Fast(er) Robust Point Cloud Alignment Using Lie Algebra

Jean-Thomas Sexton[1], Michael Morin[2], Philippe Giguère[1], and Jonathan Gaudreault[1]

[1]Department of Computer Science and Software Engineering, Université Laval, Québec, QC, Canada
[2]Department of Operations and Decision Systems, Université Laval, Québec, QC, Canada

Abstract

We present a novel Lie algebra based Iterative Reweighted Least Squares (IRLS) algorithm for robust 3D point cloud alignment. We reformulate the optimal update computation to a compact form which requires only one pass through the data. Although this reformulation does not alter the asymptotic computational complexity, it is well suited for con- temporary hardware architectures, yielding significant practical speedups. In extensive experiments on challenging benchmark datasets with added correspondence corruption, the method is consistently at least four times faster than previous literature whilst being mathematically equivalent, demonstrating it is well suited for time-critical applications.

Fast(er) Robust Point Cloud Alignment Using Lie Algebra

Jean-Thomas Sexton[1], Michael Morin[2], Philippe Giguère[1], Jonathan Gaudreault[1]

***Abstract*— We present a novel Lie algebra based Iterative Reweighted Least Squares (IRLS) algorithm for robust 3D point cloud alignment. We reformulate the optimal update computation to a compact form which requires only one pass through the data. Although this reformulation does not alter the asymptotic computational complexity, it is well suited for contemporary hardware architectures, yielding significant practical speedups. In extensive experiments on challenging benchmark datasets with added correspondence corruption, the method is consistently at least four times faster than previous literature whilst being mathematically equivalent, demonstrating it is well suited for time-critical applications.**

I. Introduction

Point cloud alignment seeks the rigid body transform that best maps one set of 3D points onto another, typically by minimizing an error metric (e.g., sum of squared distances) across matched points. This problem is notably central to the widely used Iterative Closest Points (ICP) algorithm (see [2] for a robotics-oriented review of ICP variants). As the size of real-world datasets and the need for real-time performance escalate (e.g., in autonomous driving, large-scale 3D mapping, and robotic manipulation), it becomes increasingly important to optimize every step in such pipelines, including the point-to-point alignment phase.

Closed-form solutions, e.g., Horn's method or Singular Value Decomposition (SVD), efficiently solve the alignment problem when correspondences are known. Yet, in practice, from feature matching in cluttered environments to LiDAR-based scans with partial overlaps, noise and outliers are often inevitable. Robust M-estimators, implemented via Iterative Reweighted Least Squares (IRLS), help mitigate such outliers by down-weighting them directly within the optimization framework. This makes IRLS-based methods a natural fit for pose refinement in point cloud registration pipelines that initially rely on coarse correspondence estimates (e.g., from RANSAC or an ICP pre-alignment). We refer to [15] for a comprehensive treatment of IRLS and [16] for a thorough comparison of robust M-estimators in registration contexts.

A promising avenue for improving efficiency in robust pose optimization is to adopt a Lie algebra framework, which has gained prominence in robotics over the past two decades [3], [4]. In particular, the Lie group $\mathrm{SE}(3)$ of 3D rigid body motions admits a compact, six-parameter local representation: three for rotation, three for translation. This avoids explicit constraints (e.g., orthonormality of rotation matrices or normalization of quaternions) and circumvents singularities, as found with a Euler angles parametrization. Lie algebra-based approaches have recently demonstrated accuracy and runtime advantages for specialized rigid-motion tasks [6], [14]; broader overviews can be found in [5], [7]. [1]

Despite these theoretical and practical benefits, the important subproblem of efficiently computing the optimal IRLS update in $\mathfrak{se}(3)$, the Lie algebra of the Lie Group of rigid-body motion $\mathrm{SE}(3)$, remains relatively unexplored. One elegant solution appears in [8], but as we discuss in Section II, it requires two passes over the data in an IRLS context. Importantly, this limits its scalability, as well as making it less suitable for modern hardware architectures.

We address this gap by introducing a novel scheme for computing the IRLS update in $\mathfrak{se}(3)$. Specifically we reformulate the optimal update computation to a compact form which requires only one pass through the data. Although this reformulation does not alter the asymptotic computational complexity, it is well suited for contemporary hardware architectures, yielding significant practical speedups. We validate our approach through extensive experiments on the challenging real-world datasets from [18].

The rest of this paper is organized as follows. In Section II, we lay down the mathematical formulation of the problem. Section III presents current approaches. Section IV details our proposed method, while Section V presents empirical results showing that our approach consistently achieves over a *speedup factor of four* compared to existing methods. We conclude in Section VI with final insights and directions for future work.

II. Problem Setting

Our notation roughly follows the one used in [9] and in the documentation of [10]. The problem can be posed as follows: Let $\mathbb{1}_n$ be an $n\times 1$ vector of ones, and I_3 be the 3×3 identity matrix. Let $\tilde{P}$ be the matrix containing the n 3D points of the source point cloud in homogeneous coordinates, i.e.

$$\tilde{P} = \begin{bmatrix} p_1 & \cdots & p_n \\ 1 & \cdots & 1 \end{bmatrix} = \begin{bmatrix} P \\ \mathbb{1}_n^\top \end{bmatrix} \in \mathbb{R}^{4\times n},$$

with $Q = \begin{bmatrix} q_1 & \cdots & q_n \end{bmatrix} \in \mathbb{R}^{3\times n}$ be the matrix containing the corresponding n 3D points of the target point cloud, and W be an $3n \times 3n$ block diagonal matrix whose diagonal entries are the weights for every point correspondence, we write w_i for the weight associated with the i^{th} point

[1]Department of Computer Science and Software Engineering, Université Laval, Québec, Canada jean-thomas.sexton.1@ulaval.ca, {jonathan.gaudreault,philippe.giguere}@ift.ulaval.ca

[2]Department of Operations and Decision Systems, Université Laval, Québec, Canada michael.morin@osd.ulaval.ca

[1]Other methods like Normal Distribution Transform (NDT) use grid-based probabilistic formulations for dense registration, our work focuses on refining pose from coarse point-to-point correspondence estimates. Accordingly, NDT methods are outside the scope of this paper.

correspondence. We note that typically in an IRLS context, each w_i is computed using a robust lost function at each iteration. Let T be a matrix representing our initial guess for the rigid transformation between P and Q with $T \in \mathrm{SE}(3)$ such that

$$T = \begin{bmatrix} R & t \\ \mathbf{0}^\top & 1 \end{bmatrix} \in \mathbb{R}^{4\times 4}\,,$$

with $R \in \mathrm{SO}(3)$, $t \in \mathbb{R}^3$. Let $\xi = \begin{bmatrix} \omega & v \end{bmatrix}^\top \in \mathbb{R}^6$ be a vector with $\omega \in \mathbb{R}^3$ being the rotational component and $v \in \mathbb{R}^3$ being the translational component. We also define the hat operator $\hat{}: \mathbb{R}^6 \to \mathfrak{se}(3)$ such that

$$\hat{\xi} = \begin{bmatrix} [\omega]_\times & v \\ \mathbf{0}^\top & 0 \end{bmatrix} \in \mathfrak{se}(3)\,,$$

where $[\ \ ]_\times : \mathbb{R}^3 \to \mathfrak{so}(3)$ is the skew-symmetric operator, i.e.

$$[\omega]_\times = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix}.$$

We also define the exp operator with $\exp : \mathfrak{se}(3) \to \mathrm{SE}(3)$ as the matrix exponential from the Lie algebra to the Lie group, and the π operator $\pi : \mathbb{R}^{4\times n} \to \mathbb{R}^{3\times n}$ which projects the homogeneous points of a matrix to their three-dimensional counterparts such that $\pi(\tilde{P}) = P$. In our context, this operator is equivalent to taking the first three rows of the matrix. The residual function is defined as

$$e(\xi) = \mathrm{vec}(Q - \pi(\exp(\hat{\xi})T\tilde{P})) \in \mathbb{R}^{3n}\,, \tag{1}$$

where the vec operator converts an $a \times b$ matrix to an $ab \times 1$ vector by stacking its columns, in this case converting a $3 \times n$ matrix into a $3n \times 1$ vector. The function to be minimized with IRLS is

$$E(\xi) = \left(\frac{1}{2} e(\xi)^\top W e(\xi)\right) \in \mathbb{R}\,. \tag{2}$$

We also define

$$\xi^* = \underset{\xi \in \mathbb{R}^6}{\arg\min}\, E(\xi) \tag{3}$$

as the least squares solution. The update ξ^* is then used to update the current transformation estimate during optimization. Specifically, if T_{curr} denotes the current transformation, the next estimate T_{next} is obtained via the hat operator and exponential map as defined in Section II:

$$T_{\text{next}} = \exp(\hat{\xi}) T_{\text{curr}}\,. \tag{4}$$

This update step is iteratively applied until convergence or for a fixed budget of iterations. We note that a complete treatment of the optimality and convergence of the least squares solution involving Lie algebra can be found in [8]. All solutions presented in this paper (including ours) being mathematically equivalent, they share the same previously studied optimality and convergence properties.

III. Current Approaches

In this section, we review established methods for computing the IRLS update in $\mathfrak{se}(3)$ and discuss their respective merits and limitations. First, the *straightforward* approach (Section III-A) directly solves for the IRLS update without consideration for the structure of the problem; although simple and conceptually clear, it requires forming large intermediate matrices and performing costly matrix multiplications. Next, the *adjoint* approach (Section III-B) improves efficiency by leveraging the structure of the Lie group but necessitates two passes over the data in a IRLS context. These shortcomings motivate our proposed method (presented in Section IV), which is both compact and requires only a single pass through the data.

A. Straightforward Approach

The *straightforward* approach to compute the IRLS update ξ^* is to directly solve for the IRLS update using standard linear algebra numerical routines. This is analogous to how a off-the-shelf solver would compute the update without exploiting any specialized structure. In particular, for $\xi = 0$, we have

$$e(0) = \mathrm{vec}(Q - \pi(\exp(0)TP)) = \mathrm{vec}(Q - P')\,, \tag{5}$$

where $\tilde{P}' \equiv TP$ and $P' \equiv \pi(\tilde{P}')$. One of the basic ideas of IRLS optimization is to linearize the residual function by using the first order Taylor expansion around zero:

$$e(\xi) \approx e(0) + J\xi\,, \tag{6}$$

where J is the $3n \times 6$ Jacobian. From standard IRLS formalism, the ξ minimizing $E(\xi)$ is then given by

$$\xi^* = -(J^\top W J)^{-1} J^\top W e(0)\,. \tag{7}$$

In order to get to the form given by (6), we start with the basic IRLS principle of linearizing the residual function using the first order Taylor expansion around zero:

$$\begin{aligned} e(\xi) &\approx \mathrm{vec}(Q - \pi((I + \hat{\xi})T\tilde{P})) \\ &= \mathrm{vec}(Q - P' - \pi(\hat{\xi}\tilde{P}'))\,. \end{aligned} \tag{8}$$

Using the property of the vec operator that $\mathrm{vec}(a + b) = \mathrm{vec}(a) + \mathrm{vec}(b)$, the term on the right corresponds to

$$e(0) - \mathrm{vec}(\pi(\hat{\xi}\tilde{P}'))\,. \tag{9}$$

We expand the term on the right, given that

$$\begin{aligned} \hat{\xi}\tilde{P}' &= \begin{bmatrix} [\omega]_\times & v \\ \mathbf{0}^\top & 0 \end{bmatrix} \begin{bmatrix} P' \\ \mathbb{1}_n^\top \end{bmatrix} \\ &= \begin{bmatrix} [\omega]_\times P' + v\mathbb{1}_n^\top \\ \mathbb{1}_n^\top \end{bmatrix}, \end{aligned} \tag{10}$$

so that

$$\pi(\hat{\xi}\tilde{P}') = \begin{bmatrix} [\omega]_\times p_1' & \cdots & [\omega]_\times p_n' \end{bmatrix} + v\mathbb{1}^\top\,. \tag{11}$$

Using the property of skew-symmetric matrices $[a]_\times b = a \times b = -b \times a = [b]_\times^\top a = -[b]_\times a$, we obtain

$$\pi(\hat{\xi}\tilde{P}') = -\begin{bmatrix} [p_1']_\times \omega & \cdots & [p_n']_\times \omega \end{bmatrix} + v\mathbb{1}^\top\,. \tag{12}$$

We then apply the vec operator and use the property that $\text{vec}(BCA^\top) = (A \otimes B)\text{vec}(C)$, where $\otimes$ denotes the Kronecker product:

$$\text{vec}(\pi(\hat{\xi}\tilde{P}')) = \begin{bmatrix} -[p'_1]_\times \\ \vdots \\ -[p'_n]_\times \end{bmatrix} \omega + (\mathbb{1}_n \otimes I_3)v\,. \tag{13}$$

Developing further, we have that

$$\text{vec}(\pi(\hat{\xi}\tilde{P}')) = P'^\top_{[\,]_\times}\omega + (\mathbb{1}_n \otimes I_3)v\,, \tag{14}$$

where $P'_{[\,]_\times}$ is the $3 \times 3n$ matrix containing the skew-symmetric matrices of every point in P'. Combining (8), (9), and (14), we obtain

$$e(\xi) \approx e(0) - \begin{bmatrix} P'^\top_{[\,]_\times} & (\mathbb{1}_n \otimes I_3) \end{bmatrix} \xi\,. \tag{15}$$

which matches the form of Eq. (6):

$$e(\xi) \approx e(0) + J\xi\,.$$

This implies that the Jacobian is

$$J = -\begin{bmatrix} P'^\top_{[\,]_\times} & (\mathbf{1}_n \otimes I_3) \end{bmatrix}\,. \tag{16}$$

Substituting this into the IRLS update formula (7)

$$\xi^* = -(J^\top W J)^{-1} J^\top W e(0)\,,$$

we see that $(J^\top W J)^{-1}$ has the following form

$$(J^\top W J)^{-1} = \left(\begin{bmatrix} P'_{[\,]_\times} \\ (\mathbf{1}_n^\top \otimes I_3) \end{bmatrix} W \begin{bmatrix} P'^\top_{[\,]_\times} & (\mathbf{1}_n \otimes I_3) \end{bmatrix} \right)^{-1}. \tag{17}$$

and $-J^\top W e(0)$ can be expressed as

$$-J^\top W e(0) = \begin{bmatrix} P'_{[\,]_\times} \\ (\mathbf{1}_n^\top \otimes I_3) \end{bmatrix} W e(0)\,. \tag{18}$$

Multiplying these two expressions being the *straightforward* way to compute the IRLS update ξ^*

B. Adjoint Approach

The more compact current solution from [8], which will be referred to as the *adjoint* approach has the following form:

$$\xi^* = \mathcal{T}\mathcal{M}^{-1}\mathcal{T}^\top a\,, \tag{19}$$

where

$$\mathcal{T} = \begin{bmatrix} R & 0 \\ [t]_\times R & R \end{bmatrix} \in \mathbb{R}^{6\times 6}$$

is the adjoint representation of T and where

$$\mathcal{M} = \left(\begin{bmatrix} I_3 & [\mathbf{p}]_\times \\ 0 & I_3 \end{bmatrix} \begin{bmatrix} A & 0 \\ 0 & I_3 \end{bmatrix} \begin{bmatrix} I_3 & 0 \\ -[\mathbf{p}]_\times & I_3 \end{bmatrix} \right) \in \mathbb{R}^{6\times 6}\,,$$

$$a = \begin{bmatrix} b - [\mathbf{q}]_\times (R\mathbf{p} + t) \\ \mathbf{q} - (R\mathbf{p} + t) \end{bmatrix} \in \mathbb{R}^6\,.$$

Here, $\mathbf{w} = \sum_i w_i$ is the sum of the weights w_i associated with the i^{th} point correspondence, $\mathbf{p} = \frac{1}{\mathbf{w}}\sum_i w_i p_i$, $\mathbf{q} = \frac{1}{\mathbf{w}}\sum_i w_i q_i$, and

$$b = \begin{bmatrix} \text{tr}([\begin{pmatrix} 1 & 0 & 0 \end{pmatrix}^\top]_\times R B^\top) \\ \text{tr}([\begin{pmatrix} 0 & 1 & 0 \end{pmatrix}^\top]_\times R B^\top) \\ \text{tr}([\begin{pmatrix} 0 & 0 & 1 \end{pmatrix}^\top]_\times R B^\top) \end{bmatrix} \in \mathbb{R}^3\,,$$

with

$$B = \left(\frac{1}{\mathbf{w}} \sum_i w_i (q_i - \mathbf{q})(p_i - \mathbf{p})^\top \right) \in \mathbb{R}^{3\times 3}\,,$$

and

$$A = \left(-\frac{1}{\mathbf{w}} \sum_i w_i [p_i - \mathbf{p}]_\times^2 \right) \in \mathbb{R}^{3\times 3}\,.$$

In order to efficiently implement this approach from [8] and later have a fair comparison with our own approach, one can avoid the unnecessary 6×6 matrix multiplications and the inversion of a 6×6 matrix, by using the following form for $\mathcal{M}^{-1}$:

$$\begin{aligned} \mathcal{M}^{-1} &= \begin{bmatrix} I_3 & 0 \\ -[\mathbf{p}]_\times & I_3 \end{bmatrix}^{-1} \begin{bmatrix} A & 0 \\ 0 & I_3 \end{bmatrix}^{-1} \begin{bmatrix} I_3 & [\mathbf{p}]_\times \\ 0 & I_3 \end{bmatrix}^{-1} \\ &= \begin{bmatrix} A^{-1} & -A^{-1}[\mathbf{p}]_\times \\ [\mathbf{p}]_\times A^{-1} & -[\mathbf{p}]_\times A^{-1}[\mathbf{p}]_\times + I \end{bmatrix}, \end{aligned}$$

where $A^{-1} \in \mathbb{R}^{3\times 3}$ only needs to be computed once. The strong point of this *adjoint* approach for the computation of the IRLS update ξ^* is that, given weights w_i that do not have to be recomputed during the optimization (as is the case in an ordinary weighted least squares context), it is very efficient since $\mathcal{M}$, $\mathbf{q}$, $\mathbf{p}$ and b never have to be recomputed after the first iteration.

C. Limitations

The *straightforward* approach described requires the formation of large intermediate matrices and necessitates performing expensive matrix multiplications and inversions without exploiting the underlying structure of the problem, leading to inefficiencies.

Although more compact and easier to compute than the *straightforward* approach, the *adjoint* approach does have performance issues in an IRLS context, where weights w_i are not constant during optimization since at each iteration. Not only is it necessary to update the points of $\tilde{P}$ in order to obtain new weights w_i (and necessary to recompute $\mathcal{M}$, $\mathbf{q}$, $\mathbf{p}$ and b), but this approach also requires two passes over the data: one to calculate the weighted averages ($\mathbf{p}$ and $\mathbf{q}$) and one to compute the matrices A and B.

On modern CPUs, single-pass algorithms typically outperform two-pass approaches—even if both execute the same total number of operations—because the single-pass strategy better leverages instruction-level parallelism. Superscalar architectures can issue and execute multiple instructions per clock cycle in parallel, while SIMD (Single Instruction, Multiple Data) units apply the same operation to multiple data elements concurrently. These features substantially boost throughput when loop iterations are independent.

By contrast, two-pass algorithms with sequential dependencies limit parallelism, as the second pass cannot begin until the first one concludes. This restriction often impairs full utilization of superscalar execution units and SIMD pipelines. The issue extends to GPUs as well, which rely on massively parallel threads to maximize throughput; forcing

multiple sequential passes can similarly underutilize available compute resources.

Moreover, modern CPUs and GPUs rely on efficient caching and memory access patterns to mitigate latency. A two-pass approach typically re-reads the same data twice, increasing the likelihood of cache misses or non-coalesced GPU memory accesses, both of which degrade performance. While this explanation necessarily simplifies the intricacies of contemporary CPU and GPU architectures (for more detailed discussions, see [1] and [17]), it offers a clear rationale for favoring single-pass designs in performance-critical contexts on limited hardware, such as in robotics.

In the next section, we introduce a novel algorithm, for which the results are mathematically equivalent to the one presented above. Most importantly, *it only requires a single pass through the data*, greatly speeding up its execution on modern compute hardware.

IV. Proposed Single Pass $\mathfrak{se}(3)$ Update

We now refine the *straightforward* approach by incorporating the problem structure to obtain a more efficient computation of ξ^*. First, we simply multiply the terms inside parentheses in (17), and obtain the following 6×6 matrix, which we will express as a 2×2 block matrix of 3×3 matrices such that

$$J^\top W J = \begin{bmatrix} -\sum_i w_i [p'_i]_\times^2 & \sum_i w_i [p'_i]_\times \\ -\sum_i w_i [p'_i]_\times & \mathbf{w} I_3 \end{bmatrix} \tag{20}$$

where w_i denotes the weight associated with the i^{th} point correspondence, and $\mathbf{w} \equiv \sum_i w_i$. Using the property that $\sum_i w_i [p'_i]_\times = [\sum_i w_i p'_i]_\times$, we note that it is more computationally efficient to write this expression as

$$J^\top W J = \begin{bmatrix} -\sum_i w_i [p'_i]_\times^2 & [\sum_i w_i p'_i]_\times \\ [\sum_i w_i p'_i]_\times^\top & w I_3 \end{bmatrix} . \tag{21}$$

In addition, by exploiting the identity

$$[p'_i]_\times^2 = p'_i {p'_i}^T - \|p'_i\|^2 I \,,$$

the term $\sum_i w_i [p'_i]_\times^2$ can be computed without repeatedly performing general 3×3 matrix multiplications. Instead, one may accumulate the following scalar sums in a single pass through the data

$$C_{kl} = \sum_i w_i \, (p'_i)_k (p'_i)_l, \quad k, l \in \{x, y, z\} \,,$$

Then, $\sum_i w_i [p'_i]_\times^2$ can be written as

$$\begin{bmatrix} -(C_{yy} + C_{zz}) & C_{xy} & C_{xz} \\ C_{xy} & -(C_{xx} + C_{zz}) & C_{yz} \\ C_{xz} & C_{yz} & -(C_{xx} + C_{yy}) \end{bmatrix} . \tag{22}$$

Avoiding the overhead of general matrix multiplications by relying solely on scalar accumulations and simple arithmetic operations. Next, to efficiently inverse the $(J^\top W J) \in \mathbb{R}^{6 \times 6}$ matrix, we will employ the Schur complement. The latter is a technique commonly used in numerical linear algebra to efficiently invert 2×2 block matrices [11], [12]. Consider a general 2×2 square block matrix $M = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$, we have two options for inverting this matrix. The first one is the so-called Schur complement of A, which we will denote as $K \equiv (D - CA^{-1}B)$:

$$M^{-1} = \begin{bmatrix} A^{-1} + A^{-1}BK^{-1} & -A^{-1}BK^{-1} \\ -K^{-1}CA^{-1} & K^{-1} \end{bmatrix} . \tag{23}$$

The second option is the so-called Schur complement of D, which we will denote as $S \equiv (A - BD^{-1}C)$:

$$M^{-1} = \begin{bmatrix} S^{-1} & -S^{-1}BD^{-1} \\ -D^{-1}CS^{-1} & D^{-1} + D^{-1}CS^{-1}BD^{-1} \end{bmatrix} . \tag{24}$$

The $J^\top W J$ matrix can be written as

$$J^\top W J = \begin{bmatrix} A & B \\ B^\top & D \end{bmatrix} , \tag{25}$$

where $A = -\sum_i w_i [p'_i]_\times^2$, $B = [\sum_i w_i p'_i]_\times$ and $D = \mathbf{w} I_3$, with $B^\top = -B$. By using the Schur complement in A, we would have to invert two 3×3 matrices: A and K. Conversely, by using the Schur complement in D, we have to inverse D and S. Fortunately, the inverse of D is readily obtained as $D^{-1} = \frac{1}{\mathbf{w}} I_3$, leaving us only $S \in \mathbb{R}^{3 \times 3}$ to invert. We therefore have that

$$(J^\top W J)^{-1} = \begin{bmatrix} S^{-1} & -\frac{1}{\mathbf{w}} S^{-1} B \\ \frac{1}{\mathbf{w}} B S^{-1} & \frac{1}{\mathbf{w}} (I_3 - \frac{1}{\mathbf{w}} B S^{-1} B) \end{bmatrix} , \tag{26}$$

where

$$S = (A - BD^{-1}C) = A + \frac{1}{\mathbf{w}} B^2$$

so

$$S = -\sum_i w_i [p'_i]_\times^2 + \frac{1}{\mathbf{w}} [\sum_i w_i p'_i]_\times^2 \,.$$

We now look at the $-J^\top W e(0)$ term

$$-J^\top W e(0) = \begin{bmatrix} P'_{[\,]_\times} \\ (\mathbb{1}_n^\top \otimes I_3) \end{bmatrix} W \begin{bmatrix} q_1 - p'_1 \\ \vdots \\ q_n - p\prime_n \end{bmatrix} \tag{27}$$

$$= \begin{bmatrix} \sum_i w_i p'_i \times q_i \\ \sum_i w_i (q_i - p'_i) \end{bmatrix} . \tag{28}$$

The last expression employs the fact that $[p'_i]_\times p'_i = p'_i \times p'_i = 0$. Using (28) and (26), we have that

$$\xi^* = \begin{bmatrix} S^{-1} & -\frac{1}{\mathbf{w}} S^{-1} B \\ \frac{1}{\mathbf{w}} B S^{-1} & \frac{1}{\mathbf{w}} (I_3 - \frac{1}{\mathbf{w}} B S^{-1} B) \end{bmatrix} \begin{bmatrix} \sum_i w_i p'_i \times q_i \\ \sum_i w_i (q_i - p'_i) \end{bmatrix} . \tag{29}$$

We can further simplify this expression to compute ξ^* more efficiently. We first look at the ω^* component of ξ^*, which is computed by multiplying the first row of the matrix on the left with the vector on the right:

$$\begin{aligned} \omega^* &= S^{-1} \sum_i w_i p'_i \times q_i - \frac{1}{\mathbf{w}} S^{-1} B \sum_i w_i (q_i - p'_i) \\ &= S^{-1} \left(\sum_i w_i p'_i \times q_i - \frac{1}{\mathbf{w}} \sum_i w_i p'_i \times \sum_i w_i q_i \right) , \end{aligned} \tag{30}$$

where the factorization of S^{-1} saves a matrix multiplication and where using the cross product instead of the multiplication by the skew-symmetric matrix B is more efficient. Let $\Omega = \omega^*$, we now look at the v^* component of ξ^*, we use a similar factorization procedure to obtain

$$v^* = \frac{1}{\mathbf{w}} \left(\left(\sum_i w_i p'_i \right) \times \Omega + \sum_i w_i (q_i - p'_i) \right) . \quad (31)$$

By combining (30) and (31), we obtain the proposed solution for ξ^*:

$$\xi^* = \begin{bmatrix} \Omega \\ \frac{1}{\mathbf{w}} \left(\left(\sum_i w_i p'_i \right) \times \Omega + \sum_i w_i (q_i - p'_i) \right) \end{bmatrix} . \quad (32)$$

Every quantity required to compute the IRLS update ξ^* in (32) is now obtained in a single pass through the data. This is notably achieved by leveraging the block structures of $J^\top W J$ and $J^\top e(0)$ and exploiting properties of skew-symmetric matrices in order to directly accumulate the scalar and vector sums needed to compute the update. Moreover, by a judicious application the Schur complement, the inversion of the 6×6 matrix $J^\top W J$ is reduced to that of a single 3×3 matrix S, computed directly from these aggregated quantities. Further algebraic simplifications further help reduce computational overhead by carefully avoiding repeated calculations. These modifications ensure that all necessary terms are computed efficiently in one pass through the data. Thereby avoiding the limitations described in Section III-C and taking advantage of modern processor architectures.

V. Experiments

A. Dataset

Our evaluation draws on the entire dataset collection from [18]. This benchmark provides eight real-world point cloud sequences, and are often used in robotics to benchmark point cloud registration algorithms. Table I summarizes these datasets, which encompass both indoor and outdoor locations. They are intentionally challenging, featuring rapid changes in scale, repetitive structures, and dynamic elements such as moving people or vehicles. Because these datasets are constructed from real measurements from a Hokuyo UTM-30LX 3D time-of-flight scanner, they also include typical sensor noise (on the order of 1–3 cm). Figure 1 provides an illustrative example of the measurement setup.

TABLE I

Point cloud characteristics of the 8 datasets in [18].

Dataset Name	Nb. Scans	Nb. Points per Scan
Apartment	45	365 000
Gazebo Summer	32	170 000
Gazebo Winter	32	153 000
Mountain Plain	31	102 000
Stairs	31	191 000
Wood Autumn	32	178 000
Wood Summer	37	182 000
ETH Hauptgebaude	36	191 000

To isolate the impact of the IRLS update approach independently of an upstream point-to-point matching method, we use the following procedure to create point-to-point correspondences between consecutive scans of each dataset. First, we align each source scan to its target using the ground-truth transformation and establish one-to-one matches by pairing each source point with its nearest neighbour in the target. This yields a set of correspondences, subject to sensor noise. We also create 10% global outliers by swapping the target points of random pairs of matches randomly across each individual pair of scans, simulating more severe, scene-wide mismatches. The 10% level is deliberately chosen: it is large enough to pose a genuine robustness challenge, yet still low enough for every to converge in under 100 IRLS iterations without resorting to additional RANSAC-style pruning, thereby keeping the timing comparison fair. While our main objective is to compare the computational speed of the different IRLS update approaches, introducing a small fraction of outliers (in addition to the already present sensor noise) ensures that each method is evaluated under realistic conditions. The outcome is a set of corrupted correspondences for each consecutive scan pair in each dataset, along with the known ground-truth transformations.

Fig. 1. Example image showing measurement setup for dataset ETH Hauptgebaude in [18]

B. Algorithms

We implemented an IRLS optimizer in C++ with the three distinct previously described methods to compute the IRLS update ξ^*. The *straightforward* approach is as described in Section III-A. In our implementation, we multiply the weights w_i row-wise in order to avoid forming a large $3n \times 3n$ weight matrix and relying on Eigen3's [13] LDLT solver to compute the inverse in (17) efficiently. The second, labelled *adjoint*, is the approach from [8] reviewed in Section III-B. Our own method, based on Eq. (32), is referred to simply as *ours* approach. All three share the same Huber weighting function (with parameter k arbitrarily set to 0.001) and differ only in how they compute the IRLS.

Each algorithm is applied to consecutive scan pairs across all datasets (modified as per Section V-A). The same corrupted correspondences are used for all three algorithms and

the initial guess T is always the identity transformation. The three IRLS updates are mathematically equivalent, so any performance differences arise purely from computational and implementation details. We used the well-optimized library Eigen3 [13] to efficiently handle linear algebra for the different approaches, recompiled locally to take advantage of the Intel i7-4820K CPU at 3.7 GHz used on the GNU/Linux machine used to run all experiments. We note that this processor supports SIMD through the AVX instruction set and that Eigen3 was compiled to take advantage of this feature. All experiments were conducted using a single-threaded implementation of each approach.

We compare the algorithms by recording the computation time for a fixed 100 iterations of each method's optimization loop. By standardizing iteration counts, we isolate computational overhead; as mentioned in 2, all three approaches are mathematically equivalent, and convergence properties can be found in [8]. This process is repeated for 100 trials for each algorithm, with only the randomly generated outlier correspondences differing between trials.

C. Results

Our results clearly demonstrate that the proposed single-pass update achieves a significant reduction in computational overhead compared to both the straightforward and adjoint approaches. In every dataset, *our* approach consistently outperforms the *adjoint* approach by a speedup factor of approximately four.

Table II summarizes the average computation times of each approach for consecutive scan pairs per 100 IRLS iterations across the different datasets, for 100 trials with the speedup factor of *our* approach relative to the more challenging *adjoint* approach. Figure 2 shows a histogram of these same computational times.

Although the focus of the study is on computational speed, we presents for illustration/validation purpose a 2D trajectory from the *Apartment* dataset, which was created by composing the poses for each pair of scans computed by each approach (Figure 3). In addition, Figures 4 and 5 respectively present the rotation error (in radians) and the position error (in radians) for all experiments, the whiskers representing a 95% confidence interval. These three figures illustrate that the improved computational efficiency does not compromise the quality of the pose estimates, all three methods showing strong agreement with the ground truth, despite no global optimization being performed. The *straightforward* approach seems to show some signs of slight numerical instability, perhaps because of the higher number of operations performed. As noted earlier, all three methods being mathematically equivalent, similar agreement is expected and serves more as a sanity check.

VI. Conclusion

We introduced a novel approach for the IRLS update computation in Lie algebra, for robust point cloud alignment. The approach reformulates the update computation into a compact, single-pass form. This results in significant

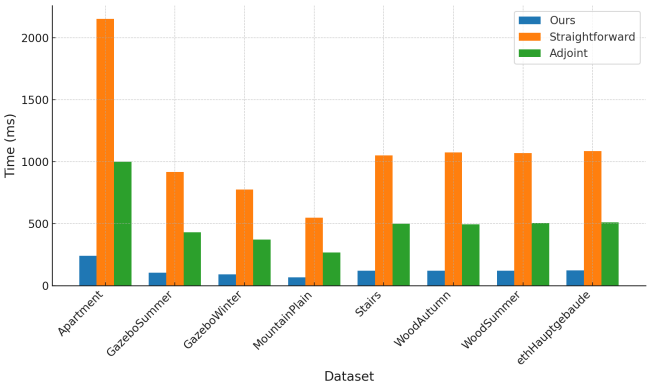

Fig. 2. Average time (in milliseconds) for each algorithm and dataset

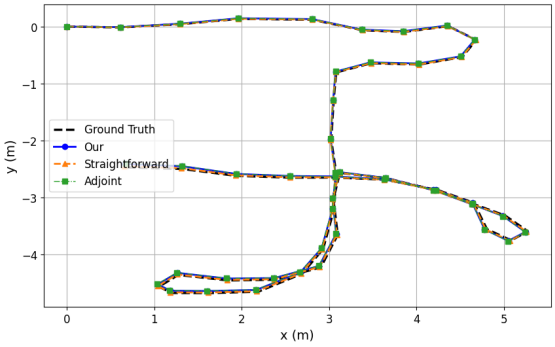

Fig. 3. 2D trajectory of the *Apartment* dataset

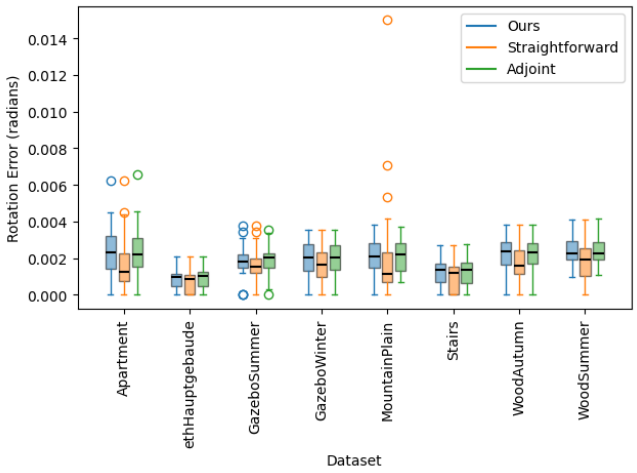

Fig. 4. Average rotation error (in radians) for each algorithm and dataset with 95% confidence interval

TABLE II

AVERAGE TIME IN MILLISECONDS FOR EACH ALGORITHM AND DATASET, WITH SPEEDUP OF OURS OVER ADJOINT

Dataset	Straightforward	Adjoint	Ours	Speedup
Apartment	2151.64	999.91	240.25	4.16
Gazebo Summer	915.58	429.68	103.16	4.17
Gazebo Winter	774.57	370.87	89.23	4.16
Mountain Plain	546.90	266.70	64.43	4.14
Stairs	1049.23	499.63	118.13	4.23
Wood Autumn	1074.71	494.87	119.45	4.14
Wood Summer	1067.72	503.31	119.11	4.23
ETH Hauptgebaude	1084.46	509.63	121.63	4.19

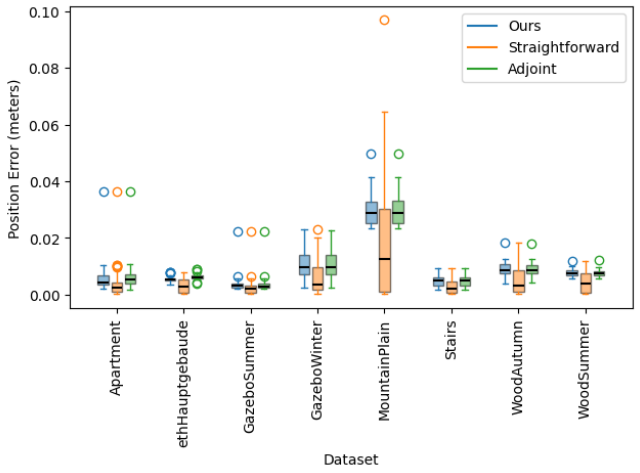

Fig. 5. Average position error (in meters) for each algorithm and dataset with 95% confidence interval

speedups in practice, as confirmed by our experiments on challenging real-world datasets.

Our approach is well suited for time-critical and resource-constrained applications in robotics, where real-time performance is essential. Future work will focus on integrating this update strategy into full SLAM pipelines, and adapting the approach to exploit parallel GPU architectures for further acceleration. In doing so, we anticipate that the benefits of our single-pass formulation will extend beyond isolated point cloud alignment tasks, to broader applications in robotics and computer vision that involve registration operations.

Our contributions advance the efficient and robust computation of pose estimation given coarse point-to-point correspondence estimates, offering a clear pathway for future improvements in both theoretical and practical domains.

REFERENCES

[1] J. L. Hennessy and D. A. Patterson, Computer Architecture: A Quantitative Approach. 6th ed., San Francisco, CA: Morgan Kaufmann, 2017.

[2] F. Pomerleau, F. Colas, R. Siegwart, and others, A review of point cloud registration algorithms for mobile robotics, Foundations and Trends in Robotics, vol. 4, no. 1, pp. 1–104, 2015.

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

[4] J. M. Selig, Lie groups and Lie algebras in robotics, in Computational Noncommutative Algebra and Applications, Springer, 2004, pp. 101–125.

[5] P.-A. Absil, R. Mahony, and R. Sepulchre, Optimization on manifolds: Methods and applications, in Recent Advances in Optimization and its Applications in Engineering: The 14th Belgian-French-German Conference on Optimization, Springer, 2010, pp. 125–144.

[6] M. Vaidis, J. Laconte, V. Kubelka, and F. Pomerleau, Improving the Iterative Closest Point Algorithm using Lie Algebra, arXiv preprint arXiv:2010.11160, 2020.

[7] J. Stillwell, Naive Lie Theory, Springer Science & Business Media, 2008.

[8] T. D. Barfoot, State Estimation for Robotics. Cambridge University Press, 2024.

[9] R. M. Murray, Z. Li, and S. S. Sastry, A Mathematical Introduction to Robotic Manipulation. CRC Press, 2017.

[10] F. Dellaert and GTSAM Contributors, borglab/gtsam, version 4.2a8, Georgia Tech Borg Lab, May 2022. [Online]. Available: https://github.com/borglab/gtsam. [Accessed: Sep. 14, 2024].

[11] L. N. Trefethen and D. Bau, Numerical Linear Algebra. SIAM, 2022.

[12] F. Zhang, The Schur Complement and Its Applications, vol. 4. Springer Science & Business Media, 2006.

[13] G. Guennebaud, B. Jacob, and others, Eigen v3, 2010. [Online]. Available: http://eigen.tuxfamily.org. [Accessed: Sep. 14, 2024].

[14] F. Dellaert, Visual SLAM tutorial: Bundle adjustment, CVPR tutorial, 2014.

[15] R. Wolke and H. Schwetlick, Iteratively reweighted least squares: Algorithms, convergence analysis, and numerical comparisons, SIAM Journal on Scientific and Statistical Computing, vol. 9, no. 5, pp. 907–921, 1988.

[16] P. Babin, P. Giguere, and F. Pomerleau, Analysis of robust functions for registration algorithms, International Conference on Robotics and Automation (ICRA), IEEE, 2019, pp. 1451–1457.

[17] W. H. Wen-Mei, D. B. Kirk, and I. El Hajj, Programming Massively Parallel Processors: A Hands-on Approach. Morgan Kaufmann, 2022.

[18] F. Pomerleau, M. Liu, F. Colas, and R. Siegwart, Challenging data sets for point cloud registration algorithms, The International Journal of Robotics Research, vol. 31, no. 14, pp. 1705–1711, 2012.