Theseus:
A Library for Differentiable Nonlinear Optimization

Appendix

A   Contributions

The contributions of the authors are as follows.

Luis Pineda led the engineering of the project, developed and implemented the core API, differentiable nonlinear solvers, motion planning example and tutorials, standard and autodiff cost functions, and backward mode experiments, coordinated with sub-teams to help design, implement, integrate and review of all aspects of the code and evaluations, wrote the paper.

Taosha Fan developed and implemented differentiable Lie groups, automatic vectorization, `functorch` support, pose graph optimization example, performance evaluations and benchmarking, helped with API design, bug fixes, did code reviews, wrote the paper.

Maurizio Monge developed and implemented all sparse linear solvers, `BaSpaCho` solver and bundle adjustment example, batching, differentiation and custom C++/CUDA backends for sparse solvers, did code reviews, wrote sparse solver and bundle adjustment sections in the paper.

Shobha Venkataraman helped develop and implement the motion planning example, helped with API design, wrote several tutorial notebooks.

Paloma Sodhi developed and implemented the tactile state estimation example, energy based learning and covariance sampling example, helped design learning API.

Ricky T. Q. Chen developed and implemented the direct loss minimization backward mode, wrote its section in the paper.

Joseph Ortiz developed and implemented the homography example, helped write its section in the paper, did some bug fixes, helped with the implicit backward mode theory.

Daniel DeTone prototyped the homography example, helped develop its final version, wrote its section in the paper.

Austin Wang implemented differentiable forward kinematics support, wrote its section in the paper.

Stuart Anderson advised on the project, managed and supported research engineers, helped edit the paper.

Jing Dong advised on the project, helped design the core API and performance evaluations, did code reviews, helped edit the paper.

Brandon Amos advised on the project, developed and implemented implicit and truncated backward modes for nonlinear solvers, helped implement implicit backwards for linear solvers, helped design and analyze backward mode experiments, did code reviews, wrote the backward modes sections and helped edit the paper.

Mustafa Mukadam led the project, set the vision and research direction, created and steered the team, provided guidance on all aspects including feature prioritization, API design and implementation, formulated evaluations and experiments, did code reviews, wrote the paper.

B   Simple example description

In this section, we describe the example in Listing 1 in more detail. The example considers fitting the curve $y = v e^x$ to a dataset of N observations $(x, y) \sim \mathcal{D}$. A standard way to solve this is to minimize the least squares objective Eq. (1) with residuals $r_i(\hat{v}) := y^{(i)} - \hat{v} e^{x^{(i)}}$, for $i = 1, ..., N$, and where $\theta := \hat{v}$. We can model this in `Theseus` with a single `CostFunction` that computes the N-dimensional vector $R(\hat{v})$ of all residuals as a function of a single optimization `Variable` $\hat{v}$ and two auxiliary variables, x and y.

The code implementing this problem starts by creating uniquely named `Variable` containers in lines 3-5. We then create an objective with the cost function (lines 11-15). We use a `CostFunction` of type `AutoDiffCostFunction` (line 12), which relies on `torch.autograd` and vectorization via `functorch` [101] to automatically compute the residual Jacobians used by the inner optimizer (see App. C). `AutoDiffCostFunction` requires providing an error function that receives optimization variables and auxiliary variables (defined in lines 7-9), and returns `torch` tensors computing the (unweighted) residual. Although not required by this problem, we also illustrate how to add a cost weight to the residuals by including a `ScaleCostWeight`, which simply scales all residuals in this cost function by a scalar (1.0 in this case). Finally, we encapsulate the objective and a Gauss-Newton optimizer into a differentiable `TheseusLayer` in line 16.
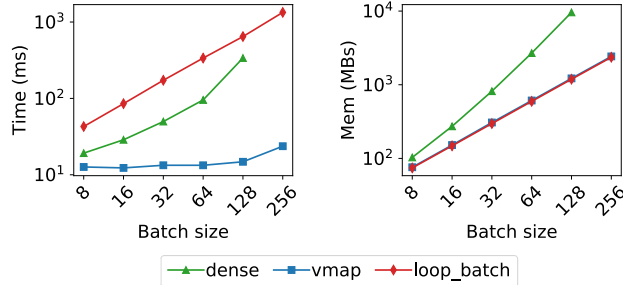
Figure 6: `AutoDiffCostFunction` time (left) and memory (right) consumption in the homography estimation example for different autograd modes and batch sizes.

To illustrate how to differentiate through this layer, we perturb the x values in the dataset so that it becomes impossible to recover the correct value of v from NLS optimization alone, and then define $\phi := x$ and $L(\theta^*(\phi)) := (\hat{v}^*(x) - v)^2$. Thus, the outer loop optimization corrects the x tensor so that the solution of the inner loop matches the true value of v.

In the code, the outer parameter is defined in line 18, with initial value for ϕ set to a perturbed function of the true x, while the outer loss is defined in line 24. In lines 21-23 we solve the NLS problem, by calling `forward()` with the current value of ϕ as the value to set for auxiliary variable named "x", and an initial value $v = 1$ for the optimization variable named "v" (with a required batch dimension in the input); we also set the backward mode to `Implicit`. The optimum found can be recovered as a tensor by getting key "v" of the dictionary returned by `forward`, which we then use to compute the outer loss in line 24. Finally, outer loop optimization is done via `torch`'s well-known autograd engine, in lines 25-26, using the Adam optimizer [102] for ϕ, defined in line 19.

C   `AutoDiffCostFunction` autograd modes

In this section, we evaluate the effect of the three different autograd modes we provide for automatically computing the jacobians for `AutoDiffCostFunction`:

- **dense** uses `torch.autograd.functional.jacobian`, which computes a dense jacobian that includes cross-batch derivatives; i.e., the derivative of j-th batch output with respect to variables in the i-th batch input. Since we only need per-sample gradients, we slice the result of this operation.

- **loop_batch** is also based on `torch.autograd.functional.jacobian`, but we manually loop over the batch before each call so that we obtain only per-sample gradients.

- **vmap** mode uses `functorch.vmap` [101] to compute per-sample gradients in a vectorized manner.

For evaluation, we use the homography example described in App. E.5, which uses `AutoDiffCostFunction` and can have significant memory requirements when computing the jacobians. Fig. 6 illustrates the advantages of using **vmap** over the other two modes, both in terms of compute time and memory. The **loop_batch** mode has similar memory requirements to **vmap**, but the compute time is significantly slower than the other two methods. Finally, **dense** mode has substantially more memory requirements than the two other methods (up to an order of magnitude higher), and runs out of memory for the largest batch size 256 used in this experiment. For a batch size of 128, **vmap** results in a speedup of 22x over the next best method (~ 15ms vs. ~ 337ms for **dense**), and almost 8x less memory (~ 1.2GBs vs. ~ 9.6GBs).

D   Differentiable Lie group details

While differentiation on the Euclidean space is straightforward, it remains challenging to do so on the non-Euclidean manifolds. In this section, we provide details about how to compute the derivatives on the tangent space of Lie groups using the projection operator [58]. The implementation of the projection operator is essential for automatic differentiation on the tangent space of Lie groups.

Suppose $F(g)$ is a function of $g \in G$ where G is a matrix Lie group and $\tau(\xi)$ a retraction map of G. For notational simplicity, let $\nabla_g F(g)$ denote the Euclidean gradient of $F(g)$ and $T_e G$ the Lie algebra of G. Following [103], the gradient on the tangent space of Lie group is a linear operator $\mathrm{D}_g F(g)$ such that

$$\mathrm{D}_g F(g) \cdot \xi = \left.\frac{\partial}{\partial s}\right|_{s=0} F\big(g\tau(s \cdot \xi)\big) \tag{5}$$

	Objective Value					
	Sphere	Torus	Cubicle	Rim	Grid	Garage
Initial	8.437×10^2	1.234×10^4	1.622×10^6	1.924×10^7	4.365×10^4	7.108×10^{-1}
Final	6.805×10^2	1.212×10^4	1.455×10^3	4.157×10^4	4.218×10^4	6.342×10^{-1}

Table 1: Initial and final objective values of `Theseus` on 3D benchmark datasets with PGO example.

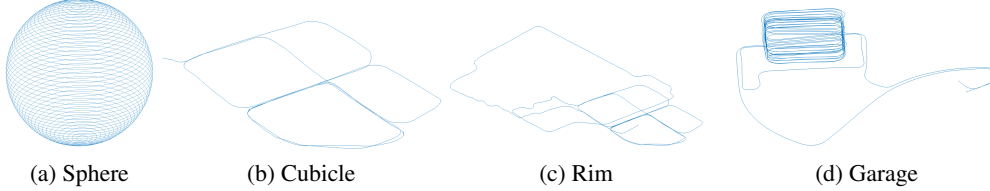(a) Sphere   (b) Cubicle   (c) Rim   (d) Garage

Figure 7: Qualitative results of `Theseus` on 3D benchmark datasets with PGO example.

holds for any Lie algebra elements $\xi \in T_e G$. As a result of the chain rule, the right-hand side of the equation above is equivalent to

$$\left.\frac{\partial}{\partial s}\right|_{s=0} F\big(g\tau(s \cdot \xi)\big) = \nabla_g F(g) \cdot \left.\frac{\partial}{\partial s}\right|_{s=0} g\tau(s \cdot \xi) = \nabla_g F(g) \cdot g\xi \quad (6)$$

where the last equality results from properties of the retraction map. Then, we conclude from Eqs. (5) and (6) that

$$D_g F(g) \cdot \xi = \nabla_g F(g) \cdot g\xi. \quad (7)$$

Therefore, there exists a projection operator $\text{proj}_g(\cdot)$ such that

$$D_g F(g) = \text{proj}_g\big(\nabla_g F(g)\big) \quad (8)$$

for any gradients on the tangent space of Lie group and their corresponding Euclidean gradients [58]. Furthermore, note that the projection operator $\text{proj}_g(\cdot)$ is a linear operator depending on $g \in G$ and can be computed in closed form.

E   Example application details

E.1   Pose graph optimization

Pose graph optimization (PGO) [60, 104, 105] is the problem of recovering unknown poses of SE(2) and SE(3) from the noisy relative pose measurements. Pose graph optimization has extensive applications in robotics [35], computer vision [106], computational biology [107], sensor networks [108], etc. In pose graph optimization, we represent unknown poses as vertices and relative pose measurements as edges. Then, it is possible to compute the relative pose errors for each pair of neighboring vertices such that a nonlinear least-squares optimization problem can be formulated for pose estimation. A more detailed introduction to pose graph optimization can be found in [35, 60, 104, 105].

`Theseus` includes a differentiable and coordinate-independent version of the relative pose errors with which it is straightforward to solve pose graph optimization. We evaluated `Theseus` on the simulated Cube dataset and a number of benchmark datasets for pose graph optimization [35, 60]. The Cube dataset simulates the 3D odometry of a robot with varying numbers of poses, loop closure probabilities, and loop closure outlier ratios, which is used to profile the time and space complexities of the forward and backward passes in `Theseus`. Furthermore, the benchmark datasets indicate that `Theseus` is capable of solving large-scale differentiable nonlinear optimization problems with comparable accuracy and efficiency to existing state-of-the-art solver like `Ceres` [37].

`Theseus` and `Ceres` attain the same objective values for all the evaluated benchmark datasets [35, 60] using the chordal initialization [109]. The inital/final objective values and qualitative results for some benchmark datasets are shown in Table 1 and Fig. 7, respectively.

E.2   Tactile state estimation

Recent work [28] explored the use of NLS optimization with learned tactile sensor observations for tactile pose estimation. The goal is to incrementally estimate sequences of object poses that are moved by a robotic hand equipped with a DIGIT tactile tensor [61]. The key insight of Sodhi
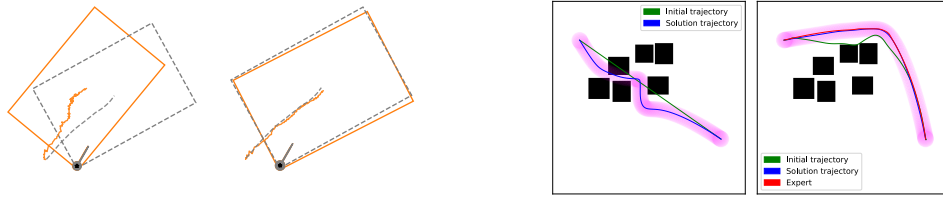
Figure 8: Examples of differentiable tactile state estimation and differentiable motion planning with Theseus. **Left:** Pose estimates before and after learning in the tactile state estimation example. Grey color indicates ground truth, and orange the estimate. The plot shows the trajectory as a curve, and the rectangle indicates the last object pose in the trajectory. **Right:** Trajectories generated by the planner before and after learning. The learned model generates initial trajectories for the optimizer to match an expert with only a few optimization steps.

et al. [28] is to use learning to transform high dimensional tactile observations into relative poses between measurement pairs. Once relative poses are available, the sensor data can be integrated into an optimization problem that solves for object poses. The objective includes four types of cost functions. One penalizes inconsistencies with the measurement coming from the learned observation model. A second one encourages the predicted poses to be consistent with a quasi-static physics model [110]. A third type adds geometric constraints by penalizing intersections between the end effector and the object using a signed distance field. Finally, a fourth cost function incorporates pose priors from a camera.

In Theseus, we implement an offline and differentiable version of the tactile state estimation problem above, using a dataset of 63 trajectories of length 25 with known ground truth poses provided by the authors of [28]; we used 56 of these as training set and the other 7 as a test set. Optimization variables are object and end effector poses (modeled as SE(2) groups) for each point in the trajectory, and the outer loss objective is the difference between the optimized object poses and the ground truth in the dataset. The learnable component corresponds to the relative pose model, using a pre-trained encoder, and finetuning the final layer via end-to-end learning through the inner loop optimization. This approach is similar to how the tactile measurements model was trained in [111], with the two main differences being that we do not use an energy-based formulation and instead directly differentiate through the inner optimizer, and we also do not consider an incremental setting. Fig. 8 (left) shows an example of estimated trajectories before and after learning.

E.3 Bundle adjustment

Bundle adjustment is the problem of optimizing a 3D reconstruction formed by a set of camera images and a set of landmarks observed and matched across the images. In every camera image a 2D coordinate is identified for the position of all observed landmarks, and the problem is initialized with an estimate of the positions of the landmarks and the camera poses. We call *reprojection error* the image-offset between where the landmark was detected on the image, and the reprojection of the landmark according to the current parameter estimation. The optimization problem consists of simultaneously tweaking the cameras poses and landmark positions, while minimizing the square-sum of all the reprojection errors; see [8, 62] for in depth exploration on bundle adjustment and its state-of-the-art.

We provide a bundle adjustment application example in Theseus, adopting the same data format of [62], with functions to generate synthetic dataset, as well as load/save open source datasets. To test bundle adjustment in a differentiable optimization setting, we add soft-kernels to the reprojection errors and setup as outer loop parameter the radius of the soft-kernel, which represents the confidence radius for reprojection errors with respect to possible outlier observations. We use as outer loss the average frame pose error from a ground truth value, such that the outer loop's task is to set the radius to a value that will make the bundle adjustment problem set the ideal soft loss radius value.

E.4 Motion planning

NLS optimization can also be used for motion planning in robotics [5], where the objective variables are robot poses and velocities on a set of discrete time steps. Cost functions include terms representing smoothness constraints modeling forward kinematics, collision avoidance penalties, and boundary conditions on start and goal states. An end-to-end differentiable version of this formulation was proposed by Bhardwaj et al. [20], where a neural model predicts state-dependent cost weights for each step in the path, and the outer loss encourages the inner loop optimization to produce paths
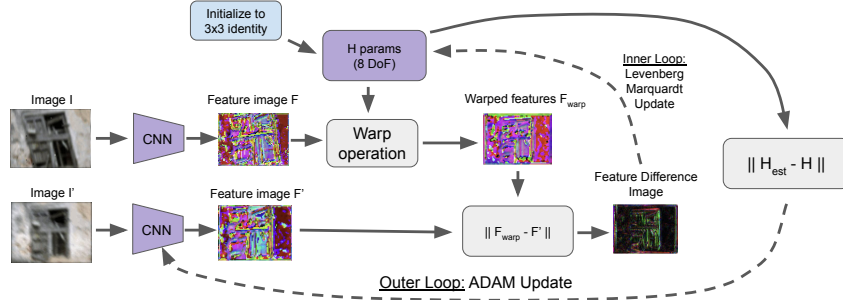
Figure 9: **Learning Robust Image Features for Homography Estimation with Theseus.** An inner loop optimization problem aligns two images via a feature-metric Levenberg-Marquardt optimization using features computed from a CNN. The outer loop uses Adam to update the weights the of the CNN that best minimize the final homography error.

matching an expert in a dataset of trajectories. As part of Theseus, we include differentiable versions of cost functions like smoothness and collision in [5], and an example of how to setup end-to-end differentiation for optimization variable initialization. That is, the model proposes initial trajectories for the optimizer, and the outer loss is set to a weighted sum of two terms, one computing closeness to the expert trajectory, and another equal to the inner loop's objective after only 2 iterations. This loss encourages the model to produce high-quality "proposals" that converge to good quality solutions quickly; an example of before/after training initial trajectories is illustrated in Fig. 8.

E.5 Homography estimation

A homography, also known as a projective transformation, is a linear mapping between a 2D point in one image x to a point in another image x', defined by a 3×3 matrix H, written as $x \sim Hx'$, where $\sim$ defines the equivalence up to scale. In addition to representing linear transformations across 2D images, the homography is also a valid approximation of the motion of 2D points observed from camera images in 3D scenes in certain scenarios such as (1) rotation-only motion between cameras (2) when the scene is planar and (3) when the scene structure is far from the camera.

One approach to solving for the parameters of the homography is through iterative optimization via dense alignment of RGB image pixels in the image through fast second order optimization methods, as is done in Lucas-Kanade optical flow algorithm [112, 113]. This approach is also known as photo-metric alignment. Though this technique performs well in many scenarios, photo-metric alignment struggles when the lighting in the scene changes significantly, because it assumes that the brightness of a pixel is constant across different views. Feature-metric optimization is an extension to photo-metric optimization that works by first passing the image I through a feature extractor function $f(\cdot)$, such as a convolutional neural network parameterized by weights w, $F = f(I; w)$. This function generates a feature map $F \in \mathbb{R}^{C \times H \times W}$, where C is a high dimensional channel number like 32, and H and W represent the image height and width respectively. In feature-metric alignment, the alignment is done at the feature map level, rather than the RGB image level.

One important question when designing a feature-metric optimization algorithm, is how to obtain the weights w that define the feature extractor. One approach used in works such as [63] uses an off-the-shelf CNN which has been trained for image classification. One benefit of using Theseus for such as task is that the learning problem can be written without deriving analytical gradients, making it much easier to rapidly prototype and explore various formulations. In our example, we demonstrate a use-case of Theseus by performing end-to-end training of a two-layer CNN using gradients obtained through the homography optimization. A high level diagram of this learning problem is presented in Fig. 9. We optimize a dense feature-metric mean-squared error term in the inner loop and a four-corner homography error in the outer loop. The four-corner error is a simple measure that computes the L2 distance of four corners of the image after being transformed by the estimated and ground truth homography, as is used in [114] as the output parameterization.

F BaSpaCho: Batched Sparse Cholesky

In this section, we provide more details for our open-source novel BaSpaCho solver (https://github.com/facebookresearch/baspacho). BaSpaCho implements the *supernodal* Cholesky algorithm [70] to achieve state-of-the art performance by exploiting dense operations via BLAS/cuBLAS. The heuristics for clustering in the supernodal algorithm evaluate the trade-offs of fragmentation in
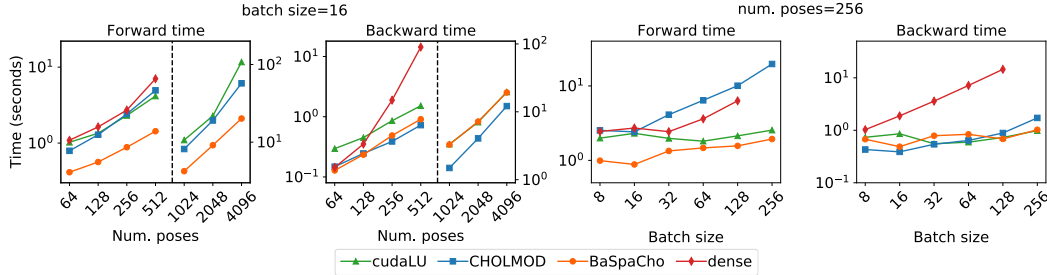
Figure 10: Forward/backward times of `Theseus` with sparse and dense solvers on different PGO problem scales.
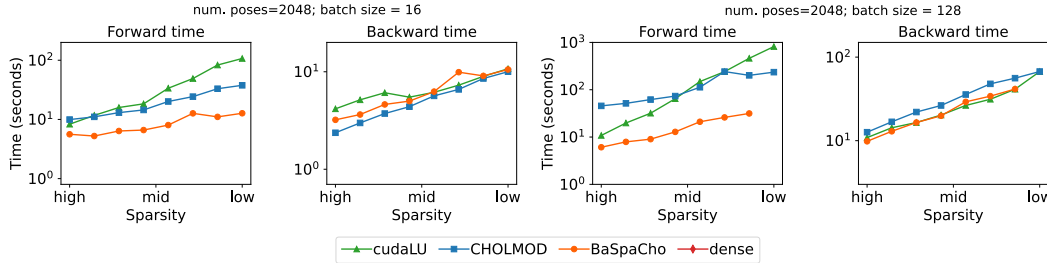

Figure 11: Forward and backward times of `Theseus` with sparse and dense solvers on PGO problems with 2048 poses and different levels of sparsity.

sparse matrices against denser matrices with explicit zero-fill. These heuristics use a computation model that takes into account the architecture (batched/CPU/GPU) that can impact preference towards sparser or denser operations, and allows for further fine-tuning and customization. Apart from a minimal memory allocation needed for the symbolic factorization, BaSpaCho does not own any allocated memory allowing the user to fully manage memory arrays. This allow us to temporarily offload GPU arrays representing factorized matrix data to the CPU when necessary. Unlike existing solvers, BaSpaCho exposes lightweight random accessors that allows the user to read and write matrix blocks in the numeric factor data. This facilitates easy bookkeeping needed by optimization methods which often re-implement block-sparse matrix structures and convert between different matrix formats in order to invoke sparse solvers like CHOLMOD.

G   Benchmark details and additional results

In this section, we present more profiling results for forward and backward pass of `Theseus`, using the same setup as Sec. 5.1. For evaluation, we used the cube datasets of PGO (see App. E.1) with different numbers of poses, batch sizes, and levels of sparsity. In addition to the forward and backward times as a function of the numbers of poses and batch sizes reported in Sec. 5.1, we further report and analyze the memory usage of `Theseus` with different solvers (cudaLU, CHOLMOD, BaSpaCho and dense) in various settings.

G.1   Forward and backward times with smaller batch size and number of poses

We profile PGO using different linear solvers (cudaLU, CHOLMOD, BaSpaCho and dense) for fixed batch size of 16 and number of poses of 256. The setup is the same as that in Sec. 5.1 except that fixed batch size and number of poses are smaller. Fig. 10 shows the average time of a full forward and backward pass. Similar to Fig. 3 with larger fixed batch size and number of poses, the sparse solvers are faster than dense. For the smallest problem considered (64 poses, 16 batch size), the total sum of average forward and backward times are 1.32s (cudaLU), 0.94s (CHOLMOD), 0.54s (BaSpaCho), 1.23s (dense) per batch. Increasing to 128 poses makes the sparse solvers noticeably faster than dense: 1.98s (dense) vs 1.80s (cudaLU), 1.53s (CHOLMOD), and 0.80s (BaSpaCho). As the problem scale increases, the gap between the sparse and dense solvers widens: for the largest problem solvable with dense (512 poses, 16 batch size) the average total times are 5.68s (cudaLU), 5.67s (CHOLMOD), 2.34s (BaSpaCho), and 21.37s (dense). The speedup over dense is ∼3.7x for cudaLU and CHOLMOD, and ∼9.1x for BaSpaCho.

G.2   Forward and backward times with respect to sparsity

We study the forward and backward pass times of `Theseus` with sparse and dense solvers for different levels of sparsity using the synthetic Cube dataset. In PGO, loop closure probability represents how
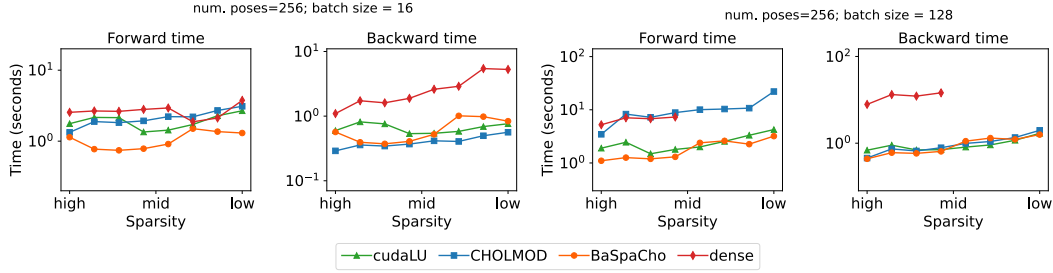
24

Figure 12: Forward and backward times of `Theseus` with sparse and dense solvers on PGO problems with 256 poses and different levels of sparsity.

likely a pose has a loop closure edge connected to the other poses, and thus, greater loop closure probability yields a less sparse Hessian. We use loop closure probabilities from 0.05 to 0.40 in increments of 0.05, to indicate the level of sparsity for Cube datasets from high (0.05) to low (0.40).

The average forward and backward times of `Theseus` on PGO problems with different levels of sparsity for fixed numbers of poses are shown in Fig. 11 for 2048 poses and in Fig. 12 for 256 poses. In both figures, the left two plots are with 16 batch size and the right two plots are with 128 batch size. As expected, it takes more time in most cases for PGO problems with lower sparsity. Since dense does not exploit the sparsity of optimization problems when solving the linear systems, forward pass of dense takes almost the same amount of time regardless of the levels of sparsity. There is still some overhead for dense as sparsity decreases, because more loop closure edges implies more cost function terms in the objective, so putting together the approximate Hessian is computationally more expensive.

G.3 Scalability of `Theseus`

In addition to forward and backward times in Sec. 5.1 and Apps. G.1 and G.2, we analyze the scalability of `Theseus` with different linear solvers (cudaLU, CHOLMOD, BaSpaCho and dense) following a similar setup to evaluation in Figs. 3 and 10.

We profile PGO with various numbers of poses from 64 to 8192 and batch sizes from 8 to 1024 in increments of power

Figure 13: Largest PGO problems `Theseus` scales to for different numbers of poses and batch sizes.

of 2. Fig. 13 shows the maximum batch sizes solvable for given numbers of poses (left) and the maximum numbers of poses solvable for given batch sizes (right). In Fig. 13, it can be seen that dense only scales to small PGO problems due to the memory limitation and fails to solve any PGO problems with 4096 poses or more, even with batch size of 1. In contrast, cudaLU, CHOLMOD and BaSpaCho successfully solve PGO problems with 8192 poses for a batch size of 32 (cudaLU, BaSpaCho) and 256 (CHOLMOD). As discussed in Sec. 5.1, cudaLU and BaSpaCho require extra GPU memory to solve linear systems, whereas CHOLMOD has all computation run on CPU, and thus can solve larger DNLS problems than cudaLU and BaSpaCho.
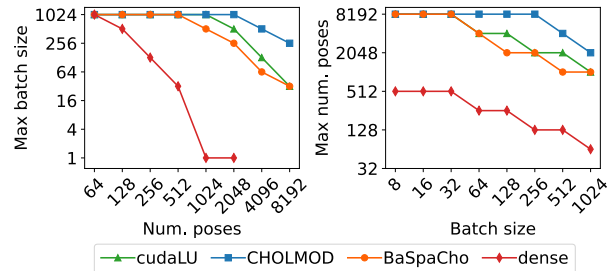
G.4 Comparison with `Ceres` on smaller batch size and number of poses

In addition to Sec. 5.2, we follow the same setup to compare `Theseus` as a stand-alone NLS optimizer with `Ceres` for PGO problems with a smaller fixed batch size of 16 and number of poses of 256. Fig. 14 shows the speedup of `Theseus` compared to `Ceres` (black dotted line). Similar to Fig. 4, `Ceres` is faster for PGO problems for small batch sizes and numbers of poses, and `Theseus` is faster as the problem scale increases.
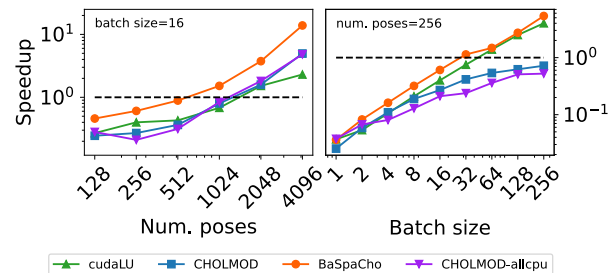

Figure 14: Speedup of `Theseus` (forward pass) over Ceres (black dashed) on different PGO problem scales.

25

H Backward mode details, additional results, and derivations

H.1 Experimental details

In Sec. 5.3 we use the tactile state estimation example to evaluate the performance of different backward modes. As mentioned in App. E.2, the dataset consists of 63 trajectories of length 25, 56 of which we use for training and 7 for test. We use a batch size of 8 and train for 100 epochs, resulting in 700 batches for averaging time and memory results. For the inner loop, we use Gauss-Newton with a step size of 0.05; in the test set we run the inner loop for 50 iterations, regardless of the number used during training. For the outer loop, we use the Adam optimizer with a learning rate of 10^{-4}, decayed exponentially by a factor of 0.98 after every epoch.

H.2 Additional results

Fig. 15 (left) shows the peak memory consumption during the forward pass. We observe the same trend from the backward pass (Fig. 5, center right), where Unroll's memory consumption increases linearly with the number of inner loop iterations, while for the other methods it remains constant. Implicit has the lowest peak memory requirement ($\sim$22MBs), followed by DLM ($\sim$29MBs).

Fig. 15 (center, right) also shows training curves for all methods. We observe that, despite higher performance in the test set, Implicit is more unstable during training and oscillates between low and high values; this suggests that careful use of early stopping and hyperparameter tuning might be required when using Implicit. The other methods are more stable, with the two truncated methods achieving the lowest training loss after Implicit. Fig. 15 (right) shows that Unroll's performance degradation, relative to other methods, with increasing number of inner loop iterations (also see Fig. 5, right) is not just a generalization issue, but also happens during training. This suggest possible numerical issues from unrolling gradients over a high number of optimization steps, as observed in prior work.
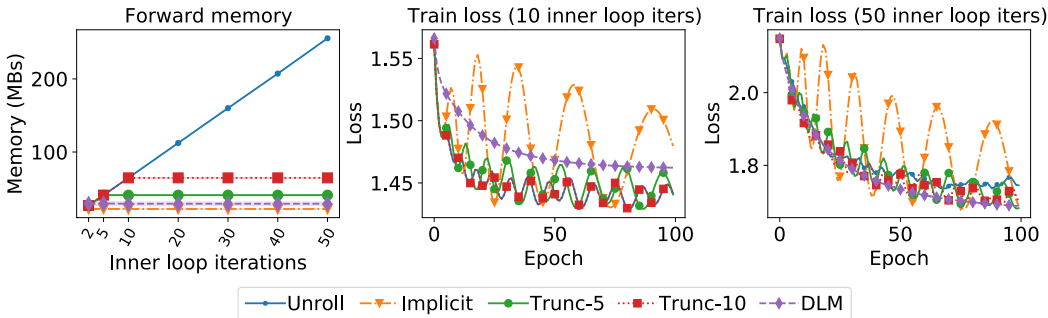

Figure 15: Additional results for backward modes comparison in tactile state estimation problem. **Left:** Memory consumption in forward pass. **Center:** Training loss when using 10 inner loop iterations. **Right:** Training loss when using 50 inner loop iterations.

H.3 Backward modes summary and limitations

Fig. 16 visualizes the backward modes and Table 2 contrasts their limitations. The table shows that all four modes can be used when learning parameters for cost functions or cost weights. However, unlike other approaches, Implicit cannot be used when learning initial values for the optimization variables, θ_{init}. Another limitation of Implicit is that the resulting gradients might be inaccurate in problems where it is not feasible to find the optimal solution to the inner optimization problem; other methods don't experience this limitation, since they compute gradients around the approximate solution found. On the other hand, both Unroll and Trunc could potentially experience vanishing or exploding gradient issues when the number of iterations to backpropagate through is large, a limitation that is not shared by Implicit and DLM. Finally, a limitation of DLM is that ϵ needs to be tuned (see Eq. (16)), which can greatly affect performance. Likewise, the number of backward iterations for Trunc may also require some tuning.

H.4 Derivations for backward modes

H.4.1 Implicit function theorem

For adjoint differentiation, we make use of the implicit function theorem, which is originally from Dini [115], and presented in Dontchev and Rockafellar [39, Theorem 1B.1] as:

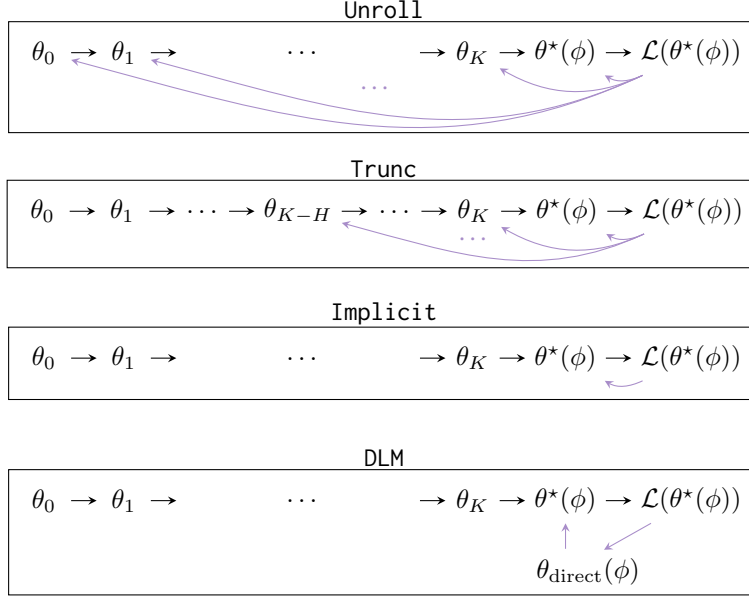|	Cannot be used for learning c_i or w_i	Cannot be used for learning θ_{init}	Requires accurate θ^* solution	Possible vanishing or exploding gradients	Requires tuning	Compute and memory usage
Unroll				✓		high
Trunc				✓	✓	medium
Implicit		✓	✓			low
DLM					✓	low

Table 2: Backward modes summary and limitations.

Figure 16: Illustration of the dependencies of the backward modes for computing $\nabla_\phi \mathcal{L}(\theta^\star)$.

Theorem 1 (Dini's implicit function theorem). *Let the roots of $g(\theta; \phi)$ define an implicit mapping $\Theta^\star(\phi)$ given by $\Theta^\star(\phi) := \{\theta \mid g(\theta; \phi) = 0\}$, where $\theta \in \mathbb{R}^m$, $\phi \in \mathbb{R}^n$, and $g : \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}^m$. Let g be continuously differentiable in a neighborhood of $(\bar\theta, \bar\phi)$ such that $g(\bar\theta; \bar\phi) = 0$, and let the Jacobian of g with respect to θ at $(\bar\theta, \bar\phi)$, i.e. $\mathrm{D}_\theta g(\bar\theta; \bar\phi)$, be non-singular. Then $\Theta^\star$ has a single-valued localization $\theta^\star$ around $\bar\phi$ for $\bar\theta$ which is continuously differentiable in a neighborhood Q of $\bar\phi$ with Jacobian satisfying*

$$\mathrm{D}_\phi \theta^\star(\tilde\phi) = -\mathrm{D}_\theta^{-1} g(\theta^\star(\tilde\phi); \tilde\phi) \mathrm{D}_\phi g(\theta^\star(\tilde\phi); \tilde\phi) \qquad \text{for every } \tilde\phi \in Q. \tag{9}$$

H.4.2 Proof of Prop. 1

Proof. Let $\bar\phi$ be a hyper-parameter resulting in a unique $\theta^\star(\bar\phi)$ and recall that the implicit mapping for Eq. (1) is defined by $g(\theta; \phi) := \nabla_\theta S(\theta, \phi)$ and is zero at the optimal parameters, i.e. $g(\theta^\star(\bar\phi), \bar\phi) = 0$. Let $h(\theta; \phi) := \theta - [\nabla_\theta^2 S(\theta; \phi)]_{\text{stop}}^{-1} \nabla_\theta S(\theta; \phi)$ be the Newton iteration where $[\cdot]_{\text{stop}}$ is a function that zeros the derivative. Differentiating h, which can be done using automatic differentiation on a Newton step, results in the implicit derivative Eq. (9):

$$\begin{aligned}
\mathrm{D}_\phi h(\theta^\star(\bar\phi); \bar\phi) &= \mathrm{D}_\phi \left(\theta - \left[\nabla_\theta^2 S(\theta^\star(\bar\phi); \bar\phi) \right]_{\text{stop}}^{-1} \nabla_\theta S(\theta^\star(\bar\phi); \bar\phi) \right) \\
&= - \left[\nabla_\theta^2 S(\theta^\star(\bar\phi); \bar\phi) \right]_{\text{stop}}^{-1} \mathrm{D}_\phi \nabla_\theta S(\theta^\star(\bar\phi); \bar\phi) \\
&= -\mathrm{D}_\theta^{-1} g(\theta^\star(\bar\phi); \bar\phi) \mathrm{D}_\phi g(\theta^\star(\bar\phi); \bar\phi)
\end{aligned} \tag{10}$$

$\square$

H.4.3 Direct loss minimization for use in `Theseus`

Originally, DLM gradient for non-linear objective functions [41] can be expressed as

$$\nabla_\phi L = \lim_{\varepsilon \to 0} g_{\text{DLM}}^\varepsilon, \quad \text{where} \quad g_{\text{DLM}}^\varepsilon \triangleq \frac{1}{\varepsilon} \left[\frac{\partial}{\partial \phi} S(\theta^*; \phi) - \frac{\partial}{\partial \phi} S(\theta_{\text{direct}}; \phi) \right] \tag{11}$$

where

$$\theta^* = \arg\min_{\hat{\theta}} S(\hat{\theta}; \phi), \qquad \text{and} \qquad \theta_{\text{direct}} = \arg\min_{\hat{\theta}} S(\hat{\theta}; \phi) - \varepsilon L(\hat{\theta}). \qquad (12)$$

However, this dependence on the loss function fits poorly in a reverse-mode automatic differentiation framework like `PyTorch`. Instead, we can construct an equivalent formulation by noting that in continuous space, we can first linearize the loss function around the current solution θ^*,

$$\hat{L}(\theta) = L(\theta^*) + \nabla_\theta L(\theta^*)(\theta - \theta^*) \qquad (13)$$

Let $v = \nabla_\theta L(\theta^*)$, then the perturbed solution becomes

$$\theta_{\text{direct}} = \arg\min_{\hat{\theta}} S(\hat{\theta}; \phi) - \varepsilon \left(L(\theta^*) + v^T(\hat{\theta} - \theta^*) \right) = \arg\min_{\hat{\theta}} S(\hat{\theta}; \phi) - \varepsilon v^T \hat{\theta}. \qquad (14)$$

Plugging this back into Eq. (11), we see that this is an algorithm which takes in a gradient vector v and computes an approximation to the vector-Jacobian product $\nabla_\phi L(\theta^*) = v \frac{\partial \theta^*}{\partial \phi}$.

As `Theseus` is designed to solve optimization problems where S is expressed as sum of squares, it cannot readily handle solving θ_{direct} as this requires adding a linear term to the objective. Instead, let us consider the following "completing the square" approach:

$$\arg\min_{\hat{\theta}} \|\varepsilon\hat{\theta}\|^2 - \varepsilon v^T \theta = \arg\min_{\hat{\theta}} \varepsilon^2 \hat{\theta}^T \hat{\theta} - \varepsilon v^T \hat{\theta} + \left(\tfrac{1}{2}v\right)^T \left(\tfrac{1}{2}v\right) = \arg\min_{\hat{\theta}} \left\|\varepsilon\hat{\theta} - \tfrac{1}{2}v\right\|^2 \qquad (15)$$

We can thus add this extra term and let

$$\theta_{\text{direct}} = \arg\min_{\hat{\theta}} S(\hat{\theta}; \phi) + \left\|\varepsilon\hat{\theta} - \tfrac{1}{2}v\right\|^2 \qquad (16)$$

This adds a small bias to the gradient due to the addition of $\|\varepsilon\hat{\theta}\|^2$ but when ε is small it shouldn't be problematic. In practice, we solve for θ_{direct} by starting from θ^* and using just one iteration of Gauss-Newton.

References

[1] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006. 1, 2, 3

[2] Christian Forster, Luca Carlone, Frank Dellaert, and Davide Scaramuzza. On-manifold preintegration for real-time visual–inertial odometry. *IEEE Transactions on Robotics*, 33(1): 1–21, 2016. 1, 3

[3] Frank Dellaert, Michael Kaess, et al. Factor graphs for robot perception. *Foundations and Trends® in Robotics*, 6(1-2):1–139, 2017. 3

[4] Timothy D. Barfoot. State estimation for robotics. *State Estimation for Robotics*, 2017. doi: 10.1017/9781316671528. 3

[5] Mustafa Mukadam, Jing Dong, Xinyan Yan, Frank Dellaert, and Byron Boots. Continuous-time gaussian process motion planning via probabilistic inference. *The International Journal of Robotics Research*, 37(11):1319–1340, 2018. 3, 22, 23

[6] Mandy Xie and Frank Dellaert. A unified method for solving inverse, forward, and hybrid manipulator dynamics using factor graphs. *arXiv preprint arXiv:1911.10065*, 2019. 3

[7] Antoni Rosinol, Marcus Abate, Yun Chang, and Luca Carlone. Kimera: an open-source library for real-time metric-semantic localization and mapping. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1689–1696. IEEE, 2020. 1, 3

[8] Bill Triggs, Philip F McLauchlan, Richard I Hartley, and Andrew W Fitzgibbon. Bundle adjustment—a modern synthesis. In *International workshop on vision algorithms*, pages 298–372. Springer, 1999. 1, 3, 7, 22

[9] Hanna Pasula, Stuart Russell, Michael Ostland, and Yaacov Ritov. Tracking many objects with many sensors. In *IJCAI*, volume 99, pages 1160–1171. Citeseer, 1999.

[10] Richard Szeliski and Sing Bing Kang. Recovering 3d shape and motion from image streams using nonlinear least squares. *Journal of Visual Communication and Image Representation*, 5 (1):10–28, 1994. 3

[11] Aleksandr Segal, Dirk Haehnel, and Sebastian Thrun. Generalized-icp. In *Robotics: science and systems*, volume 2, page 435. Seattle, WA, 2009.

[12] Tanner Schmidt, Richard A Newcombe, and Dieter Fox. Dart: Dense articulated real-time tracking. In *Robotics: Science and Systems*, volume 2, pages 1–9. Berkeley, CA, 2014. 3

[13] Johannes L Schonberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4104–4113, 2016. 1, 3

[14] Brandon Amos and J Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 136–145. JMLR. org, 2017. 1, 7

[15] Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and J Zico Kolter. Differentiable convex optimization layers. In *Advances in Neural Information Processing Systems*, pages 9558–9570, 2019. 1, 7

[16] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017. 1, 7

[17] Edward Grefenstette, Brandon Amos, Denis Yarats, Phu Mon Htut, Artem Molchanov, Franziska Meier, Douwe Kiela, Kyunghyun Cho, and Soumith Chintala. Generalized inner loop meta-learning. *arXiv preprint arXiv:1910.01727*, 2019. 1

[18] David Duvenaud, Zico Kolter, and Matthew Johnson. Deep Implicit Layers: Neural ODEs, Equilibrium Models, and Differentiable Optimization. http://implicit-layers-tutorial.org/, 2020. 1

[19] Zachary Teed and Jia Deng. Deepv2d: Video to depth with differentiable structure from motion. *arXiv preprint arXiv:1812.04605*, 2018. 2, 3

[20] Mohak Bhardwaj, Byron Boots, and Mustafa Mukadam. Differentiable gaussian process motion planning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10598–10604. IEEE, 2020. 2, 3, 5, 7, 22

[21] Krishna Murthy Jatavallabhula, Ganesh Iyer, and Liam Paull. ∇ slam: Dense slam meets automatic differentiation. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2130–2137. IEEE, 2020. 2, 3

[22] Zachary Teed and Jia Deng. Droid-slam: Deep visual slam for monocular, stereo, and rgb-d cameras. *Advances in Neural Information Processing Systems*, 34, 2021. 2, 3

[23] Chengzhou Tang and Ping Tan. BA-Net: Dense bundle adjustment networks. *7th International Conference on Learning Representations, ICLR 2019*, 2019. 2, 3

[24] Brent Yi, Michelle A Lee, Alina Kloss, Roberto Martín-Martín, and Jeannette Bohg. Differentiable factor graph optimization for learning smoothers. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1339–1345. IEEE, 2021. 2, 3, 5

[25] Hansheng Chen, Pichao Wang, Fan Wang, Wei Tian, Lu Xiong, and Hao Li. Epro-pnp: Generalized end-to-end probabilistic perspective-n-points for monocular object pose estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2781–2790, 2022. 2

[26] Zhaoyang Lv, Frank Dellaert, James M Rehg, and Andreas Geiger. Taking a deeper look at the inverse compositional algorithm. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4581–4590, 2019. 2, 3

[27] Ross Hartley, Maani Ghaffari Jadidi, Lu Gan, Jiunn-Kai Huang, Jessy W Grizzle, and Ryan M Eustice. Hybrid contact preintegration for visual-inertial-contact state estimation using factor graphs. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3783–3790. IEEE, 2018. 2, 3

[28] Paloma Sodhi, Michael Kaess, Mustafa Mukadam, and Stuart Anderson. Learning tactile models for factor graph-based estimation. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 13686–13692. IEEE, 2021. 2, 3, 5, 21, 22

[29] Markus Giftthaler, Michael Neunert, Markus Stäuble, Jonas Buchli, and Moritz Diehl. A family of iterative gauss-newton shooting methods for nonlinear optimal control. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–9. IEEE, 2018. 2, 3

[30] Federica Bogo, Angjoo Kanazawa, Christoph Lassner, Peter Gehler, Javier Romero, and Michael J Black. Keep it smpl: Automatic estimation of 3d human pose and shape from a single image. In *European conference on computer vision*, pages 561–578. Springer, 2016. 2, 3

[31] Taosha Fan, Kalyan Vasudev Alwala, Donglai Xiang, Weipeng Xu, Todd Murphey, and Mustafa Mukadam. Revitalizing optimization for 3d human pose and shape estimation: A sparse constrained formulation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 11457–11466, 2021. 2, 3

[32] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32:8026–8037, 2019. 2

[33] Joan Solà, Jeremie Deray, and Dinesh Atchuthan. A micro Lie theory for state estimation in robotics. *arXiv*, pages 1–17, 2018. URL http://arxiv.org/abs/1812.01537. 2, 5

[34] Franziska Meier, Austin Wang, Giovanni Sutanto, Yixin Lin, and Paarth Shah. Differentiable and learnable robot models. *arXiv preprint arXiv:2202.11217*, 2022. 2, 5

[35] Giorgio Grisetti, Rainer Kümmerle, Hauke Strasdat, and Kurt Konolige. g2o: A general framework for (hyper) graph optimization. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 9–13, 2011. 2, 3, 21

[36] Frank Dellaert. Factor graphs and gtsam: A hands-on introduction. Technical report, Georgia Institute of Technology, 2012.

[37] Sameer Agarwal and Keir Mierle. Ceres solver: Tutorial & reference. *Google Inc*, 2(72):8, 2012. 7, 8, 9, 21

[38] Jing Dong and Zhaoyang Lv. miniSAM: A flexible factor graph non-linear least squares optimization framework. *CoRR*, abs/1909.00903, 2019. URL http://arxiv.org/abs/1909.00903. 2, 3

[39] Asen L Dontchev and R Tyrrell Rockafellar. *Implicit functions and solution mappings*, volume 543. Springer, 2009. 2, 7, 26

[40] Tamir Hazan, Joseph Keshet, and David McAllester. Direct loss minimization for structured prediction. *Advances in neural information processing systems*, 23, 2010. 2, 8

[41] Yang Song, Alexander Schwing, Raquel Urtasun, et al. Training deep neural networks via direct loss minimization. In *International conference on machine learning*, pages 2169–2177. PMLR, 2016. 2, 8, 27

[42] Frank Dellaert and Michael Kaess. Square root sam: Simultaneous localization and mapping via square root information smoothing. *The International Journal of Robotics Research*, 25 (12):1181–1203, 2006. 3

[43] Carl T Kelley. *Iterative methods for optimization*. SIAM, 1999. 3

[44] Edgar Sucar, Kentaro Wada, and Andrew Davison. NodeSLAM: Neural Object Descriptors for Multi-View Shape Reconstruction. *Proceedings - 2020 International Conference on 3D Vision, 3DV 2020*, pages 949–958, 2020. doi: 10.1109/3DV50981.2020.00105. 3

[45] Michael Bloesch, Jan Czarnowski, Ronald Clark, Stefan Leutenegger, and Andrew J Davison. Codeslam—learning a compact, optimisable representation for dense visual slam. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2560–2568, 2018. 3

[46] Jan Czarnowski, Tristan Laidlow, Ronald Clark, and Andrew J. Davison. DeepFactors: Real-Time Probabilistic Dense Monocular SLAM. *IEEE Robotics and Automation Letters*, 5(2): 721–728, 2020. doi: 10.1109/LRA.2020.2965415. 3

[47] Muhammad Asif Rana, Mustafa Mukadam, Seyed Reza Ahmadzadeh, Sonia Chernova, and Byron Boots. Towards robust skill generalization: Unifying learning from demonstration and motion planning. In *Conference on Robot Learning*, pages 109–118. PMLR, 2017. 3

[48] Ronald Clark, Michael Bloesch, Jan Czarnowski, Stefan Leutenegger, and Andrew J Davison. Ls-net: Learning to solve nonlinear least squares for monocular stereo. *arXiv preprint arXiv:1809.02966*, 2018. 3

[49] Barbara Roessle and Matthias Nießner. End2end multi-view feature matching using differentiable pose optimization. *arXiv preprint arXiv:2205.01694*, 2022. 3

[50] Jingwei Huang, Shan Huang, and Mingwei Sun. Deeplm: Large-scale nonlinear least squares on deep learning frameworks using stochastic domain decomposition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10308–10317, 2021. 3, 6

[51] Borglab. Swiftfusion. https://github.com/borglab/SwiftFusion, 2020. 3

[52] Jonathan T Barron. A general and adaptive robust loss function. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4331–4339, 2019. 5

[53] Valentin Peretroukhin, Matthew Giamou, David M Rosen, W Nicholas Greene, Nicholas Roy, and Jonathan Kelly. A smooth representation of belief over so (3) for deep rotation learning with uncertainty. *Robotics: Science and Systems*, 2020. 5

[54] Zachary Teed and Jia Deng. Tangent Space Backpropagation for 3D Transformation Groups. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 10333–10342, 2021. doi: 10.1109/CVPR46437.2021.01020. URL http://arxiv.org/abs/2103.12032. 5

[55] Yi Zhou, Connelly Barnes, Jingwan Lu, Jimei Yang, and Hao Li. On the continuity of rotation representations in neural networks. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019-June:5738–5746, 2019. doi: 10.1109/CVPR.2019.00589.

[56] Congyue Deng, Or Litany, Yueqi Duan, Adrien Poulenard, Andrea Tagliasacchi, and Leonidas J Guibas. Vector neurons: A general framework for so (3)-equivariant networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12200–12209, 2021.

[57] Philippe Hansen-Estruch, Wenling Shang, Lerrel Pinto, Pieter Abbeel, and Stas Tiomkin. Gem: Group enhanced model for learning dynamical control systems. *arXiv preprint arXiv:2104.02844*, 2021. 5

[58] P-A Absil, Robert Mahony, and Rodolphe Sepulchre. *Optimization algorithms on matrix manifolds*. Princeton University Press, 2009. 5, 20, 21

[59] Brent Yi. jaxlie. https://github.com/brentyi/jaxlie, 2021. 5

[60] D.M. Rosen, L. Carlone, A.S. Bandeira, and J.J. Leonard. SE-Sync: A certifiably correct algorithm for synchronization over the special Euclidean group. *Intl. J. of Robotics Research*, 38(2–3):95–125, March 2019. 5, 8, 21

[61] Mike Lambeta, Po-Wei Chou, Stephen Tian, Brian Yang, Benjamin Maloon, Victoria Rose Most, Dave Stroud, Raymond Santos, Ahmad Byagowi, Gregg 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. 5, 21

[62] Sameer Agarwal, Noah Snavely, Steven M Seitz, and Richard Szeliski. Bundle adjustment in the large. In *European conference on computer vision*, pages 29–42. Springer, 2010. 5, 22

[63] Jan Czarnowski, Stefan Leutenegger, and Andrew J. Davison. Semantic texture for robust dense tracking. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV) Workshops*, Oct 2017. 5, 23

[64] Jing Dong, Byron Boots, Frank Dellaert, Ranveer Chandra, and Sudipta Sinha. Learning to align images using weak geometric supervision. In *2018 International Conference on 3D Vision (3DV)*, pages 700–709. IEEE, 2018.

[65] Chengzhou Tang and Ping Tan. BA-Net: Dense Bundle Adjustment Network. In *ICLR*, 2019.

[66] Paul-Edouard Sarlin, Ajaykumar Unagar, Måns Larsson, Hugo Germain, Carl Toft, Viktor Larsson, Marc Pollefeys, Vincent Lepetit, Lars Hammarstrand, Fredrik Kahl, and Torsten Sattler. Back to the Feature: Learning robust camera localization from pixels to pose. In *CVPR*, 2021. URL https://arxiv.org/abs/2103.09213.

[67] Zhaoyang Lv, Frank Dellaert, James Rehg, and Andreas Geiger. Taking a deeper look at the inverse compositional algorithm. In *CVPR*, 2019.

[68] Chaoyang Wang, Hamed Kiani Galoogahi, Chen-Hsuan Lin, and Simon Lucey. Deep-lk for efficient adaptive object tracking. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 627–634, 2018. 5

[69] Yanqing Chen, Timothy A Davis, William W Hager, and Sivasankaran Rajamanickam. Algorithm 887: Cholmod, supernodal sparse cholesky factorization and update/downdate. *ACM Transactions on Mathematical Software (TOMS)*, 35(3):1–14, 2008. 6

[70] Esmond Ng and Barry W Peyton. A supernodal cholesky factorization algorithm for shared-memory multiprocessors. *SIAM Journal on scientific computing*, 14(4):761–769, 1993. 7, 23

[71] Fuzhen Zhang. *The Schur complement and its applications*, volume 4. Springer Science & Business Media, 2006. 7

[72] Jonathan T Barron and Ben Poole. The fast bilateral solver. In *European conference on computer vision*, pages 617–632. Springer, 2016. 7

[73] Barak A Pearlmutter and Jeffrey Mark Siskind. Reverse-mode ad in a functional framework: Lambda the ultimate backpropagator. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 30(2):1–36, 2008. 7

[74] Chongjie Zhang and Victor Lesser. Multi-agent learning with policy prediction. In *Twenty-fourth AAAI conference on artificial intelligence*, 2010.

[75] Dougal Maclaurin, David Duvenaud, and Ryan Adams. Gradient-based hyperparameter optimization through reversible learning. In *International conference on machine learning*, pages 2113–2122. PMLR, 2015.

[76] David Belanger and Andrew McCallum. Structured prediction energy networks. In *International Conference on Machine Learning*, pages 983–992, 2016.

[77] Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. Unrolled generative adversarial networks. *CoRR*, abs/1611.02163, 2016. URL http://arxiv.org/abs/1611.02163.

[78] Tian Han, Yang Lu, Song-Chun Zhu, and Ying Nian Wu. Alternating back-propagation for generator network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[79] David Belanger, Bishan Yang, and Andrew McCallum. End-to-end learning for structured prediction energy networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 429–439. JMLR. org, 2017.

[80] David Belanger. *Deep energy-based models for structured prediction*. PhD thesis, University of Massachusetts Amherst, 2017.

[81] Jakob N Foerster, Richard Y Chen, Maruan Al-Shedivat, Shimon Whiteson, Pieter Abbeel, and Igor Mordatch. Learning with opponent-learning awareness. *arXiv preprint arXiv:1709.04326*, 2017.

[82] Vishal Monga, Yuelong Li, and Yonina C Eldar. Algorithm unrolling: Interpretable, efficient deep learning for signal and image processing. *IEEE Signal Processing Magazine*, 38(2): 18–44, 2021. 7

[83] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990. 7

[84] Herbert Jaeger. *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the" echo state network" approach*, volume 5. GMD-Forschungszentrum Informationstechnik Bonn, 2002. 7

[85] Corentin Tallec and Yann Ollivier. Unbiasing truncated backpropagation through time. *arXiv preprint arXiv:1705.08209*, 2017. 7

[86] Yuhuai Wu, Mengye Ren, Renjie Liao, and Roger Grosse. Understanding short-horizon bias in stochastic meta-optimization. *arXiv preprint arXiv:1803.02021*, 2018.

[87] Renjie Liao, Yuwen Xiong, Ethan Fetaya, Lisa Zhang, KiJung Yoon, Xaq Pitkow, Raquel Urtasun, and Richard Zemel. Reviving and improving recurrent back-propagation. In *International Conference on Machine Learning*, pages 3082–3091. PMLR, 2018.

[88] Amirreza Shaban, Ching-An Cheng, Nathan Hatch, and Byron Boots. Truncated back-propagation for bilevel optimization. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1723–1732. PMLR, 2019.

[89] Paul Vicol, Luke Metz, and Jascha Sohl-Dickstein. Unbiased gradient estimation in unrolled computation graphs with persistent evolution strategies. In *International Conference on Machine Learning*, pages 10553–10563. PMLR, 2021. 7

[90] Brandon Amos. *Differentiable Optimization-Based Modeling for Machine Learning*. PhD thesis, Carnegie Mellon University, May 2019. 7

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

[92] Stephen Gould, Basura Fernando, Anoop Cherian, Peter Anderson, Rodrigo Santa Cruz, and Edison Guo. On differentiating parameterized argmin and argmax problems with application to bi-level optimization. *arXiv preprint arXiv:1607.05447*, 2016.

[93] Jonathan Lorraine, Paul Vicol, and David Duvenaud. Optimizing millions of hyperparameters by implicit differentiation. In *International Conference on Artificial Intelligence and Statistics*, pages 1540–1552. PMLR, 2020.

[94] Mathieu Blondel, Quentin Berthet, Marco Cuturi, Roy Frostig, Stephan Hoyer, Felipe Llinares-López, Fabian Pedregosa, and Jean-Philippe Vert. Efficient and modular implicit differentiation. *arXiv preprint arXiv:2105.15183*, 2021.

[95] Matan Atzmon, Niv Haim, Lior Yariv, Ofer Israelov, Haggai Maron, and Yaron Lipman. Controlling neural level sets. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/b20bb95ab626d93fd976af958fbc61ba-Paper.pdf. 7

[96] Marin Vlastelica, Anselm Paulus, Vít Musil, Georg Martius, and Michal Rolínek. Differentiation of blackbox combinatorial solvers. *arXiv preprint arXiv:1912.02175*, 2019. 8

[97] Mathias Niepert, Pasquale Minervini, and Luca Franceschi. Implicit mle: Backpropagating through discrete exponential family distributions. *Advances in Neural Information Processing Systems*, 34, 2021. 8

[98] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. doi: 10.1038/s41586-020-2649-2. URL https://doi.org/10.1038/s41586-020-2649-2. 10

[99] Paloma Sodhi, Sanjiban Choudhury, Joshua G. Mangelson, and Michael Kaess. ICS: Incremental Constrained Smoothing for State Estimation, 2020. 10

[100] André F.T. Martins, Mário A.T. Figueiredo, Pedro M.Q. Aguiar, Noah A. Smith, and Eric P. Xing. An augmented Lagrangian approach to constrained MAP inference. *Proceedings of the 28th International Conference on Machine Learning, ICML 2011*, pages 169–176, 2011. 10

[101] Richard Zou Horace He. functorch: Jax-like composable function transforms for pytorch. https://github.com/pytorch/functorch, 2021. 10, 19, 20

[102] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 20

[103] Gregory S Chirikjian. *Stochastic models, information theory, and Lie groups, volume 2: Analytic methods and modern applications*, volume 2. Springer Science & Business Media, 2011. 20

[104] Taosha Fan, Hanlin Wang, Michael Rubenstein, and Todd Murphey. Cpl-slam: Efficient and certifiably correct planar graph-based slam using the complex number representation. *IEEE Transactions on Robotics*, 36(6):1719–1737, 2020. 21

[105] Taosha Fan and Todd Murphey. Majorization minimization methods for distributed pose graph optimization. *arXiv preprint arXiv:2108.00083*, 2021. 21

[106] Daniel Martinec and Tomas Pajdla. Robust rotation and translation estimation in multiview reconstruction. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2007. 21

[107] Amit Singer and Yoel Shkolnisky. Three-dimensional structure determination from common lines in cryo-em by eigenvectors and semidefinite programming. *SIAM journal on imaging sciences*, 4(2):543–572, 2011. 21

[108] Mihai Cucuringu, Yaron Lipman, and Amit Singer. Sensor network localization by eigenvector synchronization over the euclidean group. *ACM Transactions on Sensor Networks (TOSN)*, 8 (3):1–42, 2012. 21

[109] Luca Carlone, Roberto Tron, Kostas Daniilidis, and Frank Dellaert. Initialization techniques for 3d slam: A survey on rotation estimation and its use in pose graph optimization. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4597–4604, 2015. 21

[110] Jiaji Zhou, James A. Bagnell, and Matthew T. Mason. A fast stochastic contact model for planar pushing and grasping: Theory and experimental validation. In Nancy M. Amato, Siddhartha S. Srinivasa, Nora Ayanian, and Scott Kuindersma, editors, *Robotics: Science and Systems XIII, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, July 12-16, 2017*, 2017. doi: 10.15607/RSS.2017.XIII.040. URL http://www.roboticsproceedings.org/rss13/p40.html. 22

[111] Paloma Sodhi, Eric Dexheimer, Mustafa Mukadam, Stuart Anderson, and Michael Kaess. Leo: Learning energy-based models in factor graph optimization. In *Conference on Robot Learning*, pages 234–244. PMLR, 2022. 22

[112] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'81, page 674–679, San Francisco, CA, USA, 1981. Morgan Kaufmann Publishers Inc. 23

[113] Simon Baker and Iain A. Matthews. Lucas-kanade 20 years on: A unifying framework. *International Journal of Computer Vision*, 56(3):221–255, 2004. URL https://doi.org/10.1023/B:VISI.0000011205.11775.fd. 23

[114] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Deep image homography estimation. *RSS 2016 Workshop: Limits and Potentials of Deep Learning in Robotics*, 2016. URL http://arxiv.org/abs/1606.03798. 23

[115] Ulisse Dini. *Analisi infinitesimale*. Lithografia Gorani, 1878. 26