Real Robot Challenge – Phase 2
A Planning Approach with Learned Ensemble Models

Anonymous

Abstract

For the real robot challenge, we combine a fast zero-order optimization method (iCEM) with model learning.Our iCEM method generates strong expert trajectories using a learned forward dynamics model, which is iteratively trained on newly collected data. We use parallel forward model ensembles and use their uncertainty as a cost penalty in the optimization. Through the tight coupling between all components, our algorithm runs in real-time, rendering it suitable for real robotic applications.

TEAM NAME:
ALpha-Cube

1 Introduction

In the simulation phase, we had access to the ground truth simulation which we were using for our fast zero-order trajectory optimization (iCEM).
On the real system, we can not afford to use the ground truth simulator for planning or to use a lot of samples in the optimization loop because of the strict real-time requirement; therefore, we add model learning for fast planning.

2 Methods

2.1 Zero-order trajectory optimization

One key component of our solution is our strong and fast zero-order optimization algorithm iCEM [6] that is based on the Cross-Entropy Method (CEM) [8]. We propose an improved version of the CEM algorithm for fast planning, with novel additions including temporally-correlated actions and memory, requiring 2.7-22× less samples and yielding a performance increase of 1.2-10× in high-dimensional control problems compared to a naive implementation of CEM. For further details, please see Pinneri et al. [6].

2.2 Simulations and Model learning

There are several ways to use the simulations in order to get a well performing system on the real hardware.

A common approach is to optimize/learn a policy or model in simulations and then transfer them to the real environment. To make this successful, Domain Randomization [5] in simulations have shown to be a powerful concept. Since the simulator contains the causal model, we can intervene on various variables from this model in order to generate different environments, or different data distributions for model fitting, reducing the distribution shift when going to the real system.

Another option is to use the simulations as internal models directly and use them for planning. However, very high-performance physics simulations would be required to make that work, which are not available for the complexity of environments used here.

A third option is to use the simulations only as initialization / structural bias in model learning. The real data and the simulated data will have a very similar structure, but details are different. So fitting a model to simulated data (potentially with domain-randomization) should serve as a good starting point. A fine-tuned model is expected to have improved generalization capabilities. In our approach we follow this path.

A related idea is to make fast adjustments to the model using real observations in real-time similar to the universal successor features framework in RL [4].

2.3 Model architecture

The learned models need to have two main properties: they need to predict the dynamics well and they have to be fast enough to be used for planning.

We are considering ensembles of feed-forward neural networks to start with. To improve the quality of prediction we want to incorporate sequence models with the attention mechanism. More concretely, the Transformer architecture [9] has shown to perform well on prediction tasks of long sequences. The attention mechanism of the architecture can help better model dynamics discontinuities such as points of contact between the fingers and the cube.

We also consider to use the fact that the position of the fingers are independent from the position of the cube except in the case of interaction. This might allow us to separate the dynamics prediction of the fingers and the cube.

2.4 Policy extraction

To decrease the planning time, we are considering to learn a policy next to the forward model that can be used for warm-starting and guiding [2,3] the optimization loop. There exists a number of methods for extracting policies from an offline or off-policy collected dataset including pure behavioral cloning (BC) and DAgger [7]. Recently, offline RL methods, e.g. conservative Q learning [1], showed some promising results in that domain which let us belief that those methods are promising candidates for learning strong policies in our setting.

3 Current Approach

We have collected trajectories from the real system with random policies. These trajectories have been used to fit an ensemble of MLPs(Multi-Layer Perceptrons) as a forward-model for use in conjunction with the iCEM. The ensemble of MLPs are used to model a multi-variate normal distribution over the next step prediction, more concretely, an ensemble of K MLPs, with $\Theta = \{\theta_i | i \in [0, K-1]\}$ denoting the set of their parameters, we construct the normal distribution by taking the mean and standard deviation over all of their predictions:

$$f_\Theta(\vec{o_t}, \vec{a_t}) = \mathcal{N}(\vec{\mu}_\Theta(\vec{o_t}, \vec{a_t}), \Sigma_\Theta(\vec{o_t}, \vec{a_t})) \tag{1}$$

The one-step forward model is used in a recurrent fashion at evaluation time to generate candidate trajectories in the iCEM. We use the mean prediction of the ensemble models as the next observation estimate.

Further, we assume independence between the output dimensions, therefore the covariance matrix Σ is a diagonal matrix. We use an uncertainty penalty in the form if the variance of the forward model predictions to augment the cost of the iCEM. The cost can be written as follows ($\vec{b}$ $\vec{g}$ denoting the vector of the current and goal box position and orientation, respectively):

$$c(\vec{o_t}, \vec{b}_{t+1}, \vec{g}, f_\Theta) = ||\vec{b}_{t+1} - \vec{g}||_2 + \alpha \sqrt{\mathrm{Tr}(\Sigma_\Theta(o_t, a_t))} \tag{2}$$

where Tr is the trace (here sum of individual variances). The uncertainty penalty term ensures that the iCEM is going to prefer candidate trajectories where the models are more likely to make a good prediction and is controlled by the hyperparameter α.

We opted for position control as a means of controlling the system.

3.1 Performance

We efficiently parallelized the computation of the ensemble in the form of stacked matrix operations in order to achieve low prediction latency. With a model architecture of 6 ensembles with 3 hidden layers of size 256 and iCEM controller of horizon 15 and 128 sampled trajectories in 2 rounds we achieve 10 Hz control frequency on the real system. Using a GPU and better optimized code (currently Python) would allow a much higher control frequency.

Due to technical problems we were not able to submit our code early enough to be fully evaluated in the different tasks. Also we did not iterate the learning of the forward model yet, due to time constraints. We expect a strong increase in performance, once we do that.

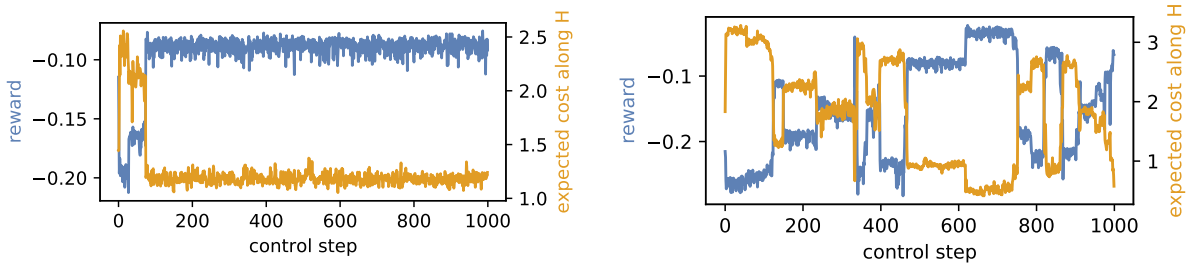

Figure 1: Reward returned by the platform (blue) vs expected cost along planning horizon H (orange). On the x-axis control steps are plotted. The real systems runs asynchronously with 1kHz while here we show the rewards and costs at 10Hz, our control frequency.

Fig. 1 shows for two particular runs the reward that comes from the platform and the expected cost along the planning horizon H predicted by our model. Both are very well aligned, indicating that the learned forward model has a clear understanding of the dynamics of the environment. On the other hand, the plot on the right-hand side is also indicative for the fact that the learned models and their integration with iCEM need further attention.

4 Next Steps

We have several aspects we want to address next:

- Iterate the training of the forward model from newly collected data
- Implement different forward model architectures
- Check the influence of the uncertainty penalty term
- Leveraging independence between box and finger manipulators for generating counterfactual data

References

[1] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *arXiv Pre-print arXiv: 2006.04779*, 2020.

[2] Sergey Levine and Pieter Abbeel. Learning neural network policies with guided policy search under unknown dynamics. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 1071–1079. Curran Associates, Inc., 2014.

[3] Sergey Levine and Vladlen Koltun. Guided policy search. In *International Conference on Machine Learning (ICML)*, 2013.

[4] Chen Ma, Dylan R Ashley, Junfeng Wen, and Yoshua Bengio. Universal successor features for transfer reinforcement learning. *arXiv preprint arXiv:2001.04025*, 2020.

[5] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 1–8. IEEE, 2018.

[6] Cristina Pinneri, Shambhuraj Sawant, Sebastian Blaes, Jan Achterhold, Joerg Stueckler, Michal Rolinek, and Georg Martius. Sample-efficient Cross-Entropy Method for real-time planning. In *Conference on Robot Learning (CORL)*, 2020. (to appear) Preprint: arXiv:2008.06389.

[7] Stéphane Ross and Drew Bagnell. Efficient reductions for imitation learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 661–668, 2010.

[8] Reuven Rubinstein and William Davidson. The cross-entropy method for combinatorial and continuous optimization. *Methodology and Computing in Applied Probability*, 1999.

[9] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.