LEGO: Learning Energy-based Models in Graph Optimization

Anonymous Author(s)
Affiliation
Address
email

Abstract:
We address the problem of learning observation models end-to-end for estimation. Robots operating in partially observable environments must infer latent states from multiple sensory inputs using observation models that capture the joint distribution between latent states and observations. This inference problem can be formulated as an objective over a graph that optimizes for the most likely sequence of states using all previous measurements. Prior work uses observation models that are either known a-priori or trained on surrogate losses independent of the graph optimizer. In this paper, we propose a method to directly optimize end-to-end tracking performance by learning observation models with the graph optimizer in the loop. This direct approach may appear, however, to require the inference algorithm to be fully differentiable, which many state-of-the-art graph optimizers are not. Our key insight is to instead formulate the problem as that of energy-based learning. We propose a novel approach, LEGO, for learning observation models end-to-end with non-differentiable graph optimizers. LEGO alternates between sampling trajectories from the graph posterior and updating the model to match these samples to ground truth trajectories. We propose a way to generate such samples efficiently using incremental Gauss-Newton solvers. We compare LEGO against baselines on datasets drawn from two distinct tasks: navigation and real-world planar pushing. We show that LEGO is able to learn complex observation models with lower errors and fewer samples.

Keywords: factor graphs, energy-based models, state estimation

1 Introduction

We focus on the problem of learning observation models end-to-end for estimation. Consider a robot hand manipulating an object with only tactile feedback. It must reason over *a sequence of* touch observations over time to collapse uncertainty about the latent object pose. A common way to solve this is as an inference over a factor graph which relies on having observation models that can map observations to states [1, 2, 3]. However, in many domains, the sensors that produce observations are complex and difficult to model. *Can we instead learn observation models from data?*

Given a batch of ground truth trajectories and corresponding measurements, how should we learn observations models? One approach would be to learn a direct mapping from measurement to state, for example as a regression or classification problem [4, 5, 6]. However, while easy to optimize given a direct supervised loss, this only minimizes a surrogate loss independent of the graph optimizer and is not guaranteed to minimize the final tracking errors that we care about. The other option would be to directly minimize final tracking errors between optimized and ground truth trajectories [7, 8]. However, many state-of-the-art factor graph optimizers e.g. iSAM2 [9] are not natively differentiable due to operations such as dynamic re-linearizations[1]. Hence, we are limited to black-box search for learning the parameters of these optimizers which is very sample inefficient [10, 11, 12].

Instead of differentiating through the optimization process, we note that what we ultimately care about is the final solution from the optimizer, which depends *only on the shape* of the optimized cost function. We would like a cost function that has low cost around the observed ground truth trajectories and high cost elsewhere. This is precisely what energy-based models (EBMs) aim to do

Submitted to the 5th Conference on Robot Learning (CoRL 2021). Do not distribute.

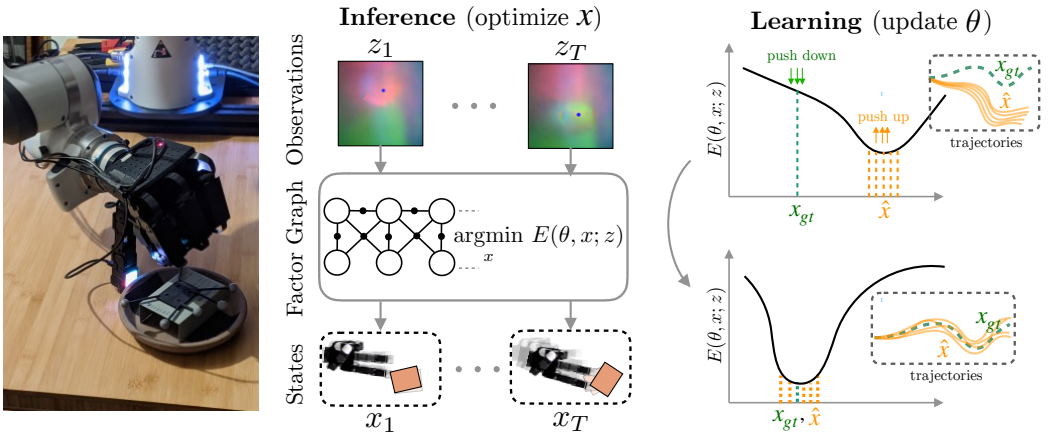


Figure 1: We learn observation models in non-differentiable graph optimizers to minimize end-to-end trajectory tracking errors. **Inference** solves for the most likely set of states x given the model and input measurements z. **Learning** updates model parameters θ from training data.

by shaping an "energy" function to be low around observed data and high elsewhere [13, 14, 15]. We cast our problem of learning observation models for estimation as energy-based learning.

We propose a novel approach, **L**earning **E**nergy-based models in **G**raph **O**ptimization (LEGO), for learning observation models end-to-end with non-differentiable graph optimizers. LEGO alternates between sampling trajectories from the graph posterior and updating the model to match the distribution of samples to ground truth trajectories. We show that we can generate such samples efficiently using incremental Gauss-Newton graph solvers that create a Gaussian approximation for the Boltzmann distribution defined over a continuous space of trajectories. To the best of our knowledge, this is the first paper to show that even non-differentiable factor graph models can be learned efficiently. Our main contributions are:

1. A novel formulation of learning observation models in estimation as energy-based learning.
2. An algorithm, LEGO, to learn observation models that minimize end-to-end tracking errors in non-differentiable factor graphs.
3. Open-source code along with empirical evaluation on two distinct tasks: synthetic navigation and real-world planar pushing.

2 Related Work

Filtering and smoothing based estimation. Observation models represent joint or conditional distributions between states and measurements. Early estimation methods utilized these models within filtering contexts such as Kalman filters and EKFs/UKFs [16, 17, 18]. Recent methods have also looked at making these filters differentiable for end-to-end learning [19, 20, 21]. However, filtering has been shown to be inconsistent for nonlinear estimation problems due to linearization based on past, marginalized states that cannot be undone [22]. Instead, a wide consensus amongst the simultaneous localization and mapping (SLAM) community is to solve these problems as smoothing or nonlinear optimization objectives [3, 2]. Unlike filtering, smoothing allows dynamic relinearization of past states as well as exploits the inherent sparsity resulting in efficient and more accurate solutions. Factor graphs are an increasingly popular way for solving such smoothing problems [1, 2, 23, 4, 24, 25], offering a flexible modeling framework while being efficient to optimize. We use the factor graph based smoothing framework in this paper.

Learning for smoothing. Typically, observation models used in the factor graph are analytic models defined a-priori [26, 27, 28, 29]. Recent work has looked at using learned models [4, 5] or learned measurement representations [23, 30] either to be used independently or to be plugged into a graph optimizer. However, these model parameters are learned on surrogate losses independent of the graph optimizer, and hence do not directly attempt to minimize final tracking errors.

An alternative approach is to view observation model parameters as hyper-parameters of the graph and perform a hyper-parameter search. This is usually done as sampling and derivative-free ap-

[1] see Appendix C for discussion on applying LEGO to differentiable optimizers.

proaches using black-box solvers [10, 11, 12]. However, the high computational burden associated with sampling and evaluation can be prohibitive for large parameter spaces.

Differentiable optimization. Instead of a search, one may learn these model parameters via a continuous gradient-based optimizer matching graph optimizer output against a desired solution. Minimizing this loss would, however, require differentiating through the argmin optimization process. One way to do this is via *unrolling* where the optimization is represented as a series of differentiable, gradient based updates [31, 32, 15]. Since smoothing is a non-linear least-squares optimization, it too can be unrolled in a similar fashion via differentiable Levenberg-Marquardt updates [33, 8] or differentiable Gauss-Newton updates [34, 35, 7]. While easy to apply, vanilla unrolling suffers from a few drawbacks notably the learned cost function can be sensitive to the *specific optimization procedure*, e.g. Amos and Yarats [36] find that unrolled gradient descent "... overfit to the number of gradient steps so that the output ... isn't even a local minimum". Differentiable optimization is also currently limited to optimizers that can be expressed as a composition of differentiable operations. This is not true for high performance optimization packages such as GTSAM, SNOPT, IPOPT [37, 38] that rely on non-differentiable heuristics containing state-of-the-art incremental Gauss-Newton solvers e.g. iSAM2 [9] that we employ.

Energy-based learning. Instead of differentiating through the optimization process, we note what we ultimately care about is the final solution from the optimizer, which depends only on the shape of the cost function. This is what energy-based models aim to do by shaping an "energy" function to be low around observed data and high elsewhere [13]. Energy-based modeling is a family of unsupervised learning methods popular in many structured prediction tasks such as inverse reinforcement learning and computer vision [39, 40, 41, 42]. Recently, a number of approaches have looked at training deep energy-based models [43, 44, 15]. The challenge with such methods is generating samples from the energy based model. Methods either rely on MCMC sampling [45, 46, 47] which does not scale well with dimension or resort to learning a separate generator [48, 49, 14]. We propose a way to generate such samples efficiently using incremental Gauss-Newton solvers.

3 Problem Formulation

We formalize our problem of learning observation models under the general theoretical framework of energy based models. We begin by describing the observation model, how it is optimized at inference time and finally how its parameters are learned from training data (Fig. 1).

3.1 Model

We define an energy-based model that captures dependencies between variables in a graph by associating a scalar energy $E(\cdot)$ to each configuration of the graph. Our goal is to model the likelihood of a sequence of latent states x given a sequence of measurements z. We adopt a factor graph framework (Fig. 1(a)) for expressing this likelihood as,

$$P_\theta(x|z) = \frac{1}{\mathcal{Z}(\theta; z)} \exp\left\{-E(\theta, x; z)\right\} = \frac{1}{\mathcal{Z}(\theta; z)} \exp\left\{-\frac{1}{2}\sum_k ||f_k(\theta, x; z)||^2\right\} \tag{1}$$

where, $x := \{x_1 \dots x_T\}$ is the sequence of latent states, and $z := \{z_1 \dots z_T\}$ is the sequence of measurements. $||f_k(\theta, x; z)||^2$ are the factor costs with $f_k(\theta, x; z)$ being the local observation model mapping a *subset* of states x and measurements z to a cost, and θ the learnable model parameters[2]. $\mathcal{Z}(\theta; z) = \int_x \exp(-E(\theta, x; z))dx$ is the normalization, also referred to as the partition function.

Note that Eq. 1 is a special case of the Boltzmann distribution where the energy $E(\cdot)$ is a sum-of-squares. Notably, when $f_k(\theta, x; z)$ is linearized, $E(\cdot)$ is quadratic in x and subsequently $P_\theta(x|z)$ becomes a Gaussian distribution. This is the structure that Gauss-Newton solvers employ, and we will leverage this fact to efficiently sample trajectories as described later in Section 4.3.

3.2 Inference

At inference time, given a sequence of measurements z, we wish to solve for the most likely sequence of latent states x. We formulate this as maximizing the log-likelihood of the model,

$$\hat{x} = \underset{x}{\operatorname{argmax}} \ \log P_\theta(x|z) \tag{2}$$

[2]This may include covariance Σ_k in the weighted norm $|| \cdot ||_{\Sigma_k}$

Substituting the model expression from Eq. 1, and dropping constants $\mathcal{Z}(\theta; z)$ and $1/2$, results in a nonlinear least-squares minimization of the form,

$$\hat{x} = \underset{x}{\text{argmax}} \ \log \frac{1}{\mathcal{Z}(\theta; z)} \exp\left\{-\frac{1}{2}\sum_k ||f_k(\theta, x; z)||^2\right\} = \underset{x}{\text{argmin}} \sum_k ||f_k(\theta, x; z)||^2 \quad (3)$$

We use an efficient, yet non-differentiable, incremental Gauss-Newton graph solver to solve for this objective. This enables both real-time inference as well as efficient sampling for the purpose of learning model parameters during training (Section 4.3).

3.3 Learning

At train time, our goal is to learn a model that explains training data of pairs (x_{gt}, z). We express this as minimizing a loss $\mathcal{L}(\theta)$ over the training data

$$\mathcal{L}(\theta) = \frac{1}{|\mathcal{D}|} \sum_{(x^i_{gt}, z^i) \in \mathcal{D}} \mathcal{L}(\theta; x^i_{gt}, z^i) \quad (4)$$

where, $\{x^i_{gt}, z^i\} \in \mathcal{D}$ is a training dataset of ground truth trajectories and measurements.

There are various choices of losses $\mathcal{L}(\cdot)$ for learning model parameters θ,

Loss 1: Minimize final tracking loss. A straightforward choice for $\mathcal{L}(\cdot)$ is to directly minimize final tracking errors between optimized $\hat{x}^i$ and ground truth x^i_{gt} trajectories, i.e. $\mathcal{L}(\theta; x_{gt}, z) = 1/|\mathcal{D}| \sum_i ||\hat{x}^i \ominus x^i_{gt}||^2_2$. While this loss directly minimizes final tracking errors, it is not differentiable since the graph inference path from θ to $\hat{x}$ is not differentiable. Hence, we are limited to black-box hyperparameter search which is very sample inefficient.

Loss 2: Minimize surrogate loss. An alternate choice for the loss $\mathcal{L}(\cdot)$ is to directly map measurements to ground truth states, for example as a regression or classification problem. Such a surrogate loss is independent of graph inference, and unlike Loss 1, is differentiable and easy to optimize. However, this does not minimize the final tracking errors that we care about.

Loss 3: Minimize energy-based loss. Finally, we consider a loss that is a function of the energy, i.e. $\mathcal{L}(E(\theta, \cdot); x^i_{gt}, z^i)$. Intuitively, this assigns a low loss to *well-behaved* energy functions, i.e. functions that give the lowest energy to training data of ground truth trajectories (correct answers) and higher energy to unseen data (incorrect answers).

This energy-based loss, unlike Loss 2, is highly correlated with the final tracking loss as it shapes the energy (or cost) landscape so as to make the inference step return trajectories closer to the ground truth. Moreover, unlike Loss 1, this loss is only a function of energies which is differentiable *irrespective of inference*. Hence, we use Loss 3 and detail our approach in the next section.

4 Approach

In Section 3, we established that the energy-based loss, Loss 3, strikes a desirable trade-off by being both differentiable while also minimizing the final tracking loss. We now define a concrete instantiation of this loss and an algorithm to efficiently minimize this.

4.1 Normalized Negative Log-Likelihood Loss

We define the overall loss as the negative log-likelihood of the data under the energy model[3]:

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

Substituting $P_\theta(x|z)$ as the Boltzman distribution expression from Eq. 1 we have,

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

[3]The motivation for such a loss comes from probabilistic modeling, notably the principal of maximum entropy moment matching. We provide a more detailed derivation of Section 4.1 in Appendix A.

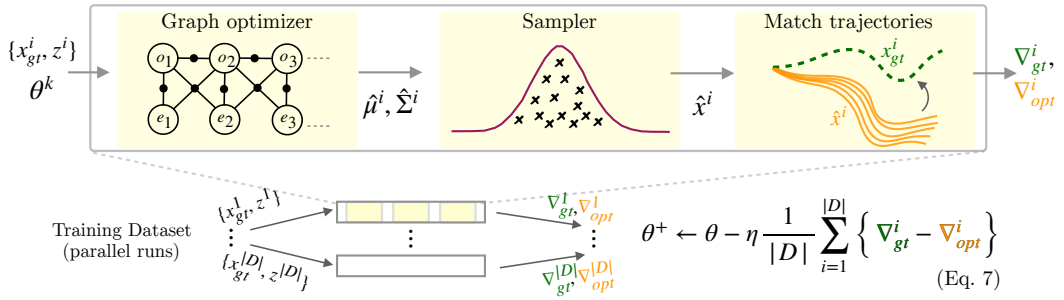


Figure 2: LEGO algorithm: For each training example, the zoomed in panel shows the three main stages: (a) solve graph optimization, (b) sample trajectories from graph posterior, and (c) compute gradients. Gradients are then pooled together to update parameters θ.

Algorithm 1 LEGO algorithm

Input Initial θ, Training data $\{x^i_{gt}, z^i\}_{i=1}^{|\mathcal{D}|}$
while <!converged> **do**
  for each training example i **do**
    $\hat{\mu}^i, \hat{\Sigma}^i = \underset{x}{\text{argmin}}\{E(\theta, x; z^i)\}$ ▷ Solve graph optimizer objective
    $\hat{x}^i \sim \mathcal{N}(\hat{\mu}^i, \hat{\Sigma}^i)$ ▷ Sample S trajectories from graph posterior
  end for
  $\theta^+ \leftarrow \theta - \eta \frac{1}{|\mathcal{D}|} \sum_{i=1}^{|\mathcal{D}|} \left\{ \nabla_\theta E(\theta; x^i_{gt}, z^i) - \frac{1}{S} \sum_{\hat{x}^i} \nabla_\theta E(\theta; \hat{x}^i, z^i) \right\}$ ▷ Update model parameters (Eq. 7)
end while

The expression above has an intuitive explanation. The first term penalizes high energy over ground truth training samples. The second term penalizes low energy everywhere else, thus acting as a "contrastive" term. This has a desirable effect of globally shaping the energy function so that the minima lies at the training samples. The contrastive term prevents the energy surface from becoming flat and thus degenerate [13]. Unfortunately, we cannot directly plug in Eq. 6 as a standard machine learning loss (e.g. in PyTorch) to optimize. This is because the second log partition term is over a continuous space of trajectories x, which renders exact computation intractable for arbitrary nonlinear cost functions [50].

Instead, let's take the gradient of this loss, and substitute $P_\theta(x|z^i)$ expression from Eq. 1,

$$\nabla_\theta \mathcal{L}(\theta) = \frac{1}{|\mathcal{D}|} \sum_{(x^i_{gt}, z^i) \in \mathcal{D}} \left[\underbrace{\nabla_\theta E(\theta; x^i_{gt}, z^i))}_{\text{ground truth samples}} - \underset{x \sim P_\theta(x|z)}{\mathbb{E}} \underbrace{\nabla_\theta E(\theta; x, z^i))}_{\text{learned distribution samples}} \right] \tag{7}$$

We see the log partition term transforms into an expectation over the learned distribution. The gradient expression above has a similar interpretation as the loss. The first term pushes down the energy of ground truth training samples. Interestingly, the second term now pulls up the energy of samples drawn from the learned Boltzmann distribution. If one is able to generate such samples, computing the gradient is straightforward.

Generally, sampling from the continuous Boltzmann distribution is intractable, relying typically on MCMC methods. But as we will detail in Section 4.3, we can indeed generate such samples efficiently since our graph optimizer maintains a Gaussian approximation of this distribution. Additionally in practice, we found that introducing a temperature term T inside the Boltzmann distribution, i.e. $P_\theta(x|z) \propto \exp(-1/T E(\theta, x; z))$, was critical for controlling the variance of the samples. Setting $T = 0$ corresponds to only using the mode of the distribution which in this case is the mean trajectory. This corresponds to minimizing a generalized perceptron loss [13].

4.2 LEGO Algorithm

Algorithm 1 summarizes the LEGO algorithm, also illustrated in Fig. 2. It takes as inputs the training data containing ground truth state and measurement pairs, i.e. $\{x^i_{gt}, z^i\}_{i=1}^{|\mathcal{D}|}$, along with initial values for the learnable observation model parameters θ. The goal is to converge to a θ that minimizes the log-likelihood loss in Eq. 5 over the given training set $i = 1 \ldots |\mathcal{D}|$.

LEGO invokes a black-box, non-differentiable graph optimizer iSAM2 in the loop to generate samples. Within each LEGO iteration, for every training example i, we invoke the graph optimizer to obtain a Gaussian approximation $\mathcal{N}(\hat{\mu}^i, \hat{\Sigma}^i)$ to the original Boltzmann distribution $P_\theta(x|z)$ about its mode. We then sample trajectories $\hat{x}^i \sim \mathcal{N}(\hat{\mu}^i, \hat{\Sigma}^i)$ from this distribution.

The next step is to compute gradients of the energy, i.e. $\nabla_\theta E(\theta; x^i_{gt}, z^i)$ and $\nabla_\theta E(\theta; \hat{x}^i, z^i)$, at the ground truth x^i_{gt} and the samples $\hat{x}^i$, respectively. Note that the gradients here are only w.r.t. θ, and not x, and hence do not require the graph optimizer to be differentiable. Once we have the two gradients, we use the difference between them to update the observation model parameters θ. These updated parameters are then used in the next LEGO iteration.

4.3 Efficient Inference via Incremental Gauss-Newton

The only thing that remains to be specified in Algorithm 1 is — How do we generate samples efficiently from the graph inference? In Section 3.2 we noted that our inference algorithm must satisfy two requirements (a) real-time inference and (b) sampling from the Boltzmann distribution. We achieve this using via incremental Gauss-Newton to solve the nonlinear least squares in Eq. 3.

Gauss-Newton proceeds by linearizing observation model functions $f_k(\cdot)$ about a point x^0,

$$f_k(\theta, x; z) = f_k(\theta, x^0 + \delta x; z) \approx f_k(\theta, x^0; z) + F_k \delta x \tag{8}$$

where, x^0 is the linearization point, $F_k = \left.\frac{\partial f_k(\theta, x; z)}{\partial x}\right|_{x^0}$ is the measurement Jacobian, and $\delta x = x - x^0$ the state update vector. Measurements z and learnable parameters θ are treated as constant within the graph optimization step. Substituting the Taylor expansion in Eq. 8 back into Eq. 3,

$$\delta x^* = \underset{\delta x}{\operatorname{argmin}} \sum_{k=1}^{K} ||F_k \delta x - \left(-f_k(\theta, x^0; z)\right)||^2 = \underset{\delta x}{\operatorname{argmin}} \, ||A\delta x - b||_2^2 \tag{9}$$

where, A, b are constructed by concatenating all F_k, $f_k(\theta, x^0; z)$ together into a single matrix and vector respectively. We then iterate $x^0 \leftarrow x^0 + \delta x^*$ until convergence.

We can solve for the objective in Eq. 9 efficiently by exploiting two properties of the problem: *sparsity* and *online* measurements. For SLAM problems, both A and $A^T A$ are large but sparse matrices, and can hence be factorized efficiently using sparse QR or Cholesky factorization respectively [51]. However, this still does not guarantee real-time inference during test time since A grows with measurements over time, making the method increasingly slower. Instead, subsequent incremental solvers [9, 52, 53] reuse matrix factorizations from previous time steps. In this work, we use iSAM2 [9], an incremental Gauss-Newton graph solver whose resulting MAP solution can be expressed as a Gaussian probability distribution with mean and covariance,

$$\hat{\mu} = \underset{x}{\operatorname{argmin}} \sum_{k=1}^{K} ||f_k(\theta, x; z)||^2, \qquad \hat{\Sigma} = \left(A^T A\right)^{-1}|_{x=\hat{\mu}} \tag{10}$$

This is the Gaussian approximation of the original Boltzmann distribution $P_\theta(x|z^i)$ which we can now sample from efficiently. The graph posterior covariance $\hat{\Sigma}$ for such a Gauss-Newton solver is obtained using just Jacobians A without needing to compute a Hessian [2]. Note that sampling $\hat{x} \sim \mathcal{N}(\hat{\mu}, \hat{\Sigma})$ is done at the tangent space of the linearized graph, i.e. $\hat{x} \leftarrow \hat{\mu} \oplus \delta x$ where $\delta x \sim \mathcal{N}(0, \hat{\Sigma})$ and $\oplus$ is a retraction [54].

5 Results and Evaluation

We evaluate LEGO on synthetic navigation and real-world planar pushing. We compare against baselines on metrics like final tracking error and sample efficiency. We implement LEGO in PyTorch [55] and interface it with the GTSAM C++ library [37], which contains several non-differentiable factor graph optimizers. We use the iSAM2 [9] optimizer for real-time, online optimization.

5.1 Overview

Experimental setup: For synthetic navigation (Datasets N1–N4), the goal is to estimate latent robot poses $x_t \in SE(2)$ from relative odometry and absolute GPS measurements. For real-world planar

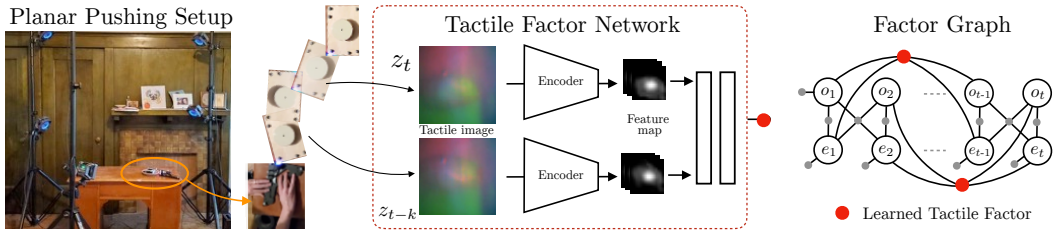


Figure 3: (a) Experimental setup for real-world planar pushing, **(b)** Learned tactile factor (red) penalizes deviations between predicted relative pose from the tactile factor network and estimated relative pose using current variable estimates in the graph.

pushing (Datasets P1–P3), the goal is to estimate latent object poses $o_t \in SE(2)$ from tactile image measurements using a DIGIT tactile sensor [56] as shown in the setup in Fig. 3. Both these problems are modeled as factor graphs with poses as variable nodes and measurements as factor nodes.

Observation model parameters: Recall factor costs $||f_k(\theta, x; z)||^2$ are defined by the local observation model $f_k(\cdot)$ and learned model parameters θ. This can be expanded out as $||f_k(\theta, x; z)||^2 := ||\mu_k(x) - z||^2_{\Sigma_k}$, where $\theta := \{\mu_k, \Sigma_k\}$ are the mean and covariance parameters. For the navigation task, we learn (a) fixed covariances $\theta := \{\Sigma_{odom}, \Sigma_{gps}\}$ (N1, N2), and (b) covariances as a function of measurements $\theta := \{\Sigma_{odom}(z), \Sigma_{gps}(z)\}$ (N3, N4). For the planar pushing task, we learn (a) fixed covariances $\theta := \{\Sigma_{tac}, \Sigma_{qs}\}$ (P1, P2), and (b) means as a function of tactile image measurements $\theta := \{\mu_{tac}(z)\}$ (P3). For all experiments, we use a diagonal covariance model.

Baselines: We compare against a learned sequence model such as LSTM, and black-box hyperparameter search methods such as CMAES [10] and Nelder-Mead, all of which optimize Loss 1. We also compare against a surrogate supervised learning method [4] which optimizes Loss 2.

Appendix B expands further on the setup, factor graph models, and additional results.

5.2 End-to-end Tracking Errors

Fig. 4 shows final tracking results for navigation datasets. We initialize LEGO with random θ such that the graph optimizer returns trajectories far from the ground truth. At the final iteration, LEGO converges to a θ that matches the optimized trajectories to ground truth. We summarize quantitative performance against baselines minimizing final tracking loss (Loss 1). LEGO consistently outperforms baselines on final rmse tracking errors. The difference is more pronounced for datasets N3, N4 where the varying covariance model has a higher dimensional parameter space.

Fig. 5 shows final tracking results for real-world planar pushing datasets. Similar to navigation, we show LEGO converging to ground truth trajectory for two different objects, a rectangle and ellipse, with differing local contact patches. We report quantitative performance on datasets P1, P2. LEGO outperforms different hand-tuned choices of covariances as used in prior work [4].

5.3 Sample Efficiency and Convergence

Black-box hyper-parameter search is sample inefficient and does not scale well with increasing dimensionality. LEGO, on the other hand, is efficient as it uses gradients. This can be seen in Fig. 4,

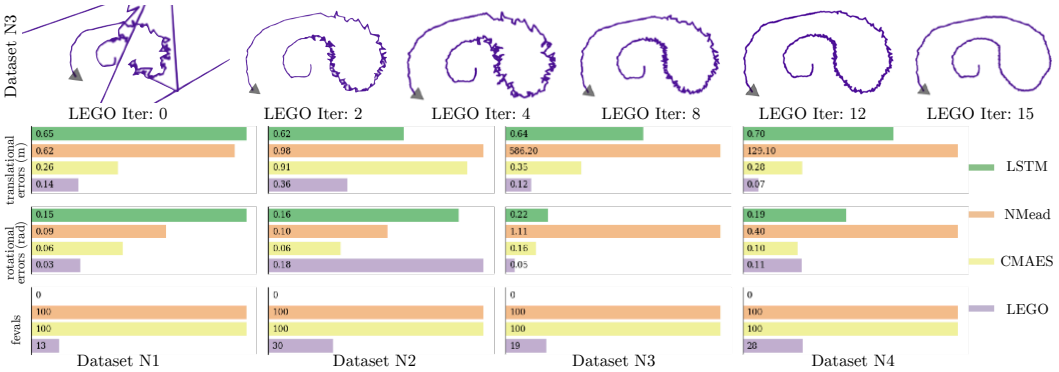


Figure 4: Synthetic Navigation Datasets: Comparison of LEGO against baselines. Each dataset N1-N4 consists of 50 varying trajectories with a 30/20 train/test split. Translational and rotational trajectory tracking errors represent average RMSE over the test set. fevals is number of black-box graph optimizer calls and is computed per data point during training.

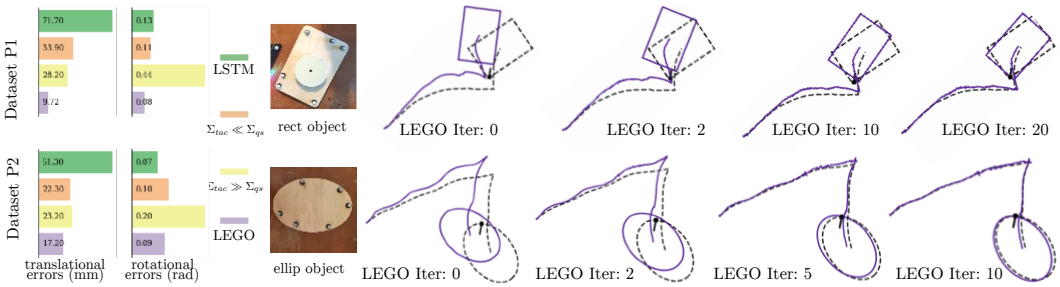


Figure 5: Real-world Planar Pushing Datasets: Each dataset P1, P2 consists of 15 varying trajectories with a 10/5 train/test split for rect, ellipse object respectively. Translational and rotational trajectory tracking errors represent average RMSE over the test set. fevals is number of black-box graph optimizer calls and is computed per data point during training.

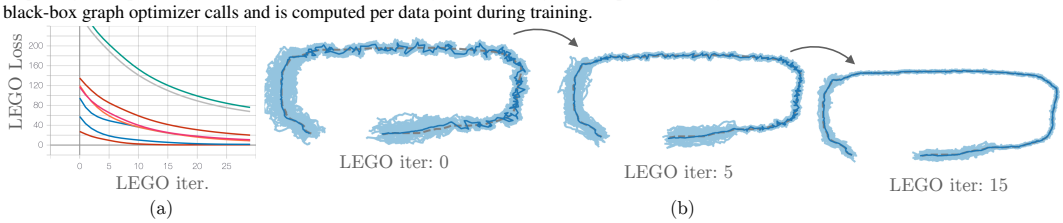
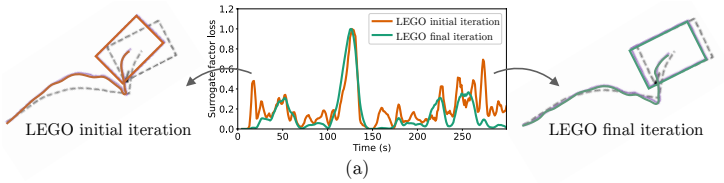


Figure 6: (a) LEGO convergence for varying initial θ on dataset N1. **(b)** Evolution of graph optimizer trajectory samples over LEGO iterations.



Dataset P3	$\theta := \mu_{tac}(z)$ rmse trans (mm)	rmse rot (rad)
Surrogate factor loss	23.6	0.22
LEGO loss	19.9	0.13

(b)

Figure 7: (a) Analysis showing that a surrogate loss, independent of the graph optimizer, may not minimize end-to-end tracking errors. **(b)** Minimizing only surrogate loss results in a higher final tracking error than minimizing the LEGO loss.

where LEGO consistently has lower graph optimizer calls (fevals). Moreover, LEGO can also be used to learn models with large parameter spaces, e.g. Fig. 7 learns parameters for a tactile factor neural network mapping image features to transformations.

Finally, Fig. 6 shows LEGO consistently converging for varying θ initializations on navigation dataset N1. We further show how the LEGO samples evolve over iterations. Initially, when the cost is incorrect, the samples are more spread out. Over iterations, as the cost converges, the samples concentrate about the ground truth trajectory.

5.4 Surrogate Loss Analysis

We further analyze why a surrogate loss (Loss 2), independent of the graph optimizer, may not minimize end-to-end tracking errors. We train a network on surrogate supervised learning loss to directly map a pair of tactile image features to ground truth poses similar to prior work [4]. Fig. 7(b) shows the per time step surrogate loss on two trajectories — an initial trajectory that deviates from ground truth, and a final converged trajectory. While the tracking errors of final is much less than initial, the surrogate loss does not strictly improve. This is further reflected in the table in Fig. 7(c) where the surrogate loss baseline has higher final tracking errors over LEGO for dataset P1.

6 Conclusion

We presented an approach, LEGO, for learning observation models end-to-end with non-differentiable graph optimizers. We compared LEGO against baselines for different observation model types on two distinct tasks of navigation and real-world planar pushing. Overall, LEGO is able to learn observation models with lower errors and fewer samples. While we investigated LEGO with non-differentiable optimizers in this paper, as future work, we would like to apply LEGO to differentiable optimizers and compare against alternate solutions such as unrolling (see Appendix C for a more detailed discussion). We would also like to scale to more complex observation models with applications to 3D object tracking using tactile sensors.

References

[1] F. Dellaert. Factor graphs: Exploiting structure in robotics. *Annual Review of Control, Robotics, and Autonomous Systems*, 4, 2020.

[2] F. Dellaert and M. Kaess. Factor graphs for robot perception. *Foundations and Trends in Robotics*, 6 (1-2):1–139, 2017.

[3] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Trans. Robotics*, 32(6):1309–1332, 2016.

[4] 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.

[5] B. Sundaralingam, A. S. Lambert, A. Handa, B. Boots, T. Hermans, S. Birchfield, N. Ratliff, and D. Fox. Robust learning of tactile force estimation through robot interaction. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2019.

[6] A. Lambert, A. Shaban, A. Raj, Z. Liu, and B. Boots. Deep forward and inverse perceptual models for tracking and prediction. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 675–682, 2018.

[7] 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.

[8] K. M. Jatavallabhula, G. Iyer, and L. Paull. GradSLAM: Dense SLAM meets automatic differentiation. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2020.

[9] 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.

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

[11] H. Hu and G. Kantor. Parametric covariance prediction for heteroscedastic noise. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2015.

[12] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. *arXiv preprint arXiv:1206.2944*, 2012.

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

[14] R. Kumar, S. Ozair, A. Goyal, A. Courville, and Y. Bengio. Maximum entropy generators for energy-based models. *arXiv preprint arXiv:1901.08508*, 2019.

[15] D. Belanger, B. Yang, and A. McCallum. End-to-end learning for structured prediction energy networks. In *International Conference on Machine Learning (ICML)*, pages 429–439, 2017.

[16] R. E. Kalman. A new approach to linear filtering and prediction problems. 1960.

[17] S. J. Julier and J. K. Uhlmann. Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 92(3):401–422, 2004.

[18] S. Thrun, D. Fox, W. Burgard, and F. Dellaert. Robust Monte Carlo localization for mobile robots. *Artificial intelligence*, 128(1-2):99–141, 2001.

[19] A. Kloss, G. Martius, and J. Bohg. How to train your differentiable filter. *arXiv preprint arXiv:2012.14313*, 2020.

[20] T. Haarnoja, A. Ajay, S. Levine, and P. Abbeel. Backprop kf: Learning discriminative deterministic state estimators. In *Advances in Neural Information Processing Systems 24 (NeurIPS)*, 2016.

[21] R. Jonschkowski, D. Rastogi, and O. Brock. Differentiable particle filters: End-to-end learning with algorithmic priors. *arXiv preprint arXiv:1805.11122*, 2018.

[22] S. J. Julier and J. K. Uhlmann. A counter example to the theory of simultaneous localization and map building. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, volume 4, pages 4238–4243, 2001.

[23] J. Czarnowski, T. Laidlow, R. Clark, and A. J. Davison. DeepFactors: Real-time probabilistic dense monocular SLAM. *IEEE Robotics and Automation Letters*, 5(2):721–728, 2020.

[24] K.-T. Yu and A. Rodriguez. Realtime state estimation with tactile and visual sensing. application to planar manipulation. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 7778–7785, 2018.

[25] R. Hartley, M. G. Jadidi, L. Gan, J.-K. Huang, J. W. Grizzle, and R. M. Eustice. Hybrid contact preintegration for visual-inertial-contact state estimation using factor graphs. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 3783–3790, 2018.

[26] R. Mur-Artal and J. D. Tardós. ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-D cameras. *IEEE Trans. Robotics*, 33(5):1255–1262, 2017.

[27] J. Engel, T. Schöps, and D. Cremers. LSD-SLAM: Large-scale direct monocular SLAM. In *Eur. Conf. on Computer Vision (ECCV)*, pages 834–849. Springer, 2014.

[28] A. S. Lambert, M. Mukadam, B. Sundaralingam, N. Ratliff, B. Boots, and D. Fox. Joint inference of kinematic and force trajectories with visuo-tactile sensing. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 3165–3171, 2019.

[29] W. Dong, J. Park, Y. Yang, and M. Kaess. GPU accelerated robust scene reconstruction. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 7863–7870. IEEE, 2019.

[30] M. Bloesch, J. Czarnowski, R. Clark, S. Leutenegger, and A. J. Davison. CodeSLAM–learning a compact, optimisable representation for dense visual slam. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, pages 2560–2568, 2018.

[31] J. Domke. Generic methods for optimization-based modeling. In *Artificial Intelligence and Statistics*, pages 318–326. PMLR, 2012.

[32] B. Amos and J. Z. Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning (ICML)*, pages 136–145, 2017.

[33] C. Tang and P. Tan. Ba-net: Dense bundle adjustment networks. In *International Conference on Learning Representations*, 2018.

[34] M. Bhardwaj, B. Boots, and M. Mukadam. Differentiable Gaussian process motion planning. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 10598–10604, 2020.

[35] L. von Stumberg, P. Wenzel, Q. Khan, and D. Cremers. Gn-net: The gauss-newton loss for multi-weather relocalization. *IEEE Robotics and Automation Letters*, 5(2):890–897, 2020.

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

[37] F. Dellaert. Factor graphs and GTSAM: A hands-on introduction. Technical report, Georgia Institute of Technology, 2012.

[38] P. E. Gill, W. Murray, and M. A. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM review*, 47(1):99–131, 2005.

[39] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey. Maximum entropy inverse reinforcement learning. In *AAAI Conf. on Artificial Intelligence*, volume 8, pages 1433–1438, 2008.

[40] M. Wulfmeier, P. Ondruska, and I. Posner. Maximum entropy deep inverse reinforcement learning. *arXiv preprint arXiv:1507.04888*, 2015.

[41] F. K. Gustafsson, M. Danelljan, R. Timofte, and T. B. Schön. How to train your energy-based model for regression-supplementary material. 2020.

[42] S. Bechtle, A. Molchanov, Y. Chebotar, E. Grefenstette, L. Righetti, G. Sukhatme, and F. Meier. Meta learning via learned loss. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 4161–4168, 2021.

[43] L.-C. Chen, A. Schwing, A. Yuille, and R. Urtasun. Learning deep structured models. In *International Conference on Machine Learning (ICML)*, pages 1785–1794. PMLR, 2015.

[44] D. Belanger and A. McCallum. Structured prediction energy networks. In *International Conference on Machine Learning (ICML)*, pages 983–992. PMLR, 2016.

[45] T. Tieleman. Training restricted boltzmann machines using approximations to the likelihood gradient. In *International Conference on Machine Learning (ICML)*, pages 1064–1071, 2008.

[46] Y. Du and I. Mordatch. Implicit generation and modeling with energy based models. In *Advances in Neural Information Processing Systems 24 (NeurIPS)*, 2019.

[47] E. Nijkamp, M. Hill, S.-C. Zhu, and Y. N. Wu. Learning non-convergent non-persistent short-run mcmc toward energy-based model. In *Advances in Neural Information Processing Systems 24 (NeurIPS)*, 2019.

[48] T. Kim and Y. Bengio. Deep directed generative models with energy-based probability estimation. *arXiv preprint arXiv:1606.03439*, 2016.

[49] Z. Dai, A. Almahairi, P. Bachman, E. Hovy, and A. Courville. Calibrating energy-based generative adversarial networks. *arXiv preprint arXiv:1702.01691*, 2017.

[50] M. J. Wainwright, T. S. Jaakkola, and A. S. Willsky. A new class of upper bounds on the log partition function. *IEEE Transactions on Information Theory*, 51(7):2313–2335, 2005.

[51] F. Dellaert and M. Kaess. Square root SAM: Simultaneous localization and mapping via square root information smoothing. *Intl. J. of Robotics Research*, 25(12):1181–1203, 2006.

[52] M. Kaess, A. Ranganathan, and F. Dellaert. iSAM: Incremental smoothing and mapping. *IEEE Trans. Robotics*, 24(6):1365–1378, 2008.

[53] P. Sodhi, S. Choudhury, J. G. Mangelson, and M. Kaess. ICS: Incremental constrained smoothing for state estimation. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2020.

[54] J. Sola, J. Deray, and D. Atchuthan. A micro lie theory for state estimation in robotics. *arXiv preprint arXiv:1812.01537*, 2018.

[55] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 24 (NeurIPS)*, pages 8024–8035, 2019.

[56] M. Lambeta, P.-W. Chou, S. Tian, B. Yang, B. Maloon, V. R. Most, D. Stroud, R. Santos, A. Byagowi, G. Kammerer, et al. DIGIT: A novel design for a low-cost compact high-resolution tactile sensor with application to in-hand manipulation. *IEEE Robotics and Automation Letters*, 5(3):3838–3845, 2020.