: `https://youtu.be/Nze1wlfmzTQ`]

1.5:36

Multirotors: Data Collection

- Motion capture system for accurate position/orientation state estimates

  [Sampling at 500 Hz, submillimeter accuracy]

  [Very costly: EUR 20k – 100k]
- On-board data logging of IMU

  [Sampling at 1000 Hz, very noisy]

1.5:37

Multirotors: Data Preprocessing

- Two data sources → Synchronization needed (incl. clock skew)

- Online Option: Send data to one computer using a low-latency link (and account for link delay)
- Offline Option: Solve optimization problem for clock skew and bias

- Some derivatives (e.g., $\mathbf{v}$) are not directly observable
  - Online Option: Use data from an online filter (e.g., Extended Kalman Filter)
  - Offline Option: Interpolate data (e.g., using splines), use analytical solution of fitted spline

- Motor delays ("easy" to measure)
  - Option 1: Include it in model explicitly
  - Option 2: Shift/filter data accordingly

1.5:38

Multirotors: Data Quantity

- Maximum Likelihood: 45 sec flight data "The pilot was careful to excite all axes, especially in yaw direction."
- NeuroBEM: 96 flights, 75 min flight data (1.8M data points) (up to 18 m/s and 47 m/s^2)

1.5:39

Literature

- State Dynamics – Parameter Estimation:

Julian Förster, (2015). System identification of the crazyflie 2.0 nano quadrocopter

Jonas Eschmann, Dario Albani, and Giuseppe Loianno, (2024). Data-driven system identification of quadrotors subject to motor delays

Michael Burri, Michael Bloesch, Zachary Taylor, Roland Siegwart, and Juan Nieto, (2018). A framework for maximum likelihood parameter identification applied on MAVs. *Journal of Field Robotics*, 35(1):5–22

Claudio Gaz, Marco Cognetti, Alexander Oliva, Paolo Robuffo Giordano, and Alessandro De Luca, (2019). Dynamic identification of the franka emika panda robot with retrieval of feasible parameters using penalty-based optimization. *IEEE Robotics and Automation Letters*, 4(4):4147–4154

- State Dynamics – Full Regression:

Stefan Schaal, Christopher G. Atkeson, and Sethu Vijayakumar, (2002). Scalable techniques from nonparametric statistics for real time robot learning. *Applied Intelligence*, 17(1):49–60

Marc Peter Deisenroth, Dieter Fox, and Carl Edward Rasmussen, (2015). Gaussian processes for data-efficient learning in robotics and control. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(2):408–423

1.5:40

Literature

- Observation-based Dynamics – Autoregression (NARX):

S. Chen, S. A. Billings, and P. M. Grant, (1990). Non-linear system identification using neural networks. *International Journal of Control*, 51(6):1191–1214

Hava T. Siegelmann, Bill G. Horne, and C. Lee Giles, (1997). Computational capabilities of recurrent NARX neural networks. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 27(2):208–215

- Observation-based Dynamics – Recurrent Model (also visual!):

Leonard Bauersfeld, Elia Kaufmann, Philipp Foehn, Sihao Sun, and Davide Scaramuzza, (2021). NeuroBEM: Hybrid aerodynamic quadrotor model. In *Robotics: Science and Systems XVII*, volume 17

Chelsea Finn and Sergey Levine, (2017). Deep visual foresight for planning robot motion. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2786–2793

Danny Driess, Zhiao Huang, Yunzhu Li, Russ Tedrake, and Marc Toussaint, (2023). Learning multi-object dynamics with compositional neural radiance fields. In *Conference on robot learning*, pages 1755–1768

Ingmar Schubert, Jingwei Zhang, Jake Bruce, Sarah Bechtle, Emilio Parisotto, Martin Riedmiller, Jost Tobias Springenberg, Arunkumar Byravan, Leonard Hasenclever, and Nicolas Heess, (2023). A generalist dynamics model for control

1.5:41

Literature

- State-Space Models (learning a *state* dynamics based on only observations):

Andreas Doerr, Christian Daniel, Martin Schiegg, Nguyen-Tuong Duy, Stefan Schaal, Marc Toussaint, and Trimpe Sebastian, (2018). Probabilistic recurrent state-space models. In *International conference on machine learning*, pages 1280–1289

1.5:42

---

not mentioned...

- Constrained ML models (Geist)
- Embed to Control
- Koopman embedding
- Dual control
- Safe Exploration

1.5:43

1.6 Imitation Learning

(slides by Marc Toussaint)

General Idea

- Given expert demonstration data $D = \{(x^i_{1:T_i}, u^i_{1:T_i})\}_{i=1}^n$

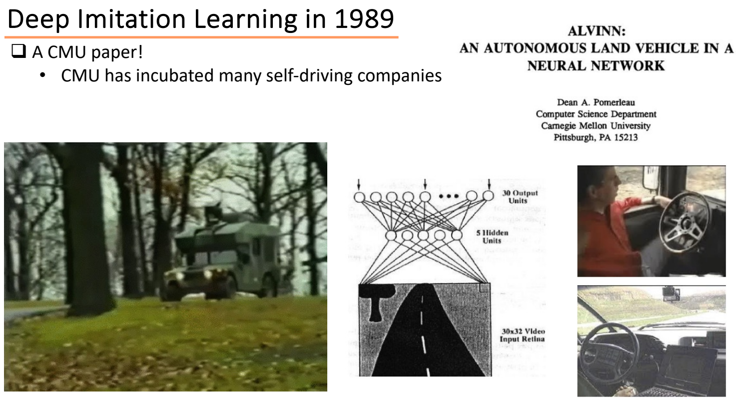
$$\begin{aligned} i &: \quad \text{episode/demonstration} \\ x^i_{1:T_i} &: \quad i\text{th state trajectory} \\ u^i_{1:T_i} &: \quad i\text{th control trajectory} \end{aligned}$$

without external rewards/objectives/costs defined
$\to$ extract the "relevant information/model/policy" to reproduce demonstrations

- Reproducing could mean various things
  - Move along similar trajectories (e.g. imitate a gesture)
  - Reproduce the *effect* of the demonstration (manipulation, flight maneuver, no traffic collisions)

1.6:1

Early Work



(Shi's lecture 5)

https://www.youtube.com/watch?v=ntIczNQKfjQ

1.6:2

Early Work

- Behavior Cloning (later called so):
  Dean A. Pomerleau, (1988). Alvinn: An autonomous land vehicle in a neural network. *Advances in neural information processing systems*, 1
- Early review paper:
  Stefan Schaal, Auke Ijspeert, and Aude Billard, (2003). Computational approaches to motor learning by imitation. *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, 358(1431):537–547

  [clarifies direct policy learning (BC) vs. trajectory imitation (and auto-control); mentiones work from the 60ies, but esp. 90ies]
- Early work named **Learning from Demonstration** (or Programming by Demonstration)
  Christopher G. Atkeson and Stefan Schaal, (1997). Robot learning from demonstration. In *ICML*, volume 97, pages 12–20

  [Idea: Avoid explicit programming → teach by demonstration. See also entries in "Handbook of Robotics"...]
- Another early survey:
  Brenna D. Argall, Sonia Chernova, Manuela Veloso, and Brett Browning, (2009). A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483

  [Distinguishes 3 kinds: behavior cloning, use data to learn dynamics (system identification), learn plans (nowadays uncommon)]

1.6:3

Outline

- Types of Imitation Learning
  – Behavior Cloning
  – Trajectory Distribution Learning (& Constraint Learning)
  – Direct (Interactive) Policy Learning
  – Inverse Reinforcement Learning (not covered today)
- Data Generation

– Distributional (domain) shift, "compound errors" in imitation, on-/off-policy
– Data augmentation or interactive data aggregation
– Collection techniques: Tele-Operation, Kinesthetic Teaching, Human Demonstrations

1.6:4

Behavior Cloning

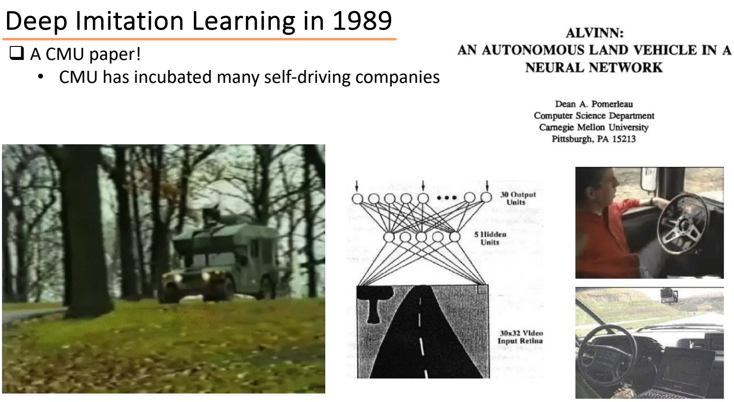
- Formulate Imitation Learning literally as *Supervised ML*
- Given data $D = \{(x^i_{1:T_i}, u^i_{1:T_i})\}_{i=1}^n$, find

$$\min_\theta \sum_{i,t} \ell(u^i_t, \pi_\theta(x^i_t)) \ , \tag{1}$$

where $\pi_\theta : x \mapsto u$ is a deterministic policy (e.g. NN) mapping states to controls

1.6:5

Behavior Cloning



(Shi's lecture 5)

1.6:6

Behavior Cloning

- Behavior Cloning literally imitates the demonstrated mapping $x \mapsto u$

- Issues:
  – But does that also imitate the *long term behavior* or *eventual effect* of the demonstrations? (Ignores distributional shift.)
  – Does it capture the "essence" of what is demonstrated?
  – Can it deal with multi-modal demonstrations? ($\to$ next week: multi-modal policies)

1.6:7

Trajectory Distribution Learning

[This is not common terminology, and seemingly skipped in other Imitation Learning lectures – unfortunately. I think this captures an essence of the problem.]

- What does it mean to capture the "essence" of data?
  - Learn a *distribution model* $p_\theta(x_{1:T})$ of demonstrated trajectories!

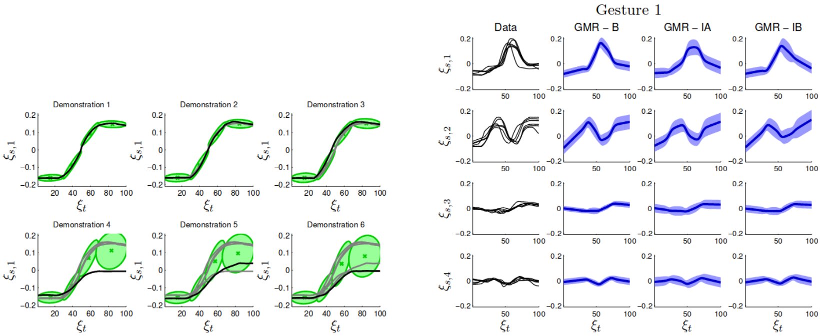
$$\max_\theta \prod_i p_\theta(x^i_{1:T_i}) \quad \text{(likelihood maximization (LM))} \; , \tag{2}$$

  where p_θ is some model class powerful enough to represent "essence"

- What are "powerful" models?
  - Transformer models, diffusion models
  - But we'll start with very basic Gaussian models
  - ...and discuss models specifically for robotic manipulation

1.6:8

Trajectory Distribution Learning: GMMs



Sylvain Calinon and Aude Billard, (2007). Incremental learning of gestures by imitation in a humanoid robot. In *Proceedings of the ACM/IEEE International Conference on Human-robot Interaction*, pages 255–262

- Embed trajectories $x_{1:T}$ in "space-time" $\{(t, x_t)\}_{t=1}^T$
- Fit a density estimator to $p(t, x_t)$ (easiest: Gaussian Mixture Model (GMM), LM well studied)
- Can be translated to control policy by reading out conditional $p(x|t)$ and using inverse dynamics

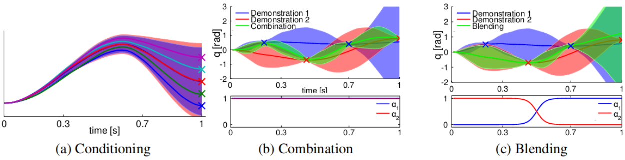
1.6:9

Trajectory Distribution Learning: GMMs

- A simple way to describe the distribution of demonstrated trajectories
- Variance of learned $p(x|t)$ captures "consistent bottlenecks" in demonstrations
  [Is that a key structure in demonstrations? Search also "Calinon constraints"]
- Can be combined with Dynamic Time Warping to temporally align demonstrations
- GMM approach is around for ~ 20 years

1.6:10

Trajectory Distribution Learning: ProMPs



We use a weight vector $\boldsymbol{w}$ to compactly represent a single trajectory. The probability of observing a trajectory $\boldsymbol{\tau}$ given the underlying weight vector $\boldsymbol{w}$ is given as a linear basis function model

$$\boldsymbol{y}_t = \begin{bmatrix} q_t \\ \dot{q}_t \end{bmatrix} = \boldsymbol{\Phi}_t^T \boldsymbol{w} + \boldsymbol{\epsilon}_y, \qquad p(\boldsymbol{\tau}|\boldsymbol{w}) = \prod_t \mathcal{N}\left(\boldsymbol{y}_t | \boldsymbol{\Phi}_t^T \boldsymbol{w}, \boldsymbol{\Sigma}_y\right), \quad (1)$$

Alexandros Paraschos, Christian Daniel, Jan R. Peters, and Gerhard Neumann, (2013). Probabilistic movement primitives. *Advances in neural information processing systems*, 26

– Nothing but (prob.) linear regression $t \mapsto x_t$ with basis function features (LM↔regression)
– Very simple distribution model over trajectories [could use GPs to kernelize]
– Related to Inference Control (AICO, ICML'09), Path Integral methods (RSS'12)
– Great flexibility to condition, compose, and blend
– Somewhat superseeds earlier work on learning movement primitives from demonstration [typically Dynamic Movement Primitives (DMPs, Schaal et al'03)]

1.6:11

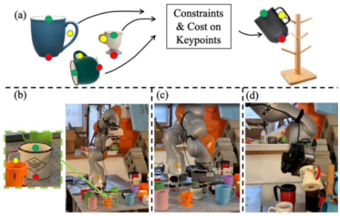
Trajectory Distribution Learning: Features & Constraints

• Think about Manipulation!

kPAM: KeyPoint Affordances for Category-Level Robotic Manipulation

Lucas Manuelli*, Wei Gao*, Peter Florence, Russ Tedrake

CSAIL, Massachusetts Institute of Technology,
{manuelli, weigao, peteflo, russt}@mit.edu
These authors contributed equally to this work.



Lucas Manuelli, Wei Gao, Peter Florence, and Russ Tedrake, (2022). KPAM: KeyPoint Affordances for Category-Level Robotic Manipulation. In Tamim Asfour, Eiichi Yoshida, Jaeheung Park, Henrik Christensen, and Oussama Khatib, editors, *Robotics Research*, volume 20, pages 132–157

1.6:12

Trajectory Distribution Learning: Features & Constraints

• Think about Manipulation!

Neural Descriptor Fields:
SE(3)-Equivariant Object Representations for Manipulation

Anthony Simeonov*[,1], Yilun Du*[,1], Andrea Tagliasacchi[2,3],
Joshua B. Tenenbaum[1], Alberto Rodriguez[1], Pulkit Agrawal†[,1], Vincent Sitzmann†[,1]
[1]Massachusetts Institute of Technology [2]Google Research [3]University of Toronto
*Authors contributed equally, order determined by coin flip. †Equal Advising.

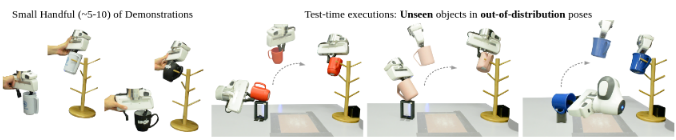


Fig. 1: Given a few (~5-10) demonstrations of a manipulation task (left), Neural Descriptor Fields (NDFs) generalize the task to novel object instances in any 6-DoF configuration, *including those unobserved at training time*, such as mugs with arbitrary 3D translation and rotation (right). NDFs are continuous functions that map 3D spatial coordinates to spatial descriptors. We generalize this to functions which encode SE(3) poses, such as those used for grasping and placing. NDFs are trained self-supervised for the surrogate task of 3D reconstruction, do not require labeled keypoints, and are SE(3)-equivariant, guaranteeing generalization to unseen object configurations.

Anthony Simeonov, Yilun Du, Andrea Tagliasacchi, Joshua B. Tenenbaum, Alberto Rodriguez, Pulkit Agrawal, and Vincent Sitzmann, (2022). Neural descriptor fields: Se (3)-equivariant object representations for manipulation. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 6394–6400

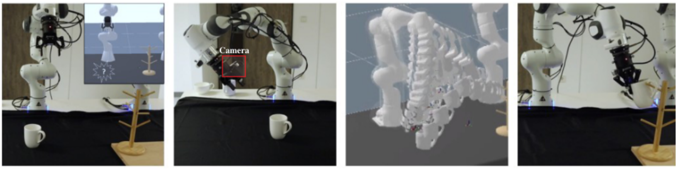
1.6:13

Trajectory Distribution Learning: Features & Constraints

- Think about Manipulation!

Deep Visual Constraints: Neural Implicit Models for Manipulation Planning from Visual Input

Jung-Su Ha Danny Driess Marc Toussaint
Learning & Intelligent Systems Lab, TU Berlin, Germany



(a) No object model (b) See (c) Plan (d) Act

Jung-Su Ha, Danny Driess, and Marc Toussaint, (2022). Deep visual constraints: Neural implicit models for manipulation planning from visual input. *IEEE Robotics and Automation Letters*, 7(4):10857–10864

1.6:14

Trajectory Distribution Learning: Features & Constraints

- Connects to large body of literature:
  - More examples: FlowBot3D, UMPNet, Bi-KVIL, "Waypoint-based imitation learning", ..
  - Human Activity Modelling, Action Segmentation:

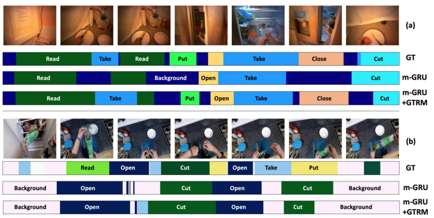


Figure 4. Qualitative comparison of results for action segmentation task on (a) EGTEA, and (b) EPIC dataset. Only part of the whole video is shown for clarity. We can see in (a) that the *take*, *put* and *close* actions are correctly detected by adding GTRM.

- What really is the essence to extract from demonstrations?

1.6:15

- Back to Behavior Cloning...

- Issues:
  - But does that also imitate the *long term behavior* or *eventual effect* of the demonstrations? **(Ignores distributional shift.)**
  - Does it capture the "essence" of what is demonstrated?

1.6:16

Distributional (Domain) Shift

- Standard ML: $x, y \sim p(x, y)$ **i.i.d.**; same p for trains & test
- Sequential Decision Processes: own policy π influences test distrib. $p_\pi(x_t)$!
  - Fundamental difference between learning in sequential decision processes and Supervised ML!
  - Also in off-policy & offline RL: We *train* a policy (or Q, V-function) with losses relative to $p_{\pi_\beta}(x_t)$ with *behavior policy* (π_β)
  - Generally called distributional shift, or Out-of-Distribution (OOD) testing

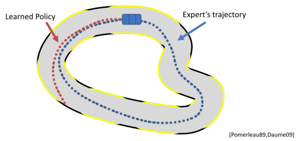
1.6:17

Distributional Shift in Behavior Cloning

- When we train policy π_θ in BC, we minimize

$$\min_\theta \sum_{i,t} \ell(u_t^i, \pi_\theta(x_t^i)) \;\leftrightarrow\; \min_\theta \mathbb{E}_{\pi^*}\{\ell(u, \pi_\theta(x))\} \tag{3}$$

but when using the policy, we generate fully different distribution



Also called **Compound Error** (Shi's lecture 5)

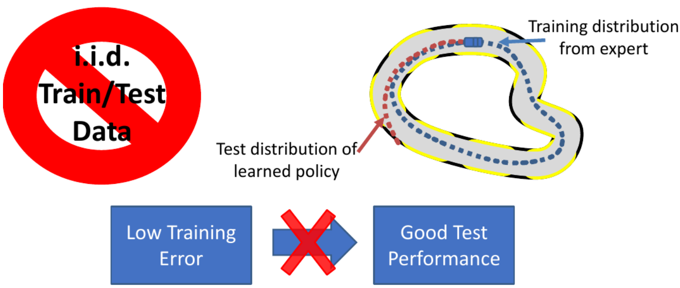
- What we should train is this:!

$$\min_\theta \mathbb{E}_{\pi_\theta}\{\ell(\pi^*(x), \pi_\theta(x))\} \tag{4}$$

1.6:18

Distributional Shift in Behavior Cloning

- BC formulates a supervised ML problem, but in view of testing, it is not:



(Shi's lecture 5)

1.6:19

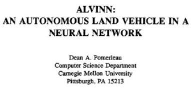
How address the Distributional Shift?

- Ensure the data better covers the eventual $p_\pi(x_t)$ of trained π
  - Enforce the expert to demonstrate also for non-optimal states (cover also non-expert situations)
  - Collect data interactively at exactly the states visited by π (DAgger)

1.6:20

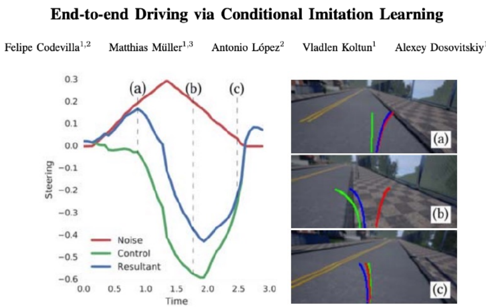
Enforcing wider expert demonstrations

- Occasionally perturb the expert! Add noise!


ALVINN:
AN AUTONOMOUS LAND VEHICLE IN A
NEURAL NETWORK

Dean A. Pomerleau
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213

"...the network must not solely be shown examples of accurate driving, but also how to recover (i.e. return to the road center) once a mistake has been made."



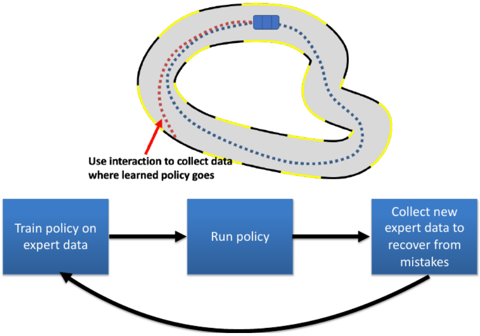
(Shi's lecture 5)

1.6:21

DAgger

Initialize $\mathcal{D} \leftarrow \emptyset$.
Initialize $\hat{\pi}_1$ to any policy in Π.
for $i = 1$ **to** N **do**
  Let $\pi_i = \beta_i \pi^* + (1 - \beta_i)\hat{\pi}_i$.
  Sample T-step trajectories using π_i.
  Get dataset $\mathcal{D}_i = \{(s, \pi^*(s))\}$ of visited states by π_i and actions given by expert.
  Aggregate datasets: $\mathcal{D} \leftarrow \mathcal{D} \bigcup \mathcal{D}_i$.
  Train classifier $\hat{\pi}_{i+1}$ on $\mathcal{D}$.
end for
Return best $\hat{\pi}_i$ on validation.

Algorithm 3.1: DAGGER Algorithm.



Stephane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell, (2011). A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning

https://www.youtube.com/watch?v=V00npNnWzSU

- This repeatedly collects data from the current π, to approximate $\min_\theta \mathbb{E}_\pi\{\ell(\pi^*(x_t), \pi_\theta(x_t))\}$

1.6:22

- From Yue's ICML'18 tutorial:

	Direct Policy Learning	Reward Learning	Access to Environment	Interactive Demonstrator	Pre-collected Demonstrations
Behavioral Cloning	Yes	No	No	No	Yes
Direct Policy Learning (Interactive IL)	Yes	No	Yes	Yes	Optional
Inverse Reinforcement Learning	No	Yes	Yes	No	Yes

- Crucial point: For DAgger we have a very different setting: Access to the environment (testing rollouts), interactively querying the expert.

1.6:23

Data Collection

1.6:24

Data Collection

- We've covered the theoretical aspect concerning distributional shift
- Data source:
  - Tele-Operation
  - Kinesthetic Teaching
  - Human Demonstrations & Motion Capture
  - Videos Only

1.6:25

Tele-Operation: Aloha

Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware

Tony Z. Zhao[1] Vikash Kumar[3] Sergey Levine[2] Chelsea Finn[1]
[1] Stanford University [2] UC Berkeley [3] Meta

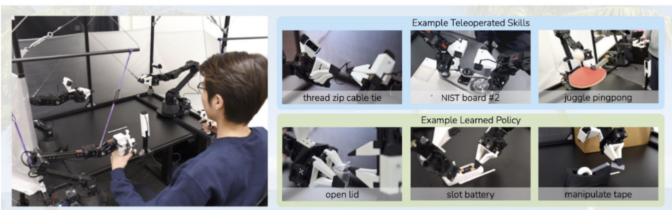


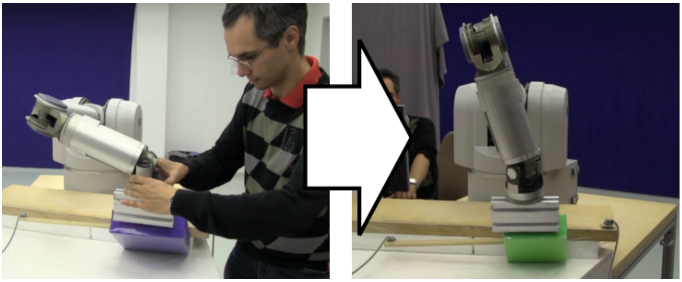
Fig. 1: *ALOHA* : *A Low-cost Open-source Hardware System for Bimanual Teleoperation*. The whole system costs <$20k with off-the-shelf robots and 3D printed components. *Left:* The user teleoperates by backdriving the leader robots, with the follower robots mirroring the motion. *Right: ALOHA* is capable of precise, contact-rich, and dynamic tasks. We show examples of both teleoperated and learned skills.

Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn, (2023). Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware

https://tonyzhaozh.github.io/aloha/

1.6:26

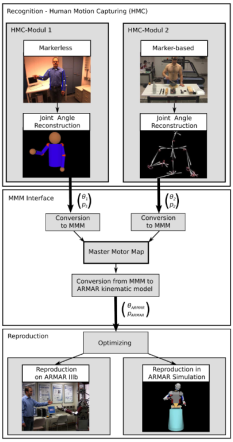
Kinesthetic Teaching



Learning movement primitives for force interaction tasks (Kober et al'15)

1.6:27

Human Demonstrations & Motion Capture



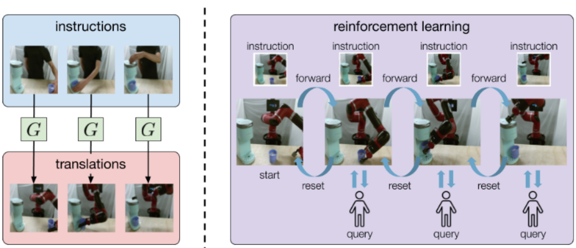
Martin Do, Pedram Azad, Tamim Asfour, and Rudiger Dillmann, (2008). Imitation of human motion on a humanoid robot using non-linear optimization. In *Humanoids 2008-8th IEEE-RAS International Conference on Humanoid Robots*, pages 545–552

1.6:28

Human Demonstrations From Video Only

AVID: Learning Multi-Stage Tasks via Pixel-Level Translation of Human Videos

Laura Smith, Nikita Dhawan, Marvin Zhang, Pieter Abbeel, and Sergey Levine
Berkeley Artificial Intelligence Research, Berkeley, CA, 94720
Email: `smithlaura@berkeley.edu`



Laura Smith, Nikita Dhawan, Marvin Zhang, Pieter Abbeel, and Sergey Levine, (2020). AVID: Learning Multi-Stage Tasks via Pixel-Level Translation of Human Videos

1.6:29

- This whole lecture talked about states! Same for observations y_t only!
  - History-input policies (analogous to autoregressive dynamics)
  - Recursive (RNN) policies (analogous to recursive dynamics)
  - Transformer policies (sequence models)

1.6:30

1.7 Imitation Learning 2

(slides by Wolfgang Hönig)

Recap

- Imitation Learning
  - Given: expert demonstration data $D = \{(x^i_{1:T_i}, u^i_{1:T_i})\}_{i=1}^n$
  - Goal: reproduce demonstrations
- Main Challenges:
  - Distributional Domain Shift Solutions:
    - Behavior Cloning: add noise
    - DAgger: interactively add additional *expert* data
    - Trajectory Distribution Learning: rely on controller
  - Data Collection Solutions:
    - Humans: teleoperation, kinesthetic teaching, motion capture, videos
    - **high-effort computations** (w.r.t. to computation or observation), e.g., *Privileged Teacher*

1.7:1

Outline Today

- Data Collection: Privileged Teacher
- Generative Models
- Case Studies
  - Quadrotor Acrobatics
  - Learning from ALOHA data
  - Transfer Learning

1.7:2

Privileged Teacher

- So far we considered to directly learn $\pi_\theta : x \mapsto u$ (or $\pi_\theta : y \mapsto u$)
- y might be high-dimensional or unstructured (e.g., RGBD sequences)
- Key insight: First learn *privileged* policy ("teacher"); use it to generate data for the "student"

  (i) Learn $\pi_{\theta_1} : z \mapsto u$ (where z contains some "ground truth" data, e.g., states, traffic lights, neighbor behavior)

  (ii) Use π_{θ_1} to generate data $D = \{(x^i_{1:T_i}, u^i_{1:T_i})\}_{i=1}^n$

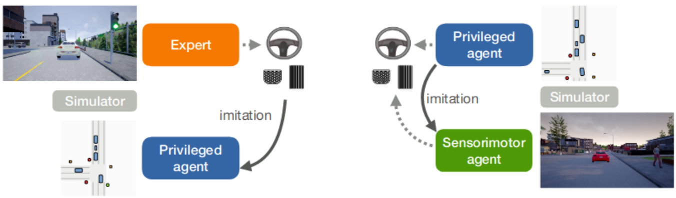
  (iii) Learn $\pi_{\theta_2} : x \mapsto u$

1.7:3

Privileged Teacher

Learning by Cheating

Dian Chen UT Austin
Brady Zhou Intel Labs, UT Austin
Vladlen Koltun Intel Labs
Philipp Krähenbühl UT Austin



(a) Privileged agent imitates the expert (b) Sensorimotor agent imitates the privileged agent

Dian Chen, Brady Zhou, Vladlen Koltun, and Philipp Krähenbühl, (2020). Learning by cheating. In *Conference on Robot Learning*, pages 66–75

https://youtu.be/u9ZCxxD-UUw

1.7:4

Privileged Teacher

- Pros and Cons compared to one-stage IL?

Pros:

- Second stage can be easily trained with DAgger
- Data augmentation simple

Cons

- Simulation-focused
- Hierarchical approach (requires domain knowledge)

1.7:5

Generative Models

- Generative Model:
  - Input: Data $D = \{d^i\}_{i=1}^n$
  - Learning: find distribution p_θ such that $d^i \sim p_\theta$
  - Inference: generate novel data $d^* \sim p_\theta$

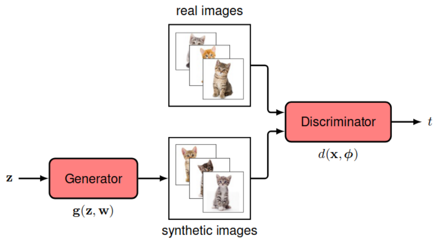
- What generative models do you know? [GAN, VAE, Diffusion, for details see:]

  Christopher M. Bishop and Hugh Bishop, (2024). *Deep Learning: Foundations and Concepts*

- Relationship to IL
  - If $D = \{(x^i_{1:T_i}, u^i_{1:T_i})\}_{i=1}^n$, we can learn *conditional* distribution $p_\theta(u_t|x_t)$
  - Can also generate solution trajectories (esp. in combination with "classic" methods)

Generative Adverserial Network (GAN)

- Train two networks (generator and discriminator)

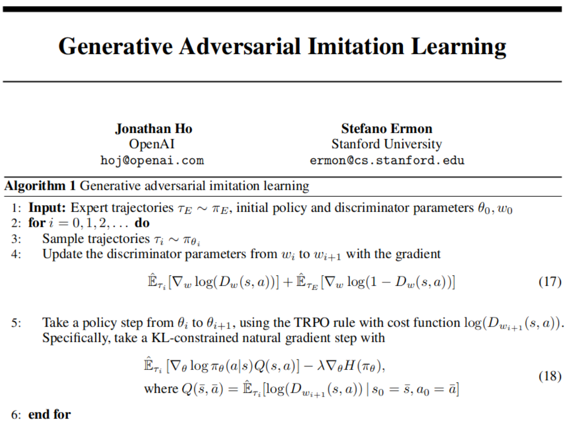


Christopher M. Bishop and Hugh Bishop, (2024). *Deep Learning: Foundations and Concepts*
Lilian Weng, (2017-08-20T00:00:00+00:00). From GAN to WGAN

- Loss function (d_ϕ should be 1 for real data):

$$\max_\omega \min_\phi -\frac{1}{N_{data}} \sum_{n \in \text{data}} \ln d_\phi(x_n) - \frac{1}{N_{gen}} \sum_{n \in \text{gen}} \ln(1 - d_\phi(g_\omega(z_n)))$$

GAN + Imitation Learning = (GAIL)



Generative Adversarial Imitation Learning

Jonathan Ho
OpenAI
hoj@openai.com

Stefano Ermon
Stanford University
ermon@cs.stanford.edu

Algorithm 1 Generative adversarial imitation learning

Input: Expert trajectories $\tau_E \sim \pi_E$, initial policy and discriminator parameters θ_0, w_0
for $i = 0, 1, 2, \dots$ **do**
Sample trajectories $\tau_i \sim \pi_{\theta_i}$
Update the discriminator parameters from w_i to w_{i+1} with the gradient

$$\hat{\mathbb{E}}_{\tau_i}[\nabla_w \log(D_w(s,a))] + \hat{\mathbb{E}}_{\tau_E}[\nabla_w \log(1 - D_w(s,a))] \quad (17)$$

Take a policy step from θ_i to θ_{i+1}, using the TRPO rule with cost function $\log(D_{w_{i+1}}(s,a))$. Specifically, take a KL-constrained natural gradient step with

$$\hat{\mathbb{E}}_{\tau_i}[\nabla_\theta \log \pi_\theta(a|s) Q(s,a)] - \lambda \nabla_\theta H(\pi_\theta), \quad \text{where } Q(\bar{s}, \bar{a}) = \hat{\mathbb{E}}_{\tau_i}[\log(D_{w_{i+1}}(s,a)) \mid s_0 = \bar{s}, a_0 = \bar{a}] \quad (18)$$

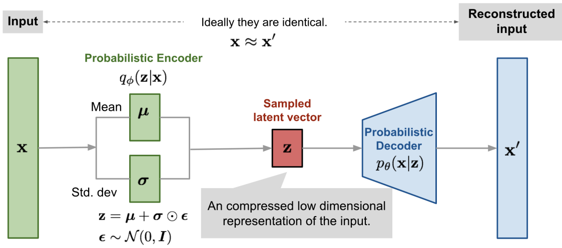
end for

- Generator is a policy $x \mapsto u$
- Discriminator has x, u as input
- Steps:
  - (i) **Rollout/Sample trajectories using generator (=policy)**
  - (ii) Update discriminator
  - (iii) Update policy

Jonathan Ho and Stefano Ermon, (2016). Generative Adversarial Imitation Learning. In *Advances in Neural Information Processing Systems*, volume 29

Variational Autoencoder (VAE)

- Train two networks (encoder and decoder)



Christopher M. Bishop and Hugh Bishop, (2024). *Deep Learning: Foundations and Concepts*
Lilian Weng, (2018-08-12T00:00:00+00:00). From Autoencoder to Beta-VAE
Stanley H. Chan, (2024). Tutorial on Diffusion Models for Imaging and Vision

ML Lecture, slides 8 and 9

- Loss function:

$$\min_{\theta,\phi} -\mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z}) + D_{\mathsf{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \,|\, p_\theta(\mathbf{z}))$$

1.7:9

Variational Autoencoder (VAE)

- Training: SGD Updates for both networks

repeat
  $\mathcal{L} \leftarrow 0$
  for $j \in \{1, \ldots, M\}$ **do**
    $\epsilon_{nj} \sim \mathcal{N}(0,1)$
    $z_{nj} \leftarrow \mu_j(\mathbf{x}_n, \boldsymbol{\phi})\epsilon_{nj} + \sigma_j^2(\mathbf{x}_n, \boldsymbol{\phi})$
    $\mathcal{L} \leftarrow \mathcal{L} + \frac{1}{2}\left\{1 + \ln \sigma_{nj}^2 - \mu_{nj}^2 - \sigma_{nj}^2\right\}$
  end for
  $\mathcal{L} \leftarrow \mathcal{L} + \ln p(\mathbf{x}_n|\mathbf{z}_n, \mathbf{w})$
  $\mathbf{w} \leftarrow \mathbf{w} + \eta \nabla_{\mathbf{w}} \mathcal{L}$ `// Update decoder weights`
  $\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} + \eta \nabla_{\boldsymbol{\phi}} \mathcal{L}$ `// Update encoder weights`
until converged
return $\mathbf{w}, \boldsymbol{\phi}$

[There is an error in the Bishop book (Alg. 19.1): μ and σ are swapped at the highlighted line]

- Inference: Sample from Normal distribution and execute decoder

1.7:10

Variational Autoencoder (VAE) + Imitation Learning

2018 IEEE International Conference on Robotics and Automation (ICRA)
May 21-25, 2018, Brisbane, Australia

Learning Sampling Distributions for Robot Motion Planning

Brian Ichter*[,1], James Harrison*[,2], Marco Pavone[1]

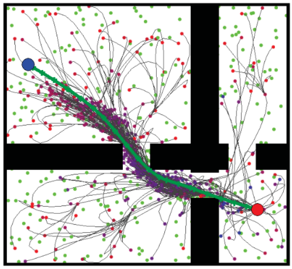
Learning Sample Distribution Methodology Outline

Offline:

1 **Input:** Data (successful motion plans, robot in action, human demonstration, etc.)
2 Construct conditioning variables y
3 Train CVAE, as in Fig. 2a

Online:

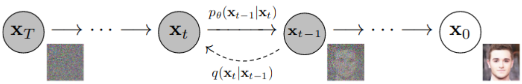
4 **Input:** Motion planning problem $(\mathcal{X}_{\text{free}}, x_{\text{init}}, \mathcal{X}_{\text{goal}})$, learned sample fraction λ
5 Construct conditioning variable y
6 Generate λN free samples from the CVAE latent space conditioned on y, as in Fig. 2b
7 Generate $(1-\lambda)N$ free samples from an auxiliary (uniform) sampler
8 Run sampling-based planner (e.g., PRM*, FMT*, RRT*)



Brian Ichter, James Harrison, and Marco Pavone, (2018). Learning Sampling Distributions for Robot Motion Planning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7087–7094

Diffusion

- Train one network that "removes" noise



Forward diffusion process: sample $\mathbf{x}_0$ and add iid Gaussian noise

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1})$$

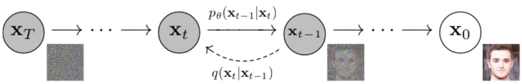
$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \mathbf{I})$$

Christopher M. Bishop and Hugh Bishop, (2024). *Deep Learning: Foundations and Concepts*
Lilian Weng, (2021-07-11T00:00:00+00:00). What are Diffusion Models?
Stanley H. Chan, (2024). Tutorial on Diffusion Models for Imaging and Vision
Jonathan Ho, Ajay Jain, and Pieter Abbeel, (2020). Denoising Diffusion Probabilistic Models. In *Advances in Neural Information Processing Systems*, volume 33, pages 6840–6851

ML Lecture, slide 11

Diffusion

- Train one network that "removes" noise



Reverse process: learn $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$

$$p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$$

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t))$$

Christopher M. Bishop and Hugh Bishop, (2024). *Deep Learning: Foundations and Concepts*
Lilian Weng, (2021-07-11T00:00:00+00:00). What are Diffusion Models?
Stanley H. Chan, (2024). Tutorial on Diffusion Models for Imaging and Vision
Jonathan Ho, Ajay Jain, and Pieter Abbeel, (2020). Denoising Diffusion Probabilistic Models. In *Advances in Neural Information Processing Systems*, volume 33, pages 6840–6851

ML Lecture, slide 11

Diffusion: Training

Algorithm 20.1: Training a denoising diffusion probabilistic model

Input: Training data $\mathcal{D} = \{\mathbf{x}_n\}$
Noise schedule $\{\beta_1, \ldots, \beta_T\}$
Output: Network parameters $\mathbf{w}$

for $t \in \{1, \ldots, T\}$ **do**
  $\alpha_t \leftarrow \prod_{\tau=1}^{t}(1-\beta_\tau)$ `// Calculate alphas from betas`
end for
repeat
  $\mathbf{x} \sim \mathcal{D}$ `// Sample a data point`
  $t \sim \{1, \ldots, T\}$ `// Sample a point along the Markov chain`
  $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{\epsilon}|\mathbf{0}, \mathbf{I})$ `// Sample a noise vector`
  $\mathbf{z}_t \leftarrow \sqrt{\alpha_t}\mathbf{x} + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}$ `// Evaluate noisy latent variable`
  $\mathcal{L}(\mathbf{w}) \leftarrow \|\mathbf{g}(\mathbf{z}_t, \mathbf{w}, t) - \boldsymbol{\epsilon}\|^2$ `// Compute loss term`
  Take optimizer step
until converged
return $\mathbf{w}$

1.7:14

Diffusion: Sampling

Algorithm 20.2: Sampling from a denoising diffusion probabilistic model

Input: Trained denoising network $\mathbf{g}(\mathbf{z}, \mathbf{w}, t)$
Noise schedule $\{\beta_1, \ldots, \beta_T\}$
Output: Sample vector $\mathbf{x}$ in data space

$\mathbf{z}_T \sim \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I})$ `// Sample from final latent space`
for $t \in T, \ldots, 2$ **do**
  $\alpha_t \leftarrow \prod_{\tau=1}^{t}(1-\beta_\tau)$ `// Calculate alpha`
  `// Evaluate network output`
  $\boldsymbol{\mu}(\mathbf{z}_t, \mathbf{w}, t) \leftarrow \frac{1}{\sqrt{1-\beta_t}}\left\{\mathbf{z}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}}\mathbf{g}(\mathbf{z}_t, \mathbf{w}, t)\right\}$
  $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{\epsilon}|\mathbf{0}, \mathbf{I})$ `// Sample a noise vector`
  $\mathbf{z}_{t-1} \leftarrow \boldsymbol{\mu}(\mathbf{z}_t, \mathbf{w}, t) + \sqrt{\beta_t}\boldsymbol{\epsilon}$ `// Add scaled noise`
end for
$\mathbf{x} = \frac{1}{\sqrt{1-\beta_1}}\left\{\mathbf{z}_1 - \frac{\beta_1}{\sqrt{1-\alpha_1}}\mathbf{g}(\mathbf{z}_1, \mathbf{w}, t)\right\}$ `// Final denoising step`
return $\mathbf{x}$

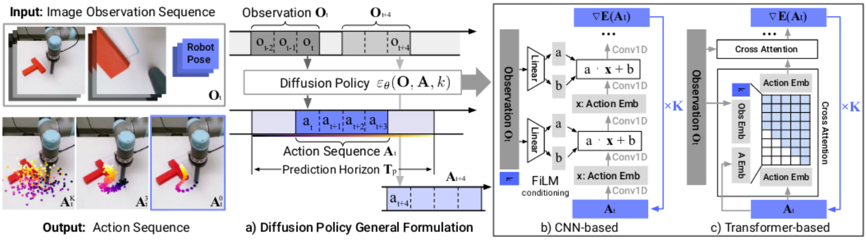
1.7:15

Diffusion + Imitation Learning

Robotics: Science and Systems 2023
Daegu, Republic of Korea, July 10-July 14, 2023

Diffusion Policy:
Visuomotor Policy Learning via Action Diffusion

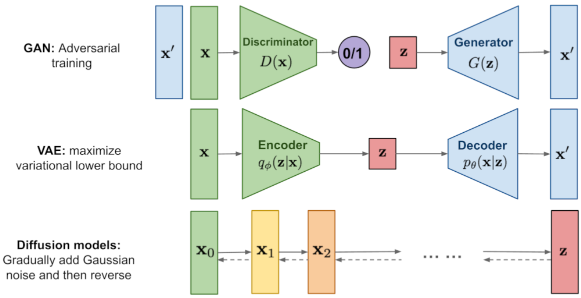
Cheng Chi[1], Siyuan Feng[2], Yilun Du[3], Zhenjia Xu[1], Eric Cousineau[2], Benjamin Burchfiel[2], Shuran Song[1]
[1] Columbia University [2] Toyota Research Institute [3] MIT
https://diffusion-policy.cs.columbia.edu



Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song, (2023). Diffusion Policy: Visuomotor Policy Learning via Action Diffusion. In *Robotics: Science and Systems XIX*

1.7:16

Comparison of Generative Models



- What are advantages / disadvantages? (e.g., sample quality, sample efficiency, distribution "coverage", ease of training)

1.7:17

Case Study: Deep Drone Acrobatics


Robotics: Science and Systems 2020
Corvalis, Oregon, USA, July 12-16, 2020

Deep Drone Acrobatics

Elia Kaufmann*‡, Antonio Loquercio*‡, René Ranftl†, Matthias Müller†, Vladlen Koltun†, Davide Scaramuzza‡

Elia Kaufmann, Antonio Loquercio, Rene Ranftl, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza, (2020). Deep Drone Acrobatics. In *Robotics: Science and Systems XVI*

`https://youtu.be/2N_wKXQ6MXA`

1.7:18

Case Study: Deep Drone Acrobatics

- Input

(i) **Abstraction** of sequence of last camera images (feature tracks)

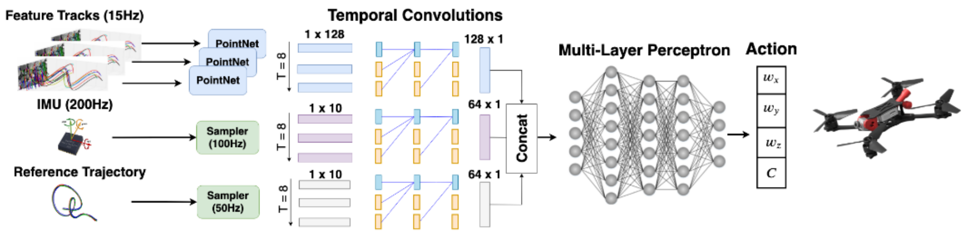
(ii) **Preprocessed** sequence of IMU data

(iii) Reference trajectory

- Output
  - Desired body rates and thrust (to be tracked by attitude controller)
- Data
  - Purely from simulation (privileged expert = optimization-based MPC controller)
- Learning
  - Privileged Teacher (here: given, not learned from human demonstrations)
  - DAgger

1.7:19

Case Study: Deep Drone Acrobatics



1.7:20

Case Study: Deep Drone Acrobatics

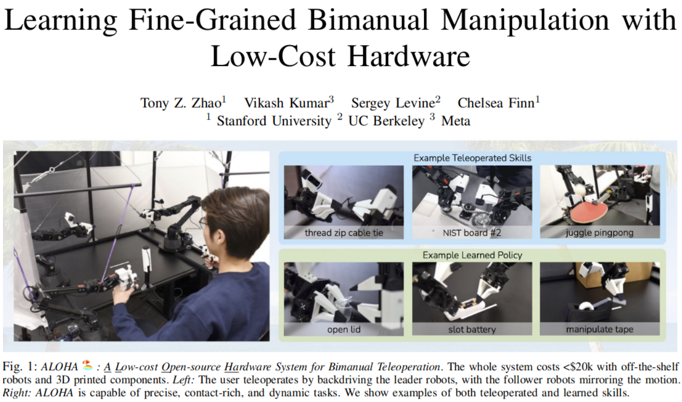
Unique design choices:

- Pre-processing of input for **sim-to-real transfer**


- Asynchronous network branch inference
- Custom DAgger rollout for **sim-to-real transfer**: only use policy if similar to expert; also include random actions

1.7:21

Case Study: Using ALOHA Data



Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware

Tony Z. Zhao[1] Vikash Kumar[3] Sergey Levine[2] Chelsea Finn[1]
[1] Stanford University [2] UC Berkeley [3] Meta

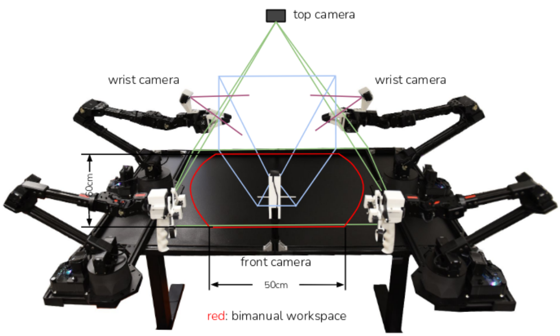
Fig. 1: *ALOHA : A Low-cost Open-source Hardware System for Bimanual Teleoperation*. The whole system costs <$20k with off-the-shelf robots and 3D printed components. *Left:* The user teleoperates by backdriving the leader robots, with the follower robots mirroring the motion. *Right:* ALOHA is capable of precise, contact-rich, and dynamic tasks. We show examples of both teleoperated and learned skills.

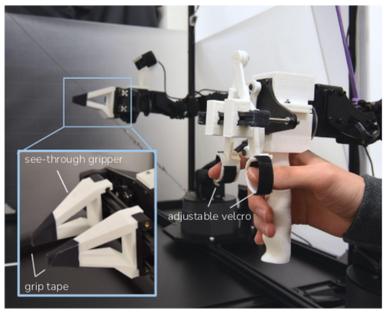
Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn, (2023). Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware

`https://tonyzhaozh.github.io/aloha/`

1.7:22

Case Study: Using ALOHA Data

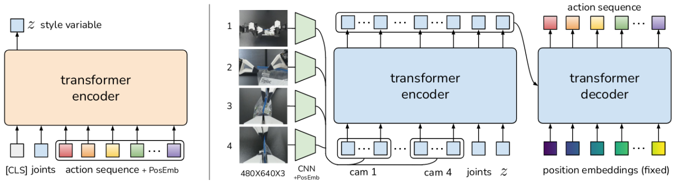




1.7:23

Case Study: Using ALOHA Data

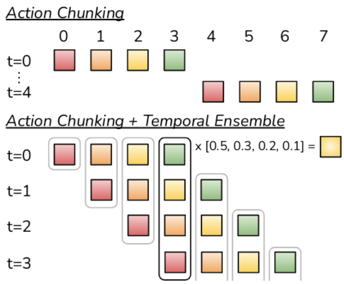
- Conditional Variational Autoencoder (CVAE)
  - Encoder: joint positions, expert action sequence ($k >> 1$)
  - Latent space: z "style" (dim=32)
  - Decoder: observations (4 RGB images), joint positions, "style" z; output: planned action sequence



1.7:24

Case Study: Using ALOHA Data

- Inference: z is always set to 0 (deterministic generator)
- Key insights: transformer architectures for encoder and decoder; MPC-style encoding (action chunks + temporal ensemble)
- Fun statistics:
  - 80 M parameters; 5h training (RTX 2080 Ti); 10ms inference
  - 50 demonstrations per task (about 20min of data)



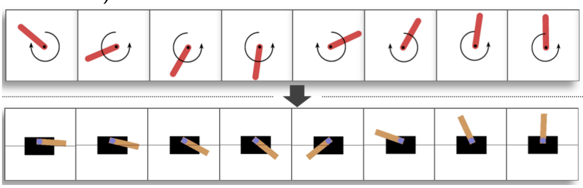
1.7:25

Case Study: Domain Adaptive Imitation Learning (DAIL)

Domain Adaptive Imitation Learning

Kuno Kim [1] **Yihong Gu** [2] **Jiaming Song** [1] **Shengjia Zhao** [1] **Stefano Ermon** [1]

- How to perform a task, given demonstrations from a different domain (viewpoint, embodiment, and/or dynamics mismatch)?



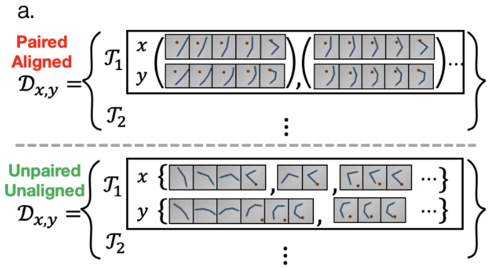
https://youtu.be/l0tc1JCN_1M

Kuno Kim, Yihong Gu, Jiaming Song, Shengjia Zhao, and Stefano Ermon, (2020). Domain Adaptive Imitation Learning. In *Proceedings of the 37th International Conference on Machine Learning*, pages 5286–5295

1.7:26

Case Study: Domain Adaptive Imitation Learning (DAIL)

- Given: **unprocessed** examples for the same tasks for robots x and y
  - $D_{x,y} = \{(D_{M_x,T_i}, D_{M_y,T_i})\}_{i=1}^N$ for N tasks $\{T_i\}_{i=1}^N$
  - Data is not paired/aligned, i.e., $s_x^{(t)}$ does not "match" $s_y^{(t)}$



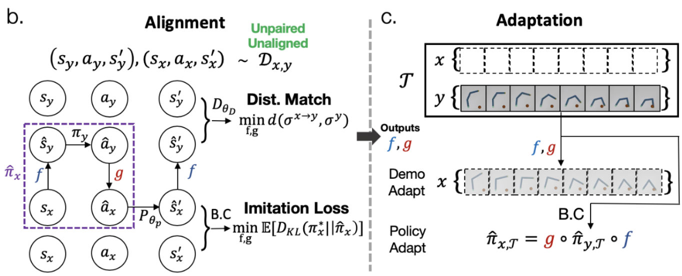
- Goal: Given a new demonstration of unseen task T_j for y, transfer/execute directly ("zero-shot") on robot x

Case Study: Domain Adaptive Imitation Learning (DAIL)

- Learning Alignment from $D_{x,y} = \{(D_{M_x,T_i}, D_{M_y,T_i})\}_{i=1}^N$:
  (i) Learn π^*_{y,T_i} for all T_i (Behavior Cloning)
  (ii) Learn mapping of states from x to y: $f_{\theta_f} : x_x \mapsto x_y$
  (iii) Learn mapping of actions from y to x: $g_{\theta_g} u_y \mapsto u_x$
  (iv) Learn dynamics/step function of x: $P^x_{\theta_P} : x_x, u_x \mapsto x_x$

Case Study: Domain Adaptive Imitation Learning (DAIL)

- Adaption
  (i) Learn π^*_{y,T_j} for new task T_j (Behavior Cloning)
  (ii) $\pi^*_{y,T_i}(x_x) = g_{\theta_g}(\pi^*_{y,T_j}(f_{\theta_f}(x_x)))$



Case Study: Domain Adaptive Imitation Learning (DAIL)

- Alignment Approach: Generative Adversarial MDP Alignment (GAMA)
  - Discriminator tries to separate real transitions $((x, u) \to x')$ from aligned transitions
  - "Generator" are f and g (deterministic)

Algorithm 1 Generative Adversarial MDP Alignment (GAMA)

input: Alignment task set $\mathcal{D}_{x,y} = \{(\mathcal{D}_{\mathcal{M}_{x,\mathcal{T}_i}}, \mathcal{D}_{\mathcal{M}_{y,\mathcal{T}_i}})\}_{i=1}^N$ of unpaired trajectories, fitted $\pi^*_{y,\mathcal{T}_i}$
while not done **do**:
for $i = 1, ..., N$ **do**:
Sample $(s_x, a_x, s'_x) \sim \mathcal{D}_{\mathcal{M}_{x,\mathcal{T}_i}}, (s_y, a_y, s'_y) \sim \mathcal{D}_{\mathcal{M}_{y,\mathcal{T}_i}}$ and store in buffer $\mathcal{B}^i_x, \mathcal{B}^i_y$
for $j = 1, ..., M$ **do**:
Sample mini-batch j from $\mathcal{B}^i_x, \mathcal{B}^i_y$
Update dynamics model with: $-\hat{\mathbb{E}}_{\pi^*_{x,\mathcal{T}_i}}[\nabla_{\theta_P}(P^x_{\theta_P}(s_x, a_x) - s'_x)^2]$
Update discriminator: $\hat{\mathbb{E}}_{\pi^*_{y,\mathcal{T}_i}}[\nabla_{\theta^i_D} \log D_{\theta^i_D}(s_y, a_y, s'_y)] + \hat{\mathbb{E}}_{\pi^*_{x,\mathcal{T}_i}}[\nabla_{\theta^i_D} \log(1 - D_{\theta^i_D}(\hat{s}_y, \hat{a}_y, \hat{s}'_y))]$
Update alignments $(f_{\theta_f}, g_{\theta_g})$ with gradients:

$$-\hat{\mathbb{E}}_{\pi^*_{x,\mathcal{T}_i}}[\nabla_{\theta_f} \log D_{\theta_D}(\hat{s}_y, \hat{a}_y, \hat{s}'_y)] + \hat{\mathbb{E}}_{\pi^*_{x,\mathcal{T}_i}}[\nabla_{\theta_f}(\hat{\pi}_{x,\mathcal{T}_i}(s_x) - a_x)^2]$$
$$-\hat{\mathbb{E}}_{\pi^*_{x,\mathcal{T}_i}}[\nabla_{\theta_g} \log D_{\theta_D}(\hat{s}_y, \hat{a}_y, \hat{s}'_y)] + \hat{\mathbb{E}}_{\pi^*_{x,\mathcal{T}_i}}[\nabla_{\theta_g}(\hat{\pi}_{x,\mathcal{T}_i}(s_x) - a_x)^2]$$

1.7:30

Conclusion

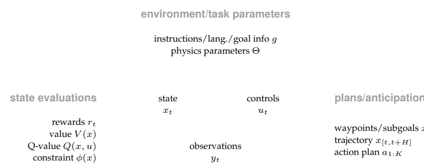
- Imitation Learning works well for robotics
  - Efficient, effective, stable training
  - Fast inference
  - State-of-the-art real-robot results (mobile robots, manipulation, planning)
- Main challenge: acquire labeled data
  - Simulation possible (e.g., make slow algorithms fast) ⇒ Use **DAgger** and/or **privileged teacher** paradigm
  - Only real data ⇒ intuitive data collection interfaces, powerful generative and sequence models, transfer learning
- Details can be tricky (what to learn [policy, trajectory, value function], how to represent inputs, network architectures)
- Not discussed (yet): How to become better than the "expert" (notion of reward)

1.7:31

1.8 Reinforcement Learning

(slides by Marc Toussaint)

I. What is learned?



- So far we discussed dynamics and imitation learning
  – The mappings we learned concerned x, y, u (including also dynamics parameters Θ and constraints $\phi(x)$)
  – Demonstration data was given, or dynamics data well-collected
  – There is no external task/cost evaluation

- In RL, we assume **rewards** r **given**, which opens a new dimension
  - We will learn state values (V-, Q-function) and a policy maximizing expected discounted rewards
  - RL is more autonomous in that it explores the world and generates its own data
  - But it relies on an externally given reward function

1.8:1

Outline

- First essentials towards modern Deep RL methods

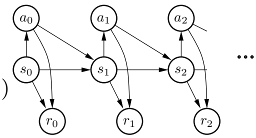
- Then a discussion of challenges

1.8:2

Markov Decision Process

- *The world:* An MDP $(\mathcal{S}, \mathcal{A}, P, R, P_0, \gamma)$ with state space $\mathcal{S}$, action space $\mathcal{A}$, transition probabilities $P(s_{t+1} \mid s_t, a_t)$, reward fct $r_t = R(s_t, a_t)$, initial state distribution $P_0(s_0)$, and discounting factor $\gamma \in [0, 1]$.
- *The agent:* A parameterized policy $\pi_\theta(a_t|s_t)$.

- Together they define the path distribution ($\xi = (s_{0:T+1}, a_{0:T})$)



$$P_\theta(\xi) = P(s_0) \prod_{t=0}^{T} \pi_\theta(a_t|s_t)\ P(s_{t+1}|s_t, a_t)$$

and the **expected discounted return** (with *discounting factor* $\gamma \in [0, 1)$)

$$J(\theta) = \mathbb{E}_{\xi \sim P_\theta} \{ \underbrace{\textstyle\sum_{t=0}^{\infty} \gamma^t r_t}_{R(\xi)} \} = \int_\xi P_\theta(\xi)\ R(\xi)\ d\xi$$

1.8:3

Value functions

[The following assumes a deterministic policy $a = \pi(s)$; stochastic $\pi(a|s)$ is handled with expectations over a.]

- The **value function** of a policy π_θ gives the return when started in state s:

$$V^\pi(s) = \mathbb{E}\{\textstyle\sum_t \gamma^t r_t \mid s_0 = s\}$$
$$V^\pi(s) = R(s, \pi(s)) + \gamma \mathbb{E}_{s'|s,\pi(s)}\{V^\pi(s')\} \qquad \text{(Bellman Equation)}$$

- The **Q-function** gives the return when starting in state s and taking first action a:

$$Q^\pi(s, a) = \mathbb{E}\{\textstyle\sum_t \gamma^t r_t \mid s_0 = s, a_0 = a\}$$
$$Q^\pi(s, a) = R(s, a) + \gamma \mathbb{E}_{s'|s,a}\{Q^\pi(s', \pi(s'))\} \qquad \text{(Bellman Equation)}$$

1.8:4

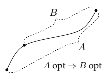
Bellman Optimality Equation

- Bellman equations ($\leftrightarrow$ *Policy Evaluation*):

$$V^\pi(s) = R(s, \pi(s)) + \gamma \mathbb{E}_{s'|s,\pi(s)}\{V^\pi(s')\}$$
$$Q^\pi(s,a) = R(s,a) + \gamma \mathbb{E}_{s'|s,a}\{Q^\pi(s', \pi(s'))\}$$

- Bellman optimality equations: ($\leftrightarrow$ Q-Iteration/Value Iteration)

$$V^*(s) = \max_a \Big[R(s,a) + \gamma \mathbb{E}_{s'|s,a}\{V^*(s')\}\Big] \;= \max_a Q^*(s,a)$$
$$Q^*(s,a) = R(s,a) + \gamma \mathbb{E}_{s'|s,a}\{\max_{a'} Q^*(s',a')\}$$
$$\pi^*(s) = \operatorname{argmax}_a Q^*(s,a)$$




Richard E. Bellman (1920–1984)

[Sketch of proof: If π^* would be other than $\operatorname{argmax}_a[\cdot]$, then $\pi' = \pi$ everywhere except $\pi'(s) = \operatorname{argmax}_a[\cdot]$ would be better.]

1.8:5

- The core question is how to actually compute them

- Model-based: (if we know or estimated the models $P(s'|s,a), R(s,a), P(s_0)$)
  - Q-Iteration, Policy Iteration

- Data-based: (if we directly use data $D = \{(s_i, a_i, r_i, s_{i+1})\}_{i=0}^n$)
  - "Reinforcement Learning"
  - TD-Learning, Q-learning, Actor-Critic
  - Modern: DDPG, TC3, SAC, etc

1.8:6

Model-based: Q-Iteration

- Bellman Optimality equation for Q^*:

$$Q^*(s,a) = R(s,a) + \gamma \mathbb{E}_{s'\,|\,s,a}\{\underbrace{\max_{a'} Q^*(s',a')}_{V^*(s')}\}$$

- **Q-Iteration:** initialize $Q_{k=0}(s,a) = 0$, then iterate:

$$\forall_s:\; V_{k+1}(s) = \max_{a'} Q_k(s,a')$$
$$\forall_{s,a}:\; Q_{k+1}(s,a) = R(s,a) + \gamma \mathbb{E}_{s'|s,a}\{V_{k+1}(s')\}$$

stopping criterion: $\max_{s,a} |Q_{k+1}(s,a) - Q_k(s,a)| \le \epsilon$

[Note: Using V_{k+1} in this iteration is like a buffer – cf. the "target network" in neural RL.]

- **Theorem:** Q-Iteration converges to the optimal state-action value function Q^*

1.8:7

Q-Iteration – Proof of convergence

- Let $\Delta_k = \|Q^* - Q_k\|_\infty = \max_{s,a} |Q^*(s,a) - Q_k(s,a)|$

$$\begin{aligned} Q_{k+1}(s,a) &= R(s,a) + \gamma \mathbb{E}_{s'|s,a}\{\max_{a'} Q_k(s',a')\} \\ &\le R(s,a) + \gamma \mathbb{E}_{s'|s,a}\Big\{\max_{a'} \Big[Q^*(s',a') + \Delta_k\Big]\Big\} \\ &= \Big[R(s,a) + \gamma \mathbb{E}_{s'|s,a}\{\max_{a'} Q^*(s',a')\}\Big] + \gamma\Delta_k \\ &= Q^*(s,a) + \gamma\Delta_k \end{aligned}$$

similarly: $Q_{k+1} \ge Q^* - \gamma\Delta_k$

- The proof translates directly also to value iteration

1.8:8

Model-based: Policy Iteration

- **Policy Evaluation:** Dynamic Programming for Q^π instead of Q^*: Iterate:

$$\forall_s: \ V_{k+1}(s) = Q_k(s, \pi(s))$$

$$\forall_{s,a}: \ Q_{k+1}(s,a) = R(s,a) + \gamma \mathbb{E}_{s'|s,a}\big\{V_{k+1}(s')\big\}$$

stopping criterion: $\max_{s,a} |Q_{k+1}(s,a) - Q_k(s,a)| \le \epsilon$

- **Policy Improvement:** Then update the policy to become better:

$$\pi(s) \leftarrow \operatorname*{argmax}_a Q(s,a)$$

- Iterating the two steps above is guaranteed to converge
- This is also called **actor-critic** (with π=actor, and Q^π=critic)

1.8:9

- The two discussed methods (Q-Iteration and Policy Iteration) can compute optimal policies, but require a known (or estimated) model
- To approximately do the same from data, we follow two strategies
  - Whenever there was an expectation $\mathbb{E}\{\cdot\}$ in these equations, we replace it by sample data
  - Whenever there was a full function update (e.g. $\forall_{s,a} : Q(s,a) \leftarrow \cdots$ or policy improvement) we need to replace it by a **data-based loss functions** and do gradient steps.

- For simplicity, the following focusses on Policy Iteration (or *actor-critic*) approaches

  [Similar strategies can be applied for "Deep Q-Learning":
  Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, and Georg Ostrovski, (2015). Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533
  But major RL methods nowadays follow actor-critic approaches]

1.8:10

Data-based: Bellman Loss for the Q-function

- Recall

$$Q^\pi(s,a) = R(s,a) + \gamma \mathbb{E}_{s'|s,a}\big\{Q^\pi(s', \pi(s'))\big\}$$

- Given data $D = \{(s_i, a_i, r_i, s_{i+1})\}_{i=0}^T$, define the **Bellman residual**:

$$\mathcal{B}^\pi(Q_\theta, \bar Q) = \mathbb{E}_{(s,a,r,s')\sim D}\{\ [Q_\theta(s,a) - r - \gamma \bar Q(s', \pi(s'))]^2\ \}$$

- This defines a supervised ML problem for Q_θ! We have Q-gradients and can do standard SGD.
  - Actually we want $\bar Q \equiv Q_\theta$, and could compute gradients also accounting for $\gamma \bar Q(s', \pi(s'))$. This is called **Bellman residual minimization**, and known since the 80ies, but has challenges [74, 45]
  - So instead, during training we fix $\bar Q$ to some "old version" of Q_θ: We set $\bar Q = Q_{\bar\theta}$ where $\bar\theta$ is a low-pass filter of θ (a **delayed** version of the current parameters θ). This stabilizes training.

1.8:11

- So, for a given policy π, $\mathcal{B}^\pi(Q_\theta, \bar Q)$ defines a loss for Q_θ
- How can we also define a loss function for the policy?

1.8:12

Data-based: Return Maximization for the Policy

- To train the policy, we choose to directly maximize expected return:

$$J(\theta) = \mathbb{E}_{\xi\sim P_\theta}\{\underbrace{\textstyle\sum_{t=0}^\infty \gamma^t R(s_t, a_t)}_{R(\xi)}\} = \int_\xi P_\theta(\xi)\ R(\xi)\ d\xi$$

  - This is not really an error, but exactly what we aim to maximize
  - All we need is the gradient $\frac{\partial}{\partial\theta} J(\theta)$

1.8:13

Policy Gradient $\frac{\partial}{\partial\theta} J(\theta)$

[The word "policy gradient" means gradient of $J(\theta)$ w.r.t. the policy parameters θ.]

- For a deterministic policy $a = \pi_\theta(s) \in \mathbb{R}^d$:

$$\frac{\partial}{\partial\theta} J(\theta) = \mathbb{E}_{s\sim P_\theta}\Big\{\frac{\partial}{\partial a} Q^{\pi_\theta}(s,a)\big|_{a=\pi_\theta(s)} \frac{\partial}{\partial\theta}\pi_\theta(s)\Big\}$$

[Derived here: [103], and led to the **Deep Deterministic Policy Gradient (DDPG)** method [70]. Is the foundation of many followups. This gradient is somewhat noisy, D4PG is an improvement.]

- For a stochastic policy $\pi_\theta(a|s)$: (standard "Policy Gradient Theorem"):

$$\frac{\partial}{\partial\theta} J(\theta) = \frac{\partial}{\partial\theta}\int P_\theta(\xi)\ R(\xi)\ d\xi = \int P_\theta(\xi) \frac{\partial}{\partial\theta}\log P_\theta(\xi) R(\xi) d\xi$$
$$= \mathbb{E}_{\xi\sim P_\theta}\Big\{\frac{\partial}{\partial\theta}\log P_\theta(\xi) R(\xi)\Big\} = \mathbb{E}_{\xi\sim P_\theta}\Big\{\textstyle\sum_{t=0}^H \gamma^t\ [\frac{\partial}{\partial\theta}\log \pi_\theta(a_t|s_t)]\ \underbrace{\textstyle\sum_{t'=t}^H \gamma^{t'-t} r_{t'}}_{Q^{\pi_\theta}(s_t,a_t)}\Big\}$$

1.8:14

RL: Interleaving training with data collection

Algorithm 1 TD3

Initialize critic networks Q_{θ_1}, Q_{θ_2}, and actor network π_ϕ with random parameters θ_1, θ_2, ϕ
Initialize target networks $\theta'_1 \leftarrow \theta_1, \theta'_2 \leftarrow \theta_2, \phi' \leftarrow \phi$
Initialize replay buffer $\mathcal{B}$
for $t = 1$ **to** T **do**
  Select action with exploration noise $a \sim \pi(s) + \epsilon$, $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward r and new state s'
  Store transition tuple (s, a, r, s') in $\mathcal{B}$

  Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{B}$
  $\tilde{a} \leftarrow \pi_{\phi'}(s) + \epsilon, \quad \epsilon \sim \mathrm{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
  $y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \tilde{a})$
  Update critics $\theta_i \leftarrow \min_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2$
  if t mod d **then**
    Update ϕ by the deterministic policy gradient:
    $\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$
    Update target networks:
    $\theta'_i \leftarrow \tau\theta_i + (1 - \tau)\theta'_i$
    $\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$
  end if
end for

- Actor-Critic style Deep RL:
  - $\frac{\partial}{\partial\theta}\mathcal{B}(Q_\theta, \bar{Q})$ provides gradient steps for Q_θ
  - $\frac{\partial}{\partial\theta}J(\theta)$ provides gradient steps for π_θ
  - gradually training both is interleaved with collecting more data

Scott Fujimoto, Herke Hoof, and David Meger, (2018). Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596

1.8:15

Techniques to improve methods

- Papers on techniques in state-of-the-art methods:
  - In Deep Q-Learning (DQN) approaches: [54] (Rainbow paper)
  - In Actor-Critic approaches: [40] (TD3 paper)
  - A state-of-the-art actor-critic method: [49] (SAC paper)
- Many ideas:
  - Replay buffers ("experience replay"): Limited buffer of experiences to train on (approximates $P_\theta(s, a, r, s')$)
  - Double Q-Learning: maintain 2 indep. Q-functions $Q_{1,2}(s, a)$ (and use min in policy update)
  - Delayed targets: low pass filter $\bar{Q}$ of Q as target
  - Smoothed policy samples: add (clipped) noise when sampling policy in Bellman loss
  - Prioritized Replay: (pick replay data where Bellman error is largest)
  - Dueling Networks: (decompose Q in value and advantage)
  - Multi-Step Learning: (n-step updates)
  - Distributional RL: (let Q-function predict return *distribution*, not mean)
  - Noisy Nets: (replace ϵ-greedy exploration by "learnt noise")

1.8:16

Discussion

- The previous material should enable you to read about modern Deep RL methods (TD3, D4PG, SAC)

- Rest of this lecture is discussion
  - Why do we actually learn Q and not V?
  - What if we have partial observability?
  - How is the data collected?
  - How are reward functions engineered?
  - Why not just use black-box optimization?

1.8:17

Why do we actually learn Q and not V?

- $Q(s,a)$ tells us what is the best action $a = \text{argmax}_a Q$
- In control, value functions are also estimated, but never Q (I think). Why?

  [E.g. the Hamilton-Jacobi-Bellman Eq: $-\frac{\partial}{\partial t}V(x,t) = \min_u \Big[c(x,u) + \frac{\partial V}{\partial x} f(x,u)\Big]$.]

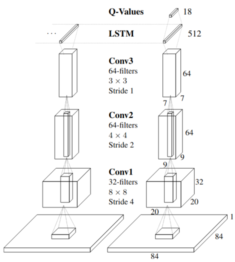
- Without Q-function, we'd somehow have to learn how to walk up-hill on V:
  – Learn an *inverse model* $(s, \Delta s) \mapsto a$
  – Learn a "flow" policy $\pi : s \mapsto \Delta s \approx \frac{\partial}{\partial s} V(s)$

1.8:18

What if we have partial observability?

- Policy has only access to observations $y_{0:t}$

$\to$ Make the Q function a recursive NN



Matthew Hausknecht and Peter Stone, (2015). Deep recurrent q-learning for partially observable mdps. In *2015 Aaai Fall Symposium Series*

1.8:19

How is the data collected?

- A core challenge in modern RL!
- Many modern methods *require* that the data is collected from the current π_θ!
  – So that $\mathbb{E}\{\cdot\}$ can be replaced by the data in the Bellman equations
  – This is called **on-policy** – we'll discuss off-policy next time
  – But π is so uninformed! So non-exploring! So iid. in each step ($\sim$ Brownian noise)
  – Check pseudo codes of mentioned methods (SAC, DDPG, TD3, etc)

- In old RL (discrete state-action spaces), things were much better!
  – **Explicit Exploit or Explore** [61] – a *must read!*
  – R-MAX [9], Optimistic value initialization, Bayesian RL
  – These methods design policies to systematically explore, typically by **systematically rewarding exploration**
  – Optimism in the face of uncertainty: Rewarding decisions with uncertain outcomes!

1.8:20

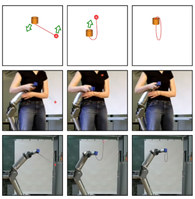
How is the data collected?

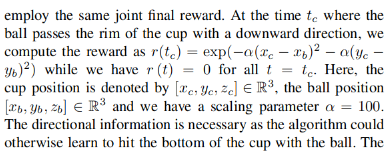
- In Deep RL: Structured noise instead of Brownian:
  Onno Eberhard, Jakob Hollenstein, Cristina Pinneri, and Georg Martius, (2022). Pink noise is all you need: Colored noise exploration in deep reinforcement learning. In *The Eleventh International Conference on Learning Representations*
- Parameter-space noise: (add noise to θ instead of a)
  Matthias Plappert, Rein Houthooft, Prafulla Dhariwal, Szymon Sidor, Richard Y. Chen, Xi Chen, Tamim Asfour, Pieter Abbeel, and Marcin Andrychowicz, (2018). Parameter Space Noise for Exploration
- Guided Policy Search
  Sergey Levine and Vladlen Koltun, (2013). Guided policy search. In *International Conference on Machine Learning*, pages 1–9
  - Use model-based trajectory optimization to generate data
- Demonstration Guided [83]
- Or just give up:
  - Offline Reinforcement Learning: Assume the data was generated somehow externally
  - Imitation Learning & Inverse RL: Learn from demonstrations

1.8:21

How are reward functions engineered?

- Reward shaping theory: You can add potentials without changing optimal policy
  Andrew Y. Ng, Daishi Harada, and Stuart Russell, (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pages 278–287
- Reward engineering:




employ the same joint final reward. At the time t_c where the ball passes the rim of the cup with a downward direction, we compute the reward as $r(t_c) = \exp(-\alpha(x_c - x_b)^2 - \alpha(y_c - y_b)^2)$ while we have $r(t) = 0$ for all $t = t_c$. Here, the cup position is denoted by $[x_c, y_c, z_c] \in \mathbb{R}^3$, the ball position $[x_b, y_b, z_b] \in \mathbb{R}^3$ and we have a scaling parameter $\alpha = 100$. The directional information is necessary as the algorithm could otherwise learn to hit the bottom of the cup with the ball. The

Jens Kober and Jan Peters, (2009). Learning motor primitives for robotics. In *2009 IEEE International Conference on Robotics and Automation*, pages 2112–2118

`https://www.youtube.com/watch?v=qtqubguikMk`

1.8:22

Why not just use black-box optimization?

- Eventually, $\max_\theta J(\theta)$ is an optimization problem
  - Instead of deriving gradients (via Bellman, and Q-functions), why not treat as black-box or **derivative-free optimization** problem?

1.8:23

Evolution Strategies as a Scalable Alternative to Reinforcement Learning

Tim Salimans **Jonathan Ho** **Xi Chen** **Szymon Sidor** **Ilya Sutskever**
OpenAI

Abstract

We explore the use of Evolution Strategies (ES), a class of black box optimization algorithms, as an alternative to popular MDP-based RL techniques such as Q-learning and Policy Gradients. Experiments on MuJoCo and Atari show that ES is a viable solution strategy that scales extremely well with the number of CPUs available: By using a novel communication strategy based on common random numbers, our ES implementation only needs to communicate scalars, making it possible to scale to over a thousand parallel workers. This allows us to solve 3D humanoid walking in 10 minutes and obtain competitive results on most Atari games after one hour of training. In addition, we highlight several advantages of ES as a black box optimization technique: it is invariant to action frequency and delayed rewards, tolerant of extremely long horizons, and does not need temporal discounting or value function approximation.

Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever, (2017). Evolution Strategies as a Scalable Alternative to Reinforcement Learning

1.8:24

- Ratio of ES timesteps to TRPO timesteps needed to reach various percentages of TRPO's learning progress at 5 million timesteps:

Environment	25%	50%	75%	100%
HalfCheetah	0.15	0.49	0.42	0.58
Hopper	0.53	3.64	6.05	6.94
InvertedDoublePendulum	0.46	0.48	0.49	1.23
InvertedPendulum	0.28	0.52	0.78	0.88
Swimmer	0.56	0.47	0.53	0.30
Walker2d	0.41	5.69	8.02	7.88

1.8:25

Deep Neuroevolution: Genetic Algorithms are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning

Felipe Petroski Such **Vashisht Madhavan** **Edoardo Conti** **Joel Lehman** **Kenneth O. Stanley** **Jeff Clune**
Uber AI Labs
{felipe.such, jeffclune}@uber.com

Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O. Stanley, and Jeff Clune, (2018). Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning

- Roughly: "Do you spend your time training nets, or simulating?"

1.8:26

	DQN	ES	A3C	RS 1B	GA 1B	GA 6B
DQN		6	6	3	6	7
ES	7		7	3	6	8
A3C	7	6		6	6	7
RS 1B	10	10	7		13	13
GA 1B	7	7	7	0		13
GA 6B	6	5	6	0	0	

Table 4. **Head-to-head comparison between algorithms on the 13 Atari games.** Each value represents how many games for which the algorithm listed at the top of a column produces a higher score than the algorithm listed to the left of that row (e.g. GA 6B beats DQN on 7 games).

- Conclusion: It varies from problem to problem what is better.
  And it is suprising that "naive" black-box ES can beat elaborate RL-methods

1.8:27

1.8:28

1.9 RL II: Offline RL & Sim2Real

(slides by Marc Toussaint)

Outline

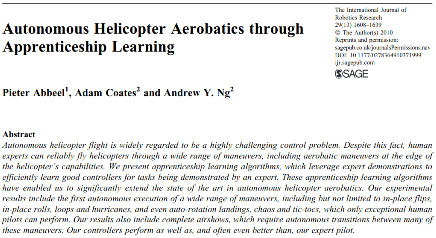
- Some RL application papers
- Offline RL (on-policy vs. off-policy)
- Sim2Real
  - Domain Randomization
  - Privileged Training & Imitation Learning
  - Domain Adaptation

1.9:1

Outline

- **Some RL application papers**
- Offline RL (on-policy vs. off-policy)
- Sim2Real
  - Domain Randomization
  - Privileged Training & Imitation Learning
  - Domain Adaptation

1.9:2



Autonomous Helicopter Aerobatics through Apprenticeship Learning

The International Journal of Robotics Research
29(13) 1608–1639
© The Author(s) 2010
Reprints and permission: sagepub.co.uk/journalsPermissions.nav
DOI: 10.1177/0278364910371999
ijr.sagepub.com
SAGE

Pieter Abbeel[1], Adam Coates[2] and Andrew Y. Ng[2]

Abstract
Autonomous helicopter flight is widely regarded to be a highly challenging control problem. Despite this fact, human experts can reliably fly helicopters through a wide range of maneuvers, including aerobatic maneuvers at the edge of the helicopter's capabilities. We present apprenticeship learning algorithms, which leverage expert demonstrations to efficiently learn good controllers for tasks being demonstrated by an expert. These apprenticeship learning algorithms have enabled us to significantly extend the state of the art in autonomous helicopter aerobatics. Our experimental results include the first autonomous execution of a wide range of maneuvers, including but not limited to in-place flips, in-place rolls, loops and hurricanes, and even auto-rotation landings, chaos and tic-tocs, which only exceptional human pilots can perform. Our results also include complete airshows, which require autonomous transitions between many of these maneuvers. Our controllers perform as well as, and often even better than, our expert pilot.

Pieter Abbeel, Adam Coates, and Andrew Y. Ng, (2010). Autonomous Helicopter Aerobatics through Apprenticeship Learning. *The International Journal of Robotics Research*, 29(13):1608–1639

http://heli.stanford.edu/

1.9:3

Article

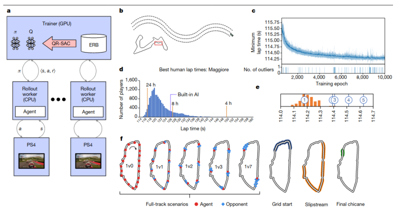
Outracing champion Gran Turismo drivers with deep reinforcement learning

https://doi.org/10.1038/s41586-021-04357-7
Received: 9 August 2021
Accepted: 15 December 2021
Published online: 9 February 2022
Check for updates

Peter R. Wurman[1✉], Samuel Barrett[1], Kenta Kawamoto[2], James MacGlashan[1], Kaushik Subramanian[3], Thomas J. Walsh[1], Roberto Capobianco[3], Alisa Devlic[3], Franziska Eckert[3], Florian Fuchs[3], Leilani Gilpin[1], Piyush Khandelwal[1], Varun Kompella[1], HaoChih Lin[3], Patrick MacAlpine[1], Declan Oller[1], Takuma Seno[2], Craig Sherstan[1], Michael D. Thomure[1], Houmehr Aghabozorgi[1], Leon Barrett[1], Rory Douglas[1], Dion Whitehead[1], Peter Dürr[3], Peter Stone[1], Michael Spranger[2] & Hiroaki Kitano[2]

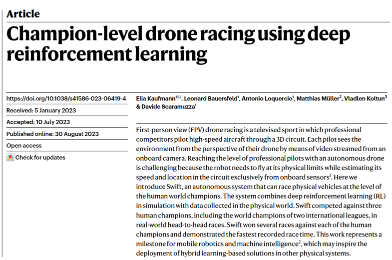
Many potential applications of artificial intelligence involve making real-time decisions in physical systems while interacting with humans. Automobile racing represents an extreme example of these conditions; drivers must execute complex tactical manoeuvres to pass or block opponents while operating their vehicles at their traction limits[1]. Racing simulations, such as the PlayStation game Gran Turismo, faithfully reproduce the non-linear control challenges of real race cars while also encapsulating the complex multi-agent interactions. Here we describe how we trained agents for Gran Turismo that can compete with the world's best e-sports drivers. We combine state-of-the-art, model-free, deep reinforcement learning algorithms with mixed-scenario training to learn an integrated control policy that combines exceptional speed with impressive tactics. In addition, we construct a reward function that enables the agent to be competitive while adhering to racing's important, but under-specified, sportsmanship rules. We demonstrate the capabilities of our agent, Gran Turismo Sophy, by winning a head-to-head competition against four of the world's best Gran Turismo drivers. By describing how we trained championship-level racers, we demonstrate the possibilities and challenges of using these techniques to control complex dynamical systems in domains where agents must respect imprecisely defined human norms.

Peter R. Wurman, Samuel Barrett, Kenta Kawamoto, James MacGlashan, Kaushik Subramanian, Thomas J. Walsh, Roberto Capobianco, Alisa Devlic, Franziska Eckert, Florian Fuchs, Leilani Gilpin, Piyush Khandelwal, Varun Kompella, HaoChih Lin, Patrick MacAlpine, Declan Oller, Takuma Seno, Craig Sherstan, Michael D. Thomure, Houmehr Aghabozorgi, Leon Barrett, Rory Douglas, Dion Whitehead, Peter Dürr, Peter Stone, Michael Spranger, and Hiroaki Kitano, (2022). Outracing champion Gran Turismo drivers with deep reinforcement learning. *Nature*, 602(7896):223–228



https://sonyresearch.github.io/gt_sophy_public/

1.9:4



Article

Champion-level drone racing using deep reinforcement learning

https://doi.org/10.1038/s41586-023-06419-4
Received: 5 January 2023
Accepted: 10 July 2023
Published online: 30 August 2023
Open access
Check for updates

Elia Kaufmann[1✉], Leonard Bauersfeld[1], Antonio Loquercio[1], Matthias Müller[2], Vladlen Koltun[3] & Davide Scaramuzza[1]

First-person view (FPV) drone racing is a televised sport in which professional competitors pilot high-speed aircraft through a 3D circuit. Each pilot sees the environment from the perspective of their drone by means of video streamed from an onboard camera. Reaching the level of professional pilots with an autonomous drone is challenging because the robot needs to fly at its physical limits while estimating its speed and location in the circuit exclusively from onboard sensors[1]. Here we introduce Swift, an autonomous system that can race physical vehicles at the level of the human world champions. The system combines deep reinforcement learning (RL) in simulation with data collected in the physical world. Swift competed against three human champions, including the world champions of two international leagues, in real-world head-to-head races. Swift won several races against each of the human champions and demonstrated the fastest recorded race time. This work represents a milestone for mobile robotics and machine intelligence[2], which may inspire the deployment of hybrid learning-based solutions in other physical systems.

Elia Kaufmann, Leonard Bauersfeld, Antonio Loquercio, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza, (2023). Champion-level drone racing using deep reinforcement learning. *Nature*, 620(7976):982–987

https://www.youtube.com/watch?v=fBiataDpGIo

1.9:5

Outline

- Some RL application papers
- **Offline RL (on-policy vs. off-policy)**
- Sim2Real
  - Domain Randomization
  - Privileged Training & Imitation Learning
  - Domain Adaptation

1.9:6

On-Policy vs. Off-Policy Methods

- **On-policy:** estimate V^π or Q^π while executing π (e.g., Policy Evaluation)
  - The value-function updates directly depend on the policy π
- **Off-policy:** estimate Q^* while executing π (e.g., Q-learning)
  - The actually executed (data-collecting) policy π is also called "behavioral policy"
  - In contrast, values Q^* are estimated for the optimal policy π^*
- Off-policy is considered more efficient, as it can use off-policy-distribution data

  [More technically: Consider you have data $D = \{(s_i, a_i, r_i, s_{i+1}, a_{i+1})\}_{i=0}^n$ collected with behavior policy π. When you make Q- or V-updates, do you take only expectations w.r.t. D? Or do you take conditional expectations $a_{i+1} \sim \pi^*(a|s_{i+1})$ w.r.t. another policy? (E.g. greedy policy.)]

  [SAC is called off-policy, because when training V it takes expectations w.r.t. $a_t \sim \pi_\theta$ (instead of w.r.t. data collected previously).]

1.9:7

Offline RL

- Motivation:
  - Separation of Concerns!
  - Separate thinking about Data Collection, and thinking about **what best to make of given data**
  - Real-world data is expensive!
  - Data collection (exploration) in RL is an issue anyway
  - No matter how RL collects data, it makes sense to study what best to make of given data

  - **The data could come from anywhere:** huge data sets of other observed agents, of human behavior, perhaps extracted from abundant video
  - The data is not collected by "our AI agent" itself – but can still be used to learn a Q^*-function and train our agent for optimal behavior

1.9:8

Offline RL

- Naive problem formulation: Given data $D = \{(s_i, a_i, r_i, s_{i+1})\}_{i=0}^n$, find θ to

$$\begin{aligned}\min_\theta \quad & \mathbb{E}_{(s,a,r,s')\sim D}\{\ [Q_\theta(s,a) - r - \gamma Q_{\bar\theta}(s', \pi(s'))]^2\ \}\\ \text{s.t.} \quad & \bar\theta \approx \theta \\ & \pi \approx \operatorname*{argmax}_\pi \mathbb{E}_{(s,a)\sim D}\{Q_\theta(s,a)\}\end{aligned}$$

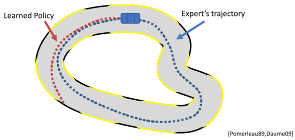
In words:
- minimize the empirical Bellman residual, with delayed $Q_{\bar\theta}$-target
- ...where eventually π becomes optimal and $\bar\theta$ converges

- That's a well-defined problem
  - We have gradients for everything: Bellman gradient, deterministic policy gradient – let's go!

1.9:9

Offline RL

- Resulting policy fails badly, due to distribution shift, just as in imitation learning:



Also called **Compound Error** (Shi's lecture 5)

- In the naive problem formulation
  - there is no penalty for "dreaming" crazy Q-values outside the data distribution
  - the trained policy is likely to exploit these arbitrary Q-values
- We don't have the DAgger option: Can't collect more data to cover reached states!

$\to$ We need to **add a penalty for leaving the data distribution**!

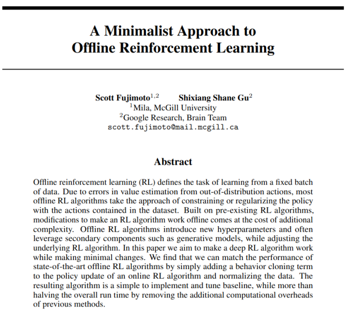
1.9:10

Offline RL

- We need to add a penalty for leaving the data distribution...
  - Many different ideas, incl. literally penalizing "distribution distance" (divergence regularization)
  - Modern versions found simple approaches:

1.9:11

TD3+BC



A Minimalist Approach to Offline Reinforcement Learning

Scott Fujimoto[1,2] Shixiang Shane Gu[2]
[1]Mila, McGill University
[2]Google Research, Brain Team
scott.fujimoto@mail.mcgill.ca

Abstract

Offline reinforcement learning (RL) defines the task of learning from a fixed batch of data. Due to errors in value estimation from out-of-distribution actions, most offline RL algorithms take the approach of constraining or regularizing the policy with the actions contained in the dataset. Built on pre-existing RL algorithms, modifications to make an RL algorithm work offline comes at the cost of additional complexity. Offline RL algorithms introduce new hyperparameters and often leverage secondary components such as generative models, while adjusting the underlying RL algorithm. In this paper we aim to make a deep RL algorithm work while making minimal changes. We find that we can match the performance of state-of-the-art offline RL algorithms by simply adding a behavior cloning term to the policy update of an online RL algorithm and normalizing the data. The resulting algorithm is a simple to implement and tune baseline, while more than halving the overall run time by removing the additional computational overheads of previous methods.

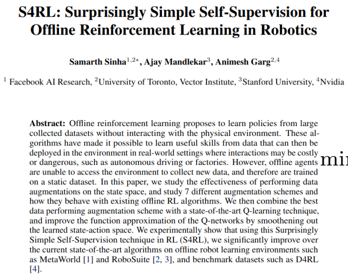
Scott Fujimoto and Shixiang Shane Gu, (2021). A minimalist approach to offline reinforcement learning. *Advances in neural information processing systems*, 34:20132–20145

- Use TD3 (twin delayed deep deterministic..)
- Simply add a BC term to the policy objective!

$$\pi \approx \operatorname*{argmax}_\pi \mathbb{E}_{(s,a)\sim D}\{\lambda Q_\theta(s,a) + (\pi(s)-a)^2\}$$

1.9:12

S4RL


S4RL: Surprisingly Simple Self-Supervision for Offline Reinforcement Learning in Robotics

Samarth Sinha[1,2*], Ajay Mandlekar[3], Animesh Garg[2,4]

[1] Facebook AI Research, [2]University of Toronto, Vector Institute, [3]Stanford University, [4]Nvidia

Abstract: Offline reinforcement learning proposes to learn policies from large collected datasets without interacting with the physical environment. These algorithms have made it possible to learn useful skills from data that can then be deployed in the environment in real-world settings where interactions may be costly or dangerous, such as autonomous driving or factories. However, offline agents are unable to access the environment to collect new data, and therefore are trained on a static dataset. In this paper, we study the effectiveness of performing data augmentations on the state space, and study 7 different augmentation schemes and how they behave with existing offline RL algorithms. We then combine the best data performing augmentation scheme with a state-of-the-art Q-learning technique, and improve the function approximation of the Q-networks by smoothening out the learned state-action space. We experimentally show that using this Surprisingly Simple Self-Supervision technique in RL (S4RL), we significantly improve over the current state-of-the-art algorithms on offline robot learning environments such as MetaWorld [1] and RoboSuite [2, 3], and benchmark datasets such as D4RL [4].

Samarth Sinha, Ajay Mandlekar, and Animesh Garg, (2022). S4rl: Surprisingly simple self-supervision for offline reinforcement learning in robotics. In *Conference on Robot Learning*, pages 907–917

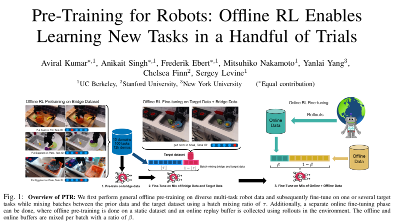
- Include a strong data augmentation in the Q-function loss

$$\min_\theta \mathbb{E}_{(s,a,r,s')\sim D}\big\{ [\tfrac{1}{I}\textstyle\sum_i Q_\theta(\mathcal{T}_i(\tilde s|s),a) - r - \gamma \tfrac{1}{I}\sum_i Q_{\bar\theta}(\mathcal{T}_i(\tilde s'|s'),\pi($$

where $\mathcal{T}_i$ generates a variant of s (they propose 7 alternative, including spatial smoothing and adversarial)

1.9:13

Offline RL Application

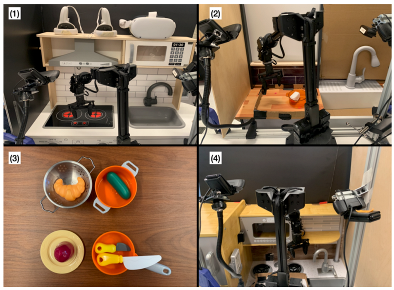

Pre-Training for Robots: Offline RL Enables Learning New Tasks in a Handful of Trials

Aviral Kumar[*,1], Anikait Singh[*,1], Frederik Ebert[*,1], Mitsuhiko Nakamoto[1], Yanlai Yang[3], Chelsea Finn[2], Sergey Levine[1]

[1]UC Berkeley, [2]Stanford University, [3]New York University (*Equal contribution)

Fig. 1: **Overview of PTR:** We first perform general offline pre-training on diverse multi-task robot data and subsequently fine-tune on one or several target tasks while mixing batches between the prior data and the target dataset using a batch mixing ratio of τ. Additionally, a separate online fine-tuning phase can be done, where offline pre-training is done on a static dataset and an online replay buffer is collected using rollouts in the environment. The offline and online buffers are mixed per batch with a ratio of β.

Aviral Kumar, Anikait Singh, Frederik Ebert, Mitsuhiko Nakamoto, Yanlai Yang, Chelsea Finn, and Sergey Levine, (2023). Pre-Training for Robots: Offline RL Enables Learning New Tasks from a Handful of Trials



https://sites.google.com/view/ptr-final/

1.9:14

Offline RL Conclusions

- Scientifically important (separation of concerns)
- Opens new dimension: Train optimal behaviors from *any* data
- Promising future applications (leverage massive data, reward re-labelled data)

1.9:15

Outline

- Some RL application papers
- Offline RL (on-policy vs. off-policy)
- **Sim2Real** (slides based on Shi's lecture)
  - Domain Randomization
  - Privileged Training & Imitation Learning

– Domain Adaptation

1.9:16

- Why train in Simulation?
  – Real-world data is expensive!
  – Many RL methods require millions of samples
  – Simulation is fast
  – Simulation is safe, can be fully explored
  – Simulation provides ground truth labels (e.g. train priviledged policy)
  – Simulations get better and better, including simulating sensors (image rendering)

1.9:17

Robot Simulators

❑ Simulator taxonomy by simulately.wiki

Simulator	Physics Engine	Rendering	Sensor🤨	Dynamics	GPU-accelerated Simulation	Open-Source
IsaacSim	PhysX 5	Rasterization; RayTracing	RGBD; Lidar; Force; Effort; IMU; Contact; Proximity	Rigid;Soft;Cloth;Fluid	✓	✘
IsaacGym	PhysX 5, Flex	Rasterization;	RGBD; Force; Contact;	Rigid;Soft;Cloth	✓	✘
SAPIEN	PhysX 5, Warp	Rasterization; RayTracing⭐;	RGBD; Force; Contact;	Rigid;Soft;Fluid	✘	✓
Pybullet	Bullet	Rasterization;	RGBD; Force; IMU; Tactile;	Rigid;Soft;Cloth	✘	✓
MuJoCo	MuJoCo	Rasterization;	RGBD; Force; IMU; Tactile;	Rigid;Soft;Cloth	✓💡	✓
CoppeliaSim	MuJoCo; Bullet; ODE; Newton; Vortex	Rasterization; RayTracing🔶;	RGBD; Force; Contact;	Rigid;Soft;Cloth	✘	✓
Gazebo	Bullet; ODE; DART; Simbody	Rasterization;	RGBD; Lidar; Force; IMU;	Rigid;Soft;Cloth	✘	✓

from Shi's lecture

1.9:18

- What are Sim2Real issues?
  – Simulation never matches real world exactly; policies overfit to simulation and fail in real
  – **Parameteric mismatches:** Other dynamics parameters, e.g. friction, inertias
  – **Non-parameteric mismatches:** Physical effects not simulated: Wind, exact fluids, sand/dust
- Approaches to tackle this:
  – Domain Randomization
  – Privileged Training & Imitation Learning
  – Domain Adaptation

1.9:19

Domain Randomization

Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World

Josh Tobin[1], Rachel Fong[2], Alex Ray[2], Jonas Schneider[2], Wojciech Zaremba[2], Pieter Abbeel[3]

***Abstract*— Bridging the 'reality gap' that separates simulated robotics from experiments on hardware could accelerate robotic research through improved data availability. This paper explores *domain randomization*, a simple technique for training models on simulated images that transfer to real images by randomizing rendering in the simulator. With enough variability in the simulator, the real world may appear to the model as just another variation. We focus on the task of object localization, which is a stepping stone to general robotic manipulation skills. We find that it is possible to train a real-world object detector that is accurate to 1.5 cm and robust to distractors and partial occlusions using only data from a simulator with non-realistic random textures. To demonstrate the capabilities of our detectors, we show they can be used to perform grasping in a cluttered environment. To our knowledge, this is the first successful transfer of a deep neural network trained *only* on simulated RGB images (without pre-training on real images) to the real world for the purpose of robotic control.**

I. INTRODUCTION

Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel, (2017). Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30

- Train a single policy to perform well in many domain variants
- Original paper focussed on perception, but works equally for any other parameter Θ

1.9:20

Domain Randomization

- Let Θ be a simulation parameter: $x_{t+1} = f(x_t, u_t; \Theta)$
- Randomly sample $\Theta \sim p(\Theta)$ at the start of each episode
- Otherwise, use standard RL
  - But since the world is "more uncertain", the RL problem becomes harder

1.9:21

- What if we train a policy $\hat\pi(s_t, \Theta)$ that get's Θ as input?

  Is that cheating? [16]

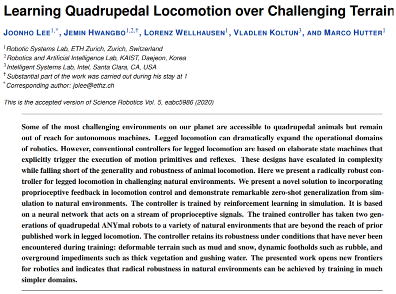
1.9:22

Privileged Training & Imitation Learning

- Priviledged RL Training:
  - We first train $\hat\pi(s_t, \Theta)$ using standard RL
  - Much easier than without access to Θ
- Sensorimotor Imitation using DAgger:
  - Then we train a policy $\pi(s_t)$ to imitate $\hat\pi(s_t, \Theta)$
  - As we can query $\hat\pi(s_t, \Theta)$, we can use DAgger! Much more efficient than plain BC
- This approach is a core paradigm beyond RL:
  - First develop a method to solve a problem using full information (could be a planner)
  - Then train a policy to imitate that method with only available (sensor) information

1.9:23

Privileged Training & Imitation Learning



Learning Quadrupedal Locomotion over Challenging Terrain

Joonho Lee[1,*], Jemin Hwangbo[1,2,†], Lorenz Wellhausen[1], Vladlen Koltun[3], and Marco Hutter[1]

[1] Robotic Systems Lab, ETH Zurich, Zurich, Switzerland
[2] Robotics and Artificial Intelligence Lab, KAIST, Daejeon, Korea
[3] Intelligent Systems Lab, Intel, Santa Clara, CA, USA
[†] Substantial part of the work was carried out during his stay at 1
[*] Corresponding author: jolee@ethz.ch

This is the accepted version of Science Robotics Vol. 5, eabc5986 (2020)

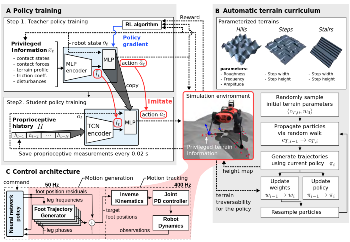
Some of the most challenging environments on our planet are accessible to quadrupedal animals but remain out of reach for autonomous machines. Legged locomotion can dramatically expand the operational domains of robotics. However, conventional controllers for legged locomotion are based on elaborate state machines that explicitly trigger the execution of motion primitives and reflexes. These designs have escalated in complexity while falling short of the generality and robustness of animal locomotion. Here we present a radically robust controller for legged locomotion in challenging natural environments. We present a novel solution to incorporating proprioceptive feedback in locomotion control and demonstrate remarkable zero-shot generalization from simulation to natural environments. The controller is trained by reinforcement learning in simulation. It is based on a neural network that acts on a stream of proprioceptive signals. The trained controller has taken two generations of quadrupedal ANYmal robots to a variety of natural environments that are beyond the reach of prior published work in legged locomotion. The controller retains its robustness under conditions that have never been encountered during training: deformable terrain such as mud and snow, dynamic footholds such as rubble, and overground impediments such as thick vegetation and gushing water. The presented work opens new frontiers for robotics and indicates that radical robustness in natural environments can be achieved by training in much simpler domains.

Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter, (2020). Learning quadrupedal locomotion over challenging terrain. *Science Robotics*, 5(47):eabc5986

https://youtu.be/txjqn8h6pjU
https://youtu.be/Xnn4sVSpSh0

1.9:24

Privileged Training & Imitation Learning



- The privileged policy gets full information as input: Exact Θ and state s_t, including terrain model
- The sensorimotor policy only sensor obs. y_t
  $\to$ the sensorimotor policy needs to use the sequence $y_{0:t}$, e.g. recursive or transformer

1.9:25

- The sensorimotor policy uses full observation sequence $y_{0:t}$ to output controls u_t...
  - What else could it predict based on $y_{0:t}$?

The unobserved physics parameters Θ!

1.9:26

Adaptive Control

- Large area within Control Theory
- Assumes environment has *varying* parameters Θ (not directly observed)

- One approach: Estimate Θ from past observations and use for control
- Robust control: Estimate posterior belief $p(\Theta|y_{0:T})$ over possible Θ and use control robust to all possibilities

1.9:27

Domain Adaptation

- In the Robot Learning community, the word *Domain Adaptation* is used for any controller that adapts (to varying unobserved Θ) based on past observations $y_{0:t}$.

- Explicit approach:
  - Train an estimator $\psi : y_{0:t} \mapsto \hat{\Theta}$
  - Then train a policy $\pi(y_{0:t}, \psi(y_{0:t}))$ for fixed ψ

- Implicit approach:
  - As in Lee et al'20
  - Just train $\pi(y_{0:t})$, but potentially imposing a representation that is also predictive for Θ

1.9:28

Sim2Real Conclusions

- (Pre-)Training in Sim became a standard in modern Robot Learning

- Sim2Real is not considered a blocker anymore:
  - Domain Randomization, Privileged Training & Sensorimotor are powerful approaches
  - Even if policies do not directly transfer $\rightarrow$ Real-World finetuning requires much less data

1.9:29

Side note: Privileged Training for Imitation Learning

- The paper below used same approach, but in the context of Imitation Learning:
  - The privileged policy imitated a human demonstrator using full access to the driving simulation
  - The sensorimotor policy imitated the privileged policy

Dian Chen, Brady Zhou, Vladlen Koltun, and Philipp Krähenbühl, (2020). Learning by cheating. In *Conference on Robot Learning*, pages 66–75

1.9:30

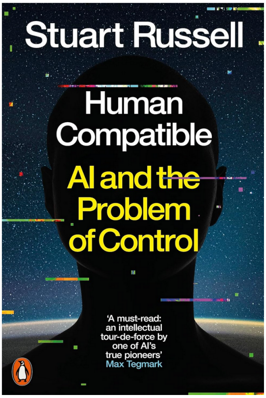
1.10 Inverse RL

(slides by Marc Toussaint)

Outline

- Value Alignment
- Inverse RL
- Preference-based RL

1.10:1



- Stuart Russell
  - Russell & Norvig: *Artificial Intelligence: A Modern Approach* (1995)
  - Decision & Game Theory

Stuart Russell, (2019). *Human compatible: AI and the problem of control*

1.10:2

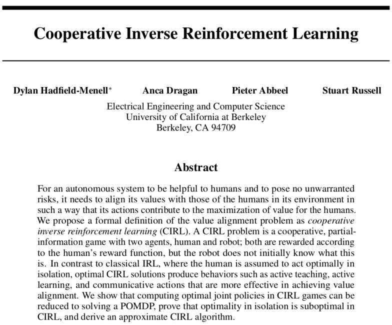
Russell: Value Alignment

- "Standard model of AI"
  - Define fixed objective; maximize

- Difficulty in defining objectives
  - Consequences (aspects of optimal behavior) unclear
  - Humans are bad at defining objectives

- Russell's proposal:
  - Systems should infer human preferences from behavior
  - Avoid overfitting
  - Large apriori uncertainty (incl. noise assumption in human behavior) to avoid overfitting

1.10:3



Cooperative Inverse Reinforcement Learning

Dylan Hadfield-Menell* Anca Dragan Pieter Abbeel Stuart Russell
Electrical Engineering and Computer Science
University of California at Berkeley
Berkeley, CA 94709

Abstract

For an autonomous system to be helpful to humans and to pose no unwarranted risks, it needs to align its values with those of the humans in its environment in such a way that its actions contribute to the maximization of value for the humans. We propose a formal definition of the value alignment problem as *cooperative inverse reinforcement learning* (CIRL). A CIRL problem is a cooperative, partial-information game with two agents, human and robot; both are rewarded according to the human's reward function, but the robot does not initially know what this is. In contrast to classical IRL, where the human is assumed to act optimally in isolation, optimal CIRL solutions produce behaviors such as active teaching, active learning, and communicative actions that are more effective in achieving value alignment. We show that computing optimal joint policies in CIRL games can be reduced to solving a POMDP, prove that optimality in isolation is suboptimal in CIRL, and derive an approximate CIRL algorithm.

Dylan Hadfield-Menell, Stuart J. Russell, Pieter Abbeel, and Anca Dragan, (2016). Cooperative inverse reinforcement learning. *Advances in neural information processing systems*, 29

- Game-theoretic formalization of *Value Alignment*
  - ..is just one possible formulation
  - example for efforts to make "Value Alignment" more rigorous

1.10:4

Outline

- Value Alignment
- **Inverse RL**
- Preference-based RL

1.10:5

Inverse Reinforcement Learning

- Instance of **Imitation Learning**; recall:
  - Given expert demonstration data $D = \{(s^i_{1:T_i}, a^i_{1:T_i})\}_{i=1}^n$ without external rewards, objectives, costs defined
  - Extract the "relevant information/model/policy" to reproduce demonstrations

- Recap: Types of Imitation Learning
  - Behavior Cloning
  - Trajectory Distribution Learning (& Constraint Learning)
  - Direct (Interactive) Policy Learning (DAgger)
  - **Inverse Reinforcement Learning**
    - Builds on the full formalism of RL

1.10:6

Inverse Reinforcement Learning

- General Idea:
  - Given expert demonstration data $D = \{(s^i_{1:T_i}, a^i_{1:T_i})\}_{i=1}^n$
  - **infer the reward function** assuming the demonstrated behavior is (approx.) optimal

- Benefits of understanding the reward function *behind* demonstrations:
  - Can apply and generalize to fully different domains, leading to different policy
  - Can be better than demonstrator

1.10:7

Inverse Reinforcement Learning

- Methods we discuss:
  - Max Margin IRL (Apprenticeship Learning)
  - Max Entropy IRL
  - Adversarial IRL

1.10:8

IRL: General Approach

- Recall the value of a policy π

$$J(\pi) = \mathbb{E}_{\xi\sim P_\pi}\left\{\sum_{t=0}^\infty \gamma^t R(s_t, a_t)\right\}$$

- Given a demonstration policy π^*, we want to find R such that for any other policy π:

$$J(\pi^*) \ge J(\pi) \quad\Leftrightarrow\quad \mathbb{E}_{\xi\sim P_{\pi^*}}\left\{\sum_{t=0}^\infty \gamma^t R(s_t, a_t)\right\} \ge \mathbb{E}_{\xi\sim P_\pi}\left\{\sum_{t=0}^\infty \gamma^t R(s_t, a_t)\right\}$$

- To simplify this, let's assume $R(s,a)$ is **linear in features** $\phi(s,a)$:

$$R(s,a) = w^\top \phi(s,a) = \sum_i w_i \phi_i(s,a) \tag{5}$$

$$\Rightarrow \quad J(\pi) = w^\top \mathbb{E}_\pi\{\textstyle\sum_{t=0}^\infty \gamma^t \phi(s_t,a_t)\} \triangleq w^\top \mu(\pi) \tag{6}$$

  and we want

$$\forall_{\pi \neq \pi^*} : \ w^\top \mu(\pi^*) \geq w^\top \mu(\pi)$$

1.10:9

Apprenticeship Learning

Apprenticeship Learning via Inverse Reinforcement Learning

Pieter Abbeel PABBEEL@CS.STANFORD.EDU
Andrew Y. Ng ANG@CS.STANFORD.EDU
Computer Science Department, Stanford University, Stanford, CA 94305, USA

Pieter Abbeel and Andrew Y. Ng, (2004). Apprenticeship learning via inverse reinforcement learning. In *Twenty-first international conference*, page 1

1.10:10

Apprenticeship Learning

- First, π^* is not really given but
  - we estimate $\mu(\pi^*) = \mathbb{E}_{\pi^*}\{\sum_{t=0}^\infty \gamma^t \phi(s_t,a_t)\}$ from the demonstration data D
  - This $\mu(\pi^*)$ is the only information used from the demonstrations
- Second, we generate a series of other policies π_i against which we discriminate π^*
- Third, formulate "discrimination" as a max margin problem:

```
initialize π_0
for i = 0, 1, 2, ... do
    w, t ← argmax_{w, t∈ℝ} t   s.t.  ‖w‖ ≤ 1 ,   ∀_{j∈{0,..,i}} : w^⊤ μ(π*) ≥ w^⊤ μ(π_j) + t
    π_{i+1} ← argmax_π J(π)     RL problem!
end for
```

1.10:11

Maximum Entropy IRL

Maximum Entropy Inverse Reinforcement Learning

Brian D. Ziebart, Andrew Maas, J.Andrew Bagnell, and **Anind K. Dey**
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
bziebart@cs.cmu.edu, amaas@andrew.cmu.edu, dbagnell@ri.cmu.edu, anind@cs.cmu.edu

Brian D Ziebart, Andrew Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning

1.10:12

Maximum Entropy IRL

[skipping details]

- First, the expert might be noisy, demonstrations ξ are assumed

$$P(\xi; w) = \frac{\exp\{w^\top \mu(\xi)\}}{\int \exp\{w^\top \mu(\xi')\}\ d\xi'}$$

- Second, find w that leads to max entropy $P(\cdot; w)$ but matches demonstrations:

$$\min_w \int P(\xi; w) \log P(\xi; w)\ d\xi$$
$$\text{s.t.}\ \ \mathbb{E}_{\xi \sim P(\xi; w)}\{\mu(\xi)\} = \mu(\pi^*)$$

1.10:13

Adversarial IRL

- Recall idea of GANs:

$$\min_G \max_D \mathbb{E}_{x \sim p_{\text{data}}}\{\log D(x)\} + \mathbb{E}_{y=G(z), z \sim p_z}\{\log[1 - D(y)]\}$$

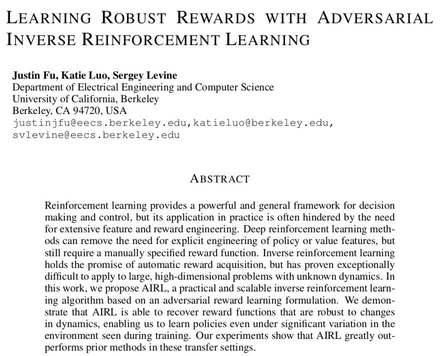
  – Train a discriminator D to label data positive, and generator's samples negative
  – Train a generator G to maximize likelihood of being classified data

  Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio, (2014). Generative adversarial nets. *Advances in neural information processing systems*, 27

- The max margin idea is very similar:
  – Find a reward function that discriminates π^* optimal from all others
  – Find other policies π_i iteratively to discriminate against

1.10:14

Adversarial IRL


LEARNING ROBUST REWARDS WITH ADVERSARIAL INVERSE REINFORCEMENT LEARNING

Justin Fu, Katie Luo, Sergey Levine
Department of Electrical Engineering and Computer Science
University of California, Berkeley
Berkeley, CA 94720, USA
justinjfu@eecs.berkeley.edu, katieluo@berkeley.edu, svlevine@eecs.berkeley.edu

ABSTRACT

Reinforcement learning provides a powerful and general framework for decision making and control, but its application in practice is often hindered by the need for extensive feature and reward engineering. Deep reinforcement learning methods can remove the need for explicit engineering of policy or value features, but still require a manually specified reward function. Inverse reinforcement learning holds the promise of automatic reward acquisition, but has proven exceptionally difficult to apply to large, high-dimensional problems with unknown dynamics. In this work, we propose AIRL, a practical and scalable inverse reinforcement learning algorithm based on an adversarial reward learning formulation. We demonstrate that AIRL is able to recover reward functions that are robust to changes in dynamics, enabling us to learn policies even under significant variation in the environment seen during training. Our experiments show that AIRL greatly outperforms prior methods in these transfer settings.

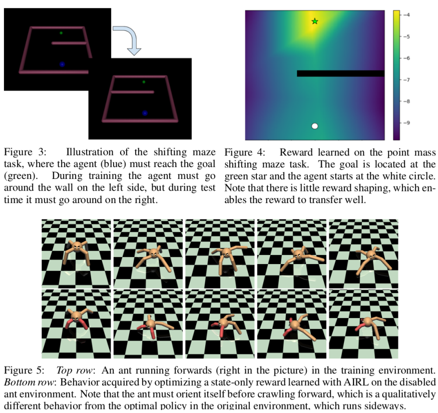


Figure 3: Illustration of the shifting maze task, where the agent (blue) must reach the goal (green). During training the agent must go around the wall on the left side, but during test time it must go around on the right.

Figure 4: Reward learned on the point mass shifting maze task. The goal is located at the green star and the agent starts at the white circle. Note that there is little reward shaping, which enables the reward to transfer well.

Figure 5: *Top row*: An ant running forwards (right in the picture) in the training environment. *Bottom row*: Behavior acquired by optimizing a state-only reward learned with AIRL on the disabled ant environment. Note that the ant must orient itself before crawling forward, which is a qualitatively different behavior from the optimal policy in the original environment, which runs sideways.

Justin Fu, Katie Luo, and Sergey Levine, (2018). Learning robust rewards with adversarial inverse reinforcement learning

Earlier similar work: [37]

1.10:15

Adversarial IRL

Algorithm 1 Adversarial inverse reinforcement learning

Obtain expert trajectories τ_i^E
Initialize policy π and discriminator $D_{\theta,\phi}$.
for step t in $\{1, \dots, \text{N}\}$ **do**
    Collect trajectories $\tau_i = (s_0, a_0, ..., s_T, a_T)$ by executing π.
    Train $D_{\theta,\phi}$ via binary logistic regression to classify expert data τ_i^E from samples τ_i.
    Update reward $r_{\theta,\phi}(s, a, s') \leftarrow \log D_{\theta,\phi}(s, a, s') - \log(1 - D_{\theta,\phi}(s, a, s'))$
    Update π with respect to $r_{\theta,\phi}$ using any policy optimization method.
end for

- The discriminator $D_{\theta,\phi}(s, a, s')$ operates on triplets and is parameterized as

$$\begin{aligned} D_{\theta,\phi}(s,a,s') &= \frac{\exp\{f_{\theta,\phi}(s,a,s')\}}{\exp\{f_{\theta,\phi}(s,a,s')\} + \pi(a|s)} \\ f_{\theta,\phi}(s,a,s') &= g_\theta(s,a) + \gamma h_\phi(s') - h_\phi(s) \\ &\approx \underbrace{r(s,a) + \gamma V(s')}_{Q(s,a)} - V(s) = A(s,a) \end{aligned}$$

  - This particular decomposition is crucial!
  - Training this way $g_\theta(s, a)$ automatically gets "reward semantics", and h_ϕ "value semantics"
  - $A(s, a)$ is called *advantage function*

1.10:16

Inverse RL Summary

- Conceptually highly interesting
- The max-margin/discrimination/adversarial idea is core to many approaches
  - Max entropy is alternative way of thinking

1.10:17

Outline

- Value Alignment
- Inverse RL
- **Preference-based RL**

1.10:18

Preference-based Learning

- In ML:
  - Given data of preference tuples $D = \{(x_1^i \succ x_2^i)\}_{i=1}^n$ (each tuple means a user preference)
  - learn a mapping $f : X \mapsto \mathbb{R}$ to minimize, e.g.

$$\sum_{i=1}^n [f(x_2^i) - f(x_1^i)]_+$$

  - Read about *label ranking, instance ranking, object ranking*

1.10:19

Preference-based RL

- Given *trajectory segment* data $D = \{(s^i_{1:T_i}, a^i_{1:T_i})\}_{i=1}^n = \{\xi^i\}_{i=1}^n$ and *preferences* $\xi^i \succ \xi^j$ for some pairs (i,j), find a reward function s.t.

$$\xi^i \succ \xi^j \quad \Rightarrow \quad \sum_{t=1}^T R(s^i_t, a^i_t) > \sum_{t=1}^T R(s^j_t, a^j_t)$$

- Long history, e.g.

  Riad Akrour, Marc Schoenauer, and Michèle Sebag, (2012). APRIL: Active preference learning-based reinforcement learning. In Peter A. Flach, Tijl De Bie, and Nello Cristianini, editors, *Machine Learning and Knowledge Discovery in Databases*, volume 7524, pages 116–131

1.10:20

Deep RL from Human Preferences

Deep Reinforcement Learning from Human Preferences

Paul F Christiano OpenAI paul@openai.com
Jan Leike DeepMind leike@google.com
Tom B Brown Google Brain* tombbrown@google.com
Miljan Martic DeepMind miljanm@google.com
Shane Legg DeepMind legg@google.com
Dario Amodei OpenAI damodei@openai.com

Paul F. Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei, (2017). Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30

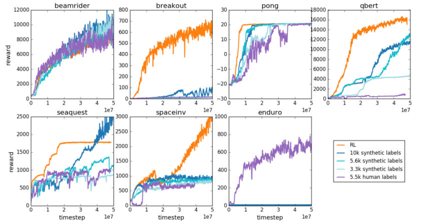


Figure 2: Results on Atari games as measured on the tasks' true reward. We compare our method using real human feedback (purple), our method using synthetic feedback provided by an oracle (shades of blue), and reinforcement learning using the true reward function (orange). All curves are the average of 3 runs, except for the real human feedback which is a single run, and each point is the average reward over about 150,000 consecutive frames.

1.10:21

Deep RL from Human Preferences

- Iteratively update a policy π and reward function R_ψ:
  - Run RL algorithm to update π with R; collect episodes
  - Select segments ξ^i from these episodes; let a human specify preferences $\xi^i \succ \xi^j$
  - Update R to minimize "preference loss"
- Assume human preferences are noisy (Bradley-Terry model)

$$P(\xi^i \succ \xi^j; R) = \frac{\exp\{\sum_{t=1}^T R(s^i_t, a^i_t)\}}{\exp\{\sum_{t=1}^T R(s^i_t, a^i_t)\} + \exp\{\sum_{t=1}^T R(s^j_t, a^j_t)\}}$$

  - Maximize likelihood $\max_\psi \sum_{\xi^i \succ \xi^j} \log P(\xi^i \succ \xi^j; R_\psi)$ for all human provided preferences

1.10:22

Robotics Application

Few-Shot Preference Learning for Human-in-the-Loop RL

Joey Hejna Stanford University jhejna@cs.stanford.edu
Dorsa Sadigh Stanford University dorsa@cs.stanford.edu

Donald Joseph Hejna III and Dorsa Sadigh, (2023). Few-shot preference learning for human-in-the-loop rl. In *Conference on Robot Learning*, pages 2014–2025

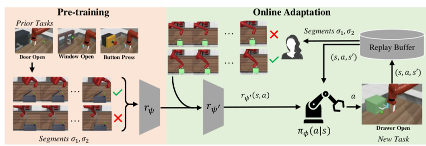


Figure 1: An overview of our method. **Pre-training (left):** In the pre-training phase we generate trajectory segment comparisons using data from a family of previously learned tasks and use them to train a reward model. **Online-Adaptation (Right):** After pre-training the reward model, we adapt it to new data from human feedback use it to train a policy for a new task in a closed loop manner.

https://sites.google.com/view/few-shot-preference-rl/home

1.10:23

1.11 Safe Learning

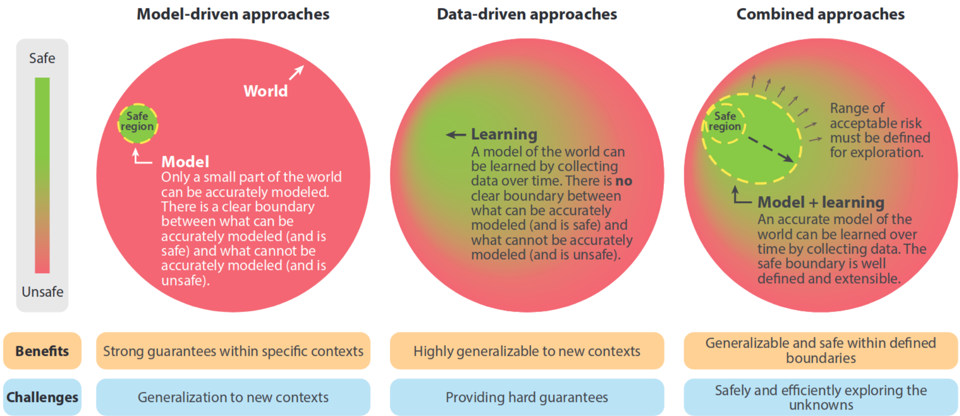
(slides by Wolfgang Hönig)

Safety

What might "safety" refer to in safe learning?

1.11:1

Motivation



Lukas Brunke, Melissa Greeff, Adam W. Hall, Zhaocong Yuan, Siqi Zhou, Jacopo Panerati, and Angela P. Schoellig, (2022). Safe Learning in Robotics: From Learning-Based Control to Safe Reinforcement Learning. *Annual Review of Control, Robotics, and Autonomous Systems*, 5:411–444

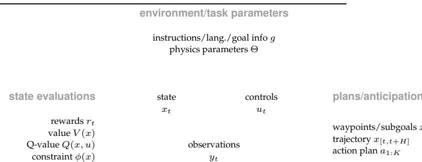
1.11:2

Outline

- Definitions of Safety and Safe Learning
- Overview of Existing Solutions (& Case Studies)
- Discussion / Open Challenges

1.11:3

What is learned?



- Consider policy $\pi : x_t \mapsto u_t$
  - Safety means (intuitively) that if we rollout π $(x_{t+1} = f(x_t, \pi(x_t)) \quad \forall t)$, we never end up in a "bad" state (e.g., collision, crash, stability/tracking) for "valid" start states x_0
  - In some cases, safety should apply while learning as well

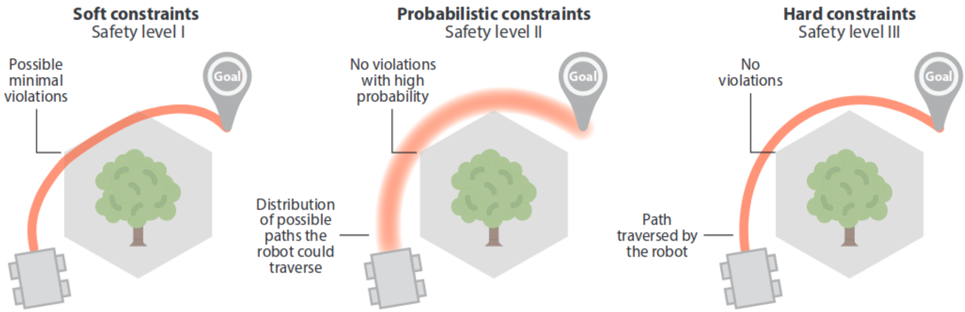
1.11:4

Definition of Safety (1)

- Dynamics $x_{k+1} = f_k(x_k, u_k, w_k)$
  - $x_k \in \mathcal{X}$ (state)
  - $u_k \in \mathcal{U}$ (action)
  - $w_k \sim \mathcal{W}$ (process noise)
  - Why f_k and not f?
- Objective $J(x_{0:N}, u_{0:N-1}) = l_N(x_N) + \sum_{k=0}^{N-1} l_k(x_k, u_k)$
- Safety constraints
  - State constraints (e.g., no collisions)
  - Input constraints (e.g., actuation limits)
  - Stability guarantees (e.g., robot converging to desired reference path)

1.11:5

Definition of Safety (2)



Lukas Brunke, Melissa Greeff, Adam W. Hall, Zhaocong Yuan, Siqi Zhou, Jacopo Panerati, and Angela P. Schoellig, (2022). Safe Learning in Robotics: From Learning-Based Control to Safe Reinforcement Learning. *Annual Review of Control, Robotics, and Autonomous Systems*, 5:411–444

1.11:6

Definition of Safety (3)

- Hard constraints (safety level 3)

$$c_k^j(x_k, u_k, w_k) \leq 0 \quad \forall k \quad \forall j$$

- Chance constraints (safety level 2)

$$Pr(c_k^j(x_k, u_k, w_k) \leq 0) \geq p^j \quad \forall k \quad \forall j \quad p^j \in [0, 1]$$

- Soft constraints (safety level 1)

$$c_k^j(x_k, u_k, w_k) \leq \epsilon_j \quad \forall k \quad \forall j$$
$l_\epsilon(\boldsymbol{\epsilon}) \geq 0$ (Cost function term)

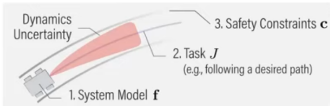
1.11:7

Definition of Safe (Control) Learning

Safe Robot Control Problem

$$\min_{\boldsymbol{\pi}_{0:N-1}, \boldsymbol{\epsilon}} \quad J(\mathbf{x}_{0:N}, \mathbf{u}_{0:N-1}) + l_\epsilon(\boldsymbol{\epsilon})$$
$$\text{s.t.} \quad \mathbf{x}_{k+1} = \mathbf{f}_k(\mathbf{x}_k, \mathbf{u}_k, \mathbf{w}_k),\ \mathbf{w}_k \sim \mathcal{W},\ \forall k \in \{0, ..., N-1\},$$
hard, probablistic, or soft safety constraints $\mathbf{c}$,
$$\mathbf{x}_0 = \bar{\mathbf{x}}_0,$$
$$\mathbf{u}_k = \boldsymbol{\pi}_k(\mathbf{x}_k)$$

Each component may be unknown or partially known!



Safe Learning Control (SLC) Design

$$\text{SLC} : (\mathcal{P}, \mathcal{D}) \mapsto \boldsymbol{\pi}_k$$

Prior Knowledge $\mathcal{P} = \{\bar{J}, \bar{\mathbf{f}}, \bar{\mathbf{c}}\}$

Data $\mathcal{D} = \{\mathbf{x}^{(i)}, \mathbf{u}^{(i)}, \mathbf{c}^{(i)}, l^{(i)}\}_{i=0}^{i=D}$

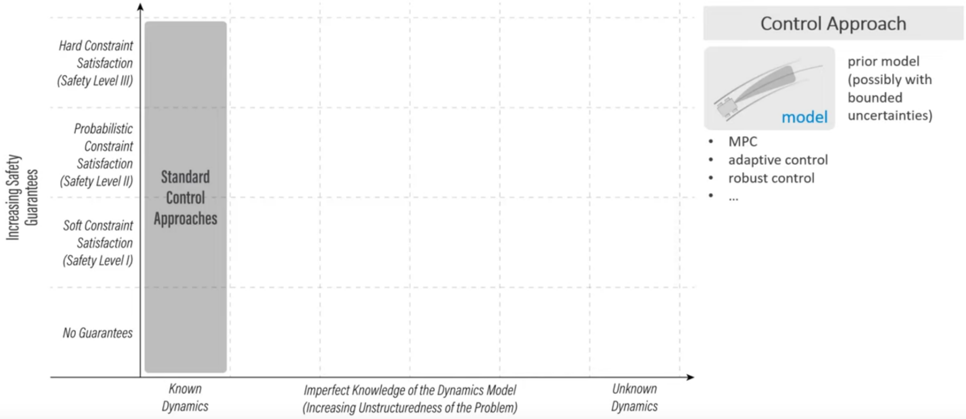
1.11:8

Relationship to (Classic) Controls

- Robust control
  - Assume disturbance bounds known
  - Find *fixed* controller that works even in the worst-case
- Adaptive controls
  - Assume environment has *varying* parameters Θ (not directly observed)
  - Controller changes *online* (e.g., by estimating Θ)
- Tube-based Model Predictive Control (MPC)
  - Robust control in MPC framework: use tighter constraints to account for unmodeled dynamics

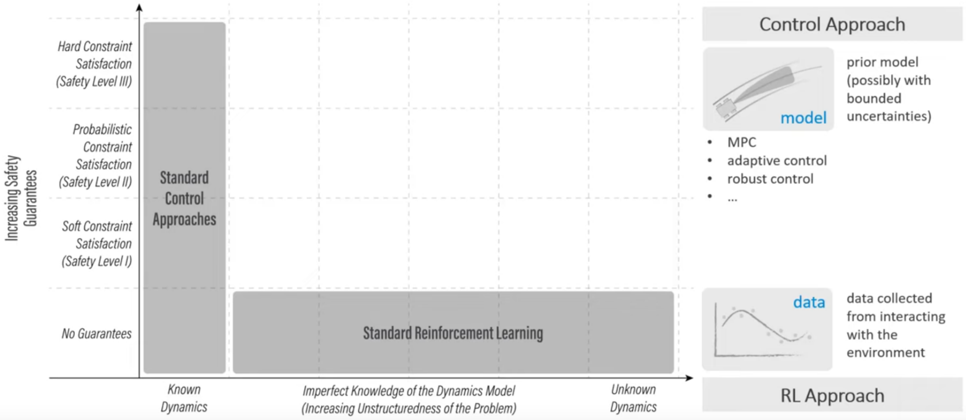
1.11:9

Relationship to (Classic) Controls



1.11:10

Relationship to (Classic) RL



1.11:11

Outline

- Definitions of Safety and Safe Learning
- **Overview of Existing Solutions (& Case Studies)**
- Discussion / Open Challenges

1.11:12

Existing Solution Strategies

(i) Safely Learn Uncertain Dynamics

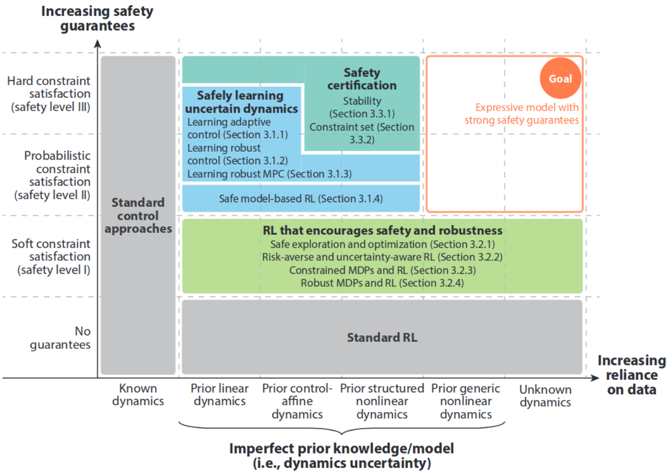
(ii) RL that Encourages Safety and Robustness

(iii) Safety Certification

[Online Adaption/Learning (dynamics, cost function, constraints, control parameters) vs Offline (update in batches)]

1.11:13

Existing Solution Strategies



Lukas Brunke, Melissa Greeff, Adam W. Hall, Zhaocong Yuan, Siqi Zhou, Jacopo Panerati, and Angela P. Schoellig, (2022). Safe Learning in Robotics: From Learning-Based Control to Safe Reinforcement Learning. *Annual Review of Control, Robotics, and Autonomous Systems*, 5:411–444

1.11:14

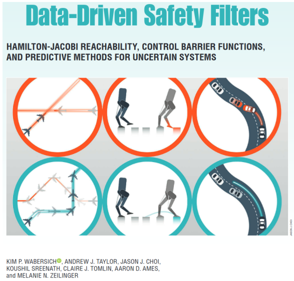
Strategy III: Safety Certification: Constraint Set

- Key idea
  - Learn policy "as usual"
  - At runtime, apply a safe action $u_{\text{safe}} = \operatorname{argmin}_u \|u - u_{\text{learned}}\|^2$ such that x_{k+1} is safe
- Safe states can be computed by
  - Control Barrier Functions (CBFs)
  - Hamilton-Jacobi Reachability Analysis
  - Predictive safety filters
    [keep track of safe control inputs that could steer back to a known safe state]

1.11:15

Strategy III: Safety Certification: Constraint Set

- More Advanced
  - If safety layer is differentiable → end-to-end training (e.g. [90])
  - Learn safety filters directly



Kim P. Wabersich, Andrew J. Taylor, Jason J. Choi, Koushil Sreenath, Claire J. Tomlin, Aaron D. Ames, and Melanie N. Zeilinger, (2023). Data-Driven Safety Filters: Hamilton-Jacobi Reachability, Control Barrier Functions, and Predictive Methods for Uncertain Systems. *IEEE Control Systems*, 43(5):137–177

1.11:16

Strategy III: Safety Certification: Stability

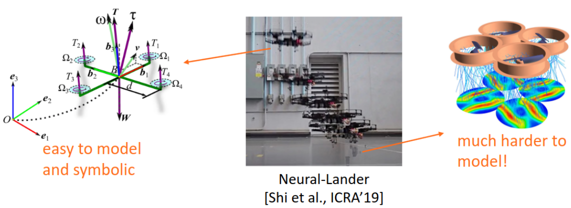
- Stability: (informal) Can the robot track the reference, even with (small) disturbances? [Formal proofs via Lyapanov functions or contraction theory]
- Typical assumptions:
  - Bounded disturbance
  - Bounded change in disturbance (Lipschitz continuous with known Lipschitz bound)
  - Unbounded control authority
- Lipschitz-based: Treat neural network as "disturbance"; limit magnitude and Lipschitz bound during training (*Spectral Normalization*) (e.g., [100])
- Region of Attraction: Lyapunov Neural Networks [88]

1.11:17

Case Study: Neural Lander (based on slides from Shi)

$$\boxed{M(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = u + f(q,\dot{q},u)}$$

- ❑ $f(q,\dot{q},u)$ is the unknown aerodynamics depending on u
- ❑ Idea: use a DNN $\hat{f}(q,\dot{q},u)$ to approximate $f(q,\dot{q},u)$
- ❑ *Q: How to guarantee stability?*



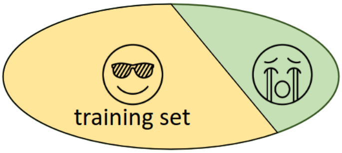
Video: https://youtu.be/FLLsGOS78ik

1.11:18

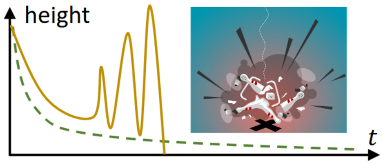
Case Study: Neural Lander (based on slides from Shi)

- Do we have to constraint the DNN? Yes! If we don't:

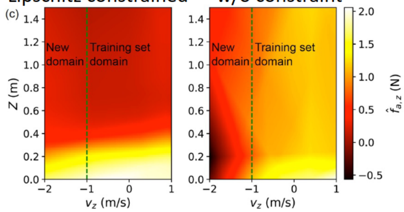
Learning perspective: $\hat{f}$ can not generalize



Control perspective: closed-loop instability



2D heatmaps of the learned $\hat{f}$



1.11:19

Strategy II: RL that Encourages Safety and Robustness

- 1. Safe Exploration and Optimization
- 2. Risk-averse RL and uncertainty-aware RL
- 3. RL for Constrained MDPs (CMDPs)
- 4. RL for Robust MDPs

1.11:20

Strategy II: RL that Encourages Safety: Safe Exploration

- Safe Exploration: only allow the policy to explore safe states

Safe Exploration in Markov Decision Processes

Teodor Mihai Moldovan MOLDOVAN@CS.BERKELEY.EDU
Pieter Abbeel PABBEEL@CS.BERKELEY.EDU
University of California at Berkeley, CA 94720-1758, USA

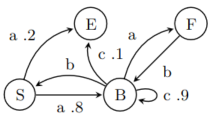


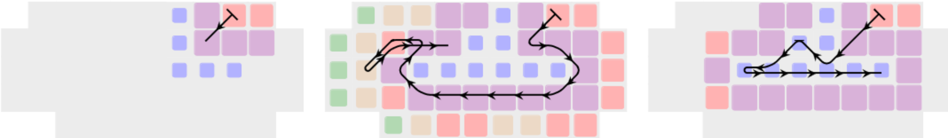
Figure 1. Starting from state S, the policy (aababab...) is safe at a safety level of .8. However, the policy (acccc...) is not safe since it will end up in the sink state E with probability 1. State-action Sa and state B can neither be considered safe nor unsafe, since both policies use them.

Teodor Mihai Moldovan and Pieter Abbeel, (2012). Safe exploration in Markov decision processes. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ICML'12, pages 1451–1458

1.11:21

Strategy II: RL that Encourages Safety: Safe Exploration

- Safe Exploration: only allow the policy to explore safe states



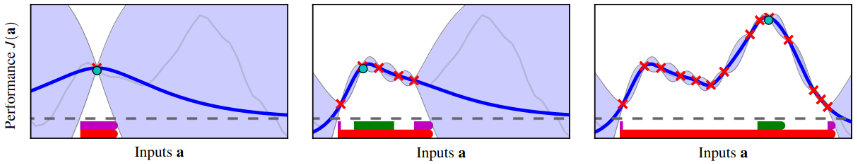
(a) Based on the available information after the first step, moving South-West is unsafe. (b) The safe explorer successfully uncovers all of the map by avoiding irreversible actions. (c) The adapted R-MAX explorer gets stuck before observing the entire map.

Teodor Mihai Moldovan and Pieter Abbeel, (2012). Safe exploration in Markov decision processes. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ICML'12, pages 1451–1458

1.11:22

Strategy II: RL that Encourages Safety: Safe Exploration

- Safe Optimization: Minimize cost function without sampling inputs that violate safety constraints, e.g., SafeOpt [7]



(a) Initial, safe parameters. (b) After 5 evaluations: local maximum found. (c) After 13 evaluations: global maximum found.

Safe set $\mathcal{S}_n$ (red): Could be potential maximizers $\mathcal{M}_n$ (green) or expanders $\mathcal{G}_n$ (magenta)

1.11:23

Case Study: SafeOpt

Algorithm 1: Modified SAFEOPT algorithm

Inputs: Domain $\mathcal{A}$
Safe threshold $J_{\min}$
GP prior $(k(\mathbf{a}_i, \mathbf{a}_j),\ \sigma_\omega^2)$
Initial, safe controller parameters $\mathbf{a}_0$

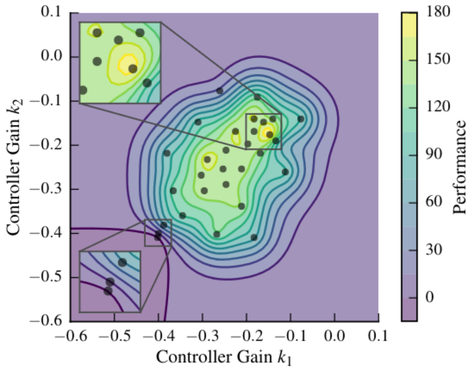
Initialize GP with $(\mathbf{a}_0,\ \hat{J}(\mathbf{a}_0))$
for $n = 1, \dots$ **do**
  $\mathcal{S}_n \leftarrow \{\mathbf{a} \in \mathcal{A} \mid l_n \geq J_{\min}\}$
  $\mathcal{M}_n \leftarrow \{\mathbf{a} \in \mathcal{S}_n \mid u_n(\mathbf{a}) \geq \max_{\mathbf{a}'} l_n(\mathbf{a}')\}$
  $\mathcal{G}_n \leftarrow \{\mathbf{a} \in \mathcal{S}_n \mid g_n(\mathbf{a}) > 0\}$
  $\mathbf{a}_n \leftarrow \operatorname{argmax}_{a \in \mathcal{G}_n \cup \mathcal{M}_n} w_n(\mathbf{a})$
  Obtain measurement $\hat{J}(\mathbf{a}_n) \leftarrow J(\mathbf{a}_n) + \omega_n$
  Update GP with $(\mathbf{a}_n,\ \hat{J}(\mathbf{a}_n))$
end

- Update sets using GPs
- From the union of safe potential maximizers or expanders, measure where the uncertainty is highest

1.11:24

Case Study: SafeOpt

Application: Safe controller gain tuning

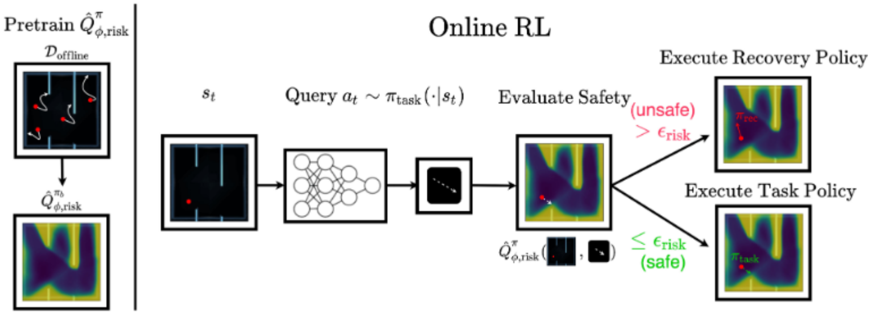


Video: https://youtu.be/GiqNQdzc5TI

1.11:25

Strategy II: RL that Encourages Safety: Safe Exploration

- Learning a safety critic: learn a Q-function that predicts "safety", e.g., [114]



Recovery RL: For intuition, we illustrate Recovery RL on a 2D maze navigation task where a constraint violation corresponds to hitting a wall.

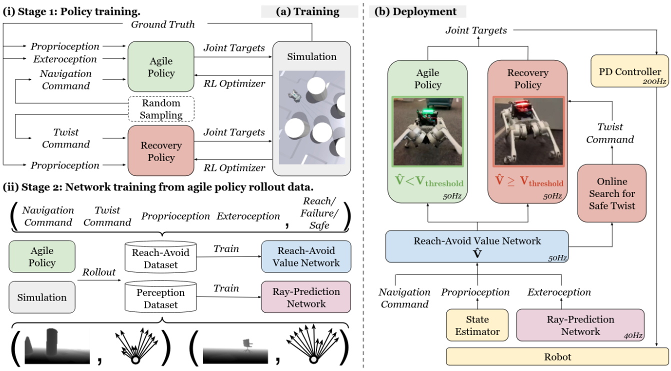
1.11:26

Strategy II: RL that Encourages Safety: Risk-averse RL

- Learn/estimate *risks* (e.g., probability of a collision)
- At runtime, prefer actions with low risk (e.g., MPC planner)

1.11:27

Case Study: Agile But Safe [52]



Web: https://agile-but-safe.github.io/

1.11:28

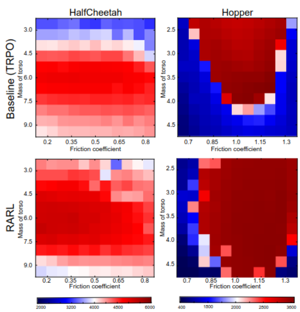
Strategy II: RL that Encourages Safety: RL for CMDPs

"However, most of the work in this area remains confined to naive simulated tasks, motivating further research on their applicability in real-world control."

1.11:29

Strategy II: RL that Encourages Safety: RL for Robust MDPs

- Robust Adversarial RL [84]



- Train two policies: a robust policy and a destabilizing adversary (that can apply random forces on the robot)
- Trained iteratively

- Domain Randomization

1.11:30

Strategy I: Safely Learn Uncertain Dynamics

- 1. Learning Adapative Control
- 2. Learning Robust Control
- 3. Learning Robust MPC
- 4. Safe Model-based RL

1.11:31

Outline

- Definitions of Safety and Safe Learning
- Overview of Existing Solutions (& Case Studies)
- **Discussion / Open Challenges**

1.11:32

Open Challenges

- Broader class of robots (hybrid dynamics, multi-robot, soft-robot, ...)
- Scalability & Sampling/Computational Efficiency
- Imperfect State Measurements
- Verification of Safety-Related Assumptions
- Automatic Inference about What is Safe

1.11:33

Discussion

- What about other learning problems?
  - Learning planners that output waypoints/trajectories (rather than a policy that outputs one action)?
  - Using humans as input (e.g., through language)?
  - Including perception (e.g., $y \mapsto u$)
  - We discussed Safe RL and safe dynamics learning; What would Safe Imitation Learning be? What would Safe Inverse RL be?
- How would you safely learn how to fly from scratch?

1.11:34

Conclusion

- Three Safety Levels: soft constraints, chance constraints, hard constraints
- Safety filters can be easily used, but are difficult to design for uncertain dynamics
- Encouraging safety has other advantages (e.g., sim-to-real transfer)
- Many practical challenges remain, especially for full robotic solutions

1.11:35

1.12 Manipulation & Grasp Learning

(slides by Marc Toussaint)

Outline

- Manipulation Intro
- Background on Grasping
- Grasp Learning Methods
- Briefly: Other Manipulation Learning

1.12:1

Manipulation is a Core Challenge in Robotics!

- Recall the "Robotics Essentials Lecture"
  - Robotics is about Articulated Multibody Systems
  - Objects in the environment are part of the "multibody system" (slide 21); have their own DOFs, but are not articulated
  - hybrid dynamics: on-off switching of manipulability; friction, stiction, slip, non-point contacts

- Think back about the last 5 lectures & exercises
  - dynamics learning, imitation learning, RL, InvRL, safe learning
  - Most work: state space ↔ robot configuration (Hopper, Walker, helicopter, UAVs, quadropeds)
  - Few works involved game environments: SpaceInvaders, Pong
  - Some works about image-based manipulation of single object: image ↔ state

1.12:2

Manipulation – Definition

- Matt Mason:

  Manipulation is when an agent moves things other than itself.

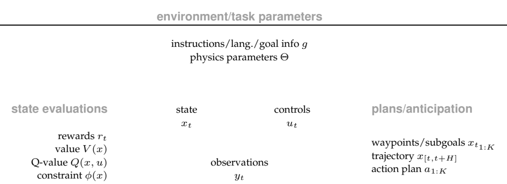
  Matthew T. Mason, (2018). Toward Robotic Manipulation. *Annual Review of Control, Robotics, and Autonomous Systems*, 1(1):1–28

- My view: *General-purpose Manipulation* ↔ *Ability to reach* any *physically possible environment configuration*

- Earlier work/definitions was fully focussed on grasping; now includes pushing, throwing, sticking, tools, ropes, any means...
- Great Lecture:
  Russ Tedrake, (2023). Robotic Manipulation - Lecture Website

1.12:3

Manipulation Learning



- What is learned?

- Policy: Image → Controls
  - Grounded in MDP formalism: $x_t, u_t \mapsto r_t, x_{t+1}$
  - is about the control process in fine time resolution

- Solutions/Constraints: Image → grasp pose, push pose
  - Not about the control process; no MDP formalism; no rewards, but $x \mapsto$success/no-success
  - The learned model predicts successful grasps, push poses, throw parameters, etc

  – These are then executed using standard control theory

1.12:4

Outline

- Manipulation Intro
- **Background on Grasping**
- Grasp Learning Methods
- Briefly: Other Manipulation Learning

1.12:5

Grasping Background

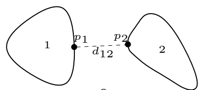
See also Chapter 12 of

Kevin M. Lynch and Frank C. Park, (2017). *Modern Robotics*

1.12:6

Contacts

- Contact between two bodies – definitions:
  – configuration $q = (q_1, q_2)$ (with $q_i \in \text{SE}(3)$ pose of ith body)
  – Their shapes define the **pairwise signed-distance** $d_{12}(q_1, q_2)$ (and its gradient)
  – Two nearest points p_1, p_2 are called **witness points**
  – We also have the contact normal $n \in \mathbb{R}^3$



- Multiple contact forces on one body:
  – One body, C contact points at position p_i, each creates **wrench** $(f_i, \tau_i) \in \mathbb{R}^6$ at p_i, totals:

$$f^{\text{total}} = \sum_{i=1}^{C} f_i \ , \quad \tau^{\text{total}} = \sum_{i=1}^{C} \tau_i + f_i \times (p_i - c)$$

  – Newton-Euler equation describes the resulting acceleration:

$$\begin{pmatrix} f^{\text{total}} \\ \tau^{\text{total}} \end{pmatrix} = \begin{pmatrix} m\dot{v} \\ I\dot{w} + w \times Iw \end{pmatrix}$$

1.12:7

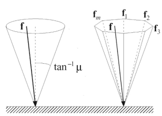
Since *"Manipulation is when an agent moves things other than itself"* these equations "fully describe" what manipulation is about: Creating contact forces to appropriately accelerate objects.

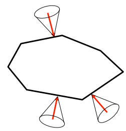
1.12:8

Contacts

- Contact Friction:
  – Point finger can not transmit torque $\Rightarrow \quad \tau_i = 0 \quad$ (better: patch models)
  – Point finger sticks only when tangentil force $f^{=} \le \mu f^{\perp} \quad (f^{\perp} = nn^\top f,\ f^{=} = f - f^{\perp})$

– The set $F_i = \{f_i : f_i^{=} \le \mu f_i^{\perp}\}$ is called the **friction cone**





- **Force closure:**
  – A **contact configuration** $\{(p_i, n_i)\}_{i=1}^C$ with friction coeff μ creates force closure
    $\Leftrightarrow$ we can generate (counter-act) arbitrary f^{total} and τ^{total} by choosing $f_i \in F_i$ appropriately.
    $\Leftrightarrow$ The *positive linear span of the fiction cones* covers the whole space of $(f^{\text{total}}, \tau^{\text{total}}) \in \mathbb{R}^6$

1.12:9

Force Closure & Force Closure Metric & Form Closure & Caging

- Force closure: The contacts can apply an arbitrary wrench (=force-torque) to the object.
- Force closure metric: Limit finger force $|f_i| \le 1$ and compute radius (=origin-distance) of convex hull
- Form closure: The object is at an isolated point in configuration space. Note: form closure $\Leftrightarrow$ frictionless force closure
- Caging: The object is not fixated, but cannot escape

1.12:10

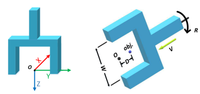
Outline

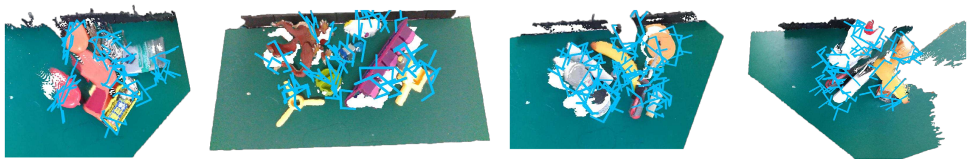
- Manipulation Intro
- Background on Grasping
- **Grasp Learning Methods**
- Briefly: Other Manipulation Learning

1.12:11

Grasp Learning

- What is learned?
  – Simplified parallel gripper:
  – Input: RGB-D image of scene
  – Output: Set of **grasps** (=gripper poses $q^{\text{gripper}} \in \text{SE}(3)$) in the scene:





  – Alternative output: A network that can score any proposed grasp

- Training data: pairs of scene (usually converted to **point cloud** P_s) and grasps

$$D = \{\ (P_s, \{q_{s,i}\}_{i=1}^{G_s})\ \}_{s=1}^S$$

1.12:12

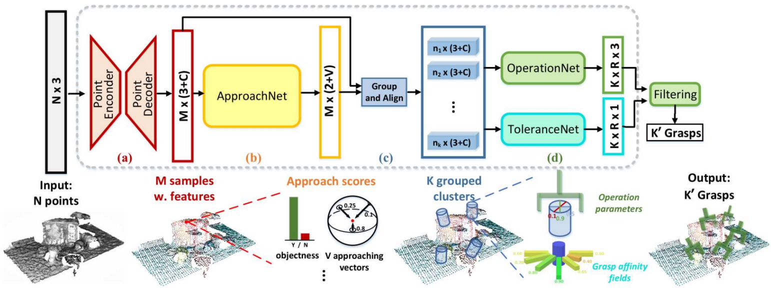
GraspNet 1

GraspNet-1Billion: A Large-Scale Benchmark for General Object Grasping

Hao-Shu Fang, Chenxi Wang, Minghao Gou. Cewu Lu[1]
Shanghai Jiao Tong University
fhaoshu@gmail.com, {wcx1997,gmh2015,lucewu}@sjtu.edu.cn

Hao-Shu Fang, Chenxi Wang, Minghao Gou, and Cewu Lu, (2020). Graspnet-1billion: A large-scale benchmark for general object grasping. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11444–11453

- Focusses on data collection (details later) $D = \{(P, \{(\underbrace{p \in P, v, D, R}_{q^{\text{gripper}} \in \text{SE}(3)}, w)_i\})\}$
- Given data, they propose architecture
  - First PCL $\to v$/success classifier per point p
  - Then predict D, R, w
  - with separate loss functions for each part



1.12:13

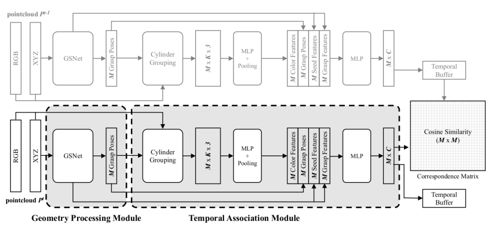
GraspNet 2

AnyGrasp: Robust and Efficient Grasp Perception in Spatial and Temporal Domains

Hao-Shu Fang, Chenxi Wang, Hongjie Fang, Minghao Gou, Jirong Liu, Hengxu Yan, Wenhai Liu, Yichen Xie, Cewu Lu, *Member, IEEE*

Hao-Shu Fang, Chenxi Wang, Hongjie Fang, Minghao Gou, Jirong Liu, Hengxu Yan, Wenhai Liu, Yichen Xie, and Cewu Lu, (2023). Anygrasp: Robust and efficient grasp perception in spatial and temporal domains. *IEEE Transactions on Robotics*

- Much more complex architecture
  https://youtu.be/dNnLgAGreec
- Also dynamic (temporally stable) predictions:
  https://www.youtube.com/watch?v=207Uo0xeLlk



1.12:14

Other Grasp Learning Work

- Classic: Identifying "antipodal" grasps in point clouds:
  Andreas Ten Pas, Marcus Gualtieri, Kate Saenko, and Robert Platt, (2017). Grasp Pose Detection in Point Clouds. *The International Journal of Robotics Research*, 36(13-14):1455–1473
- Classic: DexNet family:
  Jeffrey Mahler, Jacky Liang, Sherdil Niyaz, Michael Laskey, Richard Doan, Xinyu Liu, Juan Aparicio Ojea, and Ken Goldberg, (2017). Dex-Net 2.0: Deep Learning to Plan Robust Grasps with Synthetic Point Clouds and Analytic Grasp Metrics

  https://www.youtube.com/watch?v=i6K3GI2_EgU

- More from the "RL" side ("closed loop grasping"):

  Shuran Song, Andy Zeng, Johnny Lee, and Thomas Funkhouser, (2020). Grasping in the wild: Learning 6dof closed-loop grasping from low-cost demonstrations. *IEEE Robotics and Automation Letters*, 5(3):4978–4985

  https://www.youtube.com/watch?v=UPJjpIhXpZ8
- Contact-GraspNet

  Martin Sundermeyer, Arsalan Mousavian, Rudolph Triebel, and Dieter Fox, (2021). Contact-graspnet: Efficient 6-dof grasp generation in cluttered scenes. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 13438–13444

  https://www.youtube.com/watch?v=qRLKYSLXElM
- Using Diffusion Models

  Julen Urain, Niklas Funk, Jan Peters, and Georgia Chalvatzaki, (2023). Se (3)-diffusionfields: Learning smooth cost functions for joint grasp and motion optimization through diffusion. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5923–5930

  https://www.youtube.com/watch?v=Tk6l3WsPGMY

1.12:15

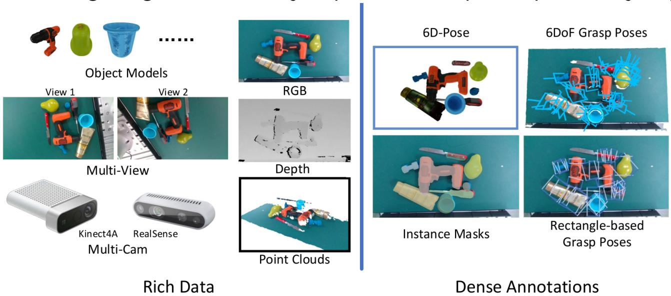
Grasp Data Collection

- My view:
  - All of the above papers show: If we have good data, we have good ideas on how to design ML architectures to predict grasps
  - Data Collection is the key!
- Two approaches:
  - Model-based labels (grasp theory, force closure)
  - Simulation-based labels

1.12:16

Model-based Grasp Labels

- GraspNet-1Billion and DexNet 2.0 papers:
  - For every point in the scene, for every (or sampled) approach direction, every offset/roll/width
  - Compute a classical grasp score: Force closure metric
  - Requires knowledge of ground truth object poses and shapes → precise object pose estimation



1.12:17

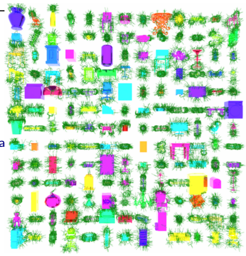
Model-based Grasp Labels

- So, force closure theory is the origin of wisdom here!
- The learning machinery "only" transfers it to the real world – predicting force closure grasps based on real RGB-D

- Cp. to imitation learning from a privileged expert! Here the privileged expert is the force closure metric assuming known object shapes.

1.12:18

Simulation-based Grasp Labels



Clemens Eppner, Arsalan Mousavian, and Dieter Fox, (2021). Acronym: A large-scale gra EEE International Conference *on Robotics and Automation (ICRA)*, pages 6222–6227

- Use generic rigid body physics simulator:
  - Throw random objects (from `ShapeNet`) into a scene (and render RGB-D image)
  - generate random grasps – smartly engineered!
  - Close and lift gripper – measure in-hand motion during both phases
  - "we simulate 17.744 million grasps, out of which 59.21% (ap- proximately 10.5 million grasps) succeed."
- So, the physics simulator (=Newton-Euler equations + contact models) is the origin of wisdom here!
  - Again, cp. to imitation learning from privileged expert (=simulation)

1.12:19

Grasp Learning Summary

- Rather advanced for standard parallel gripper; less for more complex hands
- In my view, proper data generation is key – existing methods still have deficits
- Given proper data, the advances in learning are unstoppable (stronger architectures, diffusion, etc)

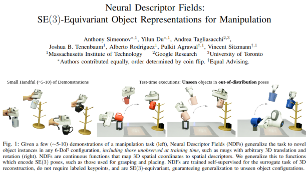
1.12:20

Manipulation Learning

- Manipulation is more than "pick-and-place"
  - manipulating articulated objects
  - pushing, throwing
  - rolling, spinning, balancing/stacking, etc.

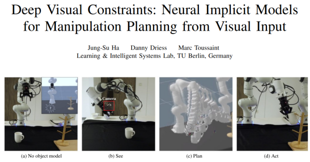
1.12:21

Recall: Extracting Constraints in Imitation Learning

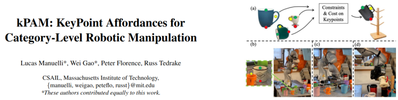

Neural Descriptor Fields:
SE(3)-Equivariant Object Representations for Manipulation

Anthony Simeonov*,1, Yilun Du*,1, Andrea Tagliasacchi2,3,
Joshua B. Tenenbaum1, Alberto Rodriguez1, Pulkit Agrawal†,1, Vincent Sitzmann†,1
1Massachusetts Institute of Technology 2Google Research 3University of Toronto
*Authors contributed equally, order determined by coin flip. †Equal Advising.

Fig. 1: Given a few (∼5-10) demonstrations of a manipulation task (left), Neural Descriptor Fields (NDFs) generalize the task to novel object instances in any 6-DoF configuration, *including those unobserved at training time*, such as mugs with arbitrary 3D translation and rotation (right). NDFs are continuous functions that map 3D spatial coordinates to spatial descriptors. We generalize this to functions which encode SE(3) poses, such as those used for grasping and placing. NDFs are trained self-supervised for the surrogate task of 3D reconstruction, do not require labeled keypoints, and are SE(3)-equivariant, guaranteeing generalization to unseen object configurations.


Deep Visual Constraints: Neural Implicit Models
for Manipulation Planning from Visual Input

Jung-Su Ha Danny Driess Marc Toussaint
Learning & Intelligent Systems Lab, TU Berlin, Germany


kPAM: KeyPoint Affordances for Category-Level Robotic Manipulation

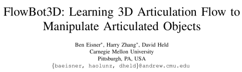
Lucas Manuelli*, Wei Gao*, Peter Florence, Russ Tedrake

CSAIL, Massachusetts Institute of Technology,
{manuelli, weigao, peteflo, russt}@mit.edu
These authors contributed equally to this work.

- Extract "constraints of success", but eventually pick-and-place

1.12:22

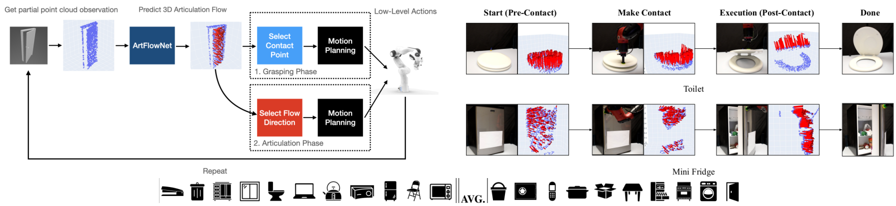
Manipulating Learning for Articulated Objects


FlowBot3D: Learning 3D Articulation Flow to
Manipulate Articulated Objects

Ben Eisner*, Harry Zhang*, David Held
Carnegie Mellon University
Pittsburgh, PA, USA
{baeisner, haolunz, dheld}@andrew.cmu.edu

Ben Eisner, Harry Zhang, and David Held, (2024). Flow-Bot3D: Learning 3D Articulation Flow to Manipulate Articulated Objects

- Assumes "gripper can be attached to any point on surface"
- Learn a mapping $P \mapsto$ flow field $F_p \in \mathbb{R}^3$ for each $p \in P$
  `https://drive.google.com/file/d/1jiEHT--WQec5diEJE6a4dMJkBnP3d36B/view`



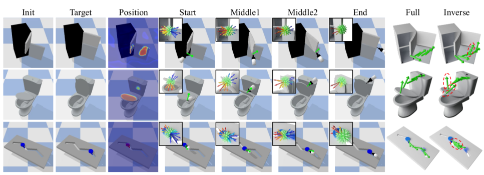
1.12:23

- Similar earlier work:


UMPNet: Universal Manipulation Policy Network for Articulated Objects

Zhenjia Xu Zhanpeng He Shuran Song
Columbia University
https://ump-net.cs.columbia.edu/

Zhenjia Xu, Zhanpeng He, and Shuran Song, (2022). Universal manipulation policy network for articulated objects. *IEEE robotics and automation letters*, 7(2):2447–2454



1.12:24

Conclusions

- Manipulation Learning is often beyond the MDP and RL framework!
- We often don't learn low-level policies, but:
  - Predicting grasps in an RGB-D scene
  - Predicting manipulability (flow) of articulated objects from RGB-D
  - Predicting keypoints/waypoints of interaction
- BUT, I think this is sooo far away from truely understanding/learning General-purpose Manipulation!

1.12:25

1.13 TAMP & Language

(slides by Marc Toussaint)

Remaining Lectures

- June 25: TAMP & Language
- July 2: Multi-Robot Learning
- July 9: Robot Learning Discussion – Lecture Feedback – Exam Info

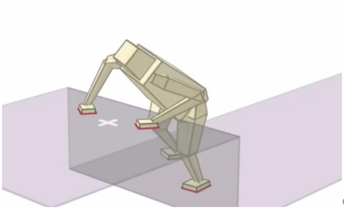
1.13:1

Outline

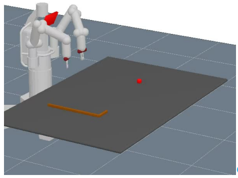
- Background on Task and Motion Planning (TAMP)
- Learning in TAMP
- Language in Robotics
- LLMs & TAMP

1.13:2

Task and Motion Planning (TAMP) examples:

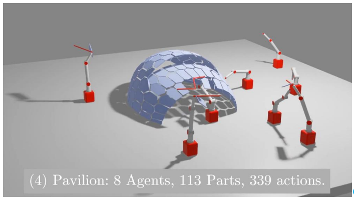


Mordatch et al: CIO (SIGGRAPH'12)



Toussaint at al: LGP (RSS'18)

Garrett et al: PDDLStream (ICAPS'20)



Hartmann et al. (IROS 20)

1.13:3

Task and Motion Planning (TAMP)

- What is the right level of "abstraction" to reason about manipulation?
  - Low-level motor commands? (Torques?)
  - Mid-level kinematic commands? (6D endeff target position/velocity)
  - Actions/skills? (Pick, place, push, throw, hit, *how long is the list?*)

1.13:4

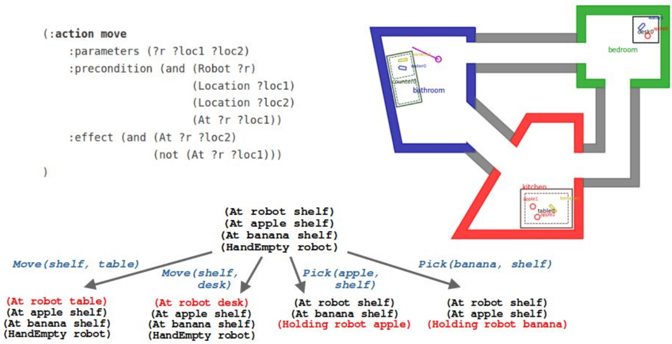
Abstractions

- What does the AI/RL researcher say about abstractions?
  - Hierarchical MDPs, Options, Hierarchical RL
  - (Classical AI: Landmarks in A* search)
  - Abstraction learning is hard:
    - Given action primitives → state abstractions clear (Konidaris' work)
    - Given state abstractions → action primitives clear ("skill discovery")
    - Classical ideas for state abstractions: identifying bottlenecks (=doors in configuration space; McGovern, Barto 2001)
  - Modern view: Data-driven: Assume tons of demonstrations and cluster-segment them

- What does the Roboticist say about abstractions?
  - Force level, motion level, task level
  - Task level: discrete symbolic state and actions (STRIPS/PDDL)

1.13:5

STRIPS/PDDL



- A symbolic state s_t is a set of grounded literals
- A symbolic action operators defines a precondition and effect
- Eventually, **his defines the set of possible successor states $s_{t+1} \in$ succ(s_t)**

1.13:6

Task and Motion Planning

- Task-level is defined by
  - symbols (predicates), objects (constants), and action operators
  - initial state s_0, goal sentence, action operators imply succ(s_t)
- Motion-level is defined by
  - world configuration space $\mathcal{X}$, goal configurations $\mathcal{X}_{\text{goal}} \subseteq \mathcal{X}$
  - feasible space $\mathcal{X}_{s,\theta} \subseteq \mathcal{X}$ depending on logic state s and *entry point* θ (action parameter)

    [$\mathcal{X}_{s,\theta}$ is called *foliation*, or multi-modal space $\quad \to \quad$ **multi-modal motion planning (MMMP)**]
- Path-Finding formulation of TAMP:
  - Find sequence of (s_i, τ_i) of symbolic states and continuous feasible paths τ_i that lead to goal:
  - Paths: $\tau_i : [0,1] \to \mathcal{X}_{s_i,\theta_i}$
  - Continuity: $\tau_i(0) = \tau_{i\text{-}1}(1)$
  - Entry points: $\theta_i = \tau_{i\text{-}1}(1)$ (e.g. action parameter, grasp, lower-dim feature of $\tau_{i\text{-}1}(1)$)
  - Goal: $s_K \models \text{goal}, \tau_K(1) \in \mathcal{X}_{\text{goal}}$

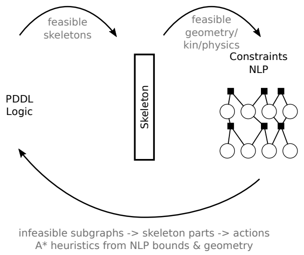
Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez, (2021). Integrated Task and Motion Planning. *Annual Review of Control, Robotics, and Autonomous Systems*, 4(1):265–293

1.13:7

TAMP as Logic-Geometric Program (LGP)

$$\min_{\substack{x:[0,KT]\to\mathcal{X}\\ s_{1:K}}} \int_0^{KT} c(\underline{x}(t))\,dt$$

$$\begin{aligned} \text{s.t.}\quad & x(0)=x_0,\\ & \forall_{t\in[0,T]}:\ \bar\phi(\underline{x}(t), s_{k(t)}) \le 0\\ & \forall_{k\in\{1,..,K\}}:\ \hat\phi(\underline{x}(t_k), s_{k\text{-}1}, s_k) \le 0\\ & s_K \models \text{goal},\ \forall_{k\in\{1,..,K\}}:\ s_k \in \text{succ}(s_{k\text{-}1}) \end{aligned}$$



- Skeleton $s_{1:K}$ defines schedule of physical modes
- Constraints $\hat\phi, \bar\phi$ define correct physics **differentiable**

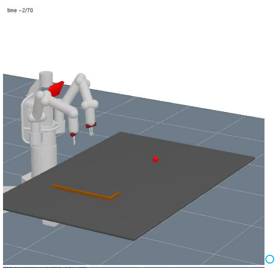
[inequalities subsume equalities; $\underline{x} = (x, \dot x, \ddot x)$]

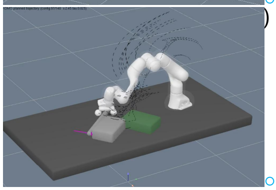
- Solving implies searching over $s_{1:K}$ and solving the corresponding NLP

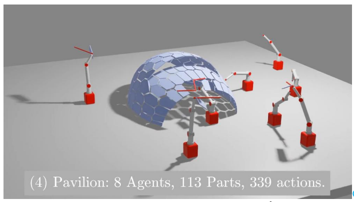
Marc Toussaint, (2015). Logic-Geometric Programming: An Optimization-Based Approach to Combined Task and Motion Planning. In *IJCAI*, pages 1930–1936
Marc A. Toussaint, Kelsey Rebecca Allen, Kevin A. Smith, and Joshua B. Tenenbaum, (2018). Differentiable physics and stable modes for tool-use and manipulation planning

1.13:8

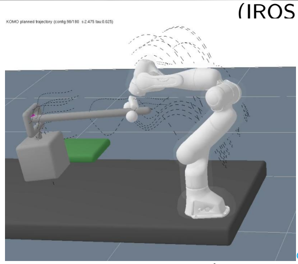
renderings(!) of example solutions...



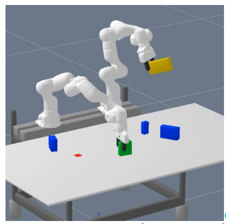




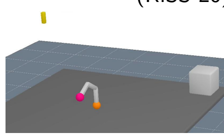
(IROS 20)



(IROS 20)



(R:SS 20)



1.13:9

Abstractions

- What does "LGP" say about abstractions?
  - There are two levels: the convex level (NLP), and the non-convex (discrete decisions)

1.13:10

Outline

- Intro to Task and Motion Planning (TAMP)
- **Learning in TAMP**

- Language in Robotics
- LLMs & TAMP

1.13:11

Is model-based TAMP a dead end?

- LGP formulates TAMP as model-based optimization problem
  - Assumption of having a world model is unrealistic (state estimation from vision ill-posed...)
  - High computation time for large problems – why plan from scratch every time?
- Opportunities for learning:
  - **Replace exact model by learned constraints $\phi(x)$**
    - The LGP definition actually only needs constraints $\phi(x)$, no explicit world model
    - Instead of hand-defining these from a model → image-conditional neural models $\phi_\theta(x; \mathcal{I})$
  - **Learn to predict plans**
    - Instead of solving from scratch, learn to predict promising actions $a_{1:K}$ from the scene image

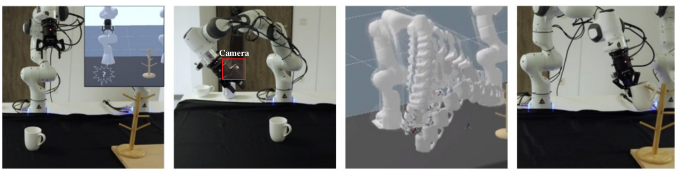
1.13:12

- Replace exact model by learned constraints $\phi(x)$:

1.13:13

Deep Visual Constraints: Neural Implicit Models for Manipulation Planning from Visual Input

Jung-Su Ha Danny Driess Marc Toussaint
Learning & Intelligent Systems Lab, TU Berlin, Germany



(a) No object model (b) See (c) Plan (d) Act

- Learn $\phi(x, \mathcal{I})$ with V input images $\mathcal{I}$ s.t.:
  - $\phi(x; \mathcal{I}) = 0 \quad \Leftrightarrow \quad x$ is correct grasp
  - $\phi(x; \mathcal{I}) = 0 \quad \Leftrightarrow \quad x$ is correct hanging
- Data generating in simulation:
  - Collect trial-and-error data on correct grasps and hanging

Jung-Su Ha, Danny Driess, and Marc Toussaint, (2022). Deep visual constraints: Neural implicit models for manipulation planning from visual input. *IEEE Robotics and Automation Letters*, 7(4):10857–10864

1.13:14

Deep Visual Constraints: Network Architecture

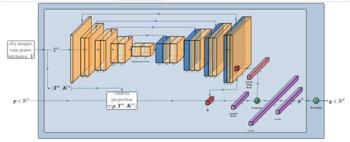


Fig. 3: PIFO (i) encodes the images $\mathcal{I}$ as pixel-wise feature images $\mathcal{F}$ via U-net, (ii) projects the query point $p \in \mathbb{R}^3$ into the pixel coordinate $z \in \mathbb{R}^3$ using known camera geometry, and (iii) computes the object representation vector $y \in \mathbb{R}^d$ by extracting the local image features at the projected points.

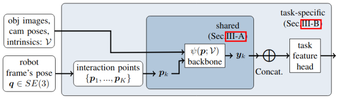


Fig. 2: The interaction feature prediction scheme of DVC

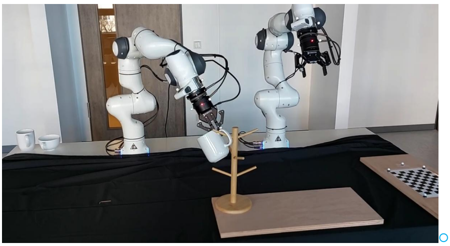
Jung-Su Ha, Danny Driess, and Marc Toussaint, (2022). Deep visual constraints: Neural implicit models for manipulation planning from visual input. *IEEE Robotics and Automation Letters*, 7(4):10857–10864

- Camera views $\mathcal{I} = \{(I^1, K^1), ..., (I^V, K^V)\}$
  Wanted: image-based constraint model
  $$\phi(x; \mathcal{I})$$
- First train a d-dimensional **field representation**
  $$y(p; \mathcal{I}) = \frac{1}{V} \sum_i \text{MLP}(\text{UNet}(I^i, K^i(x)), K^i(x))$$
  [$p \in \mathbb{R}^3$, pre-trained for shape decoding (SDF prediction)]
- Function is queried at finite set of *interaction points* $p_1(x), .., p_K(x)$ to get the feature
  $$\phi(x; \mathcal{I}) = \text{MLP}(y(p_1(x); \mathcal{I}), .., y(p_K(x); \mathcal{I}))$$
  [fine-tuned for manipulation success (trial & error in sim)]

1.13:15

Deep Visual Constraints

(No search over skeletons, no reactive MPC, just optimal path for given sequence of constraints.)



1.13:16

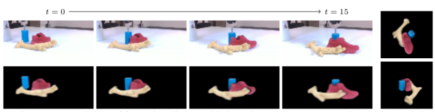
Similar: Learn Dynamics Constraints

Learning Multi-Object Dynamics with Compositional Neural Radiance Fields

Danny Driess TU Berlin **Zhiao Huang** UC San Diego **Yunzhu Li** MIT **Russ Tedrake** MIT **Marc Toussaint** TU Berlin

Danny Driess, Zhiao Huang, Yunzhu Li, Russ Tedrake, and Marc Toussaint, (2023). Learning multi-object dynamics with compositional neural radiance fields. In *Conference on Robot Learning*, pages 1755–1768

https://dannydriess.github.io/compnerfdyn/



(a) Bottom row renderings of forward predictions with dynamic model, top row ground truth (b) novel view

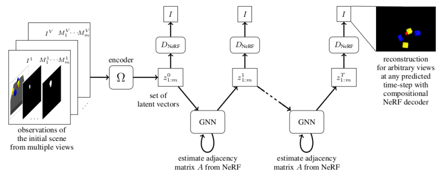


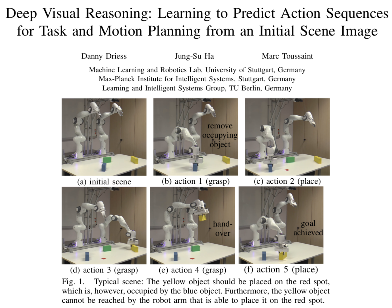
Figure 2: Overview of the dynamics prediction framework. The initial scene observations are encoded with Ω into a set of latent vectors $z_{1:m}$, each representing the objects individually. The GNN dynamics model predicts the evolution of the latent vectors. At each step, the predicted latent vectors can be rendered into an arbitrary view with the compositional NeRF decoder. Refer to the appendix for visualizations of Ω and the GNN.

- Each object has a latent code z_i^t
- learn dynamics $z_{1:m}^t \mapsto z_i^{t+1}$!

1.13:17

- Learning to predict plans..

1.13:18


Deep Visual Reasoning: Learning to Predict Action Sequences for Task and Motion Planning from an Initial Scene Image

Danny Driess Jung-Su Ha Marc Toussaint

Machine Learning and Robotics Lab, University of Stuttgart, Germany
Max-Planck Institute for Intelligent Systems, Stuttgart, Germany
Learning and Intelligent Systems Group, TU Berlin, Germany

(a) initial scene (b) action 1 (grasp) (c) action 2 (place)
(d) action 3 (grasp) (e) action 4 (grasp) (f) action 5 (place)

Fig. 1. Typical scene: The yellow object should be placed on the red spot, which is, however, occupied by the blue object. Furthermore, the yellow object cannot be reached by the robot arm that is able to place it on the red spot.

Danny Driess, Jung-Su Ha, and Marc Toussaint, (2020). Deep Visual Reasoning: Learning to Predict Action Sequences for Task and Motion Planning from an Initial Scene Image

- Data collection $D = \{\left(S^i, g^i, a^i_{1:K^i}, F^i\right)\}_{i=1}^n$
  - with scene S^i, goal g^i, actions $a^i_{1:K^i}$, feasibility F^i
  - random generated "in simulation", **model-based TAMP solver used to label feasibility**
- Train a sequential policy:
  $\pi(a_k; g, a_{1:k\text{-}1}, S) =$
  $P(\exists_{K>K} \exists_{a_{k+1:K}} : a_{1:K}\text{feasible} \,|\, a_k, g, a_{1:k\text{-}1}, S)$
  - Similar to language model: Predict next "token" a_k given previous $a_{1:k\text{-}1}$ conditional g, S

1.13:19

Deep Visual Reasoning: Network Architecture

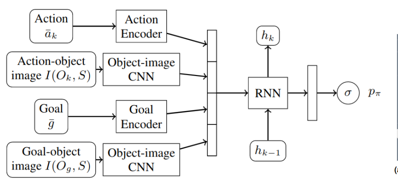
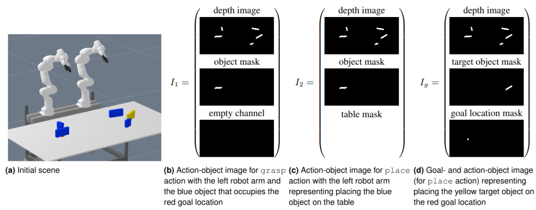


Figure 3. Proposed neural network architecture.

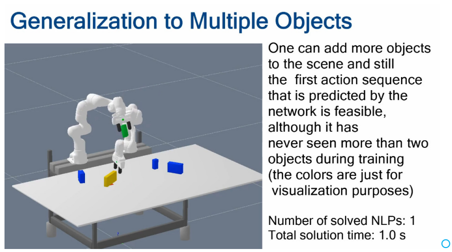


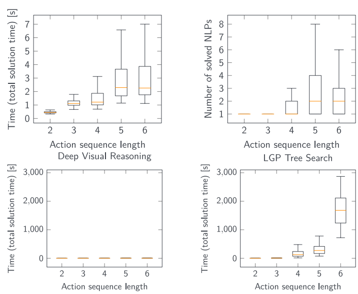
(a) Initial scene **(b)** Action-object image for `grasp` action with the left robot arm and the blue object that occupies the red goal location **(c)** Action-object image for `place` action with the left robot arm representing placing the blue object on the table **(d)** Goal- and action-object image (for `place` action) representing placing the yellow target object on the red goal location

- Uses RNN – modern version would use transformer
- Special encoding of predicates $\bar a, \bar g$ and references O (as masks)

1.13:20

Deep Visual Reasoning: Results





- Often, the first proposed action sequence is feasible

1.13:21

Outline

- Intro to Task and Motion Planning (TAMP)
- Learning in TAMP
- **Language in Robotics**
- LLMs & TAMP

1.13:22

Robots That Use Language: A Survey

Stefanie Tellex[1], Nakul Gopalan[2], Hadas Kress-Gazit[3], and Cynthia Matuszek[4]

Stefanie Tellex, Nakul Gopalan, Hadas Kress-Gazit, and Cynthia Matuszek, (2020). Robots That Use Language. *Annual Review of Control, Robotics, and Autonomous Systems*, 3(1):25–55

- Great survey on Natural Language Robot Interaction
  - Using natural language to command robots, set tasks
  - Using natural language to instruct robots, e.g. as part of demonstrations
  - Different to standard NLP or dialog systems: **language needs to be physically grounded**

1.13:23

Natural Language Robot Interaction: Examples

Figure 1: Robots used for language-based interactions.

from [112]

- robot asks for help
- human sets task (with language & gesture)
- robot "reads/comprehends" wikihow
- demonstrations via dialog
- human sets task (nagivation)
- ...
- human sets task (object identification)
- human sets task (navigation)
- human sets task (manipulation)

1.13:24

Natural Language Robot Interaction: Datasets

Dataset	Type of Data	Link to dataset
MARCO dataset (111)	Navigation instructions given to a robot to navigate a map, and the route followed.	www.cs.utexas.edu/users/ml/clamp/navigation/
Scene dataset (98)	Images and descriptions of objects in the image.	rtw.ml.cmu.edu/tacl2013_lsp/
Cornell NLVR dataset (168)	Pairs of images and logical statements about them which are true or false.	lic.nlp.cornell.edu/nlvr/
CLEVR dataset (84)	Images and question-answer pairs.	cs.stanford.edu/people/jcjohns/clevr/
Embodied Question Answering (47)	Pairs of questions and answers in simulated 3D environments. The agent needs to search the environment to find the answer.	embodiedqa.org
Visual Question Answering in Interactive Environments (65)	Pairs of questions and answers in different simulated 3D environments.	github.com/danielgordon10/thor-iqa-cvpr-2018
Room-to-Room (R2R) Navigation (4)	Panoramic views in real buildings, paired with instructions to be followed.	bringmeaspoon.org/
H2R lab language grounding datasets (9, 64)	Predicate based sub-goal conditions paired with natural language instructions.	github.com/h2r/language_datasets
Cornell Instruction Following Framework (17, 125)	Data for three separate navigation domains in 3D environments, containing instructions paired with trajectories.	github.com/clic-lab/ciff
MIT Spatial Language Understanding dataset (92, 172)	Pairs of language command and trajectories for navigation and mobile manipulation.	people.csail.mit.edu/stefie10/slu/

Table 2: Datasets used in Language Grounding and Robotics

"Data sets typically consist of natural language paired with some form of sensor-based context information about the physical environment"

1.13:25

- Previous survey highlights substantial literature on Natural Language Robot Interaction *before* rise of LLMs

  Example: https://youtu.be/VqSb-ZZuIwI?t=2523

1.13:26

CLIP (Contrastive Language-Image Pre-training)

Learning Transferable Visual Models From Natural Language Supervision

Alec Radford [*1] **Jong Wook Kim** [*1] **Chris Hallacy** [1] **Aditya Ramesh** [1] **Gabriel Goh** [1] **Sandhini Agarwal** [1] **Girish Sastry** [1] **Amanda Askell** [1] **Pamela Mishkin** [1] **Jack Clark** [1] **Gretchen Krueger** [1] **Ilya Sutskever** [1]

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, and Jack Clark, (2021). Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763

"We demonstrate that the simple pre-training task of predict- ing which caption goes with which image is an efficient and scalable way to learn SOTA image representations from scratch on a dataset of 400 million (image, text) pairs collected from the internet."

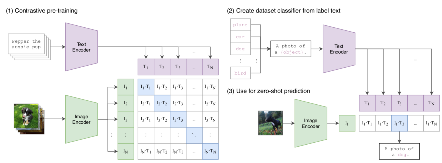


Figure 1. Summary of our approach. While standard image models jointly train an image feature extractor and a linear classifier to predict some label, CLIP jointly trains an image encoder and a text encoder to predict the correct pairings of a batch of (image, text) training examples. At test time the learned text encoder synthesizes a zero-shot linear classifier by embedding the names or descriptions of the target dataset's classes.

[Contrastive Training: "maximize the cosine similarity of the image and text embeddings of the N real pairs in the batch while minimizing the cosine similarity of the embeddings of the $N^2 - N$ incorrect pairings.]

1.13:27

CLIPort

CLIPort: What and Where Pathways for Robotic Manipulation

Mohit Shridhar [1,†] Lucas Manuelli [2] Dieter Fox [1,2]
[1]University of Washington [2]NVIDIA
mshr@cs.washington.edu lmanuelli@nvidia.com fox@cs.washington.edu

Mohit Shridhar, Lucas Manuelli, and Dieter Fox, (2022). Cliport: What and where pathways for robotic manipulation. In *Conference on Robot Learning*, pages 894–906

https://cliport.github.io/

"CLIPort: a language-conditioned imitation-learning agent that combines the broad semantic understanding (what) of CLIP with the spatial precision (where) of Transporter"

- Trains a policy $\pi : (y_i, l_l) \mapsto a_t$
  - top-down orthographic RGB-D y_t, language instruction l_t, pick-n-place 2D coordinates a_t

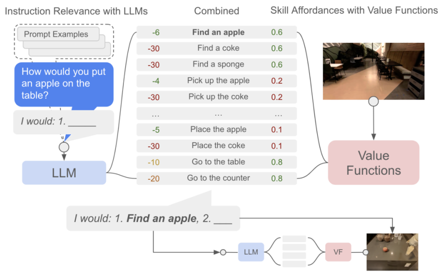
1.13:28

SayCan

Do As I Can, Not As I Say: Grounding Language in Robotic Affordances

Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, and Ryan Julian, (2023). Do as i can, not as i say: Grounding language in robotic affordances. In *Conference on Robot Learning*, pages 287–318

https://say-can.github.io/



- Use a LLM (PaLM) to predict *multiple* actions (with probabilities)
- Multiply each option with *affordance prediction* (= probability of success)

1.13:29

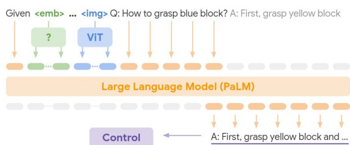
PaLM-E

PaLM-E: An Embodied Multimodal Language Model

Danny Driess [1,2] Fei Xia [1] Mehdi S. M. Sajjadi [3] Corey Lynch [1] Aakanksha Chowdhery [3] Brian Ichter [1] Ayzaan Wahid [1] Jonathan Tompson [1] Quan Vuong [1] Tianhe Yu [1] Wenlong Huang [1] Yevgen Chebotar [1] Pierre Sermanet [1] Daniel Duckworth [3] Sergey Levine [1] Vincent Vanhoucke [1] Karol Hausman [1] Marc Toussaint [2] Klaus Greff [3] Andy Zeng [1] Igor Mordatch [3] Pete Florence [1]
[1]Robotics at Google [2]TU Berlin [3]Google Research

Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence, (2023). PaLM-E: An Embodied Multimodal Language Model

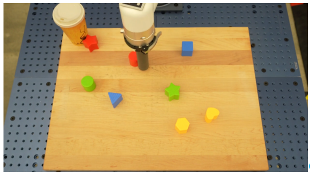
https://palm-e.github.io/

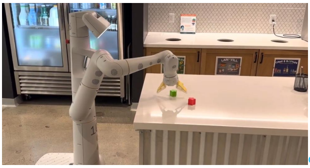


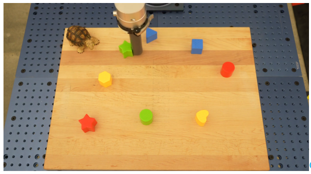
- Input: *Multi-modal sentence:*
  - Interleaves words, images (with segmentation), vectors, reference-keywords
  - All token-encoded
  - Various image encodings (ViT, object-centric ViT, OSRT, NeRFs pre-trained)
- Output:
  - Sequences of action primitives (previously trained, RT-1)

1.13:30

"Bring me the rice chips from the drawer"


"Push red blocks to the coffee cup"


"Bring me the green star"


"Push green blocks to the turtle"

1.13:31

Example input/output

- Prompt: Given <img>. Q: How to grasp the green object?.
  Target: A: First grasp the orange object and place it on the table, then grasp the green object.
- Prompt: Given <img>. Q: How to stack the white object on top of the red object?.
  Target: A: First grasp the green object and place it on the table, then grasp the white object and place it on the red object.

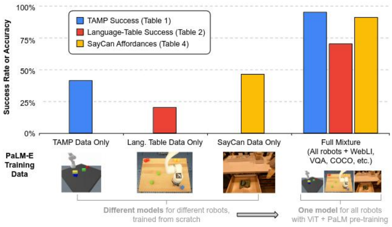
1.13:32

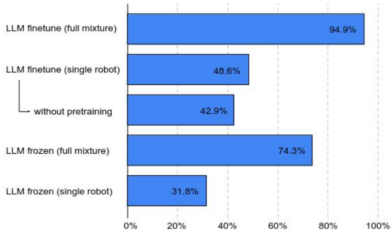
PaLM-E Evaluations

- Data sets:
  - **TAMP data (generated by our LGP-TAMP planner)**
  - Table data (previous RT1 paper)
  - SayCan data
  - Other visual/language data: WebLI, VQA, COCO, etc.
- Pre-taining:
  - LLM backbone: language, VQA (WebLI, VQA, COCO)
  - Encodings: reconstruction, auto-encoding
- Ablation studies:
  - Varying transformer sizes
  - generalization (to unseen object situations, esp. higher number of objects)
  - freezing, refining, full-learning of backbone LLM or encodings
  - with full/partial choice of data sets & sizes
  - various image encodings

1.13:33

PaLM-E evaluations





	Object-centric	LLM pre-train	Embodied VQA q_1	q_2	q_3	q_4	Planning p_1	p_2
SayCan (oracle afford.) (Ahn et al., 2022)		✓	-	-	-	-	38.7	33.3
PaLI (zero-shot) (Chen et al., 2022)		✓	-	0.0	0.0	-	-	-
PaLM-E (ours) w/ input enc:								
State	✓(GT)	✗	99.4	89.8	90.3	88.3	45.0	46.1
State	✓(GT)	✓	**100.0**	96.3	95.1	93.1	55.9	49.7
ViT + TL	✓(GT)	✓	34.7	54.6	74.6	91.6	24.0	14.7
ViT-4B single robot	✗	✓	-	45.9	78.4	92.2	30.6	32.9
ViT-4B full mixture	✗	✓	-	70.7	93.4	92.1	74.1	74.6
OSRT (no VQA)	✓	✓	-	-	-	-	71.9	75.1
OSRT	✓	✓	99.7	**98.2**	**100.0**	**93.7**	**82.5**	**76.2**

Baselines				Failure det.	Affordance
PaLI (Zero-shot) (Chen et al., 2022)				0.73	0.62
CLIP-FT (Xiao et al., 2022)				0.65	-
CLIP-FT-hindsight (Xiao et al., 2022)				0.89	-
QT-OPT (Kalashnikov et al., 2018)				-	0.63
PaLM-E-12B trained on	from scratch	LLM+ViT pretrain	LLM frozen		
Single robot	✓	✗	n/a	0.54	0.46
Single robot	✗	✓	✓	**0.91**	0.78
Full mixture	✗	✓	✓	**0.91**	0.87
Full mixture	✗	✓	✗	0.77	**0.91**

1.13:34

Follow Up: RT-2

RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, Pete Florence, Chuyuan Fu, Montse Gonzalez Arenas, Keerthana Gopalakrishnan, Kehang Han, Karol Hausman, Alexander Herzog, Jasmine Hsu, Brian Ichter, Alex Irpan, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Lisa Lee, Tsang-Wei Edward Lee, Sergey Levine, Yao Lu, Henryk Michalewski, Igor Mordatch, Karl Pertsch, Kanishka Rao, Krista Reymann, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Pierre Sermanet, Jaspiar Singh, Anikait Singh, Radu Soricut, Huong Tran, Vincent Vanhoucke, Quan Vuong, Ayzaan Wahid, Stefan Welker, Paul Wohlhart, Jialin Wu, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, Brianna Zitkovich

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, and Ayzaan Wahid, (2023). Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning*, pages 2165–2183

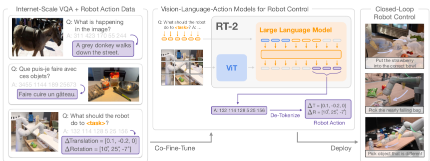


Figure 1: RT-2 overview: we represent robot actions as another language, which can be cast into text tokens and trained together with Internet-scale vision-language datasets. During inference, the text tokens are de-tokenized into robot actions, enabling closed loop control. This allows us to leverage the backbone and pretraining of vision-language models in learning robotic policies, transferring some of their generalization, semantic understanding, and reasoning to robotic control. We demonstrate examples of RT-2 execution on the project website: `robotics-transformer2.github.io`.

- quasi-continuous actions (trained end-to-end):

"terminate Δpos_x Δpos_y Δpos_z Δrot_x Δrot_y Δrot_z gripper_extension".

A possible instantiation of such a target could be: "1 128 91 241 5 101 127". The two VLMs that we finetune in our experiments, PaLI-X [16] and PaLM-E [17], use different tokenizations. For PaLI-X,

1.13:35

Conclusion

- Levels of abstraction: Force, motion, task
- Task and Motion "Planning": Core problem formulation of robotic AI
  - TAMP theory & solvers are fully model-based
  - Clear opportunities for learning: constraint learning, learning to predict plans
- Language ↔ task & action level
  - Lots of classical literature on *language grounding*
  - Connecting natural language with typical robot task descriptions (STRIPS/PDDL)
- Huge recent focus on marrying LLMs + TAMP + robotics

1.13:36

1.14 Multi-Robot Learning

(slides by Wolfgang Hönig)

Motivation: Multi-Robot Systems

- Multiple robots (typically in a team) with a common goal
- Typical promises:
  - Achieve goal faster
  - Achieve goal more robustly
  - Higher flexibility (esp. heterogeneous systems)
  - Cheaper (?)

1.14:1

Motivation: Multi-Robot Systems

- Successful (industrial) solutions
  - Warehouse logistics (Amazon Robotics, former Kiva systems)

  - Aerial Drone shows (Intel, Verity Studios)

1.14:2

Motivation: Multi-Robot System Challenges

- Controls: additional constraint for inter-robot collision avoidance
- Decision Making: information sharing, task assignment, curse-of-dimensionality for centralized approaches, safety/robustness for decentralized systems
- Perception: sensing team members, sensor fusion

1.14:3

Outline

- **Handling Dynamic Neighbors**
  - LSTMs

- CNNs
- DeepSets
- Graph Neural Networks

- Multi-Agent Reinforcement Learning (MARL)
- Discussion / Open Challenges

1.14:4

Dynamic Neighbors

- Team of robots has time-varying neighbors/observations/communication links
- Often need to learn with time-varying input dimensionality
    - Example: (Distributed) collision avoidance maps observation of neighboring robots to actions $f(\mathcal{Y}) \to u$
- Learned functions need to be **permutation-invariant** and support **dynamic domain cardinality**

1.14:5

LSTMs [32]

2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
Madrid, Spain, October 1-5, 2018

Motion Planning Among Dynamic, Decision-Making Agents with Deep Reinforcement Learning

Michael Everett[‡], Yu Fan Chen[†], and Jonathan P. How[‡]

- Key idea: Feed observations of neighbors into an LSTM (closest neighbor last)

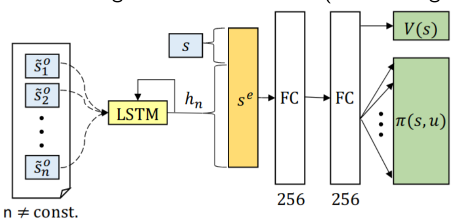


Fig. 3: Network Architecture. Observable states of nearby agents, $\tilde{s}_i^o$, are fed sequentially into the LSTM, as unrolled in Fig. 2. The final

1.14:6

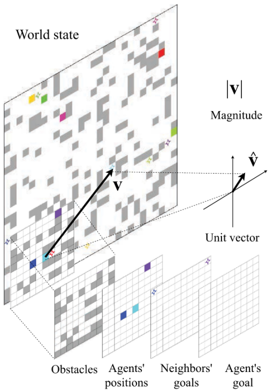
CNNs [94]


2378 IEEE ROBOTICS AND AUTOMATION LETTERS, VOL. 4, NO. 3, JULY 2019

PRIMAL: Pathfinding via Reinforcement and Imitation Multi-Agent Learning

Guillaume Sartoretti, Justin Kerr, Yunfei Shi, Glenn Wagner, T. K. Satish Kumar, Sven Koenig, and Howie Choset

- Key idea: Encode neighbor information as a picture
- Videos: `https://goo.gl/T627XD`



1.14:7

Deep Sets [131]

- Any continuous, permutation-invariant function $f(\mathcal{X})$ can be approximated:

tion of hidden state

Learns **representation** of each element

$$f(\mathcal{X}) \approx \rho\left(\sum_{x \in \mathcal{X}} \phi(x)\right)$$

superposition in hidden state

- Improvement over Convolutional NN (**CNN**): continuous space, efficiency
- Example:

 $+$  $= 9$

 $+$  $+$  $= 20$

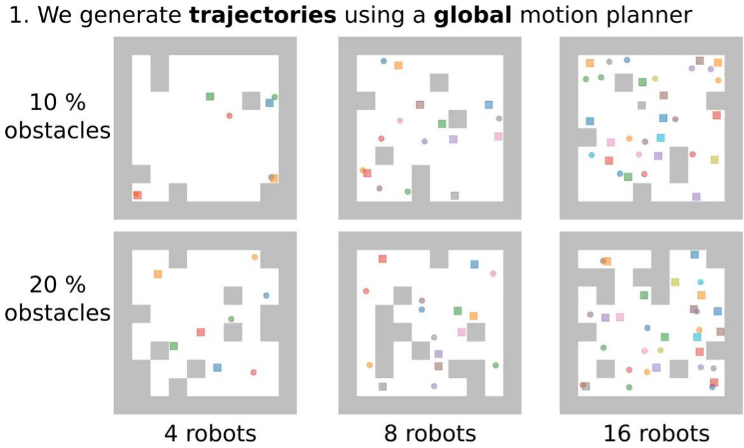
1.14:8

Case Study: GLAS [90]

- Goal: imitate (slow) centralized controller using only local observations: $\pi : y \mapsto u$
- Data: Example trajectories by solving many multi-robot motion planning instances with a centralized planner
- Approach: Behavior Cloning + Privileged Teacher

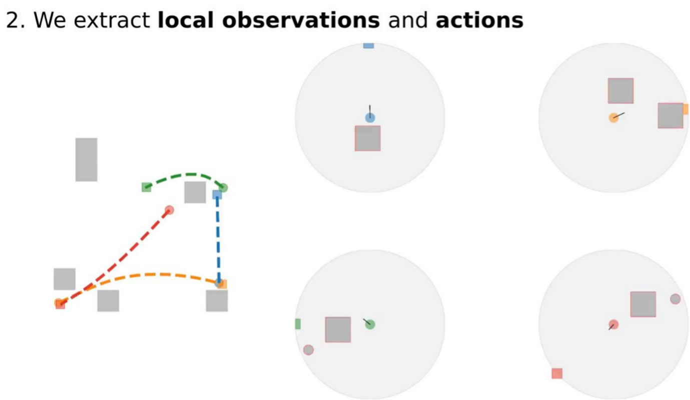
1.14:9

Case Study: GLAS [90]



1.14:10

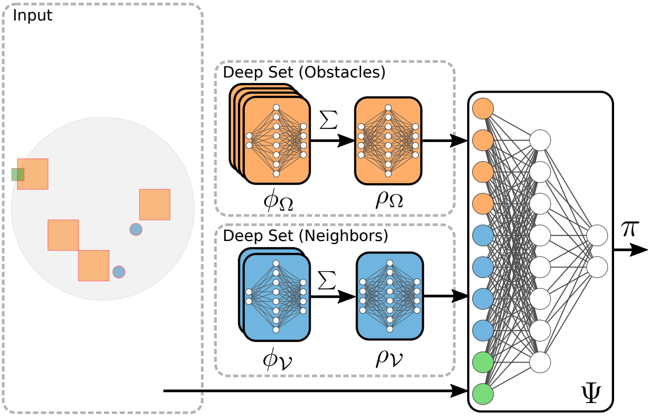
Case Study: GLAS [90]



1.14:11

Case Study: GLAS [90]

- Train (5 small feedforward networks trained jointly)



1.14:12

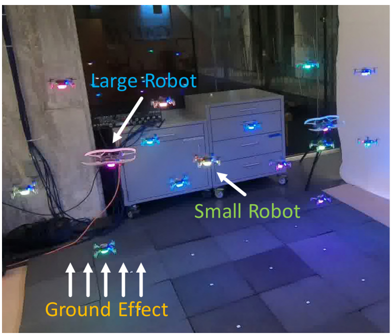
Case Study: GLAS [90]

- How would one train this in practice in pyTorch? [variable number of neighbors vs. batching]

1.14:13

Case Study: Neural-Swarm2 [98]

- Goal: predict aerodynamic interaction [unmodeled physics, as a function of neighbors' positions]



- Data: Real flight tests (synchronized trajectories with poses of robots and measured accelerations and motor commands)
- Approach: Behavior Cloning

1.14:14

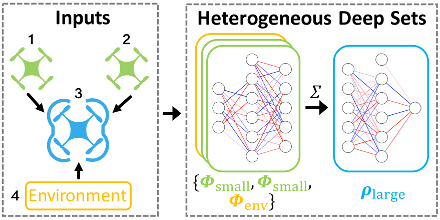
Case Study: Neural-Swarm2 [98]: Heterogeneous Deep Sets

Learns **aggregation** for type $\mathcal{I}(i)$

Learns **representation** from type $\mathcal{I}(j)$ neighbor

$$\hat{\mathbf{f}}_a^{(i)} \approx \boldsymbol{\rho}_{\mathcal{I}(i)}\left(\sum_{k=1}^{K}\sum_{\mathbf{x}^{(ij)}\in\mathbf{r}^{(i)}_{\mathrm{type}_k}} \boldsymbol{\phi}_{\mathcal{I}(j)}(\mathbf{x}^{(ij)})\right)$$

superposition in hidden state

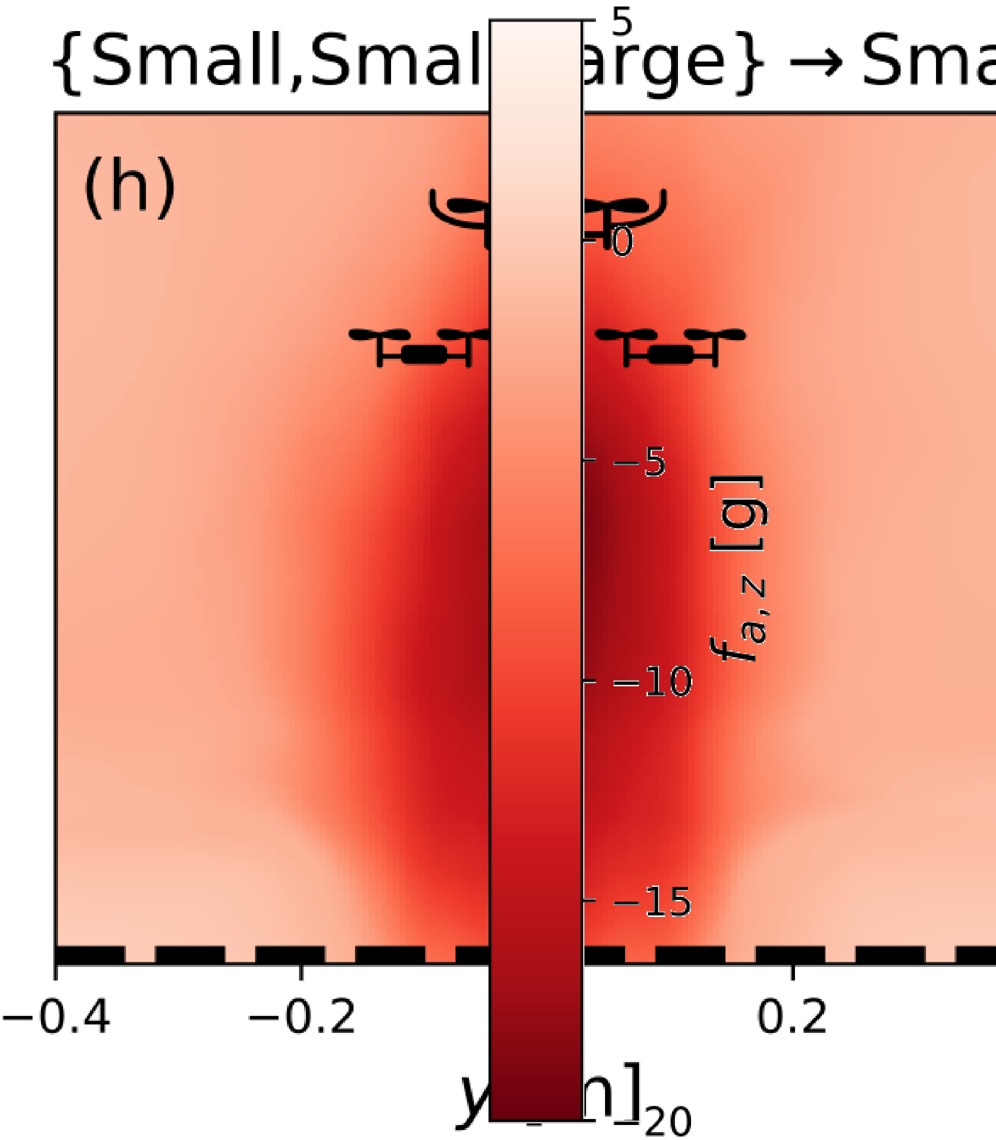


$$\mathbf{f}_a^{(3)} \approx \boldsymbol{\rho}_{\mathrm{large}}\Big(\boldsymbol{\phi}_{\mathrm{small}}(\mathbf{x}^{(31)}) + \boldsymbol{\phi}_{\mathrm{small}}(\mathbf{x}^{(32)}) + \boldsymbol{\phi}_{\mathrm{env}}(\mathbf{x}^{(34)})\Big)$$

- **Expressiveness**: can approximate any K-Group permutation-invariant function

- **Efficient**: only $2K$ networks need to be trained

Case Study: Neural-Swarm2 [98]



1.14:16

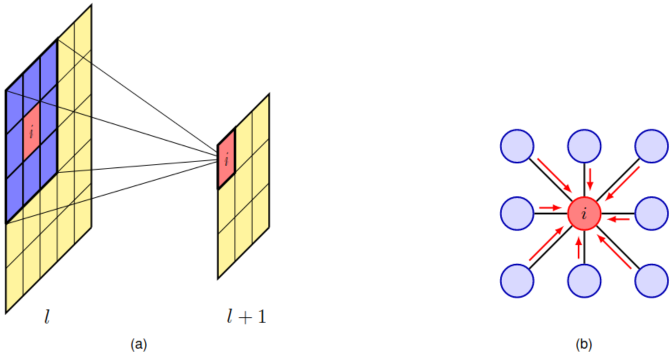
Case Study: Neural-Swarm2 [98]

https://youtu.be/Y02juH6BDxo

1.14:17

Graph Neural Networks (GNNs)

- Inspiration: CNNs as graph



Christopher M. Bishop and Hugh Bishop, (2024). *Deep Learning: Foundations and Concepts*

Graph Neural Networks (GNNs)

- Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
- Basic case: learn features for each node $n \in \mathcal{V}$
- Use L layers with D-dimensional vector $h_n^{(l)}$

Graph Neural Networks (GNNs)

Algorithm 13.1: Simple message-passing neural network

Input: Undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
Initial node embeddings $\{\mathbf{h}_n^{(0)} = \mathbf{x}_n\}$
$\text{Aggregate}(\cdot)$ function
$\text{Update}(\cdot, \cdot)$ function
Output: Final node embeddings $\{\mathbf{h}_n^{(L)}\}$

```
// Iterative message-passing
```

for $l \in \{0, \ldots, L-1\}$ **do**
  $\mathbf{z}_n^{(l)} \leftarrow \text{Aggregate}\left(\left\{\mathbf{h}_m^{(l)} : m \in \mathcal{N}(n)\right\}\right)$
  $\mathbf{h}_n^{(l+1)} \leftarrow \text{Update}\left(\mathbf{h}_n^{(l)}, \mathbf{z}_n^{(l)}\right)$
end for
return $\{\mathbf{h}_n^{(L)}\}$

Graph Neural Networks (GNNs)

- Examples for Aggregate/Update:
  - Aggregate$(\{h_m^{(l)} : m \in \mathcal{N}(n)\}) = MLP_\rho \left(\sum_{m \in \mathcal{N}(n)} MLP_\phi(h_m^{(l)}) \right)$

- $\text{Update}(h_n^{(l)}, z_n^{(l)}) = f(W_{self} h_n^{(l)} + W_{neigh} z_n^{(l)} + b)$

- Extensions to have input/output features per edge and graph [See e.g., [8]]

- Training "as usual" (on whole graphs)
- In practice: PyG `https://www.pyg.org/` or DGL `https://www.dgl.ai/`

1.14:21

Case Study: Learning to Communicate for Multi-Robot Path Finding [68]

2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

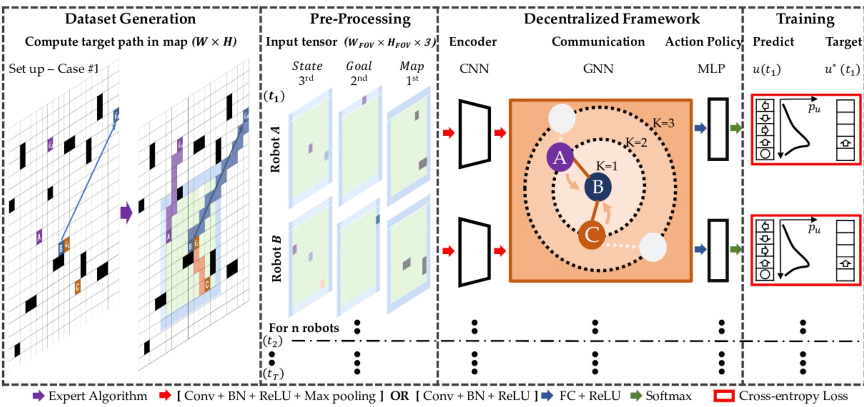
Graph Neural Networks for Decentralized Multi-Robot Path Planning

Qingbiao Li[1], Fernando Gama[2], Alejandro Ribeiro[2], Amanda Prorok[1]

- Goal: Learn how to communicate to imitate a centralized Multi-Agent Path Finding expert
- Data: Trajectories computed by a centralized expert
- Approach: IL w/ DAgger

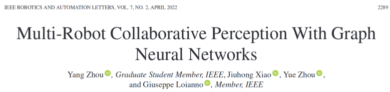
1.14:22

Case Study: Learning to Communicate for Multi-Robot Path Finding [68]



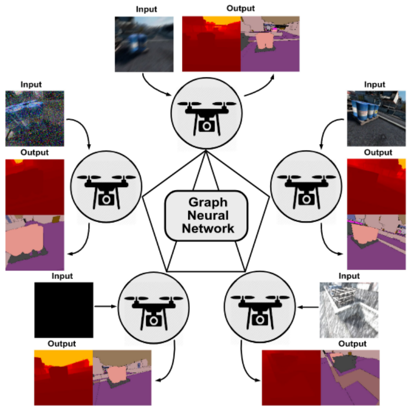
1.14:23

Case Study: Multi-Robot Perception [134]


IEEE ROBOTICS AND AUTOMATION LETTERS, VOL. 7, NO. 2, APRIL 2022 2289

Multi-Robot Collaborative Perception With Graph Neural Networks

Yang Zhou, *Graduate Student Member, IEEE*, Jiuhong Xiao, Yue Zhou, and Giuseppe Loianno, *Member, IEEE*



- Goal: Learn what to communicate for depth estimation or segmentation
- Data: Labeled Data mostly from simulator; some from real flights
- Approach: Behavior Cloning
- Video: https://youtu.be/2bdhLI3dqo0

1.14:24

GNN Applications

- Flocking (in simulation) [116, 64, 42]
- Navigation (simulation + RL) [128]
- Graph Control Barrier Function (simulation + IL w/ DAgger) [132]
- Learning to Communicate Variations [69, 42]

1.14:25

Outline

- Handling Dynamic Neighbors
  - LSTMs
  - CNNs
  - DeepSets
  - Graph Neural Networks
- **Multi-Agent Reinforcement Learning (MARL)**
- Discussion / Open Challenges

1.14:26

MARL Definition

- Single Robot: MDP $(\mathcal{S}, \mathcal{A}, P, R, P_0, \gamma)$ with state space $\mathcal{S}$, action space $\mathcal{A}$, transition probabilities $P(s_{t+1} \mid s_t, a_t)$, reward fct $r_t = R(s_t, a_t)$, initial state distribution $P_0(s_0)$, and discounting factor $\gamma \in [0, 1]$.
- Multi-Robot: Markov game $(N, \mathcal{S}, \mathcal{A}, P, R, P_0, \gamma)$ with N robots, $\mathcal{S}$ *joint* state space, $\mathcal{A} = A_1 \times A_2 \times \ldots \times A_N$ *joint* action space, reward fct $r_1, \ldots, r_N = R(s, a)$
- Goal: Find policy (or policies) that maximize expected reward

1.14:27

Rewards

- **Fully cooperative**: $r_1 = r_2 = \ldots = r_N$ [No credit assignment; difficult to train]
- **Competitive**: zero-sum games ($\sum_i r_i = 0$), prey-predator games (cooperative per team; competitive per game)
- **Mixed Cooperative-Competitive**: (local) reward shaping, to achieve a common goal

1.14:28

Learning

- **Centralized** model as stacked robot (centralized training & inference)
- **Independent Learning** each robot learns own policy (decentralized training & inference)
- **Centralized Training Decentralized Execution (CTDE)**

1.14:29

Challenges

- Non-Stationarity: if policy of other agents can't be observed, the Markov assumption is violated (e.g., distributed Q-Learning)
- Scalability: in standard policy gradient algorithms, the probability of estimating the policy gradient correctly might decrease exponentially with the number of agents [Concrete example: appendix of [71]]

1.14:30

Approaches

- Centralized critic, e.g., Multi-Agent deep deterministic policy gradient (MADDPG, [71])
- Factorized value functions, e.g., Value Decomposition Networks (VDN, [110])
- Communication Learning

1.14:31

Practical Considerations

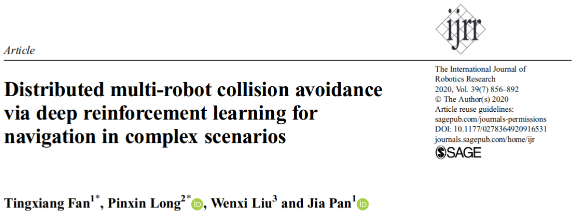
- VMAS (Vectorized Multi-Agent Simulator for Collective Robot Learning) `https://github.com/proroklab/VectorizedMultiAgentSimulator` [Simple 2D physics engine build in pyTorch]
- MARLlib `https://github.com/Replicable-MARL/MARLlib`
- More Details/Overview about MARL:

  Yutong Wang, Mehul Damani, Pamela Wang, Yuhong Cao, and Guillaume Sartoretti, (2022). Distributed Reinforcement Learning for Robot Teams: A Review. *Current Robotics Reports*, 3(4):239–257

  James Orr and Ayan Dutta, (2023). Multi-Agent Deep Reinforcement Learning for Multi-Robot Applications: A Survey. *Sensors*, 23(7):3625

1.14:32

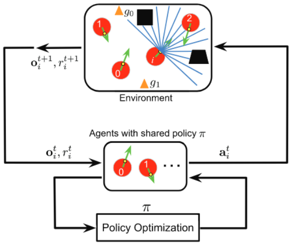
Case Study: Distributed Collision Avoidance (Ground) [33]

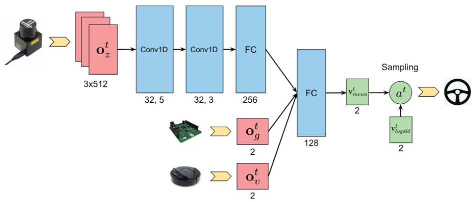

Article

Distributed multi-robot collision avoidance via deep reinforcement learning for navigation in complex scenarios

The International Journal of Robotics Research
2020, Vol. 39(7) 856–892
© The Author(s) 2020
Article reuse guidelines:
sagepub.com/journals-permissions
DOI: 10.1177/0278364920916531
journals.sagepub.com/home/ijr
SAGE

Tingxiang Fan[1*], Pinxin Long[2*], Wenxi Liu[3] and Jia Pan[1]





1.14:33

Case Study: Distributed Collision Avoidance (Ground) [33]

- Goal: find decentralized policy: $\pi : y, g \mapsto u$
- Data: Collected in simulation during RL (input LIDAR, relative goal, velocity; output: action)
- Approach: PPO (centralized learning, decentralized execution; shared policy)
- Video: https://sites.google.com/view/hybridmrca

1.14:34

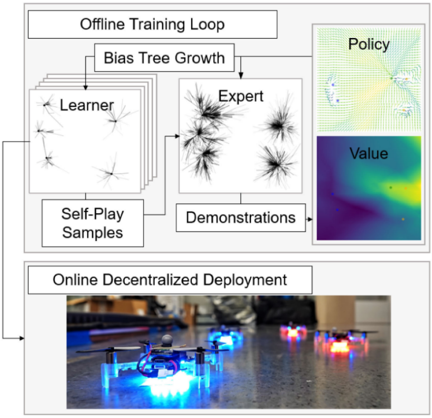
Case Study: Distributed Collision Avoidance (UAVs) [57]

- Goal: find decentralized policy: $\pi : y, g \mapsto u$
- Data: Collected in simulation during RL (input state, nearby obstacles, nearby neighbors; output: thrust per rotor)
- Approach: IPPO (centralized learning, decentralized execution; shared policy)
- Video: https://sites.google.com/view/obst-avoid-swarm-rl

1.14:35

Case Study: Neural Tree Expansion [89]

- Goal: find decentralized policies for multi-team games (e.g., reach-target avoid)



- Data: Collected with a neural-biased "expert" (large Monte-Carlo Tree Search)
- Approach: MCTS + IL + DAgger (essentially: AlphaZero in continuous state spaces)
- Video: https://youtu.be/mklbTfWl7DE

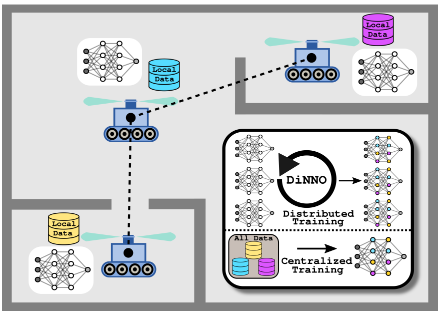
1.14:36

Outline

- Handling Dynamic Neighbors
  - LSTMs
  - CNNs
  - DeepSets
  - Graph Neural Networks
- Multi-Agent Reinforcement Learning (MARL)
- **Discussion / Open Challenges**

1.14:37

DiNNO: Distributed Neural Network Optimization [129]



- Collect data locally, local augmented Lagrangian update, share resulting weights via consensus
- Works for IL and RL
- Web: https://msl.stanford.edu/projects/dist_nn_train

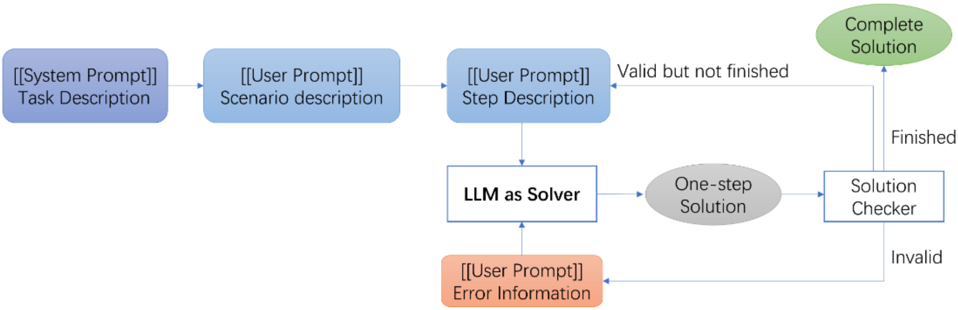
1.14:38

LLMs and Multi-Robots [18]

Why Solving Multi-agent Path Finding with Large Language Models has not Succeeded Yet

Weizhe Chen [1] Sven Koenig [1] Bistra Dilkina [1]

- (Arxiv, Jan. 2024)



1.14:39

LLMs and Multi-Robots [18]

Agent 1 is currently in (0,2), and wants to go to (3,1).
Agent 2 is currently in (1,3), and wants to go to (2,0).
The map is as follows, where '@' denotes a cell with an obstacle that an agent cannot pass, and '.' denotes an empty cell that an agent can pass.
The bottom-left cell is (0,0) and the bottom-right cell is (31,0):

....
...@
....
.@..

In the next step:
Agent 1 can move ['stay at (0, 2)', 'right to (1, 2)', 'up to (0, 3)', 'down to (0, 1)'].
Agent 2 can move ['stay at (1, 3)', 'left to (0, 3)', 'right to (2, 3)', 'down to (1, 2)'].

1.14:40

Open Challenges

- Deployment to real robots (especially RL)
- Safety (esp. partially unknown dynamics, perception)
- Interpretability (of communication)

1.14:41

Conclusion

- Multi-Robot brings new challenges
  - Large state space (or violation of Markov assumption)
  - Dynamic number of neighbors
  - Reasoning about communication
- Deep Sets: permutation invariant architecture that is easy to train and computationally efficient [useful for $\pi : x, \mathcal{N} \mapsto u$]
- GNN: Generalization of deep sets [useful for learning communication]
- Learning a decentralized policy from a centralized expert works well (IL + DAgger)
- Deployment to real robot teams remains challenging

1.14:42

2 Exercises

2.1 Weekly Exercise 1

All 4 exercises are a bit too much for a start. Question 3 is bonus.

2.1.1 Basic Inverse Kinematics

(i) Inverse kinematics (or general constraint solving) can be framed as the optimization problem

$$\min_{q \in \mathbb{R}^n} \|q - q_0\|^2 + \mu \|\phi(q)\|^2 \ , \tag{7}$$

for some constraint function $\phi : \mathbb{R}^n \to \mathbb{R}^d$. Assuming linear $\phi(q) = \phi(q_0) + J(q - q_0)$ with Jacobian J, the solution is

$$q^* = q_0 - (J^\top J + \tfrac{1}{\mu}\mathbf{I})^{-1} J^\top \ \phi(q_0) \ . \tag{8}$$

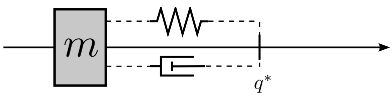
Verify this by deriving it step by step.

(ii) To enforce a hard constraint, we want to take the limit $\mu \to \infty$. But $J^\top J$ is typically not invertible (e.g., $d < n$), and you can't directly take the limit in the above solution. However, the solution to this limit is

$$q^* = q_0 - J^\top (J J^\top)^{-1} \phi(q_0) \ . \tag{9}$$

Derive this from the above. Tip: Learn about the Woodbury identity.

2.1.2 Point mass under PD control



Consider a point mass in a 1D space together with a PD control law:

- The point has mass m, and position $q(t) \in \mathbb{R}$.
- The PD controller applies linear force

$$u(t) = -k_p q(t) - k_d \dot{q}(t)$$

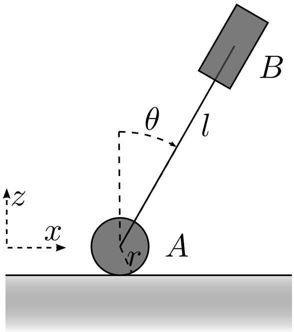
  to the point, where $k_p, k_d \in \mathbb{R}$ are positive constants.

- The resulting dynamics is $m\ddot{q}(t) = u(t)$.

(i) Given the initial state $q(0) = a, \dot{q}(0) = 0$, what is $q(t)$? (Solve the differential equation.)

(ii) The solution describes a damped oscillation around the set-point $q^* = 0$. How do you have to choose k_p and k_d such that the behavior becomes the exponential approach $q(t) = ae^{-t/\tau}$ for some time scale $\tau \in \mathbb{R}$? (This is called "critically damped".)

2.1.3 BONUS: Fun with Euler-Lagrange



Consider an inverted pendulum mounted on a wheel in the 2D x-z-plane; similar to a Segway. The exercise is to derive the Euler-Lagrange equation for this system.

(i) Describe the **pose** $p_i \in \mathbb{R}^3$ of every body in (x, z, ϕ) coordinates: its position in the x-z-plane, and its rotation ϕ relative to the world-vertical. Assume fixed parameters r: radius of the wheel, l: length of the pendulum (height of its COM).

(ii) Describe the (linear and angular) velocity $v_i = \dot{p}_i \in \mathbb{R}^3$ of every body.

(iii) Formulate the total kinetic energy $T = \frac{1}{2}\sum_i v_i^\top M_i v_i$, summing over the two bodies $i = A, B$. Note that

$$M_i = \begin{pmatrix} m_i & 0 & 0 \\ 0 & m_i & 0 \\ 0 & 0 & I_i \end{pmatrix} \tag{10}$$

with $m_i \in \mathbb{R}$ the normal mass of body i, and $I_i \in \mathbb{R}$ the rotational inertia of body i.

(iv) Formulate the potential energy U

(v) Bonus: Compute the Euler-Lagrange Equation

$$u = \frac{d}{dt}\frac{\partial L}{\partial \dot{q}} - \frac{\partial L}{\partial q} \ , \tag{11}$$

with $L = T - U$, using the minimal coordinates $q = (x, \theta)$, where x is the position of the wheel and θ the angle of the pendulum relative to the world-vertical.

2.1.4 Logistic Regression

Consider a binary classification problem with data $D = \{(x_i, y_i)\}_{i=1}^n$, $x_i \in \mathbb{R}^d$ and $y_i \in \{0, 1\}$. We define

$$f(x) = x^\top \beta \tag{12}$$

$$p(x) = \sigma(f(x)) \ , \quad \sigma(z) = 1/(1 + e^{-z}) \tag{13}$$

$$L^{\mathsf{nll}}(\beta) = -\sum_{i=1}^n \Big[y_i \log p(x_i) + (1 - y_i) \log[1 - p(x_i)] \Big] \tag{14}$$

where $\beta \in \mathbb{R}^d$ is the model parameter, $\sigma(z)$ the sigmoidal function, and $L^{\mathsf{nll}}(\beta)$ the neg-log-likelihood of the data under the model.

(i) Compute the derivative $\frac{\partial}{\partial \beta} L(\beta)$. Tip: use the fact $\frac{\partial}{\partial z}\sigma(z) = \sigma(z)(1 - \sigma(z))$.

(ii) Compute the 2nd derivative $\frac{\partial^2}{\partial \beta^2} L(\beta)$.

(iii) How is the neg-log-likelihood related to the cross-entropy? How would the above change when adding an additional regularization $\lambda \|\beta\|^2$ to the loss?

2.2 Weekly Exercise 2

2.2.1 Work with the Literature

[The links to literature sometimes point to journal sites, but they should be accessible from within TU Berlin.]

(i) Have a look at Eq. (1) of
Hava T. Siegelmann, Bill G. Horne, and C. Lee Giles, (1997). Computational capabilities of recurrent NARX neural networks. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 27(2):208–215
This paper describes a classical "NARX" model. Consider the discrete time dynamics

$$v_{t+1} = v_t + u_{t-3} \tag{15}$$
$$p_{t+1} = p_t + \tau v_{t-2} \tag{16}$$
$$y_t = p_t \ , \tag{17}$$

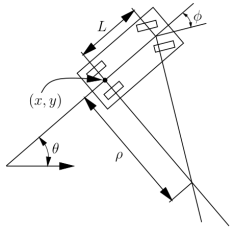
with variables (p_t, v_t), controls u_t, and sensor observation y_t. $\tau \in \mathbb{R}$ is a fixed constant. (In words: the control directly adds to the velocities – but with a delay of 3 steps! And the velocities add to the position – but with a delay of 2 steps! And we only observe position p_t, not velocities.)
Could the "NARX" model described in the paper above learn this dynamics? How would you have to choose n_u and n_y?

(ii) Also have a look at Eq. (1) of
Marc Peter Deisenroth, Dieter Fox, and Carl Edward Rasmussen, (2015). Gaussian processes for data-efficient learning in robotics and control. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(2):408–423
This is also called a state-space model. How can you define x_t for our dynamics above so that it can be represented in that form (1)?

2.2.2 System Identification of a Simple Car

Consider the dynamics model of a first order car with states $q = (x, y, \theta)^\top$ (position and orientation), actions/controls $u = (s, \phi)^\top$ (speed and steering wheel angle), and known dynamics

$$\dot{q} = f(q, u) = \begin{pmatrix} s\cos\theta \\ s\sin\theta \\ \frac{s}{L}\tan\phi \end{pmatrix}. \tag{18}$$



Here, L is the distance between the wheels and not known.

(i) Assume you have an example trajectory $D = \{(x_t, y_t, \theta_t, s_t, \phi_t)\}_{t=1}^n$, where individual datapoints were sampled at 10Hz. Formulate an optimization problem that computes the "best" L for the given data.

(ii) Find a closed-form solution for your optimization problem in a).

2.2.3 Mountain Car Dynamics Learning

This is a coding exercise. Please bring your laptop and connect to the HDMI in the tutorial to show your results. (If you upload a pdf, just include a screenshot of results in the pdf.)

Install the mountain car simulation of *gymnasium* (`https://gymnasium.farama.org/`) using

```
pip install gymnasium[classic-control]
```

The following code simulates a few steps and collects data for a dynamics learning problem:

```
import gymnasium as gym
import numpy as np
env = gym.make('MountainCarContinuous-v0', render_mode='human')

# for this problem observation=state

u_dim = env.action_space.shape[0]
x_dim = env.observation_space.shape[0]
data_input = np.zeros((0,x_dim+u_dim))
data_target = np.zeros((0,x_dim))
n_data = 200

x_state, info = env.reset()

for t in range(n_data):
    # u_controls = env.action_space.sample()  # agent policy that uses the observation and
    u_controls = np.sin([.01*t])
    x_prev = x_state
    x_state, reward, terminated, truncated, info = env.step(u_controls)
    # terminated = a terminal state (often goal state, sometimes kill state, typically wit
    #    formally: the infinite MPD transitions to a deadlock nirvana state with eternal z
    # truncated = the simulation is 'artificially' truncated by some time limited - that's

    data_input = np.vstack([data_input, np.concatenate([x_prev, u_controls])])
    data_target = np.vstack([data_target, x_state])

    if terminated or truncated:
        if truncated:
            print('-- truncated -- should not happen!')
        else:
            print('-- terminated -- goal state reached')
        x_state, info = env.reset()

env.close()

print('input data:', data_input.shape)
print('output data:', data_target.shape)
```

(i) Increase the amount of data you collect (e.g. to $n = 1000$) and learn a regression from the input to output. Use whatever ML techniques you learned about in previous courses. Also linear regression is an option, which should work particularly well if you happen to include $\cos(3x_0)$ as a feature (where x_0 is the first entry of x: the position; see the domain documentation).

(ii) The above might not work well (in the sense of generalizing to the full state space), because the controller generating the data (`u_controls = np.sin([.01*t])`) is not very explorative. Play around with alternatives that generate much better data for learning.

(iii) Assume that you could only observe the position x_0 of the car, not the velocity x_1. As the state is not fully observable, you'll need to learn an autoregression model with longer input window. Modify the code above so that the data only contains positions and controls as input, and predicts the next position.

(iv) [Added for the tutorial session, to show you an easy way of how to make use of a learned model.] First, since we know this is a physical system with observed position q and velocity $\dot q$, let's also treat is like that: The *forward* dynamics is a mapping $q, \dot q, u \mapsto \ddot q$, while the *inverse* dynamics is the mapping $q, \dot{q}, \ddot{q} \mapsto u$. Learn the inverse dynamics function (define $\ddot{q}$ as the change in velocity by a simulation step). Then use the inverse dynamics to impose a PD behavior

$$\ddot{q}^* = k_p(q^* - q) - k_d \dot{q}$$

with $q^* = 2$ and $k_p = m/\tau^2$, $k_d = 2m\xi/\tau$ (exactly as in last exercise solution), and $\tau = 50$, $\xi = 0.9$.

2.3 Weekly Exercise 3

2.3.1 Literature: DAgger

The following paper introduces DAgger (short for "Dataset Aggregation"):

Stephane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell, (2011). A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning

(i) First have a look at Section 5 (Experiments), and if you like, the youtube video `https://www.youtube.com/watch?v=V00npNnWzSU`. Two basic questions about what is mentioned in 5.1:

   – The method uses a regression technique to train the policy $\pi : y \mapsto u$ (y are observations). Which technique is used?

   – Fig. 2 mentions β_i, which is a parameter of the method that changes with iteration i. How exactly is it chosen?

(ii) Now look at the pseudo code Alg. 3.1 on page 4. The introduction of Sec. 3 explains the pseudo code. The lines 4 and 5 ("Let π_i...", and "Sample T-step...") are perhaps the hardest to really understand. Your exercise: Write explicit pseudo code of how you generate such a "T-step trajectory using π_i", where this pseudo code can only call the dynamics function $x_t = f(x_{t\text{-}1}, u_{t\text{-}1})$, the expert policy $u_t = \pi^*(x_t)$, the trained policy $u_t = \hat{\pi}_i(x_t)$, and a state initialization method $x_0 \sim p(x_0)$.

   Note: Line 4 defines π_i to be a probabilistic mixing of policies π^* and $\hat{\pi}_i$, with coefficients β_i and $1 - \beta_i$ respectively. This notation is typically used when π are stochastic policies, but "implicitly clear" also when they are deterministic.

2.3.2 Trajectory Distributions, GMMs, ProMPs

Imitation learning can also be formulated as learning the distribution of demonstrated trajectories (rather than directly the policy), and thereafter use control theory to derive controllers that imitate this distribution. The following paper is a typical representative for using Gaussian Mixture Models (GMMs) to learn the distribution of demonstrated trajectories:

Sylvain Calinon and Aude Billard, (2007). Incremental learning of gestures by imitation in a humanoid robot. In *Proceedings of the ACM/IEEE International Conference on Human-robot Interaction*, pages 255–262

Only have a look at Figures 3 and 6 – they should clarify what it means to use Gaussians to "cover" the distribution of demonstrated trajectories, and thereby learn the distribution. To enable this, a trajectory $x_t \in \mathbb{R}^n$ for $t = 1, .., T$ is embedded in $n + 1$-dimensional space (t, x_t), and then standard density estimation using GMMs applied.

Consider a dataset $D = \{x_t^i\}_{t=1,..,T}^{i=1,2}$ with two 1-dimensional trajectories of length T, namely these two:

– First demonstrated trajectory $x_t^1 = \cos(t/3)$ for $t = 1, .., 20$

– Second demonstrated trajectory $x_t^2 = \cos(t/3 - 1)$ for $t = 1, .., 20$

(i) Plot both of these demonstrations

(ii) Assume you would fit a Gaussian Mixture Model with 2 components (2 Gaussians) to this data (using a time-embedding as above), how might it look like? (Sketch on paper. Where might be their centers and the ellipse illustrating their covariance matrices?) Conditioning this distribution on a particular t, e.g. $t = 11$, what would be the conditional variance over x? (Just argue in terms of your sketching.)

(iii) Consider a fully different approach: Treat each x^i as a vector with 20 entries x^i_t. The two vectors x^1 and x^2 form our tiny data set $D = \{x^i\}_{i=1,2}$. From this data we can estimate the element-wise mean μ_t and standard deviation σ_t for each t. Sketch these analogous to the above.

[Note: The latter approach is called ProMP (Paraschos et al, NeurIPS'13).]

2.3.3 Mountain Car Imitation Learning

This is a coding exercise. Please bring your laptop and connect to the HDMI in the tutorial to show your results. (If you upload a pdf, just include a screenshot of results in the pdf.)

We use the same mountain car example as in Exercise 2, so look for more detailed instructions there, if you haven't set it up, yet.

The following "policy" was written by an expert to solve the control problem:

```
def expert(t):
    if t < 50:
        return np.array([-1.0])
    elif t < 100:
        return np.array([1.0])
    return np.array([0.0])
```

Note that this uses the time step t and not the state as input, which is why we put "policy" in quotes.

(i) Collect a sufficient amount of data and learn a real policy, i.e., a function that maps from the current state to the action. Report your achieved loss.

You may still use any ML technique, including linear regression. However, this might also be a good starting point to use pyTorch, so that you have some experience with more complicated exercises later. You can follow the official tutorial at https://pytorch.org/tutorials/beginner/basics/quickstart_tutorial.html.

Hints: You can convert data using `torch.from_numpy(data_input).float()`. A useful function is
`torch.utils.data.random_split`. From the tutorial, make sure you adjust the neural network and loss function to match our target domain.

(ii) Validate your learned policy in the gym environment. What happens if you start from a starting state that was not part of your training data (e.g., use `env.reset(options='low': 0.1, 'high': 0.4)`)?

(iii) Can DAgger help here to collect a better dataset? Explain why or why not.

2.4 Weekly Exercise 4

2.4.1 Trajectory Distribution → Control

In the context of imitation learning, assume that from demonstration data you learned a trajectory distribution as well as an inverse dynamics model and now want to use these to "execute what you learned" on a robot. This question is about how to derive a control policy from a trajectory distribution and an inverse dynamics model.

More specifically, assume you learned a trajectory distribution

$$p(x_t) = \mathcal{N}(x_t; \mu_t, \Sigma_t) \ , \quad t = 1, .., T \ , \tag{19}$$

where for each time step t you have a different mean μ_t (characterizing the mean trajectory) and covariance matrix Σ_t, as well as an inverse dynamics model

$$u = \hat{f}(x_{t\text{-}1}, x_t) \ . \tag{20}$$

(Both models, p and f were trained from the data using ML, but we neglect annotating parameters.) We assume $x_t \in \mathbb{R}^n$ and fully observable, and $u \in \mathbb{R}^d$.

During execution, assume we are at time step t and current state x_t:

(i) Formulate an optimization problem to find a reference trajectory $x^*_{t+1:T}$ for the future execution. (You want that the reference "starts" (connects with) the current state x_t, but also that it is as consistent as possible with the learned trajectory distribution p.

Think about the role of the covariance matrices Σ_t and the role of the inverse kinematics in this formulation.

(This would be called a model-predictive control (MPC) approach: One would solve this optimization problem in every control cycle and use inverse kinematics to decide on controls. Depending on how you formulated the problem, it could be solved very efficiently using Riccati methods.)

(ii) Now assume that the trajectory distribution you learned is actually bi-modal, namely

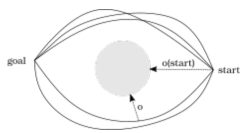
$$p(x_t) = w_t \mathcal{N}(x_t; \mu^1_t, \Sigma^1_t) + (1 - w_t)\mathcal{N}(x_t; \mu^2_t, \Sigma^2_t) \ , \tag{21}$$

where superscripts index the mode. How could you now formulate the optimization problem?

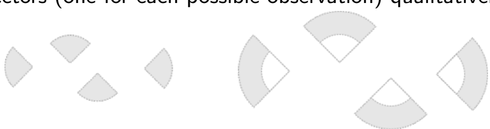
(iii) Assume you are scared away from using MPC and optimization in each control cycle. Could you also define a PD law to follow the (multi-modal) trajectory distribution? How? What would be issues?

2.4.2 Multi-Modal Distributions

Consider a circular single integrator robot with 2D single integrator dynamics ($q = (x, y)$, $u = (v_x, v_y)$, $\dot{q} = f(q, u) = u = (v_x, v_y)$). The robot is equipped with a LIDAR and processes the resulting point cloud to get observation $o = (d_x, d_y)$, i.e., a vector pointing to the closest boundary point of any obstacle, see the figure for some examples (dotted lines). From experts, we obtained five example trajectories for a given scenario with a single circular obstacle in the middle, see the figure for these trajectory (black lines). Our goal is to learn a policy that directly maps observations to controls ($\pi : o \mapsto u$).



(i) Discretize the observation into 8 parts (4 directional ranges and 2 magnitude ranges). For each of these possible input ranges, we "learn" the optimal action assuming an MSE loss. Draw the resulting action vectors (one for each possible observation) qualitatively.



(ii) Use the learned policy from a) and draw the resulting solution trajectory qualitatively.

(iii) Now consider that we learn a Gaussian Mixture Model (GMM) with two modes per discretized observation instead. Draw the resulting action distributions (one for each observation) qualitatively.

(iv) Explain how you can use the learned policy from c). Draw the resulting solution trajectory distribution qualitatively.

(v) What changes if we do not discretize the observation? Explain what possible policy function approximators you might use, what learning algorithms are applicable, and what the expected outcomes compared to b) and d) are.

2.4.3 Mountain Car Imitation Learning

This is a coding exercise. Please bring your laptop and connect to the HDMI in the tutorial to show your results. (If you upload a pdf, just include a screenshot of results in the pdf.)

We use the same mountain car example as in Exercise 2 and 3, so look for more detailed instructions there, if you have not set it up, yet.

In addition to the "policy" from last week, we now have a second expert that solves the problem as follows:

```
def expert2(t):
    if t < 50:
        return np.array([1.0])
    elif t < 100:
        return np.array([0.0]) # save some energy!
    elif t < 150:
        return np.array([1.0])
    return np.array([1.0])
```

Note that this expert decided to use a positive acceleration at the beginning, rather than the negative one that the previous expert used.

(i) Collect data from expert2 and verify that your approach from last week is able to imitate that expert.

(ii) Now mix your datasets, such that you have an equal amount of examples from expert1 (see Exercise 3) and expert2. Compare the loss and the success rate of solving the mountain car problem with this policy compared to just using data from a single expert.

(iii) Use diffusion to learn a stochastic policy using the dataset of b). Verify that your policy can solve the mountain car problem. Verify that you get a mixture of solutions mimicking both expert1 and expert2, for example by visualizing the histogram of generated control actions for the example state $x = (-0.5, 0.0)$.

Hint: We provide example code for training and sampling diffusion models for a simple noisy circular trajectory. The primary difference to your task above is that you now have to learn to sample from a *conditional* distribution $p(u_t|x_t)$. The simplest way to do so is to add the condition as an additional input to your neural network.

2.5 Weekly Exercise 5

2.5.1 Literature: SAC

The following paper introduces *Soft Actor-Critic*, a state-of-the art RL method that integrates many good ideas that have been discovered over the last decade into a rather clean algorithm:

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine, (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870

(i) First some bug hunting:

- In the Supplementary Material, Appendix A., Equation (14), there is a notational bug. Can you find it?
- In the main paper, going from Eq. (5) to (6), I think there is another bug. Can you find it?
- The line below (6) states "where the actions are sampled" – can you explain where actions are sampled?
- **Idea for another exercise:** In the paper the authors state that the gradient of the policy parameters could be estimated using the REINFORCE / likelihood ratio gradient estimator. The students could derive this one, or show that the reparametrization one has lower variance. This would link ex 1 and 2 nicely.

(ii) Now the core question: In Alg. 1 lower part you find three lines to train the parameters ψ, θ_i, ϕ, as well as a low-pass filter for $\bar\psi$.

- Find out which functions these parameters parameterize.
- Find out where these parameters are used during training, i.e., the inter-dependencies of training: For instance, when ϕ is trained, does that depend on ψ? Answer this for all parameters ψ, θ_i, ϕ.

2.5.2 The Reparametrization Trick

We typically write a conditional density as $p(x|y)$. If that depends on parameters (to be trained), we may write this as $p_\theta(x|y)$ or $p(x|y;\theta)$.

The reparametrization trick states that any (conditional) distribution $p(x|y;\theta)$ can instead be represented as a deterministic function $x = f(y,\epsilon;\theta),\ \epsilon \sim p(\epsilon)$.

(i) Given a Gaussian distribution $p_\theta(x) = \mathcal{N}(x|\mu,\Sigma)$ with parameters $\theta = (\mu,\Sigma),\ \mu \in \mathbb{R}^n,\ \Sigma \in \mathbb{R}^{n\times n}$, how can you rewrite this as deterministic $x = f_\theta(\epsilon)$ with $\epsilon \sim \mathcal{N}(0,\mathbf{I}_n), \epsilon \in \mathbb{R}^n$?

(ii) Given discrete (aka. categorical) distribution $p(x)$ over a discrete $x \in \{1,..,M\}$. How can you rerepresent sampling $x \sim p(x)$ as a deterministic function $x = f(\epsilon)$ with $\epsilon \sim \mathcal{U}[0,1]$ uniformly in the real inverval $[0,1]$?

[This is called a "trick" in a particular context: Sometimes there is a sampling step within an architecture, i.e., within a computation graph. E.g. $x \mapsto z \sim p_\theta(z|x),\ z \mapsto y = g_\theta(z)$, which is a VAC example, where the latent variable z is sampled in the "middle" of the architecture. Gradients in principle don't propagate *through* a sampling operation, and standard training would not be possible. But representing this as $x \mapsto z = f_\theta(x,\epsilon),\ z \mapsto y = g_\theta(z)$ with the sampling $\epsilon \sim p(\epsilon)$ done *outside the architecture*, gradients flow through f and g as usual, and the training process has to sample ϵ's as if it was data.]

2.5.3 Mountain Car RL using SAC

Use the SAC implementation in Stable Baselines3 to solve the Continuous Mountain Car problem: `https://stable-baselines3.readthedocs.io/en/master/modules/sac.html`.

(i) First, run SAC off-the-shelf, with default parameters using the example code provided on the above URL. In the tutorial, be able to demonstrate the final policy: Run multiple test rollouts, and compute the discounted total return (directly from the reward observations) for each test rollout.

(ii) Monitoring the training process is generally important in RL. Follow `https://stable-baselines3.readthedocs.io/en/master/guide/examples.html#callbacks-monitoring-training` to plot the training process (and generally learn about the Callback mechanism).

(iii) The SAC method has a ton of parameters. Try:

- Fixing `ent_coef` to one particular value (e.g. 10; or check the SAC paper for common choices), and report on the difference.

– The discounting factor `gamma`, e.g. to $\gamma = 0.999$.
– The network architecture (by default `net_arch = [256, 256]`). You'll have to look into code to understand the parameter, esp. the `get_actor_critic_arch` method in https://github.com/DLR-RM/stable-baselines3/blob/master/stable_baselines3/common/torch_layers.py. Try smaller networks.

2.6 Weekly Exercise 6

2.6.1 Literature: Privileged and Sensorimotor Policy Training

Here is a prominent application paper for RL:

Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter, (2020). Learning quadrupedal locomotion over challenging terrain. *Science Robotics*, 5(47):eabc5986

It uses standard RL in simulation to train a *privileged policy* (which they call "teacher policy") which has full access to the simulation's state information (e.g. exact terrain profile). In a second stage they train a *sensorimotor policy* (which they call "student policy") to imitate the privileged policy, but with sensorimotor (partially observable) input only. As the teacher policy can be queried anywhere, they can use DAgger for imitation, which simplifies imitation learning a lot.

[The general idea of training a sensor-based (=partial input) policy from a privileged (=full information) policy is old, previously called input remapping, or just surrogate model.]

Fig. 4 gives an overview of the approach. Here the questions:

(i) The input to the previleged policy is full information (exact robot & simulation state). But what is the definition of the output action $\bar{a}_t$? Looking for an answer you'll find words like "leg frequencies" and "foot position residuals" – what are these?

(ii) The Supplement S4 (pdf page 16) explains the reward function – a great example for reward engineering (in the positive sense, as this reflects the authors' understanding of "good locomotion"). Be able to explain each term and how they relate to higher level "commands".

(iii) Eq.(1) includes a second loss term, comparing $\bar{l}_t(o_t, x_t)$ with $l_t(H)$. Explain what $l_t(H)$ is and the idea of this term.

2.6.2 Episodes & Terminal States

Standard MDP theory assumes an infinite process $s_0, a_0, r_0, s_1, a_1, r_1, ...$ of states, actions and rewards. Accordingly, the return is defined as the infinite sum $\sum_{t=0}^\infty \gamma^t r_t$.

However, practical problems in the literature often involve "terminal states", and one speaks of "episodes". The following exercise clarifies how "terminal states" and "episodes" are meant in an infinite MDP.

(i) We define a terminal state as follows: Assume that in step T the agent reaches a terminal state s_T. The agent can then make a very last action a_T, and gets a final reward $r_T = R(s_T, a_T)$, but after this "there are no more states, actions, or rewards", and the total return of the agent is $\sum_{t=0}^T \gamma^t r_t$.
At first sight this is inconsistent to how MDPs are defined, because by definition they do not terminate. How can you construct a formal MDP to model such terminal states? (Tip: Extend the state space.)

(ii) Consider an MDP where the goal state is a *tunnel state*, which means that every choice of action in the goal state leads to receiving the goal reward and transitioning to a (maybe random) initial state $s \sim P(s_0)$.
Is the optimal policy for the MDP with tunnel goal state the same as the optimal policy for an MDP where the goal state is a terminal state? Provide arguments (ideally a rough proof or counterexample) for your answer.

(iii) In practice one never runs (or simulates) a process infinitely long. Instead, one typically aborts/truncates at some finite horizon T. One such truncated run is called *episode*. One then typically repeats many episodes (to collect data for learning or estimation of values/performance). When an episode was *truncated*, discuss how one could actually estimate the expected return of the policy?

2.6.3 Lunar Lander Domain Randomization

This is a coding exercise. Please bring your laptop and connect to the HDMI in the tutorial to show your results. (If you upload a pdf, just include a screenshot of results in the pdf.)

Install the lunar lander simulation of *gymnasium* (https://gymnasium.farama.org/) using

```
pip install "gymnasium[box2d]"
```

Similar to before, one can create an instance of the lunar lander (with varying wind enabled) using

```
env = gym.make('LunarLanderContinuous-v2', enable_wind=True)
```

(i) Train a policy – you should be able to reach rewards of > 200. To avoid finding new hyperparameters, use TD3 rather than SAC for training, where the default settings (with MlpPolicy and action noise) should work well.
Hint: The action noise can be defined as follows:

```
from stable_baselines3.common.noise import NormalActionNoise
action_noise = NormalActionNoise(mean=np.zeros(n_actions), sigma=0.1 * np.ones(n_act
```

(ii) Validate your policy in environments with different wind magnitudes and gravities. You can adjust these settings when making a gym environment, e.g.,

```
env = gym.make('LunarLanderContinuous-v2', enable_wind=True, gravity=-10, wind_power
```

For gravity, use values between -11 and -1; for the wind magnitude use values between 0 and 20. In which settings does your policy work well and in which does it not?

(iii) Train a policy with domain randomization on both gravity and wind_power. How does this policy compare to the other policy when validating in different settings as in b)?
Hint: You can use the callback mechanism of the policy (for `_on_rollout_end`) to add the randomization at the end of each episode. To do this, you can directly modify the parameters of the environment, e.g., set `env.gravity = np.random.uniform(min_value, max_value)`.

2.7 Weekly Exercise 7

2.7.1 Literature: Adversarial Inverse Reinforcement Learning

Here is an advanced paper on inverse RL applied to robotics problems:

Justin Fu, Katie Luo, and Sergey Levine, (2018). Learning robust rewards with adversarial inverse reinforcement learning

The paper was a big step forward in enabling Deep Learning methods for Inverse RL, namely by formulating a loss function similar to Generative Adversarial Networks (GANs) – actually following the original idea formulating InvRL as a discriminative (max margin) problem [80]. A followup paper [119] provides a nicer summary of the history of InvRL ideas and proposes improvement on Adversarial InvRL, but without robotics applications.

The paper webpage `https://sites.google.com/view/adversarial-irl` provides some videos. Here the questions:

(i) Let's start with the experiments in Section 7.2: The setting of the evaluation is *transfer learning*. Be able to explain Table 1: What are the two domains and what kind of transfer is tested? What does "TRPO, ground truth" mean (TRPO is a standard RL method)?

(ii) In Section 7.3, the setting of evaluation is *imitation learning*. How is that different to the setting of 7.2? How does AIRL compare with GAIL (a pure imitation learning method) and the TRPO expert?

(iii) The last equation in Sec. 4 (page 4) defines the discriminator $D_\theta(s,a)$. In GANs, a discriminator outputs the probability of whether the input data point is from the "original source" instead of from the learned generative model. What exactly is the meaning of the output of the $D_\theta(s,a)$ defined here?

[Note that, as in GANs, Alg. 1 describes an algorithm that also improves the "generative model" (here the learned policy π) whenever the discriminative model was improved.]

(iv) At first it might be unclear why learning $D_\theta(s,a)$ is related to extracting an underlying reward function. The last equation in Sec 6 (page 6) is quite crucial to understand this – explain roughly why the two neural nets $g_\theta(s)$ and $h_\phi(s)$ in Eq.(4) end up estimating reward and value functions.

2.7.2 Inverse RL on a Toy Control Problem

Consider a trivial control domain, with state $x \in \mathbb{R}$, controls $u \in [-1,1]$, and deterministic state transitions $x_{t+1} = x_t + u_t$.

The expert policy π^* is deterministic and chooses $\pi(x) = \text{clip}(-x)$, where $\text{clip}(x) = \max\{-1, \min\{+1, x\}\}$ (a typical notation for clipping you should get used to).

(i) What is a reward function $R(x)$ (depending on state only), such that the expert policy π^* is optimal? Derive the Q-function $Q^{\pi^*}(x,u)$ for your reward function $R(x)$ and prove that π^* is optimal. Assume a given discounting $\gamma \in [0,1)$. Is π^* the only optimal policy for your $R(x)$, or do equally optimal policies exist?

(ii) For a given γ, there exist many reward functions $R(x)$ such that π^* is optimal. (Rescaling R is trivial – neglect this.) Describe a space of alternative reward functions such that π^* is still optimal; e.g., find some (non-trivial) $F(x)$ such that for $R(x) \leftarrow R(x) + F(x)$, π^* is still optimal.

[Note, this sounds like a question about reward shaping (=changing R while guaranteeing invariance of the optimal policy) [79]. However, this question is slightly different, as we have a concrete deterministic dynamics and do not require invariance w.r.t. all possible world dynamics.]

(iii) Now, conversely, find a (minimal) variation $F(x)$ such that for $R(x) \leftarrow R(x) + F(x)$, π^* is not optimal anymore.

[This illustrates how a choice of reward function can discriminate between policies; as is implicit in adversarial InvRL.]

2.7.3 Practical Exercise: Exploration in RL

In this exercise, we will revisit the Continuous Mountain Car problem from `gym`. Previously, running SAC with default parameters from StableBaselines3 did not perform well. This week, we will explore whether exploration can make things work better.

One way to explore in RL is by adding noise to the actions taken. The paper *Pink Noise Is All You Need: Colored Noise Exploration In Deep Reinforcement Learning* (https://openreview.net/pdf?id=hQ9V5QN27eS) compares three types of noise:

- Gaussian (white) noise
- Ornstein-Uhlenbeck (OU) noise
- Pink noise

Our goal is to compare the effects of these noises on agent actions during training.

(i) Review the `ActionNoise` wrapper from StableBaselines3 (https://stable-baselines3.readthedocs.io/en/master/_modules/stable_baselines3/common/noise.html#ActionNoise), and the Pink Noise paper. Implement a child class `MyPinkNoise(ActionNoise)` that returns pink noise when called. Skeleton code is provided; you need to implement the call and reset methods.

(ii) StableBaselines3 includes implementations of Gaussian and OU noise (https://stable-baselines3.readthedocs.io/en/master/common/noise.html). Using your pink noise implementation, plot the different noise traces by plotting the 1D action on the y-axis and the time step on the x-axis with `scale=0.3` for all noises.

What do you observe?

(iii) Use all three noise types to train SAC on MountainCarContinuous with default parameters. Using `scale=0.3`, train for `total_timesteps=2e4`.

What do you observe? Plot the learning curves of all training runs.

HINT: It is not expected that all noises will lead to successful training. You do not need to adjust any SAC parameters.

2.8 Weekly Exercise 8

2.8.1 Literature: Neural Lander

Here is a paper that claims to combine safety and learning:

Guanya Shi, Xichen Shi, Michael O'Connell, Rose Yu, Kamyar Azizzadenesheli, Animashree Anandkumar, Yisong Yue, and Soon-Jo Chung, (2019). Neural Lander: Stable Drone Landing Control Using Learned Dynamics. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 9784–9790

The paper is at the intersection of control theory and learning and several other works exist to extend the idea to new domains.

Questions:

(i) Take a look at the proposed control law (8) and (12). What exactly is learned and how is the learned function applied in the controller?

(ii) The paper shows exponential stability, i.e., that the position error will go to zero quickly (around (14)). Explain in words the variables ϵ_m, L_a, and ρ. Explain how this equation tells us that the learned function needs to be Lipschitz-bounded.

(iii) Write down pseudo code on how one can use SGD or Adam and train a basic feed forward neural network with ReLU activation to have a bounded Lipschitz constant. (Use the information in the paper from III.B.)

(iv) What needs to change if $\tanh$ activation functions are used to achieve the same Lipschitz-bound?

2.8.2 Fun With Definitions

In the safe learning survey paper and the lecture, the robot dynamics were defined as $x_{k+1} = f_k(x_k, u_k, w_k)$. In RL and MDPs a transition model is used instead as $p(x_{k+1}|x_k, u_k)$. Here we look at the relationship of the two.

(i) Consider an MDP with states s, t, g and actions a, b. The transition model is $p(t|s, a) = 0.1, p(g|s, a) = 0.9, p(g|s, b) = 0.2, p(s|s, b) = 0.8, p(t|t, a) = 1, p(t|t, b) = 1, p(g|g, a) = 1, p(g|g, b) = 1$. The goal for the robot starting at s is to avoid t and reach g. What is a safe sequence of actions here? Write down an equivalent formulation using the notation in the paper/lecture.

(ii) Consider 1D single-integrator dynamics (i.e., state is position and the velocity can be controlled directly) and $\mathcal{W}$ zero-mean Gaussian: $x_{k+1} = x_k + u_k \cdot \Delta t + w_k$, where $w_k \sim N(0, \sigma^2)$. Write down an equivalent transition model.

(iii) The use of f_k allows hybrid models, where the dynamics might change over time. How can such changes be encoded in the MDP transition model?

(iv) We defined the cost as $J(x_{0:N}, u_{0:N-1}) = l_N(x_N) + \sum_{k=0}^{N-1} l_k(x_k, u_k)$. How can a discount factor be encoded here?

2.8.3 Working With Code: safe-control-gym

One implementation / benchmark for this is safe-control-gym, see

Zhaocong Yuan, Adam W. Hall, Siqi Zhou, Lukas Brunke, Melissa Greeff, Jacopo Panerati, and Angela P. Schoellig, (2022). Safe-Control-Gym: A Unified Benchmark Suite for Safe Learning-Based Control and Reinforcement Learning in Robotics. *IEEE Robotics and Automation Letters*, 7(4):11142–11149

for the paper and `https://github.com/utiasDSL/safe-control-gym` for the code on github.

You may install it locally following the instructions to try it, although some questions can also be answered just by reading the code.

```
git clone https://github.com/utiasDSL/safe-control-gym.git
cd safe-control-gym
pip install -e .
```

(i) Group the available algorithms (see the Readme file in the repo) using the taxonomy/grouping from the lecture (you may ignore the ones that have nothing to do with safety). Try to find academic references for each algorithm.

(ii) One interesting aspect of the toolbox is that it provides analytical models for the dynamics and constraints. Where are these models located for the three default systems (cartpole, quadrotor2d, quadrotor3d)?

(iii) Consider the example for a safety filter in examples/mpsc for a 2D quadrotor. How can you constrain the states and actions of the filter? Constrain the x coordinate to be within -1 and 2 and show the resulting plot(s), compared to the default setting (your choice of "unsafe" controller).

(iv) Consider the example for safe RL (examples/rl). For safe_explorer_ppo there is a pre-training and a regular training. What exactly is the difference between those two? How can you specify what safety means for your application?

2.9 Weekly Exercise 9

2.9.1 Literature: Grasp Data Collection

Here is a core paper on grasp data collection:

Hao-Shu Fang, Chenxi Wang, Minghao Gou, and Cewu Lu, (2020). Graspnet-1billion: A large-scale benchmark for general object grasping. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11444–11453

The collection of labelled grasp data is a central issue in learning-based grasing. Once such data is available, we can use strong supervised ML or diffusion methods to learn disciminative or generative models of grasps. The above paper is a good example on how grasp data generation is typically "engineered", and uses a model-based (force closure) method to provide grasp labels. (An alternative is to use a generic physical simulator, e.g., [30] is a recent paper generating a grasp dataset using the PhysX simulator.)

The questions are only about Section 3.2 and 3.3:

(i) Sec. 3.2 describes how 97,280 RGB-D images were taken. How is the camera pose known for each image? What are ArUco markers? For how many scenes were images collected?

(ii) Concerning Sec. 3.3 (paragraph "6D Pose Annotation"), how exactly are all 6D object poses annotated?

(iii) Paragraph "Grasp Pose Annotation" is the core. Provide pseudo code to what is happening in the 2nd paragraph; make the looping over objects/points/anything explicit. (Section 5.2, 2nd paragraph provides the ranges of D, A, and V.) The last paragraph describes how these object grasps are transferred to the scenes. Summarize what information the eventual dataset comprises for one scene.

2.9.2 Force Closure

This is a great robotics book:

`https://hades.mech.northwestern.edu/images/2/25/MR-v2.pdf`

The Section "Grasping and Manipulation – Exercises" contains interesting force and form closure questions, around Fig. 12.29 and 12.30.

(i) Solve Ex. 12.8 (page 507 in the pdf). Note that a twist in 3D space is a 6-vector combining a translation and rotation vector; here in 2D it is a 3-vector with 2D translation and one rotation. Sec. 12.1.6 (page 475) explains how to draw a twist as "CoR" – see footnote[1]

(ii) Solve Ex. 12.17. (I'll provide explicit equations defining force closure in the lecture.) (Ex. 12.18 is also a great exercise.)

2.9.3 Practical Exercise: Explore the Graspnet data

This exercise doesn't need much coding – the aim is simply to familiarize youself with existing datasets and conventions for learning-based grasping.

(i) Follow `https://graspnetapi.readthedocs.io/en/latest/install.html` to download and unzip all the data (sorry – lots of files... If you develop a script to do all downloads, share it with all students.)

(ii) Follow `https://graspnetapi.readthedocs.io/en/latest/example_vis.html` to visualize the grasp data. Automatically loop through all available objects (calling `showObjGrasp`), and all available scenes (calling `showSceneGrasp`).

[1] A convenient way to represent a planar twist $V = (v_x, v_y, \omega)$ (with rotation velocity ω, and translational velocities v_x, v_y) is as a **center of rotation (CoR)** at $(-v_y/\omega, v_x/\omega)$. An additional marker '+' or '-' tells if we rotate positively or negatively around this center.

What is the difference between format='rect' versus '6d'? (And why may it take minutes for format='6d'?)

(iii) The '6D grasp' documentation https://graspnetapi.readthedocs.io/en/latest/grasp_format.html#d-grasp explains how the grasp pose (translation and orientation) is stored. For a given scene (e.g. id=0), write a loop to output the grasp-translation and grasp-rotation-matrix for all grasps.

(What I do not understand: The Rectangle Grasp description seems to only describe grasps in the image plane – how it the real 3D rotation represented? Or it is not?)

2.10 Weekly Exercise 10

2.10.1 Literature: Learning to Plan in TAMP

Here is an example paper for learning to plan:

Danny Driess, Jung-Su Ha, and Marc Toussaint, (2020). Deep Visual Reasoning: Learning to Predict Action Sequences for Task and Motion Planning from an Initial Scene Image

The paper trains an image-based action sequence prediction. A follow-up paper[2] does something similar with a much more ambitious Large Manguage Model, but the above paper more clearly defines the problem in relation to TAMP. To get an overview, you may first watch the video https://www.youtube.com/watch?v=i8yyEbbvoEk.

Here are the questions:

(i) Eq. (4) defines the action sequence prediction model π. Note that S is the scene, g the goal, and $a_{1:K} \in \mathcal{T}(g, S), F_S(a_{1:K}) = 1$ means "$a_{1:K}$ is feasible and leads to goal g".

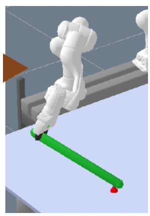
How does this π relate to modern sequence/language models, which also predict the next word/token given previous tokens? (What exactly is similar and different?)

How does this π relate to a trained state evaluation function as they are used, e.g., in modern chess/go engines? (Which score a board and therefore provide a heuristic for search. What exactly is similar and different?)

(ii) In Eq. (4), the actions a_k are input to the network. But they are encoded in a very particular way, as explained in subsection C (see also video at 0:20sec). How exactly are actions encoded?

(iii) As always, understanding the data generation is key. Section V.B (page 7) explains the data generation process, and Eq. (5) the definition of D_{data} (ingnore D_{train}). In this whole process, how often was the feasibility $F_S(a_{1:K})$ of an action sequence $a_{1:K}$ in a scene S being computed. (This computation happended fully model-based assuming full knowledge of the scene instantiated in the simulator.)

2.10.2 Optimal Sequential Manipulation in TAMP



Consider the scene on the right, where we have one robot with 7 degrees of freedom (dofs) $q \in \mathbb{R}^7$, and a stick with its pose $s \in SE(3)$ as degrees of freedom. (Ignore the triangle in the image.)

As discussed in the lecture, we consider the whole scene as a single multibody system with (q, s) as its configuration. Initially the stick is lying somewhere on the table (away from the red ball); the final goal is for the stick to touch the red ball.

[2]Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence, (2023). PaLM-E: An Embodied Multimodal Language Model

Assume that you have access to three constraint functions:

- $\phi_{\text{grasp}}(q, s) \in \mathbb{R}^3$ is a 3-dimensional constraint such that $\phi_{\text{grasp}}(q, s) = 0$ indicates a correct (stable) grasp of the stick by the robot.
- $\phi_{\text{touch}}(s) \in \mathbb{R}^1$ is a 1-dimensional constraint such that $\phi_{\text{touch}}(s) = 0$ indicates that the stick touches the red ball.
- $\phi_{\text{coll}}(q, s) \in \mathbb{R}^1$ is a 1-dimensional constraint such that $\phi_{\text{coll}}(q, s) \leq 0$ indicates that nothing in the scene collides.

(i) Formulate a mathematical program that represents the problem of optimally grasping the stick and then, with the grasped stick, optimally touching the red ball. The problem should only be about finding the grasp pose and the final pose – not yet the motions in between. As usual, optimality should reflect minimal motion effort by the robot. Assume the initial configuration is $(q_0, s_0) \in \mathbb{R}^7 \times \text{SE}(3)$.

(ii) It is quite natural to choose (q_1, s_1, q_2, s_2) as the decision variables of the above mathematical program. But can you think of an alternative, lower-dimensional parameterization of the problem?

(iii) Now modify the mathematical program above (of a) or b)) to include the full motion from the start configuration until the stick touches the ball. Use an optimality criterion as is standard in trajectory optimization.

(iv) Neglect the motion again; consider only grasp and touch. But this time consider a sequence of 4 actions: grasp-stick, place-stick, grasp-stick, touch-ball, where the 2nd action places the stick back on the table before picking it up again. Can you think of scene (stick and ball placement) where this action sequence makes sense? Instead of $(q_1, s_1, q_2, s_2, q_3, s_3, q_4, s_4)$, what would be a lower-dimensional parameterization?

(For discussion at the tutorial:) You know how path finding in a standard setting is defined as finding a collision free path.[3] How can the same sequential manipulation problem as in b) be represented as a path finding problem (respecting all constraints but neglecting optimality)?

2.11 Weekly Exercise 11

2.11.1 Literature: Neural-Swarm2

Here is the paper we discussed in the lecture that uses (and extends) deep sets for a control problem that arises in multi-robot aerial swarms [4]:

Guanya Shi, Wolfgang Honig, Xichen Shi, Yisong Yue, and Soon-Jo Chung, (2022). Neural-Swarm2: Planning and Control of Heterogeneous Multirotor Swarms Using Learned Interactions. *IEEE Transactions on Robotics*, 38(2):1063–1079

The paper is an extension of the NeuralLander paper to the multi-robot case we discussed in exercise 8.

Questions:

(i) How does the dataset exactly look like? How was the data obtained? What sensing/measurement capabilities were needed to obtain such data?

(ii) Write down pseudo code on how one can use SGD or Adam and train a 2-group permutation-invariant function using the heterogeneous deep sets proposed in (9).

[3] E.g., finding a continuous path $\gamma : [0, T] \to \mathcal{X}_{\text{free}}$ from a given start configuration $\gamma(0) = x_0$ to a final configuration $\gamma(T) \in \mathcal{X}_{\text{goal}}$ within the free configuration space $\mathcal{X}_{\text{free}} = \{x \in \mathcal{X} : \phi_{\text{coll}} \leq 0\}$.

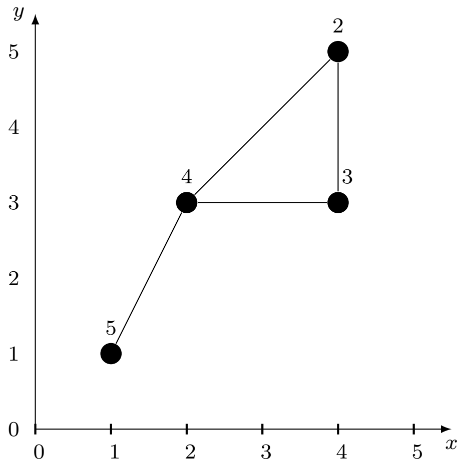
[4] A shorter and perhaps easier to follow earlier work is Guanya Shi, Wolfgang Honig, Yisong Yue, and Soon-Jo Chung, (2020). Neural-Swarm: Decentralized Close-Proximity Multirotor Control Using Learned Interactions. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3241–3247

(iii) Consider the use-case of motion planning (Fig. 6). Explain how the neural network is applied inside the motion planner.

(iv) In the considered examples for K-group permutation invariant functions, K is relatively small (4 in the paper). Consider the case where K is large or unknown, for example if we are able to measure the size of the neighboring robot (a real value). How could learning be applied in this case?

2.11.2 Encodings for Environmental Monitoring

Consider a team of robots that is spatially distributed as shown below. In the figure, circles are robots, the numbers are their associated measurements (such as temperatures), and lines indicate the existence of a communication link. The goal is to find the minimum of their sensor measurements. In this question we will explore various concrete encodings for such problem.



(i) First consider the abstract, centralized setting with function $f(\mathcal{X}) = \min_{x \in \mathcal{X}} x$, where $\mathcal{X}$ is a set of real numbers. In other words, the function takes one or more numbers as input and returns the smallest element of these numbers. Recall that the deep set

$$f(\mathcal{X}) \approx \rho\left(\sum_{x \in \mathcal{X}} \phi(x)\right) \tag{22}$$

should be able to approximate this function. Provide concrete (differentiable) functions for ρ and ϕ for this case.
Hint: You can find some inspiration in the original Deep Set paper or the paper from question 1.

(ii) Now assume the case where robots have a limited communication radius. One example is shown in the figure, where the lines show communication links. Define the `Aggregate` and `Update` functions of a simple message-passing neural network.
Demonstrate in the example above, how the node at $(1, 1)$ computes the minimum value.

(iii) How could a CNN be used for the case with limited communication radius? Be specific about the layers the CNN should have.

(iv) For the use-case outlined above, what are advantages and disadvantages of the three encodings (Deep Sets, GNN, CNN)? Consider both small (=few neighbors) and large (=many neighbors) cases.

References

[1] Pieter Abbeel, Adam Coates, and Andrew Y. Ng, (2010). Autonomous Helicopter Aerobatics through Apprenticeship Learning. *The International Journal of Robotics Research*, 29(13):1608–1639.

[2] Pieter Abbeel and Andrew Y. Ng, (2004). Apprenticeship learning via inverse reinforcement learning. In *Twenty-first international conference*, page 1.

[3] Riad Akrour, Marc Schoenauer, and Michèle Sebag, (2012). APRIL: Active preference learning-based reinforcement learning. In Peter A. Flach, Tijl De Bie, and Nello Cristianini, editors, *Machine Learning and Knowledge Discovery in Databases*, volume 7524, pages 116–131.

[4] Brenna D. Argall, Sonia Chernova, Manuela Veloso, and Brett Browning, (2009). A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483.

[5] Christopher G. Atkeson and Stefan Schaal, (1997). Robot learning from demonstration. In *ICML*, volume 97, pages 12–20.

[6] Leonard Bauersfeld, Elia Kaufmann, Philipp Foehn, Sihao Sun, and Davide Scaramuzza, (2021). NeuroBEM: Hybrid aerodynamic quadrotor model. In *Robotics: Science and Systems XVII*, volume 17.

[7] Felix Berkenkamp, Angela P. Schoellig, and Andreas Krause, (2016). Safe controller optimization for quadrotors with Gaussian processes. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 491–496.

[8] Christopher M. Bishop and Hugh Bishop, (2024). *Deep Learning: Foundations and Concepts*.

[9] Ronen I. Brafman and Moshe Tennenholtz, (2002). R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3:213–231.

[10] Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, and Ryan Julian, (2023). Do as i can, not as i say: Grounding language in robotic affordances. In *Conference on Robot Learning*, pages 287–318.

[11] Lukas Brunke, Melissa Greeff, Adam W. Hall, Zhaocong Yuan, Siqi Zhou, Jacopo Panerati, and Angela P. Schoellig, (2022). Safe Learning in Robotics: From Learning-Based Control to Safe Reinforcement Learning. *Annual Review of Control, Robotics, and Autonomous Systems*, 5:411–444.

[12] Michael Burri, Michael Bloesch, Zachary Taylor, Roland Siegwart, and Juan Nieto, (2018). A framework for maximum likelihood parameter identification applied on MAVs. *Journal of Field Robotics*, 35(1):5–22.

[13] Michael Burri, Janosch Nikolic, Helen Oleynikova, Markus W. Achtelik, and Roland Siegwart, (2016). Maximum likelihood parameter identification for MAVs. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4297–4303.

[14] Sylvain Calinon and Aude Billard, (2007). Incremental learning of gestures by imitation in a humanoid robot. In *Proceedings of the ACM/IEEE International Conference on Human-robot Interaction*, pages 255–262.

[15] Stanley H. Chan, (2024). Tutorial on Diffusion Models for Imaging and Vision.

[16] Dian Chen, Brady Zhou, Vladlen Koltun, and Philipp Krähenbühl, (2020). Learning by cheating. In *Conference on Robot Learning*, pages 66–75.

[17] S. Chen, S. A. Billings, and P. M. Grant, (1990). Non-linear system identification using neural networks. *International Journal of Control*, 51(6):1191–1214.

[18] Weizhe Chen, Sven Koenig, and Bistra Dilkina, (2024). Why Solving Multi-agent Path Finding with Large Language Model has not Succeeded Yet.

[19] Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song, (2023). Diffusion Policy: Visuomotor Policy Learning via Action Diffusion. In *Robotics: Science and Systems XIX*.

[20] Paul F. Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei, (2017). Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30.

[21] Marc Peter Deisenroth, Dieter Fox, and Carl Edward Rasmussen, (2015). Gaussian processes for data-efficient learning in robotics and control. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(2):408–423.

[22] Martin Do, Pedram Azad, Tamim Asfour, and Rudiger Dillmann, (2008). Imitation of human motion on a humanoid robot using non-linear optimization. In *Humanoids 2008-8th IEEE-RAS International Conference on Humanoid Robots*, pages 545–552.

[23] Andreas Doerr, Christian Daniel, Martin Schiegg, Nguyen-Tuong Duy, Stefan Schaal, Marc Toussaint, and Trimpe Sebastian, (2018). Probabilistic recurrent state-space models. In *International conference on machine learning*, pages 1280–1289.

[24] Danny Driess, Jung-Su Ha, and Marc Toussaint, (2020). Deep Visual Reasoning: Learning to Predict Action Sequences for Task and Motion Planning from an Initial Scene Image.

[25] Danny Driess, Zhiao Huang, Yunzhu Li, Russ Tedrake, and Marc Toussaint, (2023). Learning multi-object dynamics with compositional neural radiance fields. In *Conference on robot learning*, pages 1755–1768.

[26] Danny Driess, Zhiao Huang, Yunzhu Li, Russ Tedrake, and Marc Toussaint, (2023). Learning multi-object dynamics with compositional neural radiance fields. In *Conference on Robot Learning*, pages 1755–1768.

[27] Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence, (2023). PaLM-E: An Embodied Multimodal Language Model.

[28] Onno Eberhard, Jakob Hollenstein, Cristina Pinneri, and Georg Martius, (2022). Pink noise is all you need: Colored noise exploration in deep reinforcement learning. In *The Eleventh International Conference on Learning Representations*.

[29] Ben Eisner, Harry Zhang, and David Held, (2024). FlowBot3D: Learning 3D Articulation Flow to Manipulate Articulated Objects.

[30] Clemens Eppner, Arsalan Mousavian, and Dieter Fox, (2021). Acronym: A large-scale grasp dataset based on simulation. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6222–6227.

[31] Jonas Eschmann, Dario Albani, and Giuseppe Loianno, (2024). Data-driven system identification of quadrotors subject to motor delays.

[32] Michael Everett, Yu Fan Chen, and Jonathan P. How, (2018). Motion Planning Among Dynamic, Decision-Making Agents with Deep Reinforcement Learning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3052–3059.

[33] Tingxiang Fan, Pinxin Long, Wenxi Liu, and Jia Pan, (2020). Distributed multi-robot collision avoidance via deep reinforcement learning for navigation in complex scenarios. *The International Journal of Robotics Research*, 39(7):856–892.

[34] Hao-Shu Fang, Chenxi Wang, Hongjie Fang, Minghao Gou, Jirong Liu, Hengxu Yan, Wenhai Liu, Yichen Xie, and Cewu Lu, (2023). Anygrasp: Robust and efficient grasp perception in spatial and temporal domains. *IEEE Transactions on Robotics*.

[35] Hao-Shu Fang, Chenxi Wang, Minghao Gou, and Cewu Lu, (2020). Graspnet-1billion: A large-scale benchmark for general object grasping. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11444–11453.

[36] Chelsea Finn and Sergey Levine, (2017). Deep visual foresight for planning robot motion. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2786–2793.

[37] Chelsea Finn, Sergey Levine, and Pieter Abbeel, (2016). Guided cost learning: Deep inverse optimal control via policy optimization. In *International Conference on Machine Learning*, pages 49–58.

[38] Justin Fu, Katie Luo, and Sergey Levine, (2018). Learning robust rewards with adversarial inverse reinforcement learning.

[39] Scott Fujimoto and Shixiang Shane Gu, (2021). A minimalist approach to offline reinforcement learning. *Advances in neural information processing systems*, 34:20132–20145.

[40] Scott Fujimoto, Herke Hoof, and David Meger, (2018). Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596.

[41] Julian Förster, (2015). System identification of the crazyflie 2.0 nano quadrocopter.

[42] Fernando Gama, Qingbiao Li, Ekaterina Tolstaya, Amanda Prorok, and Alejandro Ribeiro, (2022). Synthesizing Decentralized Controllers With Graph Neural Networks and Imitation Learning. *IEEE Transactions on Signal Processing*, 70:1932–1946.

[43] Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez, (2021). Integrated Task and Motion Planning. *Annual Review of Control, Robotics, and Autonomous Systems*, 4(1):265–293.

[44] Claudio Gaz, Marco Cognetti, Alexander Oliva, Paolo Robuffo Giordano, and Alessandro De Luca, (2019). Dynamic identification of the franka emika panda robot with retrieval of feasible parameters using penalty-based optimization. *IEEE Robotics and Automation Letters*, 4(4):4147–4154.

[45] Matthieu Geist, Bilal Piot, and Olivier Pietquin, (2017). Is the Bellman residual a bad proxy? *Advances in Neural Information Processing Systems*, 30.

[46] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio, (2014). Generative adversarial nets. *Advances in neural information processing systems*, 27.

[47] Jung-Su Ha, Danny Driess, and Marc Toussaint, (2022). Deep visual constraints: Neural implicit models for manipulation planning from visual input. *IEEE Robotics and Automation Letters*, 7(4):10857–10864.

[48] Jung-Su Ha, Danny Driess, and Marc Toussaint, (2022). Deep visual constraints: Neural implicit models for manipulation planning from visual input. *IEEE Robotics and Automation Letters*, 7(4):10857–10864.

[49] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine, (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870.

[50] Dylan Hadfield-Menell, Stuart J. Russell, Pieter Abbeel, and Anca Dragan, (2016). Cooperative inverse reinforcement learning. *Advances in neural information processing systems*, 29.

[51] Matthew Hausknecht and Peter Stone, (2015). Deep recurrent q-learning for partially observable mdps. In *2015 Aaai Fall Symposium Series*.

[52] Tairan He, Chong Zhang, Wenli Xiao, Guanqi He, Changliu Liu, and Guanya Shi, (2024). Agile But Safe: Learning Collision-Free High-Speed Legged Locomotion.

[53] Donald Joseph Hejna III and Dorsa Sadigh, (2023). Few-shot preference learning for human-in-the-loop rl. In *Conference on Robot Learning*, pages 2014–2025.

[54] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver, (2018). Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

[55] Jonathan Ho and Stefano Ermon, (2016). Generative Adversarial Imitation Learning. In *Advances in Neural Information Processing Systems*, volume 29.

[56] Jonathan Ho, Ajay Jain, and Pieter Abbeel, (2020). Denoising Diffusion Probabilistic Models. In *Advances in Neural Information Processing Systems*, volume 33, pages 6840–6851.

[57] Zhehui Huang, Zhaojing Yang, Rahul Krupani, Baskın Şenbaşlar, Sumeet Batra, and Gaurav S. Sukhatme, (2024). Collision Avoidance and Navigation for a Quadrotor Swarm Using End-to-end Deep Reinforcement Learning.

[58] Brian Ichter, James Harrison, and Marco Pavone, (2018). Learning Sampling Distributions for Robot Motion Planning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7087–7094.

[59] Elia Kaufmann, Leonard Bauersfeld, Antonio Loquercio, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza, (2023). Champion-level drone racing using deep reinforcement learning. *Nature*, 620(7976):982–987.

[60] Elia Kaufmann, Antonio Loquercio, Rene Ranftl, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza, (2020). Deep Drone Acrobatics. In *Robotics: Science and Systems XVI*.

[61] Michael Kearns and Satinder Singh, (2002). Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49(2/3):209–232.

[62] Kuno Kim, Yihong Gu, Jiaming Song, Shengjia Zhao, and Stefano Ermon, (2020). Domain Adaptive Imitation Learning. In *Proceedings of the 37th International Conference on Machine Learning*, pages 5286–5295.

[63] Jens Kober and Jan Peters, (2009). Learning motor primitives for robotics. In *2009 IEEE International Conference on Robotics and Automation*, pages 2112–2118.

[64] Ryan Kortvelesy and Amanda Prorok, (2021). ModGNN: Expert Policy Approximation in Multi-Agent Systems with a Modular Graph Neural Network Architecture. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9161–9167.

[65] Aviral Kumar, Anikait Singh, Frederik Ebert, Mitsuhiko Nakamoto, Yanlai Yang, Chelsea Finn, and Sergey Levine, (2023). Pre-Training for Robots: Offline RL Enables Learning New Tasks from a Handful of Trials.

[66] Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter, (2020). Learning quadrupedal locomotion over challenging terrain. *Science Robotics*, 5(47):eabc5986.

[67] Sergey Levine and Vladlen Koltun, (2013). Guided policy search. In *International Conference on Machine Learning*, pages 1–9.

[68] Qingbiao Li, Fernando Gama, Alejandro Ribeiro, and Amanda Prorok, (2020). Graph Neural Networks for Decentralized Multi-Robot Path Planning. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 11785–11792.

[69] Qingbiao Li, Weizhe Lin, Zhe Liu, and Amanda Prorok, (2021). Message-Aware Graph Attention Networks for Large-Scale Multi-Robot Path Planning. *IEEE Robotics and Automation Letters*, 6(3):5533–5540.

[70] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra, (2019). Continuous control with deep reinforcement learning.

[71] Ryan Lowe, given-i=YI family=WU, given=YI, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch, (2017). Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. In *Advances in Neural Information Processing Systems*, volume 30.

[72] Kevin M. Lynch and Frank C. Park, (2017). *Modern Robotics*.

[73] Jeffrey Mahler, Jacky Liang, Sherdil Niyaz, Michael Laskey, Richard Doan, Xinyu Liu, Juan Aparicio Ojea, and Ken Goldberg, (2017). Dex-Net 2.0: Deep Learning to Plan Robust Grasps with Synthetic Point Clouds and Analytic Grasp Metrics.

[74] Odalric-Ambrym Maillard, Rémi Munos, Alessandro Lazaric, and Mohammad Ghavamzadeh, (2010). Finite-sample analysis of Bellman residual minimization. In *Proceedings of 2nd Asian Conference on Machine Learning*, pages 299–314.

[75] Lucas Manuelli, Wei Gao, Peter Florence, and Russ Tedrake, (2022). KPAM: KeyPoint Affordances for Category-Level Robotic Manipulation. In Tamim Asfour, Eiichi Yoshida, Jaeheung Park, Henrik Christensen, and Oussama Khatib, editors, *Robotics Research*, volume 20, pages 132–157.

[76] Matthew T. Mason, (2018). Toward Robotic Manipulation. *Annual Review of Control, Robotics, and Autonomous Systems*, 1(1):1–28.

[77] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, and Georg Ostrovski, (2015). Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533.

[78] Teodor Mihai Moldovan and Pieter Abbeel, (2012). Safe exploration in Markov decision processes. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ICML'12, pages 1451–1458.

[79] Andrew Y. Ng, Daishi Harada, and Stuart Russell, (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pages 278–287.

[80] Andrew Y. Ng and Stuart Russell, (2000). Algorithms for inverse reinforcement learning. In *Icml*, volume 1, page 2.

[81] James Orr and Ayan Dutta, (2023). Multi-Agent Deep Reinforcement Learning for Multi-Robot Applications: A Survey. *Sensors*, 23(7):3625.

[82] Alexandros Paraschos, Christian Daniel, Jan R. Peters, and Gerhard Neumann, (2013). Probabilistic movement primitives. *Advances in neural information processing systems*, 26.

[83] Karl Pertsch, Youngwoon Lee, Yue Wu, and Joseph J. Lim, (2021). Demonstration-Guided reinforcement learning with learned skills.

[84] Lerrel Pinto, James Davidson, Rahul Sukthankar, and Abhinav Gupta, (2017). Robust Adversarial Reinforcement Learning. In *Proceedings of the 34th International Conference on Machine Learning*, pages 2817–2826.

[85] Matthias Plappert, Rein Houthooft, Prafulla Dhariwal, Szymon Sidor, Richard Y. Chen, Xi Chen, Tamim Asfour, Pieter Abbeel, and Marcin Andrychowicz, (2018). Parameter Space Noise for Exploration.

[86] Dean A. Pomerleau, (1988). Alvinn: An autonomous land vehicle in a neural network. *Advances in neural information processing systems*, 1.

[87] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, and Jack Clark, (2021). Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763.

[88] Spencer M. Richards, Felix Berkenkamp, and Andreas Krause, (2018). The Lyapunov Neural Network: Adaptive Stability Certification for Safe Learning of Dynamical Systems.

[89] Benjamin Riviere, Wolfgang Honig, Matthew Anderson, and Soon-Jo Chung, (2021). Neural Tree Expansion for Multi-Robot Planning in Non-Cooperative Environments. *IEEE Robotics and Automation Letters*, 6(4):6868–6875.

[90] Benjamin Riviere, Wolfgang Honig, Yisong Yue, and Soon-Jo Chung, (2020). GLAS: Global-to-Local Safe Autonomy Synthesis for Multi-Robot Motion Planning With End-to-End Learning. *IEEE Robotics and Automation Letters*, 5(3):4249–4256.

[91] Stephane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell, (2011). A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning.

[92] Stuart Russell, (2019). *Human compatible: AI and the problem of control*.

[93] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever, (2017). Evolution Strategies as a Scalable Alternative to Reinforcement Learning.

[94] Guillaume Sartoretti, Justin Kerr, Yunfei Shi, Glenn Wagner, T. K. Satish Kumar, Sven Koenig, and Howie Choset, (2019). PRIMAL: Pathfinding via Reinforcement and Imitation Multi-Agent Learning. *IEEE Robotics and Automation Letters*, 4(3):2378–2385.

[95] Stefan Schaal, Christopher G. Atkeson, and Sethu Vijayakumar, (2002). Scalable techniques from nonparametric statistics for real time robot learning. *Applied Intelligence*, 17(1):49–60.

[96] Stefan Schaal, Auke Ijspeert, and Aude Billard, (2003). Computational approaches to motor learning by imitation. *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, 358(1431):537–547.

[97] Ingmar Schubert, Jingwei Zhang, Jake Bruce, Sarah Bechtle, Emilio Parisotto, Martin Riedmiller, Jost Tobias Springenberg, Arunkumar Byravan, Leonard Hasenclever, and Nicolas Heess, (2023). A generalist dynamics model for control.

[98] Guanya Shi, Wolfgang Honig, Xichen Shi, Yisong Yue, and Soon-Jo Chung, (2022). Neural-Swarm2: Planning and Control of Heterogeneous Multirotor Swarms Using Learned Interactions. *IEEE Transactions on Robotics*, 38(2):1063–1079.

[99] Guanya Shi, Wolfgang Honig, Yisong Yue, and Soon-Jo Chung, (2020). Neural-Swarm: Decentralized Close-Proximity Multirotor Control Using Learned Interactions. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3241–3247.

[100] Guanya Shi, Xichen Shi, Michael O'Connell, Rose Yu, Kamyar Azizzadenesheli, Animashree Anandkumar, Yisong Yue, and Soon-Jo Chung, (2019). Neural Lander: Stable Drone Landing Control Using Learned Dynamics. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 9784–9790.

[101] Mohit Shridhar, Lucas Manuelli, and Dieter Fox, (2022). Cliport: What and where pathways for robotic manipulation. In *Conference on Robot Learning*, pages 894–906.

[102] Hava T. Siegelmann, Bill G. Horne, and C. Lee Giles, (1997). Computational capabilities of recurrent NARX neural networks. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 27(2):208–215.

[103] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller, (2014). Deterministic policy gradient algorithms. In *International Conference on Machine Learning*, pages 387–395.

[104] Anthony Simeonov, Yilun Du, Andrea Tagliasacchi, Joshua B. Tenenbaum, Alberto Rodriguez, Pulkit Agrawal, and Vincent Sitzmann, (2022). Neural descriptor fields: Se (3)-equivariant object representations for manipulation. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 6394–6400.

[105] Samarth Sinha, Ajay Mandlekar, and Animesh Garg, (2022). S4rl: Surprisingly simple self-supervision for offline reinforcement learning in robotics. In *Conference on Robot Learning*, pages 907–917.

[106] Laura Smith, Nikita Dhawan, Marvin Zhang, Pieter Abbeel, and Sergey Levine, (2020). AVID: Learning Multi-Stage Tasks via Pixel-Level Translation of Human Videos.

[107] Shuran Song, Andy Zeng, Johnny Lee, and Thomas Funkhouser, (2020). Grasping in the wild: Learning 6dof closed-loop grasping from low-cost demonstrations. *IEEE Robotics and Automation Letters*, 5(3):4978–4985.

[108] Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O. Stanley, and Jeff Clune, (2018). Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning.

[109] Martin Sundermeyer, Arsalan Mousavian, Rudolph Triebel, and Dieter Fox, (2021). Contact-graspnet: Efficient 6-dof grasp generation in cluttered scenes. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 13438–13444.

[110] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, and Thore Graepel, (2018). Value-Decomposition Networks For Cooperative Multi-Agent Learning Based On Team Reward.

[111] Russ Tedrake, (2023). Robotic Manipulation - Lecture Website.

[112] Stefanie Tellex, Nakul Gopalan, Hadas Kress-Gazit, and Cynthia Matuszek, (2020). Robots That Use Language. *Annual Review of Control, Robotics, and Autonomous Systems*, 3(1):25–55.

[113] Andreas Ten Pas, Marcus Gualtieri, Kate Saenko, and Robert Platt, (2017). Grasp Pose Detection in Point Clouds. *The International Journal of Robotics Research*, 36(13-14):1455–1473.

[114] Brijen Thananjeyan, Ashwin Balakrishna, Suraj Nair, Michael Luo, Krishnan Srinivasan, Minho Hwang, Joseph E. Gonzalez, Julian Ibarz, Chelsea Finn, and Ken Goldberg, (2021). Recovery RL: Safe Reinforcement Learning With Learned Recovery Zones. *IEEE Robotics and Automation Letters*, 6(3):4915–4922.

[115] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel, (2017). Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30.

[116] Ekaterina Tolstaya, Fernando Gama, James Paulos, George Pappas, Vijay Kumar, and Alejandro Ribeiro, (2020). Learning Decentralized Controllers for Robot Swarms with Graph Neural Networks. In *Proceedings of the Conference on Robot Learning*, pages 671–682.

[117] Marc Toussaint, (2015). Logic-Geometric Programming: An Optimization-Based Approach to Combined Task and Motion Planning. In *IJCAI*, pages 1930–1936.

[118] Marc A. Toussaint, Kelsey Rebecca Allen, Kevin A. Smith, and Joshua B. Tenenbaum, (2018). Differentiable physics and stable modes for tool-use and manipulation planning.

[119] Aaron Tucker, Adam Gleave, and Stuart Russell, (2018). Inverse reinforcement learning for video games.

[120] Julen Urain, Niklas Funk, Jan Peters, and Georgia Chalvatzaki, (2023). Se (3)-diffusionfields: Learning smooth cost functions for joint grasp and motion optimization through diffusion. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5923–5930.

[121] Kim P. Wabersich, Andrew J. Taylor, Jason J. Choi, Koushil Sreenath, Claire J. Tomlin, Aaron D. Ames, and Melanie N. Zeilinger, (2023). Data-Driven Safety Filters: Hamilton-Jacobi Reachability, Control Barrier Functions, and Predictive Methods for Uncertain Systems. *IEEE Control Systems*, 43(5):137–177.

[122] Yutong Wang, Mehul Damani, Pamela Wang, Yuhong Cao, and Guillaume Sartoretti, (2022). Distributed Reinforcement Learning for Robot Teams: A Review. *Current Robotics Reports*, 3(4):239–257.

[123] Lilian Weng, (2017-08-20T00:00:00+00:00). From GAN to WGAN.

[124] Lilian Weng, (2018-08-12T00:00:00+00:00). From Autoencoder to Beta-VAE.

[125] Lilian Weng, (2021-07-11T00:00:00+00:00). What are Diffusion Models?

[126] Peter R. Wurman, Samuel Barrett, Kenta Kawamoto, James MacGlashan, Kaushik Subramanian, Thomas J. Walsh, Roberto Capobianco, Alisa Devlic, Franziska Eckert, Florian Fuchs, Leilani Gilpin, Piyush Khandelwal, Varun Kompella, HaoChih Lin, Patrick MacAlpine, Declan Oller, Takuma Seno, Craig Sherstan, Michael D. Thomure, Houmehr Aghabozorgi, Leon Barrett, Rory Douglas, Dion Whitehead, Peter Dürr, Peter Stone, Michael Spranger, and Hiroaki Kitano, (2022). Outracing champion Gran Turismo drivers with deep reinforcement learning. *Nature*, 602(7896):223–228.

[127] Zhenjia Xu, Zhanpeng He, and Shuran Song, (2022). Universal manipulation policy network for articulated objects. *IEEE robotics and automation letters*, 7(2):2447–2454.

[128] Chenning Yu, Hongzhan Yu, and Sicun Gao, (2022). Learning Control Admissibility Models with Graph Neural Networks for Multi-Agent Navigation. In *6th Annual Conference on Robot Learning*.

[129] Javier Yu, Joseph A. Vincent, and Mac Schwager, (2022). DiNNO: Distributed Neural Network Optimization for Multi-Robot Collaborative Learning. *IEEE Robotics and Automation Letters*, 7(2):1896–1903.

[130] Zhaocong Yuan, Adam W. Hall, Siqi Zhou, Lukas Brunke, Melissa Greeff, Jacopo Panerati, and Angela P. Schoellig, (2022). Safe-Control-Gym: A Unified Benchmark Suite for Safe Learning-Based Control and Reinforcement Learning in Robotics. *IEEE Robotics and Automation Letters*, 7(4):11142–11149.

[131] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola, (2017). Deep Sets. In *Advances in Neural Information Processing Systems*, volume 30.

[132] Songyuan Zhang, Kunal Garg, and Chuchu Fan, (2023). Neural Graph Control Barrier Functions Guided Distributed Collision-avoidance Multi-agent Control. In *7th Annual Conference on Robot Learning*.

[133] Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn, (2023). Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware.

[134] Yang Zhou, Jiuhong Xiao, Yue Zhou, and Giuseppe Loianno, (2022). Multi-Robot Collaborative Perception With Graph Neural Networks. *IEEE Robotics and Automation Letters*, 7(2):2289–2296.

[135] Brian D Ziebart, Andrew Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning.

[136] Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, and Ayzaan Wahid, (2023). Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning*, pages 2165–2183.