ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

V4RL VISION FOR ROBOTICS LAB
Vision for Robotics Lab
Prof. Margarita Chli

Semester Thesis

Gaussian Belief Propagation for Continuous-Time SLAM

Spring Term 2025

Supervised by:
Xinyi Li
William Talbot
Dr. David Hug
Dr. Cornelius von Einem
Prof. Margarita Chli

Author:
Sergi Sánchez Orvay

Contents

Abstract

Continuous-Time SLAM (CTSLAM) has emerged as a compelling alternative to conventional discrete-time approaches, representing the robot trajectory as a smooth function over time rather than a sequence of fixed-interval poses. This continuous formulation naturally fuses asynchronous, multi-rate sensors without explicit interpolation or aggregation, preserves full measurement fidelity, and yields more consistent estimates. However, CTSLAM often incurs higher computational cost.

Gaussian Belief Propagation (GBP) offers a distributed inference framework on factor graphs, in which variables and measurements exchange Gaussian messages to perform local, incremental updates. Unlike centralized Non-Linear Least Squares (NLLS) solvers (e.g. Ceres [1]), GBP can exploit parallelism, detect converged nodes, and avoid repeated global relinearization, making it well suited for real-time, large-scale, or collaborative SLAM applications.

Hyperion [2]—the first open-source continuous-time GBP solver—demonstrated significant speedups over existing spline implementations, yet exhibited numerical instabilities in visual SLAM scenarios. These failures hinder its deployment as a reliable system. In this work, the root causes of divergence—poorly conditioned landmark covariances and underconstrained spline tails—are systematically analysed and introduce different regularization strategies within an online CTSLAM pipeline: diagonal relaxation of precision matrices, Levenberg–Marquardt damping in node updates, message damping, and tail-fixing of spline control points. These techniques are evaluated in both a controlled simple problem and a realistic indoor sequence under varying noise, and outlier conditions. Regularized Hyperion is further benchmarked against a Ceres-based solver in full history and sliding window modes.

Results show that the combined regularization scheme yields stable convergence, high accuracy, and robustness to outliers, while Hyperion consistently outperforms Ceres in runtime—demonstrating its potential as a deployable continuous-time SLAM solution.

Symbols

Symbols

t, t_i	continuous time variable; timestamp of measurement i
$\mathbf{T}_{wb}(t)$	world-to-body pose at time t
$\mathbf{R}_{wb}(\mathbf{q}_{wb}(t))$	rotation matrix from quaternion at time t
$\mathbf{q}_{wb}(t)$	unit-quaternion interpolation at time t
$\mathbf{t}_{wb}(t)$	translation vector at time t
$\mathcal{B}_i$	spline control point (knot) i
k	spline degree
$b_{j,k}(t)$	k-th degree spline basis weight of j-th control point on interval $[t_i, t_{i+k}]$
$\boldsymbol{\Theta}$	set of all trajectory and landmark parameters
$\boldsymbol{\theta}_i \subset \boldsymbol{\Theta}$	subset of parameters involved in factor f_i
$f_i(t_i, \boldsymbol{\theta}_i)$	factor corresponding to measurement i at time t_i
$E_i(t_i, \boldsymbol{\theta}_i)$	energy (negative log-factor) of f_i
$\bar{\mathbf{r}}_i(t_i, \boldsymbol{\theta}_i)$	weighted residual of factor i
$D\bar{\mathbf{r}}_i$	Jacobian of residual $\bar{\mathbf{r}}_i$
$\boldsymbol{\tau}_i$	incremental state update
$\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$	Gaussian distribution with mean $\boldsymbol{\mu}$, and covariance $\boldsymbol{\Sigma}$
$\mathcal{N}^{-1}(\boldsymbol{\eta}, \boldsymbol{\Lambda})$	canonical form of a Gaussian with information $\boldsymbol{\eta}$, and precision $\boldsymbol{\Lambda}$
$\mathbf{m}_{f \to n}$	message from factor to node
$\mathbf{m}_{n \to f}$	message from node to factor
$\kappa(\boldsymbol{\Sigma})$	condition number of matrix $\boldsymbol{\Sigma}$
δ	diagonal relaxation constant
λ	LM damping coefficient
β	message damping blending factor
s	number of spline tail control points fixed
W	sliding window size

Acronyms and Abbreviations

IMU	Inertial Measurement Unit
SLAM	Simultaneous Localization and Mapping
CTSLAM	Continuous-Time Simultaneous Localization and Mapping
GBP	Gaussian Belief Propagation
NLLS	Non-Linear Least Squares

ADMM	Alternating Direction Method of Multipliers
AI	Artificial Intelligence
MGR	Mixed Gaussian Representation
LM	Levenberg-Marquardt
DoF	Degrees of Freedom
RMSE	Root Mean Squared Error
IRE	Initial Reprojection Error
FRE	Final Reprojection Error

Chapter 1

Introduction

Simultaneous Localization and Mapping (SLAM) is a cornerstone of autonomous systems, enabling a mobile robot to concurrently estimate its pose while constructing a map of the environment. The accuracy and robustness of SLAM are paramount for effective navigation, interaction, and task execution in complex real-world scenarios.

Traditionally, SLAM systems have predominantly relied on a discrete-time formulation, where the robot's trajectory is represented as a sequence of states at distinct, fixed time instances. This conventional approach implicitly assumes that all sensor measurements are synchronized to these predefined timestamps. While conceptually straightforward, this paradigm presents several inherent limitations, particularly with the emergence of modern high-rate and asynchronous sensors.

A primary challenge lies in the fixed temporal discretization. Measurements occurring between discrete timesteps are either aggregated or discarded, leading to a potential loss of information and the introduction of unmodeled errors. This issue is exacerbated by sensors like Inertial Measurement Units (IMUs) and rolling-shutter cameras, which provide data at very high frequencies or with internal temporal distortions that are difficult to align perfectly with a fixed global clock. Furthermore, the computational burden of discrete-time methods can scale poorly as the number and frequency of sensors increase, often requiring sophisticated workarounds like measurement aggregation or pre-integration, which can themselves introduce motion distortion and estimation biases.

1.1 Continuous-Time SLAM

To overcome the constraints of discrete-time methods, Continuous-Time SLAM (CTSLAM) has emerged as a powerful alternative. This paradigm models the robot's trajectory as a smooth, continuous function over time—typically parameterized by splines. This continuous representation allows for the evaluation of the robot's pose, velocity, and acceleration at any arbitrary point in time.

The fundamental advantage of CTSLAM is its native ability to fuse asynchronous and multi-modal sensor data. Measurements from various sensors, regardless of their individual sampling rates or internal synchronization, can be directly integrated into the optimization problem at their precise timestamps. This eliminates the need for data synchronization, aggregation, or interpolation, thereby preserving the full fidelity of the sensor information and reducing the introduction of artifacts.

By operating on a continuous trajectory, CTSLAM can also yield more accurate and geometrically consistent estimates, particularly for fast motions or dynamic environments.

Despite these significant advantages, CTSLAM approaches often entail higher computational complexity compared to their discrete-time counterparts. The optimization over a continuous function can be computationally demanding, risking their real-time deployment.

1.2 Gaussian Belief Propagation

Addressing the computational challenges of CTSLAM requires robust and efficient optimization techniques. Gaussian Belief Propagation (GBP) offers a compelling solution within this context. GBP is an iterative, message-passing algorithm that performs probabilistic inference on factor graphs. In a SLAM factor graph, variables (e.g., robot poses, landmark positions) and factors (e.g., sensor measurements, motion models) are represented as nodes, and messages conveying Gaussian distributions are passed between them.

GBP contrasts with traditional centralized Non-Linear Least Squares (NLLS) optimization approaches by its inherent distributed and asynchronous nature. This property makes GBP particularly promising for complex, large-scale SLAM problems, including those arising from continuous-time formulations, as it allows for parallel computation and flexible resource allocation. Furthermore, GBP's distributed nature lends itself well to multi-agent SLAM scenarios, where multiple robots can collaboratively build a map by exchanging information through the factor graph. Moreover, GBP explicitly models the uncertainties of estimated quantities, which can be leveraged to guide the optimization process and selectively focus computational effort on less certain estimates. This combination of explicit uncertainty, distributed inference, and scalability positions GBP as a highly suitable framework for developing efficient and robust continuous-time SLAM systems.

1.3 Project Contributions

Given the advantages of continuous-time SLAM and the promising characteristics of Gaussian Belief Propagation, the Vision for Robotics Lab undertook the development of Hyperion [2]. Hyperion is an open-source continuous-time Gaussian Belief Propagation solver designed to leverage GBP's inherent parallelism and efficiency for complex state estimation problems. While Hyperion successfully demonstrated significant speed improvements over prior spline implementations [2], it encountered numerical instabilities, particularly in visual SLAM scenarios.

This project directly addresses these critical limitations. The work systematically investigates the root causes of these numerical instabilities and proposes regularization techniques to enhance Hyperion's robustness and convergence in challenging visual environments. By rigorously evaluating these improvements and conducting a comprehensive performance comparison against the well-established Ceres [1] solver, this work aims to significantly advance Hyperion's capabilities, bringing it closer to being a truly deployable and reliable continuous-time SLAM system for real-world applications.

The specific goals of this project are:

- Investigate the source of numerical instabilities observed in Hyperion for continuous-time visual SLAM.
- Develop and implement regularization techniques to improve the stability and convergence in visual setups.
- Compare Hyperion's performance against Ceres, focusing on accuracy, runtime, and robustness.

Chapter 2

Related Work

The SLAM problem constitutes the backbone of modern robotic perception, with applications ranging from autonomous navigation and inspection to augmented reality and multi-robot exploration. Historically, SLAM research has focused on discrete-time formulations, where the robot's trajectory is represented as a sequence of poses at fixed timestamps and sensor measurements are carefully synchronized or pre-integrated [3, 4, 5]. Monocular systems such as ORB-SLAM [3] and LSD-SLAM [4] demonstrated robust mapping under constrained motion, while visual-inertial approaches (e.g. VINS-Mono [6], ORB-SLAM3 [5]) fused IMU data via pre-integration. Despite their success, discrete-time SLAM suffers from unmodeled motion during intermediate timestamps, aggregation or dropping of high-rate measurements, and bias from synchronization errors, motivating alternative formulations that avoid rigid timelines.

Continuous-Time SLAM (CTSLAM) models the trajectory as a smooth function over time that can be queried at arbitrary instants, thereby natively fusing multi-rate and asynchronous sensor data without explicit interpolation or time alignment [7, 8, 9]. Early work introduced B-spline and wavelet bases to represent 6-DoF motion [7, 10], while subsequent extensions supported rolling-shutter cameras and event sensors [8, 11]. A recent survey provides a unifying overview of continuous-time state estimation methods, highlighting their flexibility and performance benefits [12]. These methods preserve full sensor fidelity and improve estimation consistency for fast or continuous motions, however, they typically incur higher computational cost.

Gaussian Belief Propagation has recently emerged as a distributed inference alternative to centralized Non-Linear Least Squares (NLLS) solvers in SLAM problems. In GBP, variables and pairwise factors exchange Gaussian messages over a factor graph, enabling local updates and explicit uncertainty quantification [13, 14]. Unlike Alternating Direction Method of Multipliers (ADMM) or other distributed NLLS approaches [15, 16], GBP inherently accommodates the incremental addition of measurements by propagating only local changes through the graph, avoiding repeated batch linearization and global solves. Furthermore, its locality allows the algorithm to identify which nodes have converged and which still require further updates, rather than solving the entire problem at each timestep. This same locality also facilitates multi-agent information sharing without centralized coordination. Prior studies have shown its promise in multi-device localization and spatial AI [17, 18], yet its application to continuous-time trajectories remains underexplored.

Hyperion—developed within the Vision for Robotics Lab—was the first open-source

GBP-based continuous-time SLAM solver [2]. By integrating symbolic, SymForce-accelerated spline representations with Gaussian message-passing inference [19, 2], Hyperion achieved 2–110× speedups over earlier spline implementations [20] while matching the accuracy of Ceres [1] in motion-tracking benchmarks. However, the framework exhibited numerical instabilities in visual SLAM scenarios with loopy factor graphs, presenting considerable challenges to its use as a comprehensive SLAM system. The present work addresses these limitations, introducing regularization techniques that bring Hyperion closer to deployment as a robust continuous-time SLAM solution.

Chapter 3

Preliminaries

This section summarizes the key background on continuous-time trajectory parametrization, the optimization formulation for continuous-time SLAM, the Gaussian Belief Propagation framework, and the robust residual model employed in this work.

3.1 Continuous-Time Parametrization: Z-Splines

In this work, the cubic Z-spline trajectory model of Hyperion [2] is used. The continuous pose $\mathbf{T}_{wb}(t) \in SE(3)$ is written as

$$\mathbf{T}_{wb}(t) = \begin{bmatrix} \mathbf{R}_{wb}\big(\mathbf{q}_{wb}(t)\big) & \mathbf{t}_{wb}(t) \\ \mathbf{0}^\top & 1 \end{bmatrix}, \tag{3.1}$$

where $\mathbf{q}_{wb}(t)$ is the interpolated unit quaternion and $\mathbf{t}_{wb}(t)$ the interpolated translation.

The spline is defined over a sliding window of $k+1$ control knots $\{\mathcal{B}_i, \ldots, \mathcal{B}_{i+k}\}$, with each $\mathcal{B}_i = \big(t_i,\ \mathbf{q}_{wi},\ \mathbf{t}_{wi}\big)$, where t_i is the timestamp, $\mathbf{q}_{wi}$ the quaternion, and $\mathbf{t}_{wi}$ the translation at knot i. For this work, k is set to 3 for cubic Z-splines.

The quaternion interpolation follows the cumulative product formulation:

$$\mathbf{q}_{wb}(t) = \mathbf{q}_{wi} \prod_{j=1}^{3} \Big(\mathbf{q}_{w(i+j-1)}^{-1} * \mathbf{q}_{w(i+j)}\Big)^{b_{j,3}(t)}, \tag{3.2}$$

where $*$ denotes quaternion multiplication and $\{b_{j,3}(t)\}$ are the spline basis of third degree weights on $[t_i, t_{i+3}]$.

Translation is interpolated by weighted control point displacements:

$$\mathbf{t}_{wb}(t) = \mathbf{t}_{wi} + \sum_{j=1}^{3} b_{j,3}(t)\,\big(\mathbf{t}_{w(i+j)} - \mathbf{t}_{w(i+j-1)}\big). \tag{3.3}$$

Equations 3.2 and 3.3 together define $\mathbf{T}_{wb}(t)$ in Eq. 3.1. The cubic Z-splines implementations in SymForce [19] are used, yielding efficient closed-form expressions for pose, velocity, and acceleration.

3.2 Continuous-Time Optimization

Continuous-time SLAM is formulated as probabilistic inference over the trajectory and landmark parameters $\boldsymbol{\Theta} = \{\mathbf{T}_{wb}(t), \mathbf{l}_j\}$. By constructing a factor graph in

which each measurement at time t_i defines a factor f_i acting on the relevant subset $\boldsymbol{\theta}_i \subset \boldsymbol{\Theta}$, the joint posterior becomes

$$p(\boldsymbol{\Theta}) = \prod_i f_i(t_i, \boldsymbol{\theta}_i) \;\propto\; \prod_i \exp\big(-E_i(t_i, \boldsymbol{\theta}_i)\big), \tag{3.4}$$

$$\boldsymbol{\Theta}^* = \arg\max_{\boldsymbol{\Theta}} \log p(\boldsymbol{\Theta}) = \arg\min_{\boldsymbol{\Theta}} \sum_i E_i(t_i, \boldsymbol{\theta}_i) \tag{3.5}$$

Each factor f_i is instantiated as a multivariate Gaussian $\mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$, inducing the energy

$$E_i(t_i, \boldsymbol{\theta}_i) = \|\bar{\boldsymbol{r}}_i(t_i, \boldsymbol{\theta}_i)\|^2_{\boldsymbol{\Sigma}_i^{-1}} = \boldsymbol{r}_i^\top \, \boldsymbol{\Sigma}_i^{-1} \, \boldsymbol{r}_i, \tag{3.6}$$

where $\bar{\boldsymbol{r}}_i$ denotes the weighted residual. This quadratic form links the probabilistic factor to a weighted least-squares cost.

To enable Gaussian Belief Propagation, each residual $\bar{\boldsymbol{r}}_i$ is linearized around the current estimate $\boldsymbol{\theta}_i^0$:

$$\bar{\boldsymbol{r}}_i(\boldsymbol{\theta}_i) \approx \bar{\boldsymbol{r}}_i(\boldsymbol{\theta}_i^0) + D\bar{\boldsymbol{r}}_i(\boldsymbol{\theta}_i^0)\,(\boldsymbol{\theta}_i - \boldsymbol{\theta}_i^0) = \bar{\boldsymbol{r}}_i^0 + \bar{\boldsymbol{J}}_i^0 \, \boldsymbol{\tau}_i, \tag{3.7}$$

with $\bar{\boldsymbol{r}}_i^0 = \bar{\boldsymbol{r}}_i(\boldsymbol{\theta}_i^0)$, Jacobian $\bar{\boldsymbol{J}}_i^0 = D\bar{\boldsymbol{r}}_i(\boldsymbol{\theta}_i^0)$, and increment $\boldsymbol{\tau}_i = \boldsymbol{\theta}_i - \boldsymbol{\theta}_i^0$.

Substituting Eq. 3.7 into Eq. 3.6, and converting to the canonical form $\mathcal{N}^{-1}(\boldsymbol{\eta}_i^0, \boldsymbol{\Lambda}_i^0)$ yields the incremental energy

$$E_i(t_i, \boldsymbol{\theta}_i) \approx \tfrac{1}{2} \, \boldsymbol{\tau}_i^\top \boldsymbol{\Lambda}_i^0 \, \boldsymbol{\tau}_i \;-\; \boldsymbol{\tau}_i^\top \boldsymbol{\eta}_i^0, \tag{3.8}$$

$$\boldsymbol{\eta}_i^0 = -\,\bar{\boldsymbol{J}}_i^{0\top} \, \bar{\boldsymbol{r}}_i^0, \quad \boldsymbol{\Lambda}_i^0 = \bar{\boldsymbol{J}}_i^{0\top} \, \bar{\boldsymbol{J}}_i^0 \tag{3.9}$$

Here, $\boldsymbol{\Lambda}_i^0$ is the precision matrix and $\boldsymbol{\eta}_i^0$ the corresponding information vector. This dual representation—energies E_i and incremental information $(\boldsymbol{\eta}_i^0, \boldsymbol{\Lambda}_i^0)$—facilitates the local conditioning and marginalization steps in Gaussian Belief Propagation.

3.3 Gaussian Belief Propagation

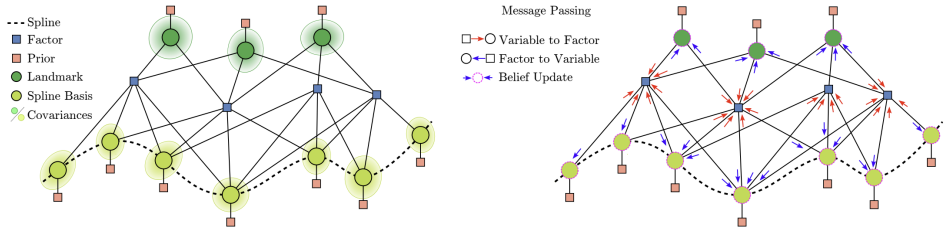
Gaussian Belief Propagation is an iterative, message-passing algorithm for performing inference on factor graphs, where variables and measurement factors exchange Gaussian messages to refine local beliefs. Figure 3.1(a) shows the bipartite structure arising in continuous-time SLAM, with spline control points and landmarks as variable nodes and sensor measurements as factor nodes. During each iteration (Fig. 3.1(b)), variable-to-factor and factor-to-variable messages propagate incrementally, enabling localized updates and explicit uncertainty handling. The following paragraphs detail the computation of node updates, node-to-factor messages, factor updates, and factor-to-node messages.

Node Updates Each variable node n_j maintains a belief $B(n_j) = \mathcal{N}^{-1}(\boldsymbol{\eta}_{n_j}, \boldsymbol{\Lambda}_{n_j})$. During an update, all incoming factor-to-node messages in the neighborhood $N(n_j)$

$$\mathbf{m}_{f_i \to n_j} = \mathcal{N}^{-1}(\boldsymbol{\eta}_{f_i \to n_j}, \boldsymbol{\Lambda}_{f_i \to n_j}), \quad f_i \in N(n_j)$$

are combined by multiplying their distributions. In the canonical form this reduces to summing the information vectors and precision matrices, optionally including a prior $P(n_j) = \mathcal{N}^{-1}(\boldsymbol{\eta}_{n_j}^p, \boldsymbol{\Lambda}_{n_j}^p)$:

$$\boldsymbol{\eta}_{n_j} = \boldsymbol{\eta}_{n_j}^p + \sum_{f_i \in N(n_j)} \boldsymbol{\eta}_{f_i \to n_j}, \quad \boldsymbol{\Lambda}_{n_j} = \boldsymbol{\Lambda}_{n_j}^p + \sum_{f_i \in N(n_j)} \boldsymbol{\Lambda}_{f_i \to n_j} \tag{3.10}$$



(a) Factor Graph **(b)** Message Passing

Figure 3.1: (a) Factor graph for continuous-time SLAM. (b) Message-passing schedule: red arrows are variable-to-factor messages $\mathbf{m}_{n\to f}$, blue arrows are factor-to-variable messages $\mathbf{m}_{f\to n}$, and purple circles indicate belief updates.

The simple summation in Eq. 3.10 holds for Euclidean variables but does not directly apply to states on a Lie group G. To handle non-Euclidean states on $SE(3)$, a Mixed Gaussian Representation (MGR) is employed. Under the MGR, each incoming message $\mathbf{m}_{f_i\to n_j} = \mathcal{N}^{-1}(\boldsymbol{\mu}_{f_i\to n_j}, \boldsymbol{\Lambda}_{f_i\to n_j})$ must be expressed in the tangent space at a common reference $\boldsymbol{\mu}^0_{n_j}$ and $\boldsymbol{\Lambda}^0_{n_j}$. Denoting $\boxminus$ and $\boxplus$ as the group-minus and group-plus operators on $SE(3)$, the transformation follows:

$$\boldsymbol{\tau}^0_{f_i\to n_j} = \boldsymbol{\mu}_{f_i\to n_j} \boxminus \boldsymbol{\mu}^0_{n_j} \in \mathbb{R}^{\dim(\mathfrak{g})}, \tag{3.11}$$

$$\boldsymbol{\Lambda}^0_{f_i\to n_j} = \left[\frac{\partial\, \boldsymbol{\tau}^0_{f_i\to n_j}}{\partial\, \boldsymbol{\mu}_{f_i\to n_j}}\right]^\top \boldsymbol{\Lambda}_{f_i\to n_j} \left[\frac{\partial\, \boldsymbol{\tau}^0_{f_i\to n_j}}{\partial\, \boldsymbol{\mu}_{f_i\to n_j}}\right] \in \mathbb{R}^{\dim(\mathfrak{g})\times\dim(\mathfrak{g})} \tag{3.12}$$

These warped increments are then summed over the factor neighborhood $N(n_j)$ (with optional step-size α_{n_j}) to yield:

$$\boldsymbol{\tau}^+_{n_j} = \alpha_{n_j} \sum_{f_i\in N(n_j)} \boldsymbol{\Lambda}^0_{f_i\to n_j}\, \boldsymbol{\tau}^0_{f_i\to n_j}, \quad \boldsymbol{\Lambda}^+_{n_j} = \sum_{f_i\in N(n_j)} \boldsymbol{\Lambda}^0_{f_i\to n_j} \tag{3.13}$$

Finally, the updated belief is mapped back to $SE(3)$ and its precision re-warped via the Jacobian of $\boxplus$:

$$\boldsymbol{\mu}_{n_j} = \boldsymbol{\mu}^0_{n_j} \boxplus \boldsymbol{\tau}^+_{n_j}, \tag{3.14}$$

$$\boldsymbol{\Lambda}_{n_j} = \left[\frac{\partial\, (\boldsymbol{\mu}^0_{n_j} \boxplus \boldsymbol{\tau})}{\partial\, \boldsymbol{\tau}}\right]^\top_{\boldsymbol{\tau}=\boldsymbol{\tau}^+_{n_j}} \boldsymbol{\Lambda}^+_{n_j} \left[\frac{\partial\, (\boldsymbol{\mu}^0_{n_j} \boxplus \boldsymbol{\tau})}{\partial\, \boldsymbol{\tau}}\right]_{\boldsymbol{\tau}=\boldsymbol{\tau}^+_{n_j}} \tag{3.15}$$

This sequence ensures that all message summation and belief updates respect the non-Euclidean structure of $SE(3)$ while retaining the efficiency of information space operations.

Node-to-Factor Messages After computing the accumulated increments $\boldsymbol{\tau}^+_{n_j}$ and $\boldsymbol{\Lambda}^+_{n_j}$ over all neighboring factors Eq. 3.13, each node must extract the contribution destined for a specific factor f_k. This is done by subtracting f_k from the sum in Eqs. 3.13, yielding

$$\boldsymbol{\tau}^+_{n_j\to f_k} = \sum_{f_i\in N(n_j)\setminus f_k} \boldsymbol{\Lambda}^0_{f_i\to n_j}\, \boldsymbol{\tau}^0_{f_i\to n_j}, \tag{3.16}$$

$$\boldsymbol{\Lambda}^+_{n_j\to f_k} = \sum_{f_i\in N(n_j)\setminus f_k} \boldsymbol{\Lambda}^0_{f_i\to n_j} \tag{3.17}$$

Together with the node's current linearization point $\boldsymbol{\mu}^0_{n_j}$, the triplet $\left(\boldsymbol{\mu}^0_{n_j},\, \boldsymbol{\tau}^+_{n_j\to f_k},\, \boldsymbol{\Lambda}^+_{n_j\to f_k}\right)$ defines the outgoing message $\mathbf{m}_{n_j\to f_k}$.

Factor Updates Each factor f_i connects a set of variable neighbor nodes $N(f_i) = \{n_{j_1}, \dots, n_{j_m}\}$ and is linearized at their current estimates. Its own information form $\mathcal{N}^{-1}(\boldsymbol{\eta}^0_{f_i}, \boldsymbol{\Lambda}^0_{f_i})$ (from Eq. 3.9) is combined with the incoming node-to-factor messages by concatenating their information vectors and forming a block-diagonal precision:

$$\boldsymbol{\eta}^+_{N(f_i)\to f_i} = \begin{bmatrix} \boldsymbol{\eta}^+_{n_{j_1}\to f_i} \\ \vdots \\ \boldsymbol{\eta}^+_{n_{j_m}\to f_i} \end{bmatrix}, \quad \boldsymbol{\Lambda}^+_{N(f_i)\to f_i} = \begin{bmatrix} \boldsymbol{\Lambda}^+_{n_{j_1}\to f_i} & 0 & \cdots & 0 \\ 0 & \boldsymbol{\Lambda}^+_{n_{j_2}\to f_i} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \boldsymbol{\Lambda}^+_{n_{j_m}\to f_i} \end{bmatrix}$$

The factor's intermediate information parameters are then

$$\boldsymbol{\eta}'_{f_i} = \boldsymbol{\eta}^0_{f_i} + \boldsymbol{\eta}^+_{N(f_i)\to f_i}, \quad \boldsymbol{\Lambda}'_{f_i} = \boldsymbol{\Lambda}^0_{f_i} + \boldsymbol{\Lambda}^+_{N(f_i)\to f_i} \tag{3.18}$$

These aggregate all neighboring node contributions before marginalizing out individual nodes in the next step.

Factor-to-Node Messages Given the intermediate factor parameters $\boldsymbol{\eta}'_{f_i}$ and $\boldsymbol{\Lambda}'_{f_i}$ from Eq. 3.18, the message to a specific node $n_a \in N(f_i)$ is obtained by marginalizing out the remaining nodes. For clarity, consider f_i connected to two nodes n_a and n_b.

$$\boldsymbol{\eta}'_{f_i} = \begin{bmatrix} \boldsymbol{\eta}'_a \\ \boldsymbol{\eta}'_b \end{bmatrix}, \quad \boldsymbol{\Lambda}'_{f_i} = \begin{bmatrix} \boldsymbol{\Lambda}'_{aa} & \boldsymbol{\Lambda}'_{ab} \\ \boldsymbol{\Lambda}'_{ba} & \boldsymbol{\Lambda}'_{bb} \end{bmatrix}$$

Marginalizing out n_b via the Schur complement yields the message in information form:

$$\boldsymbol{\eta}_{f_i\to n_a} = \boldsymbol{\eta}'_a \; - \; \boldsymbol{\Lambda}'_{ab} \left(\boldsymbol{\Lambda}'_{bb}\right)^{-1} \boldsymbol{\eta}'_b, \tag{3.19}$$

$$\boldsymbol{\Lambda}_{f_i\to n_a} = \boldsymbol{\Lambda}'_{aa} \; - \; \boldsymbol{\Lambda}'_{ab} \left(\boldsymbol{\Lambda}'_{bb}\right)^{-1} \boldsymbol{\Lambda}'_{ba} \tag{3.20}$$

For factors with more than two nodes, one performs an in-place permutation of $\boldsymbol{\eta}'_{f_i}$ and $\boldsymbol{\Lambda}'_{f_i}$ so that the target node's block appears last, then applies the same Schur complement formula to marginalize out all other nodes.

Finally—as in the node updates—the resulting $(\boldsymbol{\eta}_{f_i\to n_a}, \boldsymbol{\Lambda}_{f_i\to n_a})$ are warped back to the manifold frame at $\boldsymbol{\mu}^0_{n_a}$ using the group-plus operator $\boxplus$ and its Jacobian, analogous to Eqs. 3.14, 3.15.

Robust Residuals and Energies Outliers in sensor measurements can disproportionately skew quadratic costs and hamper convergence. To mitigate this, a robust loss function $\rho(\cdot)$ is applied to each squared residual prior to constructing the Gaussian factor. In this work, the Huber loss is used:

$$\rho(r) = \begin{cases} \frac{1}{2}\, r^2, & |r| \le \gamma, \\ \gamma\,(|r| - \frac{1}{2}\,\gamma), & |r| > \gamma, \end{cases} \tag{3.21}$$

where γ is the threshold separating quadratic and linear regimes. The robust energy for factor i becomes

$$\tilde{E}_i(t_i, \boldsymbol{\theta}_i) = \tfrac{1}{2}\, \rho\big(\bar{\mathbf{r}}_i(t_i, \boldsymbol{\theta}_i)^\top \bar{\mathbf{r}}_i(t_i, \boldsymbol{\theta}_i)\big), \tag{3.22}$$

This strategy preserves GBP's local, parallel updates while reducing sensitivity to spurious measurements.

Chapter 4

Problem Formulation

An online continuous-time SLAM pipeline is evaluated in which a front-end visual SLAM method processes each raw image to yield, at timestamp t_i, an estimated pose $\hat{\mathbf{T}}_{wb}(t_i)$, a set of landmark estimates $L_i = \{\hat{\mathbf{l}}_j\}_{j=1}^{N}$, and their corresponding feature observations $F_i = \{\mathbf{u}_j\}_{j=0}^{N}$. These noisy outputs $(t_i, \hat{\mathbf{T}}_{wb}(t_i), L_i, F_i)$ are streamed into the Hyperion optimizer (Fig. 4.1).

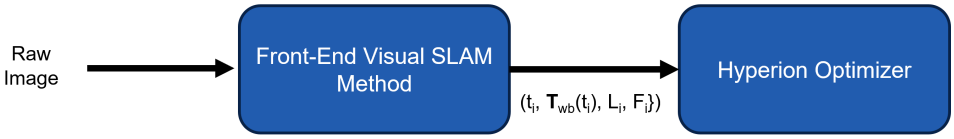


Figure 4.1: Data flow: the front-end visual SLAM method provides $(t_i, \hat{\mathbf{T}}_{wb}(t_i), L_i, F_i)$ to Hyperion for online continuous-time optimization.

Within Hyperion, each $\hat{\mathbf{T}}_{wb}(t_i)$ initializes the $k+1$ control points of the cubic Z-spline covering t_i, while each landmark–feature pair $(\hat{\mathbf{l}}_j, \mathbf{u}_j)$ gives rise to a reprojection factor connecting that landmark node and the corresponding spline control point nodes. An example factor graph is shown in Fig. 4.2.

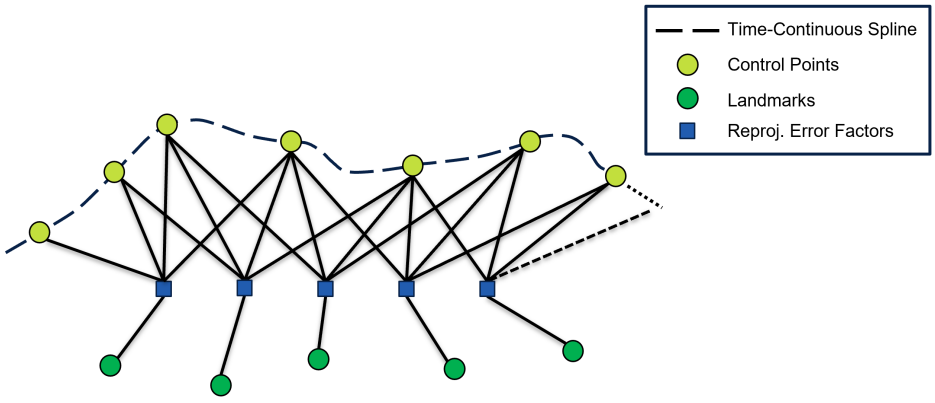


Figure 4.2: Factor graph: yellow circles are spline control points, green circles landmarks, and blue squares reprojection factors.

As each new measurement tuple arrives, its reprojection factors are appended and Gaussian Belief Propagation executes localized message-passing updates on

only the non-converged spline control points and landmark nodes. This online, growing-graph formulation highlights GBP's ability to focus computation where it is needed—detecting and updating unconverged nodes—while avoiding the increasingly large global solves of traditional NLLS.

Chapter 5

Instability Analysis

In prior work with Hyperion, continuous-time GBP exhibited unstable behavior when applied to visual SLAM scenarios. In this chapter, these numerical instabilities are systematically investigated and the resulting convergence failures are analysed. By uncovering the principal causes of divergence, the motivation for the regularization techniques introduced in the next chapter is established.

5.1 Toy SLAM Problem Setup

To diagnose GBP instabilities in a controlled setting, a synthetic simple SLAM scenario with 50 landmarks and a camera trajectory of 200 poses following a winding path is employed. Ground truth poses and landmark positions are generated once, then perturbed to create challenging initial estimates: camera poses (± 0.20 m/rad) and landmark (± 0.20 m/rad), and all feature observations are corrupted with zero-mean Gaussian pixel noise of $\sigma = 1$ px.

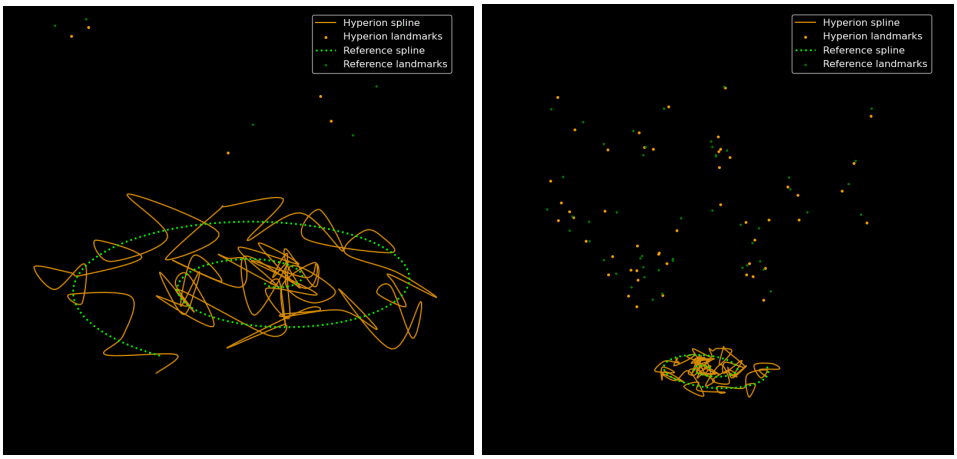


Figure 5.1: Left: ground truth (green) next to perturbed initial (orange) trajectory. Right: ground truth landmarks and trajectory (green) next to initial noisy estimates (orange).

Figure 5.1 (left) shows the reference spline alongside the perturbed initialization, while the right figure shows the reference landmarks and trajectory with perturbed ones. This simple yet illustrative setup facilitates visualization of divergence behaviors.

5.2 Energy Evolution

Hyperion is evaluated in its original, unregularized form on the toy problem. At each new measurement tuple, reprojection factors are appended and run a full GBP solve, recording the normalized energy (energy divided by number of factors) during the inner message-passing iterations.

Figure 5.2 plots these inner-iteration energy curves for the first 50 solves (and every 5 solve times), with the dashed grey line indicating the growing number of factors.

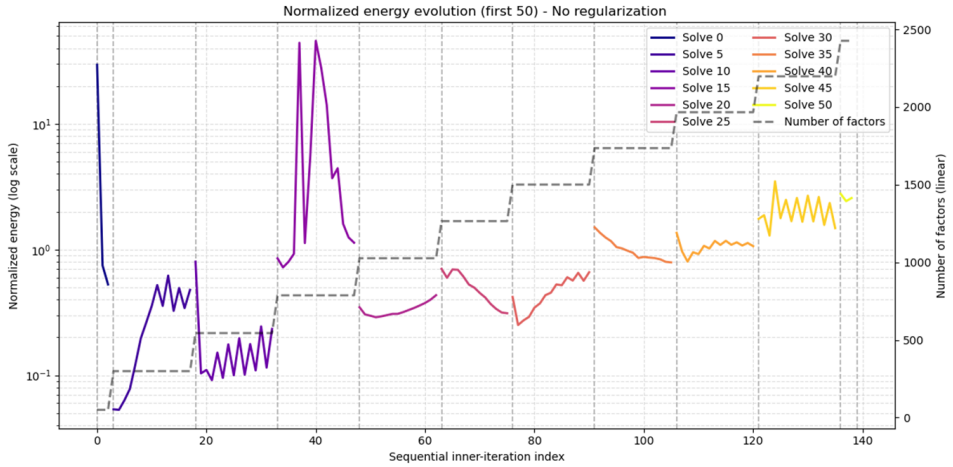


Figure 5.2: Normalized energy evolution (log scale) over inner GBP iterations for unregularized Hyperion. Colors denote successive solves; grey dashed line shows factor count. To ease the visualization, the energy evolution is plotted once every five times new measurements are added to the graph and it is optimized (however, it is actually solved every time a new measurements tuple is received by the optimizer as explained in Chapter 4).

As factors accumulate, the solver's energy repeatedly spikes and fails to decrease, indicating rapid divergence of the message-passing updates. Even early in the sequence (blue and purple curves), the energy oscillates and grows, and by Solve 15 (magenta) the divergence becomes huge. These instabilities demonstrate that, without regularization, Hyperion cannot reliably converge in continuous-time visual SLAM, precluding its deployment in practical settings.

5.3 Covariance and Condition Number Analysis

Gaussian Belief Propagation relies on local information updates that use each node's marginal covariance to weight incoming messages. If a covariance matrix becomes nearly singular, can cause message-passing updates to blow up and the solver to diverge. The covariances of the nodes are monitored to diagnose if the covariance matrices are ill-conditioned.

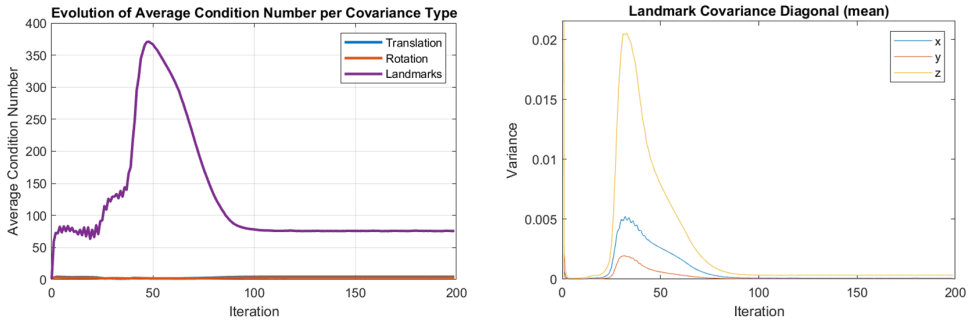
Formally, given a real symmetric positive-definite covariance matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{n \times n}$, its condition number is defined as

$$\kappa(\boldsymbol{\Sigma}) = \frac{\sigma_{\max}(\boldsymbol{\Sigma})}{\sigma_{\min}(\boldsymbol{\Sigma})}, \tag{5.1}$$

where $\sigma_{\max}$ and $\sigma_{\min}$ are the largest and smallest singular values of $\boldsymbol{\Sigma}$. A condition number $\kappa \approx 1$ indicates a well-conditioned matrix, whereas a big κ signals that $\boldsymbol{\Sigma}$ is close to singular along some axis—risking large, unstable updates.

In our toy SLAM experiment, after each graph solve every node's marginal covariances are extracted: the translational and rotational blocks from each spline control point, and each landmark's position covariance. Then, κ is computed for each matrix and report the mean across all nodes of each type. Figure 5.3 (a) shows that, as the graph grows with additional reprojection factors, spline node covariances remain well-conditioned ($\kappa \approx 1$), but landmark covariances rapidly become ill-conditioned ($\kappa > 350$). Such high κ for landmarks implies that this covariance matrices are close to be singular and can lead to instabilities during optimization.

To identify what drives the landmark position matrices to this poor condition, the average diagonal entries (variances) of the landmark covariance matrices are examined along the axes. As seen in Fig. 5.3 (b), the z-axis variance dominates while x and y variances remain low. This means that there is much more uncertainty about the z direction than about the x and y axes. This large disparity stems from monocular scale ambiguity: depth is far less constrained than horizontal directions. The resulting near-singularity of landmark covariances along z explains the high condition numbers in Fig. 5.3 (a) and the corresponding GBP divergence. These findings motivate the need for regularization strategies in continuous-time visual SLAM.



(a) $\delta = 0.001$ (b) $\delta = 1$

Figure 5.3: (a) Mean condition number κ of translational (blue), rotational (orange), and landmark (purple) covariances after each graph solve; (b) Mean diagonal variances of landmark covariances in x (blue), y (orange), and z (yellow) axes after each solve.

Chapter 6

Regularization Techniques

To address the numerical instabilities identified in Chapter 5, this work evaluates different complementary regularization strategies within the Hyperion framework: diagonal relaxation of factor precision matrices, Levenberg–Marquardt damping in the node update step, message damping of factor-to-node updates, and tail-fixing regularization. The following sections introduce each technique.

6.1 Diagonal Relaxation

Diagonal relaxation mitigates ill-conditioning by preventing any single factor from becoming overly confident. Concretely, when constructing each linearized factor f_i, its precision matrix $\boldsymbol{\Lambda}_i$ is modified as

$$\tilde{\boldsymbol{\Lambda}}_i = \boldsymbol{\Lambda}_i + \delta\, \mathbf{I}, \tag{6.1}$$

where $\mathbf{I}$ is the identity matrix and $\delta > 0$ is a relaxation constant. The corresponding information vector is left unchanged, $\tilde{\boldsymbol{\eta}}_i = \boldsymbol{\eta}_i$.

By uniformly increasing all eigenvalues of $\boldsymbol{\Lambda}_i$, this relaxation reduces the condition number of each factor precision and limits the magnitude of downstream message updates. In effect, no measurement can dominate the inference, improving numerical stability at the cost of slightly underweighting all factors.

6.2 Levenberg-Marquardt Damping

Levenberg–Marquardt (LM) damping improves stability of the node update step by tempering the influence of directions with low information. After aggregating incoming messages at node n_j to form $(\boldsymbol{\eta}_{n_j}, \boldsymbol{\Lambda}_{n_j})$ (Eq. 3.10), the precision matrix is modified as

$$\tilde{\boldsymbol{\Lambda}}_{n_j} = \boldsymbol{\Lambda}_{n_j} + \lambda\, \mathrm{diag}(\boldsymbol{\Lambda}_{n_j}), \tag{6.2}$$

where $\lambda > 0$ is a damping coefficient. The information vector remains $\tilde{\boldsymbol{\eta}}_{n_j} = \boldsymbol{\eta}_{n_j}$.

By uniformly increasing the diagonal entries of $\boldsymbol{\Lambda}_{n_j}$, this damping prevents excessively large updates along poorly constrained directions, reducing oscillations and divergence.

6.3 Message Damping

Message damping smooths oscillatory behavior by blending each newly computed factor-to-node message with its previous value. Let

$$\mathbf{m}_{f_i \to n_j}^{\text{new}} = \mathcal{N}^{-1}(\boldsymbol{\eta}_{f_i \to n_j}^{\text{new}}, \boldsymbol{\Lambda}_{f_i \to n_j}^{\text{new}})$$

be the message obtained from Eqs. 3.19, 3.20. The damped message is then

$$\mathbf{m}_{f_i \to n_j} = \mathcal{N}^{-1}\Big((1-\beta)\, \boldsymbol{\eta}_{f_i \to n_j}^{\text{old}} + \beta\, \boldsymbol{\eta}_{f_i \to n_j}^{\text{new}},\ (1-\beta)\, \boldsymbol{\Lambda}_{f_i \to n_j}^{\text{old}} + \beta\, \boldsymbol{\Lambda}_{f_i \to n_j}^{\text{new}} \Big), \quad (6.3)$$

where $0 < \beta \leq 1$ is the damping factor, and old refers to the message from the previous GBP iteration. By tempering abrupt changes in message information, damping mitigates oscillations in loopy graphs and promotes gradual convergence.

6.4 Tail-Fixing Regularization

In CTSLAM problems, it is usual that the most recently added spline control points—the tail of the trajectory—are underconstrained (Fig. 6.1). This lack of measurements on the tail segment leads to ill-conditioning and backward propagation of instability.

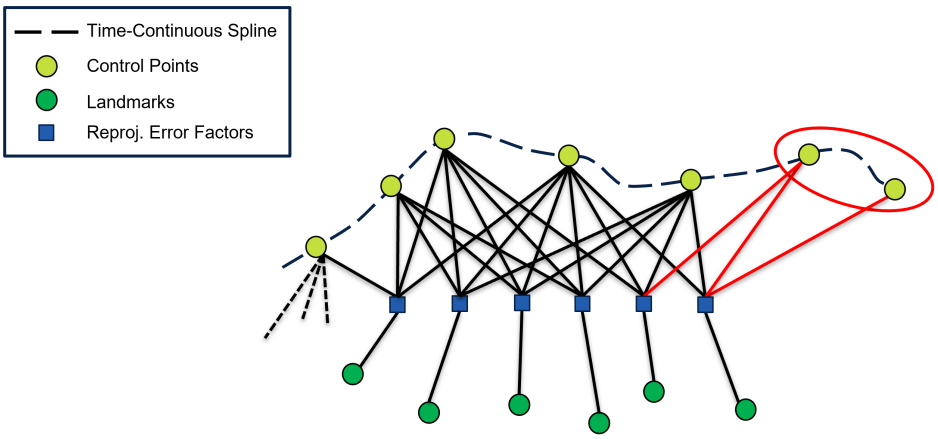


Figure 6.1: Illustration of tail underconstraint: the last spline control points have fewer connected reprojection factors, causing numerical instabilities.

To eliminate this source of divergence, tail-fixing regularization is introduced: at each graph update, the last s control points are fixed, holding their poses constant rather than optimizing them. By removing these degrees of freedom from the optimization, the tail is prevented from destabilizing earlier estimates.

Chapter 7

Experiments

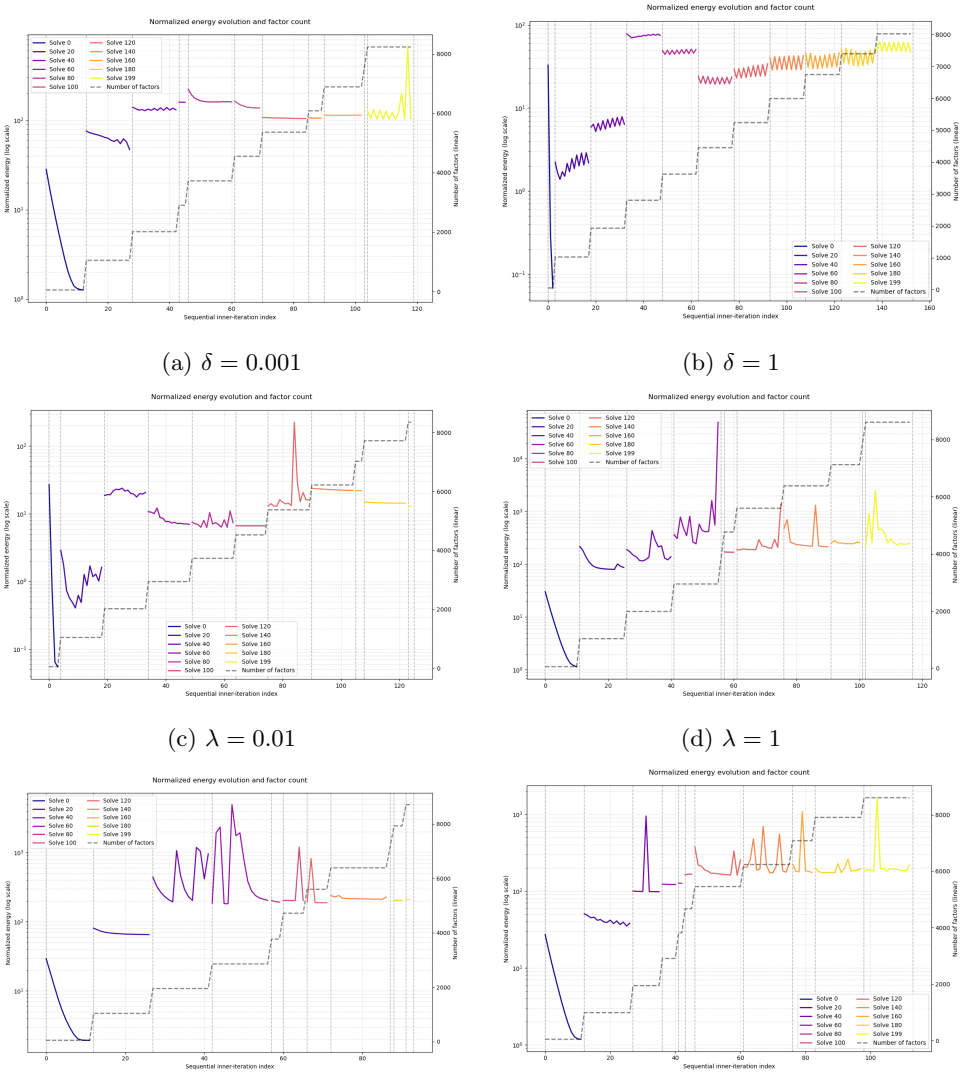
In this chapter, the proposed regularization strategies are evaluated in two stages. Initially, the toy SLAM scenario introduced in Chapter 5—comprising 50 landmarks over 200 poses with noisy initialization—is used to evaluate how diagonal relaxation, Levenberg–Marquardt damping, and message damping enhance stability and convergence in an online continuous-time setting. Subsequently, a more challenging indoor sequence, synthesized from EuRoC [21] trajectory data and a simulated indoor environment, is employed. In the indoor experiments, tail-fixing regularization is added to the three previous techniques, and Hyperion's accuracy, runtime, and outlier robustness are benchmarked against a Ceres-based implementation.

7.1 Toy SLAM Evaluation

The toy SLAM example offers a controlled setting to assess the effect of the regularization strategies on GBP's stability and convergence. Starting from the same perturbed initialization described in Chapter 5, Hyperion was run online with each regularizer applied in isolation. For each run, the normalized energy (total energy divided by factor count) was logged over the inner GBP iterations immediately following each graph update. Additionally, the evolution of factor count is plotted to illustrate how new measurements expand the graph before each solve.

In all experiments for this setup, the threshold for the Huber loss function was set to $\gamma = 1.345$. Additionally, node marginal covariances for both landmarks and spline control points were initialized to the identity matrix. As GBP progresses, covariances can rapidly contract, inflating the corresponding precision matrix entries. For this reason, precision matrix elements can take values in a wide range from units to some thousands. Accordingly, for diagonal relaxation (Sec. 6.1), the relaxation constant δ was swept logarithmically over $[0.001, 100]$ to assess its ability to prevent overconfidence. In Levenberg–Marquardt damping (Sec. 6.2), the damping coefficient λ was varied between 0.01 and 1, spanning from mild to strong diagonal augmentation relative to the original precision magnitudes. Finally, for message damping (Sec. 6.3), the blending factor β was tested across values in $(0, 1)$, where $\beta = 0$ would retain the previous message entirely and $\beta = 1$ would apply the new message without damping.

When each regularization strategy was applied in isolation, no single method was sufficient to restore stable convergence. Figure 7.1 presents representative runs for each individual regularizer at different parameter settings. All curves exhibit energy increases and oscillations similar to the unregularized baseline.



(a) $\delta = 0.001$ (b) $\delta = 1$

(c) $\lambda = 0.01$ (d) $\lambda = 1$

(e) $\beta = 0.25$ (f) $\beta = 0.9$

Figure 7.1: Normalized energy evolution (log scale) and factor count for each regularization strategy: (a),(b) diagonal relaxation with $\delta = 0.001, 1$; (c),(d) LM damping with $\lambda = 0.01, 1$; (e),(f) message damping with $\beta = 0.25, 0.9$.

When all three techniques are combined—applying diagonal relaxation at factor construction, LM damping during node updates, and message damping on factor-to-node exchanges—GBP convergence and stability improve markedly: relaxation prevents overly confident factors, damping smooths large node update steps, and message damping mitigates oscillations, yielding a robust online continuous-time SLAM solution. Figure 7.2 shows the normalized energy evolution for $\delta = 10$, $\lambda = 0.1$, and $\beta = 0.75$. The energy consistently decreases each time new factors are added and the graph is solved. With these settings, the trajectory position RMSE is 2.18e-3 m, the trajectory rotation RMSE is 5.78e-4 rad, and the landmark RMSE is 2.47e-3 m. In Figure 7.3 (a), the unoptimized (noisy) spline and landmarks are plotted against ground truth, while (b) shows the optimized spline and landmarks

closely matching the reference.

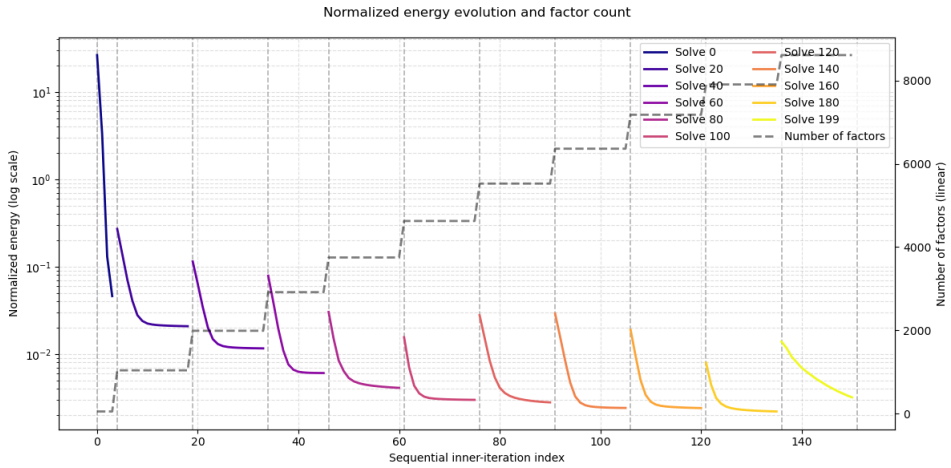
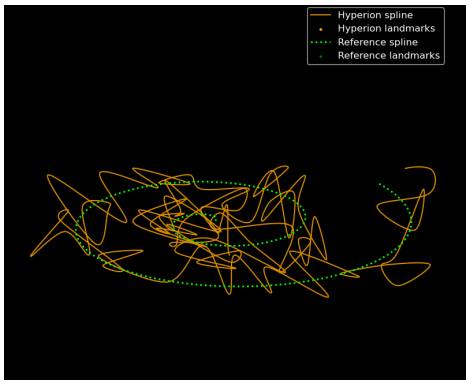
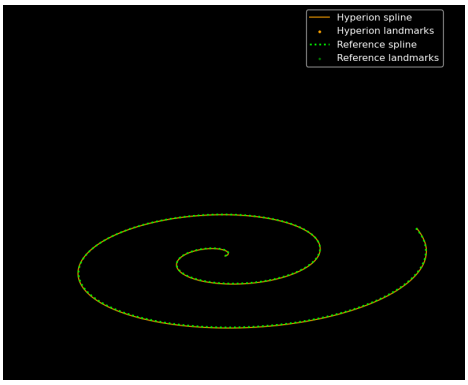


Figure 7.2: Normalized energy evolution (log scale) and factor count with $\delta = 10$, $\lambda = 0.1$, $\beta = 0.75$.



(a) Noisy initialization (orange) vs. ground truth (dashed green).



(b) Optimized result (orange) vs. ground truth (dashed green).

Figure 7.3: Visualization of the results with $\delta = 10$, $\lambda = 0.1$, $\beta = 0.75$: (a) perturbed initialization against reference, (b) optimized trajectory and landmarks.

By combining diagonal relaxation, LM damping, and message damping, the toy SLAM example achieved consistent stability and convergence improvements, with monotonic energy descent and millimeter and sub-degree accuracy. However, this controlled setup remains far simpler than real-world conditions. Accordingly, the following section evaluates the optimizer under more challenging scenarios to assess its robustness and performance in a more complex environment.

7.2 Indoor Environment Sequence

A more realistic evaluation is conducted on a 30-second segment of the V1_01_easy EuRoC [21] trajectory, within which an indoor environment is synthetically generated by surrounding the camera path with outer and inner walls. Each wall is

represented by a uniform grid of point landmarks lying in its plane. At each timestamp, 15% of the total landmarks that project into the camera's field of view (≈ 130 points per frame) are treated as detected features. To emulate feature tracker failures, a 25% dropout probability is applied: each detected feature has a 25% chance of being omitted from the measurement set. The resulting environment and ground truth trajectory are illustrated in Fig. 7.4.

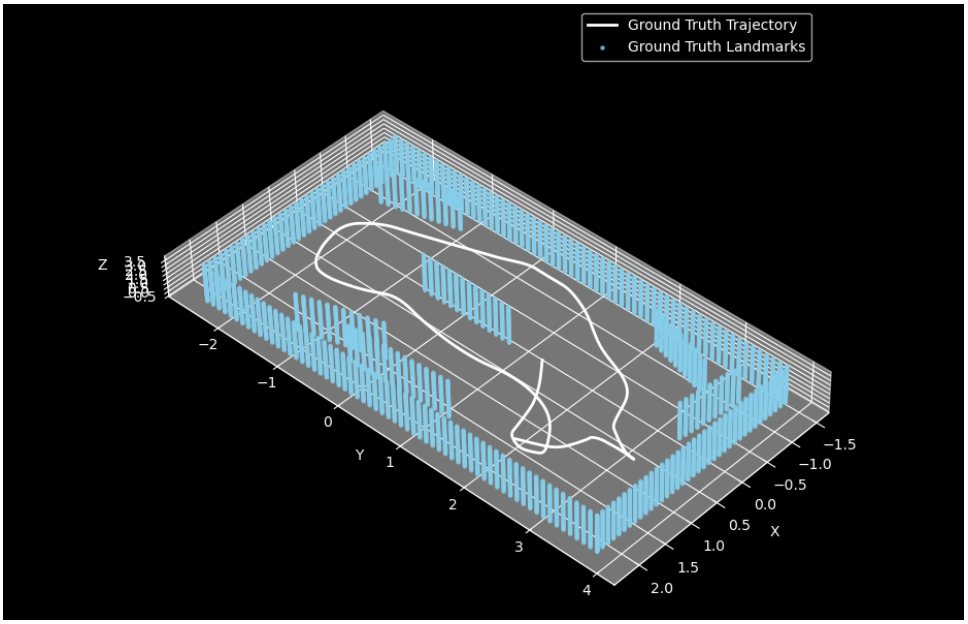


Figure 7.4: Synthetic indoor environment: four outer walls and six interior partitions are modeled as planar landmark grids, overlaid with 30 seconds of ground truth camera trajectory from EuRoC [21] V1_01_easy.

In addition to the three core regularizers, the tail-fixing strategy (Sec. 6.4) was included in this evaluation. To systematically characterize performance, the spline control points and landmarks were initialized with synthetic reprojection noise at 5, 10, and 25 pixel levels. A full sweep was then performed over the four regularization parameters: $\delta \in \{0, 0.1, 0.01, 0.001\}$, $\lambda \in \{0, 0.1, 0.01\}$, $\beta \in \{0.5, 0.75, 1\}$, and tail-fix count $s \in \{0, 1, 2, 3\}$. Parameter ranges were first narrowed by manual pre-selection, ensuring tractable grids including all meaningful values for each parameter. Marginal covariance matrices were initialized to the identity matrix for both landmarks and control points, and the Huber threshold γ was set to the standard deviation of the reprojection errors of the initial measurements of the problem. For each combination and noise level, the first 120 frames of the indoor sequence were optimized online; total solve time, final mean reprojection error, trajectory position RMSE, and rotation RMSE were recorded. Trajectory errors were computed by aligning the optimized and ground truth poses via the Umeyama method using EVO [22].

Table 7.1 summarizes the best configurations found by our parameter sweep across all noise levels and tail-fix settings. In every initialization condition, the best results overall were achieved when fixing the last $s = 2$ spline control points and applying a light diagonal relaxation. These findings suggest that combining tail-fixing with gentle precision relaxation yields the most stable and accurate results.

IRE [px]	λ	β	δ	s	Time [s]	FRE [px]	RMSE Pos [m]	RMSE Rot [rad]
5	0.0	1.0	0.1	0	**3.61**	1.68e-2	2.31e-3	2.78e-3
5	0.01	1.0	0.1	1	4.07	2.10e-2	2.94e-3	5.35e-3
5	0.0	1.0	0.001	2	3.96	**4.21e-3**	**4.21e-4**	**5.12e-4**
5	0.1	0.75	0.001	3	4.49	9.90e-2	8.14e-4	1.52e-2
10	0.0	1.0	0.001	0	40.04	4.26e-1	3.47e-2	1.26e-1
10	0.0	0.5	0.001	1	37.16	3.75e-1	4.46e-2	1.40e-1
10	0.0	1.0	0.001	2	**3.76**	**4.11e-3**	**4.13e-4**	**6.39e-4**
10	0.1	1.0	0.001	3	6.11	2.11e-1	6.38e-3	1.01e-2
25	0.0	1.0	0.1	0	27.84	1.99e-1	3.95e-2	7.29e-2
25	0.0	1.0	0.1	1	30.43	3.66e-1	2.15e-2	1.28e-1
25	0.01	1.0	0.1	2	**4.64**	**9.96e-2**	**2.87e-3**	**3.05e-3**
25	0.1	1.0	0.001	3	29.89	4.65e-1	9.71e-3	1.92e-2

Table 7.1: Results of the parameter sweep experiments on 120 first frames of the indoor sequence. For each initial reprojection error (IRE) level of 5, 10, and 25 pixels and tail-fix count s, the configuration yielding the lowest final reprojection error (FRE) is shown, along with solve time, trajectory position RMSE, and rotation RMSE. Best results per metric and noise level are **bolded**, and second best are underlined.

7.2.1 Comparison Against Ceres

Gaussian Belief Propagation's locality also yields runtime benefits in online CT-SLAM. To quantify this, Hyperion [2] was benchmarked against a Levenberg–Marquardt NLLS implementation in Ceres [1] under identical solver settings (number of threads, maximum number of inner iterations, function and parameter tolerances). Figure 7.5 plots the solve times (log scale) over the sequence as the number of residuals grows. Hyperion completes the entire optimization in under two minutes, whereas Ceres requires over four hours. One of the reasons of this disparity arises because Ceres must perform a full Cholesky factorization on an ever-expanding global system at each inner iteration, incurring superlinear cost as the graph grows. In contrast, Hyperion's GBP updates only the affected local subgraph, incurring linear complexity with the number of added factors. Moreover, GBP can skip converged nodes—once a node's update falls below the parameter tolerance, it is no longer evaluated—whereas Ceres's global termination criterion requires nearly all parameters to settle below the tolerance before stopping.

A sliding window variant was implemented to bound computational cost in both Hyperion and Ceres. At each timestamp, only the most recent W poses and the associated landmarks, and reprojection factors are retained in the factor graph. For Hyperion, the earlier parameter sweep over δ, λ, and β was repeated using window lengths $W \in 20, 50, 100$. For Ceres, to keep run times tractable, the window was fixed at $W = 20$, and only the tail-fix count $s \in 0, 1, 2, 3$ was varied.

Table 7.2 summarizes the sliding-window results over the first 120 frames of the indoor sequence. In every case, a tail-fix count of $s = 2$ yielded the lowest final reprojection error—both for Hyperion and for Ceres—confirming the efficacy of fixing two spline control points at the trajectory's tail. Moreover, Hyperion consistently outperformed Ceres in runtime: even with the largest window ($W = 100$), Hyperion's solve time remained under 10 seconds, whereas Ceres required over 30 seconds for all experiments with $W = 20$. Accuracy was also in Hyperion's favor across most settings; when using the same window size only under the most extreme initialization noise (25 px) did Ceres edge out Hyperion. Finally, the best

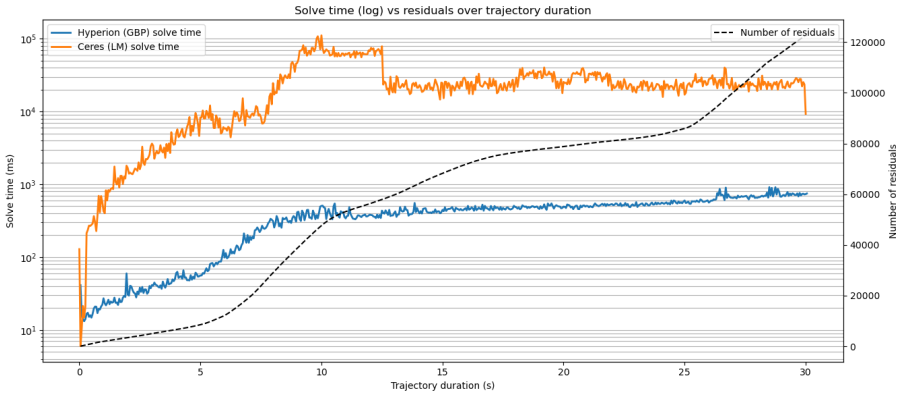


Figure 7.5: Time (log scale) needed to solve the graph in the online CTSLAM optimization problem along with cumulative residuals for Hyperion (GBP) and Ceres (LM) over the 30-second sequence.

Hyperion configurations overall for the 5 px and 10 px noise levels combined $s = 2$ with light diagonal relaxation once again. Interestingly, under 25 px noise a Levenberg–Marquardt damping of $\lambda = 0.1$ together with slight message damping produced the smallest errors, except for the bigger window size where message damping was not needed.

Solver	IRE [px]	λ	β	δ	s	W	Time [s]	FRE [px]	RMSE Pos [m]	RMSE Rot [rad]
Hyperion (GBP)	5	0.0	1.0	0.1	2	20	4.70	3.65e-2	**5.12e-4**	3.16e-4
Hyperion (GBP)	5	0.01	1.0	0.1	2	50	**4.37**	2.38e-2	**5.12e-4**	4.27e-4
Hyperion (GBP)	5	0.0	1.0	0.1	2	100	4.57	**6.41e-3**	7.43e-4	**1.51e-4**
Ceres (LM)	5	-	-	-	2	20	33.22	1.82e-1	1.67e-3	5.92e-3
Hyperion (GBP)	10	0.0	1.0	0.1	2	20	**2.32**	7.05e-2	2.29e-4	**1.19e-4**
Hyperion (GBP)	10	0.0	1.0	0.1	2	50	3.59	2.63e-2	**2.10e-4**	**1.24e-4**
Hyperion (GBP)	10	0.0	1.0	0.1	2	100	6.59	**5.84e-3**	9.71e-4	5.87e-4
Ceres (LM)	10	-	-	-	2	20	37.27	3.67e-1	1.42e-3	5.93e-3
Hyperion (GBP)	25	0.1	0.75	0.1	2	20	**4.14**	1.61	1.01e-2	3.95e-2
Hyperion (GBP)	25	0.1	0.75	0.1	2	50	7.68	7.81e-1	5.52e-3	1.93e-2
Hyperion (GBP)	25	0.1	1.0	0.1	2	100	8.66	**1.70e-1**	**2.19e-3**	**3.36e-3**
Ceres (LM)	25	-	-	-	2	20	44.67	1.26	7.33e-3	9.02e-3

Table 7.2: Comparison of Hyperion (GBP) and Ceres (LM) under varying sliding-window sizes and noise levels for the first 120 frames of the indoor sequence. For each initial reprojection error (IRE) of 5, 10, and 25 pixels and window size W, the configuration $(\lambda, \beta, \delta, s)$ yielding the lowest final reprojection error (FRE) is shown, along with solve time, trajectory position RMSE, and rotation RMSE. Best results per noise level are **bolded**, and second-best results are underlined.

Finally, the robustness of Hyperion and Ceres to measurement outliers was assessed by injecting additional spurious and mismatched feature observations into the indoor sequence. Spurious detections were simulated by generating random pixels on the image plane and assigning them to random landmarks, while mismatches were created by randomly reassociating existing features with incorrect landmarks. Various outlier rates were tested to quantify each solver's resilience under challenging, corrupted inputs.

Table 7.3 reports the outcomes of our outlier robustness experiments, comparing three solver configurations—Hyperion (full history), Hyperion (sliding window), and Ceres (sliding window, $W = 20$)—under initial reprojection noise levels of 5, 10, and 25 pixels and outlier rates of 5 %, 10 %, and 15 %. For each noise–outlier

combination, the same parameter sweep over λ, β, δ, s, and W was applied, and the setting minimizing final reprojection error was selected.

Across all scenarios, the full history Hyperion solver attained the best accuracy, with final reprojection errors up to an order of magnitude lower than the other methods. Even in the worst cases, Hyperion (full history) completed in roughly 15 seconds, whereas Ceres required 60 seconds or more. Ceres' performance degrades markedly once outlier rates are greater or equal to 10 %, although its limited window size ($W = 20$) likely exacerbates this behavior. Hyperion's sliding window variant proved far more robust than Ceres when allowed a larger window, often matching full history accuracy by leveraging $W = 100$. In general, a tail-fix count of $s = 2$ yielded the most consistent results; by contrast, optimal settings for δ, λ, and β varied significantly across conditions, making broad conclusions difficult beyond the universal benefit of fixing the spline tail.

Solver	% Outl.	IRE [px]	λ	β	δ	s	W	Time [s]	FRE [px]	RMSE Pos [m]	RMSE Rot [rad]
Hyperion (GBP)	5	5	0.01	1.0	0.1	2	-	4.75	4.38e-2	1.43e-3	2.92e-3
Hyperion (GBP)	5	5	0.01	0.75	0.1	2	50	**3.73**	**4.01e-2**	**6.11e-4**	**4.31e-4**
Ceres (LM)	5	5	-	-	-	2	20	57.30	1.84	5.23e-3	1.66e-2
Hyperion (GBP)	5	10	0.01	1.0	0.1	2	-	6.94	**4.72e-2**	1.82e-3	2.94e-3
Hyperion (GBP)	5	10	0.1	1.0	0.1	2	100	**5.14**	7.17e-2	**1.65e-3**	**1.62e-3**
Ceres (LM)	5	10	-	-	-	2	20	50.54	1.9159	7.83e-3	2.04e-2
Hyperion (GBP)	5	25	0.1	1.0	0.1	2	-	9.52	**1.25e-1**	**3.71e-3**	**2.79e-3**
Hyperion (GBP)	5	25	0.1	0.75	0.1	2	100	**6.23**	2.61e-1	4.64e-3	6.71e-3
Ceres (LM)	5	25	-	-	-	2	20	65.70	3.86	1.95e-2	5.63e-2
Hyperion (GBP)	10	5	0.0	1.0	0.1	3	-	9.48	1.52e-1	**2.02e-3**	**2.82e-3**
Hyperion (GBP)	10	5	0.0	1.0	0.1	2	100	**7.07**	**1.33e-1**	5.54e-3	1.70e-2
Ceres (LM)	10	5	-	-	-	2	20	82.16	17.57	4.04e-2	4.86e-2
Hyperion (GBP)	10	10	0.0	1.0	0.1	2	-	14.27	**2.98e-1**	1.86e-2	5.30e-2
Hyperion	10	10	0.1	1.0	0.1	3	100	**9.78**	7.31e-1	**8.31e-3**	**1.03e-2**
Ceres (LM)	10	10	-	-	-	2	20	71.45	5.34	2.60e-2	5.03e-2
Hyperion (GBP)	10	25	0.0	1.0	0.1	2	-	17.01	**6.34e-1**	**1.76e-2**	**4.65e-3**
Hyperion (GBP)	10	25	0.1	1.0	0.1	3	100	**15.77**	2.91	2.01e-2	6.06e-2
Ceres (LM)	10	25	-	-	-	2	20	92.96	17.81	1.15e-1	1.92e-1
Hyperion (GBP)	15	5	0.1	0.75	0.001	3	-	**8.13**	**1.89e-1**	**3.83e-3**	**2.81e-3**
Hyperion (GBP)	15	5	0.1	0.75	0.1	3	100	11.91	1.11	4.53e-3	4.33e-3
Ceres (LM)	15	5	-	-	-	2	20	82.75	105.26	1.80e-1	4.97e-1
Hyperion (GBP)	15	10	0.1	1.0	0.1	2	-	10.19	**3.18e-1**	**7.21e-3**	**8.52e-3**
Hyperion (GBP)	15	10	0.1	1.0	0.1	2	100	**9.49**	1.32	8.42e-3	9.87e-3
Ceres (LM)	15	10	-	-	-	2	20	94.57	12.01	3.90e-2	5.69e-2
Hyperion (GBP)	15	25	0.1	1.0	0.1	2	-	15.31	**6.96e-1**	**1.59e-2**	**1.13e-2**
Hyperion (GBP)	15	25	0.1	0.75	0.1	2	100	**11.03**	2.22	2.01e-2	3.83e-2
Ceres (LM)	15	25	-	-	-	2	20	95.08	14.43	4.04e-2	4.33e-2

Table 7.3: Results of the outlier-robustness experiments comparing Hyperion (full history), Hyperion (sliding window $W = 20, 50, 100$), and Ceres (sliding window $W = 20$) for the first 120 frames of the indoor sequence. Initial reprojection error (IRE) levels of 5, 10, and 25 pixels and outlier rates of 5 %, 10 %, 15 % are swept, and for each combination, the configuration with lowest final reprojection error (FRE) is reported alongside solve time, trajectory position RMSE, and rotation RMSE. For each initialization, best metric values are **bolded** and second best are underlined.

Chapter 8

Conclusions

This work aimed to advance continuous-time visual SLAM with the Hyperion [2] solver by addressing numerical limitations identified in prior studies. A systematic analysis revealed that scale ambiguity in monocular setups leads to poorly conditioned landmark covariance matrices, which in turn cause GBP to diverge.

To mitigate this, three regularization techniques—diagonal relaxation of factor precision matrices, Levenberg–Marquardt damping in node updates, and message damping—were integrated into an online CTSLAM pipeline. Although none of these methods alone eliminated instability, their combined application produced reliable convergence and robust stability in a controlled toy problem.

Building on these findings, a fourth strategy—fixing the spline's tail control points—was introduced for a more complex indoor sequence. This approach countered the common underconstraint at the end of the spline and proved essential, delivering the best overall results when the last two control points were held fixed alongside the other regularizers.

Finally, the regularized GBP implementation in Hyperion [2] was benchmarked against a Ceres Levenberg–Marquardt solver [1] under same online and sliding window conditions. Hyperion demonstrated markedly superior efficiency, completing full history optimizations in minutes versus hours for Ceres, while maintaining accuracy and robustness even at high outlier rates. With sliding windows, Hyperion consistently matched or exceeded Ceres in accuracy and outlier-robustness across all tested window sizes, yet required significantly less computation time.

Overall, these contributions bring Hyperion substantially closer to a deployable continuous-time SLAM solution, offering efficient performance, robustness to noise and outliers, and accuracy that overcomes the limitations of previous visual SLAM frameworks.

Chapter 9

Future Work

While the proposed regularization strategies bring Hyperion closer to a practical continuous-time SLAM solution, several limitations remain to be addressed.

Control Point Initialization In our cubic Z-spline implementation, each new measurement influences four control points, three of which lack prior initialization when being first added to the factor graph. This issue was sidestepped by initializing all new knots to the noisy front-end estimates—looking ahead three timesteps. In real deployments, such future measurements are unavailable, and no standard method exists to invert a continuous spline to control point values. Future work should explore principled initialization schemes—such as leveraging previous knot values, IMU dead-reckoning, or learned priors—to improve convergence and optimality without assuming access to future data.

Adaptive Regularization While fixing the last two spline control points consistently stabilized the optimizer, optimal settings for diagonal relaxation, LM damping, and message damping varied with noise and outlier conditions. Static parameter choices may not generalize across environments. Developing adaptive strategies—where regularization parameters are tuned online based on convergence metrics or covariance statistics—could further enhance robustness.

Marginalization. Sliding-window optimization trades accuracy and outlier robustness for runtime efficiency. Marginalization offers a principled way to remove old nodes while retaining their information, thereby bounding problem size without discarding past evidence. Integrating marginalization into Hyperion's GBP framework would enable sustained long-term operation with both high accuracy and bounded computational cost.

Real-World and Multi-Sensor Evaluation. Our experiments used synthetic visual data to precisely characterize numerical behavior. Future validation should employ real front-end SLAM pipelines and real sensor streams to assess performance under real-world conditions. Future work should also potentially include other sensor, or multi-agent inputs where CTSLAM with Gaussian Belief Propagation hold significant potential. Continuous-time GBP's natural support for asynchronous, multi-modal, and distributed inference makes it an attractive candidate for these richer SLAM scenarios.

Addressing these directions will be crucial to fully realize continuous-time GBP's

promise for robust, real-time SLAM in complex, dynamic environments with Hyperion.

Bibliography

[1] S. Agarwal and K. Mierle, "Ceres Solver," https://github.com/ceres-solver/ceres-solver, 2023.

[2] D. Hug, I. Alzugaray, and M. Chli, "Hyperion – A fast, versatile symbolic Gaussian Belief Propagation framework for Continuous-Time SLAM," in *Computer Vision – ECCV 2024*. Cham: Springer Nature Switzerland, 2024, pp. 215–231.

[3] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós, "ORB-SLAM: A Versatile and Accurate Monocular SLAM System," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[4] J. J. Engel, T. Schöps, and D. Cremers, "LSD-SLAM: Large-Scale Direct Monocular SLAM," in *European Conference on Computer Vision*, 2014.

[5] C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. M. Montiel, and J. D. Tardós, "ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual-Inertial and Multi-Map SLAM," *CoRR*, vol. abs/2007.11898, 2020.

[6] T. Qin, P. Li, and S. Shen, "VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator," *CoRR*, vol. abs/1708.03852, 2017.

[7] P. Furgale, T. D. Barfoot, and G. Sibley, "Continuous-time batch estimation using temporal basis functions," in *2012 IEEE International Conference on Robotics and Automation*, 2012, pp. 2088–2095.

[8] S. Lovegrove, A. Patron-Perez, and G. Sibley, "Spline Fusion: A Continuous-Time Representation for Visual-Inertial Fusion with Application to Rolling Shutter Cameras," in *British Machine Vision Conference*, 2013, pp. 2–8.

[9] D. Droeschel and S. Behnke, "Efficient Continuous-Time SLAM for 3D Lidar-Based Online Mapping," *CoRR*, vol. abs/1810.06802, 2018.

[10] S. Anderson, F. Dellaert, and T. D. Barfoot, "A hierarchical wavelet decomposition for continuous-time SLAM," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 373–380.

[11] E. Mueggler, G. Gallego, and D. Scaramuzza, "Continuous-Time Trajectory Estimation for Event-based Vision Sensors," in *Proceedings of Robotics: Science and Systems (RSS)*, Jul. 2015.

[12] W. Talbot, J. Nubert, T. Tuna, C. Cadena, F. Dumbgen, J. Tordesillas, T. D. Barfoot, and M. Hutter, "Continuous-Time State Estimation Methods in Robotics: A Survey," *ArXiv*, vol. abs/2411.03951, 2024.

[13] J. Ortiz, T. Evans, and A. J. Davison, "A visual introduction to Gaussian Belief Propagation," *CoRR*, vol. abs/2107.02308, 2021.

[14] A. J. Davison and J. Ortiz, "FutureMapping 2: Gaussian Belief Propagation for Spatial AI," *CoRR*, vol. abs/1910.14139, 2019.

[15] P. Bänninger, I. Alzugaray, M. Karrer, and M. Chli, "Cross-Agent Relocalization for Decentralized Collaborative SLAM," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023, pp. 5551–5557.

[16] Z. Peng, Y. Xu, M. Yan, and W. Yin, "ARock: an Algorithmic Framework for Asynchronous Parallel Coordinate Updates," *ArXiv*, vol. abs/1506.02396, 2015.

[17] R. Murai, I. Alzugaray, P. H. Kelly, and A. J. Davison, "Distributed Simultaneous Localisation and Auto-Calibration Using Gaussian Belief Propagation," *IEEE Robotics and Automation Letters*, vol. 9, pp. 2136–2143, 2024.

[18] J. Ortiz, T. Evans, E. Sucar, and A. J. Davison, "Incremental Abstraction in Distributed Probabilistic SLAM Graphs," arXiv preprint arXiv:2109.06241, 2021.

[19] H. Martiros, A. Miller, N. Bucki, B. Solliday, R. Kennedy, J. Zhu, T. Dang, D. Pattison, H. Zheng, T. Tomic, P. Henry, G. Cross, J. VanderMey, A. Sun, S. Wang, and K. Holtz, "SymForce: Symbolic Computation and Code Generation for Robotics," in *Proceedings of Robotics: Science and Systems*, 2022.

[20] C. Sommer, V. Usenko, D. Schubert, N. Demmel, and D. Cremers, "Efficient Derivative Computation for Cumulative B-Splines on Lie Groups," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 11 145–11 153.

[21] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart, "The EuRoC micro aerial vehicle datasets," *The International Journal of Robotics Research*, 2016.

[22] M. Grupp, "evo: Python package for the evaluation of odometry and SLAM," https://github.com/MichaelGrupp/evo, 2016.