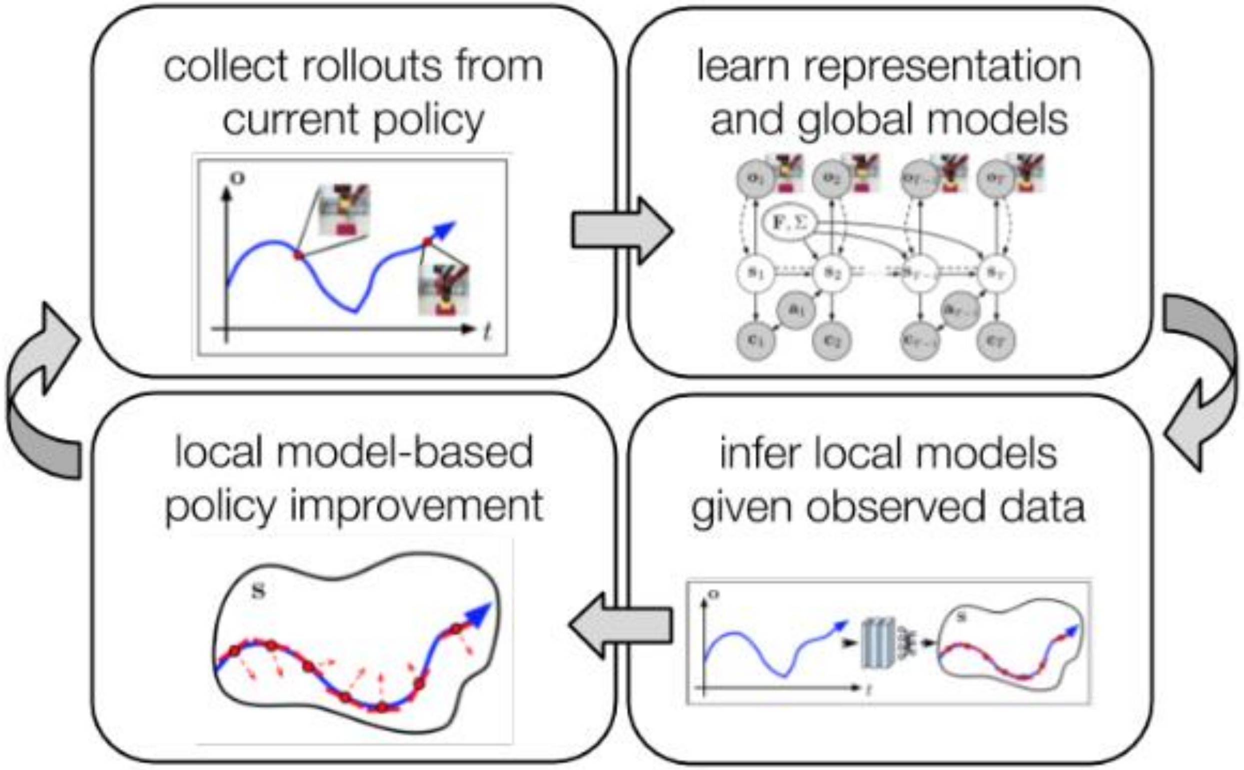
SOLAR: Deep Structured Representations For Model-based Reinforcement Learning

Marvin Zhang*, Sharad Vikram*, Laura Smith, Pieter Abbeel,
Matthew J Johnson, Sergey Levine

11/27/2021



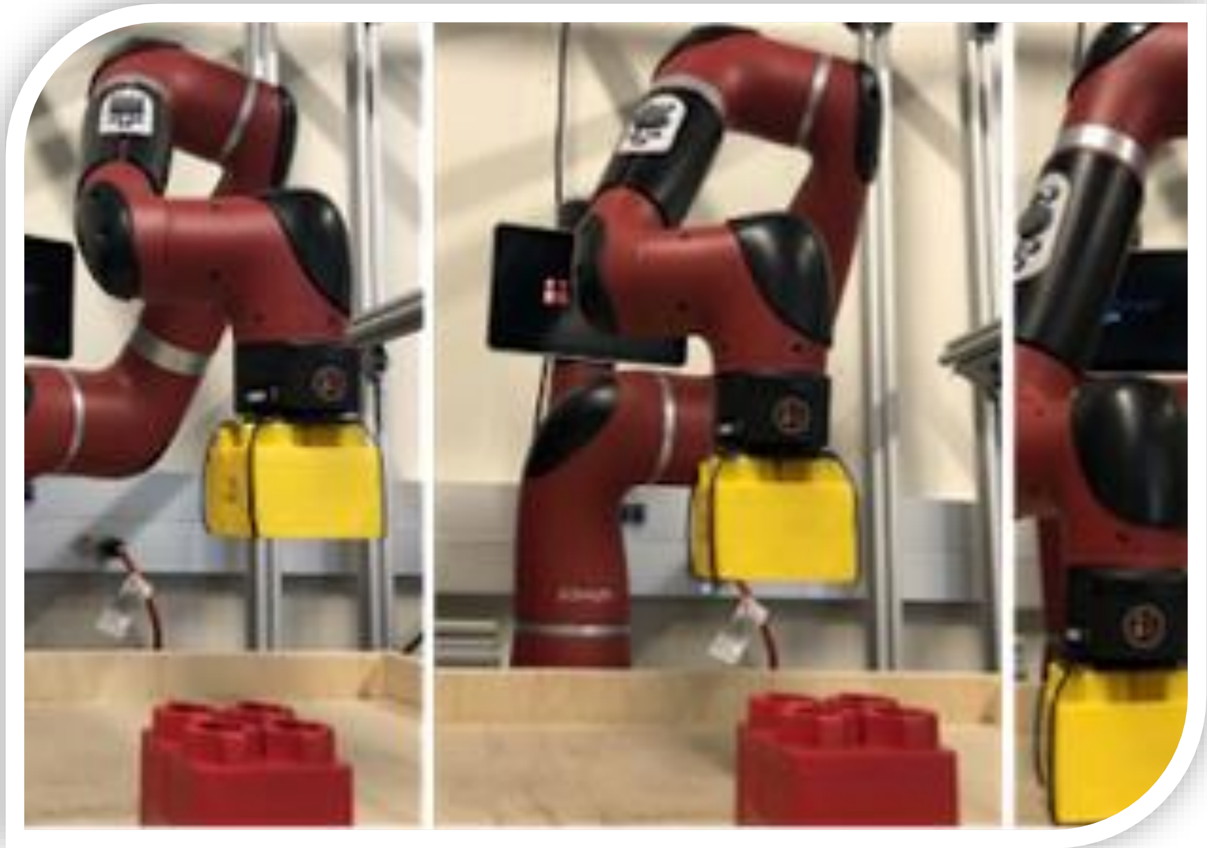
Steven Lawrence,
David R. Cheriton School of Computer Science

UNIVERSITY OF WATERLOO | FACULTY OF MATHEMATICS

Introduction

- RL established itself in simulation by surpassing human abilities.
- Real-world applications hindered by scalability.
- Proposed scaling solutions:
  - World/Global model-based RL
  - Unsupervised meta-learning
  - Representation learning

What is the problem?



- In simulation RL agents have access to low level state information.
- Unrealistic expectation for real-world scenarios.
- How to solve Partially-Observable MDPs from complex RGB images only?
- SOLAR provides a solution for such a scenario.

Proposed solution?

- SOLAR: Stochastic optimal control with latent representations
- Efficiently learns policies directly from raw high-dimensional image observations.
- How?
  - Model-based RL
  - Builds a global model through environment interaction
  - Representation Learning
  - Uses its learned representation and global model to make inferences to explain local model observations.
  - LQR-FLM policy optimization

BEFORE WE DIVE IN...

Some Background

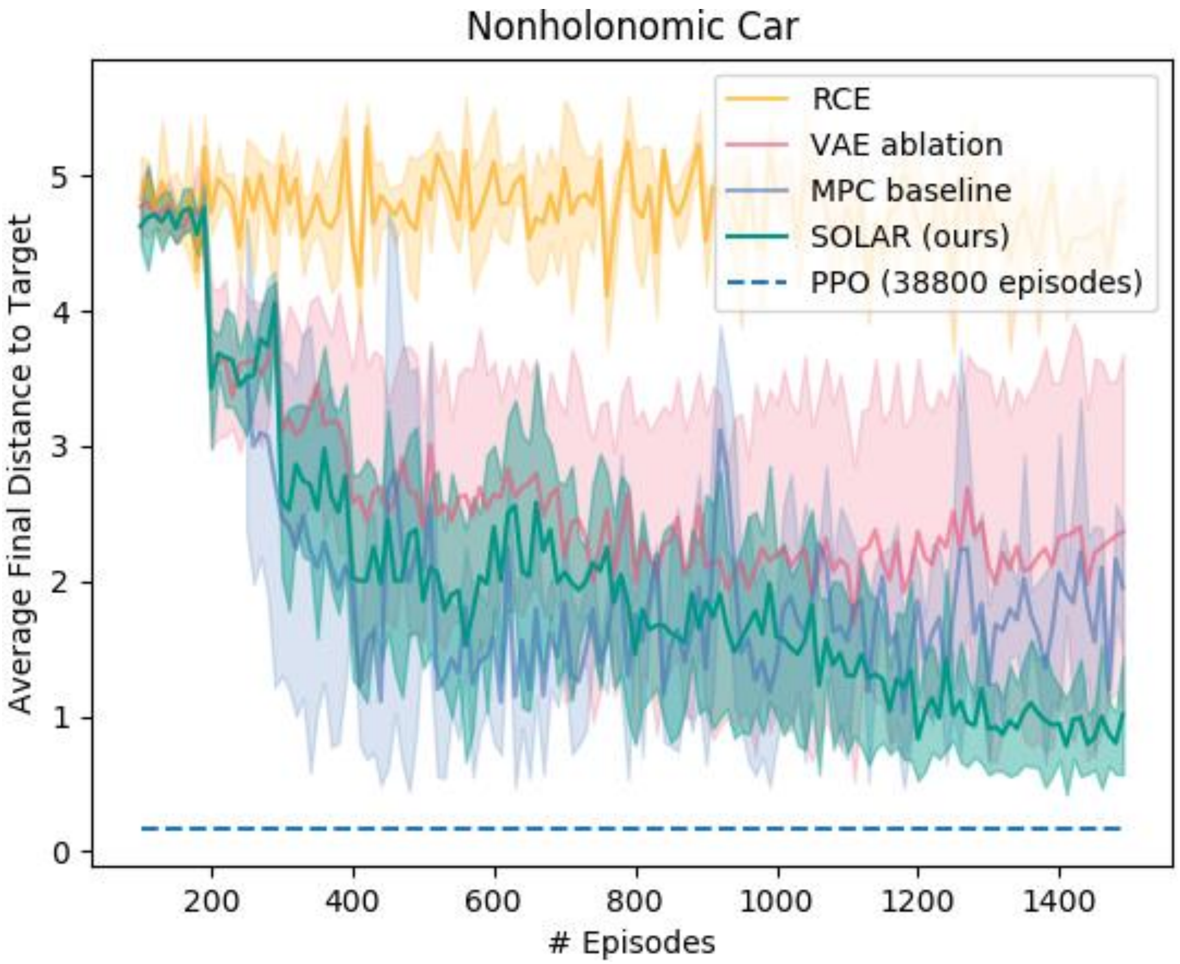
Model-Based Reinforcement Learning

Advantages

- Significantly more efficient that model-free agents.
    - Model-free requires 2-3 orders of magnitude more samples

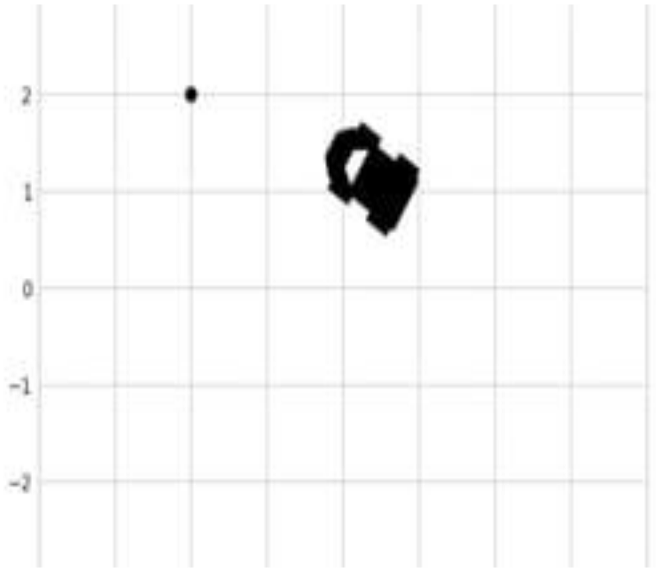
Disadvantages

- Can be outperformed by model-free agents
- Modeling Bias: an imperfect model can result in poor real-world performance.

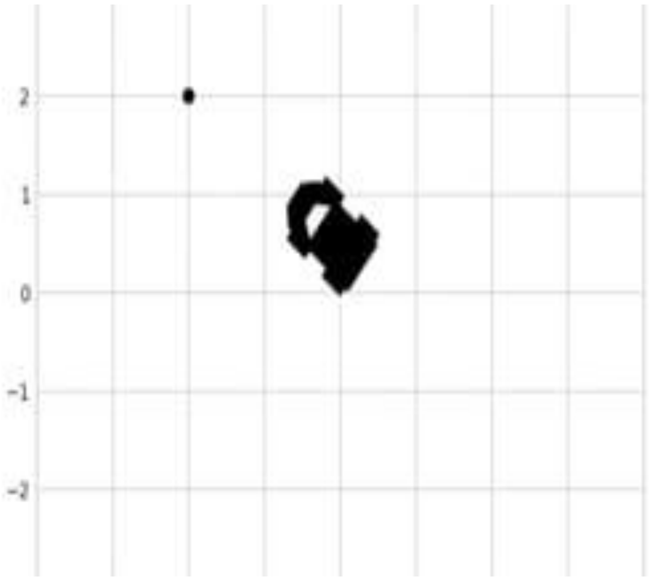


Model-Based Reinforcement Learning – Modeling Bias

SOLAR



PPO



Representation Learning

- Allows for automatic feature detection and classification from raw input.
- Allows an agent to learn and use new features to complete tasks.
- SOLAR will optimize its learned latent representation to infer local models for policy optimization.

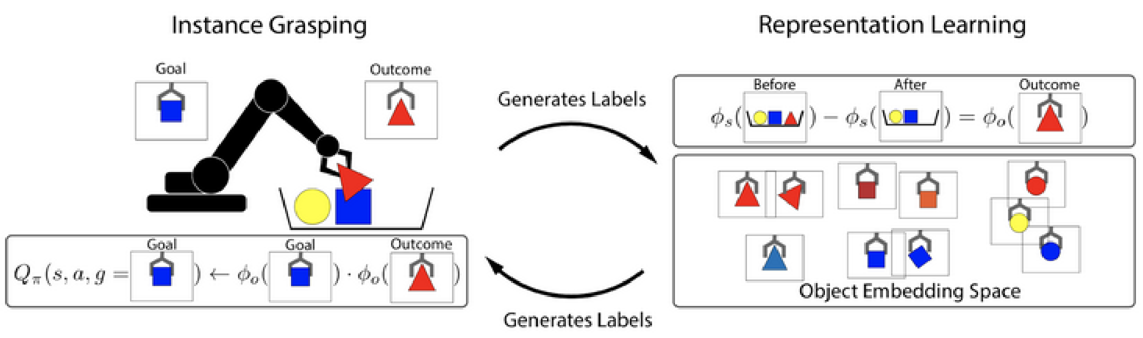


Fig. 18. A conceptual illustration of how grasp2vec learns an object-centric state embedding. (Image source: Jang & Devin et al., 2018)

Linear-Quadratic Regulator with Fitted Linear Models (LQR-FLM)

- What is LQR?
  - LQR is a popular solution for optimizing LQ problems in control theory.
- What is FLM and why does SOLAR use it?
  - LQR will compute a policy that is not globally correct, resulting in a poor real-world policy.
  - LQR computes policies that are very close to the distribution of the data collection policy.
  - Fitted Linear Models (FLM) imposes a KL-Divergence constraint
    - Allows penalization for deviation from the previous policy,
    - and cost optimization.

LET'S DIVE IN...

SOLAR Algorithm

Overview

- Algorithm may be broken down into 3 parts:
  - Lines 1-3: Pre-training where representation and global-model is learned.
  - Line 5: Inference and RL in the Latent Space
  - Line 6: Perform policy update with inferred dynamics.
- Optional:
  - Line 8: Update model with data collected from updated policy (line 7)

Algorithm 1 SOLAR

Input: # iterations K; # trajectories N_{init}, N
Input: model and policy hyperparameters $\xi_{\mathcal{M}}, \xi_\pi$
Output: final model $\mathcal{M}$, final policy $\pi^{(K)}$

1: $\pi^{(0)} \leftarrow \textsc{InitializePolicy}(\xi_\pi)$
2: $\mathcal{D} \leftarrow \textsc{CollectData}(N_{\text{init}}, \pi^{(0)})$
3: $\mathcal{M} \leftarrow \textsc{TrainModel}(\mathcal{D}, \xi_{\mathcal{M}})$ (Section 3)
4: **for** iteration $k \in \{1, \ldots, K\}$ **do**
5: $\quad \{\mathbf{F}_t, \Sigma_t\}_t \leftarrow \textsc{InferDynamics}(\mathcal{D}, \mathcal{M})$ (Section 4)
6: $\quad \pi^{(k)} \leftarrow \text{LQR-FLM}(\pi^{(k-1)}, \{\mathbf{F}_t, \Sigma_t\}_t, \mathcal{M})$ (Section 2)
7: $\quad \mathcal{D} \leftarrow \textsc{CollectData}(N, \pi^{(k)})$
8: $\quad$ (optional) $\mathcal{M} \leftarrow \textsc{TrainModel}(\mathcal{D}, \xi_{\mathcal{M}})$
9: **end for**

Representation and Global Model Learning

- Provides a starting point for local model inferences.
- Objective is to learn two posterior distributions:
  - Over dynamics parameters,
  - and over latent trajectories

Algorithm 1 SOLAR

Input: # iterations K; # trajectories N_{init}, N
Input: model and policy hyperparameters $\xi_{\mathcal{M}}, \xi_{\pi}$
Output: final model $\mathcal{M}$, final policy $\pi^{(K)}$

$\pi^{(0)} \leftarrow \text{INITIALIZEPOLICY}(\xi_{\pi})$
$\mathcal{D} \leftarrow \text{COLLECTDATA}(N_{\text{init}}, \pi^{(0)})$
$\mathcal{M} \leftarrow \text{TRAINMODEL}(\mathcal{D}, \xi_{\mathcal{M}})$ (Section 3)
for iteration $k \in \{1, \ldots, K\}$ **do**
  $\{\mathbf{F}_t, \Sigma_t\}_t \leftarrow \text{INFERDYNAMICS}(\mathcal{D}, \mathcal{M})$ (Section 4)
  $\pi^{(k)} \leftarrow \text{LQR-FLM}(\pi^{(k-1)}, \{\mathbf{F}_t, \Sigma_t\}_t, \mathcal{M})$ (Section 2)
  $\mathcal{D} \leftarrow \text{COLLECTDATA}(N, \pi^{(k)})$
  (optional) $\mathcal{M} \leftarrow \text{TRAINMODEL}(\mathcal{D}, \xi_{\mathcal{M}})$
end for

Inference in the Latent Space and Policy Update

- Alternates between colleting batch data to fit local models and update policy.
- Uses learned representation and global models to enable local model methods.
- Global dynamics model is used to fit local dynamics models.
- Conditioned on the data from the current policy.
- Uses LQR-FLM control to update the policy

Algorithm 1 SOLAR

Input: # iterations K; # trajectories N_{init}, N
Input: model and policy hyperparameters $\xi_{\mathcal{M}}, \xi_\pi$
Output: final model $\mathcal{M}$, final policy $\pi^{(K)}$

1: $\pi^{(0)} \leftarrow$ InitializePolicy(ξ_π)
2: $\mathcal{D} \leftarrow$ CollectData($N_{\text{init}}, \pi^{(0)}$)
3: $\mathcal{M} \leftarrow$ TrainModel($\mathcal{D}, \xi_{\mathcal{M}}$) (Section 3)
4: **for** iteration $k \in \{1, \ldots, K\}$ **do**
5: $\{\mathbf{F}_t, \Sigma_t\}_t \leftarrow$ InferDynamics($\mathcal{D}, \mathcal{M}$) (Section 4)
6: $\pi^{(k)} \leftarrow$ LQR-FLM($\pi^{(k-1)}, \{\mathbf{F}_t, \Sigma_t\}_t, \mathcal{M}$) (Section 2)
7: $\mathcal{D} \leftarrow$ CollectData($N, \pi^{(k)}$)
8: (optional) $\mathcal{M} \leftarrow$ TrainModel($\mathcal{D}, \xi_{\mathcal{M}}$)
9: **end for**

Data Collection and Model Update

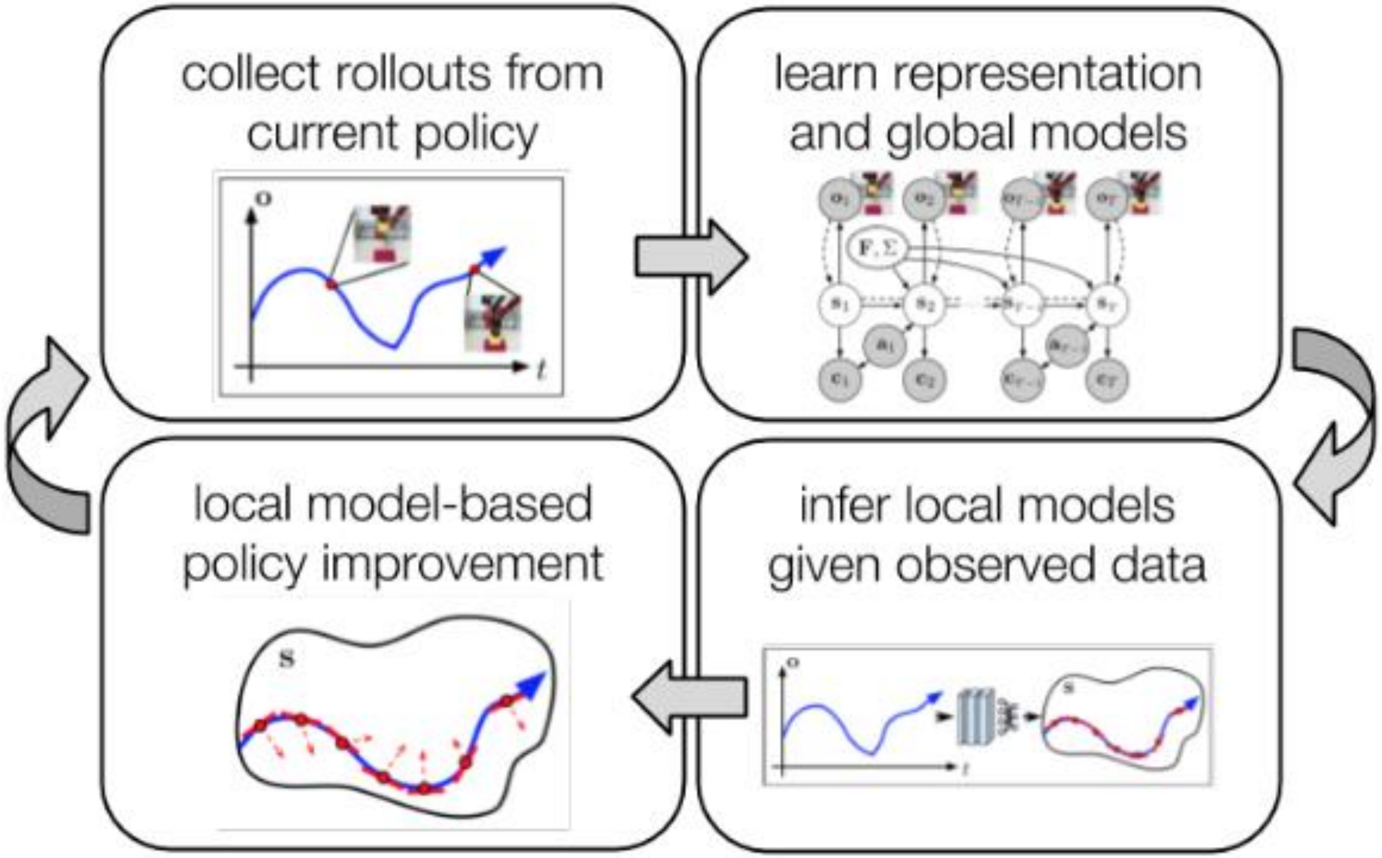
- New policy is used to collect batch data
- Optionally, can update the trained model with the new data.

Algorithm 1 SOLAR

Input: # iterations K; # trajectories N_{init}, N
Input: model and policy hyperparameters $\xi_{\mathcal{M}}, \xi_{\pi}$
Output: final model $\mathcal{M}$, final policy $\pi^{(K)}$

$\pi^{(0)} \leftarrow \textsc{InitializePolicy}(\xi_\pi)$
$\mathcal{D} \leftarrow \textsc{CollectData}(N_{\text{init}}, \pi^{(0)})$
$\mathcal{M} \leftarrow \textsc{TrainModel}(\mathcal{D}, \xi_{\mathcal{M}})$ (Section 3)
for iteration $k \in \{1, \ldots, K\}$ **do**
  $\{\mathbf{F}_t, \Sigma_t\}_t \leftarrow \textsc{InferDynamics}(\mathcal{D}, \mathcal{M})$ (Section 4)
  $\pi^{(k)} \leftarrow \text{LQR-FLM}(\pi^{(k-1)}, \{\mathbf{F}_t, \Sigma_t\}_t, \mathcal{M})$ (Section 2)
  $\mathcal{D} \leftarrow \textsc{CollectData}(N, \pi^{(k)})$
  (optional) $\mathcal{M} \leftarrow \textsc{TrainModel}(\mathcal{D}, \xi_{\mathcal{M}})$
end for

High-level overview of SOLAR alogorithm

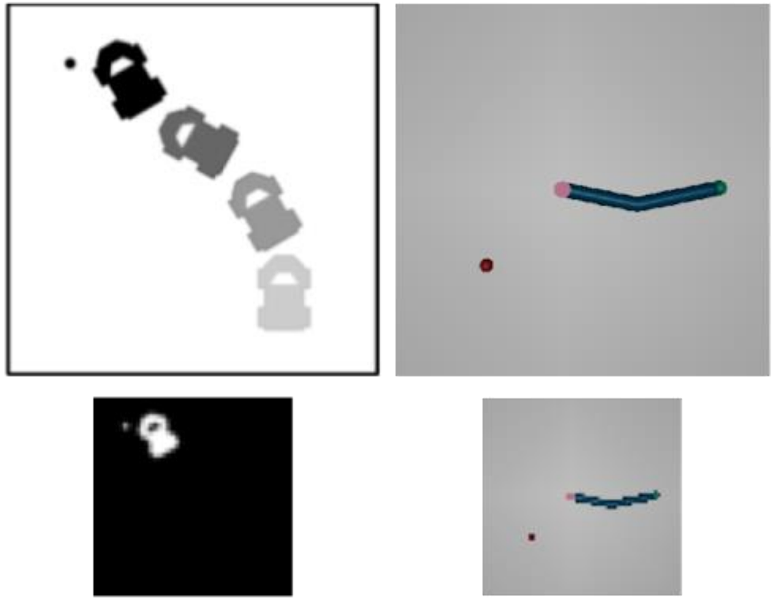


HOW DOES IT PERFORM?

Experiments & Results

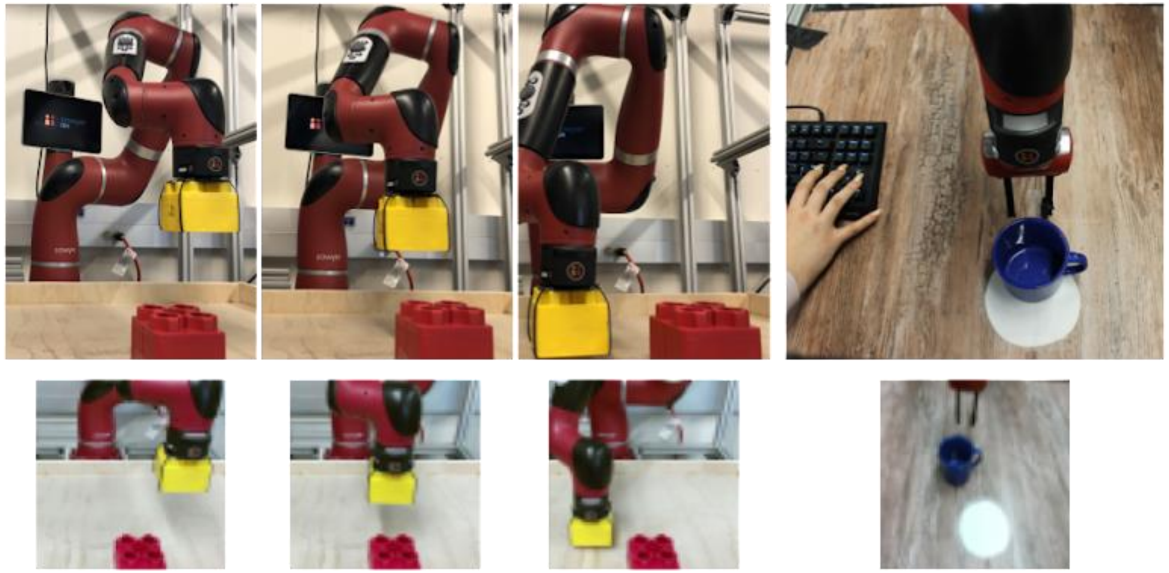
Experiments

- Simulation:
  - Nonholonomic Car
    - Starts bottom right and needs to reach goal in the top left.
    - Controls its acceleration and steering velocity
    - 64x64 images as state observations
  - Reacher
    - 2-DoF arm has to reach a fixed target
    - 64x64 images as state observations



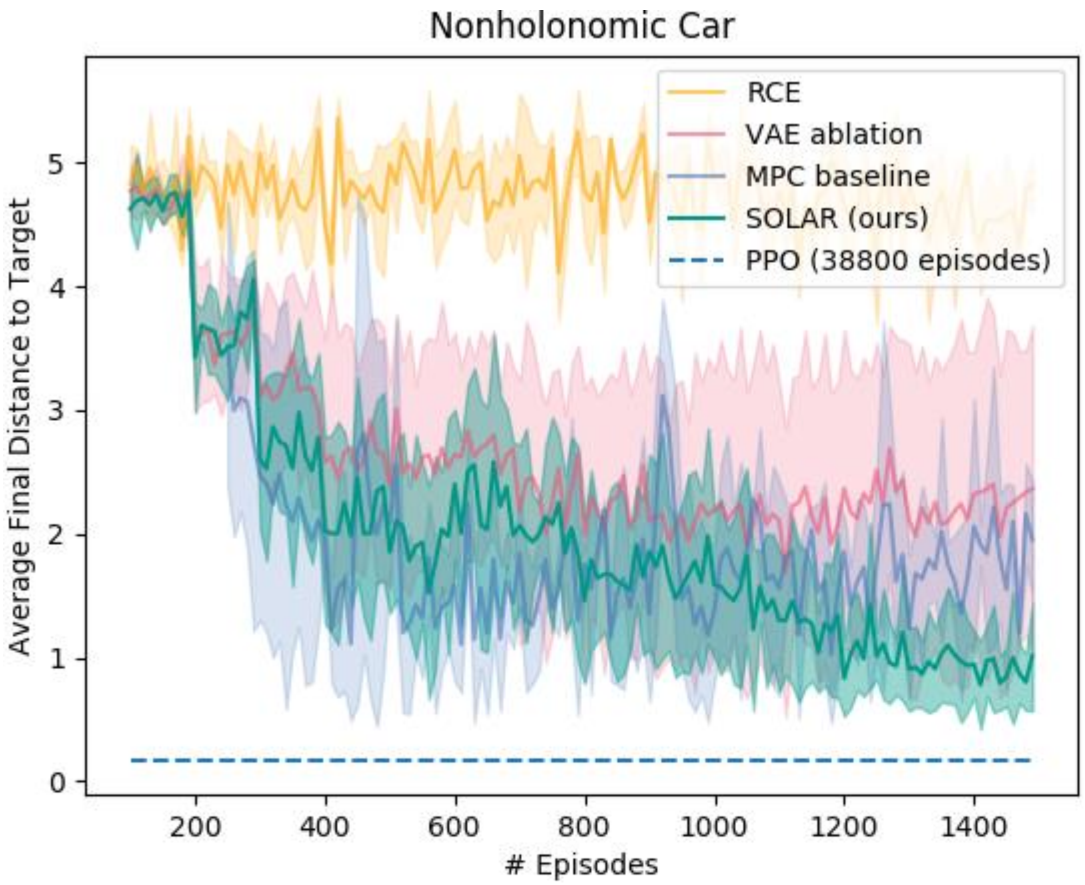
Experiments

- Real-World Robot Task:
  - Sawyer Lego block stacking (2 variations)
    - 7-DoF Sawyer robotic arm to stack Lego blocks
    - 64x64x3 images as state observations with no auxiliary information
  - Sawyer pushing
    - Objective is to push a mug onto a white coaster
    - Used only sparse binary rewards
    - 64x64x3 images as state observations with no auxiliary information



Results - Nonholonomic Car

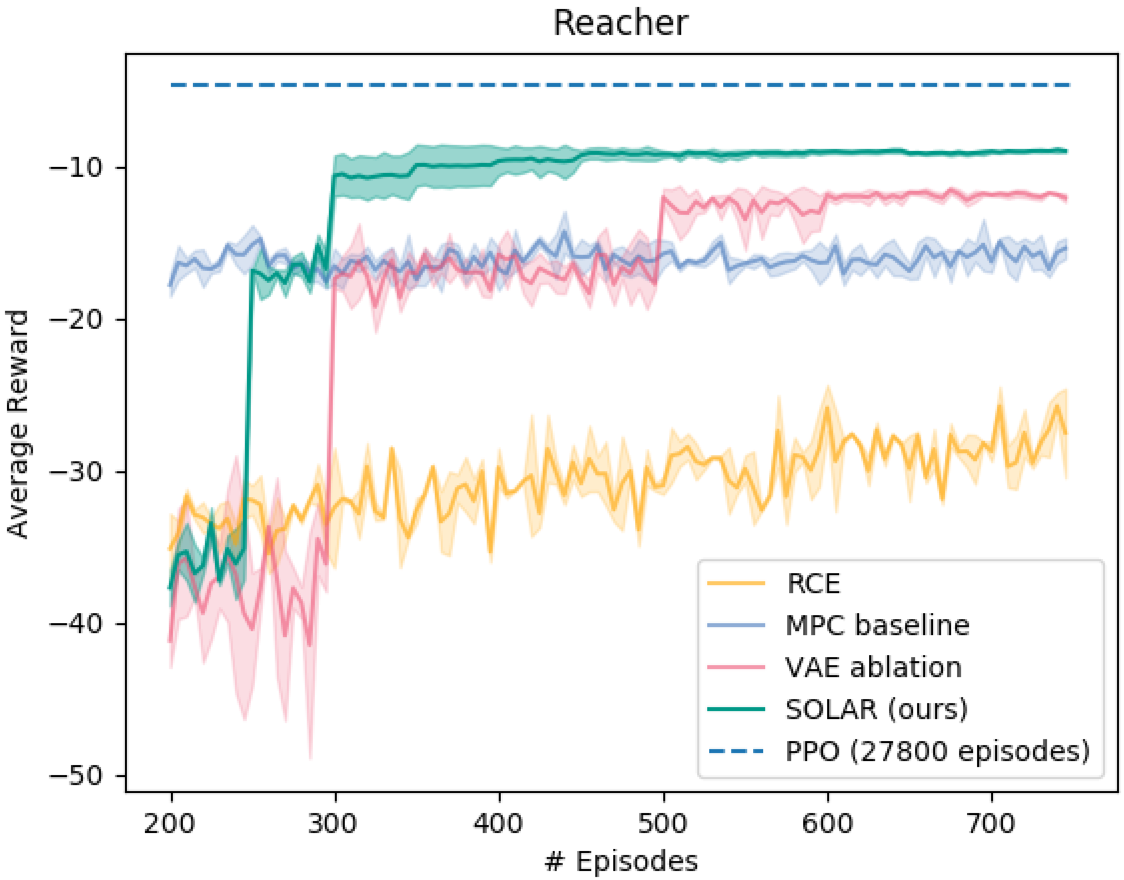
- SOLAR and the MPC baseline learn with about 1500 episodes
- PPO outperforms SOLAR but required 25x more data.
- The model based agents experience jittery results from model bias.



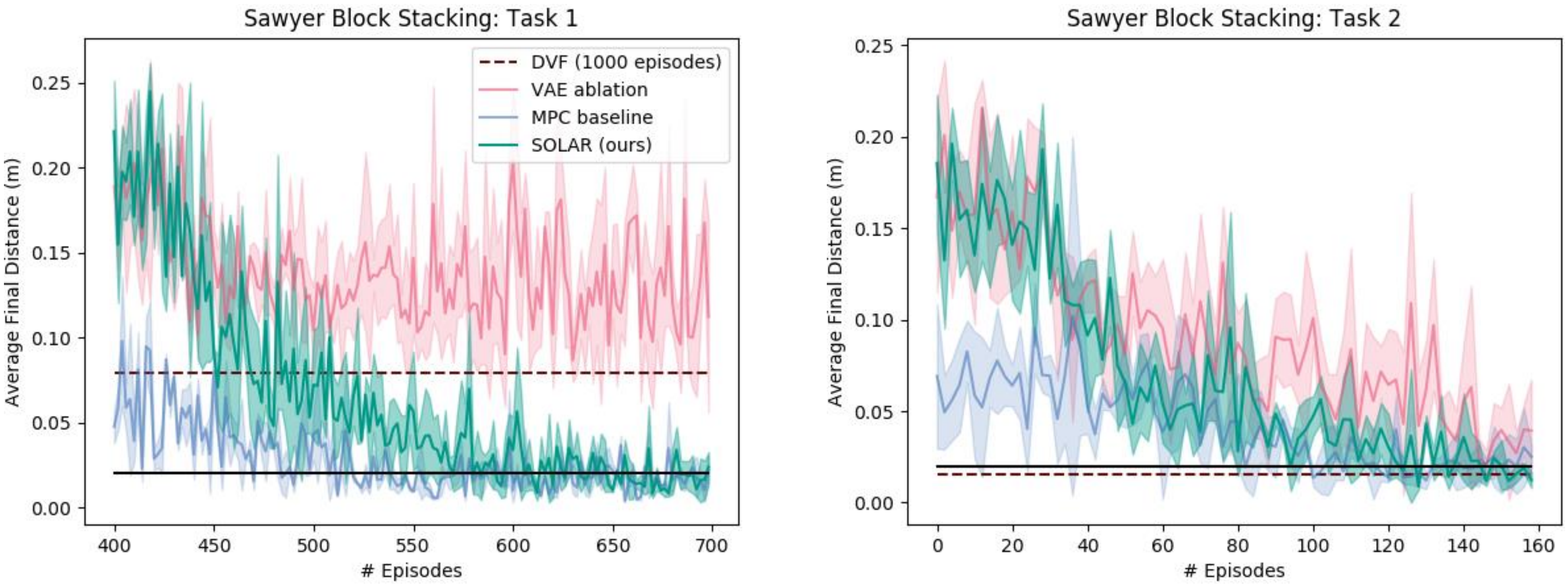
UNIVERSITY OF WATERLOO | FACULTY OF MATHEMATICS

Results - Reacher

- SOLAR performs significantly better than the other model based agents.
- PPO outperforms SOLAR but requires 40x more data.
- MPC has better initial behavior but can not perform long horizon planning, whereas SOLAR can.



Results - Lego Stack



Results – Sawyer Push

- Using only sparse rewards, SOLAR learns in 1 hour of interaction time.
- DVF performs worse with more time and using a smaller model.
- Highlighting the efficiency of SOLAR

Method	Final Distance to Goal (cm)	Episodes per Seed
DVF (Ebert et al., 2018)	4.50 ± 2.60	280
SOLAR (ours)	1.85 ± 0.86	250

Table 1. Sawyer Pushing with Sparse Rewards

Conclusion

- SOLAR:
  - Efficiently learns real-world robot tasks.
  - Uses raw high-dimensional image observations.
  - Outperforms other model-based algorithms including SoTA DVF.
  - Significantly more efficient that model-free algorithms with comparable results.

UNIVERSITY OF
WATERLOO

FACULTY OF MATHEMATICS