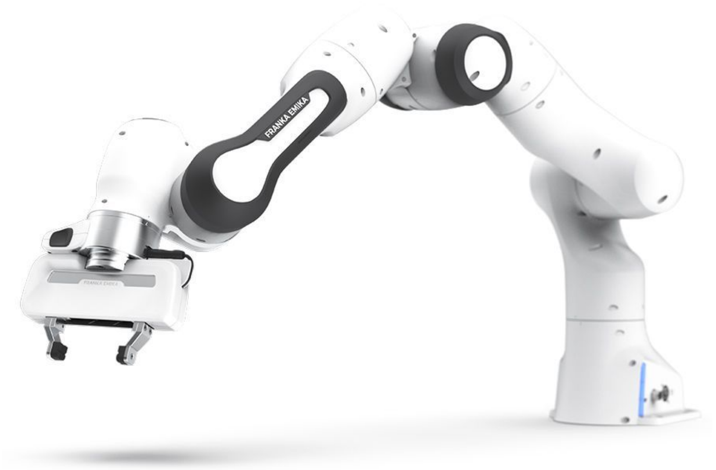
Learning to Control a Low-Cost Manipulator using Data-Efficient Reinforcement Learning

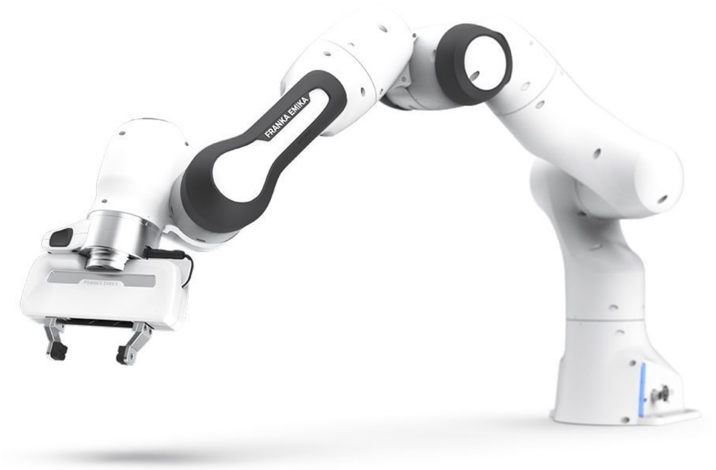
Presenter: Rahul Menon

September 28, 2023

Motivation: Manipulators



Motivation: Manipulators

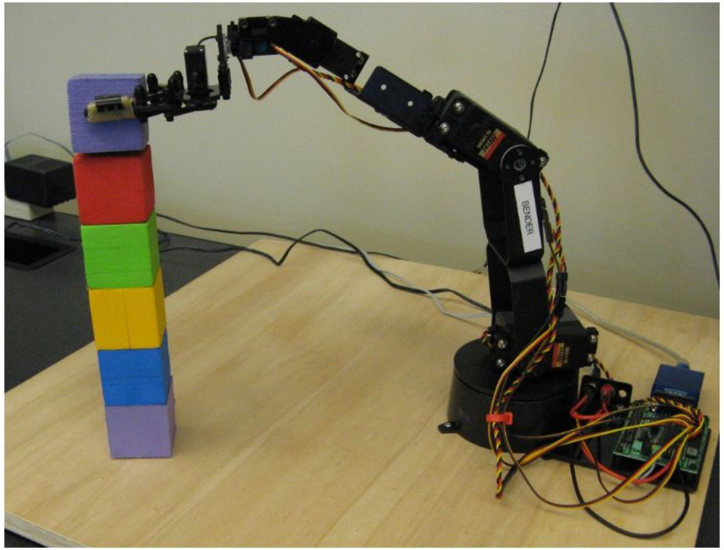


$10,000’s each

Motivation: Manipulators

Expensive manipulators have better sensors and more precise movement

Motivation: Manipulators



Low-cost manipulators have worse sensors and less precision

Motivation: Model-based RL

- Most RL for robotics is inefficient
- Large amounts of data is hard to gather
  - especially for cheap manipulators with limited sensing and durability
- Good policy initialization (e.g. imitation methods) requires experts
- High quality system models known *a priori* don’t work in complex environments

- Need to use *data-efficient* reinforcement learning
  - PILCO — model-based policy search method

Prior Work

- Closing the Learning-Planning Loop with Predictive State Representation [RSS 2010]
  - learning a dynamics model over latent space for value iteration
  - <u>thousands</u> of trajectories to learn model in a <u>discrete</u> domain with <u>no uncertainty</u> reasoning
- Gaussian Processes in Reinforcement Learning [NIPS 2004]
  - uses GPs for modeling dynamics for long-term planning for RL
  - can't deal with state-space constraints like obstacles
- PILCO: A Model-Based and Data-Efficient Approach to Policy Search [ICML 2011]
  - learns probabilistic dynamics model so uncertainty can be used for long-term planning
- Low-cost Accelerometers for Robotic Manipulator Perception [IROS 2010]
  - focused on developing cheap hardware instead of data-efficient decision making

Key Challenge

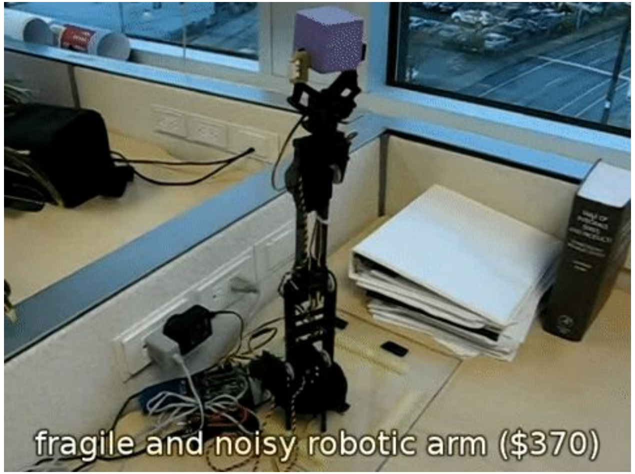
Inexpensive hardware requires:

1) policies robust to imprecision and limited sensing
2) data-efficient learning to derive those policies.

Problem Setting

- Low-cost manipulator + low-cost sensor
  - arm: ~$370
  - depth camera: ~$120
  - total cost: ~$500 << $10,000+
- Task is stacking blocks one at a time
- Block tracking is done via colored blob tracking
  - state space is center of current block
- Only the arm positioning learned, the wrist and grip controls are not learned
  - action space is PWM settings to servos
  - wrist is fixed, gripper opens after 5s (episode length)

- Want to generate policy that minimizes total cost



$$\pi : \mathbb{R}^3 \to \mathbb{R}^4, \mathbf{x} \mapsto \mathbf{u}$$

$$J^\pi = \sum_{t=0}^{T} \mathbb{E}[c(\mathbf{x})]$$

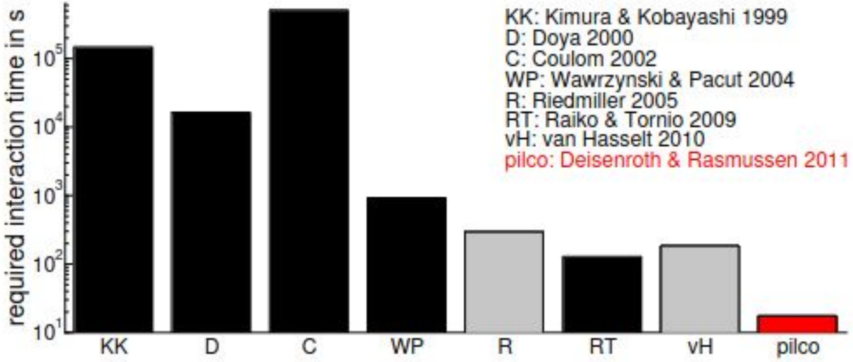
$$\pi^* = \arg\min_\pi J^\pi$$

PILCO Algorithm

Probabilistic Inference for Learning COntrol — previous work by same authors

Algorithm 1 PILCO

init: Set controller parameters ψ to random.
Apply random control signals and record data.
repeat
    Learn probabilistic GP dynamics model using all data
    repeat ▷ Model-based policy search
        Approx. inference for policy evaluation: get $J^\pi(\psi)$
        Gradients $\mathrm{d}J^\pi(\psi)/\mathrm{d}\psi$ for policy improvement
        Update parameters ψ (e.g., CG or L-BFGS).
    until convergence; **return** ψ^*
    Set $\pi^* \leftarrow \pi(\psi^*)$.
    Apply π^* to robot (single trial/episode); record data.
until task learned

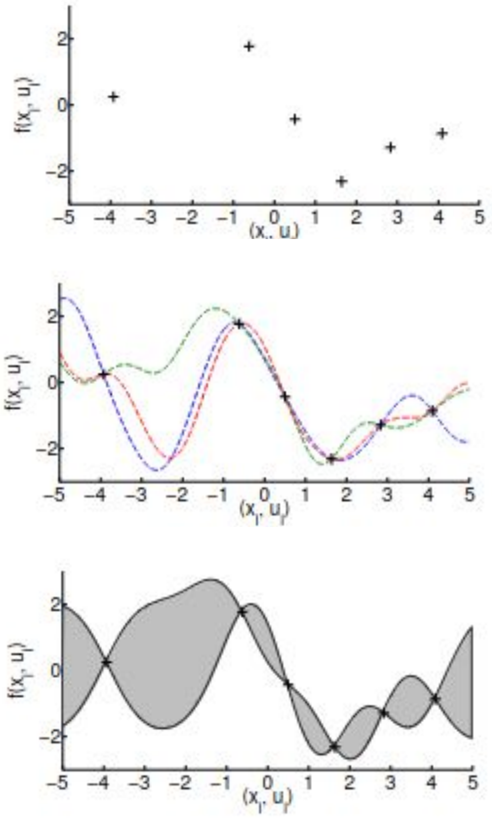


Approach: PILCO Algorithm

Randomly sample control/state space

1. **Learn GP model of dynamics**
2. Estimate cost for policy with new model
3. Update policy based on model + costs
4. Try the new policy

Gaussian Processes (GPs) provide distributions of functions **and the uncertainty**

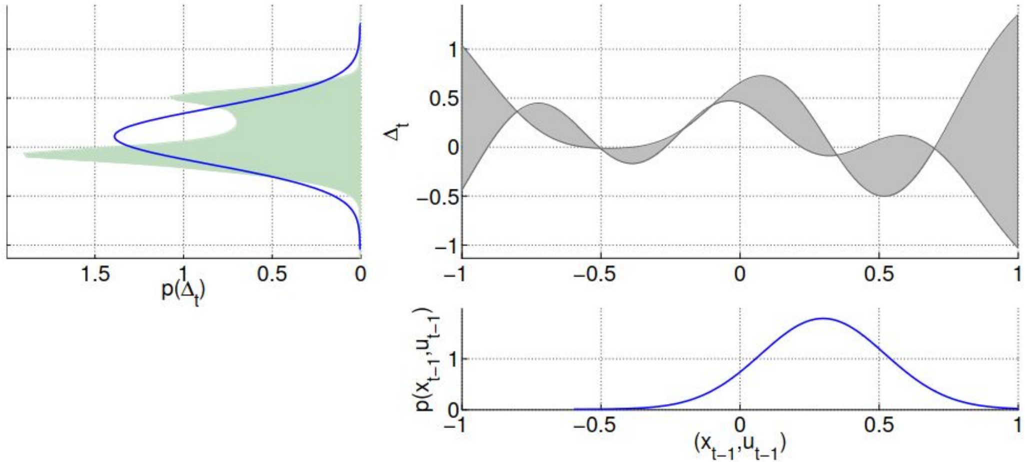


Approach: PILCO Algorithm

Randomly sample control/state space

1. Learn GP model of dynamics
2. **Estimate cost for policy with new model**
3. Update policy based on model + costs
4. Try the new policy

$$J^{\pi} = \sum_{t=0}^{T} \mathbb{E}[c(\mathbf{x})]$$



Approach: PILCO Algorithm

Randomly sample control/state space

1. Learn GP model of dynamics
2. Estimate cost for policy with new model
3. **Update policy based on model + costs**
4. Try the new policy

- Analytically construct gradient of cost with respect to policy parameters
- Construct optimized policy with gradient-based non-convex optimizers

$$\mathbb{E}_{\mathbf{x}_t}[c(\mathbf{x}_t)] = \int c(\mathbf{x}_t)\mathcal{N}\big(\mathbf{x}_t \mid \boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t\big)\,\mathrm{d}\mathbf{x}_t$$

$$\frac{\mathrm{d}\mathcal{E}_t}{\mathrm{d}\psi} = \frac{\partial\mathcal{E}_t}{\partial\boldsymbol{\mu}_t}\frac{\mathrm{d}\boldsymbol{\mu}_t}{\mathrm{d}\psi} + \frac{\partial\mathcal{E}_t}{\partial\boldsymbol{\Sigma}_t}\frac{\mathrm{d}\boldsymbol{\Sigma}_t}{\mathrm{d}\psi}\,.$$

Approach: PILCO Algorithm

Randomly sample control/state space

1. Learn GP model of dynamics
2. Estimate cost for policy with new model
3. Update policy based on model + costs
4. **Try the new policy**

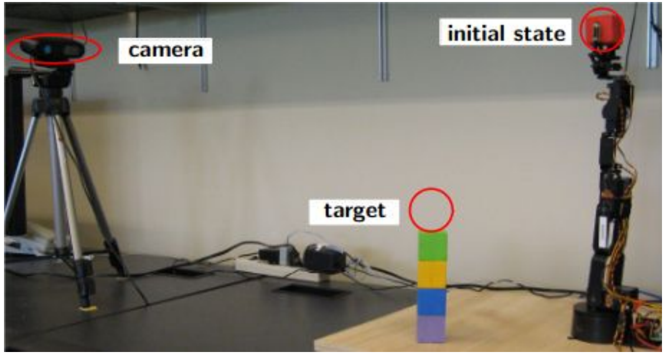
Experimental Setup

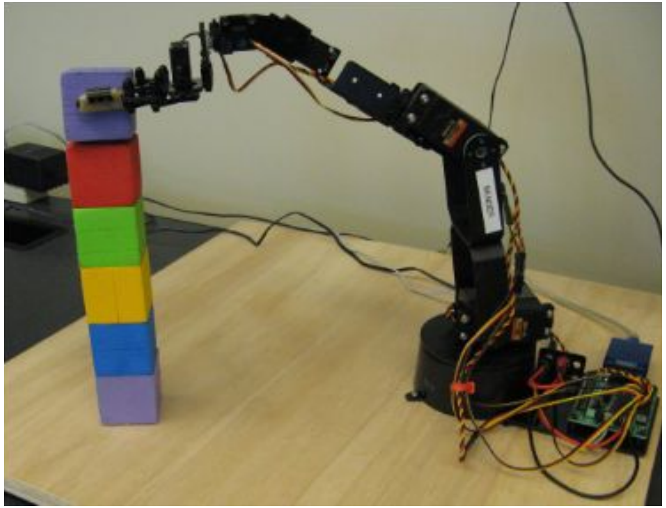
- Block stacking task
- GP model learns dynamics of block in gripper
- Time horizon of 5 seconds
  - claw opens after 5 seconds
- Control parameterized as linear, run at 2 Hz

$$\pi(\mathbf{x}) = \mathbf{u} = \mathbf{A}\mathbf{x} + \mathbf{b}$$

$$c(\mathbf{x}_t) = -\sum_{k=1}^{4} \exp\left(-\frac{1}{2}\frac{\|\mathbf{x}_t - \mathbf{x}_{\text{target}}\|}{\sigma_k^2}\right)$$

$$\sigma = \{b/4, b/2, b, 2b\}$$

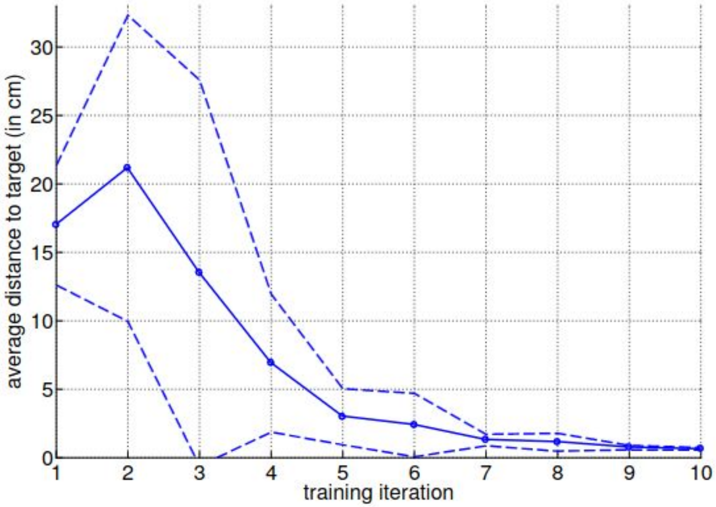




Independent Controllers

- New controller learned for stacking each block
- Care about minimizing total training time

- Overall ~50s per controller
    - ~230 seconds total to learn to stack blocks from scratch



Sequential Transfer Learning

- Reusing controller and dynamics model from the previous block
- Resulted in much faster task completion (90s vs 230s)
  - usually only required 2 more trials per block to achieve equal performance to independent controller

TRANSFER LEARNING GAINS (SETUP 1).

	B2	B2–B3	B2–B4	B2–B5	B2–B6
trials (seconds) independent controllers	10 (50)	19 (95)	28 (140)	37 (185)	46 (230)
trials (seconds) sequential controllers	10 (50)	12 (60)	14 (70)	16 (80)	**18 (90)**
speedup (independent/sequential)	1	1.58	2	2.31	2.56

Sequential Transfer Learning Consistency

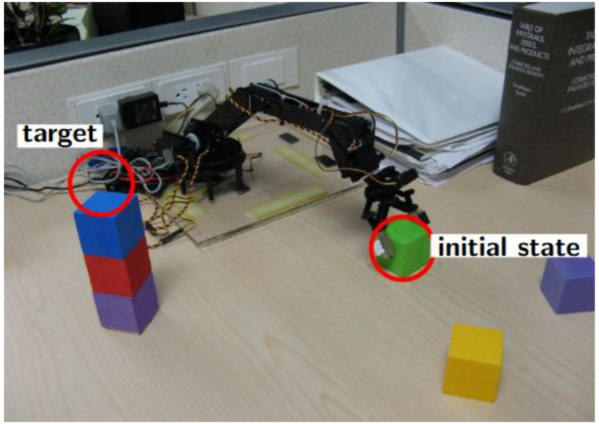
- 4 different random initializations, 10 learning trials, 10 test trials
- Most failures caused by knocking off the top block on existing stack
  - B4 Independent controller didn't learn enough in the 10 learning trials
- Deposit failure is not given to model, but still learns good deposit rate
  - High precision control is difficult because the arm is very jerky

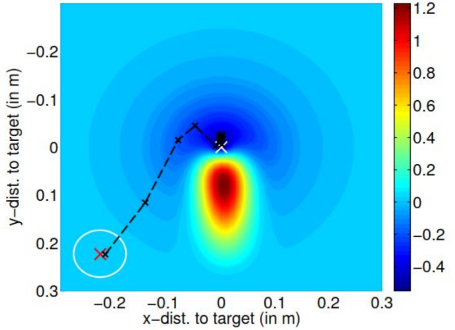
AVERAGE BLOCK DEPOSIT SUCCESS IN 10 TEST TRIALS AND FOUR DIFFERENT (RANDOM) LEARNING INITIALIZATIONS (SETUP 1).

	B2	B3	B4	B5	B6
independent controllers	92.5%	80%	42.5%	96%	100%
sequential controllers	92.5%	87.5%	82.5%	95%	95%

Collision Avoidance

- Initial position changed to be below target
- Cost function includes penalty for going close to other blocks
- Still want to learn control policies quickly, but now want to minimize collisions
    - Including collisions during training!





(b) Two-dimensional slice through the cost function with obstacles encoded.

Collision Avoidance

- Using independent controller setup (no transfer learning)
- Single random trial + 10 training trials

- Learns much safer controller compared to not using obstacle costs
- Safer controller has similar distance error but better deposit rate

EXPERIMENTAL RESULTS FOR PLANNING WITH AND WITHOUT COLLISION AVOIDANCE (SETUP 2).

without collision avoidance	B2	B3	B4	B5	B6
collisions during training	12/40 (30%)	11/40 (27.5%)	13/40 (32.5%)	18/40 (45%)	21/40 (52.5%)
block deposit success rate	50%	43%	37%	47%	33%
distance (in cm) to target at time T	1.39 ± 0.81	0.73 ± 0.36	0.65 ± 0.35	0.71 ± 0.46	0.59 ± 0.34
with collision avoidance	B2	B3	B4	B5	B6
collisions during training	0/40 (0%)	2/40 (5%)	1/40 (2.5%)	3/40 (7.5%)	1/40 (2.5%)
block deposit success rate	90%	97%	90%	70%	97%
distance (in cm) to target at time T	0.89 ± 0.80	0.65 ± 0.33	0.67 ± 0.46	0.80 ± 0.37	1.34 ± 0.56

Limitations + Future Work

- PILCO uses analytic gradients of J^{π} which restricts cost functions
  - could avoid using sampling-based methods of gradient estimation
- Training was not real-time: took 1-3 mins to learn a policy for a given dynamics model, difficult to adapt to new tasks/state space changes on the fly
- Becomes computationally inefficient when the state space is high dimensional
  - Authors had some success with using sparse GPs, but they don't work as well with very complicated dynamics or high frequency sampling
  - Could try modeling dynamics model with neural networks that predict state + covariance instead of GPs

Extended Readings

- PILCO: A Model-Based and Data-Efficient Approach to Policy Search [ICML 2011]
- Gaussian Processes for Machine Learning [MIT Press, 2006]
- Improving PILCO with Bayesian Neural Network Dynamics Models [ICML 2016]
- When to Trust Your Model: Model-Based Policy Optimization [NIPS 2019]

Discussion Questions

Do modern policy generation techniques need to be changed for use on inexpensive (i.e. less precise and lower feedback) hardware?

Is data-efficient learning from scratch the correct problem to solve? Is it easier than solving the sim2real problem and/or making better simulations?

Should the robotics research community have more of a focus on lower-end, more accessible hardware?