Micro-data policy search

Learning in a handful of trials

Jean-Baptiste Mouret & Konstantinos Chatzilygeroudis

Part 1

Priors on Policy Structures

Konstantinos Chatzilygeroudis

Direct Policy Search

- We assume the world is a system described as follows:

$$x_{t+1} = f(x_t, u_t) + \omega_t$$

- In policy search, we assume that we control our robots/agents through a parameterized policy $\pi(u|x, \theta)$, θ are the parameters of the policy

- We assume the existence of an immediate reward function $r(x_t, u_t, x_{t+1})$

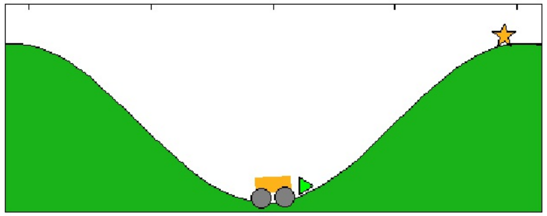
- The goal of policy search is to find the policy parameters that maximize long-term reward:

$$J(\theta) = \mathbb{E}[\sum_{t=0}^{T-1} r(x_t, u_t, x_{t+1})\,|\theta]$$

Example: Continuous Mountain Car

See notebook

- Continuous Mountain Car
  - we want to “climb up” the mountain
  - we control the force applied to the car
  - but we do not have enough power to climb directly
  - state: $[x, \dot{x}]$
  - Continuous version of the problem



https://en.wikipedia.org/wiki/Mountain_car_problem

Example: Continuous Mountain Car

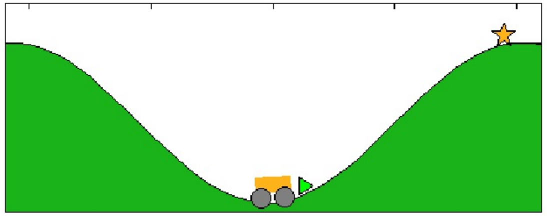
OpenAI gym environment (notebook)

```python
import gym

env = gym.make("MountainCarContinuous-v0")
max_episode_steps = 999
```

```python
steps = 0
total_steps = 0
max_steps = 1500

state = env.reset()
while True:
    action = env.action_space.sample() # sample random action
    next_state, reward, done, _ = env.step(action) # step the world
    state = next_state.copy()
    # env.render() # we could even render
    steps = steps + 1
    total_steps = total_steps + 1
    if total_steps >= max_steps:
        break
    if done or steps >= max_episode_steps:
        state = env.reset()
        steps = 0
```



https://en.wikipedia.org/wiki/Mountain_car_problem

- Continuous Mountain Car
  - we want to “climb up” the mountain
  - we control the force applied to the car
  - but we do not have enough power to climb directly
  - state: $[x, \dot{x}]$

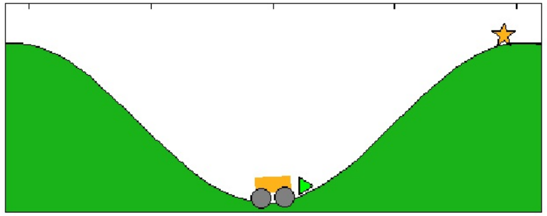
Example: Continuous Mountain Car

Execute an episode! (notebook)

```python
def f(x, display=False):
    X = x.copy()
    X = X.reshape((1, D+A))
    M = X[0, :D]
    b = X[0, -A]

    state = env.reset()
    steps = 0
    total_reward = 0.
    while True:
        action = np.array([M @ state + b])
        next_state, reward, done, _ = env.step(action)
        total_reward += reward
        if display:
            env.render()
        steps = steps + 1
        if done or steps >= max_episode_steps:
            break
        state = next_state.copy()

    return -total_reward
```



https://en.wikipedia.org/wiki/Mountain_car_problem

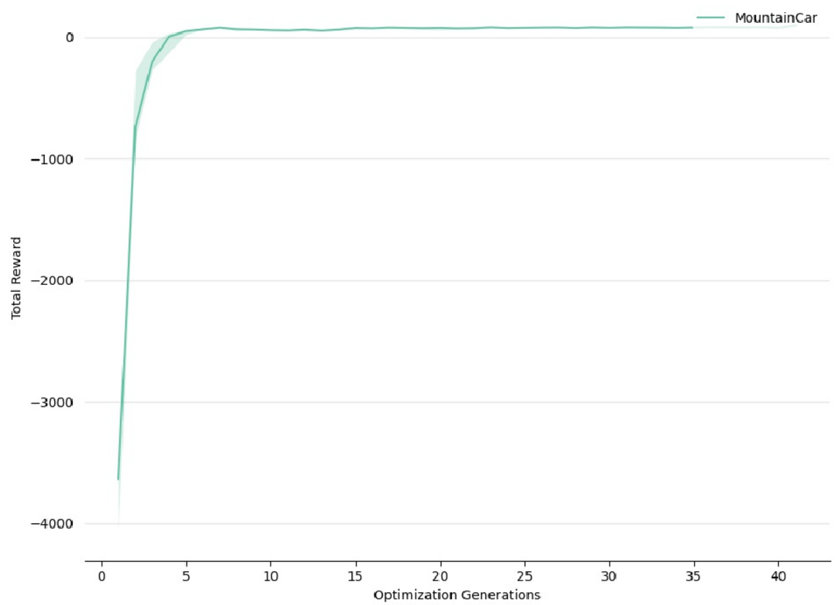
Simple Linear Policy

- s the current state
- $F = As + b$

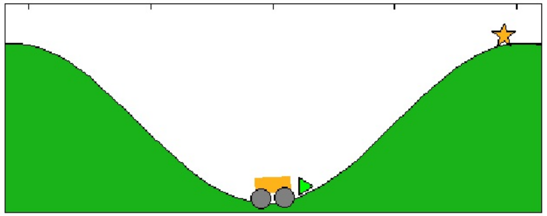
- Parameters of the policy
  - A
  - b

Example: 2D dynamic particle

Let's optimize for the policy (notebook)



We need around 500 (5*100) episodes to solve the task!!!



https://en.wikipedia.org/wiki/Mountain_car_problem

Policy Search

- We treat the problem as black-box optimization
- We use the simplest evolutionary strategy
- Population of 100 individuals

$\boldsymbol{\theta} = (\boldsymbol{\mu}, \sigma), p_{\boldsymbol{\theta}}(\boldsymbol{x}) \sim \mathcal{N}(\boldsymbol{\mu}, \sigma^2 \boldsymbol{I})$ → Gaussian distribution

1. $\boldsymbol{\theta} = \boldsymbol{\theta}_0$
2. $P = \{\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_{N-1} \mid \boldsymbol{x}_i = \boldsymbol{\mu}_t + \sigma_t^2 \boldsymbol{y}_i,\ \boldsymbol{y}_i \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})\}$, population of N
3. $F = \{J(\boldsymbol{x}_0), J(\boldsymbol{x}_1), \ldots, J(\boldsymbol{x}_{N-1})\}$
4. $P_{elite} = \{\boldsymbol{x}_k | \boldsymbol{x}_k$ in top-λ given $F\}$
5. $\boldsymbol{\mu}_{t+1} = \frac{1}{\lambda} \sum_{\boldsymbol{x}_k \in P_{elite}} \boldsymbol{x}_k$ (sample mean)
6. $\sigma_{t+1}^2 = \frac{1}{\lambda} \sum_{\boldsymbol{x}_k \in P_{elite}} (\boldsymbol{x}_k - \boldsymbol{\mu}_t)^2$ (sample variance)
7. Back to 2 until convergence

Priors in policies

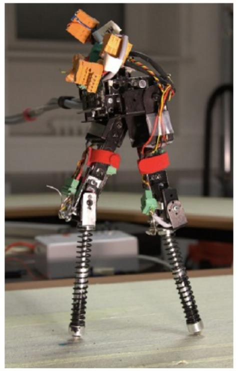
- Direct policy search seems effective

- ... but how can we take into account expert knowledge to learn faster?

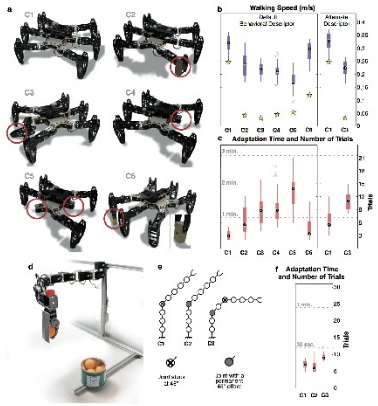
- One effective way is to use “structured” policy types
  - e.g., way-points of a trajectory to follow
  - QP-based whole-body controller
  - Dynamical Movement Primitives
  - Autonomous Dynamical Systems
  - ...

Types of policy structures

- **Hand-designed task-specific policies**
  - Finite state machines
  - Open-loop policies
  - ...



Calandra, R., Seyfarth, A., Peters, J., & Deisenroth, M. P. (2016). Bayesian optimization for learning gaits under uncertainty. Annals of Mathematics and Artificial Intelligence, 76(1), 5-23.



Cully, A., Clune, J., Tarapore, D., & Mouret, J. B. (2015). Robots that can adapt like animals. Nature, 521(7553), 503-507.

- **Trajectory-based policies**
  - Dynamical Movement Primitives (DMPs)

$$\omega\ddot{\boldsymbol{\xi}} = \underbrace{\alpha(\beta(\boldsymbol{\xi}^g - \boldsymbol{\xi}) - \dot{\boldsymbol{\xi}})}_{\text{Spring-damper system}} + \underbrace{s\, f_{\boldsymbol{\theta}}(s)}_{\text{Forcing term}}$$

$$\omega\dot{s} = -\alpha_s s.$$

  - Autonomous Dynamical System (no time)

$$\dot{\boldsymbol{\xi}} = \pi_{\text{seds}}(\boldsymbol{\xi})$$

  - ...

Trajectory-based policies

- **Goals**
  - Trade-off between expressivity and searchability
  - Exploit dynamical systems properties (e.g., stability, smoothness)
- **Trajectory-based policies**
  - Dynamical Movement Primitives (DMPs)

$$\omega \ddot{\boldsymbol{\xi}} = \underbrace{\alpha(\beta(\boldsymbol{\xi}^g - \boldsymbol{\xi}) - \dot{\boldsymbol{\xi}})}_{\text{Spring-damper system}} + \underbrace{s\, f_{\boldsymbol{\theta}}(s)}_{\text{Forcing term}}$$

s converges to zero

$$\omega \dot{s} = -\alpha_s s.$$

  - Autonomous Dynamical Systems (no time)

$$\dot{\boldsymbol{\xi}} = \pi_{\text{seds}}(\boldsymbol{\xi})$$

$$\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = -\frac{1}{T} \sum_{n=1}^{N} \sum_{t=0}^{T^n} \log \mathcal{P}(\xi^{t,n}, \dot{\xi}^{t,n} | \boldsymbol{\theta})$$

subject to

$$\begin{cases} (\boldsymbol{a})\ b^k = -A^k \xi^* \\ (\boldsymbol{b})\ A^k + (A^k)^T \prec 0 \\ (\boldsymbol{c})\ \Sigma^k \succ 0 & \forall k \in 1 \dots K \\ (\boldsymbol{d})\ 0 < \pi^k \leq 1 \\ (\boldsymbol{e})\ \sum_{k=1}^{K} \pi^k = 1 \end{cases}$$

Khansari-Zadeh, S.M. and Billard, A., 2011. Learning stable nonlinear dynamical systems with gaussian mixture models. IEEE Transactions on Robotics, 27(5), pp.943-957.

How to learn the policies/controllers?

- **Non-differentiable policies**
  - Any black-box optimizer
  - e.g., Bayesian optimization
  - e.g., evolutionary strategies
  - ....

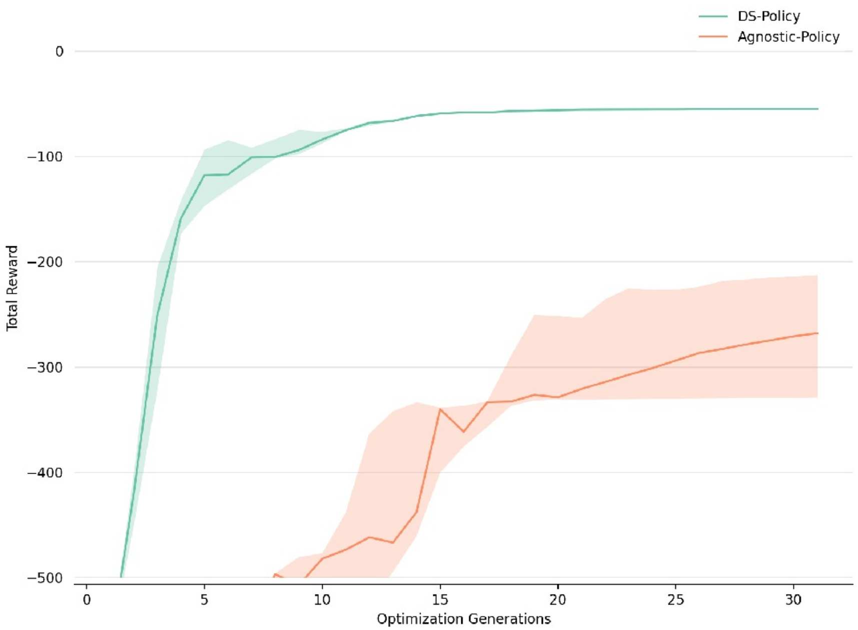
- **Differentiable policies**
  - Any suitable optimizer
  - e.g., all of the above ;)
  - e.g., gradient based optimization
  - e.g., policy gradient methods!
  - ....

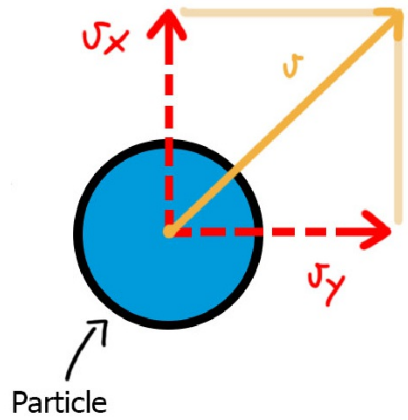
- **Learning from demonstrations**
  - Use an offline buffer/achive of previous interactions
  - Might not contain actions!
  - Dynamical system-based policies operate superbly here

- **Trial-and-error Learning**
  - Warm-start with demos?

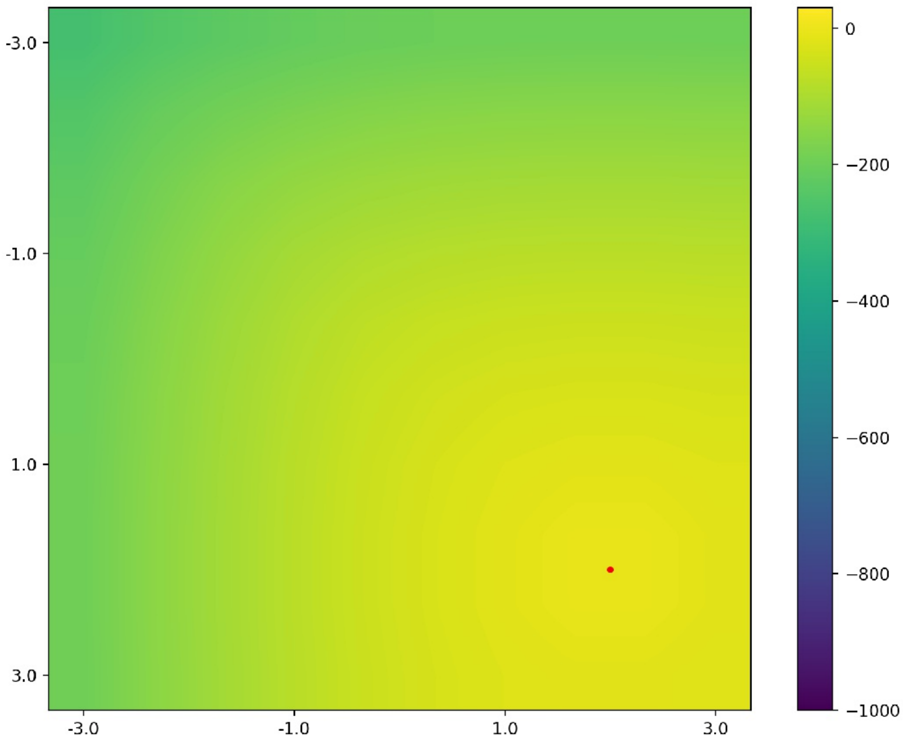
  - Data-efficient optimization
    - Bayesian Optimization
    - (1+1)-CMA-ES
    - ...

But... what do we really gain?

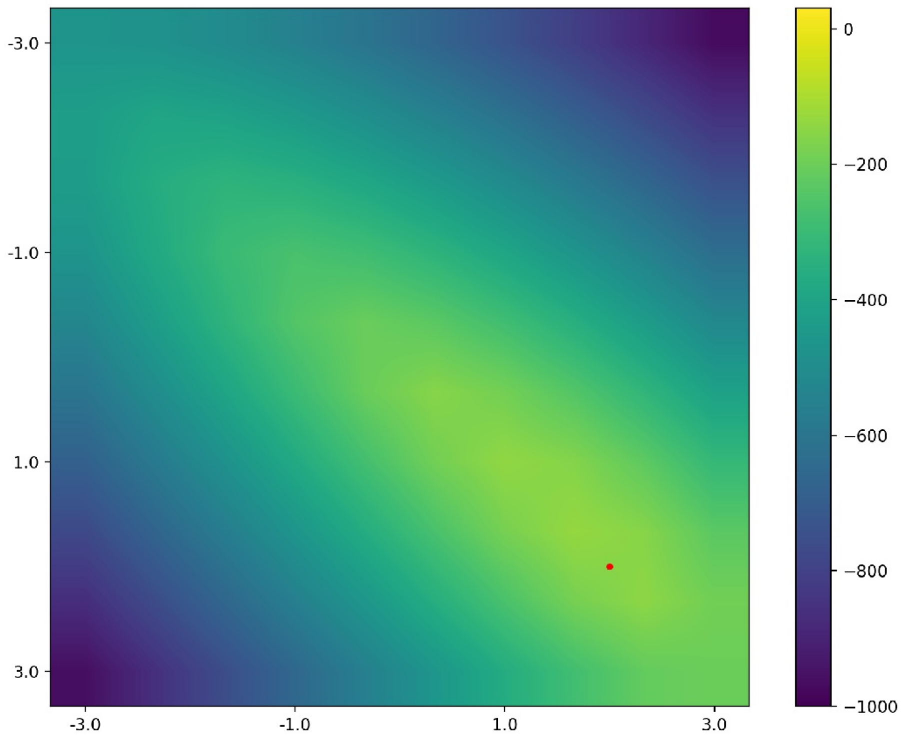




But... what do we really gain?



Structured Policy

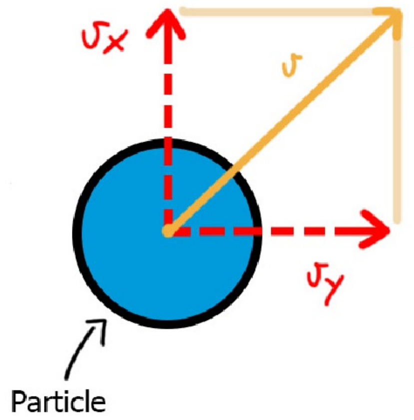


Agnostic Policy

Example: 2D dynamic particle

See notebook

- 2D particle
  - mass of 1kg
  - we control the force applied to the particle
  - state: $[x, y, \dot{x}, \dot{y}]$
  - equations of motion:
    - $p = [x, y], v = [\dot{x}, \dot{y}], a = F/m$
    - $v_{t+1} = v_t + a * dt$
    - $p_{t+1} = p_t + v_{t+1} * dt$
  - target is to reach a point with zero velocity



Example: 2D dynamic particle

Particle environment (notebook)

```python
class ParticleEnv:
    def __init__(self):
        self.dt = 0.01
        self.m = 1.
        self.b = 0.1
        self.max_force = 2.
        self.max_vel = 5.
        self.target = np.array([2., 2.])

        self.reset()

    def reset(self, random_initial=False):
        if random_initial:
            high = np.array([2., 2., 0.5, 0.])
            self.state = np.random.uniform(low=-high, high=high)
        else:
            self.state = np.array([0., 0., 0., 0.])  # pos, vel
        return self.state
```

```python
def step(self, u):
    u = np.clip(u, -self.max_force, self.max_force)

    p = self.state[:2]
    v = self.state[2:]

    acc = u/self.m
    n_skip = 5
    for _ in range(n_skip):
        v = v + acc * self.dt
        v = np.clip(v, -self.max_vel, self.max_vel)
        p = p + v * self.dt

    reward = -np.linalg.norm(p-self.target)

    self.state = np.array([p[0], p[1], v[0], v[1]])

    return self.state, reward
```

- 2D particle
  - mass of 1kg
  - we control the force applied to the particle
  - state: $[x, y, \dot{x}, \dot{y}]$
  - equations of motion:
    - $p = [x, y], v = [\dot{x}, \dot{y}], a = F/m$
    - $v_{t+1} = v_t + a * dt$
    - $p_{t+1} = p_t + v_{t+1} * dt$
  - target is to reach a point with zero velocity

Example: 2D dynamic particle

How does an agnostic policy looks like? (notebook)

```
def unstructured_policy(x, display=False, random_initial=False, initial_state=None):
    X = x.copy()
    X = X.reshape((1, 4+2))
    M = X[0, :4]
    b = X[0, 4:]

    def pol(state):
        return np.array([M @ state + b]).reshape((2,))

    return func(pol, display, random_initial, initial_state)
```

Simple Linear Policy

- s the current state
- $F = As + b$

- Parameters of the policy
    - A
    - b

Example: 2D dynamic particle

How does a structured policy looks like? (notebook)

```
def structured_policy(x, display=False, random_initial=False, initial_state=None):
    X = x.copy()
    X = X.reshape((1, 4+2))
    M = np.diag(np.exp(X[0, :4]))
    t = X[0, 4:]
    t = np.array([t[0], t[1], 0., 0.])  # we assume zero target velocity

    def pol(state):
        vel_commands = (M @ (t-state)).reshape((4, 1))
        Kp = 10.
        Kd = 10.
        u = Kp*vel_commands[:2] + Kd*vel_commands[2:]
        return u.reshape((2,))

    return func(pol, display, random_initial, initial_state)
```

Stable Linear Policy

- t is the target
- s the current state
- $\dot{s}_d = A(t - s)$
- $F = K\dot{s}_d$
- if A is positive definite, then the state will always converge to target

- Parameters of the policy
  - A
  - t

Example: 2D dynamic particle

Let's optimize the policies (notebook)

```
all_vals_structured = []
all_vals_unstructured = []
final_mus_structured = []
final_mus_unstructured = []
N_runs = 5

for _ in range(N_runs):
    mu = np.zeros((6, 1)) # initial estimate
    sigma = np.ones((6, 1))
    # run optimization and get result
    final_mu_structured, _, values_structured, _ = run_es(structured_policy, mu, sigma, 20, 5, 30, verbose=True)
    all_vals_structured += [values_structured]
    final_mus_structured += [final_mu_structured]

    mu = np.ones((6, 1)) # initial estimate
    sigma = np.ones((6, 1))
    # run optimization and get result
    final_mu_unstructured, _, values_unstructured, _ = run_es(unstructured_policy, mu, sigma, 20, 5, 30, verbose=True)
    all_vals_unstructured += [values_unstructured]
    final_mus_unstructured += [final_mu_unstructured]
```

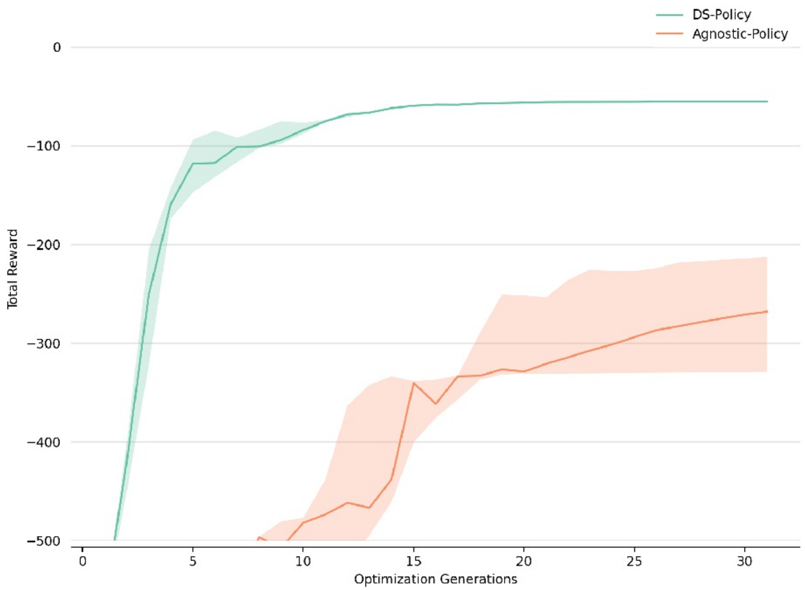
Policy Search

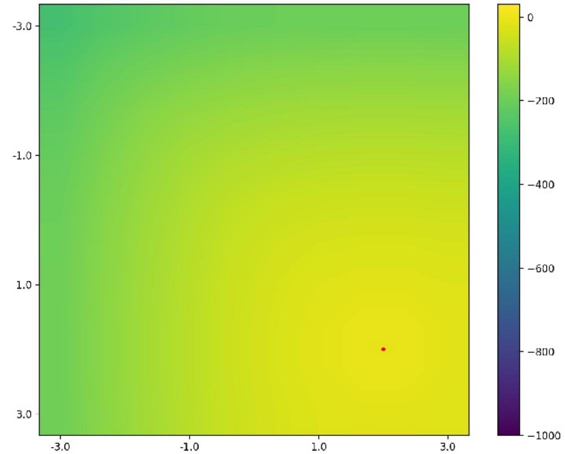
- We treat the problem as black-box optimization
- We use the simplest evolutionary strategy

$\boldsymbol{\theta} = (\boldsymbol{\mu}, \sigma), p_{\boldsymbol{\theta}}(\boldsymbol{x}) \sim \mathcal{N}(\boldsymbol{\mu}, \sigma^2 \boldsymbol{I})$ → Gaussian distribution

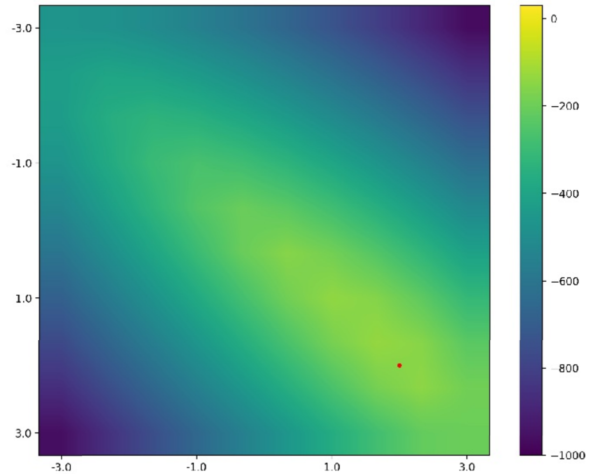
1. $\boldsymbol{\theta} = \boldsymbol{\theta}_0$
2. $P = \{\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_{N-1} \mid \boldsymbol{x}_i = \boldsymbol{\mu}_t + \sigma_t^2 \boldsymbol{y}_i,\ \boldsymbol{y}_i \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})\}$, population of N
3. $F = \{J(\boldsymbol{x}_0), J(\boldsymbol{x}_1), \ldots, J(\boldsymbol{x}_{N-1})\}$
4. $P_{elite} = \{\boldsymbol{x}_k | \boldsymbol{x}_k$ in top-λ given $F\}$
5. $\boldsymbol{\mu}_{t+1} = \frac{1}{\lambda} \sum_{\boldsymbol{x}_k \in P_{elite}} \boldsymbol{x}_k$ (sample mean)
6. $\sigma_{t+1}^2 = \frac{1}{\lambda} \sum_{\boldsymbol{x}_k \in P_{elite}} (\boldsymbol{x}_k - \boldsymbol{\mu}_t)^2$ (sample variance)
7. Back to 2 until convergence

Results?



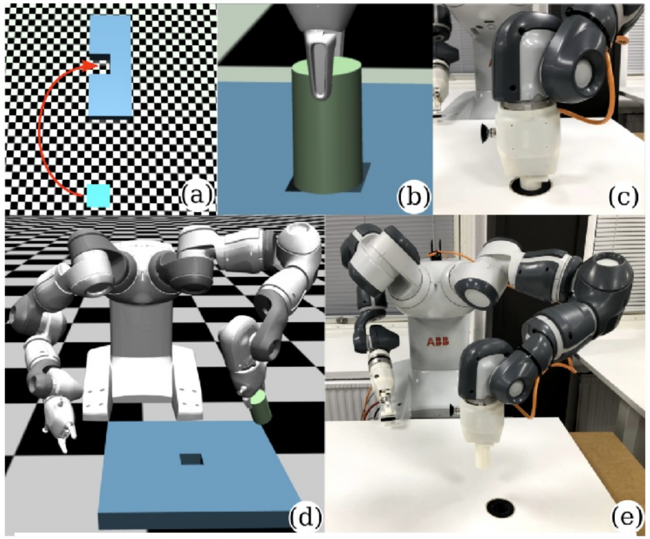


Structured Policy



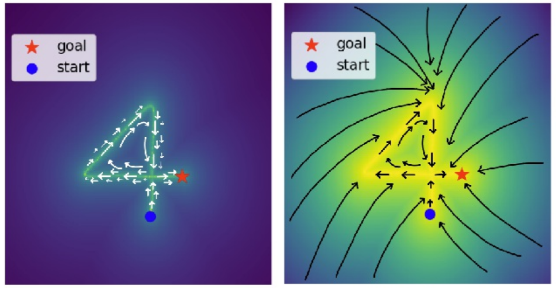
Agnostic Policy

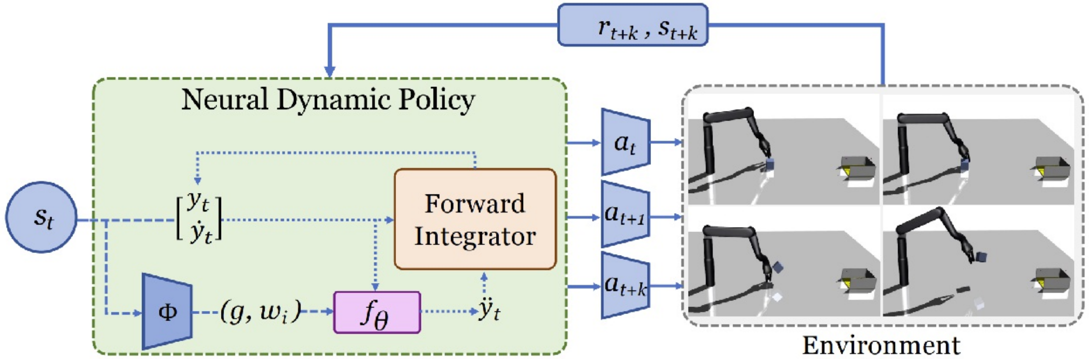
Does it work with real robots/systems?

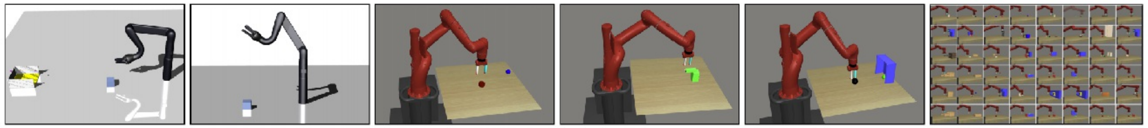


$$
\begin{aligned}
\mathbf{u} &= \pi(\mathbf{x}, \dot{\mathbf{x}}; \eta, \psi, \phi) \\
&= -\frac{\partial}{\partial \mathbf{x}}(\mathbf{x}^T \mathbf{S}(\eta)\mathbf{x} + \Psi(\mathbf{x}; \psi)) - \mathbf{D}(\dot{\mathbf{x}}; \phi)\dot{\mathbf{x}} \\
&= -\mathbf{S}(\eta)\mathbf{x} - \frac{\partial}{\partial \mathbf{x}}\Psi(\mathbf{x}; \psi) - \mathbf{D}(\dot{\mathbf{x}}; \phi)\dot{\mathbf{x}}
\end{aligned}
$$

Khader, S.A., Yin, H., Falco, P. and Kragic, D., 2021. Learning Deep Neural Policies with Stability Guarantees. arXiv preprint arXiv:2103.16432.

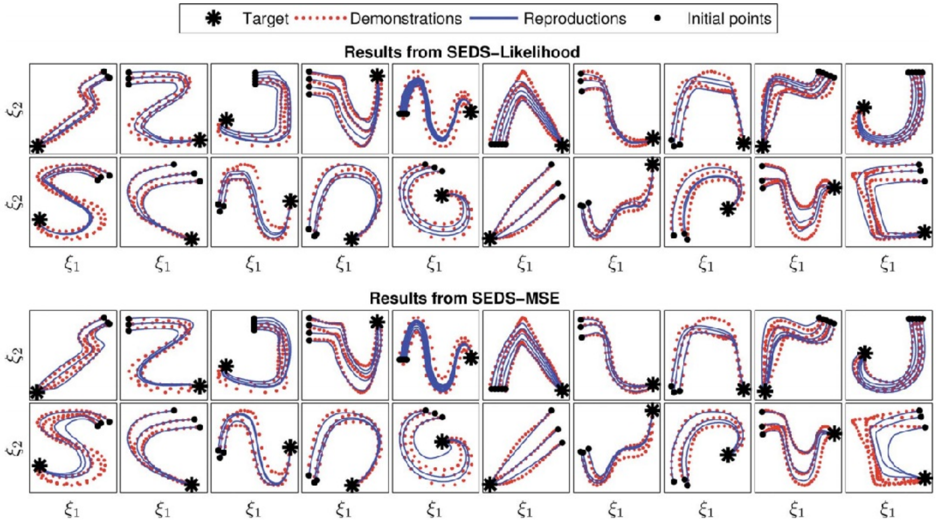




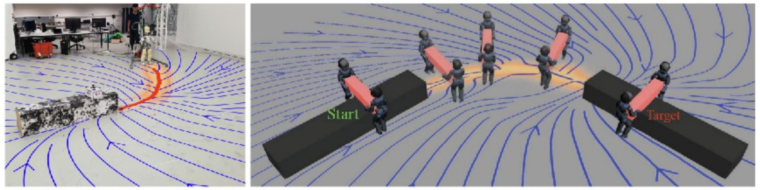


Bahl, S., Mukadam, M., Gupta, A. and Pathak, D., 2020. Neural Dynamic Policies for End-to-End Sensorimotor Learning. NeurIPS.

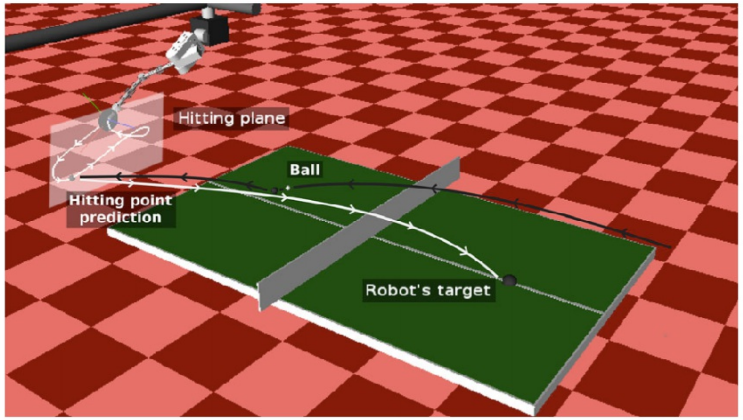
Does it work with real robots/systems?



Khansari-Zadeh, S.M. and Billard, A., 2011. Learning stable nonlinear dynamical systems with gaussian mixture models. IEEE Transactions on Robotics, 27(5), pp.943-957.

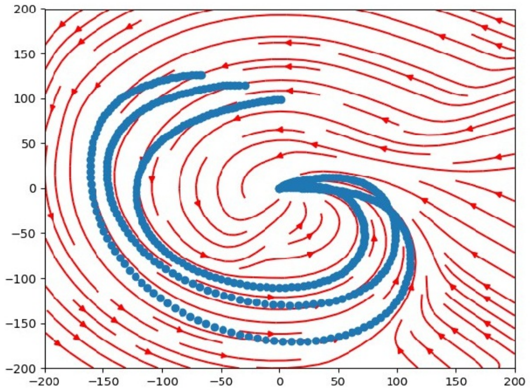


Figueroa, N., Faraji, S., Koptev, M. and Billard, A., 2020, May. A Dynamical System Approach for Adaptive Grasping, Navigation and Co-Manipulation with Humanoid Robots. In 2020 IEEE International Conference on Robotics and Automation (ICRA) (pp. 7676-7682). IEEE.



Paraschos, A., Daniel, C., Peters, J. and Neumann, G., 2018. Using probabilistic movement primitives in robotics. Autonomous Robots, 42(3), pp.529-551.

Conclusions: Priors on Policy Structures



Selecting the policy parameter space is important:

- We need expressive representations

- We need representations that can be efficiently searchable

Exploiting Dynamical Systems

- Long history in the robotics literature
- Specific but quite generic
- Desired properties:
  - Stability
  - Smoothness
  - Explainability

$$\omega\ddot{\boldsymbol{\xi}} = \underbrace{\alpha(\beta(\boldsymbol{\xi}^g - \boldsymbol{\xi}) - \dot{\boldsymbol{\xi}})}_{\text{Spring-damper system}} + \underbrace{s\, f_{\boldsymbol{\theta}}(s)}_{\text{Forcing term}}$$

$$\omega\dot{s} = -\alpha_s s.$$