Learning Observation Models with Incremental Non-Differentiable Graph Optimizers in the Loop for Robotics State Estimation

Mohamad Qadri [1] **Michael Kaess** [1]

Abstract

We consider the problem of learning observation models for robot state estimation with incremental non-differentiable optimizers in the loop. Convergence to the correct belief over the robot state is heavily dependent on a proper tuning of observation models which serve as input to the optimizer. We propose a gradient-based learning method which converges much quicker to model estimates that lead to solutions of much better quality compared to an existing state-of-the-art method as measured by the tracking accuracy over unseen robot test trajectories.

1. Introduction

Robot state estimation is the problem of inferring the state of a robot (a set of geometric or physical quantities such as position, orientation, contact forces etc.) given sensor measurements. The problem is typically formulated as Maximum a Posteriori (MAP) inference over factor graphs where each node (robot state) is connected to other states via soft constraints or potentials (factors) which are distilled from sensor measurements. Given a factor graph, a potential way to perform the inference step is to convert it to a chordal Bayes Net using an exact inference technique (i.e. variable elimination) and then in turn, convert the Bayes Net to a tree like structure (i.e. junction tree) where inference can be made easier. However, this procedure can become ill-suited for real-time state estimation especially if the number of variables in the problem increases with time (i.e. as the robot navigates the environment). The *Bayes tree* (Kaess et al., 2011) was introduced to tackle this problem. It is a tree structure where nodes are formed from the maximal cliques of a chordal Bayes net. The Bayes tree is directed and maintains the conditional independences described by the original Bayes net. In addition, it allows for fast incremental inference which corresponds to simple tree editing.

[1]Robotics Institute, Department of Computer Science, Carnegie Mellon University, Pittsburgh, United States. Correspondence to: Mohamad Qadri <mqadri@andrew.cmu.edu>.

Published at the Differentiable Almost Everything Workshop of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. July 2023.

iSAM2 (Kaess et al., 2012) is an optimizer that leverages the Bayes tree to solve incremental (i.e. as the number of factors N grows continuously) inference problems formulated as a factor graph. In essence, it solves problems of the form:

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} \sum_{i=1}^{N} \frac{1}{2} ||g_i(\mathbf{x}_i) - z_i||^2_{\theta_i} \tag{1}$$

or equivalently the MAP inference problem:

$$\mathbf{x}^* = \arg\max_{\mathbf{x}} \prod_{i}^{N} \phi_i(\mathbf{x}_i) \tag{2}$$

where ϕ_i are the potentials assumed to take the form $\phi_i \propto \exp\left(-\frac{1}{2}||g_i(\mathbf{x}_i) - z_i)||^2_{\theta_i}\right)$, $\mathbf{x}$ is the vector of unknown states, $\mathbf{x}_i$ a subset of $\mathbf{x}$, and $\mathbf{z} = \{z_i\}$ are the sensor measurements. Batch solvers such as Levenberg-Marquardt perform the typical linearize-solve loop which quickly becomes prohibitively expensive as the size of the problem grows. iSAM2, on the other hand, leverages the Bayes tree and uses *fluid relinarization* and *partial state updates* making it the de-facto optimizer for online smoothing-based state estimation problems in robotics. However, these features also prevent iSAM2 from being differentiable (i.e. dependence on relinearization thresholds, inherent non-differentiable operations such as removal and re-insertion of tree nodes.)

On the other hand, the quality of the solution of eq.1 is largely dependent on a proper selection of the observation models $\{\theta_i\}$ which parameterize the joint or conditional distributions over states and measurements. However, due the non-differentiability of iSAM2, current methods to tune these parameters are generally sampling-based which are slow and may converge to poor optimas. In this work, we propose a gradient-based optimization-based approach to tune $\{\theta_i\}$ with iSAM2 in the loop using a direct tracking error loss. We compare our method with a state-of-the-art sampling-based method on a synthetic robot navigation example and show that our procedure converges order-of-magnitude faster to a better solution as measured by the tracking accuracy over unseen robot test trajectories.

2. Related Work

Early state estimation techniques such as the family of Kalman Filters (Julier & Uhlmann, 2004; Thrun et al., 2001)

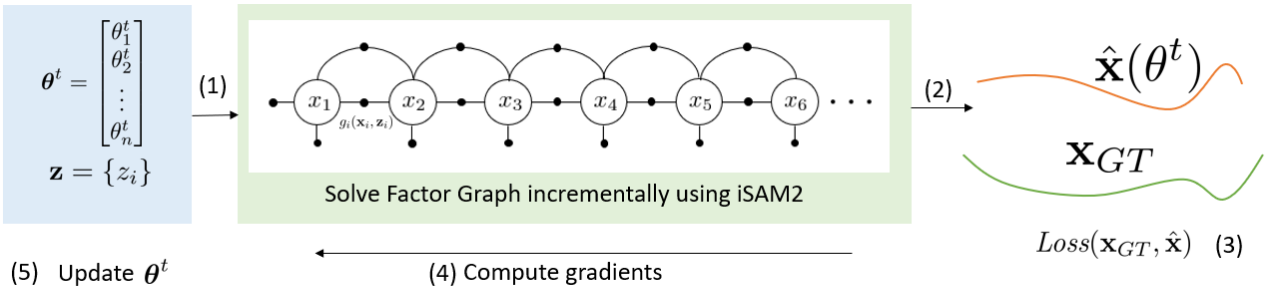

Figure 1. The parameter vector $\boldsymbol{\theta}^t$ serve as input to the least squares problem which is solved by iSAM2. Gradients of the solution with respect to the parameters are computed and used to update $\boldsymbol{\theta}^{t+1}$

rely on the Markov assumption to enable real-time performance. Recent algorithms have been proposed to make these filters differentiable (Kloss et al., 2021; Sun et al., 2016; Haarnoja et al., 2016). However, the inherent reliance on the Markov assumption and the inability to re-linearize past states can lead to convergence to poor solutions. Hence, state-of-the-art robotic state estimation algorithms moved to factor graph-based solutions which encode the inherent temporal structure avoiding the need to marginalize past states and providing methods to relinearize past estimates (Kaess et al., 2012; Qadri et al., 2022). However, observation models on factor graphs are either pre-specified and fixed (Lambert et al., 2019; Engel et al., 2014) or learned using surrogate losses independent of the graph optimizer or final tracking performance (Sodhi et al., 2021; Sundaralingam et al., 2019; Czarnowski et al., 2020). Methods that differentiate through the $\arg\min$ operator in eq. 1 by unrolling the optimizer can be used to learn these models (Yi et al., 2021; Jatavallabhula et al., 2020; Bechtle et al., 2021). However, these techniques are typically sensitive to hyperparameters such as the number of unrolling steps (Amos & Yarats, 2020) in addition to suffering from vanishing as well as high bias and variance gradients. Recently, a novel method *LEO* (Sodhi et al., 2022) took advantage of the probabilistic view offered by iSAM2 (as a solver of eq.2) to provide a way to learn observation models, with iSAM2 in the loop, by minimizing a novel tracking error. In essence, at every training iteration LEO samples trajectories from the posterior distribution (a joint Gaussian distribution over the states) and the deviation with respect to the ground truth trajectory is minimized using an energy-based loss. In this work, we use LEO as our baseline.

3. Method

In this work, we view the incremental optimizer (iSAM2) as a function $f : \mathcal{X} \times \boldsymbol{\theta} \rightarrow \mathcal{X}$ which takes initial estimates of the state $\mathbf{x} \in \mathcal{X} : \mathcal{M}_1 \times \ldots \times \mathcal{M}_n$ at timestep t (the number of states increases with time), as well as parameters $\boldsymbol{\theta} = \{\theta_i \mid \theta_i \in \mathcal{S}_{++}^{n_i}\}$ and returns an estimate of the optimal state $\mathbf{x}_* \in \mathcal{X}$ after performing N update steps. Here, $\mathcal{M}_i$ is a Lie Group (for example the special Euclidean group $SE(n)$) and $\mathcal{S}_{++}^{n_i}$ is the set of $n_i \times n_i$ positive definite matrices .

Our goal is to learn the parameters $\{\theta_i\}$ using gradient-based methods from observed ground truth robots trajectories $\mathbf{x}_{GT}$. Specifically, We consider the following inner-outer optimization procedure:

$$\text{Inner Loop:} \quad \hat{\mathbf{x}} = \arg\min_{\mathbf{x}} \sum_i \frac{1}{2} ||g_i(\mathbf{x}) - z_i||_{\theta_i}^2 \tag{3}$$

$$\text{Outer Loop:} \quad \min_{\boldsymbol{\theta}} L(\hat{\mathbf{x}}(\boldsymbol{\theta}), \mathbf{x}_{GT}) \tag{4}$$

Note that the gradient $\frac{\partial L}{\partial \boldsymbol{\theta}}$, requires an estimate of $\frac{\partial \hat{\mathbf{x}}}{\partial \boldsymbol{\theta}}$. Although iSAM2 includes different non-differentiable operations, the gradient of the solution with respect to the parameter vector $\frac{\partial \hat{\mathbf{x}}}{\partial \boldsymbol{\theta}}$ exists and can be computed by the implicit function theorem (Dontchev et al., 2009) as done in existing work in convex optimization (Amos & Kolter, 2017; Agrawal et al., 2019). Note that the original theorem consider functions operating on vector spaces. However, the theorem can readily be extended to other manifolds by applying the appropriate group operations.

The Implicit Function Theorem:
Let $\hat{\mathbf{x}} := \{\mathbf{x} \mid f(\mathbf{x}, \boldsymbol{\theta}) = 0\}$ *where* $\mathbf{x} \in \mathcal{X}$ *and* $\boldsymbol{\theta} = \{\theta_i \mid \theta_i \in \mathcal{S}_{++}^n\}$. *Let* f *be continuously differentiable in the neighborhood of* $(\hat{\mathbf{x}}, \boldsymbol{\theta})$ *namely* $\nabla_{\mathbf{x}} f(\hat{\mathbf{x}}, \boldsymbol{\theta})$ *be nonsingular. Then:*

$$\frac{\partial \hat{\mathbf{x}}}{\partial \boldsymbol{\theta}} = -\left(\frac{\partial f(\hat{\mathbf{x}}(\boldsymbol{\theta}), \boldsymbol{\theta})}{\partial \mathbf{x}}\right)^{-1} \frac{\partial f(\hat{\mathbf{x}}(\boldsymbol{\theta}), \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \tag{5}$$

The gradient in eq. 5 can be derived and computed analytically. However, we note that since the size of parameter vector $\boldsymbol{\theta}$ is typically small (for example, each θ_i has a maximum of 6 free parameters when working with elements in $SE(2)$), numerical differentiation proved to be efficient especially when coupled with parallelization on CPU.

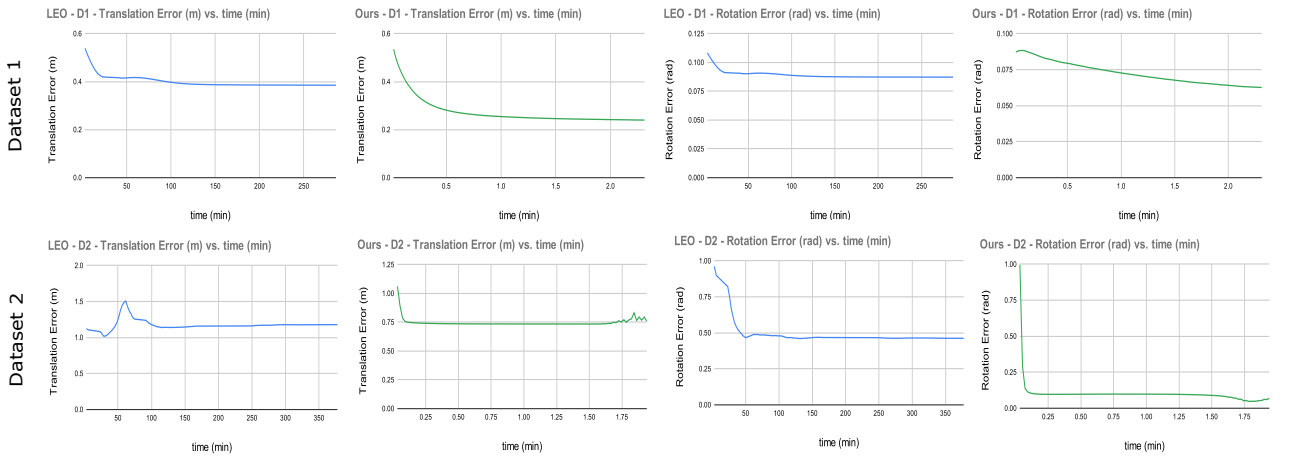

Figure 2. Convergence as a function of training time measured in minutes. Both methods were run for 100 training iterations. Our method converges after a few seconds while each iteration of LEO takes approximately 2.5 minutes. Additionally, our method converges to parameters that track the training trajectories more accurately.

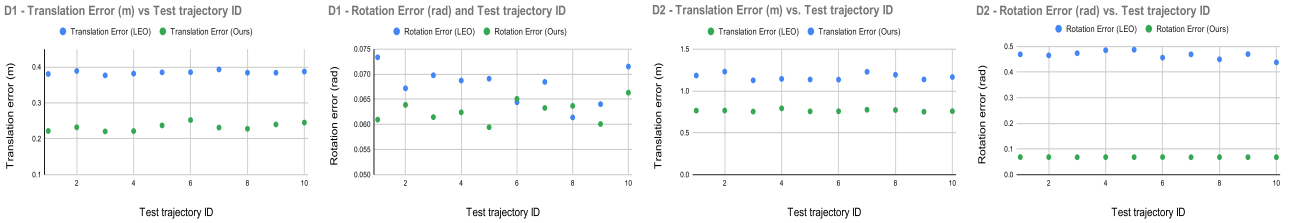

Figure 3. Accuracy on the test set. Our method converges to parameters that generalize and track the test trajectories more accurately.

3.1. Numerical Jacobians over Lie Groups

3.1.1. Jacobians on Vector Spaces

Given a multivariate function $f : \mathbb{R}^m \to \mathbb{R}^n$, the Jacobian is defined as the $n \times m$ matrix:

$$\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_m} \\ \vdots & \vdots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_m} \end{bmatrix} \tag{6}$$

where $\frac{\partial f}{\partial x_i} = \lim_{h \to 0} \frac{f(\mathbf{x}+h\mathbf{e}_i)-f(\mathbf{x})}{h}$ and $\mathbf{e}_i$ is the ith standard basis of $\mathbb{R}^m$.

3.1.2. Left Jacobians on Lie Groups

We can similarly define the left Jacobian of functions acting on manifolds $f : \mathcal{N} \to \mathcal{M}$ as the linear map from the Lie algebra $T_\epsilon(\mathcal{N})$ of $\mathcal{N}$ to $T_\epsilon(\mathcal{M})$, the Lie algebra of $\mathcal{M}$:

$$\frac{{}^{\epsilon}Df(\mathcal{X})}{D\mathcal{Y}} = \lim_{\tau \to 0} \frac{f(\tau \oplus \mathcal{Y}) \ominus f(\mathcal{Y})}{\tau} \tag{7}$$

$$= \lim_{\tau \to 0} \frac{\mathrm{Log}(f(\mathrm{Exp}(\tau) \circ \mathcal{Y}) \circ f(\mathcal{Y})^{-1})}{\tau} \tag{8}$$

where $\mathcal{Y} \in \mathcal{N}$, τ is a small increment defined on $T_\epsilon(\mathcal{N})$. The Exp operator map elements from a Lie Group to its algebra while the Log operator map elements from the algebra to the group. $\oplus$, $\ominus$, and $\circ$ are the plus, minus, and composition operators respectively (Sola et al., 2018) where:

$$\tau \oplus \mathcal{Y} = \mathrm{Exp}(\tau) \circ \mathcal{Y} \tag{9}$$

$$\tau = \mathcal{Y}_1 \ominus \mathcal{Y}_2 = \mathrm{Log}(\mathcal{Y}_1 \circ \mathcal{Y}_2^{-1});\ \mathcal{Y}_1, \mathcal{Y}_2 \in \mathcal{N} \tag{10}$$

In this work, $\mathcal{N} = \mathcal{S}^{n_1}_{++} \times \ldots \times \mathcal{S}^{n_m}_{++}$ and $\mathcal{M} = \mathcal{X}$. Additionally, we assume that each vector θ_i are the square root elements of some corresponding diagonal positive definite matrix $\Sigma_i \in \mathcal{S}^{n_i}_{++}$. i.e., we define the following map:

$$\theta_i = \left(\mathrm{diag}^{-1}(\Sigma_i)\right)^2 \in \mathbb{R}^{n_i} \tag{11}$$

Hence, $\tau \in \mathbb{R}^{n_i}$ and the operator $\oplus$ is the standard addition on vector space $\mathbb{R}^{n_i}$.

3.2. Tracking Loss

Let a parameter estimate $\boldsymbol{\theta}^t \in \mathbb{R}^m$. The outer loss is the regularized mean squared error between the estimated trajectory $\hat{\mathbf{x}}(\boldsymbol{\theta}^t)$ and the ground truth $\mathbf{x}_{GT}$. Let $\mathcal{D}$ be the training set, T be the total number of states in $\mathbf{x}_{\mathrm{GT}}$, D be the sum of

the lie algebra dimensions of all states, and $\lambda \in \mathbb{R}$:

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{2|\mathcal{D}|}\sum_{j=1}^{|\mathcal{D}|} ||\mathbf{vec}(\hat{\mathbf{x}}^j(\boldsymbol{\theta}^t) \ominus \mathbf{x}_{GT})||_2^2 + \lambda||\boldsymbol{\theta}^t||_2^2 \quad (12)$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = \frac{1}{|\mathcal{D}|}\sum_{j=1}^{|\mathcal{D}|} S(\hat{\mathbf{x}}^j(\boldsymbol{\theta}^t))^T \cdot \underbrace{\mathbf{vec}(\hat{\mathbf{x}}^j(\boldsymbol{\theta}^t) \ominus \mathbf{x}_{GT})}_{\in \mathbb{R}^{TD}} + 2\lambda\boldsymbol{\theta}^t \quad (13)$$

where $\mathbf{vec}$ is the vectorization operator and $S(\hat{\boldsymbol{x}}(\boldsymbol{\theta}^t)) \in \mathbb{R}^{TD\times m}$ is a matrix such that each row r is equal to:

$$S(\hat{\mathbf{x}}^j(\boldsymbol{\theta}^t))_r = \mathbf{vec}\left(\frac{\partial \hat{\mathbf{x}}^j}{\partial \theta_{ij}}\right) = \lim_{\tau_{ij}\to 0} \frac{\mathbf{vec}(\mathrm{Log}(\hat{\mathbf{x}}^j(\tilde{\boldsymbol{\theta}}^t) \circ \hat{\mathbf{x}}^j(\boldsymbol{\theta}^t)^{-1}))}{\tau_{ij}} \quad (14)$$

where $\tilde{\theta}_{ij}^t = \theta_{ij}^t + \tau_{ij}$ and $\tilde{\boldsymbol{\theta}}^t = \boldsymbol{\theta}^t$ otherwise. By the implicit function theorem, the gradient $\frac{\partial \hat{\mathbf{x}}}{\partial \theta_{ij}}$ exists and is estimated in eq. 14 using finite differencing by perturbing the parameter θ_{ij} by τ_{ij} . Parameters are then updated using gradient descent with learning rate α:

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t - \alpha \cdot \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} \quad (15)$$

4. Results

Table I. Comparison between LEO and Ours (final training runtime and error, test error statistics).

	LEO (D_1)	Ours (D_1)	LEO (D_2)	Ours (D_2)
Num training iterations	100	100	100	100
Parallelization enabled during training	yes	**no**	yes	**no**
Time per training iteration (min)	2.863	**0.025**	3.773	**0.022**
Num of training trajectories	5	5	5	5
Length of training trajectories	300	**100**	300	**100**
Avg training RMS translation error (m)	0.385	**0.236**	1.179	**0.730**
Avg training RMS rotation Avg error (rad)	0.087	**0.062**	0.467	**0.051**
Num test trajectories	20	20	20	20
Length of test trajectories	300	300	300	300
Avg test RMS translation error (m)	0.384	**0.238**	1.153	**0.733**
Avg test RMS rotation error (rad)	0.067	**0.063**	0.463	**0.050**

We use two synthetic planar (i.e. in $SE(2)$) robotic navigation datasets D_1 and D_2 consisting of GPS and odometric measurements. Each dataset uses a different set of parameters $\{\theta_{\text{GPS}}, \theta_{\text{odom}}\}_{D_i}$ to generate N trajectories (more details in appendix A.1). We use 5 training ground truth trajectories of length[1] 300 to train LEO[2] and 5 training trajectories of length 100 to train our method. We use a set of 20 unseen trajectories (all of length 300) to test the tracking performance given our final learned parameters (the testing set is fixed for both methods). For LEO, we enable multithreaded parallel trajectory sampling while do not use any form of parallelization with our technique.

[1]Length refers to the number of nodes in the graph optimizer

[2]We use the official implementation of LEO by Paloma et al.

At each training iteration, iSAM2 optimizes the inner loop (eq. 3) objective for both methods:

$$\hat{\mathbf{x}} = \arg\min_{\mathbf{x}} G(\boldsymbol{\theta}, \mathbf{x}; \mathbf{z}) = \arg\min_{\mathbf{x}} \frac{1}{2}\sum_i \Big(||g^{\text{gps}}(x_i) - z_i^{\text{gps}}||^2_{\theta_{\text{gps}}} + \frac{1}{2}||g^{\text{odom}}(x_{i-1}, x_i) - z_{i-1,i}^{\text{odom}}||^2_{\theta_{\text{odom}}} \Big) \quad (16)$$

Our method then updates the parameters $\boldsymbol{\theta}$ using eq.15 while LEO minimizes the following energy-based loss:

$$\mathcal{L}(\theta) = \frac{1}{|\mathcal{D}|}\sum_{(\mathbf{x}_{\text{gt}}^j, \mathbf{z}^j)\in\mathcal{D}} E(\boldsymbol{\theta}, \mathbf{x}_{\text{gt}}^j; \mathbf{z}^j) + \log \int_{\mathbf{x}} e^{-E(\boldsymbol{\theta}, \mathbf{x}; \mathbf{z}^j)} d\mathbf{x} \quad (17)$$

where $(\mathbf{x}_{\text{gt}}^j, \mathbf{z}^j)$ is a ground truth sample from the training set $\mathcal{D}$, the energy $E(\boldsymbol{\theta}, \mathbf{x}; \mathbf{z}) := G(\boldsymbol{\theta}, \mathbf{x}; \mathbf{z})$, and the integral is over the space of trajectories. Fig. 2 shows the training time and root mean squared error (RMSE) at each iteration. Fig. 3 shows the RMSE on the test set, and table I gives a summary of the results. We note that our method converges to parameters that lead to better tracking accuracy on all unseen test trajectories. In addition, while LEO needs to generate samples from a high dimensional probability distribution during training which is a time consuming process, our method generates gradients by directly comparing deviation from the training trajectories leading to order-of-magnitude faster training time. We additionally analyze the performance of both methods as a function of the training set size and provide the results in appendix A.2.

5. Conclusion and Future Work

We presented a gradient-based learning algorithm which estimates observation models with non-differentiable optimizers (iSAM2) in the loop for robotic state estimation. While current state of the art algorithms require sampling trajectories from the posterior distribution to bypass the non-differentiability of the optimizer, our technique learns parameters by formulating the problem as a bilevel optimization procedure where gradients are generated through numerical differentiation.

For future work, we want to extend our algorithm to learn parameters $\{\theta_i\}$, with iSAM2 in the loop, that are themselves functions of observations i.e. $\theta_i(z_i, \phi_i)$ where ϕ_i can, for example, be the weights of a jointly trained neural network. Indeed, the outputs of the network can be perturbed to approximate $\frac{\partial \hat{\mathbf{x}}}{\partial \theta_i}$ as proposed in this work and then simply chained with the gradient $\frac{\partial \theta_i}{\partial \phi}$ (obtainable from existing auto-differentiation packages such as PyTorch) to get the gradient of the optimized output trajectory with respect to network weights. We plan to compare the sample efficiency, training time, and generalization performance of our method against different baselines. Finally, we plan to train our algorithm on real robotics data and deploy our state estimator on real robotics platforms.

References

Agrawal, A., Amos, B., Barratt, S., Boyd, S., Diamond, S., and Kolter, J. Z. Differentiable convex optimization layers. *Advances in neural information processing systems*, 32, 2019.

Amos, B. and Kolter, J. Z. Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning*, pp. 136–145. PMLR, 2017.

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

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

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

Dontchev, A. L., Rockafellar, R. T., and Rockafellar, R. T. *Implicit functions and solution mappings: A view from variational analysis*, volume 11. Springer, 2009.

Engel, J., Schöps, T., and Cremers, D. Lsd-slam: Large-scale direct monocular slam. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part II 13*, pp. 834–849. Springer, 2014.

Haarnoja, T., Ajay, A., Levine, S., and Abbeel, P. Backprop kf: Learning discriminative deterministic state estimators. *Advances in neural information processing systems*, 29, 2016.

Jatavallabhula, K. M., Iyer, G., and Paull, L. slam: Dense slam meets automatic differentiation. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2130–2137. IEEE, 2020.

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

Kaess, M., Ila, V., Roberts, R., and Dellaert, F. The bayes tree: An algorithmic foundation for probabilistic robot mapping. In *Algorithmic Foundations of Robotics IX: Selected Contributions of the Ninth International Workshop on the Algorithmic Foundations of Robotics*, pp. 157–173. Springer, 2011.

Kaess, M., Johannsson, H., Roberts, R., Ila, V., Leonard, J. J., and Dellaert, F. isam2: Incremental smoothing and mapping using the bayes tree. *The International Journal of Robotics Research*, 31(2):216–235, 2012.

Kloss, A., Martius, G., and Bohg, J. How to train your differentiable filter. *Autonomous Robots*, 45(4):561–578, 2021.

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

Qadri, M., Sodhi, P., Mangelson, J. G., Dellaert, F., and Kaess, M. Incopt: Incremental constrained optimization using the bayes tree. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6381–6388, 2022. doi: 10.1109/IROS47612.2022.9982178.

Sodhi, P., Kaess, M., Mukadam, M., and Anderson, S. Learning tactile models for factor graph-based estimation. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 13686–13692. IEEE, 2021.

Sodhi, P., Dexheimer, E., Mukadam, M., Anderson, S., and Kaess, M. Leo: Learning energy-based models in factor graph optimization. In *Conference on Robot Learning*, pp. 234–244. PMLR, 2022.

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

Sun, W., Venkatraman, A., Boots, B., and Bagnell, J. A. Learning to filter with predictive state inference machines. In *International conference on machine learning*, pp. 1197–1205. PMLR, 2016.

Sundaralingam, B., Lambert, A. S., Handa, A., Boots, B., Hermans, T., Birchfield, S., Ratliff, N., and Fox, D. Robust learning of tactile force estimation through robot interaction. In *2019 International Conference on Robotics and Automation (ICRA)*, pp. 9035–9042. IEEE, 2019.

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

Yi, B., Lee, M. A., Kloss, A., Martín-Martín, R., and Bohg, J. Differentiable factor graph optimization for learning smoothers. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1339–1345. IEEE, 2021.

A. Appendix

A.1. Additional Details on Experimental Setup

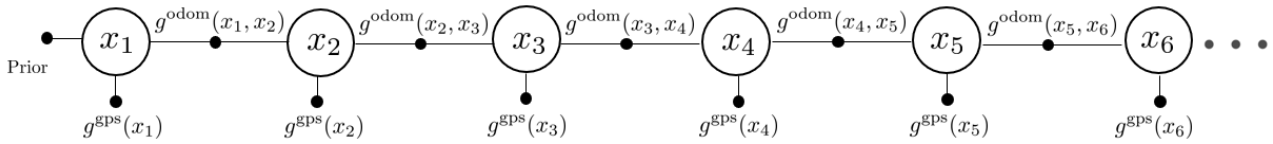

Figure 4. The factor graph used to generate the synthetic robot navigation trajectories.

Figure 4 shows the structure of the factor graph used to generate the synthetic robot navigation trajectories. A unary GPS factor $g^{\text{gps}}(x_i)$ is added to each pose x_i while a binary odometry factor $g^{\text{gps}}(x_i, x_{i-1})$ is specified between poses. To simulate realistic robot navigation trajectories for each of datasets D_1 and D_2, Gaussian noise $\sim \mathcal{N}(0, \theta_{\text{odom}})$ is injected to ground truth relative odometry measurements while Gaussian noise $\sim \mathcal{N}(0, \theta_{\text{gps}})$ is added to absolute ground truth GPS measurements.

A.2. Additional Experiments

We analyze our method and LEO as a function of the training set size. Tables II and III show the average training and testing RMS errors on datasets D_1 and D_2 respectively. We note that our method possesses a much faster training time as well as learns parameters that generalize better to unseen test trajectories regardless of the training set size.

Table II. Dataset D_1 - Comparison between LEO and Ours (final training runtime and error, test error statistics).

Num trajectory in training set	1		5		10		20		30	
Algorithm	LEO	Ours	LEO	Ours	LEO	Ours	LEO	Ours	LEO	Ours
Num training iterations	100	100	100	100	100	100	100	100	100	100
Parallelization enabled during training	yes	**no**	yes	**no**	yes*	**no**	yes*	**no**	yes*	no
Time per training iteration (min)	0.937	**0.005**	2.863	**0.025**	4.370	**0.045**	4.899	**0.091**	7.56	**0.136**
Average training RMS translation error (m)	0.379	**0.257**	0.385	**0.236**	0.383	**0.243**	0.386	**0.243**	0.385	**0.236**
Average training RMS rotation Avg error (rad)	0.071	**0.060**	0.087	**0.062**	0.076	**0.063**	0.079	**0.068**	0.079	**0.064**
Num test trajectories	20	20	20	20	20	20	20	20	20	20
Length of test trajectories	300	300	300	300	300	300	300	300	300	300
Average test RMS translation error (m)	0.385	**0.237**	0.384	**0.238**	0.385	**0.238**	0.384	**0.240**	0.384	**0.235**
Average test RMS rotation error (rad)	0.067	**0.059**	0.067	**0.063**	0.067	**0.063**	**0.067**	0.069	0.067	**0.066**

Table III. Dataset D_2 - Comparison between LEO and Ours (final training runtime and error, test error statistics).

Num trajectory in training set	1		5		10		20		30	
Algorithm	LEO	Ours	LEO	Ours	LEO	Ours	LEO	Ours	LEO	Ours
Num training iterations	100	100	100	100	100	100	100	100	100	100
Parallelization enabled during training	yes	**no**	yes	**no**	yes*	**no**	yes*	**no**	yes*	no
Time per training iteration (min)	0.934	**0.004**	3.773	**0.022**	4.403	**0.038**	4.901	**0.075**	6.96	**0.113**
Average training RMS translation error (m)	1.129	**0.715**	1.179	**0.730**	1.090	**0.731**	1.057	**0.718**	1.104	**0.761**
Average training RMS rotation Avg error (rad)	0.443	**0.056**	0.467	**0.051**	0.421	**0.055**	0.407	**0.056**	0.429	**0.045**
Num test trajectories	20	20	20	20	20	20	20	20	20	20
Length of test trajectories	300	300	300	300	300	300	300	300	300	300
Average test RMS translation error (m)	1.132	**0.728**	1.153	**0.733**	1.583	**0.735**	1.043	**0.734**	1.093	**0.771**
Average test RMS rotation error (rad)	0.447	**0.051**	0.468	**0.050**	0.419	**0.055**	0.402	**0.056**	0.427	**0.045**

For LEO, we faced compute memory problems on larger training sets (indicated by an asterix in tables II and III) which required us to decrease the number of threads and samples (needed to compute the loss in eq. 17). The values of the hyperparameters used per training set size are specified in table IIII.

Table IIII. Hyperparameters used to train LEO per training set size

Num trajectory in training set	1	5	10	20	30
Number of threads	4	4	3	2	2
Number of samples generated per training trajectory per iteration	10	10	8	4	4