Learning from demonstration
Imitation learning

Reza Ghabcheloo

IHA 4506 Advanced Robotics

TAMPERE UNIVERSITY OF TECHNOLOGY

Imitation learning

- Learning to do an act by “observing” it

- Note that observation is not necessarily only by cameras. We can observe joint angles, force, tactile, pressure, power

Why learning from demonstration

- Transferring many years of experience to a robot, without needing to program it
- Non-robotics operators can program the robot: frequent need to re-program welding objects with different shape
- Programming difficult to program tasks: earth moving

What's Hidden in the Hidden Layers?

The contents can be easy to find with a geometrical problem, but the hidden layers have yet to give up all their secrets

David S. Touretzky and Dean A. Pomerleau

tions, we fed the network road images taken under a wide variety of viewing angles and lighting conditions. It would be impractical to try to collect thousands of real road images for such a data set. Instead, we developed a synthetic road-image generator that can create as many training examples as we need.

To train the network, 1200 simulated road images are presented 40 times each, while the weights are adjusted using the back-propagation learning algorithm. This takes about 30 minutes on Carnegie Mellon's Warp systolic-array supercomputer. (This machine was designed at Carnegie Mellon and is built by General Electric. It has a peak rate of 100 million floating-point operations per second and can compute weight adjustments for back-propagation networks at a rate of 20 million connections per second.)

Once it is trained, ALVINN can accurately drive the NAVLAB vehicle at about 3½ miles per hour along a path through a wooded area adjoining the Carnegie Mellon campus, under a variety of weather and lighting conditions. This speed is nearly twice as fast as that achieved by non-neural-network algorithms running on the same vehicle. Part of the reason for this is that the forward pass of a back-propagation network can be computed quickly. It takes about 200 milliseconds on the Sun-3/160 workstation installed on the NAVLAB.

The hidden-layer representations ALVINN develops are interesting. When trained on roads of a fixed width, the network chooses a representation in which hidden units act as detectors for complete roads at various positions and orientations. When trained on roads of variable

continued

Photo 1: *The NAVLAB autonomous navigation test-bed vehicle and the road used for trial runs.*

1989, ALVINN first self-driving car using NN

Learning and robotics

- Perception
- Control

Imitation learning

- Imitation learning is a class of methods that reproduces desired behavior based on expert demonstrations.
  - Transferring skills from human to a robotic system
- We need to record demonstrations by experts and learn a policy to reproduce the demonstrated behavior from the recorded data
  - How should we record data of the expert demonstrations? motion capture systems, teleoperation, kinesthetic, etc
  - What should we imitate? redundant information, unnecessary motions

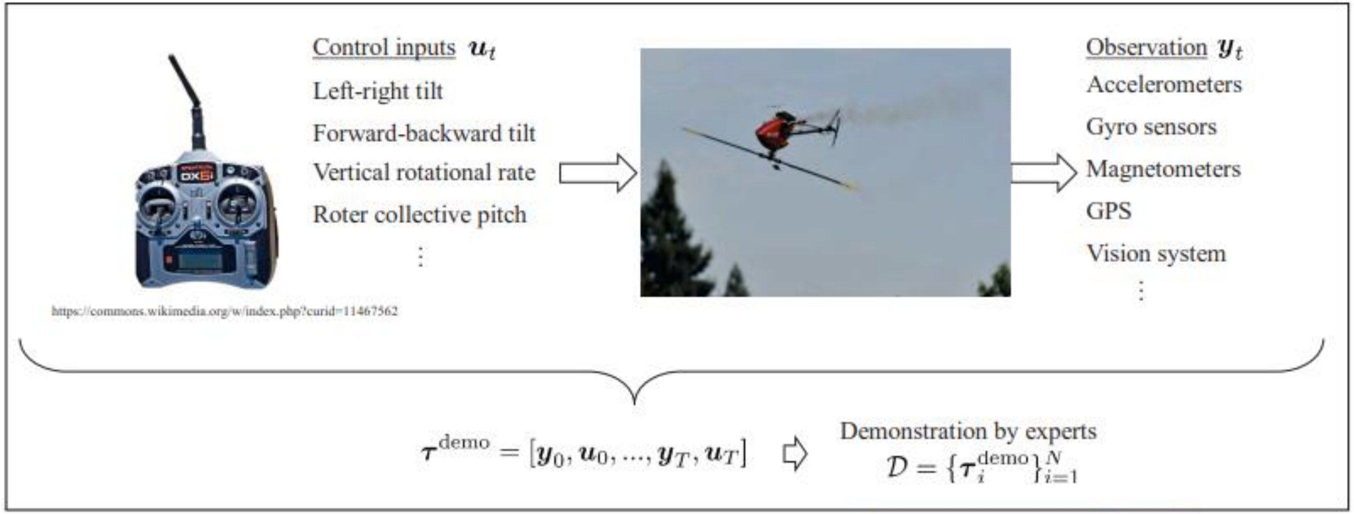
Algorithmic aspects

- How should we represent the policy? symbolic representation (*pick*, *move*, *place*), trajectory-based representation ($x(t)$), and action-state space representation ($u = -kx$).
- How should we learn the policy? Many options, usually related to the choice of policy representation.

Imitation learning

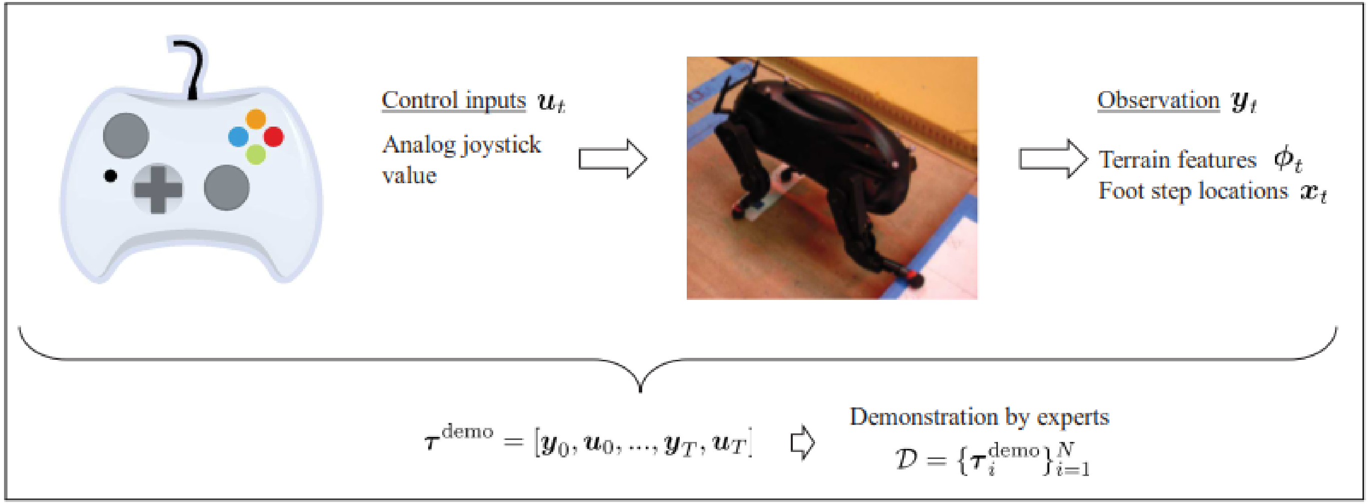
- **Behavioral cloning**
  - directly reproducing desired behavior
  - Often good representation when directly mapping features to actions
- **Inverse optimal control / Inverse reinforcement learning**
  - Learning hidden objective of the desired behavior (cost/reward)
  - Better for more deliberative and long term planning

Example (Behavioral cloning)



(a) Learning of acrobatic RC helicopter maneuvers [Abbeel et al., 2010]. The trajectories for acrobatic flights are learned from a human expert's demonstrations. To control the system with highly nonlinear dynamics, iterative learning control was used.

Example (Inverse optimal control)



(c) Learning quadruped robot locomotion [Zucker et al., 2011]. The footstep planning was addressed as an optimization of the reward/cost function, which was recovered from the expert demonstrations. Learning the reward/cost function allows the footstep planning strategy to be generalized to different terrains.

Behavioral cloning (BC)

- Simplest form of imitation learning
- Expert supplies the data $x_1, u_1, x_2, u_2, \ldots$
- A policy is learned $u = \pi(x), u \sim \pi(u|x)$
- Often $x = (q, \dot{q})$
- It can be treated as a supervised learning

Standard supervised learning

- Expert data set $D = \{x_1, u_1, x_2, u_2, \ldots\}$
- Policy

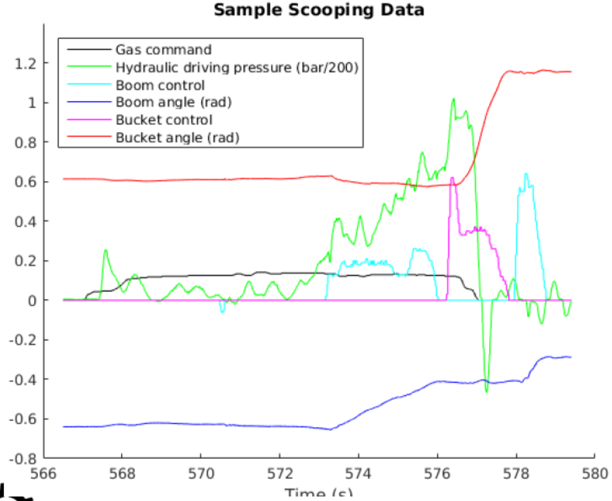
$$u = \pi(x) = \phi(x)^T \theta$$

$$u \sim \pi(u|x) = N(u; \mu(x), \Sigma(x))$$

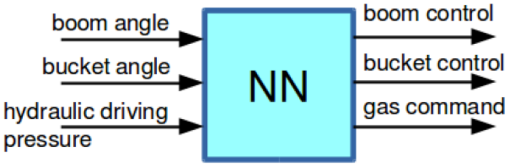
Usually leads to a *regression* problem

- Clean-up effect: Quality of reproduction is better than demonstration! The noise of demonstration has been removed.

Demonstration and data collection

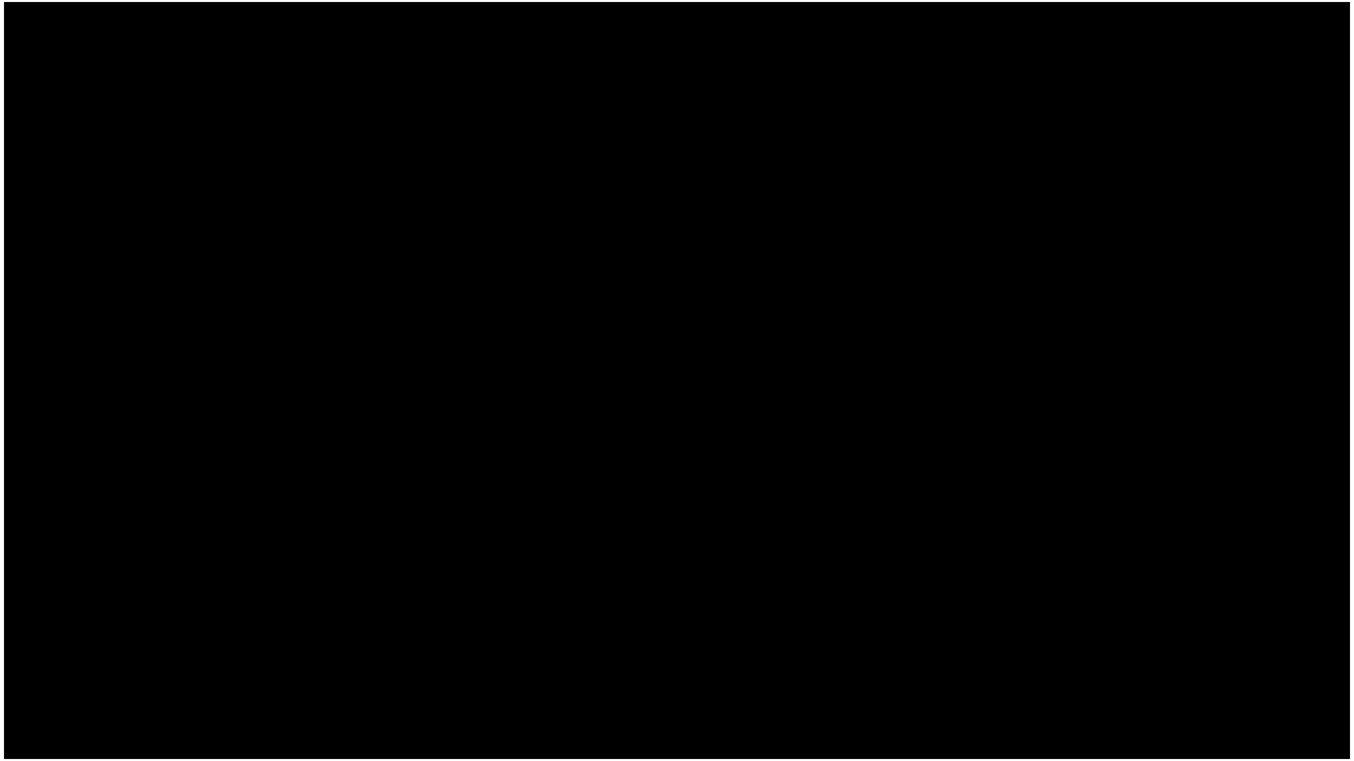


Learning



Autonomous execution

LfD earth moving



Imitation learning (model free BC) vs supervised learning

- Stability: the learner will encounter unknown states during execution
- Distribution of the data: they often differ between demo and actual execution. In supervised learning assumption is i.i.d.

Representations Trajectory

- Trajectory representation

Directly learn desired trajectories $\tau_d(t) = \pi(x_0, \theta)$ and use a motion controller to track it

- Possible representation
  - Linear bases function $\tau_d(t) = \phi(t)^T\theta = \sum \theta_k t^k$

Representations action-state space

- Action-state space representation

Learn the policy that maps state to action

- Possible representation

$$u = \pi(x) = \phi(x)^T \theta = -kx$$

$$u \sim \pi(u|x) = N(u; \mu(x), \Sigma(x))$$

Behavioral cloning

Algorithm 1 Abstract of behavioral cloning

Collect a set of trajectories demonstrated by the expert $\mathcal{D}$

Select a policy representation $\pi_{\boldsymbol{\theta}}$

Select an objective function $\mathcal{L}$

Optimize $\mathcal{L}$ w.r.t. the policy parameter $\boldsymbol{\theta}$ using $\mathcal{D}$

return optimized policy parameters $\boldsymbol{\theta}$

Loss functions

- Least square

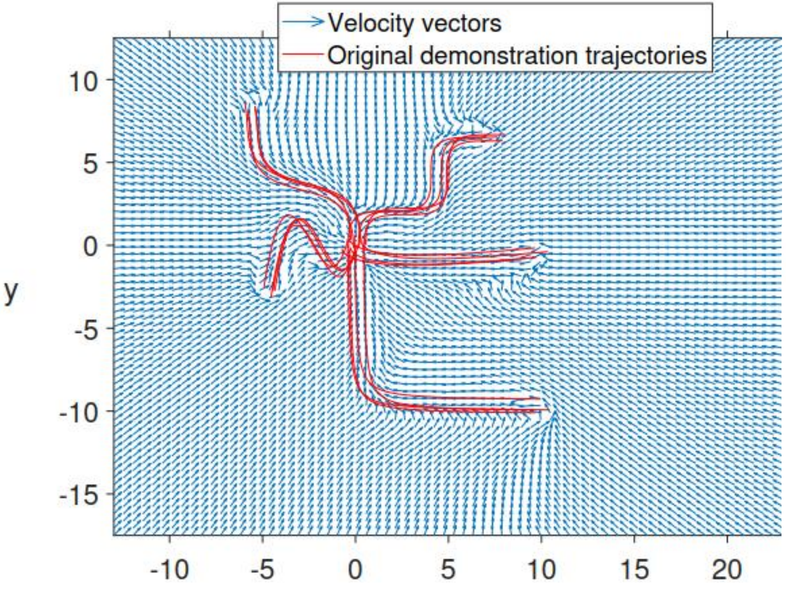
$$L = \left(x^{L} - x^{demo}\right)^{T}\left(x^{L} - x^{demo}\right)$$

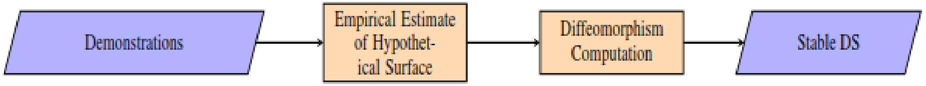
Dynamic motion primitive (S. Schaal), Probabilistic motion primitives (Paraschos), SEDS (Billar),… use LS loss function

Dynamical systems approach

1. Lyapunov function based
    - Khansari-Zadeh and Billard (mixture of Gaussian functions)
    - Neumann and Steil (other related methods summarized in Neumann's RAS'15 paper)
2. Diffeomorphic matching
    - Perrin and Schlehuber-Caissier
    - Andrei, Arun and Reza
3. Dynamic Motion Primitives
    - S. Schaal, J Peters

DS approach





(a) Pipeline

<u>The challenge</u>

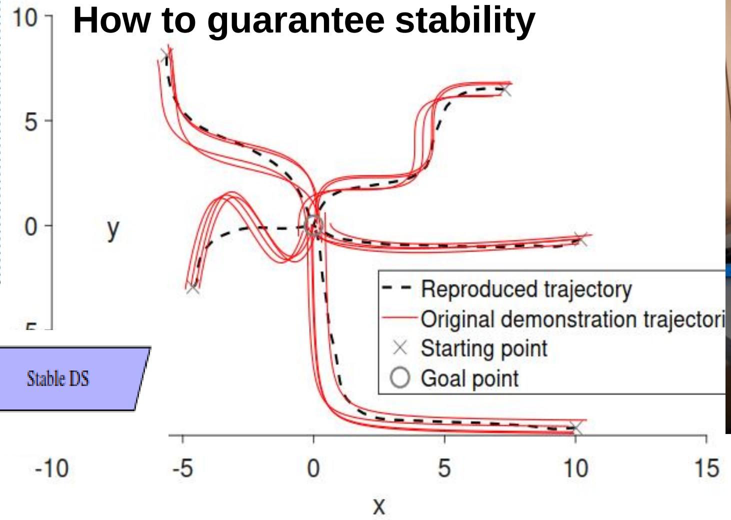
How to learn from few demonstrations

(most compact representation of the behavior)

How to reproduce/control

How to generalize

How to guarantee stability



Ghabcheloo, 2018

Mathematics

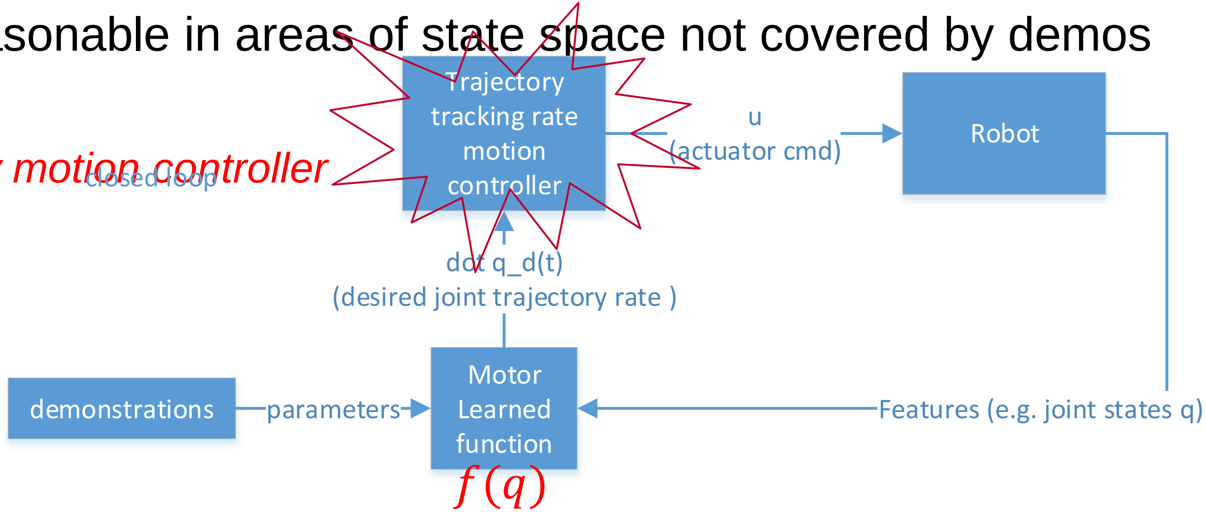
- Learning DS $\dot{x} = f(x)$
- Given set of demonstrations

  $Y = \{y_n(t_k)\}, i = 1, \ldots, N \text{ and } k = 1, \ldots, K$
- N number of demonstrations
- K number of time samples
- t_K demonstration duration

Important features of DS approach

- Guaranteed stability in closed form implementation (reaching target)
- Adapts to changes in the environment and perturbation
- Robust generalization (invariance space and time)
- Behave reasonable in areas of state space not covered by demos

Note the velocity motion controller



Stable Estimator of Dynamical System (SEDS)

$$\dot{\xi} = \hat{f}(\xi) = \sum_{k=1}^{K} h^k(\xi)(A^k \xi + b^k)$$

$$\begin{cases} A^k = \Sigma^k_{\dot{\xi}\xi}(\Sigma^k_{\xi})^{-1} \\ b^k = \mu^k_{\dot{\xi}} - A^k \mu^k_{\xi} \\ h^k(\xi) = \frac{\mathcal{P}(k)\mathcal{P}(\xi|k)}{\sum_{i=1}^{K} \mathcal{P}(i)\mathcal{P}(\xi|i)} \end{cases}$$

Uses quadratic Lyapunov function

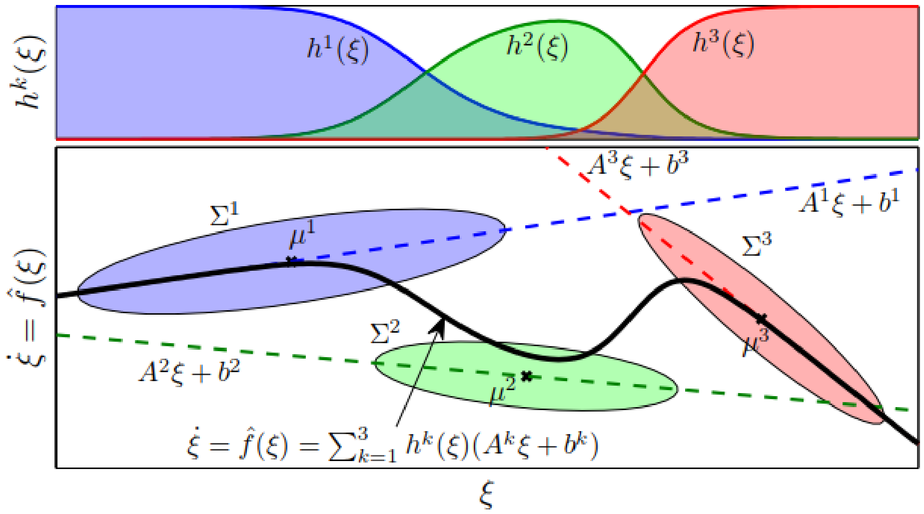
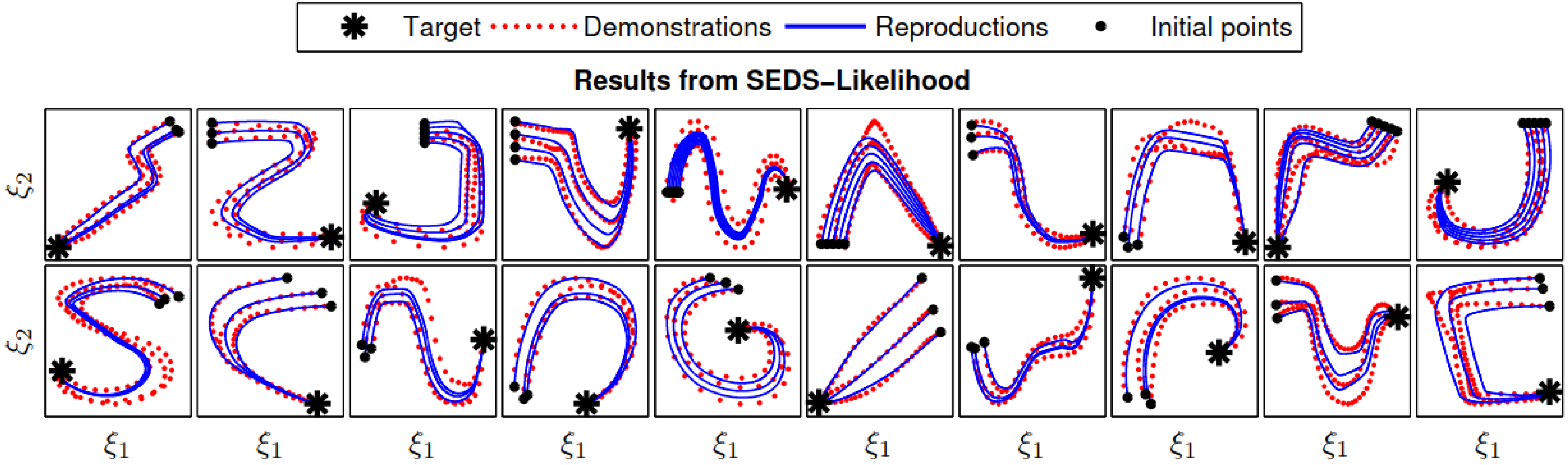


Fig. 4. Illustration of parameters defined in Eq. 8 and their effects on $\hat{f}(\xi)$ for a 1-D model constructed with 3 Gaussians. Please refer to the text for further information.

SEDS



Target
Demonstrations
Reproductions
Initial points
Results from SEDS–Likelihood
ξ_2
ξ_2
ξ_1

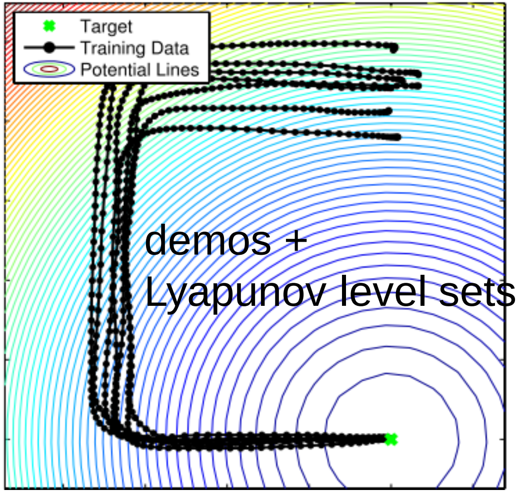
SEDS limitations

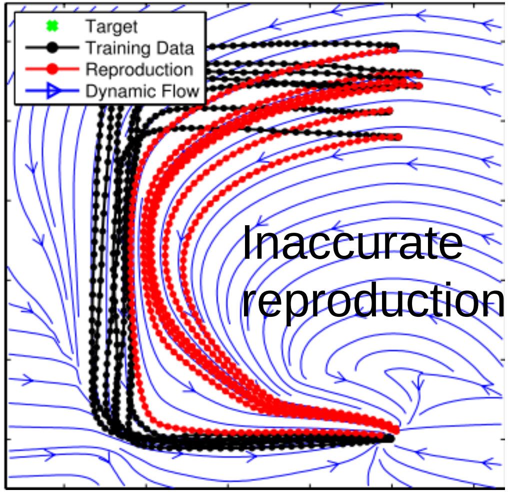
SEDS quadratic Lyapunov function

The distance $(x - x^*)$ can only get smaller

Not useful for imitating complex motions

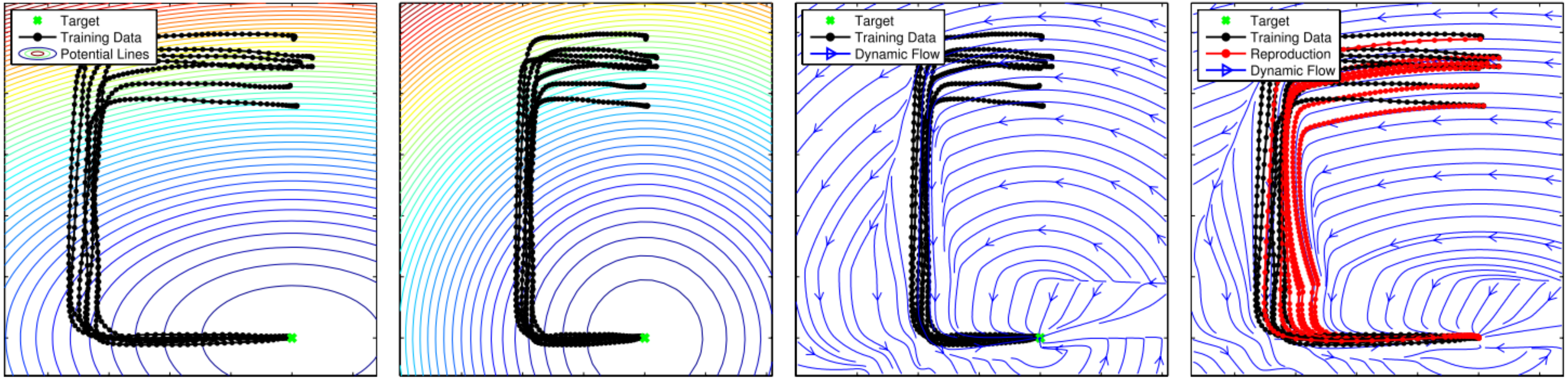
Nuemann: τ-SEDS





Neumann's τ-SEDS

- Transform the data to a space where SEDS work; produce the DS; transform back to the original space



Dynamic Motion Primitives

S Schaal, J Peters

- A stable linear DS perturbed by a nonlinear term
- The nonlinear term $f(s)$ is learned from demos

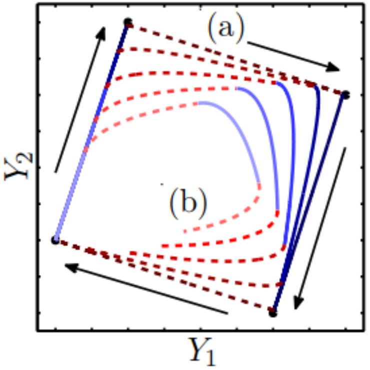
$$\tau\dot{v} = k_p(g - x) - k_d v - k_p(g - x_0)s + k_p f(s)$$

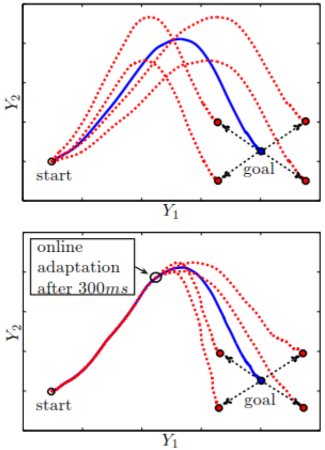
$$\tau\dot{x} = v$$

Obstacle can be added

Goals can change

DMP can be combined





Inverse Optimal Control (Inverse Reinforcement Learning)

- The underlying hypothesis is that human actions are motivated by well-defined rewards and penalties and thus, the problem of IOC boils down to obtaining mathematical models for these rewards and penalties, and then applying classical optimal control methods.

Diffeomorphic matching based

- Find a diffeomorphic match between the data and line $y = x$
- Use it to map orbits (solution trajectories) of $\dot{x} = -x$
- This will results a DS, which is compatible with the demonstration

Diffeomorphic matching

- Given points $X = (x_i)_{i=1,\ldots,N}$ and $\mathrm{Y} = (y_i)_{i=1,\ldots,N}$ compute a diffeomorphism ϕ that maps (x_i) to (y_i) either accurately or approximately

Find ϕ that minimizes

$$\mathrm{dist}(\phi(X), X)$$

The algorithm introduced in Perrin and Schlehuber-Caissier SysContLetters'16

- K=150 number of iteration
- $\beta = 0.5$ learning rate
- $\mu = 0.9$, safety margin
- Move $\mathbf{x}$ in the direction of $\mathbf{v}$

$\psi_{\rho,\mathbf{c},\mathbf{v}}(\mathbf{x}) = \mathbf{x} + exp(-\rho^2\|\mathbf{x}-\mathbf{c}\|^2)\mathbf{v}$

$\phi_K = \psi_{\rho_K,\mathbf{p}_K,\mathbf{v}_K} \circ \cdots \circ \psi_{\rho_1,\mathbf{p}_1,\mathbf{v}_1}$

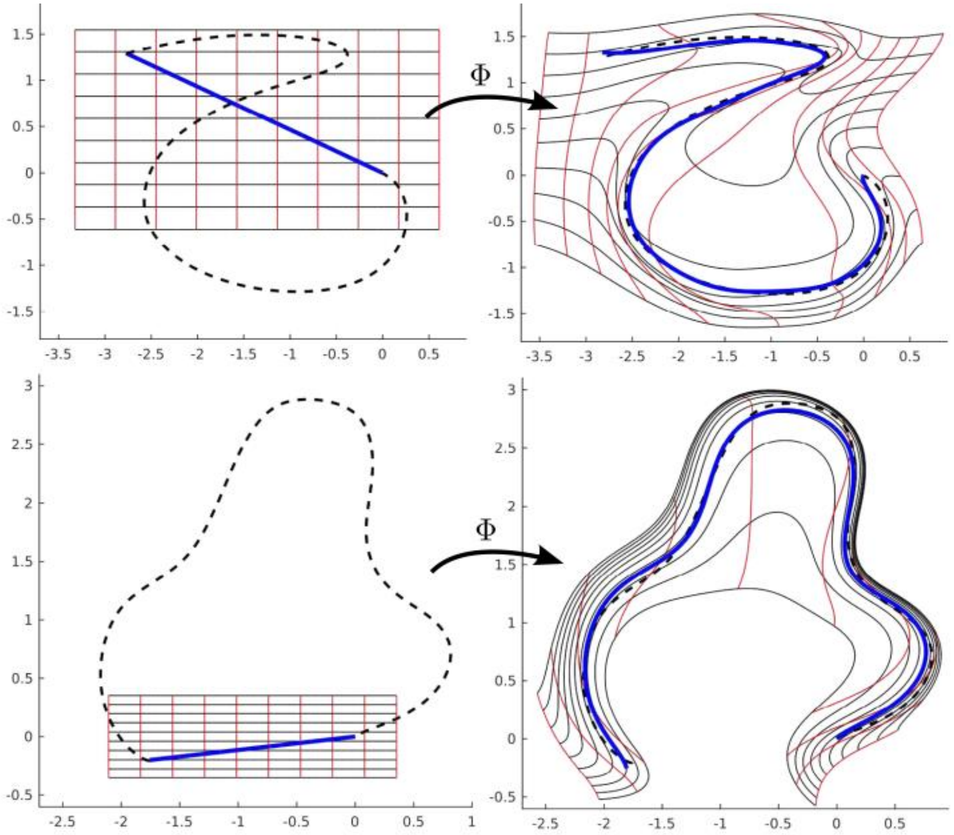
To make sure $\phi_K(0) = 0$, at the end set $\mathbf{p}_{K+1} = \phi_K(0)$ and $\mathbf{v}_{K+1} = -\phi_K(0)$

Input: $\mathbf{X} = (\mathbf{x}_i)_{i\in\{0,\ldots,N\}}$ and $\mathbf{Y} = (\mathbf{y}_i)_{i\in\{0,\ldots,N\}}$
Parameters: $K \in \mathbb{N}_{>0}$, $0 < \mu < 1$, $0 < \beta \leq 1$

$\mathbf{Z} = (\mathbf{z}_i)_{i\in\{0,\ldots,N\}}$
$\mathbf{Z} := \mathbf{X}$
for $j = 1$ to K **do**
    $m := \arg\max_{i\in\{0,\ldots,N\}} (\|\mathbf{z}_i - \mathbf{y}_i\|)$
    $\mathbf{p}_j := \mathbf{z}_m$
    $\mathbf{q} := \mathbf{y}_m$
    $\mathbf{v}_j := \beta(\mathbf{q} - \mathbf{p}_j)$
    $\rho_j := \arg\min_{\rho\in[0,\mu\rho_{\max}(\mathbf{v}_j)]} \left(\mathrm{dist}(\psi_{\rho,\mathbf{p}_j,\mathbf{v}_j}(\mathbf{Z}), \mathbf{Y})\right)$
    $\mathbf{Z} := \psi_{\rho_j,\mathbf{p}_j,\mathbf{v}_j}(\mathbf{Z})$
end for
return $(\rho_j)_{j\in\{1,\ldots,K\}}$, $(\mathbf{p}_j)_{j\in\{1,\ldots,K\}}$, $(\mathbf{v}_j)_{j\in\{1,\ldots,K\}}$

Find the best ρ at each iteration

Example



Diff. matching to a straight line

- $Y = (y_i)_{i=1,\ldots,N}$
- $X = (x_i)_{i=1,\ldots,N} = \left(y_0 + \frac{i}{N}(y_N - y_0)\right)_{i=1,\ldots,N}$

that is a line connecting y_0 to y_N

They claim their algorithm run 57 times faster and 2.7times better accuracy that SoA algorithms

Theorem

Definition 4. Two DS $\dot{\mathbf{x}} = f(\mathbf{x})$ and $\dot{\mathbf{x}} = g(\mathbf{x})$ are said to be diffeomorphic, or smoothly equivalent, if there exists a diffeomorphism $\Phi : \mathbb{R}^d \to \mathbb{R}^d$ such that:

$$\forall \mathbf{x} \in \mathbb{R}^d, \quad g(\Phi(\mathbf{x})) = J_\Phi(\mathbf{x}) f(\mathbf{x}),$$

where $J_\Phi(x)$ is the Jacobian matrix: $J_\Phi(\mathbf{x}) = \frac{\partial \Phi}{\partial \mathbf{x}}(\mathbf{x})$. If Φ is a $\mathcal{C}^k$-diffeomorphism, then the DS are said to be $\mathcal{C}^k$-diffeomorphic.

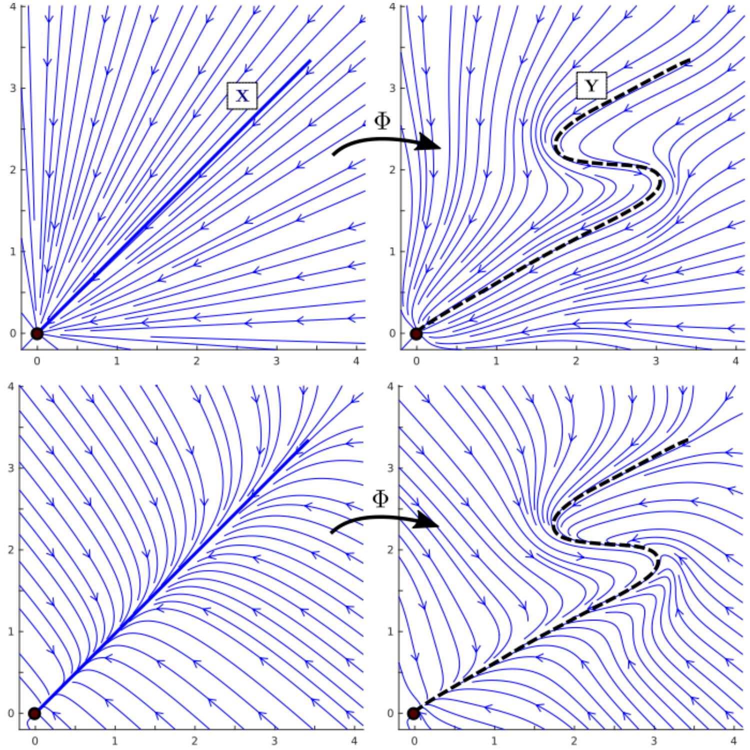
Theorem 3. *If two DS $\dot{\mathbf{x}} = f(\mathbf{x})$ and $\dot{\mathbf{x}} = g(\mathbf{x})$ are diffeomorphic, then if one is globally asymptotically stable, both are.*

How do we use this theorem

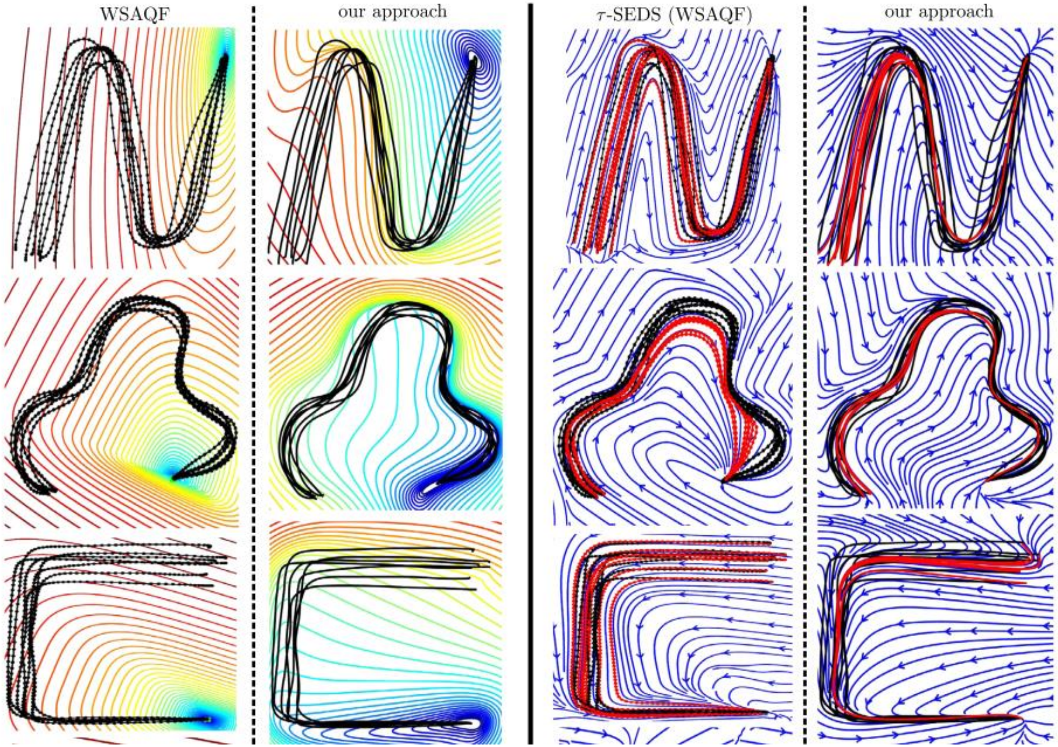
- Start with DS1 $\dot{\mathbf{x}} = -\gamma(\mathbf{x})\mathbf{x}$
- $\gamma(\mathbf{x}) = \begin{cases} \frac{\|\mathbf{y}(0)\|}{N\Delta t\|\mathbf{x}\|} & \|\mathbf{x}\| \geq \frac{\|\mathbf{y}(0)\|}{N} \\ \frac{\|\mathbf{y}(0)\|}{N} & \text{otherwise} \end{cases}$ adjusts the velocity without modifying the orbits
- The DS1 is stable and $\mathbf{x}(i\Delta t) = \frac{N-i}{N}\mathbf{y}(i\Delta t)$
- ϕ then transforms the DS1 to DS2 which reproduces the demonstrations and the velocity profiles
- DS2 $\dot{\mathbf{x}} = -\gamma\left(\phi^{-1}(\mathbf{x})\right)J_{\phi}(\phi^{-1}(\mathbf{x}))\ \phi^{-1}(\mathbf{x})$

Effect of choice of \gamma

- Both rows reproduce the demo but with different behavior in other places of the state space



Diffeomorphism approach ("our approach")



One can also calculate a compatible Lyapunov function

Theorem 4. *Let* $\dot{\mathbf{x}} = f(\mathbf{x})$ *and* $\dot{\mathbf{x}} = g(\mathbf{x})$ *be two* $\mathcal{C}^1$*-diffeomorphic DS, and let* Φ *be a* $\mathcal{C}^1$*-diffeomorphism such that* $\forall \mathbf{x} \in \mathbb{R}^d$, $g(\Phi(\mathbf{x})) = J_\Phi(\mathbf{x}) f(\mathbf{x})$. *If* L *is a Lyapunov function for* $\dot{\mathbf{x}} = f(\mathbf{x})$, *then* $L \circ \Phi^{-1}$ *is a Lyapunov function for* $\dot{\mathbf{x}} = g(\mathbf{x})$.

Some very good read papers

- An Algorithmic Perspective on Imitation Learning, 2018 Foundations and Trends® in Robotics, T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel and J. Peters (182 pages)
- The limits and potentials of deep learning for robotics, IJRR 2018, Niko Sünderhauf, Oliver Brock, Walter Scheirer, Raia Hadsell, Dieter Fox, Jürgen Leitner, Ben Upcroft, Pieter Abbeel, Wolfram Burgard, Michael Milford, and Peter Corke (16 pages)
- ALVINN original paper
- Ideas of some slides are borrowed from Jan Peters' course on ML.