LEO: Learning Energy-based Models in Factor Graph Optimization

Paloma Sodhi[1,2], Eric Dexheimer[1], Mustafa Mukadam[2], Stuart Anderson[2], Michael Kaess[1]
[1]Carnegie Mellon University, [2] Facebook AI Research

Appendix

A Applying LEO to Differentiable Optimizers

We discuss here how LEO can be applied to differentiable optimization libraries. It requires two simple changes, (a) disable gradients from the inner loop optimizer block, and (b) replace the direct final tracking loss (Loss 1) with an energy-based loss (Loss 3). To illustrate how LEO performs compared to alternate solutions such as unrolling, we choose the uni-dimensional regression task from [1]. This enables us to visualize the learned 2D energy surfaces.

Setup In the regression task, the goal is to learn a network that maps points (x, y) to an energy value, given samples from a ground truth function. The ground truth data is generated from the function $g(x) = x\sin(x)$ for $x \in [0, 2\pi]$. We model $P(y|x) \propto \exp\{-E(\theta, y; x)\}$ where $E(\theta, y; x) = ||f(\theta, y; x)||_2^2$. $E(.)$ is the energy function expressed as a sum-of-squares of $f(\theta, y; x)$, similar to Section 3.1, to apply Gauss-Newton solvers. $f(\theta, y; x)$ is modeled as a neural network with θ as the learnable network parameters.

LEO minimizes an energy-based loss (Loss 3), i.e. $\mathcal{L}(E(\theta, \cdot); y^i_{gt})$, resulting in a similar update rule of the form,

$$\theta^+ \leftarrow \theta - \frac{1}{|\mathcal{D}|}\sum_{i=1}^{|\mathcal{D}|}\left\{\nabla_\theta E(\theta; y^i_{gt}) - \frac{1}{S}\sum_{\hat{y}}\nabla_\theta E(\theta; \hat{y}^i)\right\} \tag{1}$$

We solve for this update using the Adam optimizer in Pytorch. We use a batch Gauss-Newton solver as our underlying inner loop optimizer (LEO GN).

The unrolling baselines minimize a final tracking loss (Loss 1), i.e. $\mathcal{L}(\theta; y_{gt}) = {}^1/_{|\mathcal{D}|}\sum_i ||\hat{y}^i \ominus y^i_{gt}||_2^2$, where $\hat{y} := \underset{y}{\text{argmin}}\ ||f(\theta, y; x)||_2^2$. We use both unrolled gradient descent (Unrolled GD) and unrolled Gauss-Newton (Unrolled GN) [2].

For all methods, we use two initialization schemes for the underlying optimizer. The first scheme initializes from $y = 0$ for all x, as followed in [1]. The second scheme initializes from the ground truth $g(x) = x\sin(x)$ as suggested in [2].

Results Fig. 1 shows the learned energy surfaces for all 3 methods using initialization scheme 1, i.e., initializing the optimizer from zero. At iteration 0, all 3 approaches are initialized with a random energy function. While all 3 approaches end up with a low final loss, only LEO converges to an energy function with a single basin about the ground truth samples. The unrolled baselines converge to energy functions with multiple minima, not all of them on the ground truth samples. This is because the energy function need only be good enough for the optimizer, when initialized from $y = 0$ to reach the ground truth for a fixed number of unrolling steps. It does not guarantee a similar convergence when executed with different number of unrolling steps or initialization from different y values.

Fig. 2 shows the learned energy surfaces for initialization scheme 2, i.e., initializing the optimizer from ground truth. Similar to the previous figure, all 3 methods initialize from random energy functions and converge to a low final loss. But only LEO converges to a similar energy function as before, i.e., one with a single basin around the ground truth. The unrolling baselines converge to

5th Conference on Robot Learning (CoRL 2021), London, UK.

a different energy function than in scheme 1. In this case, the energy function need only be good enough for the optimizer, when initialized from ground-truth, to stay on the ground truth.

For both schemes, LEO alone converges to the same energy function, as it is less sensitive to the underlying optimization process. In future work, we aim to apply LEO on differentiable optimizers for robot estimation tasks. This would require a factor graph optimization library similar to GTSAM but implemented in a differentiable manner. There are recent efforts on this front [2, 3, 4], but there isn't a single, stable open-source library yet. We hope our analysis on this regression task would encourage others in the community to try out LEO in their respective differentiable optimizer libraries.

B Comparison to other samplers

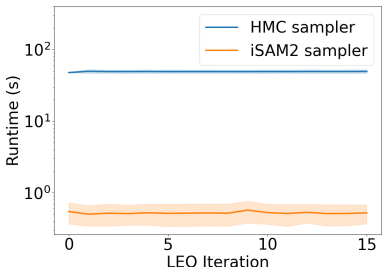

Figure 3: Runtime comparison against an HMC sampler

We compare run times against an HMC sampler implemented in the hamiltorch library [5]. The sampler simulates a set of Hamiltonian differential equations which upon simulating generates a sequence of samples.

We run this on the 2D navigation dataset which have a 900-dimensional state space (3-DOF * 300 steps). The HMC sampler is sampling from the true posterior distribution while the incremental Gauss-Newton, iSAM2 [6], sampler that we use samples from a Gaussian approximation that it maintains online. However, we found similar convergence at train time using both samplers which suggests that the Gaussian approximation was reasonable for our applications.

In terms of run time, we found the HMC sampler run time to be two orders of magnitude greater than the iSAM2 sampler as shown in Fig. 3. This is expected since the HMC sampler evaluates the cost as well the gradients of the cost every time a sample is generated which requires looping through all the factors and can be expensive. In contrast, the iSAM2 sampler allows for directly sampling from the Gaussian that is significantly faster. When used in an online setting, as new factors get added to the graph, the HMC sampler would have to recompute the posterior distribution. On the other hand, the iSAM2 sampler efficiently and incrementally updates the Gaussian approximation by leveraging sparsity.

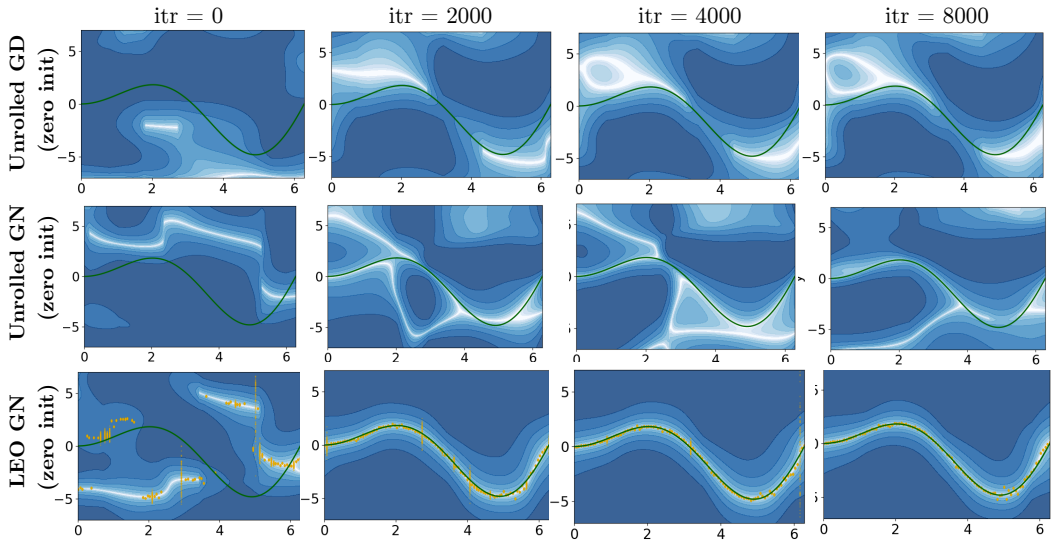

Figure 1: Evolution of learned 2D energy surfaces using **initialization scheme 1**. Contour surfaces show the normalized energy surfaces. Ground truth function $x\sin(x)$ in green, LEO samples in orange. Lighter colors correspond to lower energy.

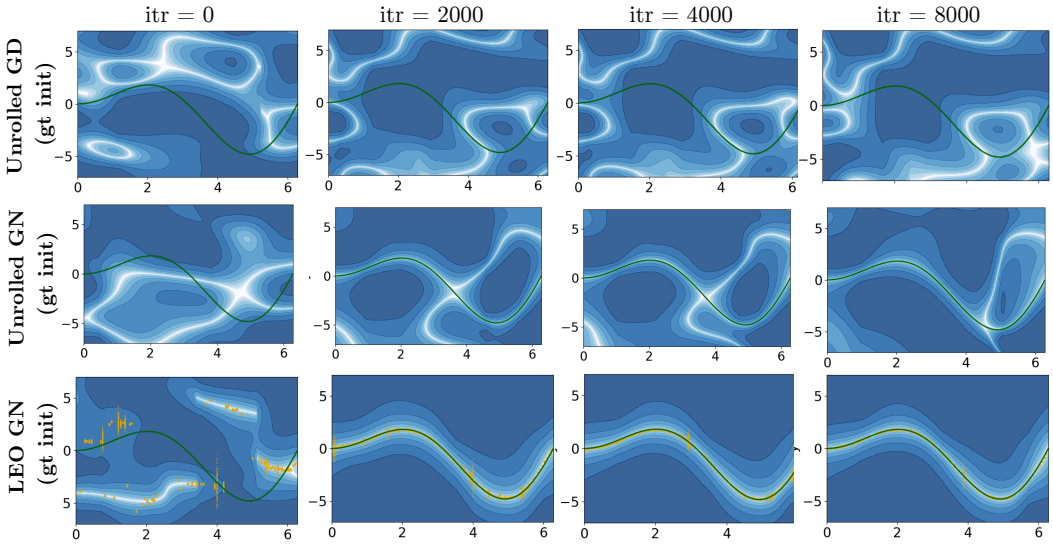

Figure 2: Evolution of learned 2D energy surfaces using **initialization scheme 2**. Contour surfaces show the normalized energy surfaces. Ground truth function $x \sin(x)$ in green, LEO samples in orange. Lighter colors correspond to lower energy.

C Normalized Negative Log-Likelihood Loss: Additional Details

In Section 4.1, we defined the energy-based loss as the normalized negative log-likelihood loss. The motivation for such a loss comes from probabilistic modeling, notably the principle of maximum entropy moment matching.

Principle of Maximum Entropy Assume a ground truth distribution $P_{gt}(x, z)$ that generates x and z. We would like to estimate a distribution $P_\theta(x|z)$ that matches this. The true ground truth distribution is unknown, but we can sample from it. This allows us to estimate $P_\theta(x|z)$ by matching empirical moments computed from these samples (Fig. 4).

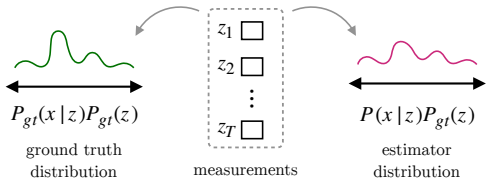

Figure 4: Ground truth distribution is unknown and only observable via measurements $z_{1:t}$. The goal is to estimate a distribution $P(x|z)$ that matches observed moments.

However, there is a continuous space of probability distributions that match these empirical moments. Which of these distributions do we pick? The maximum entropy (MaxEnt) principle suggests we pick the distribution which is least committal, i.e. the one with maximum entropy [7]. We can write out this objective of maximizing entropy of the distribution $P_\theta(x|z)$ subject to matching moment constraints against the unknown ground truth distribution as,

$$\begin{aligned} &\max \; -\sum_x P_\theta(x|z) \log P_\theta(x|z) \\ &s.t. \mathop{\mathrm{E}}_{z\sim P_{gt}(z)} \mathop{\mathrm{E}}_{x\sim P_\theta(x|z)} \Phi(x,z) = \mathop{\mathrm{E}}_{(x,z)\sim P_{gt}(x,z)} \Phi(x,z) \end{aligned} \tag{2}$$

where, $\Phi(\cdot)$ are the moment functions. Applying KKT conditions for Eq. 2 and expanding the dual results in an unconstrained objective of the form [7, 8],

$$\begin{aligned} &\max_\theta \mathop{\mathrm{E}}_{\substack{x,z\sim \\ P_{gt}(x,z)}} \log P_\theta(x|z) \\ &P_\theta(x|z) \propto \exp(-\theta^T \Phi(x,z)) \end{aligned} \tag{3}$$

where, θ are the Lagrange multipliers. The term $\theta^T \Phi(x,z) = E(\theta, x; z)$ can be interpreted as a linear "energy function" such that $P_\theta(x) \propto \exp(-E(\theta, x; z))$ puts a high energy on low probability solutions. This linear class of cost functions can be lifted to a richer class of parametric nonlinear

functions $E(\theta, x; z)$. Note that this may induce a duality gap in optimizing Eq. 2, but is commonly used in practice when parameterizing energy functions as neural networks [9, 10, 11]. The maximum entropy solution can then be written out as,

$$\begin{aligned} &\max_{\theta} \mathop{\mathrm{E}}_{\substack{x,z\sim \\ P_{gt}(x,z)}} \log P_\theta(x|z) \\ &P_\theta(x|z) \propto \exp(-E(\theta, x; z)) \end{aligned} \tag{4}$$

Eq. 4 asks us to maximize the likelihood of ground truth trajectories where the likelihood takes the form of a Boltzmann distribution, where lower energy implies higher probability mass. In other words, we want to find energy function $E(\theta, x; z)$ that places maximal probability mass on the ground truth trajectory. Writing out this objective over a training dataset $(x_{gt}, z) \in \mathcal{D}$ results in the NLL loss that we saw earlier in Section 4.1,

$$\mathcal{L}(\theta) = \frac{1}{|\mathcal{D}|} \sum_{(x^i_{gt}, z^i)\in\mathcal{D}} -\log P_\theta(x^i_{gt}|z^i) \tag{5}$$

Let's expand out the steps in Section 4.1 in more detail. Substituting the expression for Boltzmann distribution $P_\theta(x|z)$ results in,

$$\begin{aligned} \mathcal{L}(\theta) &= \frac{1}{|\mathcal{D}|} \sum_{(x^i_{gt}, z^i)\in\mathcal{D}} -\log(\exp(-E(\theta; x^i_{gt}, z^i))) + \log Z(\theta; z^i) \\ \mathcal{L}(\theta) &= \frac{1}{|\mathcal{D}|} \sum_{(x^i_{gt}, z^i)\in\mathcal{D}} E(\theta; x^i_{gt}, z^i) + \log \int_x \exp(-E(\theta; x, z^i))dx \end{aligned} \tag{6}$$

Take the gradient of this loss,

$$\nabla_\theta \mathcal{L}(\theta) = \frac{1}{|\mathcal{D}|} \sum_{(x^i_{gt}, z^i)\in\mathcal{D}} \left[\nabla_\theta E(\theta; x^i_{gt}, z^i)) + \frac{1}{Z(\theta; z^i)} \int_x \nabla_\theta E(\theta; x, z^i) \exp(-E(\theta; x, z^i))\right] \tag{7}$$

Substituting in $P_\theta(x|z^i)$ from Section 3.1 we have the resulting gradient expression that we finally obtained in Section 4.1,

$$\nabla_\theta \mathcal{L}(\theta) = \frac{1}{|\mathcal{D}|} \sum_{(x^i_{gt}, z^i)\in\mathcal{D}} \left[\underbrace{\nabla_\theta E(\theta; x^i_{gt}, z^i))}_{\text{ground truth samples}} - \mathop{\mathbb{E}}_{\substack{x\sim \\ P_\theta(x|z)}} \underbrace{\nabla_\theta E(\theta; x, z^i))}_{\text{learned distribution samples}}\right] \tag{8}$$

Negative log likelihood approximation Since directly sampling from the true posterior $P_\theta(x|z) = \frac{1}{\mathcal{Z}(\theta)} e^{-E(\theta,x;z)}$ is intractable, in LEO, we instead sample from a Gaussian approximation of this distribution, $\hat{P}_\theta(x|z) = \mathcal{N}(\mu, \Sigma)$ where μ, Σ are obtained as shown in Section **??**. Hence, we effectively minimize an approximation of the true NLL loss,

$$\hat{\mathcal{L}}(\theta) = \frac{1}{|\mathcal{D}|} \sum_{(x^i_{gt}, z^i)\in\mathcal{D}} -\log \hat{P}_\theta(x^i_{gt}|z^i) \tag{9}$$

The difference between the approximated loss and the true loss is,

$$\begin{aligned} \hat{\mathcal{L}}(\theta) - \mathcal{L}(\theta) &= \frac{1}{|\mathcal{D}|} \sum_{(x^i_{gt}, z^i)\in\mathcal{D}} -\log \hat{P}_\theta(x^i_{gt}|z^i) - \frac{1}{|\mathcal{D}|} \sum_{(x^i_{gt}, z^i)\in\mathcal{D}} -\log P_\theta(x^i_{gt}|z^i) \\ &= \frac{1}{|\mathcal{D}|} \sum_{(x^i_{gt}, z^i)\in\mathcal{D}} \frac{P_\theta(x^i_{gt}|z^i)}{\hat{P}_\theta(x^i_{gt}|z^i)} \\ &= \mathop{\mathbb{E}}_{(x_{gt}, z)\sim\mathcal{D}} \frac{P_\theta(x_{gt}|z)}{\hat{P}_\theta(x_{gt}|z)} \leq \max_{(x_{gt}, z)\in\mathcal{D}} \log \frac{P_\theta(x_{gt}|z)}{\hat{P}_\theta(x_{gt}|z)} \end{aligned} \tag{10}$$

The difference in losses is bounded by the maximum log density ratio between the true and approximated distributions. The ratio is bounded as long as the denominator is not zero, i.e. the approximation has full support over the dataset.

D Results and Evaluation: Additional Details

D.1 Setup and Factor Graph Models

Fig. 5 shows the two factor graphs modeling the (a) synthetic navigation and (b) real-world planar pushing tasks. For both graphs, poses are modeled as variable nodes and measurements as factor nodes. The graph inference objective is to solve for the latent poses (variables) given measurements (factors).

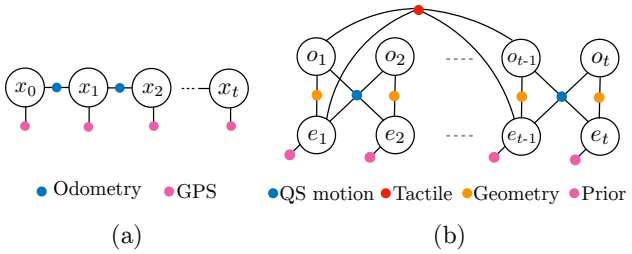

Figure 5: Factor graphs for (a) synthetic navigation (b) real-world planar pushing.

Synthetic Navigation To replicate a typical robot navigation setup, we generate odometry measurements by adding Gaussian noise to *relative* ground truth state estimates and GPS measurements by adding Gaussian noise to *absolute* ground truth estimates. We generate 4 datasets N1–N4, each with a different covariance setting. N1, N2 has measurements generated from fixed covariances $\Sigma := \{\Sigma_{odom}, \Sigma_{gps}\}$. N3, N4 has measurements generated from covariances varying between two sets of values, i.e. $\Sigma := \{\Sigma_{odom}(z), \Sigma_{gps}(z)\}$, where z are binary measurements simulating an ambient light detector that determines whether the robot is in an indoor or outdoor environment.

To solve for the sequence of states $x := \{x_1 \dots x_T\}$ from measurements z, we provide the following objective for the graph optimizer to minimize,

$$\hat{x} = \underset{x}{\text{argmin}} \sum_{k} \{ \; ||f_{odom}(x_{k-1}, x_k) - z^{odom}_{k-1,k}||^2_{\Sigma_{odom}(z)} + ||f_{gps}(x_k) - z^{gps}_{k}||^2_{\Sigma_{gps}(z)} \; \} \tag{11}$$

The observation model parameters that we learn for this task are the fixed and varying covariances, i.e. $\theta := \{\Sigma_{odom}, \Sigma_{gps}\}, \{\Sigma_{odom}(z), \Sigma_{gps}(z)\}$.

Real-world planar pushing For the planar pushing setup, states $x := \{x_1 \dots x_T\}$ in the graph are the planar object and end-effector poses at every time step, with $x_t = [o_t \; e_t]^T$, where $o_t, e_t \in SE(2)$. Factors in the graph incorporate tactile observations $f_{tac}(\cdot)$, quasi-static pushing dynamics $f_{qs}(\cdot)$, geometric constraints $f_{geo}(\cdot)$, and priors on end-effector poses $f_{eff}(\cdot)$.

To solve for the sequence of state $x := \{x_1 \dots x_T\}$ from measurements z, we pass in the following objective for the graph optimizer to minimize,

$$\begin{aligned} \hat{x} = \underset{x}{\text{argmin}} \sum_{k} \{ \; & ||f^{\phi}_{tac}(o_{k-w}, o_k, e_{k-w}, e_k)||^2_{\Sigma_{tac}} + ||f_{qs}(o_{k-1}, o_k, e_{k-1}, e_k)||^2_{\Sigma_{qs}} + \\ & ||f_{geo}(o_k, e_k)||^2_{\Sigma_{geo}} + ||f_{eff}(e_k)||^2_{\Sigma_{eff}} \; \} \end{aligned} \tag{12}$$

The observation model parameters that we learn for this task are the tactile factor network weights and the tactile and quasi-static factor covariances, i.e. $\theta := \{\phi, \Sigma_{tac}, \Sigma_{qs}\}$. As also seen in Fig. 5, quasi-static motion factors are added between consecutive poses $\{k-1, k\}$ and tactile factors are added between non-consecutive poses $\{k-w, k\}$, where w is some window length. The tactile observation factors effectively act as loop closure factors typically used in SLAM graph optimizers for global drift correction. More details on the factor graph models can be found in prior work [12].

D.2 Baselines

For the hyper-parameter search baselines we use off-the-shelf solvers like CMA-ES [13], and scipy optimizers such as Nelder-Mead. For the learned sequence model baseline, we use an LSTM.

LSTM architecture For the synthetic navigation task, we directly regresses to absolute poses. At each time step, the absolute pose and odometry measurements are concatenated as inputs, and the network predicts the 2D position and the sin and cos of the rotation angle. We use a 2-layer LSTM

with 64 hidden units, followed by a 2 dense layers of 32 hidden units before decoding into the 4D output. The loss is a weighted sum of translation and rotation errors. In the real-world planar pushing datasets, we do not have direct measurements of the object pose. Therefore, we transform all end-effector trajectories to start at the origin, which improves the generalization. Each input consists of the end-effector pose observation and the 2d object contact observation, and the same architecture as the navigation dataset is used. In addition, we found that increasing the sequence length with each epoch improved performance.

Fig. 6 shows qualitative results for the LSTM on the (a) synthetic navigation and (b) real-world planar pushing datasets. The performance on the planar pushing object trajectories is not as good since, unlike navigation where we had noisy GPS measurements, tactile measurements only give partially observable information about the object state.

Acknowledgements

This work was supported through the Facebook FRAIM program. We thank Frank Dellaert for insightful feedback and suggestions on the paper.

References

[1] B. Amos and D. Yarats. The differentiable cross-entropy method. In *International Conference on Machine Learning*, pages 291–302. PMLR, 2020.

[2] B. Yi, M. Lee, A. Kloss, R. Martín-Martín, and J. Bohg. Differentiable factor graph optimization for learning smoothers. *arXiv preprint arXiv:2105.08257*, 2021.

[3] https://github.com/borglab/SwiftFusion.

[4] Package Contributors and Ecosystem. Caesar.jl, v0.10.2, 2021. https://github.com/JuliaRobotics/Caesar.jl.

[5] A. D. Cobb and B. Jalaian. Scaling hamiltonian monte carlo inference for bayesian neural networks with symmetric splitting. *Uncertainty in Artificial Intelligence*, 2021. https://github.com/AdamCobb/hamiltorch.

[6] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. Leonard, and F. Dellaert. iSAM2: Incremental smoothing and mapping using the Bayes tree. *Intl. J. of Robotics Research*, 31(2):216–235, Feb. 2012.

[7] E. T. Jaynes. Information theory and statistical mechanics. *Physical review*, 106(4), 1957.

[8] B. D. Ziebart. *Modeling purposeful adaptive behavior with the principle of maximum causal entropy*. PhD thesis, 2010.

[9] Y. LeCun, S. Chopra, R. Hadsell, M. Ranzato, and F. Huang. A tutorial on energy-based learning. *Predicting structured data*, 1(0), 2006.

[10] S. Levine, Z. Popovic, and V. Koltun. Nonlinear inverse reinforcement learning with Gaussian processes. In *Advances in Neural Information Processing Systems 24 (NeurIPS)*, 2011.

[11] C. Finn, P. Christiano, P. Abbeel, and S. Levine. A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models. *arXiv preprint arXiv:1611.03852*, 2016.

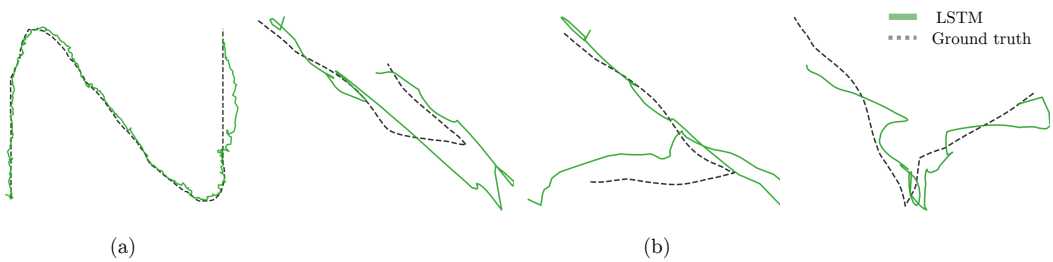

Figure 6: Qualitative LSTM results on **(a)** a synthetic navigation robot trajectory **(b)** planar pushing object trajectories.

[12] P. Sodhi, M. Kaess, M. Mukadam, and S. Anderson. Learning tactile models for factor graph-based estimation. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2021.

[13] N. Hansen, Y. Akimoto, and P. Baudis. CMA-ES/pycma on Github. Zenodo, DOI:10.5281/zenodo.2559634, 2019. URL https://doi.org/10.5281/zenodo.2559634.