Imperial College London
Department of Computing

Gaussian Belief Propagation for Real-Time Decentralised Inference

Joseph Ortiz

28th February 2023

Supervised by Professor Andrew Davison

Submitted in part fulfilment of the requirements for the degree of PhD in Computing and the Diploma of Imperial College London. This thesis is entirely my own work, and, except where otherwise indicated, describes my own research.

Abstract

For embodied agents to interact intelligently with their surroundings, they require perception systems that construct persistent 3D representations of their environments. These representations must be rich; capturing 3D geometry, semantics, physical properties, affordances and much more. Constructing the environment representation from sensory observations is done via Bayesian probabilistic inference and in practical systems, inference must take place within the power, compactness and simplicity constraints of real products. Efficient inference within these constraints however remains computationally challenging and current systems often require heavy computational resources while delivering a fraction of the desired capabilities.

Decentralised algorithms based on local message passing with in-place processing and storage offer a promising solution to current inference bottlenecks. They are well suited to take advantage of recent rapid developments in distributed asynchronous processing hardware to achieve efficient, scalable and low-power performance.

In this thesis, we argue for Gaussian belief propagation (GBP) as a strong algorithmic framework for distributed, generic and incremental probabilistic estimation. GBP operates by passing messages between the nodes on a factor graph and can converge with arbitrary asynchronous message schedules. We envisage the factor graph being the fundamental master environment representation, and GBP the flexible inference tool to compute local in-place probabilistic estimates. In large real-time systems, GBP will act as the 'glue' between specialised modules, with attention based processing bringing about local convergence in the graph in a just-in-time manner.

This thesis contains several technical and theoretical contributions in the application of GBP to practical real-time inference problems in vision and robotics. Additionally, we implement GBP on novel graph processor hardware and demonstrate breakthrough speeds for bundle adjustment problems. Lastly, we present a prototype system for incrementally creating hierarchical abstract scene graphs by combining neural networks and probabilistic inference via GBP.

Acknowledgements

I'm extremely grateful for all the support I have received over the course of my studies, without which the work in this thesis would not have been possible.

Principally I'd like that thank my supervisor Prof. Andrew Davison for his excellent guidance and advice throughout my thesis. He has shown me the importance of thinking deeply and critically and avoiding a herd mentality within research. It has been a really enjoyable journey working with Andy in both forming high level long term research visions and in working through the low level details of Gaussian belief propagation. I'm also immensely grateful to all my collaborators and colleagues in the Dyson Robotics Lab and at Meta AI. In particular I'd like to thanks Edgar Sucar, Riku Murai, Kentaro Wada, Seth Nabarro, Talfan Evans, Tristan Laidlow and Mustafa Mukadam for countless fascinating discussions.

I'm very fortunate to have had wonderful support throughout my PhD from my friends and family. Papi and Sana have been a great inspiration and it was always lovely to catch up with them in College. Lastly, Ash and my housemates have been amazingly supportive and kind throughout.

Contents

List of Tables

List of Figures

CHAPTER 1

Introduction: Gaussian Belief Propagation for Spatial AI

Contents

In this introduction, we lay out from the top down our argument for Gaussian belief propagation as a strong algorithm for inference in Spatial AI. We begin by discussing perception in general before focusing in on the real-time embedded perception capabilities required in Spatial AI. Existing methods are then discussed before motivating decentralised graph based inference in the context of evolving hardware and scene representations. Lastly, we introduce Gaussian belief propagation and place it amongst other related algorithms.

1.1 Perception

We begin by setting out the ambitious goal of building general embodied artificial intelligence (AI) systems that can interact usefully with their environments. When building such systems, based on our understanding of decision making in humans and other animals, we often separate decision making into the two stages of **perception** and **action**. This separation, makes our embodied AI systems more structured, modular and interpretable by enforcing a two stage approach in which a perception module first processes observations into an environment representation that is then passed to an action module to make decisions. The alternative is end-to-end decision making in which observations are directly regressed to actions, however for tasks that require long term spatial memory this approach can struggle [Gervet et al., 2022].

In this thesis, we focus on the perception problem, defined as the task of building and maintaining a representation of the environment based on incoming sensory observations. These observations could be visual, auditory, tactile or other non-human modalities such as depth scans from a LiDAR sensor or spectroscopy measurements. One challenge with considering perception in isolation is that the environment representation should depend on the specific task at hand. For example, a self-driving car should focus perception on detecting road markings while an indoor robot assistant may be most concerned with mapping objects within a house.

Despite these variations, we believe that there are a number of core properties that general perception systems for a broad range of challenging tasks should share. Focusing on building perception systems with these general capabilities is a worthy and ambitious goal, as it may be the basis for generally intelligent embodied systems of the future and in the shorter term they can be modified for specific narrow tasks. We propose that in order to enable general intelligent action, we need perception systems that produce environment representations with the following properties:

3D geometric understanding. As the world is 3D we of course need our environment representation to have some understanding of 3D geometry. We are intentionally vague about what we mean by 3D geometric understanding. The geometry could be represented explicitly (e.g. stored in a voxel grid, point cloud or mesh) or implicitly (e.g. contained within the weights of a fully end-to-end system or some black box neural model such as a NeRF [Mildenhall et al., 2020]) or some combination of both. Clearly an understanding of 3D geometry is fundamental for any task that involves moving within or interacting with the environment.

Generative. The goal in perception is to infer the internal parameters of the environment repres-

entation from sensory observations and there are two main paradigms for inference. The first is inverse or **discriminative modelling** in which input observations are processed in a feedforward manner to produce the internal parameters. The second involves a **generative model** which is run in the forward direction, generating simulated observations from a particular configuration of the internal parameters. Inference in the generative paradigm is performed by using the generative model to guide iterative adjustments to the internal parameters so that the simulated observations and true observations are similar. These adjustments are usually performed through optimisation in which we seek the most likely internal parameters given the observations.

The generative modelling paradigm is generally preferred in practical perception systems for a number of reasons. First, most discriminative models can only process fixed-sized inputs, although recent approaches using graph neural networks [Bronstein et al., 2017a] can go beyond this requirement. Generative models on the other hand are able to flexibly fuse any type of full or partial observation by comparing against the relevant part of the simulated observation. Second, generative modelling is simpler as the world is more ordered in the forward causal direction [Welling, 2020]. As a result, it is easier to embed biases, priors or structure representing known physical processes in the generative direction, for example embedding volume rendering in image generation. On the other hand, we can embed some invariances and equivariances in discriminative models [Cohen and Welling, 2016], but it remains difficult to produce robust discriminative models.

Generative modelling is also crucial for intelligent decision making in which an agent can run mental simulations to evaluate the effects of actions. This is the basis of model based reinforcement learning [Sutton and Barto, 2018]. A further advantage of generative modelling is the ability to hallucinate/infer partially observed properties of the environment and detect when observations do not match up with expectations. A final difference is that discriminative inference is a one-shot process, while generative inference is a slow iterative process. There are speculative arguments [Bengio, 2019, Kahneman, 2017] that one-shot discriminative inference may better suited to simpler intuitive tasks while slow generative inference may be more appropriate for reasoning.

Interpretable. This proposition is somewhat contested, but there is a strong argument for interpretable environment representations. Most practical perception systems today are based on traditional geometric estimation problems like simultaneous localisation and mapping (SLAM) or visual inertial odometry (VIO). In these problems, Bayesian probabilistic inference and graphical models provide the fundamental framework for building interpretable environment representations and for forward modelling. Given that these core geometric estimation problems will surely

remain the foundation of future Spatial AI systems, we argue that we should build additional perception capabilities on top of this same Bayesian graphical modelling framework.

Deep learning has demonstrated the power of learned hierarchical abstractions for both discriminative and generative modelling and there are those that argue that end-to-end black box models are sufficient for representing environments for decision making. However, we believe that for environment representations to be truly useful, we should embed learned abstractions into our existing Bayesian graphical model framework. This has the advantage of being a principled Bayesian framework, allowing us to build on existing robust geometric estimation systems and also maintaining measures of uncertainty which can be useful for continual learning. A further advantage is that we can make diverse interpretable queries to our generative model – we simply condition on variables we know and use inference to determine properties of the desired variables. Neural models are not interpretable nor flexible enough to handle these diverse queries and can only carry out predetermined feedfoward computations.

1.2 Spatial AI

Having outlined our goals and the properties of an ideal perception system we now define Spatial AI. Spatial AI is the real-time vision-driven capability that robots and other devices need to understand and interact intelligently with the spaces around them, while satisfying the constraints such as power usage, compactness, robustness and simplicity enforced by real products. It is not a static but an incremental problem, where a persistent scene model with many heterogeneous elements must be stored and updated continually using data from various sources. Some data will be a flow of geometric measurements from a metric sensor; other measurements could come from a neural network; and yet more could be prior information from assumptions made at the start or communicated later on, such as the calibration parameters of a robot's drive system. All of this data must be combined consistently into the chosen environment representation, which could be complicated and heterogeneous, consisting of multiple geometric and semantic representations such as meshes, volumes, neural fields, learned shape spaces, CAD models and semantic labels.

1.3 Probabilistic Estimation on Factor Graphs

We now dig deeper and consider in more detail the computation involved in performing Bayesian probabilistic inference on graphical models represented by Spatial AI problems. Given incre-

mental uncertain measurements coming from priors, cameras, other sensors, the goal is to construct a persistent environment representation in real-time. Specifically, the environment is represented by a number of unknown/hidden variables (robot locations, map geometry, map labels, etc.), and our goal is to extract estimates of these quantities.

Bayesian probability theory [Jaynes, 2003] is the fundamental framework for consistent fusion of many uncertain sources of data and it is at the core of practical systems in machine learning and robotics [Ghahramani, 2015]. A probabilistic model relates unknown variables of interest to observable, known or assumed quantities and most generally takes the form of a graph whose connections encode those relationships. **Inference** is the process of forming the posterior distribution to determine properties of the unknown variables, given the observations, such as their most probable values (MAP inference) or their full marginal distributions (marginal inference).

Factor graphs [Dellaert, 2021] are a powerful and general representation of the probabilistic structure of inference problems. They are undirected bipartite graphs whose nodes are either variables or factors. The variables represent unknown numerical parameters of the system whose values we wish to estimate. The factors which join these variables represent constraints imposed by measurements from sensors or other information about the system (such as priors) which we are directly able to access. Each factor connects to the subset of variables it depends on, and specifies a probabilistic dependence between this subset given any observations that the factor may depend on. Strictly observations can be treated as observed variables in the graph, but it is often simpler to treat them as part of the probabilistic factor function.

A variable is denoted x_i, and a factor is denoted f_s. The subset of variables which is connected to a particular factor f_s is denoted $\mathbf{x}_s$. The interpretation is that $f_s(\mathbf{x}_s) = p(z_s|\mathbf{x}_s)$ is the probability of the numerical measurement z_s captured at factor node f_s given the variables $\mathbf{x}_s$. The topology of a factor graph defines the factorisation structure of the whole probabilistic model, in that all factors f_s are independent of each other. The vector of all variables is $\mathbf{x} = [x_1, x_2, \ldots]^\top$, and therefore the total joint probability distribution over all variables is the product of all factors:

$$p(\mathbf{x}) = \prod_s f_s(\mathbf{x}_s) \,. \tag{1.1}$$

Interesting estimation problems in computer vision and robotics can invariably be analysed to determine their factor graph structure. Dellaert and colleagues in particular [Dellaert, 2021, Dellaert and Kaess, 2017] have played an important role in increasing understanding of the power of the factor graph formulation in our field.

Since each factor is a function of some subset of the variables, the joint distribution is some tangled function of all of the variables involved in the graph. Our goal in inference is to determine the **marginal probability distributions** of the variables of interest – this gives an individual probabilistic estimate which takes all of the measurement information in the factors into account. The tangled form of the product which is the full joint distribution means that inference is usually a computationally challenging problem.

1.3.1 Existing Approaches

Almost all serious, scalable probabilistic estimation is based on the core assumption of Gaussianity in 'most' measurement distributions and 'most' posterior variable distributions, 'most' of the time. We say this with full knowledge that many other representations of distributions have been used, from sampling to other explicit functional parameterisations. But again and again, we come back to Gaussians due to their fundamental properties of fitting real-world statistical processes and the efficient representation of high-dimensional distributions they allow as the 'central' distribution of probability theory [Jaynes, 2003].

The most important techniques in current Spatial AI estimation are all Gaussian-based techniques such as Extended Kalman Filtering and non-linear least squares estimation. Gaussian-based methods have very close links to linear algebra, because optimising Gaussian likelihoods is equivalent to least-squares minimisation which involves the solution of linear systems of equations. When we write down the joint probability distribution (Equation 1.1) represented by a Gaussian factor graph, the result is a product of Gaussians. Finding the most probable variable values is equivalent to minimising the negative log of this probability distribution, which is a sum of terms which are quadratic functions of the variables. To find the minimum of this sum, we form the full information matrix for the problem and then solve the resulting linear system involving this information matrix. If the Gaussian measurement functions are non-linear in the variables this is done iteratively, forming a new linear system at each step.

For iterative non-linear least squares solvers the core computational cost is solving the linear system at each step. The key to efficiency is to exploit sparsity structure in the information matrix which is inverted to solve the linear system. The information matrix is formed using the Jacobians $\mathtt{J}$ of the measurement functions and the precision matrices Λ as: $\mathtt{J}^\top \Lambda \mathtt{J}$. Many decades of work have been devoted to studying the structure of this matrix and efficient algorithms for inverting it. Many classical inversion methods take advantage of sparsity patterns or some degree of par-

allelisation, such as Cholesky decomposition, Conjugate gradients, Jacobi method, Gauss-Seidel method, Red-Black ordering, Multigrid, etc. There are also numerous highly efficient non-linear least squares solvers such as Ceres [Agarwal and Mierle, 2012], g20 [Kummerle et al., 2011] and GTSAM [Dellaert, 2012] that are widely used in Spatial AI problems. Good recent discussion of different optimisation methods in the context of robot vision was given by PhD theses by Zienkiewicz [Zienkiewicz, 2017], Engel [Engel, 2017] or Newcombe [Newcombe, 2012] or in the Ceres Manual [Agarwal and Mierle, 2012]. We also provide a more detailed description of inference methods related to Gaussian belief propagation in Section 1.6.

Particular sub-problems in Spatial AI have well-known information matrix structure and consequently certain solvers are well-suited to their specific sparsity structure. For instance, bundle adjustment is the problem of jointly refining a relatively small number of cameras poses and a large number of 3D points that are observed by these cameras. On a CPU bundle adjustment is well tackled using Cholesky decomposition [Triggs et al., 1999] or on a GPU by the conjugate gradient method [Wu et al., 2011]. Surface reconstruction on a regular grid, where measurements from a sensor are combined with smoothness priors, can be parallelised with methods like the Primal Dual algorithm [Chambolle and Pock, 2011]. Pure visual-inertial odometry can be tackled well with sliding window filtering or non-linear optimisation [Mourikis and Roumeliotis, 2007, Leutenegger et al., 2014].

In Spatial AI, extra difficulty arises due to the fact that the estimation problem is incremental; estimates are needed in real-time and measurements are continually arriving. There has been much analysis of the trade-offs between filtering approaches which marginalise out old variables such as historic robot pose estimates and others which repeatedly solve a whole estimation problem from scratch [Strasdat et al., 2010]. Approaches such as iSAM2 [Kaess et al., 2012] stand out as progress on taking a flexible approach to scalable incremental estimation. iSAM2 is a CPU algorithm that uses a dynamic data structure called the Bayes Tree to represent a good approximation to the full factor graph of SLAM problems. It does so in such as way that most updates can be carried out with local computation, with more substantial global operations needed only in response to rarer events such as loop closure.

1.3.2 Current bottlenecks

Davison [Davison, 2018], set out the case that there are still orders of magnitude of improvement needed in efficient inference performance to deliver the capabilities needed for breakthrough

products such as lightweight home robots or AR glasses. Efficient inference on Spatial AI graphs remains extremely computationally challenging and is currently a limiting factor in many embedded systems. The computational challenges arise not only due to the high computational complexity of calculations but also due to the complexity of system design.

Current prototype Spatial AI systems attempting to process this heterogeneous flow of data into complicated persistent representations often have severe limitations. Prototype systems such as iMAP [Sucar et al., 2021] and iLabel [Zhi et al., 2021] require heavy computing resources with large data flows in and out of CPU and GPU RAM while delivering a fraction of the capability needed. Other systems make strong approximations of probabilistic structure for efficient real-time performance [Rosinol et al., 2020]. For example, SemanticFusion [McCormac et al., 2017] (based on ElasticFusion [Whelan et al., 2015]) runs by decoupling camera tracking and map updates into independent, alternating estimation processes. Further, the semantic labelling does not feedback in to improve the geometric estimation (for example smoothing regions confidently labelled as floor or walls). These systems are representative examples of current prototype systems, that on the whole either require heavy compute resources and/or strong approximations to achieve real-time performance, while only delivering a small fraction of the capabilities needed.

In industry there is much ongoing effort to optimise and engineer Spatial AI systems for embedded implementation. However, there are many fundamental changes needed still across algorithms, processors and sensors to cross the performance gap required for real products.

1.4 Decentralised Graph Based Inference

Given the current bottlenecks in inference algorithms for Spatial AI, and current trends in compute hardware and scene representation, we make the following proposition.

Proposition 1. *General, efficient and scalable inference in Spatial AI requires decentralised graph based algorithms that operate via local message passing on the factor graph with in-place local processing and data storage.*

We are proposing inference algorithms which implement Spatial AI inference in a decentralised manner on a distributed computational resource, by storing the factor graph as the master representation and operating on it in-place using local computation and message passing to implement full global inference. We imagine messages continually bubbling around a large factor graph, which is changing continually with the addition of new measurement factors and variable nodes,

and perhaps never reaching full convergence, but always being close in a way which can be controlled. It may be that estimation processes will proceed in an attention-driven way, using a lot of computation to bring accurate estimates to important local areas, which then are allowed to fade to a less up-to-date state once attention moves on, in a 'just-in-time' style [Weerasekera et al., 2018].

This proposition means we must get away from the idea that a 'god's eye view' of the global structure of the graph will ever be available. We want methods where each node can operate with minimum knowledge of the whole graph structure, for example storing only local information about itself and its near neighbours. It is important here to clarify the difference between distributed systems, in which computation is spread between nodes, and decentralised systems, which are a specific type of distributed system in which decisions are made locally in the graph rather than at a central node. Distributed algorithms with a master node can require a significant amount of data transfer to and from the master node. As a result, we propose fully decentralised algorithms where computation and decision making is done in-place in the graph.

We now justify our proposition by considering developments in computing hardware and scene representation.

1.4.1 Evolution of Computing Hardware

We believe that computing hardware is at the beginning of a new era. We are moving away from a reliance on processors and memory systems designed either for completely general purpose use (CPUs) or computer graphics (GPUs) and towards an era where AI, and perhaps Spatial AI in particular, are significant enough applications to drive the development of custom computing hardware. GPUs are currently the main workhorses of AI computation and are certainly very good for many tasks in computer vision, but we believe that the future of Spatial AI compute will require the development of much more flexible storage and computation [Davison, 2018]. (We highly recommend the recent PhD thesis of Julien Martel for ambitious thinking about this whole area [Martel, 2019]).

In Sutter's article [Sutter, 2011], he anticipates a long-term trend towards a 'hardware jungle' of parallel, heterogeneous, distributed and asynchronous computing systems which communicate in a peer-to-peer manner. This will be at several levels: across networks of multiple smart devices operating in the same environment; across the many sensors, actuators and processors within individual embodied devices; and even within single processor chips themselves. Evidence of this in Spatial AI is the emergence of many specialised chips and architectures to accelerate import-

ant computer vision algorithms such as feature detection, tracking, the whole visual odometry pipeline, and convolutional neural networks (CNNs) for object detection or segmentation.

For single processor chips, a trend towards multicore designs with distributed on-core memory and ad-hoc communication is rapidly emerging [Lacey, 2019, Gui et al., 2019]. An example of this is graph processors such as Graphcore's IPU [Graphcore, 2022] which have taken quite general design choices towards enabling a different type of processing. The IPU is a massively parallel chip, where each processing core has a large amount of high performance on-chip memory and the cores are arranged in an all-to-all graph structure with rapid inter-core communication. In the IPU, computation works best when the data can be distributed around the chip close to the cores where it is operated on and there is no need to communicate with external off-chip memory. For these novel distributed processors, our measure of what is an efficient algorithm needs to change from the CPU standard of total computation time to multi-dimensional metrics also accounting for storage and data transmission.

Given this trend in computing hardware towards a 'hardware jungle' of parallel, heterogeneous, distributed and asynchronous computing systems, decentralised inference algorithms will be the key to implementing **efficient**, **scalable** and **low power** inference.

Efficiency clearly entails massive parallelism and algorithms which are distributable are the most trivial to parallelise. Algorithms that operate with purely local computation and message passing at the level of individual nodes in the graph represent the extreme of distributable algorithms and can be implemented in a node-wise parallel fashion.

Decentralisation is key to the scalability of inference algorithms, as methods which are decentralised can scale arbitrarily and leverage all available asynchronous and heterogeneous compute within the 'hardware jungle'. Sutton's 'Bitter Lesson' of machine learning (ML) research [Sutton, 2019] states that: *'general methods that leverage computation are ultimately the most effective, and by a large margin'*. A reminder of this is the success of CNN-driven deep learning, which is well suited to GPUs, the most powerful widely-available processors in recent years. This lesson suggests that decentralised inference algorithms will be the most effective at large-scale inference given the availability of large decentralised asynchronous compute resources.

Key to low power performance is intermingling data storage and computation to reduce the 'bits $\times$ millimetres' through which data is moved [Sze et al., 2017]. This is particularly true due to the closed-loop, incremental nature of Spatial AI, where new data must be continually compared to and combined with stored models. Decentralised local message passing algorithms reduce this

data movement by storing data close to where it is computed on and limiting data movement to informative messages. In the best case, message passing communication is only needed between conditionally dependent nodes in the graph when there is new information to communicate.

In contrast to decentralised inference, let us consider the opposite proposition of a global centralised inference method via a monolithic cloud-based processor. This centralised approach would not be able to leverage all heterogeneous and asynchronous compute resources and even if feasible would not be desirable, for communication bandwidth and privacy reasons.

1.4.2 Evolution of Scene Representations

Current systems both in academia and industry fall vastly short of the target of 3D generative graph-based scene representations outlined in Section 1.1. Consequently, in parallel with developments in fitting inference algorithms to hardware, innovation in expressive and compact scene representations is crucial. These representations need to be efficient to build and maintain as well as usable in conditional generative queries for decision making.

Two examples of promising novel scene representations of late are compressed coded representations learned via probabilistic auto-encoding and neural fields. CodeSLAM [Bloesch et al., 2018] and follow ups [Zhi et al., 2019, Sucar et al., 2020] use variational auto-encoders to find coded, compressed representations of geometry, semantic labels, and objects which can then be optimised based on multi-view data. Neural fields represent an environment in the weights of a coordinate based multi-layer perceptron (MLP) that can be optimised to fit to a specific scene [Park et al., 2019, Mescheder et al., 2019, Mildenhall et al., 2020]. Recent prototype systems have shown that neural fields can be an efficient and expressive representation for real-time incremental perception systems [Sucar et al., 2021, Zhu et al., 2022, Ortiz et al., 2022a].

In this thesis, we do not focus on the development of novel scene representations, but we note that increasingly scene representations are becoming more **large**, **dynamic**, **heterogeneous** and **highly interconnected** [Bengio, 2017, Davison and Ortiz, 2019]. These changes mean that decentralised inference algorithms will be well suited to the next generation of scene representations, all well as being very efficient on novel hardware.

Large. As we become more interested in building large city-scale reconstructions, the required storage and computation will become infeasible on a single centralised processor and must be distributed amongst many compute nodes.

Dynamic. Spatial AI is an incremental problem in which the factor graph is continually changing with new asynchronous measurements and abstractions. Current centralised approaches rely on fixed sparsity structure and are therefore not well suited to the dynamically changing Spatial AI graphs. Decentralised methods involve only local operations that are agnostic to the overall graph structure meaning that if parallelised, the time per iteration is largely independent of the graph structure. This makes decentralised methods well suited to inference on dynamic Spatial AI graphs.

Heterogeneous and interconnected. Lastly, scene representations are becoming increasingly heterogeneous. We have mentioned several innovations in scene representations for Spatial AI, but we envisage that the Spatial AI graph will be a highly heterogeneous hierarchical graph of abstractions, much like a scene graph [Armeni et al., 2019, Rosinol et al., 2020, Wald et al., 2020, Wu et al., 2021, Ortiz et al., 2022b] or the Live Maps vision from Meta Reality Labs [Meta, 2019]. The graph will contain many different entities: for example low level 3D geometry in the form or volumes, points, or implicit fields; large 3D structures like planes, floors and room layouts; semantic 3D objects in the form of coded shapes, CAD models or implicit fields; and other annotations such as object affordances or physics properties of surfaces such as friction coefficients. The key point is that all of these abstractions must be inferred jointly – for example geometry is useful to inferring semantics and semantics is useful for inferring geometry. The resulting factor graph will be highly heterogeneous due to all the different entities and highly interconnected due to the joint inference over all interdependent hierarchies. Heterogeneity and high interconnection erodes sparsity structure in the graph, meaning that current inference methods will be highly inefficient on Spatial AI graphs. We instead need decentralised algorithms where the cost of each local iteration does not depend on the graph structure meaning fast inference is maintained for dense heterogenous graphs.

1.5 Gaussian Belief Propagation

Having motivated the need for decentralised graph based inference that operates with local storage and computation, we now propose belief propagation (BP) and specifically Gaussian belief propagation (GBP) as a strong candidate algorithm. BP is able to perform in-place inference on a factor graph with entirely local storage and processing at each node in the graph and message passing communication. Each variable and factor node processes messages with no knowledge about the rest of the graph other than its direct neighbours, and BP can converge with arbitrary,

asynchronous message passing schedules which need no global coordination. In a certain sense, an algorithm which works like this represents 'assuming the worst' – that no knowledge of the structure of an estimation problem is available to enable intelligent design of processing.

This is the reason both that BP is well worth studying as an end point in a continuum of possible methods, but also that it is unlikely to form the whole solution to practical estimation. What we foresee is that BP could form a general estimation 'glue' between specifically engineered hardware/software modules for particular tasks; or be particularly valuable in highly dynamic, rapidly reconfiguring estimation problems where management of computation can carry on in a decentralised way.

Belief propagation has an extensive literature and is a well-known inference algorithm for calculating per-node marginals from a joint distribution. BP was originally developed in the 1980s by Pearl [Pearl, 1988] as an exact inference algorithm on tree-structured graphs. In tree graphs, BP guarantees exact marginal computation with one sweep of messages down from an arbitrary root node to the leaf nodes and then back up [Bishop, 2006]. 'Loopy' belief propagation applies the same message passing rules to loopy graphs with cycles and empirically often accurately computes the true marginals for these graphs [Pearl, 1988, Kschischang et al., 2001, Murphy et al., 1999]. Gaussian belief propagation is a special case of loopy BP applied to Gaussian graphical models in which we also infer Gaussian-distributed marginals. GBP has both more extensive mathematical guarantees of correctness [Weiss and Freeman, 2000] and stronger empirical performance [Bickson, 2008] than general loopy BP. Unlike graph neural networks (GNNs) [Scarselli et al., 2008, Bronstein et al., 2017a, Battaglia et al., 2018] which learn edge and node updates that are applied over a fixed number of message passing steps, BP applies probabilistic message passing updates with iterative convergent behaviour.

We are not the first to consider applying GBP to Spatial AI problems, although we believe that it has received much less interest than it deserves in this context. BP has been shown to work well for image processing tasks on regular image grids [Felzenszwalb and Huttenlocher, 2006]. In 3D computer vision, BP with discrete variables has been used to provide initialisations for non-linear least squares bundle adjustment [Crandall et al., 2011]. Paskin *et al.* [Paskin, 2003] built a junction tree of a filtered SLAM graph which was kept sparse by removing edges and used GBP for inference. Most relevant to us, Ranganathan *et al.* in 'Loopy SAM' [Ranganathan et al., 2007] used GBP for a robot mapping application, and their experiments have many similarities with the demonstrations we will give later in this thesis. Outside of Spatial AI, loopy BP is perhaps best known for its successful application to error-correcting codes [McEliece et al., 1998].

Despite these applications, GBP has not yet been applied broadly to machine learning or Spatial AI problems. One issue that has precluded the use of general Loopy BP is that it lacks convergence guarantees. However, a second and perhaps more relevant issue given modern hardware trends, is that its computational properties have not fitted the dominant processing paradigms of recent decades, CPUs and GPUs. Consequently, other factor graph inference algorithms have been preferred which take advantage of global problem structure to operate much more rapidly and robustly than belief propagation on a CPU.

As we have outlined, we believe that recent advances in computing hardware and scene representation make this the right time to re-evaluate decentralised inference algorithms and in particular GBP as a candidate inference algorithm for Spatial AI. Indeed there has been a recent resurgence in interest in the algorithm [George et al., 2017, Lázaro-Gredilla et al., 2021, Satorras and Welling, 2021, Kuck et al., 2020, Ortiz et al., 2020, Opipari et al., 2021]. Much of this work has investigated combining deep learning with GBP [Satorras and Welling, 2021, Kuck et al., 2020, Opipari et al., 2021] or using GBP for inference in structured probabilistic generative models [George et al., 2017, Lázaro-Gredilla et al., 2021].

In Box 1.1, we summarise the key properties of GBP that make it a strong algorithmic framework for probabilistic estimation in Spatial AI.

Box 1.1: Key Properties of Belief Propagation

1. **Decentralised** - can do inference in-place on the factor graph with local distributed processing, storage and message passing.
    - Distributability enables GBP to leverage any extra available compute whether on a single chip or in networks of processors.
    - Can achieve low power implementation due to minimal data transfer only between conditionally dependent nodes.
    - Can operate via 'just in time' convergence focusing compute on task-relevant regions - this is also attractive from an energy consumption perspective.
2. **Probabilistic** - estimates uncertainties by doing marginal inference as opposed to MAP inference.
3. **Iterative and convergent** - is not run over a fixed number of steps.
    - GBP can be run continually in the background and as new data arrives, the problem can be arbitrarily edited without interrupting inference.
4. **Asynchronous** - convergence can be reached via asynchronous updates.
    - This is important in decentralised systems without a global clock or where communication delays are non-negligible such as in edge computing.

Another great strength of GBP is the straightforward and fully local nature of implementation. Unlike most previous estimation methods that are instantiated in large and highly optimised solver libraries for a CPU, the details of GBP can be easily and efficiently implemented on any particular distributed platform. We hope that a set of standard formats for how these platforms should pass messages between each other will emerge and enable our vision of GBP as the glue between various specialised estimation methods. We take steps towards designing this standard message passing format for localisation in our Robot Web paper [Murai et al., 2022] (described in Section 3.5). This vision is closely related to the idea of clustered belief propagation in which subsets of nodes form clusters that are stored on the same device and use a centralised method for local inference within the cluster and GBP for global inference taking into account constraints from other clusters.

In the limit of a purely distributed implementation, each node (either variable or factor) can be

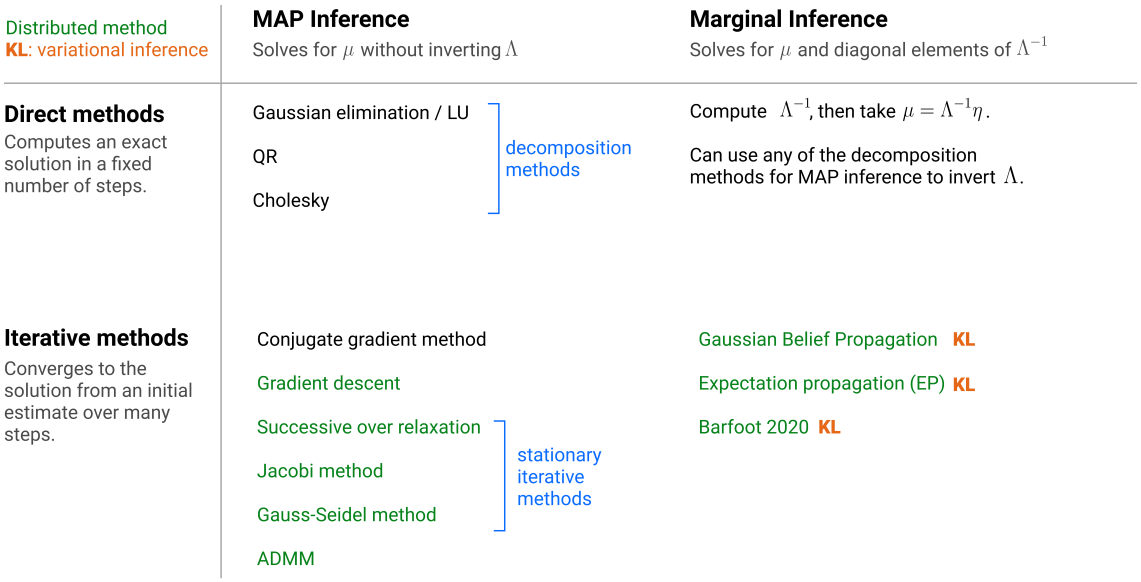


Figure 1.1: Non-exhaustive table of algorithms for solving MAP and marginal inference in Gaussian models. (See Halsted *et al.*'s [Halsted et al., 2021] survey paper for a good review of distributed methods for pose graph optimisation.)

hosted on separate processor, or tile of a graph processor. The most intensive computation a node needs to carry out is the matrix inversion needed for marginalisation at a factor node and the dimension of this matrix is usually small. In the common case of graphs with only unary or binary factors (which connect to one or two variable nodes), the maximum dimension is the maximum individual variable node dimension. In addition to computationally cheap operations, communication is also efficient as all messages between nodes take the form of usually low-dimensional Gaussians, which can be represented by small vectors and matrices.

1.6 Related Algorithms

In this section, we discuss algorithms related to GBP for MAP and marginal inference. Inference in Gaussian models is equivalent to solving the linear system $\Lambda\mu = \eta$; solving for μ in MAP inference, and for μ and the diagonal elements of Λ^{-1} in marginal inference. When the measurement functions have a non-linear dependence on the variables, inference is performed by iteratively solving linearised Gaussian versions of the true non-linear problem. MAP inference in Gaussian models is equivalent to minimising a non-linear least squares objective.

Figure 1.1 presents a non-exhaustive overview of related algorithms and we will discuss some of the approaches in more detail in the remainder of this section.

There are many efficient libraries for inference of non-linear problems such as Ceres [Agarwal

and Mierle, 2012], g20 [Kummerle et al., 2011], GTSAM [Dellaert, 2012], Theseus [Pineda et al., 2022] and SymForce [Martiros et al., 2022]. These libraries generally implement MAP inference and use either trust region methods or line search to guide the repeated linear steps. Trust region methods approximate the objective using a model within a trust region; for example, Gauss-Newton uses a quadratic model meaning the factors are approximated as Gaussians as in GBP. Line search methods choose a descent direction and then step size at each iteration. These libraries usually employ direct linear solvers based on matrix decomposition which compute the exact solution to the linear system in a fixed number of steps. Decomposition methods factorise the matrix Λ into a product of matrices that are easier to invert and often exploit sparsity in the linear system. In contrast to direct methods, iterative methods converge to the solution of the linear system over many iterations. Conjugate gradient is a common iterative method that these libraries often employ for solving large sparse linear systems.

These libraries focus on efficient centralised inference generally on a CPU with some providing GPU support. We now turn to discuss distributed inference algorithms and relate them to GBP.

Distributed Gradient Descent (DGD). Gradient descent can be implemented as a decentralised message passing algorithm on a factor graph where factor to variable messages are local gradients and variable nodes average local incoming gradients to update their state. Being a first order method, the convergence of distributed gradient descent is slow and most research focuses on finding accelerated DGD methods [Olson et al., 2006, Grisetti et al., 2009, Cristofalo et al., 2020]. Block coordinate descent successively minimises along blocks of coordinate directions and has been used for distributed pose graph optimisation (PGO) [Tian et al., 2019] as well as SLAM [Tian et al., 2021]. Majorisation minimisation methods successively minimise an upper bound on the objective and have been used for distributed PGO [Fan and Murphey, 2020]. Like GBP, a positive property of DGD is that it can be implemented in a stochastic asynchronous manner [Tian et al., 2020], however DGD tends to converge a lot slower than GBP. The reason for this is that GBP messages are full Gaussian distributions consisting of a mean and a covariance describing the uncertainty. These messages are therefore far more informative than the local direction of steepest descent in DGD.

Stationary iterative methods. Stationary iterative methods solve MAP inference by updating an estimate using a residual based on the original linear system. Common algorithms are the Gauss-Seidel method, Jacobi method, successive over-relaxation and symmetric successive over-relaxation. These methods generally update the variables one at a time and can be implemented as decentralised message passing with a fixed schedule [Barooah and Hespanha, 2005, Aragues et al.,

2011]. In Spatial AI, stationary iterative methods have been employed for mapping [Duckett et al., 2000, Choudhary et al., 2017] and refining sensor networks [Delouille et al., 2004]. Recently distributed Gauss-Seidel has been used as a back-end optimisation module in DOOR-SLAM [Lajoie et al., 2020] and Cieslewski *et al.* [Cieslewski et al., 2018]. Ranganathan *et al.* [Ranganathan et al., 2007] showed that computing the MAP solution using GBP is equivalent to a modified form of Gauss-Seidel relaxation. As GBP does full marginal inference, maintaining uncertainties over the beliefs, it tends to converge faster than stationary iterative methods while also providing marginal covariance estimates.

Alternating Direction Method of Multipliers (ADMM). A popular family of methods for distributed MAP inference are based on the consensus alternating direction method of multipliers (ADMM) [Rockafellar, 1976]. ADMM minimises an augmented Lagrangian with an additional penalty term on feasible solutions (as in the method of multipliers). As in dual decomposition, it updates the primal variables in a distributed manner and then gathers the residual updates to update the dual variables. Although most implementations update the dual variables at a centralised master node, the dual update can also be distributed making ADMM a fully decentralised message passing algorithm. ADMM can be viewed as doing sequential/alternating Gauss-Seidel updates on the primal and dual variables. ADMM methods have several drawback; they have redundant dual variables and computation for achieving consensus, they require careful tuning of the penalty parameter, they can be slow to converge, and lastly most implementations use a master node.

ADMM has been used in various distributed bundle adjustment applications [Eriksson et al., 2016, Ramamurthy et al., 2020, Mayer, 2019, Zhu et al., 2017, Zhang et al., 2017a, Zhou et al., 2020b, Demmel et al., 2020]. Eriksson *et al.* [Eriksson et al., 2016] use point consensus between subproblems with Douglas-Rachford proximal splitting of the cameras. To reduce the overhead of point consensus, camera consensus and splitting points can be more efficient [Zhang et al., 2017a]. Demmel *et al.* [Demmel et al., 2020] use a consensus method similar to parallel block coordinate descent for photometric bundle adjustment problems.

Expectation Propagation (EP). EP [Minka, 2013] is a variational inference method that can be implemented as decentralised message passing on the factor graph. EP approximates the true posterior p with a factorised variational distribution of exponential factors: $q(\theta) = \prod_i \tilde{f}_i(\theta)$. It operates by iterating through the factors and minimising the forward KL divergence for a single factor by matching sufficient statistics (moment matching). Note that EP is not minimising the forward KL between the approximate and true posteriors, but rather locally at the factors. Similar to GBP, EP has no convergence guarantees but if it does converge the stationary point is a fixed

point of the objective. If the factorisation of q is chosen such that it is fully factorised and all factors are unary, then the EP updates are equivalent to local belief propagation message passing. EP has been successfully applied to skill ranking in Microsoft's TrueSkill [Minka et al., 2018] however, despite its potential utility, to the best of our knowledge it has not been applied to Spatial AI problems.

Barfoot's method. Barfoot [Barfoot, 2020] proposed a decentralised message passing algorithm for marginal inference based on results from Takahashi *et al.* [Takahashi, 1973]. The algorithm has similarities to GBP but is guaranteed to converge in both the mean and covariance to the true marginal quantities. These guarantees come at the cost of additional communication edges and memory compared to GBP. These additional edges mean that is not fully local like GBP as communication is required between conditionally independent nodes. This method has not yet been applied to real practical non-linear problems in Spatial AI and we look forward to future developments and applications.

Other methods. There exist many approaches that do not fall cleanly into any of the described method classes. Leung *et al.* show that decentralised consensus-based message passing can achieve exact centralised solutions for distributed PGO [Leung et al., 2010] and SLAM [Leung et al., 2012]. Decentralised data fusion smoothing and mapping (DDF-SAM) [Cunningham et al., 2010] uses single robot smoothing and mapping with Gaussian elimination and a local neighbourhood graph optimiser for distributed SLAM.

We have outlined a whole range of different distributed inference methods, however we have only come across two algorithms that perform marginal inference via fully decentralised message passing with local storage and processing. Gaussian belief propagation and Barfoot's method both possess these properties and we identify them both as strong candidate inference algorithms for Spatial AI. We have not included EP here, which also has these properties, due to the fact that it does not perform well for multi-modal distributions which are ubiquitous in Spatial AI. Although Barfoot's method can provide some theoretical guarantees unlike GBP, it has yet to be demonstrated for real Spatial AI problems and involves extra computation and communication compared to GBP. There are clearly strong connections between GBP and Barfoot's method and we hope that ideas can be transferred to bring convergence guarantees to GBP.

Our key argument in this introduction has been that we want a marginal inference method that is decentralised, probabilistic and can converge asynchronously and GBP stands as the strongest candidate algorithm. It represents the extreme case that maximises parallelism and minimises communication – two principles that are at the core of scalable and low-power computation and

are key to the vision of GBP as the 'glue' between many decentralised specialised modules.

We now detail the contributions carried out in this thesis before laying out the structure of the remainder of the thesis.

1.7 Contributions

The work in this thesis is based on the following publications:

- Andrew J. Davison, Joseph Ortiz. **FutureMapping 2: Gaussian Belief Propagation for Spatial AI**. *arXiv preprint arXiv:1910.14139, 2019*. [Davison and Ortiz, 2019]
- Joseph Ortiz, Talfan Evans, Andrew J. Davison. **A Visual Introduction to Gaussian Belief Propagation**. *Self-published at* `https://gaussianbp.github.io/`. [Ortiz et al., 2021]
- Joseph Ortiz, Mark Pupilli, Stefan Leutenegger, Andrew J. Davison. **Bundle Adjustment on a Graph Processor**. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* 2020. [Ortiz et al., 2020]
- Joseph Ortiz, Talfan Evans, Edgar Sucar, Andrew J. Davison. **Incremental Abstraction in Distributed Probabilistic SLAM Graphs**. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA),* 2022. [Ortiz et al., 2022b]
- Riku Murai, Joseph Ortiz, Sajad Saeedi, Paul Kelly, Andrew J. Davison. **A Robot Web for Distributed Many-Device Localisation**. *arXiv preprint arXiv:2202.03314, 2022*. [Murai et al., 2022]

While not described directly, the following publications were done in conjunction with this thesis:

- Edgar Sucar, Shikun Liu, Joseph Ortiz, Andrew J. Davison. **iMAP: Implicit Mapping and Positioning in Real-Time**. *Proceedings of the International Conference on Computer Vision (ICCV), 2021*. [Sucar et al., 2021]
- Joseph Ortiz, Alexander Clegg, Jing Dong, Edgar Sucar, David Novotny, Michael Zollhoefer, Mustafa Mukadam. **iSDF: Real-Time Neural Signed Distance Fields for Robot Perception**. *Proceedings of Robotics: Science and Systems (RSS), 2022*. [Ortiz et al., 2022a]

- Luis Pineda, Taosha Fan, Maurizio Monge, Shobha Venkataraman, Paloma Sodhi, Ricky Chen, Joseph Ortiz, Daniel DeTone, Austin Wang, Stuart Anderson, Jing Dong, Brandon Amos, Mustafa Mukadam. **Theseus: A Library for Differentiable Nonlinear Optimization**. *Neural Information Processing Systems (NeurIPS), 2022*. [Pineda et al., 2022]

Along with this thesis, we provide several open source code repositories for GBP:

- For our project *Bundle Adjustment on a Graph Processor* [Ortiz et al., 2020], we provide code for both our Python and Poplar implementations at: https://github.com/joeaortiz/gbp and https://github.com/joeaortiz/gbp-poplar.
- As part of our article *A Visual Introduction to Gaussian Belief Propagation* [Ortiz et al., 2021], we provide a GBP Library Colab Notebook as a simple starting point the interested reader at: https://colab.research.google.com/drive/1-nrE95X4UC9FBLR0-cTnsIP_XhA_PZKW?usp.

1.8 Thesis Structure

The remainder of the thesis is structured as follows:

Chapter 2 – Technical Introduction to Gaussian Belief Propagation. We formally introduce probabilistic inference and factor graphs before deriving belief propagation for tree graphs. Gaussian belief propagation for Gaussian models is then presented before discussing how GBP can be extended to handle non-Euclidean variables with Lie Group theory. This section contains standard derivations apart from the extension to non-Euclidean variables which describes original contributions from the Robot Web paper [Murai et al., 2022].

Chapter 3 – Beyond the Standard Algorithm. This chapter presents details and tricks for making GBP work well on real practical Spatial AI problems. We use a variety of 1D and 2D SLAM-like problems to illustrate the utility of message schedules, robust factors and other details such as damping. Notably, we show that GBP can converge well with arbitrary random message schedules and that local robust factors can effectively identify outlying measurements. These findings are novel contributions from our two position/tutorial papers: FutureMapping2 [Davison and Ortiz, 2019] and the online visual introduction to GBP [Ortiz et al., 2021]. In the final part of this chapter, we briefly detail our novel Robot Web communication protocol for distributed private GBP message passing between networks of devices [Murai et al., 2022].

Chapter 4 – Bundle Adjustment on a Graph Processor. This chapter presents the work from our paper [Ortiz et al., 2020] which applies GBP, implemented on a graph processor, to bundle adjustment. We demonstrate rapid breakthrough inference speeds, 24x faster than the Ceres solver on a CPU.

Chapter 5 – Incremental Abstraction in Distributed Probabilistic SLAM Graphs. This chapter presents work from our paper [Ortiz et al., 2022b] which proposes an incremental abstraction framework for SLAM as well as a routing method for implementing GBP on fixed communication graphs. We demonstrate the planar abstraction of real scenes producing more dense, semantic and compact scene graphs.

Chapter 6 – Conclusions. Here we conclude the thesis with a discussion of the research presented and suggestions for future work.

CHAPTER 2

Technical Introduction to Gaussian Belief Propagation

Contents

2.1 Probabilistic Inference

Probabilistic inference is the problem of estimating statistical properties of unknown variables $\mathbf{x}$ from known or observed quantities $\mathbf{d}$ (the data). For example, one might be interested in inferring tomorrow's weather ($\mathbf{x}$) from historic data ($\mathbf{d}$), or the 3D structure of an environment ($\mathbf{x}$) from a video sequence ($\mathbf{d}$).

Bayesian inference proceeds by first defining a probabilistic model that describes the relationships between data and variables: $p(\mathbf{x}, \mathbf{d})$. The sum and product rules of probability are then used to form the probability of the unknown variables given the data, which is known as the posterior distribution:

$$p(\mathbf{x}|\mathbf{d}) = \frac{p(\mathbf{x}, \mathbf{d})}{p(\mathbf{d})} \,. \tag{2.1}$$

The sum rule is $p(\mathbf{x}) = \sum_{\mathbf{y}} p(\mathbf{x}, \mathbf{y})$ and the product rule is $p(\mathbf{x}, \mathbf{y}) = p(\mathbf{y}|\mathbf{x})p(\mathbf{x})$. The posterior summarises our belief about $\mathbf{x}$ after seeing $\mathbf{d}$ and can be used for decision making or other downstream tasks.

Given the posterior, we can compute various properties of $\mathbf{x}$, for example:

1. The most likely configuration of the variables $\mathbf{x}_{\text{MAP}} = \arg\max_{\mathbf{x}} p(\mathbf{x}|\mathbf{d})$.
2. The marginal posteriors $p(x_i|\mathbf{d}) = \sum_{\mathbf{x} \setminus x_i} p(\mathbf{x}|\mathbf{d})$, which summarise our belief about each individual variable x_i given $\mathbf{d}$. Note that $\mathbf{x} \setminus x_i$ denotes all elements of $\mathbf{x}$ except x_i; we will equivalently later use the shorthand $\mathbf{x}_{-i}$.

These two calculations are known as 1) **maximum a posteriori (MAP) inference** and 2) **marginal inference** respectively. An important difference is that MAP inference produces a point estimate while marginal inference retains information about uncertainty.

2.2 Factor Graphs

The Hammersley-Clifford theorem tells us that any positive joint distribution $p(\mathbf{x})$ can be represented as a product of factors f_i, one per clique, where a clique is a subset of variables $\mathbf{X}_i$ in which each variable is connected to all others:

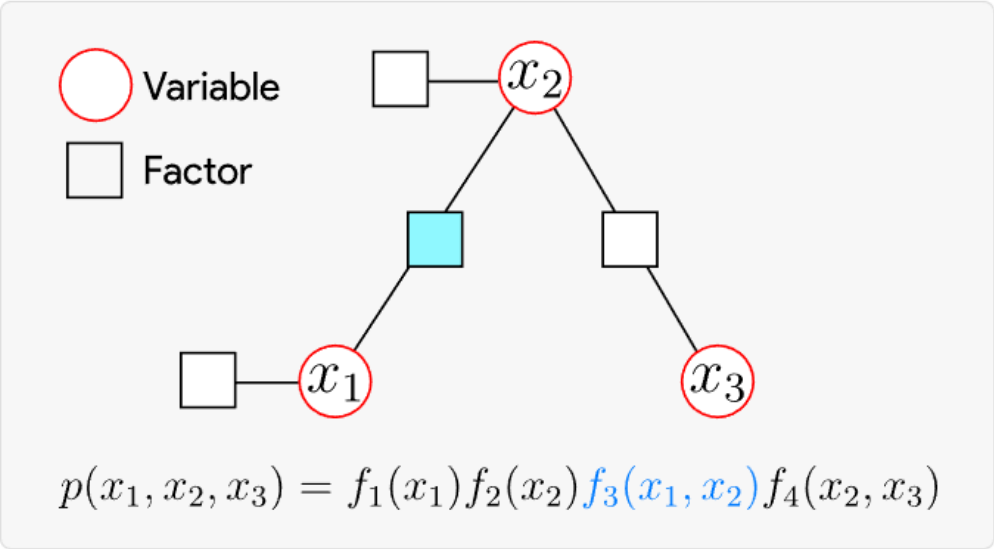


Figure 2.1: Simple factor graph representing the joint distribution over 3 variables. The joint distribution can be factorised into 4 terms, each represented in the diagram by a factor. The factor $f_3(x_1, x_2)$ is highlighted in blue in the both the diagram and the equation to show the correspondence. See the full interactive version of this figure at: https://gaussianbp.github.io/#factor_graph

$$p(\mathbf{x}) = \prod_i f_i(\mathbf{X}_i) \tag{2.2}$$

Factorised representations can be very convenient as they expose structure in a model. Factor graphs are a natural representation of this factorisation and provide a way of visualising conditional independence structure and an interpretation of some calculations as operations on the graph [Dellaert and Kaess, 2017].

In the factor graphs in this thesis, circles and squares represent variable and factor nodes respectively, with edges connecting each factor to the variables it depends on. An example of a simple factor graph is shown in Figure 2.1. By explicitly representing the factors as nodes in the graph, factor graphs clearly emphasise the conditional independence structure of the problem - the lack of a factor directly connecting two variables means they are conditionally independent given all other variables.

Mathematically, two variables x_i and x_j are conditionally independent given all other variables $\mathbf{x}_{-ij}$ if:

$$p(x_i, x_j|\mathbf{x}_{-ij}) = p(x_i|\mathbf{x}_{-ij})p(x_j|\mathbf{x}_{-ij}) \ . \tag{2.3}$$

An equivalent condition is:

$$p(x_i|x_j, \mathbf{x}_{-ij}) = p(x_i|\mathbf{x}_{-ij}) \ . \tag{2.4}$$

Intuitively, if $\mathbf{x}_{-ij}$ causes both x_i and x_j, then if we know $\mathbf{x}_{-ij}$ we don't need to know about x_i to predict x_j or about x_j to predict x_i. Conditional independence is often written in shorthand as: $x_i \perp x_j | \mathbf{x}_{-ij}$.

2.2.1 Energy based models interpretation

Factor graphs can also be presented as energy based models [LeCun et al., 2006] where each factor f_i defines an energy $E_i \geq 0$ associated with a subset of the variables $\mathbf{X}_i$:

$$f_i(\mathbf{X}_i) \propto e^{-E_i(\mathbf{X}_i)} \ . \tag{2.5}$$

This formalism is closely related to the Boltzmann distribution in statistical physics which gives the probability of a state i as a function of the energy of the state and the temperature of the system:

$$p_i = \frac{e^{-E_i/kT}}{\sum_j e^{-E_j/kT}} \ , \tag{2.6}$$

where k is the Boltzmann constant, T is the temperature of the system and j sums over all available states.

In energy based models, finding the most likely variable configuration is equivalent to minimising the negative log probability or the sum of factor energies:

$$\mathbf{x}_{\text{MAP}} = \arg\min_{\mathbf{x}} -\log p(\mathbf{x}) = \arg\min_{\mathbf{x}} \sum_i E_i(\mathbf{X}_i) \ . \tag{2.7}$$

2.3 The Belief Propagation Algorithm

In this section we present a derivation of the Belief Propagation algorithm for tree graphs before discussing *loopy* Belief Propagation on graphs with loops.

2.3.1 Derivation for Tree Graphs

We first introduce the general theory of Belief Propagation, focusing on the Sum-Product Algorithm due to Pearl [Pearl, 1982], and following the notation and derivation given in Bishop's book 'Pattern Recognition and Machine Learning' [Bishop, 2006]. Here the representation of probability distributions is not specified, and could be probability tables for discrete variables or an arbitrary probability distribution function for continuous variables. We will go on to derive the

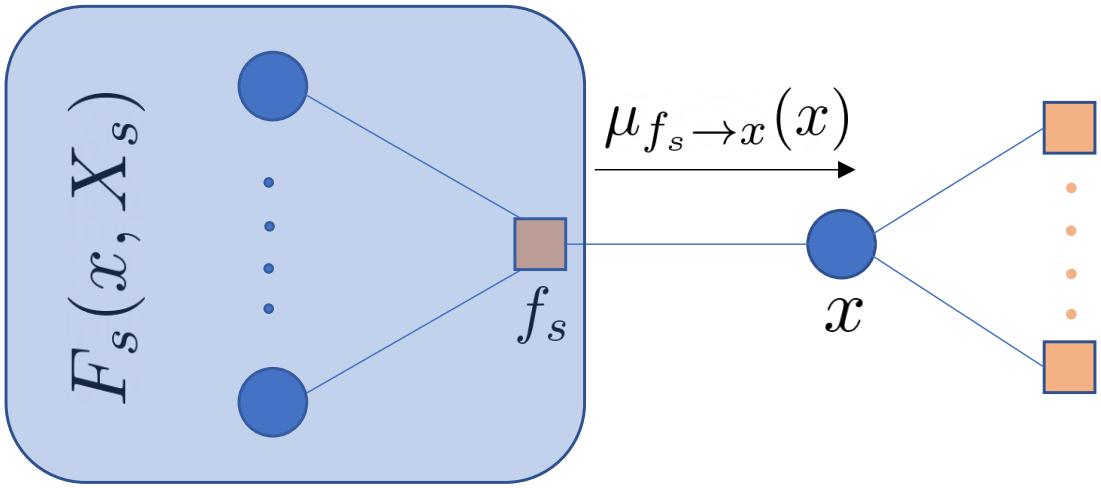


Figure 2.2: A variable node x connects to factor nodes including f_s, from which it receives message $\mu_{f_s \to x}(x)$.

specific Gaussian case in Section 2.5. We give a lot of detail on the mathematical derivations in this section, with the aim of producing a fully understandable and complete tutorial.

We start from Equation 2.2 which defines the probability distribution over all variables in a factor graph as a product of all factors. We are interested in marginal inference, in which we aim to determine the marginal distribution over variables of interest. Choosing to start with one particular variable x, the sum rule tells us that its marginal distribution is found by taking the joint distribution, and summing over all of the other variables:

$$p(x) = \sum_{\mathbf{x} \setminus x} p(\mathbf{x}) \, . \tag{2.8}$$

For the moment, we assume that our factor graph has a tree structure, which means that it has no loops, and that there is precisely one route through the graph between any two nodes.

Consider Figure 2.2 which focuses on an arbitrary variable x within a tree factor graph. Variable x is directly connected to a number of factors f_s. Every other factor in the graph is connected to x indirectly via exactly one of these factors, so we can divide the whole graph into the same number of subsets as the factors f_s, and write the whole joint probability distribution as a product of these subsets:

$$p(\mathbf{x}) = \prod_{s \in n(x)} F_s(x, \mathbf{X}_s) \, . \tag{2.9}$$

Here $n(x)$ is the set of factor nodes that are neighbours of x; F_s is the product of all factors in the group associated with f_s; and $\mathbf{X}_s$ is the vector of all variables in the subtree connected to x via f_s. Note that previously we had defined the subset of variables connected to factor f_s as $\mathbf{X}_s$;

however for this derivation, we redefine the set $\mathbf{X}_s$ to exclude x and include all variables in the subtree connected to x via f_s. Now, combining Equations 2.8 and 2.9:

$$p(x) = \sum_{\mathbf{x} \setminus x} \left[\prod_{s \in n(x)} F_s(x, \mathbf{X}_s) \right] . \tag{2.10}$$

We can reorder the sum and product to obtain:

$$p(x) = \prod_{s \in n(x)} \left[\sum_{\mathbf{X}_s} F_s(x, \mathbf{X}_s) \right] . \tag{2.11}$$

It is important to have a good intuition for what has happened with this switch. Each term $F_s(x, \mathbf{X}_s)$ is the product of many factors; so it is a multivariate function of x and all of the other variables in that branch of the tree. In Equation 2.10, we first multiply all of the F_s terms together, to get a single joint function of all variables in the whole tree. In the sum, we then marginalise out over all other variables to be left with a marginal function only over our variable of interest x.

In Equation 2.11, on the other hand, we perform marginalisation first, taking each product of factors in a branch $F_s(x, \mathbf{X}_s)$ and summing over all other variables to obtain a function only of x in the square bracket for each branch. We then just calculate the product of these branch functions of x to obtain the final marginal distribution over x.

We can start to see now the idea of using message passing terminology to describe this process. Continuing to use Bishop's notation, we define:

$$\mu_{f_s \to x}(x) = \sum_{\mathbf{X}_s} F_s(x, \mathbf{X}_s) . \tag{2.12}$$

This term $\mu_{f_s \to x}(x)$ can be considered as a *message* from factor f_s to variable x. The message has the form of a probability distribution over variable x only, and is the marginalised probability over x as the result of considering all factors in one branch of the tree: in other words, it is *what that branch of the tree says about the marginal probability distribution of* x. If variable x receives such a message from all of the branches it is connected to, it can pool this information, and calculate its final marginal distribution by simply multiplying these messages together:

$$p(x) = \prod_{s \in n(x)} \mu_{f_s \to x}(x) . \tag{2.13}$$

Next, we go further into one of the branches of the tree, and break down the products of factors $F_s(x, \mathbf{X}_s)$ as follows:

$$\begin{aligned} F_s(x, \mathbf{X}_s) &= f_s(x, x_1, \ldots, x_M) \times G_1(x_1, \mathbf{X}_{S_1}) \ \ldots \ G_M(x_M, \mathbf{X}_{S_M}) \\ &= f_s(x, x_1, \ldots, x_M) \prod_{m \in n(f_s)} G_m(x_m, \mathbf{X}_{S_m}) \end{aligned} \tag{2.14}$$

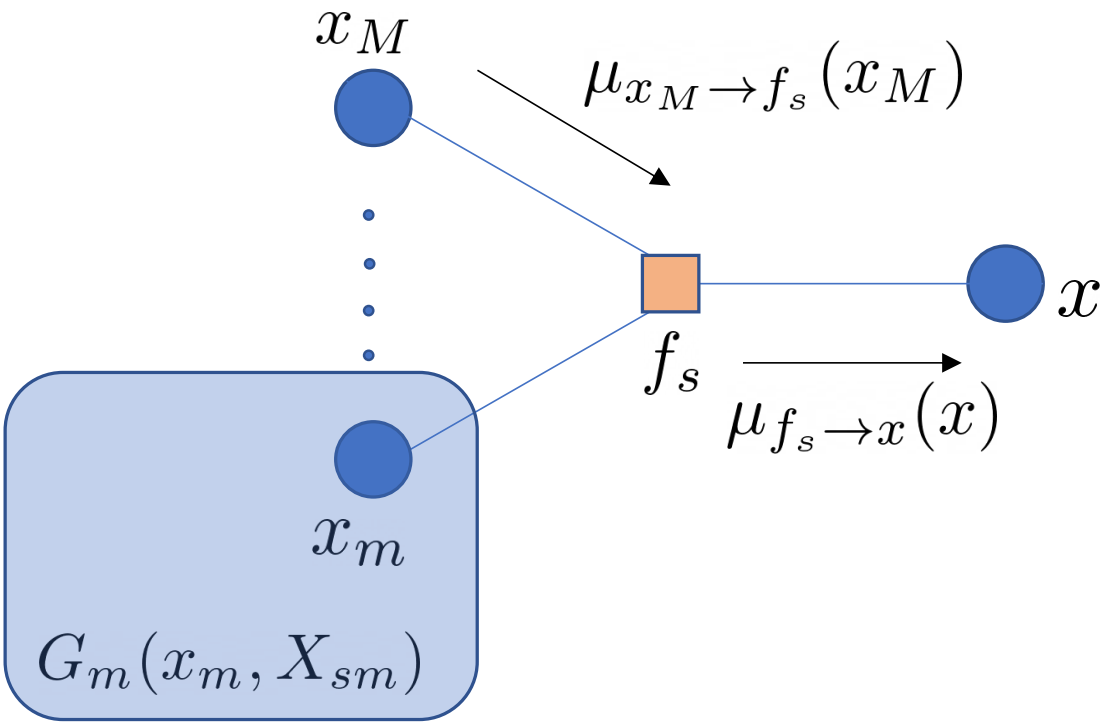


Figure 2.3: Factor f_s connects variable x to M other neighbouring variables $x_m \in \{x_1 \dots x_M\}$, each of which is the root of a sub-branch containing a product of factors G_m.

Here, referring to Figure 2.3, f_s, the factor which connects x to this branch, is a function of x as well as M other neighbouring variables $x_m \in x_1 \dots x_M$. Each of these variables connects to a sub-branch containing a product of factors G_m which is a function of variable x_m and other variables $\mathbf{X}_{S_m}$. Substituting into Equation 2.12:

$$\begin{aligned}
\mu_{f_s \to x}(x) &= \sum_{\mathbf{X}_s} \left[f_s(x, x_1, \dots, x_M) \prod_{m \in n(f_s)} G_m(x_m, \mathbf{X}_{S_m}) \right] \\
&= \sum_{x_1, \dots, x_M} \sum_{\mathbf{X}_{S_1}, \dots, \mathbf{X}_{S_M}} \left[f_s(x, x_1, \dots, x_M) \prod_{m \in n(f_s)} G_m(x_m, \mathbf{X}_{S_m}) \right] \\
&= \sum_{x_1, \dots, x_M} f_s(x, x_1, \dots, x_M) \prod_{m \in n(f_s)} \sum_{\mathbf{X}_{S_1}, \dots, \mathbf{X}_{S_M}} G_m(x_m, \mathbf{X}_{S_m}) \,. \qquad (2.15)
\end{aligned}$$

Note we have made use of the fact that $\mathbf{X}_s = (x_1, \dots, x_M, \mathbf{X}_{S_1}, \dots, \mathbf{X}_{S_M})$ to separate out the sum.

We can now define the second type of message, this time from variable to factor:

$$\mu_{x_m \to f_s}(x_m) = \sum_{\mathbf{X}_{sm}} G_m(x_m, \mathbf{X}_{S_m}) \,, \qquad (2.16)$$

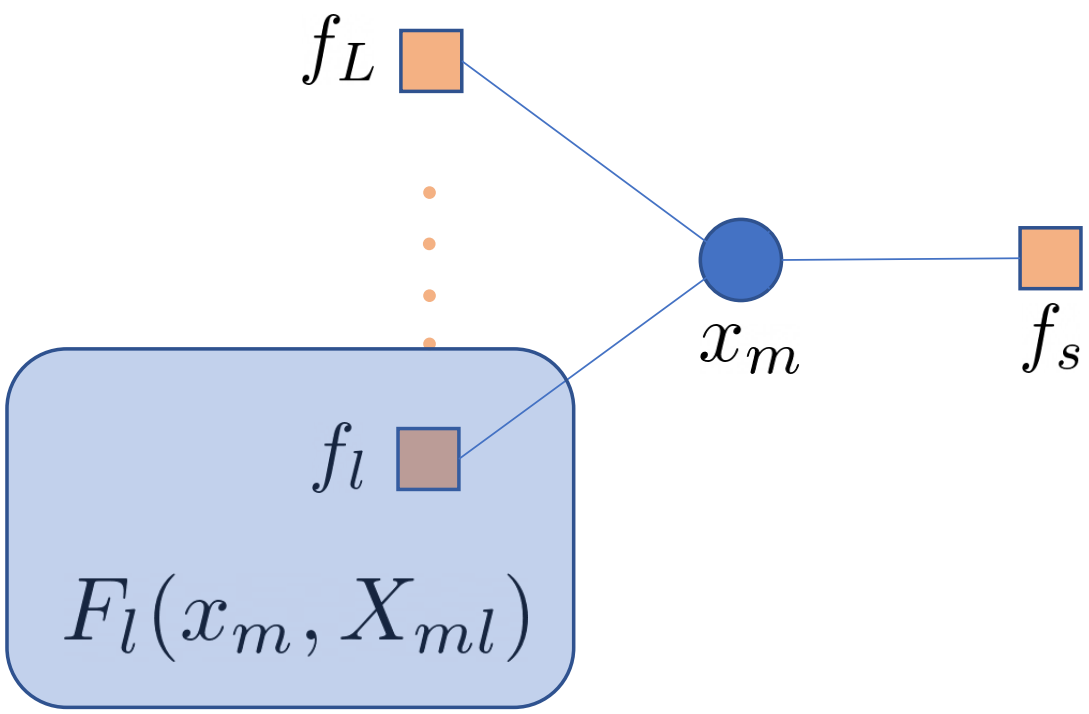


Figure 2.4: x_m, one of the variable neighbours of f_s, connects f_s to the product of factors $G_m(x_m, \mathbf{X}_{S_m})$ which we break down as $\prod_{l \in n(x_m) \setminus f_s} F_l(x_m, \mathbf{X}_{m_l})$.

and substitute into Equation 2.15 to get:

$$\mu_{f_s \to x}(x) = \sum_{x_1, \ldots, x_M} f_s(x, x_1, \ldots, x_M) \prod_{m \in n(f_s)} \mu_{x_m \to f_s}(x_m) \,. \tag{2.17}$$

We see here one half of the full recursive solution we are looking for: an expression for messages from factors to variables in terms of the messages those factors have received from other variables. We need just a few more steps to find ther other half of this. We need to take one more step deeper into the tree. Consider Figure 2.4, which now centres on x_m, one of the variable neighbours of f_s, which connects f_s to the product of factors $G_m(x_m, \mathbf{X}_{S_m})$. We break down this product as follows:

$$G_m(x_m, \mathbf{X}_{S_m}) = \prod_{l \in n(x_m) \setminus f_s} F_l(x_m, \mathbf{X}_{m_l}) \,. \tag{2.18}$$

We see that the total product factorises into terms $F_l(x_m, \mathbf{X}_{m_l})$, each of which is the product of the set of factors from the whole graph which connects to x_m via factor f_l. (We have broken down $\mathbf{X}_{S_m}$, the set of all variables connected to f_s via x_m, into subsets $\mathbf{X}_{m_l}$ which connect to x_m via factor f_l.)

We substitute this factorisation into Equation 2.16:

$$\mu_{x_m \to f_s}(x_m) = \sum_{\mathbf{X}_{s_m}} \prod_{l \in n(x_m) \setminus f_s} F_l(x_m, \mathbf{X}_{m_l}) \,, \tag{2.19}$$

and as we have seen before swap the order of the sum and product to obtain:

$$\mu_{x_m \to f_s}(x_m) = \prod_{l \in n(x_m) \setminus f_s} \sum_{\mathbf{X}_{m_l}} F_l(x_m, \mathbf{X}_{m_l}) \,. \tag{2.20}$$

Here we recognise the form of a message from factor to variable as defined in Equation 2.12, and substitute to obtain:

$$\mu_{x_m \to f_s}(x_m) = \prod_{l \in n(x_m) \setminus f_s} \mu_{f_l \to x_m}(x_m) \,. \tag{2.21}$$

We now have all we need for the full sum-product algorithm, and can focus on Equations 2.13, 2.17 and 2.21. Equation 2.13 says that in order to calculate the marginal distribution for x, we should multiply together all of messages received from each of its neighbouring factor nodes. Each of those messages has the form of a probability distribution over x only.

Stepping out to any one of the neighbouring factor nodes, we see the work that needs to be done at a factor node in Equation 2.17. A factor node receives messages from a number of variables, and must calculate a new message to send out. The messages that factor has received from other variables, each of which is a function of that one other variable, are all multiplied. We then multiply this product by the probability distribution representing the factor itself. We then marginalise out all variables other than the one to which the message will be sent, to leave a function of that variable only and that is the message that is sent.

One more step out, Equation 2.21 shows what happens at a variable node. It receives messages from a number of factors, all of which are functions of the variable, and multiplies these together to generate the message it sends on to the next factor.

It should now be clear that these two steps are simply repeated recursively through the whole tree. In order to find the marginal distribution for x, we start from all of the leaf nodes of the factor graph relative to x, which can be either variables or nodes, and pass messages inwards towards x. When each node has received messages from all outer nodes, it can perform its calculation to generate the correct message to pass inwards. This continues recursively all the way to x at the root of the tree.

One remaining detail is how to initialise the leaf nodes, and this is simply dealt with. A variable leaf node sends a message $\mu_{x \to f}(x) = 1$ to its only connected factor, and a factor leaf node sends

$\mu_{f\to x}(x) = f(x)$. These are seen to be correct from looking at Equations 2.17 and 2.21 if we imagine a set of null factors with flat probability distributions surrounding the main tree.

So we know how to find the marginal distribution at a chosen node x within a tree by defining that node as the root and passing messages recursively in towards it from all of the leaf nodes. If we required marginal distributions for *all* variable nodes in the tree, clearly we could simply repeat this procedure for each one. However, this would require a huge amount of wasted work. Imagine two variable nodes which are close together in a large tree. Defining either as the root node would lead to large equal branch and leaf structures in distant parts of the tree, and exactly the same computation in these regions would be repeated.

In fact, it is quite easy to see that we can find the marginal distribution for every variable node using only double the amount of work required to find the marginal for one variable. During the leaves-to-root message passing procedure to find the marginal for x, every variable and factor node along the way will have received incoming messages from all of its neighbours apart from the one to which it must transmit an outgoing message in the direction towards x. Once the messages get all the way the x, the root is then 'fully informed' and has a final marginal distribution which takes into account all of the information in the graph. Therefore, if we now send a second series of messages outwards from the root back to the leaves, we will fill in the missing incoming message for every variable node and can therefore calculate a fully informed marginal for each node.

In this way belief propagation is able to efficiently determine marginal distributions for every variable in a tree graph with a one time forward / backward sweep of message passing through the graph.

2.3.2 Belief Propagation on Loopy Graphs

As outlined in the derivation, BP was originally developed for graphs that are trees, in which any two nodes are connected by exactly one path. Most factor graphs for practical estimation problems are not trees however, but contain loops. This leads to two possibilities for the use of BP methods. One is to convert a general graph into a tree by combining nodes via graph cliques. These will be perfect trees, but with large compound nodes, and leads to the junction tree family of methods. The other is to retain the full loopy graph, but apply BP methods as if the graph was a tree, and keep iterating until convergence is reached. In this case, rather than exactly computing the marginal distributions after one sweep up and down through the graph using Equation 2.13: $p(x) = \prod_{s\in n(x)} \mu_{f_s\to x}(x)$, we maintain a belief estimate $b(x) = \prod_{s\in n(x)} \mu_{f_s\to x}(x)$ which is

updated and converges on the true marginal distribution after many iterations:

$$b(x) \to p(x) \ . \tag{2.22}$$

This approach is called *loopy Belief Propagation*, and has been shown to converge to useful solutions in many problems.

The simplest variant of loopy BP sends messages from all nodes at every iteration in a synchronous fashion (we discuss message schedules in detail in Section 3.2). Figure 2.5 illustrates how BP is applied to trees and graphs with loops respectively.

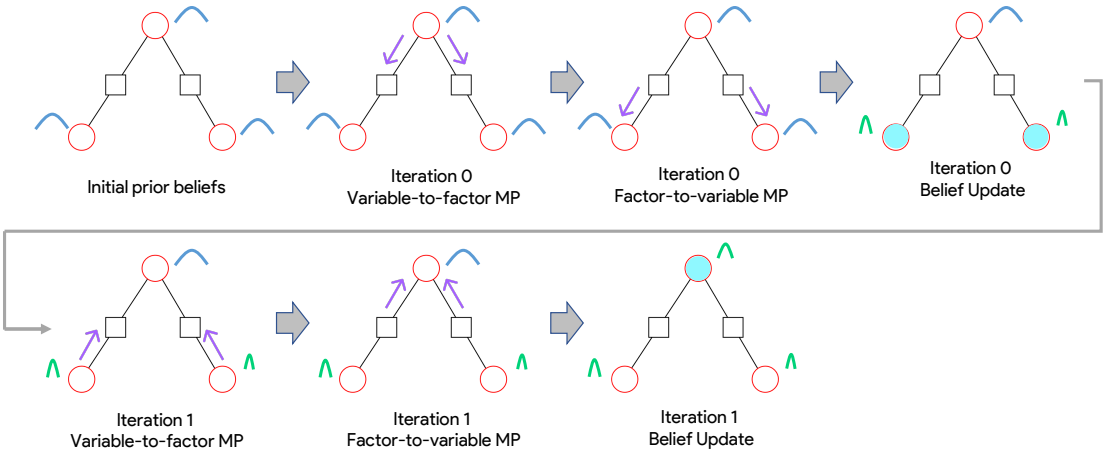
As BP was originally developed for trees, its application to loopy graphs was at first empirical [Murphy et al., 1999]. Theoretical grounds for applying the same update rules to loopy graphs were later developed [Yedidia et al., 2000, Weiss and Freeman, 2000, Wainwright and Jordan, 2008] that explain loopy BP as an approximate variational inference method in which inference is cast as an optimisation problem. Instead of directly minimising the factor energies (as is done in MAP inference), loopy BP minimises the KL divergence between the true posterior and a variational distribution (the beliefs) which we use as a proxy for the marginals after optimisation. Loopy BP can be derived via constrained minimisation of an approximation of the KL divergence known as the Bethe free energy [Yedidia et al., 2000]. We provide this variational derivation of belief propagation in Appendix 7.1.

As the Bethe free energy is non-convex, loopy BP is not guaranteed to converge and even when it does it may converge on the wrong marginals. An exception to this is Gaussian models for which, on convergence, the estimated marginal means are guaranteed to be exact. Empirically, however BP generally converges to the true marginals although for very loopy graphs it can fail [Murphy et al., 1999, Wainwright and Jordan, 2008].

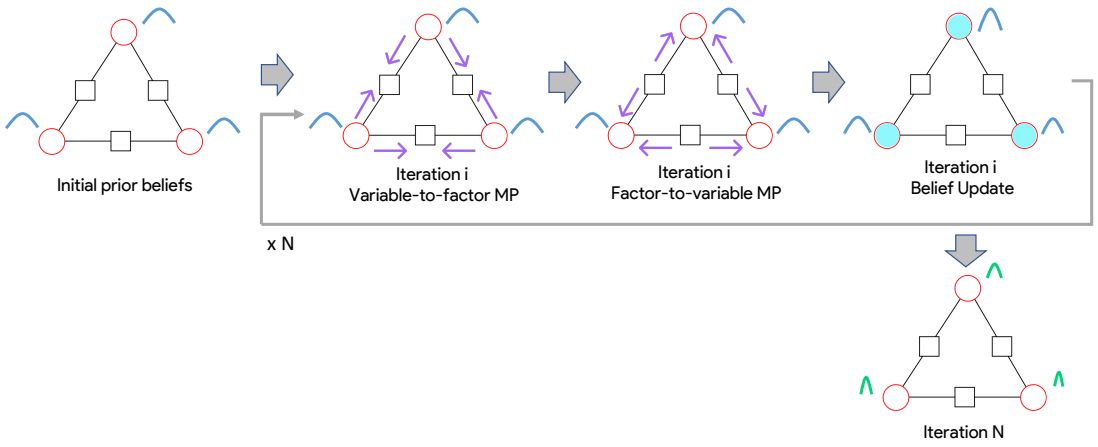
Most interesting problems have loopy structures and so for the remainder of this thesis we will use BP to refer to loopy BP. So far, we have outlined the BP equations, without specifying the form of the factor, message or belief distributions. After summarising the key properties of BP, we turn to Gaussian belief propagation which is a special form of continuous BP for Gaussian models in which all probability distributions are Gaussian.

2.3.3 Algorithm Summary and Intuitions

Here we summarise the core properties and equations of Belief Propagation. Belief propagation is an algorithm for marginal inference, i.e. it computes the marginal posterior distribution for each



(a) **Belief propagation on tree graphs**. The belief distributions are displayed with blue curves, which become green curves when the beliefs are equal to the true marginal distributions. The true marginals are computed after one full sweep from the root node (arbitrarily chosen at the top) down to the leaf nodes and back up. This takes 2 iterations of BP. We could equally have chosen the root node as the left node and sweep messages left to right and then back right to left.



(b) **Belief propagation on loopy graphs**. The belief distributions are displayed with blue curves, which become green curves when the beliefs are equal to the true marginal means. Here we show belief propagation on a loopy graph using a synchronous message passing schedule in which all variables and all factors send messages to all adjacent nodes on every iteration. After running N iterations of synchronous loopy BP, the beliefs converge on the true marginal distribution. The number of iterations (N) required for convergence depends on many properties including the graph topology, the factors and initialisation.

Figure 2.5: Belief Propagation on Trees and Loopy Graphs. See the full video version of this figure at: https://gaussianbp.github.io/#mp_videos

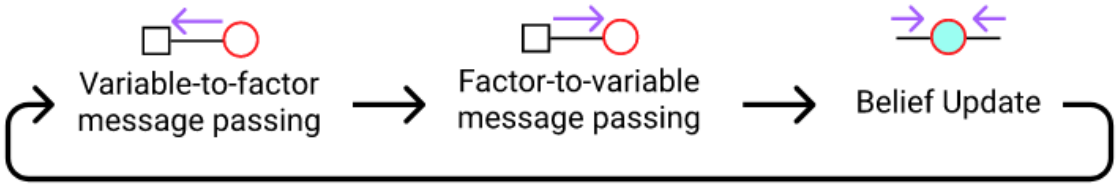


Figure 2.6: The 3 phases of Belief Propagation with their graphical interpretation.

variable from the set of factors that make up the joint posterior. BP is intimately linked to factor graphs by the following property: *belief propagation can be implemented as iterative message passing on the posterior factor graph.*

The algorithm operates by iteratively updating a node's locally stored belief by sending and receiving messages from neighbouring nodes in the factor graph. As illustrated in Figure 2.6, each iteration of BP consists of 3 phases: variable-to-factor message passing, factor-to-variable message passing, and belief update. In Box 2.1, we summarise the BP equations for reference and provide intuition for each phase of BP.

Key to understanding why belief propagation is efficient is considering the least efficient way to compute the marginal distribution for a variable. The naive way would be to take a product of all of the factors to give the joint distribution and then marginalise over all other variables. This simultaneous marginalisation over all other variables is expensive. For example, in the discrete case, if each variable takes k discrete values then marginalising over all but one variable requires summing k^{N-1} terms, where N is the number of variables. Belief propagation instead marginalises over minimal independent subsets of variables using the conditional dependency information which is encoded in the graph topology. If we consider the variable in question as the root node then BP is efficient as it moves the marginalisation down the graph towards the leaf nodes by marginalising over minimal subsets of conditionally dependent variables. To compute the marginal distributions for a tree graph with discrete variables containing only pairwise factors, belief propagation requires summing only $2N_f k^2$ terms, where N_f is the number of factors.

For a further intuitive exploration of BP that relates the equations to message passing on the factor graph, see our interactive diagram at https://gaussianbp.github.io/#bp_equations. A screenshot of this interactive diagram is shown in Figure 2.7.

Box 2.1: Belief Propagation Equations and Intuition

Belief Update

The variable node beliefs are computed by taking a product of the incoming messages from all adjacent factors, each of which represents that factor's belief on the receiving node's variables.

Equation 2.13

$$p(x) = \prod_{s \in n(x)} \mu_{f_s \to x}(x)$$

Factor-to-variable Message Passing

To send a message to an adjacent variable node, a factor aggregates messages from all other adjacent variable nodes and marginalizes over all the other nodes' variables to produce a message that expresses the factor's belief over the receiving node's variables.

Equation 2.17

$$\mu_{f_s \to x}(x) = \sum_{x_1, \ldots, x_M} f_s(x, x_1, \ldots, x_M) \prod_{m \in n(f_s)} \mu_{x_m \to f_s}(x_m)$$

Variable-to-Factor Message Passing

A variable-to-factor message tells the factor what the belief of the variable would be if the receiving factor node did not exist. This is computed by taking the product of the messages the variable node has received from all other adjacent factor nodes.

Equation 2.21

$$\mu_{x_m \to f_s}(x_m) = \prod_{l \in n(x_m) \setminus f_s} \mu_{f_l \to x_m}(x_m)$$

2.4 Gaussian Models

We now focus on Gaussian models in which all factors and therefore the joint posterior are univariate / multivariate Gaussian distributions. Gaussians are a convenient choice for a number of reasons:

1. They accurately represent the distribution for many real world events [Jaynes, 2003].
2. They have a simple analytic form.
3. Complex operations can be expressed with simple formulae.

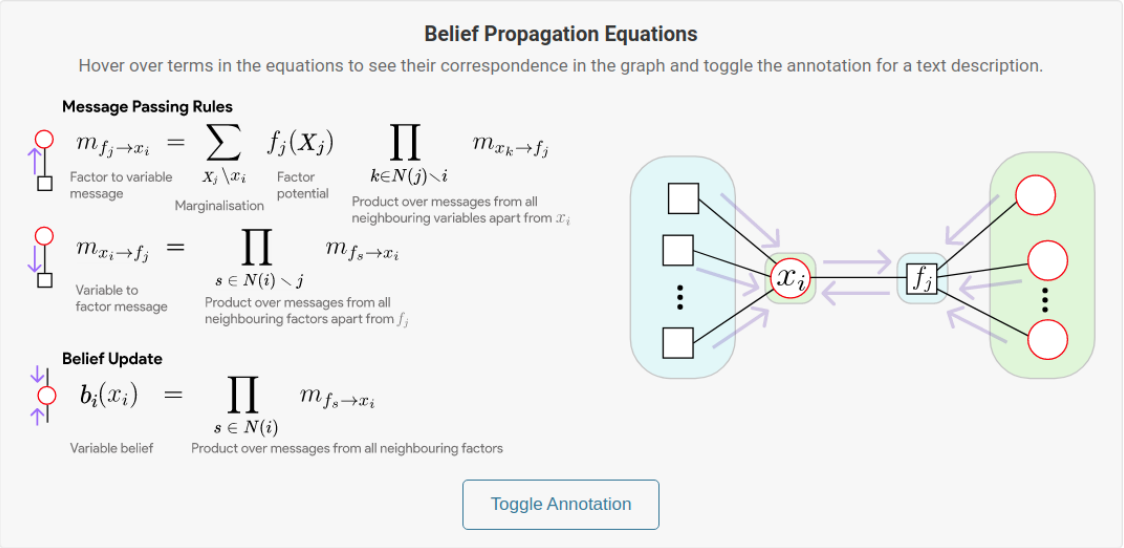


Figure 2.7: Annotated Belief Propagation equations and a visual exploration of the relation to message passing on the factor graph. Note in this screenshot messages are denoted with the character m rather than μ and the belief is denoted as b. See the full interactive version of this figure at: https://gaussianbp.github.io/#bp_equations.

4. They are closed under marginalisation, conditioning and taking products (up to normalisation).

A Gaussian factor or in general any multivariate Gaussian distribution can be written in the exponential form $p(\mathbf{x}) \propto e^{-E(\mathbf{x})}$ with a quadratic energy function. There are two ways to write the quadratic energy which correspond to the two common parameterisations of multivariate Gaussian distributions: the **moments form** [1] and the **canonical / information form**. A multivariate Gaussian distribution over a vector variable $\mathbf{x}$ is expressed in the moments form as:

$$p(\mathbf{x}) = \mathcal{N}(\mathbf{x};\, \boldsymbol{\mu}, \Sigma) = K_1 e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top \Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})}\,, \tag{2.23}$$

where $\boldsymbol{\mu}_m$ is the mean of the distribution and Σ is its covariance. The equivalent canonical or information form is:

$$p(\mathbf{x}) = \mathcal{N}^{-1}(\mathbf{x};\, \boldsymbol{\eta}, \Lambda) = K_2 e^{-\frac{1}{2}\mathbf{x}^\top \Lambda \mathbf{x} + \boldsymbol{\eta}^\top \mathbf{x}}\,, \tag{2.24}$$

The information vector $\boldsymbol{\eta}$ is related to the mean vector $\boldsymbol{\mu}$ by the relation:

$$\boldsymbol{\eta} = \Lambda\, \boldsymbol{\mu}\,, \tag{2.25}$$

and the precision matrix is the inverse of the covariance matrix:

$$\Lambda = \Sigma^{-1}\,. \tag{2.26}$$

[1] It's called the moments form as it is parameterised by the first moment and the second central moment of the distribution.

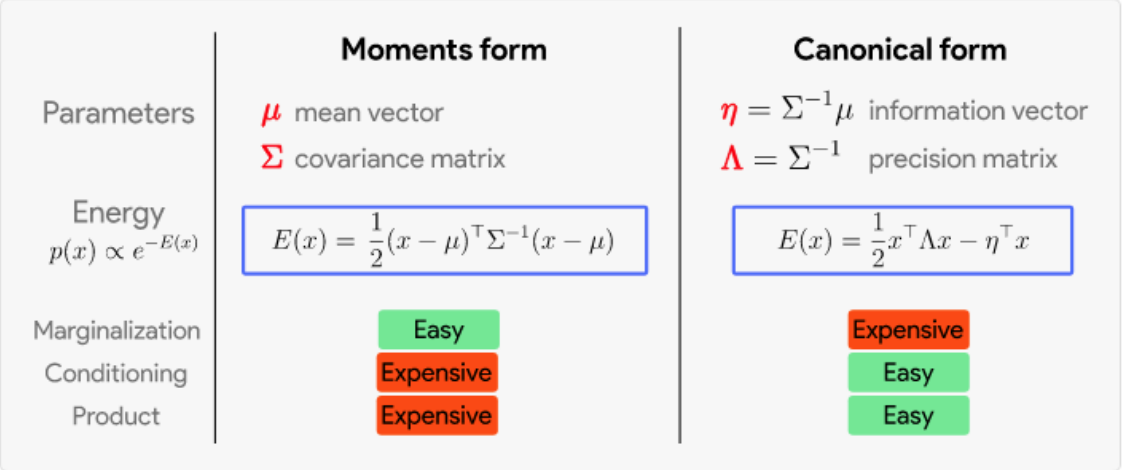


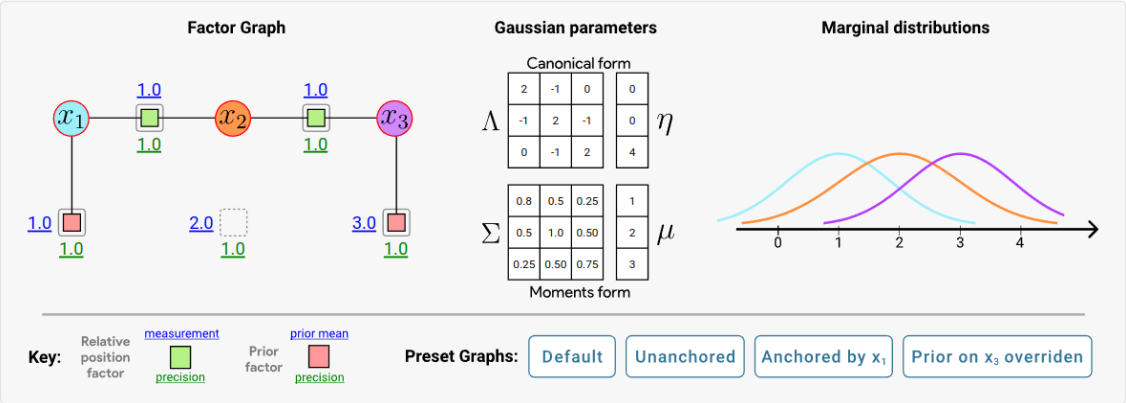
Figure 2.8: Comparison between the two parameterisations of the Gaussian Distribution: the moments form and the canonical / information form.

The two forms have different normalising constants $K_1 \neq K_2$, although we will not need to calculate the value of either in GBP. For an extended exploration of the information form for SLAM readers see Eustice *et al.* [Eustice et al., 2005].

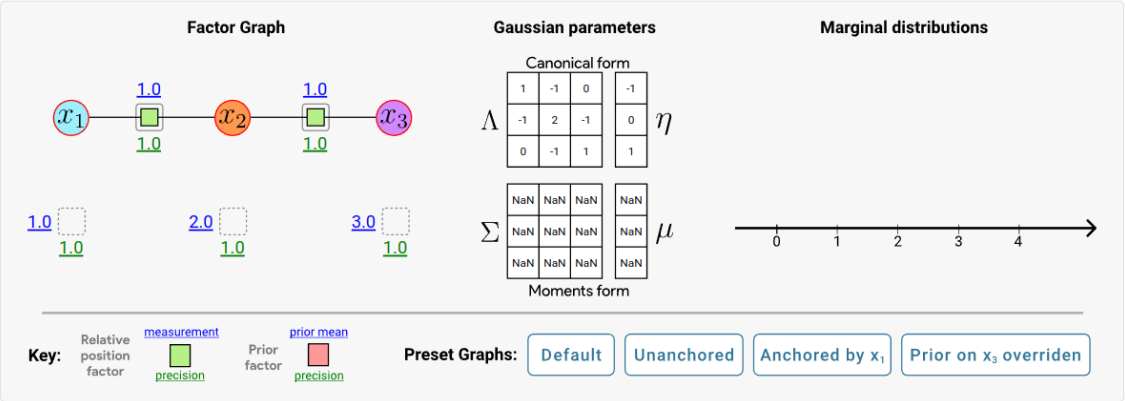
The key properties of each of these parameterisations are summarised in Figure 2.8. The canonical / information form is often preferred when performing inference, for two main reasons. Firstly, taking a product is simple in the canonical form, so it is easy to form the joint posterior from the factors. Secondly, the precision matrix is sparse and relates closely to the structure of the factor graph.

The precision matrix describes direct associations or conditional dependence between variables. If entry (i, j) of the precision matrix is zero then equivalently, there is no factor that directly connects x_i and x_j in the graph. This can be seen in Figure 2.9a where $\Lambda_{13} = \Lambda_{31} = 0$ because x_1 and x_3 have no factor directly connecting them.

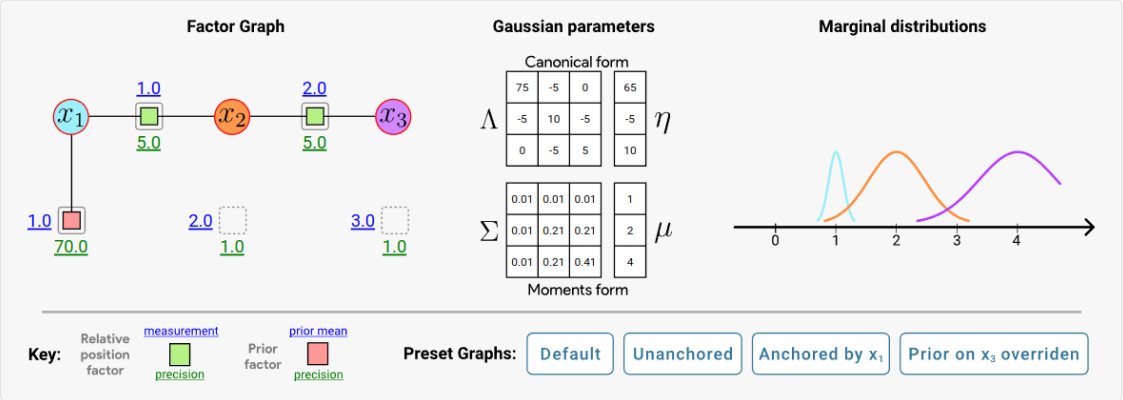
On the other hand, the covariance matrix describes induced correlations between variables and is dense as long as the graph is one single connected component. Unlike the canonical form, the moments form is unable to represent unconstrained distributions, as can be seen in Figure 2.9b. The unanchored graph contains only relative positional information which can be represented with zero information in the canonical form but corresponds to an infinite covariance in the covariance form along dimensions which are fully unconstrained. We encourage the reader to explore the full interactive diagram of Figure 2.9 at https://gaussianbp.github.io/#gaussian_gm and edit the graph to build intuition for Gaussian models and the relationship with the canonical form.



(a) Default



(b) Unanchored



(c) Anchored by x_1

Figure 2.9: Comparison between the parameters of the Canonical and Moments form of the Gaussian Distribution for 3 different graph topologies. We strongly encourage the reader to view the full interactive version of this figure at: https://gaussianbp.github.io/#gaussian_gm.

2.4.1 From Gaussian Inference to Linear Algebra

The joint distribution corresponding to a factor graph in which all factors are Gaussian can be represented as a single multivariate Gaussian distribution (since the energy terms are additive) in the canonical form:

$$P(\mathbf{x}) \propto e^{-\frac{1}{2}\mathbf{x}^\top \Lambda \mathbf{x} + \boldsymbol{\eta}^\top \mathbf{x}} . \tag{2.27}$$

MAP inference corresponds to computing the parameters $\mathbf{x}_{\text{MAP}}$ that maximise the above joint distribution. We can compute the gradient of the log-probability (the total energy):

$$\nabla_{\mathbf{x}} E = \nabla_{\mathbf{x}} \log P(\mathbf{x}) = -\Lambda \mathbf{x} + \boldsymbol{\eta} , \tag{2.28}$$

and solve for $\nabla_{\mathbf{x}} E = 0$. From here, we see that MAP inference in a Gaussian system reduces simply to solving:

$$\mathbf{x}_{\text{MAP}} = \Lambda^{-1} \boldsymbol{\eta} = \boldsymbol{\mu} , \tag{2.29}$$

which as expected is equal to the mean.

Marginal inference computes the per-variable marginal posterior distributions. In the moments form, the marginal distribution of x_i is:

$$p(x_i) = \int p(\mathbf{x}) d\mathbf{x}_{-i} \propto e^{-\frac{1}{2}(x_i - \mu_i)^\top \Sigma_{ii}^{-1} (x_i - \mu_i)} , \tag{2.30}$$

where the mean parameter μ_i is the i^{th} element of the joint mean vector and the covariance Σ_{ii} is entry (i, i) of the joint covariance matrix. The vector of marginal means for all variables is therefore the joint mean vector $\boldsymbol{\mu} = \Lambda^{-1}\boldsymbol{\eta} = \mathbf{x}_{\text{MAP}}$ and the marginal variances are the diagonal entries of the joint covariance matrix $\Sigma = \Lambda^{-1}$.

We can therefore summarise inference in Gaussian models as solving the linear system of equations:

$$A\mathbf{x} = \mathbf{b} \quad \Leftrightarrow \quad \Lambda \boldsymbol{\mu} = \boldsymbol{\eta} . \tag{2.31}$$

MAP inference solves for $\boldsymbol{\mu}$ while **marginal inference** solves for both $\boldsymbol{\mu}$ and the block diagonal elements of Λ^{-1}.

2.5 Gaussian Belief Propagation

Having introduced Gaussian models as models in which all factors and therefore the joint distribution are Gaussian distributions, we now discuss Gaussian belief propagation a form of continuous

BP applied to Gaussian models. Due to the closure properties of Gaussians, the beliefs and messages are also Gaussians and GBP operates by storing and passing around information vectors and precision matrices. The GBP equations are a specific case of the BP equations presented earlier.

Unlike general loopy BP, GBP is guaranteed to compute the exact marginal means on convergence, although the same is unfortunately not true for the variances which often converge to the true marginal variances, but are sometimes overconfident for loopy graphs [Weiss and Freeman, 2000]. We provide a proof of the exactness of the marginal means of GBP on convergence in Appendix 7.2. Although GBP does not in general have convergence guarantees, there some convergence conditions [Bickson, 2008, Du et al., 2018, Su and Wu, 2015] as well as methods to improve chances of convergence. We discuss improving convergence of GBP in more detail in Section 3.4.

In the case where the relationship between factors and variables is linear, and all factors have a Gaussian probability distribution, it is well understood that inference leads to a multivariate Gaussian distribution over the variables. It is also well established that factor graphs which have factors with non-linear dependence on the variables can be solved using efficient second order iterative optimisation (this is the class of non-linear least squares methods).

Here we will show how belief propagation is implemented in the Gaussian case which is the norm in robotics and computer vision.

2.5.1 Factor Definition

We will start with a general specification of a factor which will be familiar to anyone used to probabilistic estimation in computer vision and robotics. Suppose that a robot has a sensor which is configured to observe a quantity which is a function of the state variables of the robot. Note that we now switch from scalars to vectors here for the variable state space. The observed measurement or data $\mathbf{z}$ is generally modelled as $\mathbf{z} \sim \mathbf{h}(\mathbf{x}) + \boldsymbol{\epsilon}$, where $\mathbf{h}(\mathbf{x})$ simulates the data generation process from the state variables and $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \Sigma_n)$ is centred Gaussian noise. Rearranging, we see that the residual vector is Gaussian distributed:

$$\mathbf{r} = \mathbf{z} - \mathbf{h}(\mathbf{x}) \sim \mathcal{N}(0, \Sigma_n) \,, \tag{2.32}$$

allowing us to form the factor:

$$f(\mathbf{x};\mathbf{z}) = Ke^{-\frac{1}{2}\mathbf{r}^\top\Sigma_n^{-1}\mathbf{r}} \tag{2.33}$$

$$= Ke^{-\frac{1}{2}(\mathbf{z}-\mathbf{h}(\mathbf{x}))^\top\Sigma_n^{-1}(\mathbf{z}-\mathbf{h}(\mathbf{x}))} \tag{2.34}$$

$$= Ke^{-E(\mathbf{x};\mathbf{z})}\,, \tag{2.35}$$

where we have defined the general factor energy:

$$E(\mathbf{x};\mathbf{z}) = \frac{1}{2}(\mathbf{z}-\mathbf{h}(\mathbf{x}))^\top\Sigma_n^{-1}(\mathbf{z}-\mathbf{h}(\mathbf{x}))\,. \tag{2.36}$$

This expression represents the probability of obtaining vector measurement $\mathbf{z}$ from the sensor as a function of the set of involved variables $\mathbf{x}$. The form of the function is a squared exponential with a scaling factor K for normalisation whose value we will not need to calculate. Within the exponential, we see an inner product. This involves $\mathbf{h}$, the function which describes the dependence of the measurement on the variables, and $\mathbf{z}$, the value actually measured. Matrix Σ_n is the covariance or inverse precision of the measurement. Note that we can also use factors of this form for priors which are not sensor measurements but assumptions or external knowledge, such as smoothness priors.

In summary, to specify a Gaussian factor, we need:

- $\mathbf{h}(\mathbf{x})$, the functional form of the dependence of the measurement on the local state variables.
- $\mathbf{z}$, the actual observed value of the measurement.
- Σ_n, the symmetric covariance matrix that defines the noise model of the measurement.

2.5.2 Linearising Factors

In this section we discuss how to compute the Gaussian information form parameters for linear and non-linear factors. This process of forming the precision matrix and information vector is commonly used throughout the non-linear least squares literature [Dennis Jr, 1977, Lourakis, 2010]. For example, this linearisation procedure is done jointly over all factors in each step of the Gauss-Newton algorithm. The presentation in this section is therefore standard in the nonlinear least squares literature but to the best of our knowledge it is the first application of these linearisation tools to handle non-linear factors with GBP.

The general formula in Equation 2.34 for a factor represents a Gaussian distribution over the observed measurement $\mathbf{z}$. For linear measurement functions $\mathbf{h}(\mathbf{x})$, the resulting linear factor is

also a Gaussian distribution in the variables $\mathbf{x}$. To derive the Gaussian factor over $\mathbf{x}$, we begin with our general factor energy (Equation 2.36):

$$E(\mathbf{x};\mathbf{z}) = \frac{1}{2}(\mathbf{z}-\mathbf{h}(\mathbf{x}))^\top \Sigma_n^{-1}(\mathbf{z}-\mathbf{h}(\mathbf{x}))\ ,$$

and our goal is to manipulate the energy into the form of a Gaussian distribution over $\mathbf{x}$ in the information form:

$$E(\mathbf{x};\mathbf{z}) = \frac{1}{2}\mathbf{x}^\top \Lambda \mathbf{x} - \boldsymbol{\eta}^\top \mathbf{x}\ . \tag{2.37}$$

After transforming the energy into this form, we can identify the parameters $\boldsymbol{\eta}$ and Λ of the factor distribution.

To begin with, any linear measurement function can be generally written as:

$$\mathbf{h}(\mathbf{x}) = \mathtt{A}\mathbf{x} + \mathbf{b}\ , \tag{2.38}$$

where $\mathbf{b} \in \mathbb{R}^m$ is a constant vector and $\mathtt{A} \in \mathbb{R}^{m\times n}$ is a constant matrix for $\mathbf{z} \in \mathbb{R}^m$ and $\mathbf{x} \in \mathbb{R}^n$.

Substituting Equation 2.38 into the general factor energy (Equation 2.36) and rearranging:

$$\begin{aligned} E(\mathbf{x};\mathbf{z}) &= \frac{1}{2}\left[\mathbf{z}-\mathtt{A}\mathbf{x}-\mathbf{b}\right]^\top \Sigma_n^{-1}\left[\mathbf{z}-\mathtt{A}\mathbf{x}-\mathbf{b}\right] \\ &= \frac{1}{2}\left[(\mathbf{z}-\mathbf{b})-\mathtt{A}\mathbf{x}\right]^\top \Sigma_n^{-1}\left[(\mathbf{z}-\mathbf{b})-\mathtt{A}\mathbf{x}\right] \\ &= \frac{1}{2}\Big[(\mathbf{z}-\mathbf{b})^\top\Sigma_n^{-1}(\mathbf{z}-\mathbf{b}) + (\mathtt{A}\mathbf{x})^\top\Sigma_n^{-1}\mathtt{A}\mathbf{x} \\ &\qquad -(\mathbf{z}-\mathbf{b})^\top\Sigma_n^{-1}\mathtt{A}\mathbf{x} - (\mathtt{A}\mathbf{x})^\top\Sigma_n^{-1}(\mathbf{z}-\mathbf{b})\Big]\ . \end{aligned} \tag{2.39}$$

The first of the four terms here is a constant which doesn't depend on $\mathbf{x}$ and so we can drop it into the normalising constant. The third and fourth are equal (one is the transpose of the other, and both are scalars), so we can simplify to:

$$\begin{aligned} E(\mathbf{x};\mathbf{z}) &= \frac{1}{2}(\mathtt{A}\mathbf{x})^\top\Sigma_n^{-1}\mathtt{A}\mathbf{x} - (\mathbf{z}-\mathbf{b})^\top\Sigma_n^{-1}\mathtt{A}\mathbf{x} \\ &= \frac{1}{2}\mathbf{x}^\top(\mathtt{A}^\top\Sigma_n^{-1}\mathtt{A})\mathbf{x} - (\mathbf{z}-\mathbf{b})^\top\Sigma_n^{-1}\mathtt{A}\mathbf{x} \\ &= \frac{1}{2}\mathbf{x}^\top(\mathtt{A}^\top\Sigma_n^{-1}\mathtt{A})\mathbf{x} - \left(\mathtt{A}^\top\Sigma_n^{-1}(\mathbf{z}-\mathbf{b})\right)^\top\mathbf{x}\ . \end{aligned} \tag{2.40}$$

Matching this with the energy of the Gaussian information form (Equation 2.37): $E(\mathbf{x};\mathbf{z}) = \frac{1}{2}\mathbf{x}^\top\Lambda\mathbf{x} - \boldsymbol{\eta}^\top\mathbf{x}$, we can identify the Gaussian information form parameters of the linear factor as:

$$\boldsymbol{\eta} = \mathtt{A}^\top\Sigma_n^{-1}(\mathbf{z}-\mathbf{b}) \tag{2.41}$$

$$\Lambda = \mathtt{A}^\top\Sigma_n^{-1}\mathtt{A}\ . \tag{2.42}$$

As in all types of scalable Gaussian-based estimation, we need to be able to also handle non-linear measurement functions. Non-linear measurement functions result in non-Gaussian factor distributions over the variables $\mathbf{x}$ and so it is standard practise to use local linear versions of non-linear measurement functions. This is done via a first order Taylor expansion which approximates the function value at state $\mathbf{x}$ close to $\mathbf{x}_0$ as:

$$\mathbf{h}(\mathbf{x}) \approx \mathbf{h}(\mathbf{x}_0) + \mathtt{J}(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0) \,, \tag{2.43}$$

where $\mathtt{J}(\mathbf{x}_0) \in \mathbb{R}^{m \times n}$ is the Jacobian matrix evaluated at $\mathbf{x}_0$:

$$\mathtt{J}(\mathbf{x}_0) = \left.\frac{\partial \mathbf{h}}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}_0} . \tag{2.44}$$

Writing the Taylor expansion as:

$$\mathbf{h}(\mathbf{x}) \approx \mathtt{J}(\mathbf{x}_0)\mathbf{x} + (\mathbf{h}(\mathbf{x}_0) - \mathtt{J}(\mathbf{x}_0)\mathbf{x}_0) \,, \tag{2.45}$$

we can identify $\mathtt{A} = \mathtt{J}(\mathbf{x}_0)$ and $\mathbf{b} = \mathbf{h}(\mathbf{x}_0) - \mathtt{J}(\mathbf{x}_0)\mathbf{x}_0$ from the linear measurement function in Equation 2.38. Substituting our expressions for $\mathtt{A}$ and $\mathbf{b}$ into Equations 2.41 and 2.42 yields the information form parameters for a linearised general non-linear factor. Note that function $\mathbf{h}$ could be any non-linear function, for example a trained neural network [Czarnowski et al., 2020] or a Gaussian process [Mukadam et al., 2018]. It is worth noting that the factor linearisation procedure is similar to the moment matching procedure in EP when the variational factor distributions are modelled as Gaussians. We summarise the equations for factor linearisation in Box 2.2.

Box 2.2: Factor Linearisation

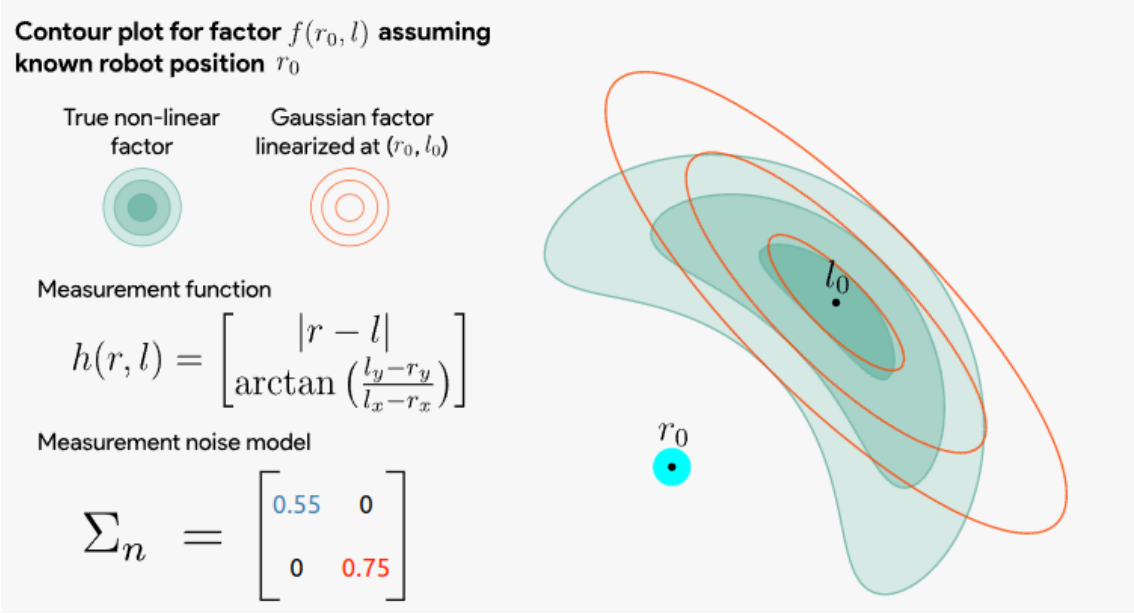
In summary, a non-linear factor f with measurement function $\mathbf{h}(\mathbf{x})$, measurement $\mathbf{z}$ and measurement covariance Σ_n can be linearised about $\mathbf{x}_0$ to produce a Gaussian factor $f(\mathbf{x}; \mathbf{z}, \mathbf{x}_0) = \mathcal{N}^{-1}(\mathbf{x}; \boldsymbol{\eta}, \Lambda)$ with information form parameters:

$$\boldsymbol{\eta} = \mathtt{J}(\mathbf{x}_0)^\top \Sigma_n^{-1} \left[\mathbf{z} - \mathbf{h}(\mathbf{x}_0) + \mathtt{J}(\mathbf{x}_0)\mathbf{x}_0\right] \tag{2.46}$$

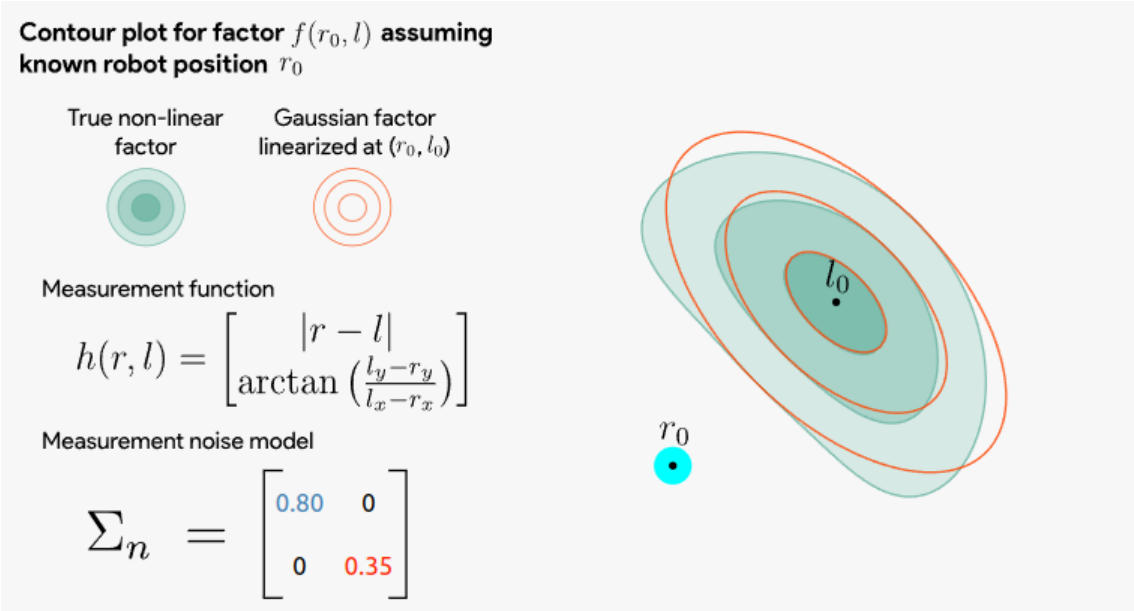
$$\Lambda = \mathtt{J}(\mathbf{x}_0)^\top \Sigma_n^{-1} \mathtt{J}(\mathbf{x}_0) \,, \tag{2.47}$$

where $\mathtt{J}(\mathbf{x}_0)$ is the Jacobian matrix evaluated at the linearisation point.

To visualise how factor linearisation works, consider a robot moving in a plane that measures the 2D distance and angle to a landmark also in the plane. The current estimates for the position of the robot and landmark are r_0 and l_0 respectively, and the observed measurement $\mathbf{z} = \mathbf{h}(\mathbf{r}_0, \mathbf{l}_0)$. In Figure 2.10, we show both the true non-linear factor and the Gaussian approximated factor conditioned on $\mathbf{r} = \mathbf{r}_0$, so that the distribution is only over the landmark position.



(a)



(b)

Figure 2.10: Linearisation of a range and bearing measurement factor. The top and bottom parts of this figure show the same factor with different covariance matrices Σ_n describing the noise associated with the measurement. In both cases, the linearised Gaussian factor is a good approximation of the true factor close to the linearisation point. In (a), the approximation degrades significantly further away from the linearisation point as the underlying factor cannot be well approximated everywhere by a Gaussian. See the full interactive version of this figure in this section of the interactive article: https://gaussianbp.github.io/#non-linear-relationships

The accuracy of the approximate Gaussian factor depends on the linearity of the measurement function $\mathbf{h}$ at the linearisation point. In this case, as the measurement function is reasonably is smooth, the linear approximation is good close to the linearisation point l_0, while further away, the approximation can degrade depending on the measurement covariance values. In practice, during optimisation we can avoid this region of poor approximation by relinearising frequently. As GBP is local, a just-in-time approach to linearisation [Ranganathan et al., 2007] can be used in which factors are relinearised individually when the current estimate of the adjacent variables strays significantly from the linearisation point.

2.5.3 Message Passing at a Variable Node

Let us now consider the computation which happens at nodes to implement message passing. Remember that in belief propagation, messages always have the form of a probability distribution in the state space of the variable node either sending or receiving the message. In GBP, each message will therefore take the form of an information vector and precision matrix in that state space.

Referring back to Box 2.1, we first consider the processing that happens at a variable node $\mathbf{x}_m$ during message passing. A variable node is connected to a number of factors, and during a typical message passing step it receives incoming messages from all of these except one, and must generate an outgoing message to send to the remaining factor. All of the messages involved are in the state space of node $\mathbf{x}_m$. Restating Equation 2.21 with $\mathbf{x}_m$ now a vector variable, we simply need to multiply together all of the incoming messages to generate the outgoing message:

$$\mu_{\mathbf{x}_m \to f_s}(\mathbf{x}_m) = \prod_{l \in n(\mathbf{x}_m) \setminus f_s} \mu_{f_l \to \mathbf{x}_m}(\mathbf{x}_m) \,. \tag{2.48}$$

Each incoming message is represented by an information vector and a precision matrix:

$$\mu_{f_l \to \mathbf{x}_m}(\mathbf{x}_m) = \mathcal{N}^{-1}(\mathbf{x}_m; \boldsymbol{\eta}_{f_l \to \mathbf{x}_m}, \Lambda_{f_l \to \mathbf{x}_m}) \,. \tag{2.49}$$

We obtain the information vector and precision matrix of the outgoing message $\mu_{\mathbf{x}_m \to f_s}(\mathbf{x}_m) = \mathcal{N}^{-1}(\mathbf{x}_m; \boldsymbol{\eta}_{\mathbf{x}_m \to f_s}, \Lambda_{\mathbf{x}_m \to f_s})$ by simply adding the incoming information form parameters:

$$\boldsymbol{\eta}_{\mathbf{x}_m \to f_s} = \sum_{l \in n(\mathbf{x}_m) \setminus f_s} \boldsymbol{\eta}_{f_l \to \mathbf{x}_m} \tag{2.50}$$

$$\Lambda_{\mathbf{x}_m \to f_s} = \sum_{l \in n(\mathbf{x}_m) \setminus f_s} \Lambda_{f_l \to \mathbf{x}_m} \,. \tag{2.51}$$

This is because when we multiply several Gaussian expressions, we simply add the exponents.

2.5.4 Message Passing at a Factor Node

A factor node receives incoming messages from a number of variable nodes, and must process these to produce an outgoing message to the target variable node. Looking back at Equation 2.17, factor to variable messages are computed as:

$$\mu_{f_s \to \mathbf{x}_m}(\mathbf{x}_m) = \sum_{\mathbf{X}_s \setminus \mathbf{x}_m} f_s(\mathbf{X}_s) \prod_{i \in n(f_s)} \mu_{\mathbf{x}_i \to f_s}(\mathbf{x}_i) \, . \tag{2.52}$$

First, the incoming messages are multiplied together, and this product is also multiplied by the factor distribution itself. Each of the incoming messages is a function of the state space of the variable node it comes from, while the factor potential is a function of all of the variables connected to the factor, including the output variable. The full product is therefore also a function of all variables. Finally, all variables other than the output variable are marginalised out from this joint distribution, and the result is a function only in the output variable's state space — this is the outgoing message.

In Section 2.5.2, we described how to compute the information form parameters for Gaussian factors with both linear and non-linear measurement functions. We begin our factor-to-variable message computation with the factor information form parameters $\boldsymbol{\eta}_s, \Lambda_s$, which parameterise the factor as: $f_s(X_s) = \mathcal{N}^{-1}(\mathbf{X}_s; \boldsymbol{\eta}_s, \Lambda_s)$

Starting with this information vector and precision matrix for the factor, we add the incoming messages from input variables. This vector and matrix addition is done 'in place'. In the vector of variables $\mathbf{X}_s$ associated with the factor, there should be a partitioning into contiguous sets which come from each connected variable node. E.g. let us consider a factor with three connected variable nodes, where $\mathbf{x}_3$ is the output node in this case. The set of set of variables connecting to the factor can be partitioned as:

$$\mathbf{X}_s = \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \mathbf{x}_3 \end{pmatrix} , \tag{2.53}$$

and the information vector and precision matrix are partitioned in the same way:

$$\boldsymbol{\eta}_s = \begin{pmatrix} \boldsymbol{\eta}_{s1} \\ \boldsymbol{\eta}_{s2} \\ \boldsymbol{\eta}_{s3} \end{pmatrix} \tag{2.54}$$

$$\Lambda_s = \begin{bmatrix} \Lambda_{s11} & \Lambda_{s12} & \Lambda_{s13} \\ \Lambda_{s21} & \Lambda_{s22} & \Lambda_{s23} \\ \Lambda_{s31} & \Lambda_{s32} & \Lambda_{s33} \end{bmatrix} . \tag{2.55}$$

Taking the product with the messages coming from input nodes $\mathbf{x}_1$ and $\mathbf{x}_2$ we get:

$$\boldsymbol{\eta}_{Cs} = \begin{pmatrix} \boldsymbol{\eta}_{s1} + \boldsymbol{\eta}_{\mathbf{x}_1 \to f_s} \\ \boldsymbol{\eta}_{s2} + \boldsymbol{\eta}_{\mathbf{x}_2 \to f_s} \\ \boldsymbol{\eta}_{s3} \end{pmatrix} \tag{2.56}$$

$$\boldsymbol{\Lambda}_{Cs} = \begin{bmatrix} \boldsymbol{\Lambda}_{s11} + \boldsymbol{\Lambda}_{\mathbf{x}_1 \to f_s} & \boldsymbol{\Lambda}_{s12} & \boldsymbol{\Lambda}_{s13} \\ \boldsymbol{\Lambda}_{s21} & \boldsymbol{\Lambda}_{s22} + \boldsymbol{\Lambda}_{\mathbf{x}_2 \to f_s} & \boldsymbol{\Lambda}_{s23} \\ \boldsymbol{\Lambda}_{s31} & \boldsymbol{\Lambda}_{s32} & \boldsymbol{\Lambda}_{s33} \end{bmatrix} . \tag{2.57}$$

To complete message passing, from this joint distribution we must marginalise out all variables apart from those of the output node, in this case $\mathbf{x}_3$. Eustice *et al.* [Eustice et al., 2005] give the formula for marginalising a general partitioned Gaussian state in the information form. If the joint distribution is:

$$\boldsymbol{\eta} = \begin{pmatrix} \boldsymbol{\eta}_\alpha \\ \boldsymbol{\eta}_\beta \end{pmatrix} \tag{2.58}$$

$$\boldsymbol{\Lambda} = \begin{bmatrix} \boldsymbol{\Lambda}_{\alpha\alpha} & \boldsymbol{\Lambda}_{\alpha\beta} \\ \boldsymbol{\Lambda}_{\beta\alpha} & \boldsymbol{\Lambda}_{\beta\beta} \end{bmatrix} , \tag{2.59}$$

then the marginal distribution over the variables α, achieved by integrating over the variables β, is:

$$\boldsymbol{\eta}_{M\alpha} = \boldsymbol{\eta}_\alpha - \boldsymbol{\Lambda}_{\alpha\beta}\boldsymbol{\Lambda}_{\beta\beta}^{-1}\boldsymbol{\eta}_\beta \tag{2.60}$$

$$\boldsymbol{\Lambda}_{M\alpha} = \boldsymbol{\Lambda}_{\alpha\alpha} - \boldsymbol{\Lambda}_{\alpha\beta}\boldsymbol{\Lambda}_{\beta\beta}^{-1}\boldsymbol{\Lambda}_{\beta\alpha} . \tag{2.61}$$

To apply these formulae to the partitioned state of Equations 2.56 and 2.57, we first reorder the vector and matrix to bring the output variable to the top. For our example where $\mathbf{x}_3$ is the output variable, we reorder the conditioned information vector and precision matrix to:

$$\boldsymbol{\eta}_{CRs} = \begin{pmatrix} \boldsymbol{\eta}_{s3} \\ \boldsymbol{\eta}_{s1} + \boldsymbol{\eta}_{\mathbf{x}_1 \to f_s} \\ \boldsymbol{\eta}_{s2} + \boldsymbol{\eta}_{\mathbf{x}_2 \to f_s} \end{pmatrix} \tag{2.62}$$

$$\boldsymbol{\Lambda}_{CRs} = \begin{bmatrix} \boldsymbol{\Lambda}_{s33} & \boldsymbol{\Lambda}_{s31} & \boldsymbol{\Lambda}_{s32} \\ \boldsymbol{\Lambda}_{s13} & \boldsymbol{\Lambda}_{s11} + \boldsymbol{\Lambda}_{\mathbf{x}_1 \to f_s} & \boldsymbol{\Lambda}_{s12} \\ \boldsymbol{\Lambda}_{s23} & \boldsymbol{\Lambda}_{s21} & \boldsymbol{\Lambda}_{s22} + \boldsymbol{\Lambda}_{\mathbf{x}_2 \to f_s} \end{bmatrix} . \tag{2.63}$$

We then identify sub-blocks $\alpha = \mathbf{x}_3$ and $\beta = \mathbf{x}_1\mathbf{x}_2$ between Equations 2.58, 2.59 and Equations 2.62, 2.63, and apply Equations 2.60 and 2.61 to obtain the marginal distribution over $\mathbf{x}_3$.

The marginalised information vector and precision matrix forms the outgoing message to variable node $\mathbf{x}_3$:

$$\mu_{f_s \to \mathbf{x}_3} = \mathcal{N}^{-1}(\mathbf{x}_3; \boldsymbol{\eta}_{Mx_3}, \Lambda_{Mx_3}) \, . \tag{2.64}$$

2.5.5 Belief Update

To compute the belief at a variable node, we take the product of incoming messages from all adjacent factors (Equation 2.13):

$$b_m(\mathbf{x}_m) = \prod_{s \in n(x_m)} \mu_{f_s \to \mathbf{x}_m} \, . \tag{2.65}$$

These messages are Gaussian distributions over $\mathbf{x}_m$ and the product of these messages is evaluated by summing the information vectors and precision matrices. The belief parameters $\boldsymbol{\eta}_m$ and Λ_m are therefore:

$$\boldsymbol{\eta}_m = \sum_{s \in n(x_m)} \boldsymbol{\eta}_{f_s \to \mathbf{x}_m} \tag{2.66}$$

$$\Lambda_m = \sum_{s \in n(x_m)} \Lambda_{f_s \to \mathbf{x}_m} \, . \tag{2.67}$$

2.6 GBP with Lie Groups

So far we have assumed that all state space variables can be represented by vectors that are part of a Euclidean vector space. In most realistic robotics applications however, this is not the case. Robot pose for example is represented by a rotation and translation, where careful thought about parameterisation is needed. Here we detail how Gaussian belief propagation can be extended to handle state space variables that belong to Lie Groups.

This section contains original contributions that are adapted from our Robot Web paper [Murai et al., 2022]. Although the machinery for handling inference on manifolds is well studied [Boumal, 2023, Absil et al., 2009, Mattamala, 2021], the application of these tools to GBP here is novel. We do not provide results of the application of GBP with Lie Groups to decentralised robot localisation in this thesis, and refer the interested reader to the Robot Web paper for experimental results.

2.6.1 Manifolds

Although this section discusses how to perform GBP with Lie Groups, we begin by discussing manifolds and introducing concepts that will be useful later, as Lie Groups are both differentiable manifolds and groups. We do not describe how to perform inference on manifolds in general in this thesis and refer the reader to textbooks by Boumal [Boumal, 2023] or Absil [Absil et al., 2009] on the topic.

Euclidean vectors, which we have assumed to be the objects of interest up to now, are manifolds as well as more complex objects such as rigid body transformations and rotation matrices. Differentiable manifolds may have a non-Euclidean structure globally, but locally they can be approximated as Euclidean spaces using local tangent spaces that have the same dimension as the number of degrees of freedom of the manifold. To work with manifolds, two operations are required:

- **retract**. This maps a vector $\boldsymbol{\tau}$ in the tangent space at $\mathbf{x}_0$ back to the manifold: $\mathbf{x} = \text{retract}_{\mathbf{x}_0}(\boldsymbol{\tau})$.
- **local**. This is the inverse operation that maps an element $\mathbf{x}$ into the tangent space at another element $\mathbf{x}_0$: $\boldsymbol{\tau} = \text{local}_{\mathbf{x}_0}(\mathbf{x})$.

The retract operation can be used to define uncertainty and in particular Gaussian distributions on manifolds. This is done by defining distributions in the tangent space and then mapping them back to the manifold using the retract operation. A Gaussian distribution around a point $\mathbf{x}$ is represented in the tangent space at that point as follows:

$$\mathcal{N}(\mathbf{x}, \Lambda^{-1}) = \text{retract}_{\mathbf{x}}(\xi) \,, \tag{2.68}$$

where:

$$\xi \sim \mathcal{N}(0, \Lambda^{-1}) \,. \tag{2.69}$$

2.6.2 Lie Groups

We now discuss Lie groups which are both manifolds and groups. We refer the reader to Solà *et al.*'s 'A micro Lie theory for state estimation in robotics' [Solà et al., 2018] for an excellent and detailed tutorial on Lie Groups in robotics, and we broadly follow their notation here. A group contains elements $\mathcal{X}$ that satisfy the 4 group axioms with respect to the group composition oper-

ation $\circ$. These axioms are closure under composition, existence of an identity element, existence of an inverse element and associativity.

As Lie Groups are manifolds, they have the retract and local operations defined. These operations are defined at the identity element I using the Exponential and Log map respectively: $\text{retract}_I(\boldsymbol{\tau}) = \text{Exp}(\boldsymbol{\tau})$ and $\text{local}_I(\mathcal{X}) = \text{Log}(\mathcal{X})$.

Remembering how Gaussian distributions are defined on manifolds, we can define a Gaussian distribution over, for example, 3D rotations which are the $\mathbf{SO}(3)$ Lie Group. A Gaussian distribution over a rotation is represented by the parameters $(\bar{\mathcal{X}}, \Lambda)$, where Λ is a precision matrix in the tangent space at the mean of the Gaussian distribution $\bar{\mathcal{X}}$.

With this at hand, we can now represent beliefs and factor distributions as Gaussians over Lie groups, by defining the precision in the tangent space at the mean. One important choice is how to represent messages. Messages can either be represented in the full Gaussian form above or as perturbations around some stored element. We believe it is important to use the above full Gaussian form for messages and consequently we make the choice that: *All messages to or from Lie Group variables take the form of a point estimate, represented by the full over-parameterised Group element, together with a precision matrix defined in the tangent space around that element.* We prefer this approach because it gives maximum flexibility and minimises the need for independent devices to have knowledge or memory of each other.

For example, in multi-device localisation which is the application in our Robot Web paper [Murai et al., 2022], one robot may make a range measurement of another robot it observes. A factor is then added to the graph for this measurement and the details of the factor are stored at the robot that made the observation. Choosing the messages to be sent as a mean point estimate and a precision matrix in the tangent plane at the mean, means that the receiving robot doesn't need to know anything about the factor or which tangent plane the message it receives belongs to. For this reason, this particular choice of message representation makes distributed message passing both simple and flexible. The Robot Web protocol for distributed message passing is related to this choice and is discussed further in Section 3.5.

Before proceeding with the derivation of GBP with Lie Groups, we define a shorthand for two frequently used operations. The circled-plus and circled-minus operators relate group elements $\mathcal{X}$ and vectors in its tangent space $\boldsymbol{\tau}$. Circled-plus is defined as:

$$\mathcal{X} \oplus \boldsymbol{\tau} = \mathcal{X} \circ \text{Exp}(\boldsymbol{\tau}) \,. \tag{2.70}$$

Circled-minus is defined as:

$$\mathcal{Y} \ominus \mathcal{X} = \mathrm{Log}(\mathcal{X}^{-1} \circ \mathcal{Y}) \in T_{\mathcal{X}}\mathcal{M} \,, \tag{2.71}$$

and produces an element in the tangent space at $\mathcal{X}$, denoted $T_{\mathcal{X}}\mathcal{M}$.

2.6.3 Linearising Factors

Reproducing Equation 2.34, the general definition of a Gaussian factor is:

$$f(\mathbf{x};\mathbf{z}) = Ke^{-\frac{1}{2}\left[(\mathbf{z}-\mathbf{h}(\mathbf{x}))^\top \Lambda (\mathbf{z}-\mathbf{h}(\mathbf{x}))\right]} \,,$$

where we define the factor energy (Equation 2.36):

$$f(\mathbf{x};\mathbf{z}) = Ke^{-E(\mathbf{x};\mathbf{z})} \quad \text{with} \quad E(\mathbf{x};\mathbf{z}) = \frac{1}{2}(\mathbf{z}-\mathbf{h}(\mathbf{x}))^\top \Sigma_n^{-1}(\mathbf{z}-\mathbf{h}(\mathbf{x})) \,.$$

The factor represents the probability of obtaining a vector measurement $\mathbf{z}$ from a sensor as a function of the state variables $\mathbf{x}$. To send messages from a factor node, $f(\mathbf{x};\mathbf{z})$ must be a Gaussian distribution over the state variables $\mathbf{x}$. If $\mathbf{h}$ is nonlinear and the variables belong to a Euclidean state space like $\mathbb{R}^2$ or $\mathbb{R}^3$, then we can use Equations 2.46 and 2.47 to linearise the factor to form a Gaussian in $\mathbf{x}$.

Here we consider the case where the variable belongs to a Lie Group. We now denote the state variable $\mathcal{X}$ to highlight it is a member of a Lie Group. As the variable is a Lie Group element, Gaussian messages to and from this variable take the form of a group element describing the mean, and a precision matrix in the tangent space at the mean. To perform message passing at a factor, we must first express the factor as a Gaussian in a chosen tangent space and then transform all messages into this same tangent space to carry out the calculation. We make the standard choice of using the tangent plane at the most recent estimate of the state, which we denote $\mathcal{X}_0$.

The first task is to compute the Gaussian factor in this tangent space at $\mathcal{X}_0$. When computing Gaussian factors for Euclidean variables (in Section 2.5.2), our goal was to find the Gaussian parameters $\boldsymbol{\eta}$ and Λ such that the energy can be expressed as:

$$E(\mathbf{x};\mathbf{z}) = \frac{1}{2}\mathbf{x}^\top \Lambda \mathbf{x} - \boldsymbol{\eta}^\top \mathbf{x} \,. \tag{2.72}$$

For Lie group variables, we instead want to find the parameters $\boldsymbol{\eta}$ and Λ of a Gaussian in the tangent plane such that the energy can be expressed as:

$$E(\mathcal{X};\mathbf{z}) = \frac{1}{2}\boldsymbol{\tau}^\top \Lambda \boldsymbol{\tau} - \boldsymbol{\eta}^\top \boldsymbol{\tau} \,, \tag{2.73}$$

where $\boldsymbol{\tau} = \mathcal{X} \ominus \mathcal{X}_0$. This same calculation applies for factors with both linear and non-linear measurement functions $\mathbf{h}$.

As before, we make use of the first-order Taylor expansion to linearise the measurement function around $\mathcal{X}_0$, however now using the $\ominus$ operator:

$$\mathbf{h}(\mathcal{X}) \approx \mathbf{h}(\mathcal{X}_0) + \mathtt{J}(\mathcal{X} \ominus \mathcal{X}_0) \tag{2.74}$$

$$= \mathbf{h}(\mathcal{X}_0) + \mathtt{J}\boldsymbol{\tau} \,. \tag{2.75}$$

Here $\mathtt{J}$ is the Jacobian of the measurement function with respect to the tangent space element:

$$\mathtt{J} = \left.\frac{\partial \mathbf{h}}{\partial \boldsymbol{\tau}}\right|_{\mathcal{X}=\mathcal{X}_0} . \tag{2.76}$$

Remembering back to Section 2.5.2, we showed that substituting $\mathbf{h}(\mathbf{x}) = \mathtt{A}\mathbf{x} + \mathbf{b}$ into the general equation for a factor yields a Gaussian over $\mathbf{x}$ in the information form with parameters (Equations 2.41 and 2.42):

$$\boldsymbol{\eta} = \mathtt{A}^\top \Sigma_n^{-1}(\mathbf{z} - \mathbf{b})$$

$$\Lambda = \mathtt{A}^\top \Sigma_n^{-1} \mathtt{A} \,.$$

Instead, here we are substituting in the Taylor expansion $\mathbf{h}(\mathcal{X}) = \mathtt{J}\boldsymbol{\tau} + \mathbf{h}(\mathcal{X}_0)$ (Equation 2.75). Identifying $\mathtt{A} = \mathtt{J}$ and $\mathbf{b} = \mathbf{h}(\mathcal{X}_0)$, this substitution will yield a Gaussian $f(\mathcal{X}; \mathbf{z}, \mathcal{X}_0) = \mathcal{N}^{-1}(\boldsymbol{\tau}; \boldsymbol{\eta}, \Lambda)$ in the chosen tangent space with energy $E(\mathcal{X}; \mathbf{z}, \mathcal{X}_0) = \frac{1}{2}\boldsymbol{\tau}^\top \Lambda \boldsymbol{\tau} - \boldsymbol{\eta}^\top \boldsymbol{\tau}$, where:

$$\boldsymbol{\eta} = \mathtt{J}^\top \Sigma_n^{-1}(\mathbf{z} - \mathbf{h}(\mathcal{X}_0)) \tag{2.77}$$

$$\Lambda = \mathtt{J}^\top \Sigma_n^{-1} \mathtt{J} \,. \tag{2.78}$$

Note that the same calculations above could be applied to factors in which the state space is a composition of Euclidean and Lie Group variables. For example, for a factor connected to one $\mathbb{R}^2$ and two $\mathbf{SE}(2)$ variables:

$$\mathcal{X} = \begin{bmatrix} \mathcal{X}_1 \\ \mathcal{X}_2 \\ \mathcal{X}_3 \end{bmatrix} \in \langle \mathbb{R}^2, \mathbf{SE}(2), \mathbf{SE}(2) \rangle \,, \tag{2.79}$$

the Gaussian factor would be formed in the compound tangent space:

$$\boldsymbol{\tau} = \begin{bmatrix} \boldsymbol{\tau}_1 \\ \boldsymbol{\tau}_2 \\ \boldsymbol{\tau}_3 \end{bmatrix} \in \langle \mathbb{R}^2, T_{\mathcal{X}_{2,0}}\mathcal{M}, T_{\mathcal{X}_{3,0}}\mathcal{M} \rangle \,. \tag{2.80}$$

Here the tangent spaces are chosen around the group elements $\mathcal{X}_{2,0}$ and $\mathcal{X}_{3,0}$.

2.6.4 Message Passing at a Variable Node

Consider a Lie Group variable $\mathcal{X}_m$ connected to $N \geq 1$ factor nodes $l = 1 \dots N$. In a message passing step it must output a message to one of these factors $l = s$. Each of the $N - 1$ incoming messages is represented by a group element and a precision matrix in its tangent space:

$$\mu_{f_l \to \mathcal{X}_m} = \mathcal{N}(\bar{\mathcal{X}}_{f_l \to \mathcal{X}_m}, \Lambda^{-1}_{f_l \to \mathcal{X}_m}) \ . \tag{2.81}$$

To combine these messages, we must first transform all of the messages to Gaussians in our chosen tangent space at the previous estimate, which we denote $\mathcal{X}_{m,0}$. This is a sensible choice of tangent space because it requires minimal transformation of messages which introduce approximation errors. To transform the mean of an incoming message into this tangent space we perform:

$$\boldsymbol{\tau}_l = \bar{\mathcal{X}}_{f_l \to \mathcal{X}_m} \ominus \mathcal{X}_{m,0} \ . \tag{2.82}$$

The precision matrix is approximately transformed as:

$$\Lambda_l = \mathtt{J}_r^{\top}(\boldsymbol{\tau}_l) \Lambda_{f_l \to \mathcal{X}_m} \mathtt{J}_r(\boldsymbol{\tau}_l) \ , \tag{2.83}$$

where $\mathtt{J}_r(\boldsymbol{\tau})$ is the right Jacobian of $\mathcal{X} = \mathrm{Exp}(\boldsymbol{\tau})$:

$$\mathtt{J}_r(\boldsymbol{\tau}) = \lim_{\epsilon \to 0} \frac{\mathrm{Exp}(\boldsymbol{\tau} + \epsilon) \ominus \mathrm{Exp}(\boldsymbol{\tau})}{\epsilon} \ . \tag{2.84}$$

Now that all $\boldsymbol{\tau}_l$ and Λ_l are defined in the tangent space at $\mathcal{X}_{m,0}$, we can add all precision matrices to determine the total precision:

$$\Lambda_a = \sum_{l \neq s}^{N} \Lambda_l \ . \tag{2.85}$$

Next we take the product of the messages to obtain the tangent vector of the outgoing message:

$$\boldsymbol{\tau}_a = \Lambda_a^{-1} \sum_{l \in n(\mathcal{X}_m) \setminus s}^{N} \Lambda_l \boldsymbol{\tau}_l \ . \tag{2.86}$$

Finally we retract this tangent vector on $\mathcal{X}_{m,0}$ to obtain the group element which is the mean of the outgoing message:

$$\bar{\mathcal{X}}_{\mathcal{X}_m \to f_s} = \mathcal{X}_{m,0} \oplus \boldsymbol{\tau}_a \ , \tag{2.87}$$

and transform the total precision to the tangent space of the outgoing message mean $\bar{\mathcal{X}}_{\mathcal{X}_m \to f_s}$:

$$\Lambda_{\mathcal{X}_m \to f_s} = \mathtt{J}_r^{-\top}(\boldsymbol{\tau}_a) \Lambda_a \mathtt{J}_r^{-1}(\boldsymbol{\tau}_a) \ . \tag{2.88}$$

The outgoing message to factor s is then:

$$\mu_{\mathcal{X}_m \to f_s} = \mathcal{N}(\bar{\mathcal{X}}_{\mathcal{X}_m \to f_s}, \Lambda^{-1}_{\mathcal{X}_m \to f_s}) \ . \tag{2.89}$$

Note that the message sent is the mean element and the precision matrix, rather than the inverse precision matrix. We use the inverse precision in Equation 2.89 only for the purpose of writing the message in the covariance form — this applies to all messages.

2.6.5 Message Passing at a Factor Node

Having discussed how to linearise a factor to produce a Gaussian in the tangent space at the linearisation point, we now describe how to perform the message computation in this tangent space. Consider our example factor, connected to one $\mathbb{R}^2$ and two $\mathbf{SE}(2)$ variables:

$$\mathcal{X}_s = \begin{bmatrix} \mathcal{X}_1 \\ \mathcal{X}_2 \\ \mathcal{X}_3 \end{bmatrix} \in \langle \mathbb{R}^2, \mathbf{SE}(2), \mathbf{SE}(2) \rangle \, . \tag{2.90}$$

The factor $f_s(\mathcal{X}_s) = \mathcal{N}^{-1}(\boldsymbol{\tau}_s; \boldsymbol{\eta}_s, \Lambda_s)$ can be evaluated in the tangent space at $\mathcal{X}_{s,0}$, where:

$$\boldsymbol{\tau}_s = \begin{bmatrix} \mathcal{X}_1 - \mathcal{X}_{1,0} \\ \mathcal{X}_2 \ominus \mathcal{X}_{2,0} \\ \mathcal{X}_3 \ominus \mathcal{X}_{3,0} \end{bmatrix} \in \langle \mathbb{R}^2, T_{\mathcal{X}_{2,0}}\mathcal{M}, T_{\mathcal{X}_{3,0}}\mathcal{M} \rangle \, . \tag{2.91}$$

As before, we can partition the information vector and precision matrix as:

$$\boldsymbol{\eta}_s = \begin{pmatrix} \boldsymbol{\eta}_{s1} \\ \boldsymbol{\eta}_{s2} \\ \boldsymbol{\eta}_{s3} \end{pmatrix} \tag{2.92}$$

$$\Lambda_s = \begin{bmatrix} \Lambda_{s11} & \Lambda_{s12} & \Lambda_{s13} \\ \Lambda_{s21} & \Lambda_{s22} & \Lambda_{s23} \\ \Lambda_{s31} & \Lambda_{s32} & \Lambda_{s33} \end{bmatrix} \, . \tag{2.93}$$

We choose the third variable as the output variable and so take the product of the factor and incoming messages from variables $\mathcal{X}_1$ and $\mathcal{X}_2$. This conditioning on the states of variables $\mathcal{X}_1$ and $\mathcal{X}_2$ is achieved by adding the information parameters:

$$\boldsymbol{\eta}_{Cs} = \begin{pmatrix} \boldsymbol{\eta}_1 + \boldsymbol{\eta}_{\mathcal{X}_1 \to f_s} \\ \boldsymbol{\eta}_2 + \boldsymbol{\eta}_{\mathcal{X}_2 \to f_s} \\ \boldsymbol{\eta}_3 \end{pmatrix} \tag{2.94}$$

$$\Lambda'_{Cs} = \begin{bmatrix} \Lambda_{11} + \Lambda'_{\mathcal{X}_1 \to f_s} & \Lambda_{12} & \Lambda_{13} \\ \Lambda_{21} & \Lambda_{22} + \Lambda'_{\mathcal{X}_2 \to f_s} & \Lambda_{23} \\ \Lambda_{31} & \Lambda_{32} & \Lambda_{33} \end{bmatrix} \, . \tag{2.95}$$

The parameters $(\boldsymbol{\eta}_{\mathcal{X}_i \to f_s}, \Lambda'_{\mathcal{X}_i \to f_s})$ describe the incoming message $\mu_{\mathcal{X}_i \to f_s} = \mathcal{N}(\bar{\mathcal{X}}_{\mathcal{X}_i \to f_s}, \Lambda^{-1}_{\mathcal{X}_i \to f_s})$ from variable $\mathcal{X}_i$ transformed into the tangent space at $\mathcal{X}_{i,0}$:

$$\boldsymbol{\tau}_i = \bar{\mathcal{X}}_{\mathcal{X}_i \to f_s} \ominus \mathcal{X}_{i,0} \tag{2.96}$$

$$\Lambda'_{\mathcal{X}_i \to f_s} = \mathtt{J}_r^{\top}(\boldsymbol{\tau}_i) \Lambda_{\mathcal{X}_i \to f_s} \mathtt{J}_r(\boldsymbol{\tau}_i) \tag{2.97}$$

$$\boldsymbol{\eta}_{\mathcal{X}_i \to f_s} = \Lambda'_{\mathcal{X}_i \to f_s} \boldsymbol{\tau}_i \, . \tag{2.98}$$

To complete message passing, from this joint distribution parameterised by $(\boldsymbol{\eta}_{Cs}, \Lambda_{Cs})$, we marginalise out variables $\mathcal{X}_1$ and $\mathcal{X}_2$ to obtain a Gaussian distribution over the output variable $\mathcal{X}_3$. For this we follow Equations 2.60 and 2.61 to obtain the outgoing message in the chosen tangent space at the outgoing variable $T_{\mathcal{X}_{3,0}}\mathcal{M}$. We denote the parameters of this message in the tangent space: $\mathcal{N}^{-1}(\boldsymbol{\tau}_3; \boldsymbol{\eta}_{M3}, \Lambda_{M3})$. Lastly, we transform this message to a Lie Group element with precision matrix in its own tangent space:

$$\boldsymbol{\tau}_{M3} = \Lambda^{-1}_{M3} \boldsymbol{\eta}_{M3} \tag{2.99}$$

$$\bar{\mathcal{X}}_{f_s \to \mathcal{X}_3} = \mathcal{X}_{3,0} \oplus \boldsymbol{\tau}_{M3} \tag{2.100}$$

$$\Lambda_{f_s \to \mathcal{X}_3} = \mathtt{J}_r^{-\top}(\boldsymbol{\tau}_{M3}) \Lambda_{M3} \mathtt{J}_r^{-1}(\boldsymbol{\tau}_{M3}) \, . \tag{2.101}$$

The outgoing message is then:

$$\mu_{f_s \to \mathcal{X}_3} = \mathcal{N}(\bar{\mathcal{X}}_{f_s \to \mathcal{X}_3}, \Lambda^{-1}_{f_s \to \mathcal{X}_3}) \, . \tag{2.102}$$

2.6.6 Belief Update

At any stage, we can calculate a new marginal distribution at a variable node $\mathcal{X}_m$ by taking the product of all the latest incoming messages from neighbouring factors. For non-Euclidean manifold variables, we need to do this calculation in a chosen tangent space. We use the tangent space at the previously calculated belief mean of the variable (or some simple initialisation if this is the first time we are doing a belief update at this node) which we call $\bar{\mathcal{X}}_{m,0}$. All incoming messages from adjacent factors f_s:

$$\mu_{f_s \to \mathcal{X}_m} = \mathcal{N}(\bar{\mathcal{X}}_{f_s \to \mathcal{X}_m}, \Lambda^{-1}_{f_s \to \mathcal{X}_m}) \, , \tag{2.103}$$

can be transformed into this tangent space $T_{\bar{\mathcal{X}}_{m,0}}\mathcal{M}$ using Equations (2.82) and (2.83) to yield Gaussians with parameters $\boldsymbol{\eta}_s$ and Λ_s for each factor.

Next we sum all precision matrices to determine the total precision:

$$\Lambda_a = \sum_{s \in N(m)} \Lambda_s \, , \tag{2.104}$$

and combine the messages to obtain the total tangent vector:

$$\boldsymbol{\tau}_a = \Lambda_a^{-1} \sum_{s \in N(m)} \Lambda_s \boldsymbol{\tau}_s \ . \tag{2.105}$$

Finally we retract this tangent vector on $\bar{\mathcal{X}}_{m,0}$ to obtain the group element which is the mean of the new belief of the variable:

$$\bar{\mathcal{X}}_m = \bar{\mathcal{X}}_{0,m} \oplus \boldsymbol{\tau}_a \ . \tag{2.106}$$

The belief precision is lastly obtained by transforming the total precision into the tangent space $T_{\bar{\mathcal{X}}_m}\mathcal{M}$:

$$\Lambda_m = \mathtt{J}_r^{-\top}(\boldsymbol{\tau}_a)\Lambda_a \mathtt{J}_r^{-1}(\boldsymbol{\tau}_a) \ . \tag{2.107}$$

CHAPTER 3

Beyond the Standard Algorithm

Contents

We have introduced Gaussian belief propagation in its basic form as a probabilistic inference algorithm for Gaussian estimation problems. However, to solve real practical problems with GBP, we often need a number of extra details and tricks which we discuss in this chapter. Later chapters focus on large scale applications of GBP to practical problems.

3.1 Example Applications

Before discussing how GBP can be extended to handle general and practical problems in Spatial AI, we first introduce a number of simple example problems. All examples are linear problems so that we can simply compute and compare against the true marginals. These will be referred back to throughout this chapter to illustrate various characteristics of GBP.

3.1.1 1D Surface Estimation

In our first example, the goal is to reconstruct a height map surface from a set of point measurements. Each measurement has a perfectly known horizontal position, and Gaussian uncertainty in the vertical direction. We also have a Gaussian smoothness assumption over the surface.

We consider a one-dimensional height map here. We wish to estimate the surface heights at a quantised set of horizontal positions, and we define a variable node for the height at each of these positions. There are an arbitrary number of measurements, each of which results in a factor node in the graph. The smoothness model is a Gaussian constraint on the relative height of every pair of consecutive variables. This results in another set of factor nodes in the graph joining neighbouring nodes. The factor graph of this problem is shown in Figure 3.1. This simple example is used in Kevin Murphy's recent textbook [Murphy, 2023] to present belief propagation and Figure 9.5 in his textbook is generated with our Python code (gauss-bp-1d-line.ipynb).

State representation. The problem is concerned with estimating the one-dimensional height y_i of a number variables. Each variable node is located at an x coordinate x_i that is treated as a fixed / observed variable in the graph.

Measurement factors. We have a number of measurements z_m which describe the height of the surface at a perfectly known location x_m. All height measurements have an associated fixed measurement covariance Σ_m.

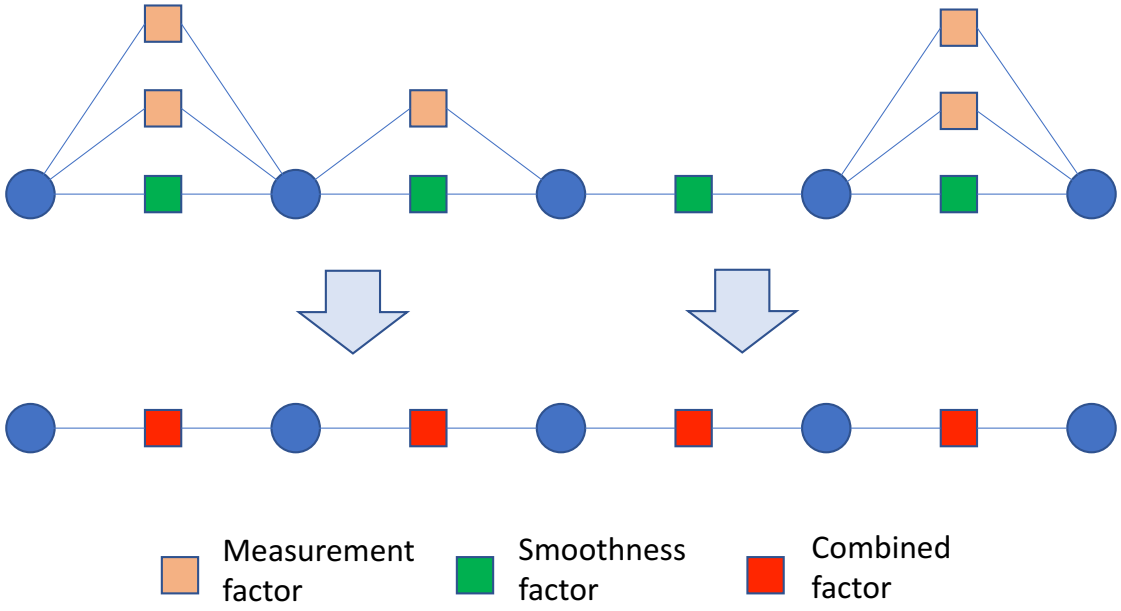


Figure 3.1: Factor graph for 1D surface reconstruction. Blue variable nodes are evenly spaced along the surface. Every consecutive pair is linked by a smoothness factor (green). There are also measurement factors (orange) between some variables nodes where measurements are recorded. Since all factors have linear measurement functions, we combine all factors between each pair of variables into a single compound linear factor. This results in a factor graph with a chain structure, shown at the bottom.

A measurement at horizontal location x_m is assumed to have a linearly interpolated dependence on the state variables having x coordinates x_1 and x_2 which lie either side of it. We define the weighting:

$$\lambda_m = \frac{x_m - x_1}{x_2 - x_1} \,, \tag{3.1}$$

as the horizontal displacement between the measurement position and the position of the first variable node, as a proportion of the horizontal displacement between the two variable nodes. The measurement function is a linear combination of the height values of the variables on either side of the measurement:

$$h_m(y_1, y_2; \lambda_m) = (1 - \lambda_m)y_1 + \lambda_m y_2 \,. \tag{3.2}$$

The Jacobian of this linear measurement function is:

$$\mathtt{J}_m = \begin{bmatrix} 1 - \lambda_m & \lambda_m \end{bmatrix} \in \mathbb{R}^{1\times 2} \,, \tag{3.3}$$

where:

$$h_m(y_1, y_2; \lambda_m) = J_m \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \,. \tag{3.4}$$

Smoothness factors. A simple smoothness factor is defined between every pair of consecutive variable nodes. All smoothness factors have an associated fixed covariance Σ_s. The measurement

function is:

$$h_s(y_1, y_2) = y_2 - y_1 \ . \tag{3.5}$$

The scalar 'measurement', $z_s = 0$, is set to zero to encourage consecutive variable nodes to take the same value. The Jacobian is:

$$\mathtt{J}_s = \begin{bmatrix} -1 & 1 \end{bmatrix} \in \mathbb{R}^{1\times 2} \ . \tag{3.6}$$

Combining factors. Any factors can be arbitrarily combined into a single compound factor by taking the product of the factors, for example:

$$f_{12}(x_1, x_2) f_{23}(x_2, x_3) = f_{\text{compound}}(x_1, x_2, x_3) \ . \tag{3.7}$$

Combining factors so that the resulting graph is a tree structure is the basis of the junction tree class of algorithms. If the combined factors are linear then the resulting compound factor is also linear. Here we combine all smoothness and measurement factors that connect to the same pair of variables. This leads to chain graph with no loops as shown in Figure 3.1. GBP is known to converge perfectly with one pass in each direction for tree graphs such as a chain.

General implementation details

We have two implementations of this 1D surface estimation example: one Python implementation and one JavaScript implementation as part of an interactive diagram. As outlined in Chapter 1, GBP is well-suited for efficient, highly parallel, distributed implementations due to its local message passing character. Our simple example implementations are centralised prototypes for future distributed implementations, and use classes to simulate the decentralisation of data and processing.

In both implementations, we create `VariableNode` or `FactorNode` classes that are instantiated for each node in the factor graph. These classes contain the general code for message passing from variable and factor nodes respectively. Child classes of these parent classes implement the particular state space or factor function models for the problem in question.

The two implementations differ on where the messages are stored. In the Python implementation, an `Edge` object is instantiated for every connection between a variable and a factor, and stores the latest messages in both directions along this edge. So when a variable or factor node needs to carry out a message passing step, it reads the appropriate incoming messages from all edges it is connected to apart from one, performs the calculation, and then writes the message to the outgoing edge.

In the JavaScript implementation, the factor-to-variable messages are stored at the factor node that is sending the message and variable-to-factor messages are ephemeral. The variable-to-factor messages are not computed at the variable nodes but instead the variable node sends its current belief to adjacent factor nodes. The incoming variable-to-factor message is recovered by dividing the variable belief by the previous factor-to-variable message. In this way, variable and factor nodes communicate directly with each other without intermediate edges.

To form a best up-to-date estimate at a variable node at any point in time, we can read and add factor-to-variable message parameters from *all* connected edges or factor nodes. Similarly, if a factor node reads and adds all incoming variable-to-factor messages to its factor potential at any moment, we get the current estimate of its energy based on all information available. This can be used for instance to relinearise it (or as we will see later, to apply a robust weighting).

Initialisation is also usually not problematic because our parameterisation of Gaussian distributions in the information form means that we can safely represent the uncertainty over variables even if the factors connecting to them do not fully constrain their degrees of freedom (i.e. the precision matrices are singular) and covariances would not be defined. However, if we do wish to visualise uncertainties from a covariance we can add weak stabilising unary factors.

Python implementation. The Python CPU simulation, with a measurement dataset read in from a text file and interactive graphics, is available at `http://www.doc.ic.ac.uk/~ajd/bp1d.py`. The code requires a straightforward Python3 installation with NumPy for numerics and PyGame for interactive visualisation.

JavaScript implementation. The JavaScript implementation is part of the online visual introduction to GBP [Ortiz et al., 2021]. Screenshots of the interactive figure for 1D surface estimation are shown in Figure 3.5 and the full interactive version can be found at `https://gaussianbp.github.io/#gbp1d`. Our interactive diagram makes use of Node.js to handle data and dynamic content and Svelte to automatically write code during compilation that updates the DOM reactively based on changes in the app. The code for all interactive figures is available here: `https://github.com/gaussianBP/gaussianBP.github.io`.

3.1.2 Image denoising

The problem of image denoising is a 2D generalisation of our 1D surface estimation example. The goal is to estimate the denoised intensities at each pixel given a noisy image by using smoothness

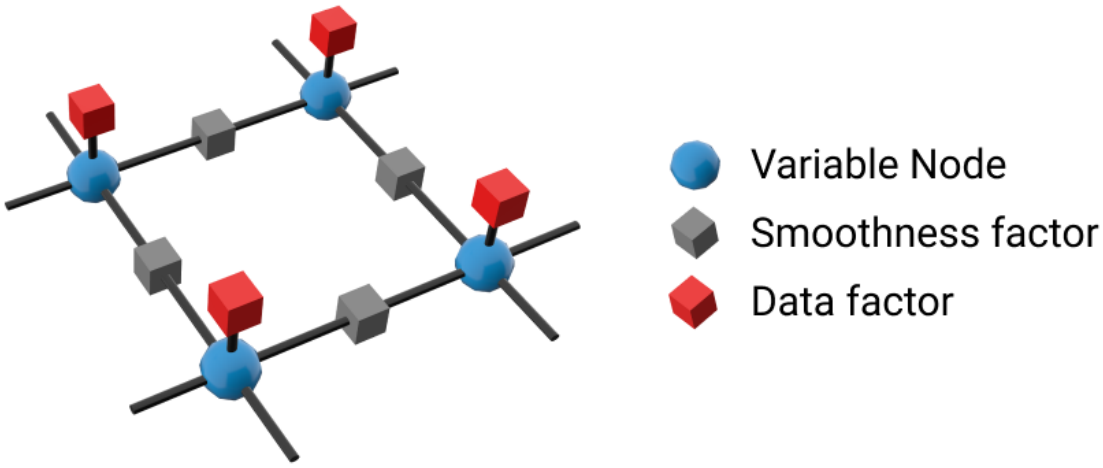


Figure 3.2: Section of the factor graph for image denoising.

constraints to remove the noise. The factor graph remains linear, however unlike in the 1D case, the graph is loopy meaning iterative message passing will be needed to converge on a global solution.

State representation. We are estimating the one-dimensional greyscale pixel intensities y_{ij} at a discrete pixel location (i, j).

Measurement / data factors. Different to 1D surface estimation where measurement factors were at any location, they are now at the same discrete pixel locations as the variable nodes with one per node. As a result the data factors only connect to a single variable node, now acting like a prior on that variable's value. The data factors have an identity measurement function: $h_m(y) = y$, covariance Σ_m and measurement z_m which is the intensity value in the noisy image at that pixel.

Smoothness factors. Smoothness factors connect each variable node to the 4 nodes at adjacent pixels. All smoothness factors have an associated covariance Σ_s. As before, the measurement function is:

$$h_s(y_1, y_2) = y_2 - y_1 \ . \tag{3.8}$$

The factor graph for the image denoising problem is shown in Figure 3.2.

Implementation details. This example is implemented in the interactive figure at https://gaussianbp.github.io/#attentiongl. This implementation is based on the JavaScript GBP library described for the 1D surface estimation however in this case, the implementation is parallelised on GPU hardware. We use WebGL to parallelise GBP which can be implemented using a similar computational structure as applying a 3x3 filter to the image. GPUs operate in a SIMD (same instruction multiple data) paradigm which enables them to be very efficient at

applying the same operation in parallel across a grid structure. In this problem, the factor graph is a grid and all factors are the same, so image processing with GBP is well-suited for GPU acceleration. With a NVIDIA GeForce RTX 2080 Ti GPU, our implementation can perform 10000 iterations of synchronous GBP per second on a 600x600 pixel image.

3.1.3 2D Pose Graph

Our next example is a linear 2D pose graph problem. This is a simple version of the pose graph optimisation problem in mobile robotics where many robots or a single exploring robot must estimate their global locations from a network of purely relative measurements. This application of GBP still only involves linear factors, but again the graph is loopy.

State representation. We would like to estimate the 2D position / pose of each variable node in a plane:

$$\mathbf{x}_i = \begin{bmatrix} x_i \\ y_i \end{bmatrix} \in \mathbb{R}^2 \,. \tag{3.9}$$

Relative position factors. Each factor encodes a 2D Euclidean relative pose measurement $\mathbf{z} \in \mathbb{R}^2$ between two nodes. The measurement function is:

$$\mathbf{h}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_j - \mathbf{x}_i \,, \tag{3.10}$$

and the factor has fixed diagonal measurement covariance:

$$\Sigma_n = \begin{bmatrix} \sigma_n^2 & 0 \\ 0 & \sigma_n^2 \end{bmatrix} \,. \tag{3.11}$$

The measurement function is linear and can be expressed as:

$$\mathbf{h}(\mathbf{x}_i, \mathbf{x}_j) = \mathsf{J} \begin{bmatrix} x_i & y_i & x_j & y_j \end{bmatrix}^\top \,, \tag{3.12}$$

with Jacobian:

$$\mathsf{J} = \begin{bmatrix} -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix} \,. \tag{3.13}$$

Unary / prior factors. We also include a unary factor connected to each node which sets a prior on each pose. The unary factor connected to variable node $\mathbf{x}_0$ anchors this node to the origin with a very small covariance. All other unary factors have high covariance, which encourage the poses to lie within the bounds of the plane. The prior factors are implemented with measurement function: $h(\mathbf{x}) = \mathbf{x}$. The measurement $\mathbf{z}$ sets the prior mean and the prior covariance is Σ_p.

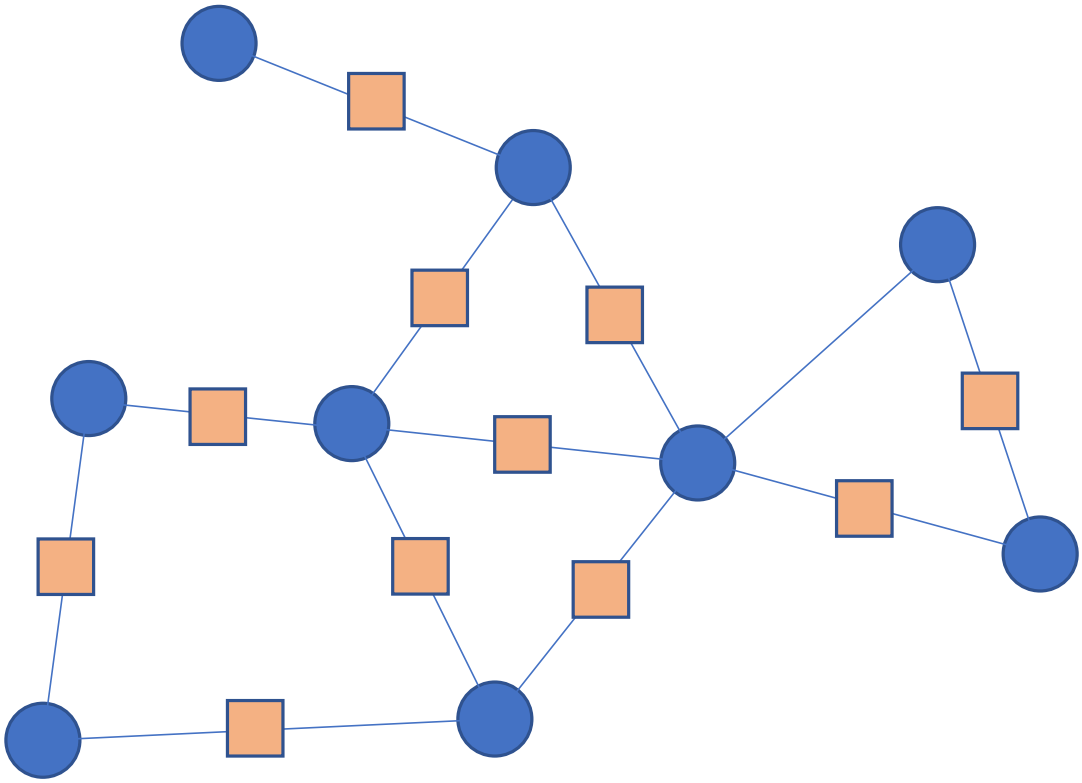


Figure 3.3: Factor graph for a 2D pose graph problem. Variable nodes (blue) scattered around the 2D plane are linked by randomly generated measurement factors (orange) between nearby nodes. Note that in our implementation we also use some unary factors on the variables (one strong, to anchor one variable, the others very weak), which are not displayed here.

The factor graph for this 2D pose graph problem is shown in Figure 3.3.

Implementation details. We again have two implementations of this example. The Python simulation available at http://www.doc.ic.ac.uk/~ajd/bpmap.py. Two interactive figures produced with JavaScript implement this 2D pose graph example and available at https://gaussianbp.github.io/#gbp_intuition and https://gaussianbp.github.io/#playground. The first allows the user to pass message through 3 increasingly loopy graphs — a chain, a loop and a grid. In the second the user can construct their own pose graph of arbitrary topology, set the initialisation and then choose how messages are passed through the graph.

3.1.4 Incremental SLAM

The final example is a 2D incremental SLAM problem in which a robot is moving in a 2D environment, but is simplified by using only linear factors. We simulate a 2D cartesian SLAM problem, where as a 'robot' translates it leaves a history of pose variable nodes, with each consecutive pair

joined by a factor on their relative locations representing an odometry measurement. Scattered throughout the simulated 2D environment are landmarks that the robot can observe. From each new robot pose, factors are added to the graph to represent measurements of the landmarks within a bounded distance. All measurements in the simulation have randomly sampled Gaussian noise, using different but constant covariances for the odometry factors and measurement factors. There is no rotation in the simulation, and all measurements are in Cartesian space, so this is again a formulation where the dependence of measurements on variables is purely linear. The mathematical details of variable and factor message passing are the same as in the 2D constraint graphs of Section 3.1.3 with the main differences being that we are estimating both robot and landmark positions and that the problem is incremental. We have a strong pose factor attached to the first robot variable node, anchoring this node and effectively defining the coordinate frame for SLAM.

See the factor graph for our simulated 2D SLAM problem in Figure 3.4.

State representation. We are concerned with estimating the historic poses of the moving robot: $\mathbf{x}_t = [x_t, y_t]^\top \in \mathbb{R}^2$ as well as the landmark positions: $\mathbf{l}_i = [x_i, y_i]^\top \in \mathbb{R}^2$.

Odometry factors. The odometry factors constrain the relative pose of successive robot positions $\mathbf{x}_t$ and $\mathbf{x}_{t+1}$. They take the same form as the relative position factors in the previous 2D pose graph example. The measurement function is: $\mathbf{h}(\mathbf{x}_t, \mathbf{x}_{t+1}) = \mathbf{x}_{t+1} - \mathbf{x}_t$, measurement covariance: Σ_o and measured odometry $\mathbf{z}_o \in \mathbb{R}^2$.

Measurement factors. Measurement factors constraint the relative position between a robot pose and a landmark that it has observed. They again have the mathematical form as the 2D relative position factors. The measurement function is: $\mathbf{h}(\mathbf{x}_t, \mathbf{l}_i) = \mathbf{l}_i - \mathbf{x}_t$, measurement covariance: Σ_m and landmark position measured by the robot: $\mathbf{z}_m \in \mathbb{R}^2$.

Prior factors. As in the 2D pose graph problem, we include a strong prior on the robot pose $\mathbf{x}_0$ and a weak prior on all other robot poses and landmarks constraining them to lie in the plane.

Implementation details. We again have two interactive implementations of this incremental example. The Python simulation is available at `http://www.doc.ic.ac.uk/~ajd/bpslam.py`. The interactive online figure, produced in JavaScript, is available at `https://gaussianbp.github.io/#playground`. In both examples, the robot can be controlled using the WASD keys.

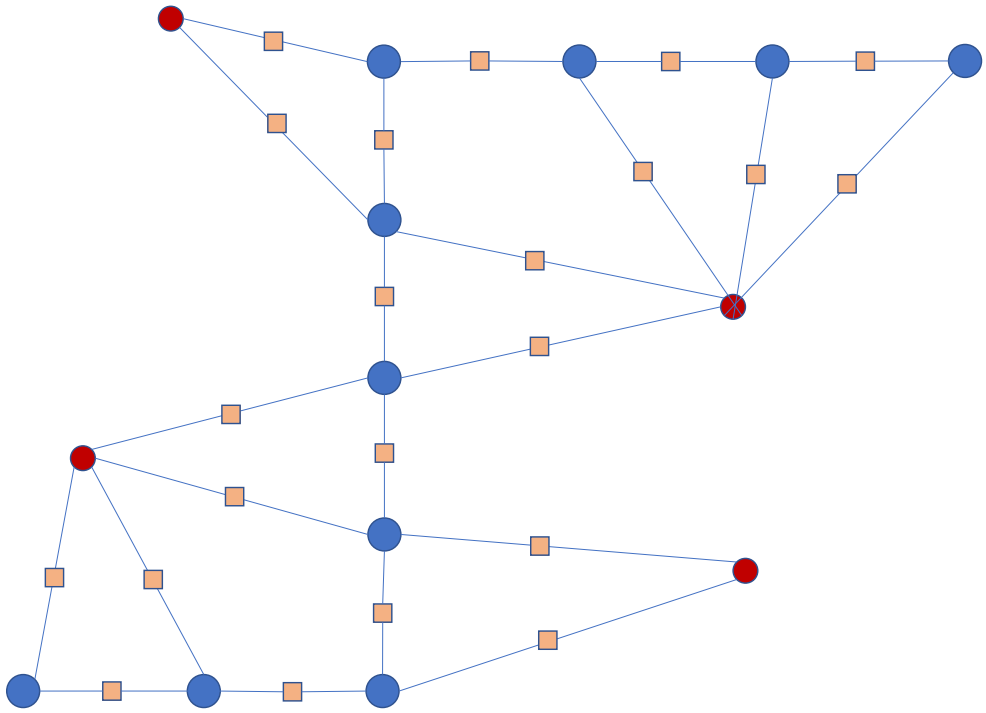


Figure 3.4: Factor graph for a 2D SLAM problem. Blue nodes are variable nodes describing the robot position while red nodes are landmark variable nodes. The orange squares are measurment factors representing relative 2D constraints.

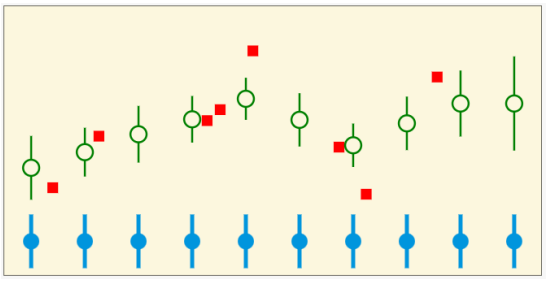
3.2 Message Schedules

Having introduced various example problems, in this section we begin our discussion of making GBP work in practice by examining various serial and parallel message schedules. We begin with serial schedules on tree graphs that guarantee exact marginal computation before moving on to serial and parallel schedules for loopy graphs. Through examples, we aim to build an intuitive understanding of GBP message passing and demonstrate that GBP can converge with an arbitrary message schedule.

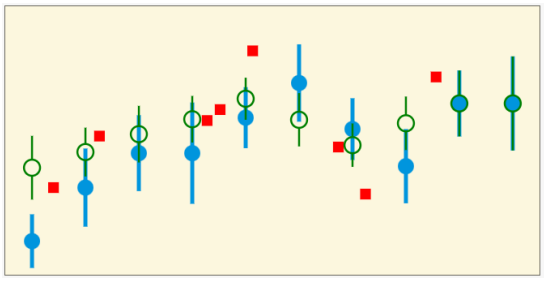
3.2.1 Serial Schedules for Tree Graphs

Serial message passing schedules are defined as schedules in which only one message is sent through the graph at each time step. Here we discuss two types of serial schedules, the *sweep* schedule and the *random* schedule, which represent the two extremes of efficiency for serial processing on tree graphs.

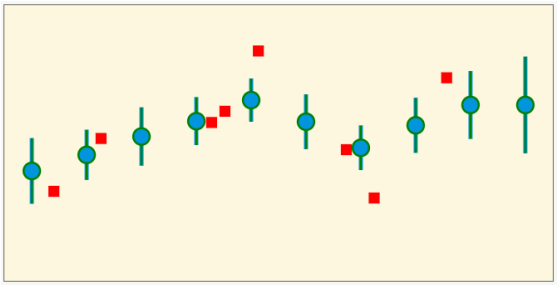
For tree graphs, GBP computes the exact marginals for a given node once that node has received



(a) **Iteration 0**. Before any message passing, the belief distributions (blue) are displayed at the bottom of the screen.



(b) **Iteration 9**. After a left-to-right sweep of messages, the rightmost node has received messages from all other nodes and its belief exactly computes the true marginal. All other nodes have not yet received messages from nodes to their right and so their beliefs are not yet the true marginals.



(c) **Iteration 18**. After a sweep back and forth through the chain, all nodes have received all messages from all other nodes. All beliefs now exactly compute the true marginal distributions.

Figure 3.5: Message passing with a sweep schedule exactly computes all of the marginal posterior distributions for this 1D surface estimation problem, represented by a chain graph. The y values being estimated are the vertical positions of the nodes and the horizontal x positions are fixed. The red squares display the locations of the surface measurements however pairwise smoothing factors between adjacent nodes are not displayed. The belief distributions are shown in blue, with the mean at the centre of the circle, and the line representing ± 1 standard deviation of uncertainty. The true marginal posterior distributions are shown in green. See the full interactive version of this figure at: https://gaussianbp.github.io/#gbp1d.

message from all other nodes. As previously discussed, the most efficient way (using a minimal number of messages) for all nodes to receive messages from all other nodes in trees is a sweep schedule (also known as floodfill). A sweep schedule involves sending messages out from an arbitrarily chosen root node down to the leaves and then back up to the root node, after which all nodes have received messages from all other nodes. In this case, the number of messages required for exact inference is twice the number of edges in the graph.

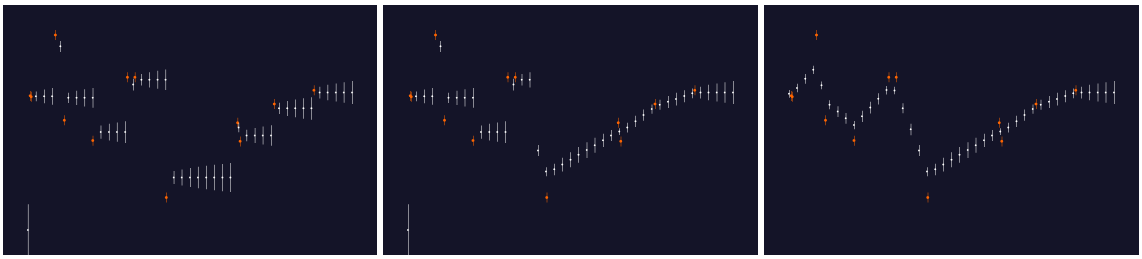
We visualise a sweep schedule for a 1D surface estimation problem in Figure 3.5 using our JavaScript simulation. In this example, the root node is chosen to be the leftmost node and the graph is a chain. After a sweep of messages from left-to-right in Figure 3.5b, the rightmost node has exactly computed its marginal as it has received messages from all other nodes. The second right-

most node has not yet received a message from the rightmost node, however it appears to have converged on the true marginal because the message it will receive from the rightmost node will have the same mean as its current belief estimate. After the right-to-left sweep of messages back through the graph, Figure 3.5c shows that all nodes have now received messages from all other nodes and the beliefs exactly compute the true marginals.

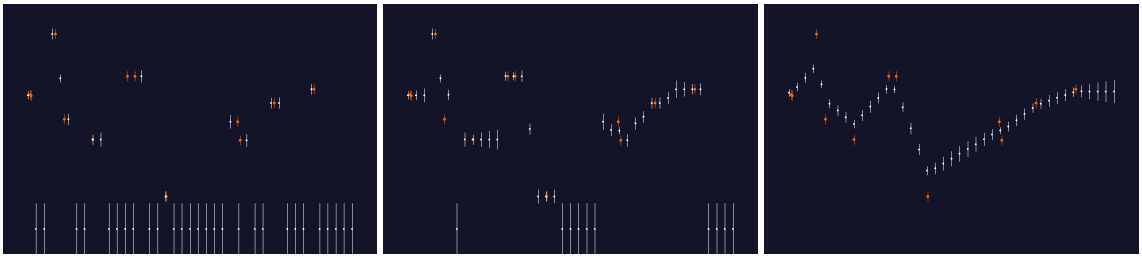
At the other end of the efficiency spectrum is the random schedule in which one message is sent along a random edge at each time step. For tree graphs in particular, a random schedule is a very inefficient way of communicating messages from all nodes to all other nodes, as is required for inference. In Figure 3.6 using our Python simulation, we compare a sweep and random schedule for a larger 1D surface estimation problem. In the top row, we again see the sweep schedule with the leftmost node chosen as the root. With one full traversal in both directions, all variables are fully 'informed' from all parts of the graph, and we achieve the globally optimal solution. In the bottom row, we see the progress of a fully random message passing schedule, where many steps will incur wasted work if the variable or factor sending the message has not itself updated since its last message. However, the most interesting thing to observe here is that full convergence to the global optimum is still reliably reached after some thousand iterations, with purely random and distributable processing. This is of course to be expected, as for tree graphs we know exactly how many messages need to be sent for exact inference and in what order. For example, for variable 3 to receive a message from variable 1, variable 1 needs to first send a message to variable 2 which then later sends a message to variable 3. A random schedule will eventually achieve this sequence of messages for all source and target variables given a large enough number of random messages.

3.2.2 Schedules for Loopy Graphs

Unlike tree graphs which have a fixed sequence of messages required to compute the exact marginals, loopy graphs have no such guarantee and instead we must choose a message schedule that we hope will lead to convergence over many iterations. There are many possible message passing schedules for loopy graphs which vary empirically in both the rate of convergence and ability to converge. We begin by discussing serial schedules for loopy graphs before moving on to consider parallel schedules that can take advantage of parallel processing.



(a) **Sweep schedule.** Messages are passed first from left-to-right and then back from right-to-left. After 80 steps / messages, information has propagated all the way from left-to-right. Messages return back from right-to-left and by 160 messages we have reached an exact solution. Note that variables further from measurements have larger marginal variances.



(b) **Random serial schedule.** At each step we randomly choose one of the 80 edges which connects a variable to a factor, and also randomly choose to pass a message either from factor to variable or from variable to factor. Clearly this is a very inefficient strategy, especially for a serial processor, and many messages must be passed. Eventually after several thousand random messages, all nodes have received messages from all other nodes, and we reach the same globally exact solution.

Figure 3.6: Sweep and random schedule for a larger 1D surface estimation problem. Measurements are shown with ± 1 standard deviation of uncertainty in red, and 41 evenly spaced variable state estimates are shown in white. Each measurement defines a factor involving the two horizontally closest variables, and a smoothness factor also joins every pair of adjacent variables. Variables which have not yet received an informative message are initialised to a default zero value (at the bottom of the screen) with large variance. The numbers count the total number of messages sent or equivalently the iteration/time step in this serial processing paradigm.

Serial Schedules

The choice of serial schedule can affect the rate of convergence for loopy graphs (see Koller and Friedman [Koller and Friedman, 2009], pages 407-409). The random serial schedule, from the previous section, empirically is remarkably reliable for loopy graphs however convergence is very slow due to the large proportion of redundant messages. A fixed 'round-robin' schedule in which nodes send messages in a fixed order is also effective and tends to converge faster than a random serial schedule. Better yet, it's possible to prioritise sending messages that we think will contribute most to the overall convergence of the system (there is evidence that the brain may apply a similar economical principle [Evans and Burgess, 2019]). This is the idea behind residual

belief propagation (RBP) [Elidan et al., 2006] and similar variants [Sutton and McCallum, 2007, Ranganathan et al., 2007] which form a message queue according to the norm of the difference from the previous message. The downside of these approaches is that it can be computationally expensive to build the message queue.

Other serial schedules for loopy graphs take inspiration from the sweep schedule that performs exact marginal computation on tree graphs. Using the fact that we can always divide a loopy graph into a number of overlapping sub-trees, the tree reparameterisation (TRP) schedule iteratively sends sweeps up and down each of these sub-trees. Each sub-tree is therefore recursively locally converged and then moved away from convergence when an overlapping sub-tree is converged. If the problem structure can be cleanly separated into a number of minimally overlapping sub-trees, then the TRP schedule can be a good choice.

Parallel Synchronous Schedule

As discussed, it is trivial to distribute GBP as messages are computed locally at variable and factor nodes and then passed locally through the graph. Parallel processing is therefore easily applied by having many nodes compute and send messages in parallel at each step. The parallel schedule in which all variable nodes send messages to all their adjacent variable nodes (via factor nodes) at each step is the *synchronous* schedule.

Synchronous scheduling was briefly mentioned when discussing BP on loopy graphs in Section 2.3.2. We outlined that a synchronous update consists of 3 phases: variable-to-factor message passing, factor-to-variable message passing and belief updates. In the first phase, all variable nodes compute and send messages to all adjacent factor nodes based on their latest belief. In the second phase, all factor nodes compute and send messages to all adjacent variable nodes based on their incoming messages. In the final phase, all variable nodes update their local belief estimate based on their incoming messages.

In a distributed GBP implementation, when the number of available parallel processes is larger than the maximum number of nodes in the graph, the computational cost of a synchronous update is approximately constant with respect to the size of graph. This is because as the graph grows, a synchronous update simply uses more of the parallel processes without any additional serial processing. In our example Python CPU implementation, we do not have this availability of a large number of parallel processes and so simulate this parallelism by looping through all variable-to-factor messages, then factor-to-variable messages and lastly belief updates.

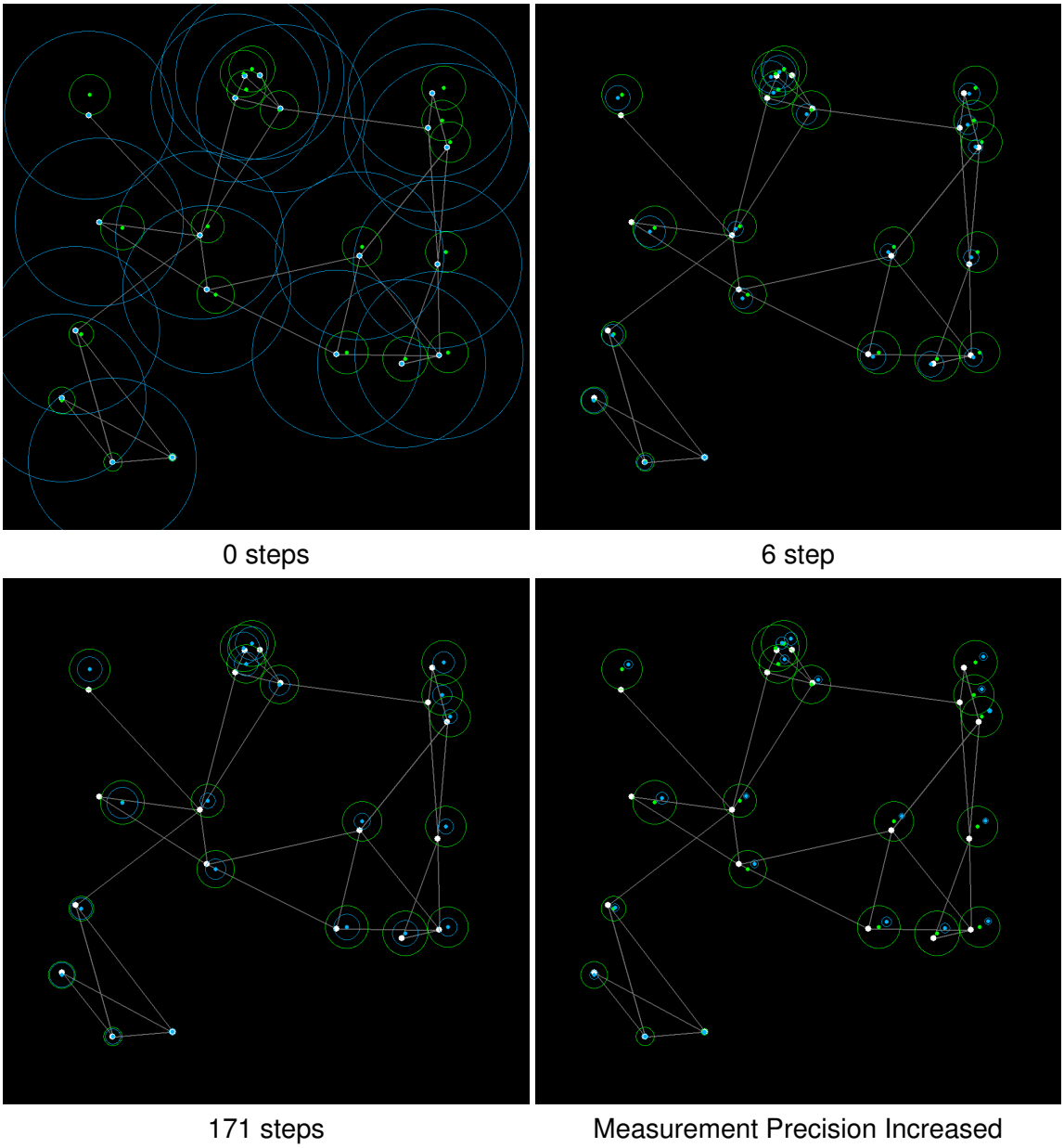


Figure 3.7: **Synchronous GBP for linear 2D pose graph optimisation.** The problem has 20 variable nodes and 50 relative measurements. The ground truth poses are shown in white and grey lines are drawn between two variables connected by a measurement factor. The belief estimates are displayed in blue (mean point locations and one standard deviation as a circle) and the true marginal distributions in green. The variable at the bottom centre of the scene has a strong prior factor anchoring the problem. We see that only a few steps of synchronous message passing are needed for the graph to reach good relative estimates. After 171 steps, information has fully propagated from the anchored node and the mean estimates from GBP are indistinguishable from the true marginal solution, though the covariances are in many cases overconfident. In the final panel, we show that after convergence (or at any other time) we can make dynamic and local changes to the factors, the effects of which are transmitted to the whole graph with no changes to the GBP algorithm. Here we increase the precision of all measurement factors which changes the mean estimates because the weak pose factors are trusted less. (Note that we did not account for this in the green batch solution shown, which does not change.)

We visualise the progress of a simulated synchronous message passing schedule for the 2D pose graph example in Figure 3.7, and discuss its progress in the caption, including a comparison with the true marginal distributions. The true marginals for this linear problem are computed by adding all factors into a single large information matrix, and inverting this matrix to find the marginal mean and covariance for all variables. First and most importantly, we see that synchronous message passing results in beliefs that are consistent with the true marginals. In particular, as dictated by the theory [Weiss and Freeman, 2000], on convergence the belief means exactly compute the true marginal means while the covariances are often over-confident, particularly in the loopier parts of the graph.

Although in this small example it takes a large number of iterations for the means to converge on the true marginals, we notice that accurate relative estimates are obtained after fairly few steps. This highlights a second important property of GBP — it can achieve approximate local convergence without full global convergence. Due to the factorised structure of GBP [Diehl et al., 2018], global inference is achieved by jointly solving many interdependent local sub-problems. Local message passing can yield accurate relative local solutions which estimate the marginals up to global corrections that come from more distant parts of the graph. This can be clearly seen in Figure 3.7, in which after 6 synchronous iterations local parts of the graph have accurate relative poses which differ from the true marginals by similar global offsets. At this stage the estimates in the graph are most likely highly useful for any application where relative information is important, such as robot navigation. The difference between 6 steps and 171 steps is that the rooted node with a strong pose factor is able to propagate absolute information around the whole graph.

In the final panel of Figure 3.7, we give an example of the extreme flexibility of GBP estimation, and the ability of decentralised estimation methods to work in an editable, reversible way. After convergence, we change the precision of all of the factor nodes to a stronger value, and the result of this quickly propagates through the graph, with no global coordination needed. Any number of dynamic changes like this can easily be dealt with, which we think will be important in the future of Spatial AI, where for instance some human input or machine learning process produces an updated value of a prior assumption (e.g. surface smoothness) which with most estimation methods would have become 'baked into' the representation.

We also explore GBP with a synchronous schedule for the incremental SLAM example in Figure 3.8. In this simulation, as the robot moves the factor graph is generated automatically. Synchronous parallel message passing is constantly ongoing in the background optimising the full current graph. We observe that, due to its local nature, it takes many iterations for GBP to make

large global corrections induced by global events like loop closure. This will motivate multigrid and abstraction approaches to speed up the convergence of GBP in Section 3.4.3.

In Figure 3.9, we compare a random serial schedule with synchronous GBP for a 2D pose graph problem representing a grid. We again remarkably find that GBP can converge reliably to the true marginals with arbitrary random serial schedules. It takes 200 random messages in our example to converge on the true marginals while synchronous updates can converge in 8 steps. We strongly encourage the reader to explore the full interactive version of Figure 3.9 at https://gaussianbp.github.io/#gbp_intuition. This example aims to build intuition for GBP by exploring the effect of individual messages. The reader can click on a variable node to send messages to its adjacent variables and observe how neighbouring beliefs are updated. Through playing with this example, the reader should become confident that GBP can converge well with arbitrary message schedule.

Partially Synchronous / Attention based Schedules

There are many partially synchronous schedules in the middle ground between a fully synchronous parallel schedule and serial schedules. For example, when the number of parallel processes is fewer than the number of nodes in the graph or there is a fixed compute budget at each step, a fully synchronous schedule may not be desirable. Partially synchronous schedules choose a subset of the nodes to send messages at each step. This subset can be deterministic based on a user specification or random as in message dropout in which there is a fixed a probability of sending a message along a particular edge at each step. Partially synchronous schedules can also be more robustly convergent than synchronous schedules, as the fact that all nodes send messages based on their previous beliefs in fully synchronous updates can increase the chances of oscillations and non-convergence.

Deterministic partially synchronous schedules can be used when we are interested in approximate local solutions [1]. In these cases, GBP can operate in a **just-in-time** or **attention-driven** fashion, focusing synchronous processing on parts of the graph to solve local sub-problems as the task demands. This attention-driven scheduling can be very economical with compute and energy, only sending the most task-critical messages.

[1] One simple example is mapping two connected rooms. An accurate local map of one room can be constructed by focusing processing on the part of the factor graph in that room. For some applications this may be sufficient while for others it may be important to build a map with an accurate absolute position which may require longer range message passing between the parts of the graph corresponding to each separate room.

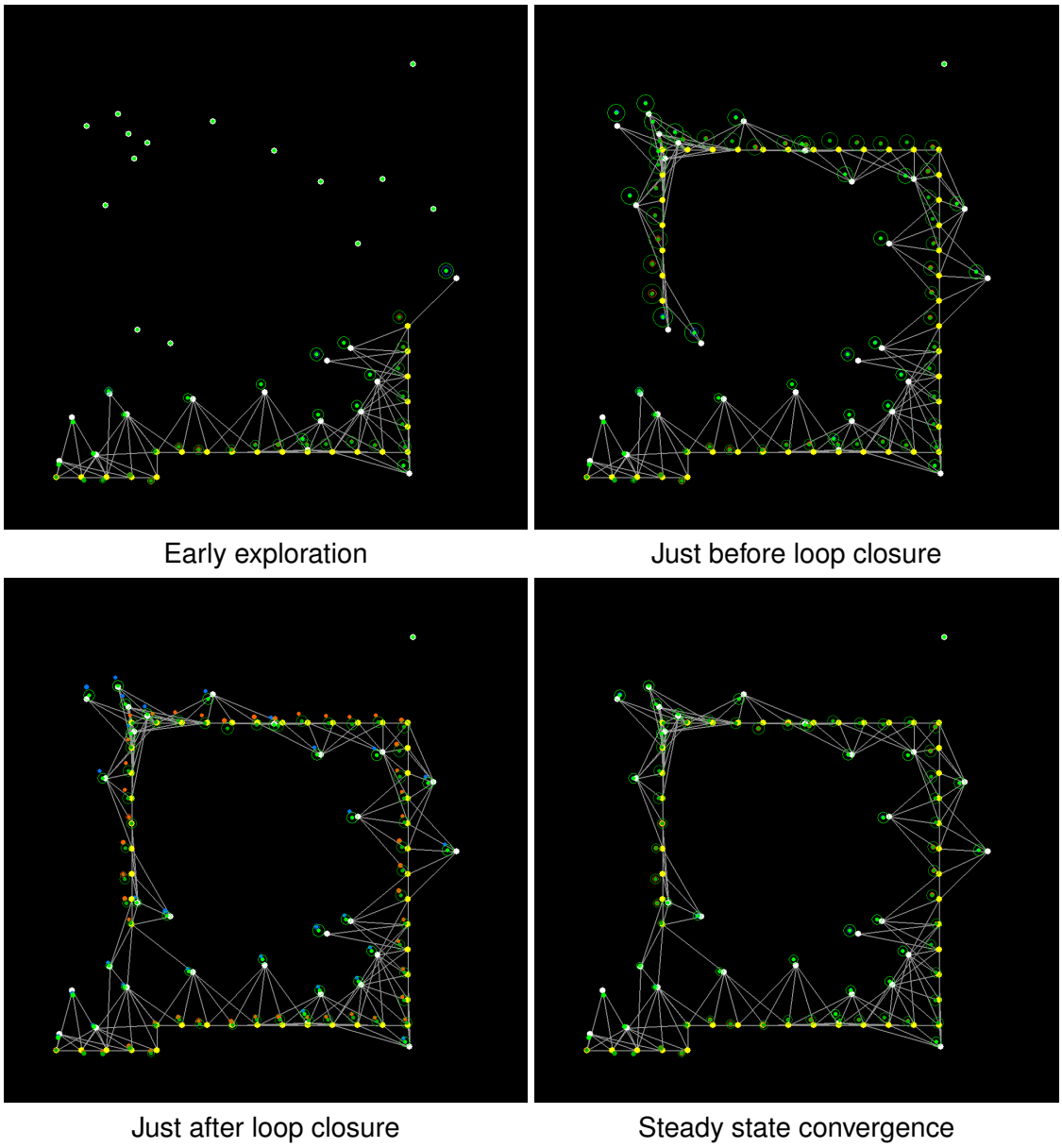
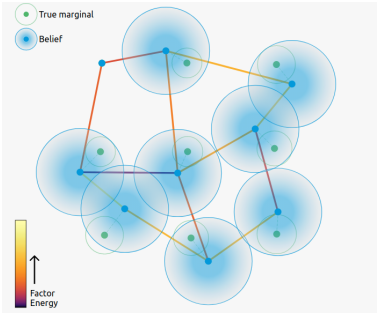
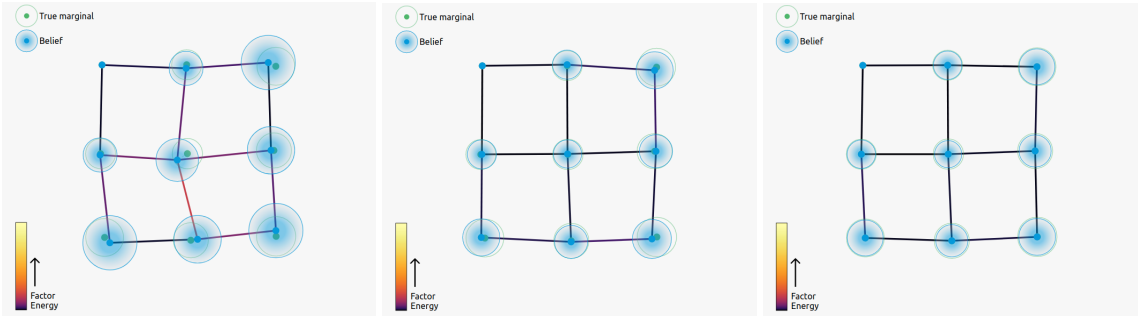


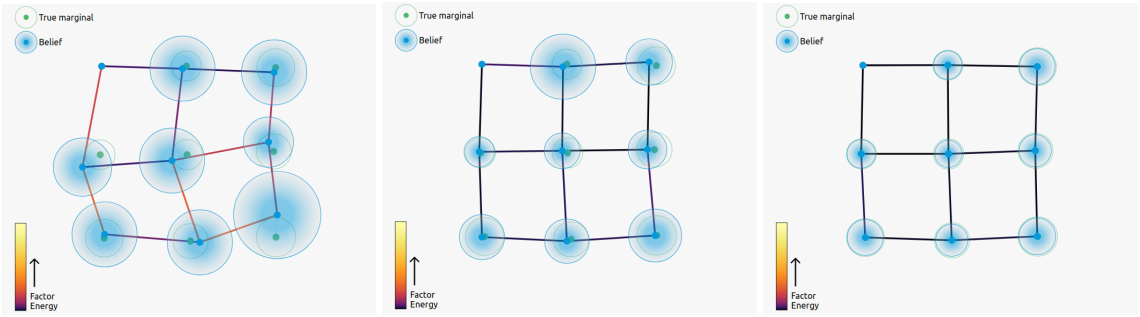
Figure 3.8: Synchronous schedule applied to the 2D incremental SLAM example. The code is available at: http://www.doc.ic.ac.uk/~ajd/bpslam.py and the robot is moved with the WASD keys. A moving 'robot', with history of ground truth poses shown as yellow dots, explores a scene containing landmarks (ground truth positions are white dots), and from each pose makes observations of nearby landmarks to incrementally build the factor graph shown in grey. Red and blue dots show the robot and landmark mean estimates obtained from GBP, which runs continually on the growing factor graph. We also superimpose the true marginal distributions in green for comparison. We see close agreement between the GBP and the true marginals until loop closure occurs, where the large correction needed takes a large number of GBP iterations to propagate fully around the graph. However, even very soon after loop closure we see that the relative information in the GBP estimate is good, with nearby nodes having estimates differing from the batch solution by similar amounts.



(a) Initialisation.



(b) Synchronous message passing.



(c) Random serial message passing.

Figure 3.9: Synchronous and serial random message passing on a grid graph pose optimisation problem. The numbers indicate how many steps of message passing have taken place. For the random serial message passing this is equal to the total number of messages sent. We strongly encourage the reader to view the full interactive version of this figure at: https://gaussianbp.github.io/#gbp_intuition.

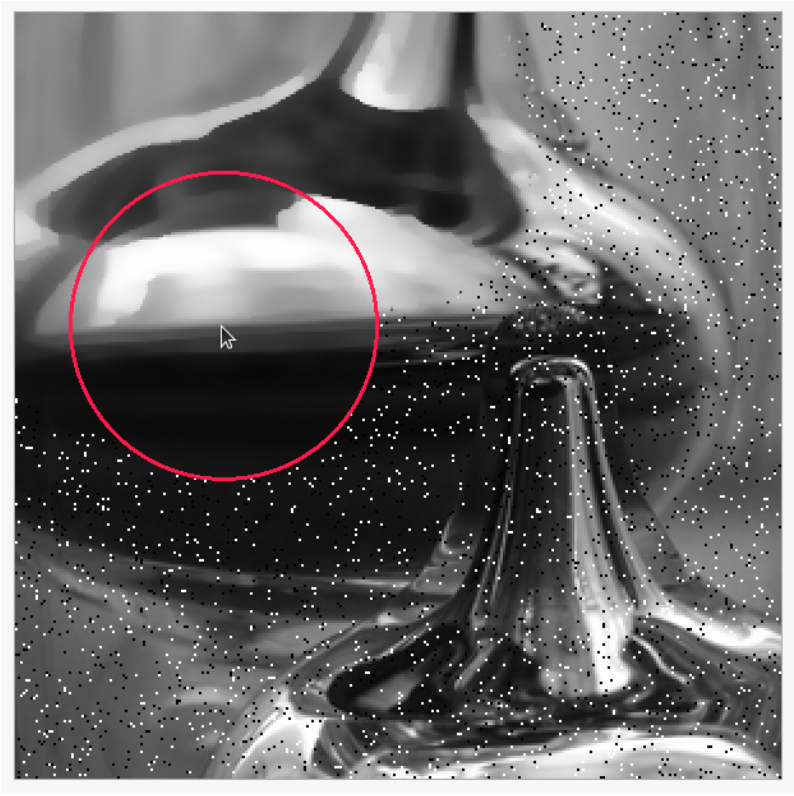


Figure 3.10: Attention based synchronous message passing in the red circle around the mouse for image denoising. The image has been corrupted by adding salt and pepper noise. See the full interactive version of this figure at: https://gaussianbp.github.io/#attentiongl.

An example of attention-driven message passing for image denoising is shown in Figure 3.10, which captures a screenshot from our interactive diagram at https://gaussianbp.github.io/#attentiongl. Note that there are no long-range connections in the image denoising graph, so local message passing can produce the true local marginals as the effect of more distant parts of the graph is negligible. In the interactive figure, the user can move their mouse to control the region of synchronous message passing and achieve local convergence in the graph.

For incremental problems, when some parallelism is available, a sensible approach similar to residual BP would be to send the most informative k messages at each step. Similar to this, individual nodes could decide only to come alive and pass messages when their incoming messages have changed by a significant enough amount, perhaps judged using information theoretic measures [Davison, 2005]. This is the idea behind the 'Wildfire' algorithm in Loopy SAM [Ranganathan et al., 2007]. Information theoretic measures will also be crucial in the longer term in deciding on the number of bits needs to specify the quantities used in message passing.

More speculatively, we envisage that attention based processing will be important in large real-time systems. Most obviously, at the current location where a robot is making sensor measure-

ments and must decide on action, 'attention' could actively be focused with a high rate of message passing, while other parts of the graph are partially or completely neglected, to be picked up and updated later on as needed 'just in time'. We imagine an attention spotlight which moves around the graph, bringing it to life. Depending on memory constraints, graph regions out of the current attention spotlight might even be much abstracted and simplified to low resolution approximated forms, maintaining only the main shape and connectivity, perhaps in an analogue of the way that a human brain remembers distant places. When a moving device revisits these places, they can easily feed on incoming data to be brought back to the high resolution needed for local action.

From a computational perspective, there may need to be a particular region of the processor held aside as the current active workspace, where enough precompiled space is retained such that the live part of the graph can be copied, unpacked and subject to full rate processing (the Real-Time Loop part of the 'Spatial AI Brain' shown in Figure 4 of [Davison, 2018]). Between the active workspace and the rest of the graph there will need to be some special graph infrastructure such as routing nodes to interface between live workspace and long-term graph memory.

3.2.3 Asynchronous Schedules

So far, we have assumed that all parallel schedules operate in a synchronous fashion, where all nodes compute and broadcast messages simultaneously in a given step. In fact, this is far from a requirement and GBP can operate with arbitrary and asynchronous message passing. As a consequence, GBP can readily be used in systems with no global clock and varying local compute budgets such as on neuromorphic hardware or between a group of distributed devices [Micusik and Evangelidis, 2020].

3.3 Robust Factors using M-Estimators

Although requiring all factors to be Gaussian is a convenient assumption, most interesting problems involve non-Gaussian factors. We already discussed in Section 2.5.2 how non-linear measurement functions yield non-Gaussian factors that can be iteratively restored to the Gaussian form via linearisation. In this section, we discuss a second cause of non-Gaussian factors: non-Gaussian data distributions.

Most practical estimation methods in computer vision and robotics take account of the fact that sensors, especially outward-facing ones like cameras, have a measurement probability distribution

which is not truly Gaussian. The classic behaviour is that the distribution is closely Gaussian when the sensor is essentially 'working', and reporting measurements that are close to ground truth apart from small variations due to quantisation and similar, but then some percentage of the time the sensor will report wildly incorrect 'garbage' measurements. For instance, if a camera is reporting the image location of matched image features, false correspondences happen sometimes and give measurements arbitrarily far away from ground truth. If we plot the measurement distribution of such a sensor we see a distribution which looks like a Gaussian centrally but is more 'heavy-tailed'. In optimisation and estimation, such behaviour is modelled using a family of 'robust' functions called M-Estimators.

One approach for incorporating these robust costs is to treat them as part of the non-linear measurement function and use the factor linearisation procedure in Section 2.5.2 to re-weight the cost. This approach is commonly used in nonlinear least squares optimisation for example in Ceres [Agarwal and Mierle, 2012]. Inspired by Agarwal *et al.* [Agarwal et al., 2012], we take a different approach based on scaling the covariance of the factor. Although both approaches are valid and we would expect to behave similarly, one key difference is that in regions where the gradient of the robust cost is zero, the outgoing messages from the robust factor has zero information when linearisation is used, while scaling the covariance never yields zero messages.

To the best of our knowledge, this is the first use of robust factors in GBP and the application of the covariance scaling theory to GBP is novel. We now detail the covariance scaling method for handling robust functions in GBP with completely local processing and then provide examples of its practical utility, demonstrating for the first time global outlier rejection using purely local GBP processing.

3.3.1 Implementation via Covariance Scaling

Consider the general factor definition:

$$f(\mathbf{x};\mathbf{z}) = Ke^{-\frac{1}{2}E(\mathbf{x};\mathbf{z})}\,, \tag{3.14}$$

where E is the factor's 'least squares' energy:

$$E(\mathbf{x};\mathbf{z}) = (\mathbf{z}-\mathbf{h}(\mathbf{x}))^\top \Sigma_n^{-1}(\mathbf{z}-\mathbf{h}(\mathbf{x}))\,. \tag{3.15}$$

The term:

$$M = \sqrt{(\mathbf{z}-\mathbf{h}(\mathbf{x}))^\top \Sigma_n^{-1}(\mathbf{z}-\mathbf{h}(\mathbf{x}))}\,, \tag{3.16}$$

is the unitless Mahalanobis distance, representing the number of standard deviations that the measurement is away from the mean of the distribution. So for a standard Gaussian constraint, the energy is simply the square of the Mahalanobis distance: $E = M^2$. In robust estimation, we modify the factor energy by setting a threshold level on M beyond which we change the energy to a function which rises less steeply than the standard quadratic form.

Let us first consider the commonly used Huber function [Huber, 1964, Huber, 1981] which transitions from quadratic to linear beyond a threshold $M \geq N_\sigma$. A factor with Huber loss has the following energy:

$$E(\mathbf{x}; \mathbf{z}) = \begin{cases} M^2 & M_s \leq N_\sigma \\ 2N_\sigma M_s - N_\sigma^2 & M_s \geq N_\sigma \end{cases}, \tag{3.17}$$

such that the two parts of the function match up in terms of both value and gradient at the discontinuity $M = N_\sigma$. The distribution induced by the Huber energy is a Gaussian distribution close to the mean and a Laplace distribution in the tails. The probability density function for the Laplace distribution is:

$$p(x; \mu, \beta) = \frac{1}{2b} \exp\left(\frac{-|x - \mu|}{b}\right). \tag{3.18}$$

Now, in GBP, every message takes the form of an information vector and precision matrix representing a Gaussian distribution. So what we do to represent the effect of the non-Gaussian part of a robust factor is to find the Gaussian distribution which has the same value energy, and pass a message with that precision instead. This is similar to the Dynamic Covariance Scaling method in [Agarwal et al., 2012]. We ask what Mahalanobis distance we must be from the mean in a standard quadratic energy to be equivalent to the Huber energy in the linear region. Specifically, we need to find M_R such that:

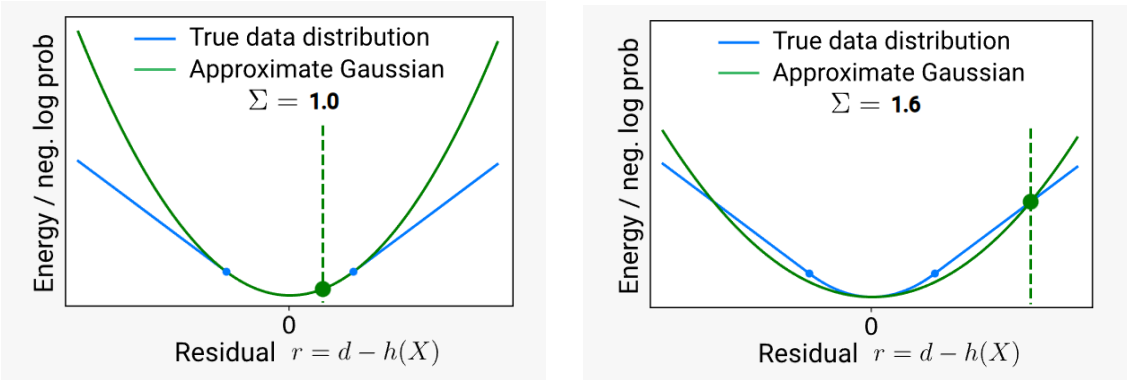
$$M_R^2 = 2N_\sigma M - N_\sigma^2. \tag{3.19}$$

Therefore, we can calculate the factor by which the energy of the constraint should be reduced:

$$k_R = \frac{M_R^2}{M^2} = \frac{2N_\sigma}{M} - \frac{N_\sigma^2}{M^2}. \tag{3.20}$$

Remembering the information form of Equation 2.24, we see that this is achieved by multiplying both the precision matrix and information vector by this factor. This process of scaling a Gaussian to approximate the Huber loss is shown in Figure 3.11.

To summarise, to use a Huber loss on a factor, every time that factor is to pass a message we first use *all* the latest incoming messages from variables in order to form its state vector $\mathbf{x}$. Then we evaluate the current Mahalanobis distance M using Equation 3.16. We test this against the



(a) When the residual is within the inlier region, the Gaussian parameters are scaled by a factor of 1 as the Huber loss is simply the original Gaussian within this operating region.

(b) When the residual is in the linear region, the Gaussian information parameters are scaled by a factor < 1, such that the Gaussian covariance increases to match the energy at this point.

Figure 3.11: The Huber loss function is shown with a blue line as a function of the residual and labelled the true data distribution. The threshold at which the Huber loss transitions is indicated with the blue dots on the curve. As the x axis is the residual rather than the Mahalanobis distance M, this transition happens at $r = \pm N_\sigma \Sigma_n$ where the residual is a scalar. The green dot shows the recorded value of the residual $r = z - h(x)$ (note that the measurement is denoted with d rather than z in the figure). The green curve shows the scaled quadratic energy or scaled Gaussian used to approximate the Huber loss at this pont. See the full interactive version of this figure at: https://gaussianbp.github.io/#scaling

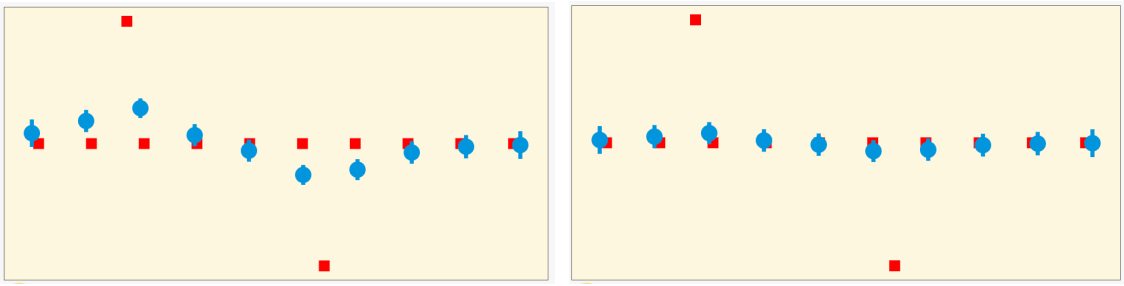
N_σ cutoff we have set for this factor (which might be 4.0 or something similar), representing the number of standard deviations from the mean for which we expect Gaussian behaviour. If $M \leq N_\sigma$ we are in the Gaussian zone and use the standard linearised precision matrix and information vector for the message calculation. If $M \geq N_\sigma$, we temporarily scale the linearised precision matrix and information vector by a factor k_R as calculated in Equation 3.20 for the purposes of this message pass only.

We can use the same method to handle other robust losses. For instance, the Tukey loss function which is Gaussian up to $M \leq N_\sigma$ and then constant beyond is implemented with a factor $k_R = \frac{N_\sigma^2}{M^2}$.

We now explore further the role of robust factors in 3 of our examples: 1D surface estimation, image denoising and incremental SLAM.

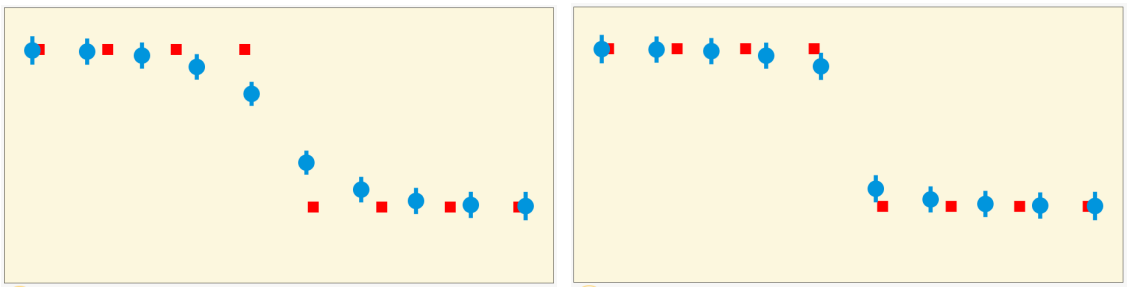
3.3.2 1D surface estimation with Robust Factors

The 1D surface estimation example (Section 3.1.1) is a simple domain in which the effect of the Huber loss function can be clearly observed. We first consider the problem of estimating the 1D surface given a set of measurements on a flat line plus 2 outlying measurements, as shown in Fig-



(a) Squared loss function. (b) Huber loss function.

Figure 3.12: 1D surface estimation with 2 outlying measurements. The goal is to estimate the height (y value) at 10 discrete x values given 12 surface measurements (red squares). There are 10 in-lying measurements on a line and 2 outlying measurements. The beliefs after convergence of GBP are shown in blue; the means are at the centre of the circles and the lines display ± 1 standard deviation of uncertainty in the belief. See the full interactive version of this figure at: https://gaussianbp.github.io/#gbp1d_robust in which the user can click to place their own measurement factors in the plane, run synchronous GBP and toggle the Huber loss on and off.



(a) Squared loss function. (b) Huber loss function.

Figure 3.13: 1D surface estimation of surface with a discrete step. The surface measurements are shown with red squares and the belief means and standard deviations after GBP convergence are shown with blue circles and lines. See the full interactive version of this figure at: https://gaussianbp.github.io/#gbp1d_robust

ure 3.12. When assuming a Gaussian data distribution in Figure 3.12a, the reconstructed points are strongly drawn towards the outlying measurements, as not fitting the line to these outlying points incurs a large energy cost. With the Huber loss in Figure 3.12b, GBP is able to reconstruct the line more accurately by reducing the error associated with the outlying measurements by moving them into the robust linear regime. The reason for this behaviour is that the Huber distribution fits the data distribution better, assuming that the measurements away from the line are outliers.

The second 1D surface estimation example has 8 measurements arranged in a step function, as shown in Figure 3.13. With a squared loss function / Gaussian data distribution in Figure 3.13a, the reconstructed points are smoothed out across this step discontinuity. In contrast, with a Huber loss in Figure 3.13b the discontinuity is somewhat preserved. With the Huber loss, the smoothness factor at the discontinuity is given a linear cost in the outlier regime, allowing the reconstruction

to better fit the discontinuity.

These two 1D examples illustrate that a Huber loss on the measurement factors and smoothness factors is crucial for effective outlier rejection and retaining discontinuities in the data. Of course, one should not conclude that it is generally better to use a Huber loss. However many estimation problems contain outlying measurements or make smoothness assumptions that broadly hold but are sometimes broken meaning that the Huber loss fits the true measurement distribution better than a Gaussian with exponentially small tails.

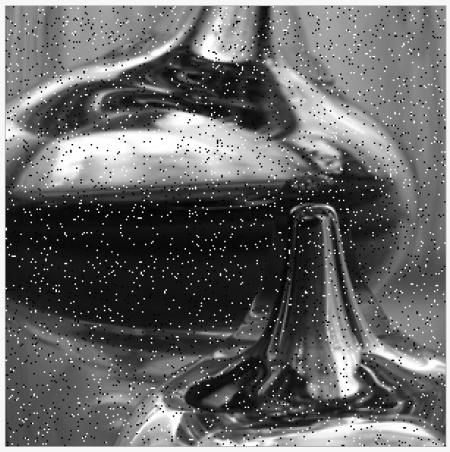
3.3.3 Image denoising with Robust Factors

We explained in Section 3.1.2 that image denoising can be thought of as the 2D generalisation of 1D surface estimation. Here we show that, like in the 1D case, a Huber loss function is crucial for practically useful image denoising that can filter out noise without over-smoothing the detail in the image. In Figure 3.14, we show the results of denoising an image corrupted with salt and pepper noise using GBP with and without the Huber loss function. With squared loss functions, we are unable to remove the salt and pepper noise from the image even with strong smoothness factors. In contrast, with the Huber loss we are able to remove the noise while retaining the image detail as shown in the case with weaker smoothness factors.

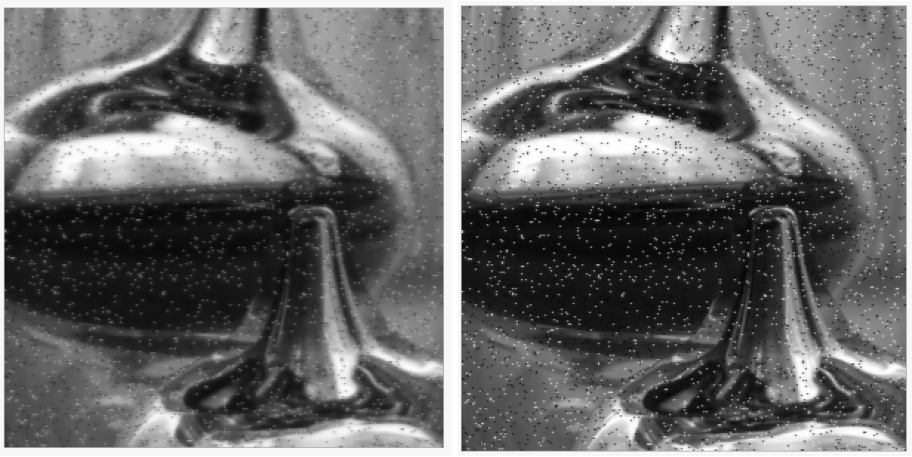
3.3.4 Incremental SLAM with Robust Factors

In Figure 3.15 we show the performance of GBP for incremental SLAM when a random $\frac{1}{50}$ of all measurements have a large error added, and all factors use a robust Huber kernel. This can be tried out as part of our Python simulation `http://www.doc.ic.ac.uk/~ajd/bpslam.py`, pressing 'r' to enter robust mode. GBP with robust factors has the impressive capability to detect outlier measurements in a local and lazy manner, with erroneous measurements which were not immediately apparent often determined much later when enough support builds up for a better hypothesis. Playing with the simulation is the best way to get a good feel for this.

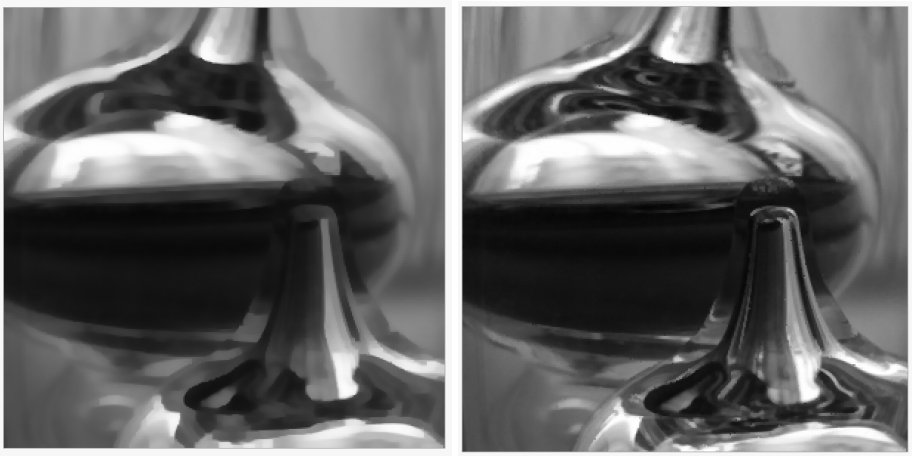
The example in Figure 3.15 demonstrates that robust factors allow lazy data association during GBP (reminiscent of [Olson and Agarwal, 2013]), where the robust status of factors can change dynamically during ongoing graph optimisation. This gives the ability to reject poor measurements immediately or after enough contradictory alternative data has been received. As we shall see in Chapter 5, this enables the incremental abstraction of point cloud data into planes via GBP,



(a) Original noisy image.

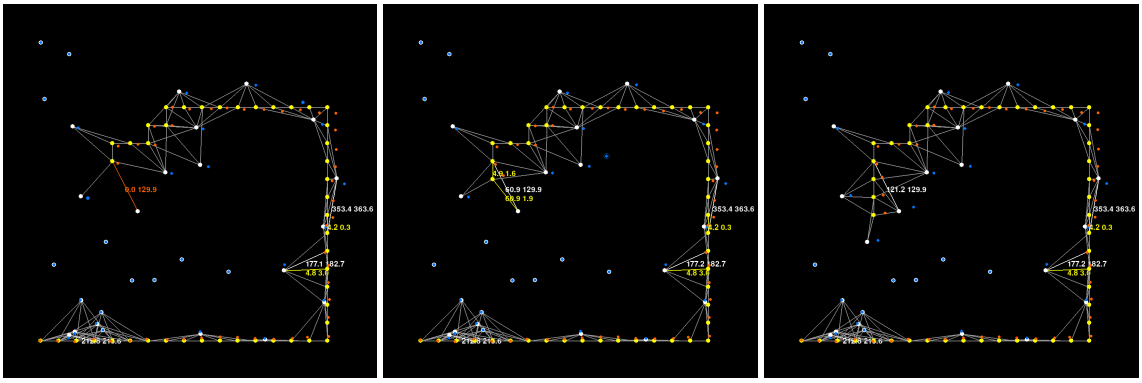


(b) Denoised with squared loss. *Left* stronger smoothness factors, *right* weaker smoothness factors.



(c) Denoised with Huber loss. *Left* stronger smoothness factors, *right* weaker smoothness factors.

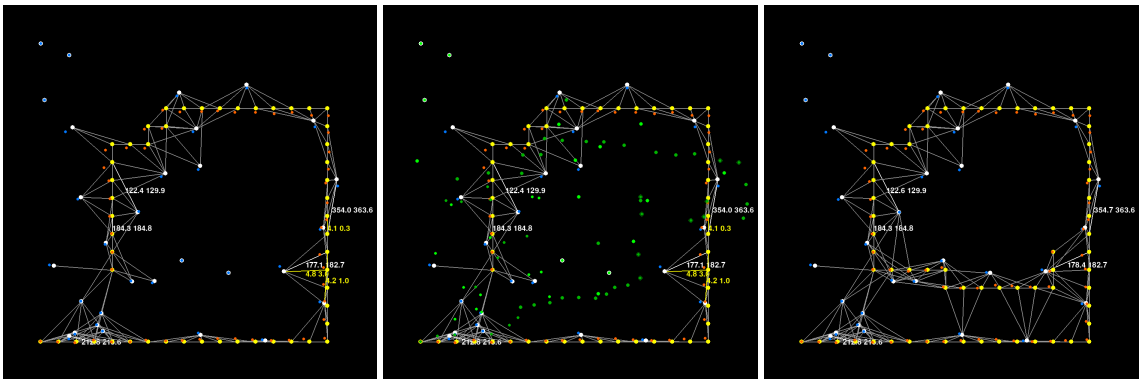
Figure 3.14: Image denoising with and without Huber loss function. The original image is corrupted with salt and pepper noise. The results show the denoised image after GBP has converged. See the full interactive version of this figure at: https://gaussianbp.github.io/#attentiongl.



New erroneous measurement (red)

Second measurement of landmark. White and yellow factors balance error

Further good measurements; error rejected. Erroneous factor is white; others grey



Loop closure

Comparison with non-robust batch solution (green)

After more measurements, 4 outliers confidently identified

Figure 3.15: 2D SLAM simulation with random erroneous measurements and robust factors. Every time a factor passes a message, it re-evaluates its Mahalonobis distance. In this simulation, we use a Huber loss with Mahalonobis distance threshold of 4.0 to transition from quadratic to linear cost. One in 50 ground truth measurements in the simulation has a large error added. In the visualisation, factors are colour-coded: **grey** if the Mahalonobis distances of both the ground truth measurement and the current factor estimate are both below 4.0 (normal case); **white** if both distances are above 4.0 (an erroneous measurement that BP has recognised as such and is treating with the linear part of the Huber function; **red** for a factor with high ground truth distance but low in the factor: this is an erroneous measure not yet recognised; and **yellow** for a factor with high factor distance but low ground truth distance; this is a good measurement being 'unfairly' treated as an outlier. All non-grey factors display their estimated and true Mahalonobis distances numerically. When a first erroneous measurement is made to initialise a new landmark, there are no contradictory good measurements to oppose it. In the second and third panels, additional measurements of the landmark support each other and push the erroneous measurement far into the linear Huber region. Good estimates are seen throughout the graph after loop closure, and a much better result than the green batch solution which does not take account of outliers (of course, we could have used robust methods there as well). More exploration and further loop closure happens by the last panel, and four erroneous measurements are now very confidently identified.

where the robust status of the proposed planar abstractions is determined on the fly [Ortiz et al., 2022b].

This example in Figure 3.15 shows how errors that slip through the front-end measurement part of a SLAM system (such as visual feature matching) could be cleaned up by back-end estimation. However, in the longer term, we are interested in how GBP and graph-based estimation could be used for the whole of a Spatial AI system, including front-end data association like the matching in a sparse or dense SLAM system, avoiding the need for ad-hoc algorithms such as RANSAC. For instance, in dense SLAM (such as the ICP tracking in KinectFusion [Newcombe et al., 2011]), each pixel measurement from a depth camera is associated with several possible locations in a dense model, and this association is refined iteratively through ICP. We could replace this process with GBP, where the measurement might be connected by several factors to different scene points, with mutually exclusive robust factors who would fight it out via GBP, in collaboration with other factors, until the most probable associations are reached. This multi-modal approach achieves a discrete model-selection capability and potentially could be efficiently implemented on a distributed close-to-sensor processor.

In practice, robust factors are employed for all real estimation problems using GBP to handle the presence of outlying measurements in real data. For example, they were crucial for solving bundle adjustment [Ortiz et al., 2020], multi-device co-localisation [Murai et al., 2022] and scene flow [Scona et al., 2022].

Another possible application of robust factors is towards the goal of learning abstractions in large structured generative models trained with GBP. Our interpretation is that here robust factors can play a similar role to non-linearities in neural networks, activating or deactivating constraints in the graph.

3.4 Improving Convergence

Although GBP empirically works well on loopy graphs, it is well known that loopy GBP does not always converge. Here we discuss strategies for improving the chances and speed of convergence for loopy graphs. This section contains many useful tricks for practitioners of GBP that were obtained qualitatively through many experiments on a variety of problems in vision and robotics. It is important to note that GBP is not reliant on any of the suggestions in this section for convergence, sensible application of these suggestions can however help accelerate convergence although the

success can be problem dependent. See chapter 22 in Murphy's textbook [Murphy, 2012] for a complementary discussion of methods to improving convergence.

Before discussing suggestions for improving convergence, we briefly highlight some key results in the limited literature around convergence guarantees in GBP. Note that these convergence guarantees are not original work from this thesis.

3.4.1 Convergence Guarantees

We discussed in the reproduced proof (in the appendix, Section 7.2) that on convergence, the belief means exactly compute the true marginal means for linear problems. Of course, this guarantee does not hold for general non-linear problems in which the true marginal distributions are not Gaussians. For non-linear problems, we know that the converged belief means are the true marginal means for the final linearised problem, but it is not guaranteed that this final linearisation will be at the true MAP solution.

Even for linear problems, although we have a guarantee for the correctness of the belief means on convergence, there is no guarantee that GBP will converge in general and in some extreme cases, GBP can exhibit oscillatory or divergent behaviour. There are however some specific cases in which loopy GBP is guaranteed to converge. We begin by noting that when marginalising a Gaussian distribution in the information form, the marginal variance does not depend on the information vector of the initial distribution while the marginal mean does depend on the information matrix of the initial distribution. Therefore, in GBP the variances are updated independently to the means, while the means are updated using the variances. This leads to distinct convergence properties for the belief means and belief covariances. In general, when the variables at each node are vector-valued, Du *et al.* [Du et al., 2017] showed that if we stack all information parameters of the initial factor-to-variable messages into a large matrix, then when this matrix is positive semi-definite the belief covariances always converge on the same values.

Once the variances have converged, Du *et al.* [Du et al., 2017] showed that we can form a linear system for the belief mean vectors:

$$\mathbf{B}^{t+1} = Q\,\mathbf{B}^{t}\,, \tag{3.21}$$

where $\mathbf{B}$ is a concatenation of the belief mean at all variable nodes and Q is a square matrix composed of row blocks. Each row block in Q corresponds to an edge in the graph connecting a variable node to a factor node. Q depends on the converged values of the marginal variances and so cannot be pre-computed in advance to determine the value of its spectral radius. This linear

system for $\mathbf{B}$ is convergent when the largest absolute value of the eigenvalues of Q (also known as the spectral radius) is less than 1: $\rho(Q) < 1$.

3.4.2 Message Schedules

As discussed in Section 3.2 there are various serial and synchronous message schedules for GBP. Although message schedules can affect convergence properties, we found little difference when applying more complex schedules such a residual BP [Elidan et al., 2006] in vision problems. As a result, we found a simple synchronous schedule to perform broadly well across all tasks and would recommend this schedule as a starting point to practitioners.

3.4.3 Multiscale Learning

Propagating information from one node to another with GBP takes the same number of iterations as the number of hops between the nodes. For nearby nodes in a local region, information can be communicated in a small number of iterations and consensus can be reached quickly, while for distant nodes, a global consensus can take many more iterations to be established. We observed this in Figure 3.8 of the previous section in which a loop closure correction took many iterations to be propagated through the graph. This is an inherent property of local algorithms and can be summarised as: low frequency errors decay more slowly than the high frequency errors.

Regular grid structured graphs appear a lot in computer vision (e.g. image processing) and in discretised boundary value problems (e.g. solving for the temperature profile along a rod). Accelerating convergence in such grid graphs has been well-studied in the field of Multigrid methods [Briggs et al., 2000]. The idea is to coarsen the grid which transforms low frequency errors into higher frequency errors that decay faster. After convergence in the coarsened grid, the solution is used to initialise inference in the original grid which now has smaller low frequency errors. This is the idea behind coarse-to-fine optimisation which is used in many grid-based problems where it is simple to build a coarser graph. In one notable work [Felzenszwalb and Huttenlocher, 2006], the authors demonstrate much faster inference for stereo, optical flow and image restoration with multiscale BP.

Mulitgrid methods can only be applied to graphs with a grid-like structure where it is possible to build equivalent coarsened representations. In general, most problems are more unstructured and it is not clear how to build a coarsened or abstracted representation of the original problem. In

the general case, we see two possible ways to build hierarchy into a model. A network could be trained to directly predict specific abstractions that form long range connections when included in the graph [Ortiz et al., 2022b]. Second, the graph could contain additional constraints that define a generic hierarchical structure (much like a neural network) and then the abstractions themselves are also inferred [George et al., 2017].

3.4.4 Relinearisation Schedules

Although all examples we have discussed in this chapter are linear, GBP is applied in exactly the same way for non-linear problems with the additional iterative relinearisation of the factors about the most recent belief means. For non-linear problems, the speed of convergence and ability to convergence can depend on the relinearisation strategy. One possible strategy is to delay relinearisation such that we iterate message passing until convergence, and then only after convergence relinearise all factors and repeat. This is broadly the approach of the Gauss-Newton method which solves the successive linear problems with direct factorisation based solvers. With this linearisation schedule it would therefore be possible to construct a Gauss-Newton solver using GBP as the linear solver.

Delayed relinearisation in GBP can converge very slowly as we must run GBP to convergence many times to solve a single non-linear problems. As a result, a more aggressive local relinearisation strategy is attractive because each factor is always using its current estimates to linearise. The most aggressive relinearisation strategy is to relinearise every factor on every step. When close to a the non-linear solution, this aggressive relinearisation strategy can accelerate convergence towards the solution, however when far from a solution in a highly non-linear landscape, it can lead to oscillations. Consequently, we find a good middle ground stratgey is to set a threshold on the difference between the current belief mean and the linearisation point, and then relineraise factors individually whenever this threshold is exceeded. This provides a good trade off between maintaining a good linear approximation, not wasting computation by relinearising excessively and preventing oscillations due to excessive relinearisation.

Dynamically changing the factors as we do with robust kernels does not seem to cause any instability in optimisation.

3.4.5 Linear System Damping

Inspired by Levenberg-Marquardt (LM) optimisation, which is a modification of Gauss-Newton that damps the linear system with an adaptable parameter λ, we have found empirically that applying a similar damping in GBP improves convergence. This linear system damping in GBP is novel and although we have not performed a thorough analysis, we found that it consistently improves convergence across a variety of vision problems. We first briefly review Newton's method, Gauss-Newton and Levenberg-Marquardt, before transferring the ideas to GBP.

Newton's method is an algorithm for finding minimum of a general objective:

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} \; S(\mathbf{x}) \; . \tag{3.22}$$

The algorithm iteratively takes steps using first and second order gradients to minimise the objective. Let the current estimate be $\mathbf{x}_0$, the second order Taylor expansion at a nearby point $\mathbf{x}$ about $\mathbf{x}_0$ is:

$$S(\mathbf{x}) \approx S(\mathbf{x}_0) + \frac{\partial S(\mathbf{x})}{\partial \mathbf{x}}\bigg|_{\mathbf{x}=\mathbf{x}_0} (\mathbf{x}-\mathbf{x}_0) + \frac{1}{2}(\mathbf{x}-\mathbf{x}_0)^\top \frac{\partial^2 S(\mathbf{x})}{\partial \mathbf{x}^2}\bigg|_{\mathbf{x}=\mathbf{x}_0} (\mathbf{x}-\mathbf{x}_0) \; . \tag{3.23}$$

We want to find a point $\mathbf{x}$ which is a critical point of the objective where:

$$\frac{\partial S(\mathbf{x})}{\partial \mathbf{x}} \approx \frac{\partial S(\mathbf{x})}{\partial \mathbf{x}}\bigg|_{\mathbf{x}=\mathbf{x}_0} + \frac{\partial^2 S(\mathbf{x})}{\partial \mathbf{x}^2}\bigg|_{\mathbf{x}=\mathbf{x}_0} (\mathbf{x}-\mathbf{x}_0) = 0 \; . \tag{3.24}$$

This is known as the normal equation.

Solving for $\mathbf{x}$, we see that the we should choose the next value as:

$$\mathbf{x} = \mathbf{x}_0 - \left(\frac{\partial^2 S(\mathbf{x})}{\partial \mathbf{x}^2}\bigg|_{\mathbf{x}=\mathbf{x}_0} \right)^{-1} \frac{\partial S(\mathbf{x})}{\partial \mathbf{x}}\bigg|_{\mathbf{x}=\mathbf{x}_0} \; . \tag{3.25}$$

This is the familiar Newton's method which uses first and second order derivatives of the objective to update the current estimate $\mathbf{x}_0$ to an improved estimate $\mathbf{x}$. Intuitively, Newton's method forms local quadratic approximations of the objective and moves directly to the minimum of the quadratic function. As a consequence, Newton's method jumps directly to the minimum in one step for quadratic objectives.

While Newton's method makes no assumptions about the form of the object, Gauss-Newton and Levenberg-Marquardt build on Newton's method for non-linear least squares objectives of the form:

$$S(\mathbf{x}) = \mathbf{r}(\mathbf{x})^\top \Lambda \mathbf{r}(\mathbf{x}) \; , \tag{3.26}$$

where $\mathbf{r}(\mathbf{x})$ is a non-linear function. Note that minimising a non-linear least squares objective is equivalent to doing MAP inference in a non-linear Gaussian graphical model.

For least squares objectives, we can derive the derivatives required for a Newton update step:

$$\frac{\partial S(\mathbf{x})}{\partial \mathbf{x}} = \mathtt{J}_{\mathbf{r}}(\mathbf{x})^{\top} \Lambda \mathbf{r}(\mathbf{x}) \tag{3.27}$$

$$\frac{\partial^2 S(\mathbf{x})}{\partial \mathbf{x}^2} = \mathtt{J}_{\mathbf{r}}(\mathbf{x})^{\top} \Lambda \mathtt{J}_{\mathbf{r}}(\mathbf{x}) + \mathtt{H}_{\mathbf{r}}(\mathbf{x}) \Lambda \mathbf{r}(\mathbf{x}) \ , \tag{3.28}$$

where $\mathtt{J}_{\mathbf{r}} = \frac{\partial \mathbf{r}(\mathbf{x})}{\partial \mathbf{x}}$ and $\mathtt{H}_{\mathbf{r}} = \frac{\partial^2 \mathbf{r}(\mathbf{x})}{\partial \mathbf{x}^2}$ are the Jacobian and Hessian of $\mathbf{r}(\mathbf{x})$.

Gauss-Newton makes the assumption that the Hessian of the objective function can be approximated using only the first term in Equation 3.28 as $\mathbf{r}$ is small:

$$\frac{\partial^2 S(\mathbf{x})}{\partial \mathbf{x}^2} \approx \mathtt{J}_{\mathbf{r}}(\mathbf{x})^{\top} \Lambda \mathtt{J}_{\mathbf{r}}(\mathbf{x}) \ . \tag{3.29}$$

Under this assumption the updated Gauss-Newton update step can be written as:

$$\mathbf{x} = \mathbf{x}_0 - \left(\mathtt{J}_{\mathbf{r}}(\mathbf{x}_0)^{\top} \Lambda \mathtt{J}_{\mathbf{r}}(\mathbf{x}_0) \right)^{-1} \mathtt{J}_{\mathbf{r}}(\mathbf{x}_0)^{\top} \Lambda \mathbf{r}(\mathbf{x}_0) \ . \tag{3.30}$$

Gauss-Newton has the same interpretation as Newton's method of forming a quadratic or Gaussian approximation of the problem and then moving to the minimum of the resulting problem by solving the normal equation (Equation 3.30) for the update: $\mathbf{x} - \mathbf{x}_0$.

Levenberg-Marquardt modifies the Gauss-Newton update by adding a damping factor to the approximate Hessian of the objective:

$$\mathbf{x} = \mathbf{x}_0 - \left(\mathtt{J}_{\mathbf{r}}(\mathbf{x}_0)^{\top} \Lambda \mathtt{J}_{\mathbf{r}}(\mathbf{x}_0) + \lambda I \right)^{-1} \mathtt{J}_{\mathbf{r}}(\mathbf{x}_0)^{\top} \Lambda \mathbf{r}(\mathbf{x}_0) \ . \tag{3.31}$$

For very small λ, the LM update approaches the Gauss-Newton update while for very large λ the update approaches gradient descent with learning rate λ^{-1}:

$$\mathbf{x} - \mathbf{x}_0 \approx -\lambda^{-1} \ \mathtt{J}_{\mathbf{r}}(\mathbf{x}_0)^{\top} \Lambda \mathbf{r}(\mathbf{x}_0) = -\lambda^{-1} \ \frac{\partial S(\mathbf{x})}{\partial \mathbf{x}} \bigg|_{\mathbf{x}=\mathbf{x}_0} \ . \tag{3.32}$$

To control convergent behaviour, the value of λ is typically modified following a simple schedule with two parameters $0 < a < b$. Typical values are $a = 2$ and $b = 10$. If after computing the new value $\mathbf{x}$, the value of the residual has decreased i.e. $S(\mathbf{x}) < S(\mathbf{x}_0)$, then the update is accepted and λ is decreased by a factor $\frac{1}{a}$. If the residual increases, then we do not accept the update and instead increase λ by a factor b and compute a new update. LM therefore interpolates between a Gauss-Newton update and gradient descent step based on changes in the residual in previous steps. The motivation is that gradient descent can avoid Gauss-Newton getting stuck in local minima and then as we approach the minimum, $\lambda \to 0$ and the updates tend towards Gauss-Newton steps. This procedure makes sense because convergence close to a minima is faster with Gauss-Newton as any function is approximately quadratic about its minimum.

The damping is applied to the approximated Hessian of the objective which is also the information matrix of the system. Linking this to Gaussian factor graphs, LM is damping the precision matrix of the joint distribution representing the whole problem. This is equivalent to adding an extra factor that places a prior around the current estimate with covariance $\frac{1}{\lambda}$.

We can now apply these same ideas locally to Gaussian belief propagation. Instead of damping the linear system globally, we can apply this same damping to each individual factor. This damping is applied whenever the factor is relinearised and each factor maintains its own λ parameter that is updated based on changes in the local energy rather than the global energy. This simple modification leads to increase stability and improved convergence for large scale bundle adjustment problems.

3.4.6 Message Damping

Message damping is a technique commonly used to speed up and improve chances of convergence in very loopy graphs. The idea behind message damping is to use momentum to reduce chances of oscillation and accelerate progress towards a minimum in ravine like energy landscapes. Message damping has been shown to both empirically [Murphy et al., 1999] and theoretically [Malioutov et al., 2006, Su and Wu, 2015] improves convergence without affecting the fixed points of GBP. Although there are several variations of message damping that have been proposed in the literature [Murphy et al., 1999, Malioutov et al., 2006, Su and Wu, 2015], here we discuss log space message damping which we found worked best for our problems [Ortiz et al., 2020].

Damping operates by replacing the message at time t with a combination of the message at time t and time $t-1$:

$$\tilde{\mu}_t = \mu_t^{\beta}\, \tilde{\mu}_{t-1}^{(1-\beta)} \,, \tag{3.33}$$

which is a weighted sum in log-space:

$$\log \tilde{\mu}_t = \beta\, \log \mu_t + (1-\beta)\, \log \tilde{\mu}_{t-1} \,. \tag{3.34}$$

For GBP, message damping in log-space corresponds to damping the information vector and precision matrix as a weighted sum:

$$\tilde{\eta}_t = \beta\, \eta_t + (1-\beta)\, \tilde{\eta}_{t-1} \tag{3.35}$$

$$\tilde{\Lambda}_t = \beta\, \Lambda_t + (1-\beta)\, \tilde{\Lambda}_{t-1} \,. \tag{3.36}$$

Standard BP is recovered when the damping parameter: $\beta = 1$. $\beta = 0$ corresponds to not updating the messages and sending the message from the previous iteration which leaves the graph unchanged. Message damping can be applied to both the variable-to-factor messages and factor-to-variable messages, however we find that only applying it to factor-to-variable messages is more effective. Message dropout can be thought of as a form of message damping where $\beta = 1$ for nodes that are sending messages and $\beta = 0$ for other nodes. Lastly, message damping has proven crucial in the successful application of GBP to medium/large scale bundle adjustment [Ortiz et al., 2020] and SLAM problems [Ortiz et al., 2022b].

3.5 Robot Web Protocol

Here we briefly detail the Robot Web communication protocol, proposed in our paper [Murai et al., 2022] for many-device localisation. By proposing and demonstrating GBP as a strong framework for decentralised inference [Ortiz et al., 2020, Davison and Ortiz, 2019, Ortiz et al., 2022b], my work laid the foundation for using GBP for many device localisation which was lead by the first author Riku Murai. Throughout this project, I assisted with both implementation and the development of the Robot Wel Protocol for distributing the factor graph amongst the devices.

As discussed in the introduction, an important systems focus will be the definition of interfaces which allow multiple devices/robots to pass messages between each other. Individual devices will have their own individual estimation algorithms, sensors and hardware, but could use GBP as the general 'glue language' to share probabilistic information and come to agreement over global estimation matters. For example, a swarm of robotic devices could organise themselves into a regular grid via only local computation and communication.

What will be important to achieve such capabilities will be standards for inter-operation and messages which can be deployed for communication. Inspired by the creation of the World Wide Web [Berners-Lee, 1999], we propose the Robot Web communication protocol for GBP which is specifically focused on many-device localisation. In our solution, each robot stores and maintains its own part of the full factor graph and updates and publishes a Robot Web Page of outgoing message for other robots to download and read when possible. Robots communicate via ad-hoc asynchronous messages and robots can join or leave the web at any time. The Robot Web is designed simply for scalability; all communication is via a simple interface and without the need for any privileged information about other robots or even how many are involved.

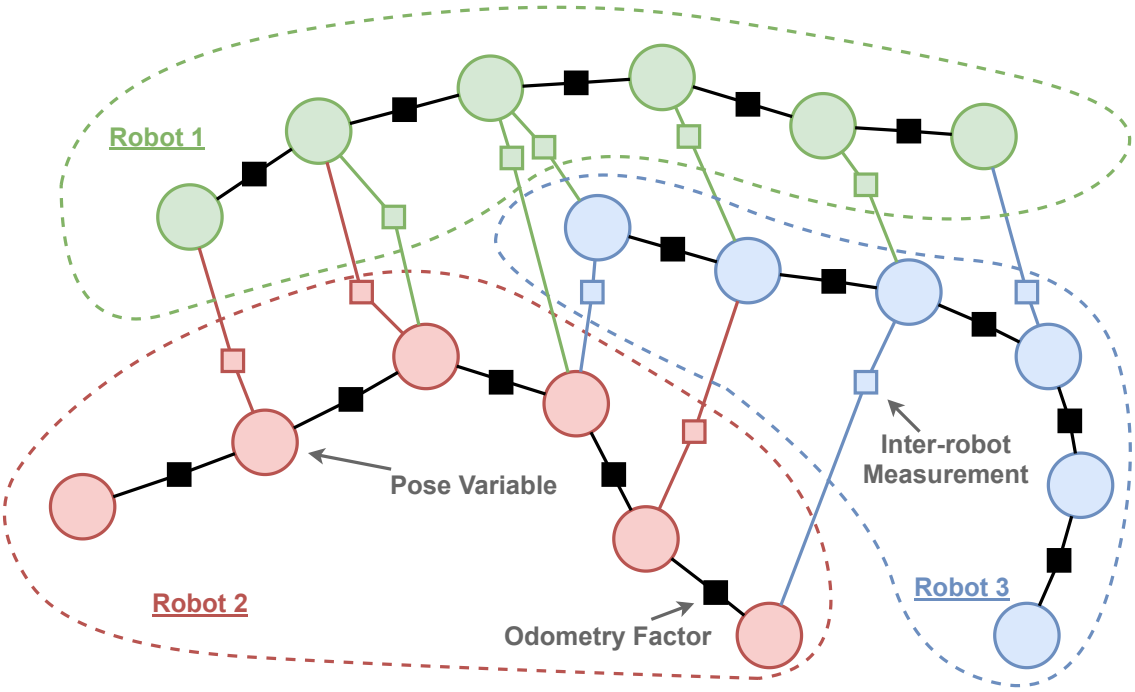


Figure 3.16: In the Robot Web, we assume that a set of robots move through a space while using their sensors to observe each other. The circles represent the variables, and the squares are the factors. Responsibility for storing and updating the full factor graph is divided up between the multiple robots participating, as shown by the coloured regions separated by dotted lines. Each robot maintains its own pose variable nodes, odometry factors, and factors for the inter-robot measurements made by its sensors, and carries out continuous GBP on this graph fragment. Message passing across dotted line boundaries is via Robot Web Pages published and updated by each robot, and happens on an asynchronous and ad-hoc basis.

The key challenge is to distribute responsibility for the full factor graph by dividing it up amongst the robots taking part. An important design choice is that factors representing inter-robot measurements are stored on the robot making the measurement. This is sensible because the details of the measurement factors depend on the type and calibration of the sensor involved and in this way those details only need to be known by the robot carrying the sensor. We also assume for now that all robots have globally synchronised clocks for timestamping measurements.

In the Robot Web paper [Murai et al., 2022], we experiment with the Robot Web protocol and GBP with non-Euclidean variables (presented in Section 2.6) for the problem of distributed multi-device localisation. Figure 3.16 shows a factor graph for distributed multi-device localisation in which the graph is divided between the distributed robots. We refer the interested reader to the full paper for further details which are beyond the scope of this thesis.

CHAPTER 4

Bundle Adjustment on a Graph Processor

Contents

In this chapter, we describe work from our paper *Bundle Adjustment on a Graph Processor* [Ortiz et al., 2020] that was presented at the Conference on Computer Vision and Pattern Recognition (CVPR) in 2020.

Graph processors such as Graphcore's Intelligence Processing Unit (IPU) are part of the major new wave of novel computer architecture for AI, and have a general design with massively paral-

lel computation, distributed on-chip memory and very high inter-core communication bandwidth which allows breakthrough performance for message passing algorithms on arbitrary graphs.

We show for the first time that the classical computer vision problem of bundle adjustment (BA) can be solved extremely fast on a graph processor using Gaussian belief propagation. Our simple but fully parallel implementation uses the 1216 cores on a single IPU chip to, for instance, solve a real BA problem with 125 keyframes and 1919 points in under 40ms, compared to 1450ms for the Ceres CPU library. Further code optimisation will surely increase this difference on static problems, but we argue that the real promise of graph processing is for flexible in-place optimisation of general, dynamically changing factor graphs representing Spatial AI problems. We give indications of this with experiments showing the ability of GBP to efficiently solve incremental SLAM problems, and deal with robust cost functions and different types of factors.

A video presentation of the method and results are available at: https://www.youtube.com/watch?v=TqeN8aQNgd0. We also provide code for our Python implementation at: https://github.com/joeaortiz/gbp and our Poplar implementation at: https://github.com/joeaortiz/gbp-poplar.

4.1 Introduction

Real-world applications which require a general real-time Spatial AI capability from computer vision are becoming more prevalent in areas such as robotics, UAVs and AR headsets, but it is clear that a large gap still exists between the ideal performance required and what can be delivered within the constraints of real embodied products, such as low power usage. An increasingly important direction is the design of processor and sensor hardware specifically for vision and AI workloads to replace the general purpose CPUs, GPUs and frame-based video cameras which are currently prevalent [Davison, 2018, Nardi et al., 2015]. The space of AI and vision algorithm design continues to change rapidly and we believe that it is not the right time to make very specific decisions such as 'baking in' a particular SLAM algorithm to processor hardware, except perhaps for very specific use cases.

However, new architectures are emerging which have made quite general design choices about processing for AI workloads. Efficient and low power computation must be massively parallel and minimise data transfer. To this end, storage and processing should be distributed, and as much computation as possible should happen 'in place'. A key example is Graphcore's Intelligence

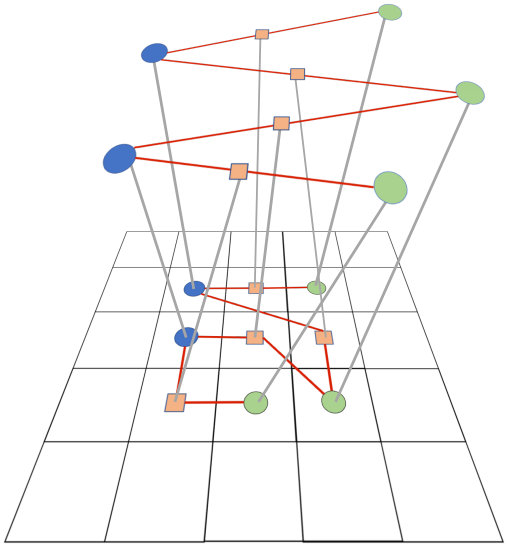


Figure 4.1: We map a bundle adjustment factor graph onto the tiles (cores) of Graphcore's IPU and show that Gaussian belief propagation can be used for rapid, distributed, in-place inference for large problems. Here we display the most simple mapping in which each node in the factor graph is mapped onto a single arbitrary tile. Keyframe nodes are blue, landmark nodes are green and measurement factor nodes are orange.

Processing Unit (IPU) [Graphcore, 2022], which implements this concept within a single large chip which is composed of 1216 cores called *tiles*, each with local memory arranged in a fully connected graph structure. It is massively parallel like a GPU, but its tiles have a completely different interconnect structure. The IPU has breakthrough performance for algorithms which have a sparse graph message passing character. The key early commercial use case for the IPU is as a flexible deep learning accelerator [Lacey, 2019], primarily in the cloud, but we believe that it has much more general potential for Spatial AI computation.

In this work we consider bundle adjustment (BA), a central element of 3D visual processing which is representative of many geometric estimation problems, and show that Gaussian belief propagation can perform rapid optimisation of BA problems on a single IPU chip.

GBP is a special case of general loopy belief propagation, a well known technique in probabilistic estimation, but it has previously only been minimally used in geometric vision and robotics problems [Davison and Ortiz, 2019]. It is an algorithm which can be run on a CPU, but is not necessarily competitive there compared to alternative optimisation techniques which take global account of the structure of a problem. However, GBP can be mapped to a graph processor due to its fully distributed nature to take full advantage of the massively parallel capability of an IPU.

We present the first implementation of BA on a graph processor, with breakthrough optimisation speed for a variety of diverse sequences in which we record an average speed advantage 24x over

the Ceres library on a CPU. Our implementation is simple and preliminary, implemented with only 1000 lines of Poplar™C++ code, and there is surely much room for future performance optimisation.

Positive characteristics of our GBP approach include: extremely fast local convergence, the ability to use robust cost functions to reject outlying measurements, and the ability to easily deal with dynamic addition of variables and data and rapidly re-optimise solutions. We highlight these aspects in our results, and argue as in [Davison and Ortiz, 2019] for the huge potential for graph processing and GBP in general incremental factor graph optimisation for Spatial AI. It would be straightforward and efficient to incorporate factors from additional priors and sensors into this framework, such as smoothness of scene regions due to recognition, and continue to optimise for global estimates with all computation and storage done in-place on a graph processor.

4.2 Related Work

Factor graphs are commonly used in geometric vision to represent the structure of constraints in estimation problems [Bloesch et al., 2018, Engel et al., 2017, Folkesson and Christensen, 2004, Kaess et al., 2008, Lu and Milios, 1997, Mur-Artal et al., 2015]. In particular, for bundle adjustment [Triggs et al., 1999] researchers have leveraged the global structure of these constraints to design efficient inference algorithms [Agarwal et al., 2009, Jeong et al., 2010].

Several works have taken the approach of converting the loopy factor graph into a tree [Kaess et al., 2012, Paskin, 2003]. iSAM2 [Kaess et al., 2012] uses variable elimination to convert the loopy factor graph to a Bayes tree while [Paskin, 2003] uses a junction tree-like method which employs maximum likelihood projections to remove edges. This category of methods differs from our approach in that it requires periodic centralised computation to convert the loopy constraint graph into a tree.

More closely related to our work, [Crandall et al., 2011] and [Ranganathan et al., 2007] use Loopy Belief Propagation for geometric estimation problems, though with CPU implementation. [Crandall et al., 2011] uses discrete BP to provide an initialisation for Levenberg-Marquardt refinement in BA, and Loopy SAM [Ranganathan et al., 2007] uses GBP to solve a SLAM-like problem for a relatively small 2D scene.

In the domain of computer architecture, there has been substantial recent effort to design specific hardware for vision algorithms [Saeedi et al., 2018, Zhang et al., 2017b]. This is particularly

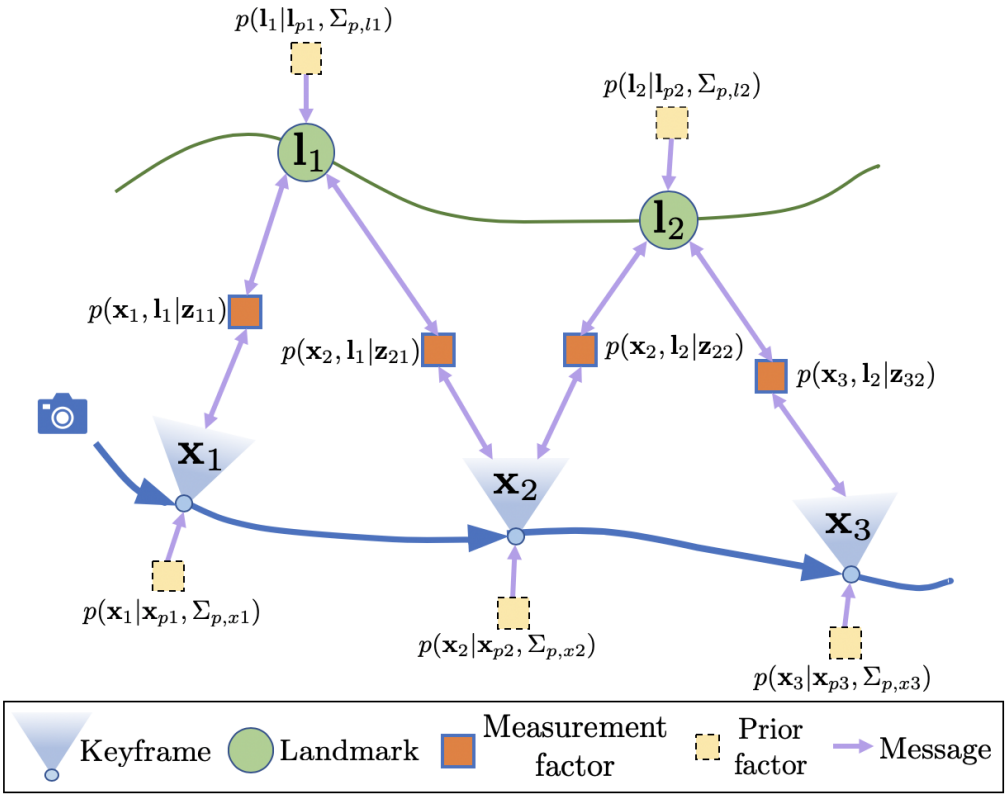


Figure 4.2: **Bundle adjustment factor graph.** Measurement factors connect keyframes and the landmarks they observe. Keyframes and landmarks are instantiated with an automatically generated weak prior factor. Messages are sent from all factors to adjacent keyframe and landmark nodes and from keyframe and landmark nodes to adjacent measurement factor nodes.

evident in industry, where we have seen development of chips such as the HoloLens' HPU and the Movidius VPU series, though the main accelerations achieved to date have been in vision front-ends such as feature matching.

Other related research has made use of parallelism on existing hardware to accelerate BA. Multicore BA [Wu et al., 2011] proposed an inexact but parallelisable implementation for CPUs or GPUs, while [Gupta et al., 2010] advocated a hybrid GPU and CPU implementation. More generally, [DeVito et al., 2017] accelerated non-linear least squares problems in graphics by automatically generating GPU solvers.

4.3 The Bundle Adjustment Factor Graph

Bundle adjustment is the problem of jointly refining the set of variables $V = X \cup L$, where $X = \{\mathbf{x}_i\}_{i=1:N_k}$ is the set of keyframe poses and $L = \{\mathbf{l}_j\}_{j=1:N_l}$ is the set of landmark locations, subject to a set of constraints which define the error we want to minimise. Specifically, we include

two types of error terms: reprojection errors and prior errors. The reprojection error penalises the distances between the projections of landmarks into the image plane of the keyframes that observe them and the set of measurements corresponding to these observations $Z = \{\mathbf{z}_{km}\}$. The prior error terms try to maximise the probability that the current variable values were drawn from the corresponding prior distribution $\{\mathcal{N}(\mathbf{x}_i; \mathbf{x}_{p_i}, \Sigma_{p,xi}),\ \mathcal{N}(\mathbf{l}_j; \mathbf{l}_{p_j}, \Sigma_{p,lj})\}_{i=1:N_k, j=1:N_l}$. The prior terms are required to set the overall scale for monocular problems and to condition the messages from the measurement factors which would otherwise only constrain 2 degrees of freedom. Given an initialisation point, the priors are automatically generated such that they are a factor of 100 weaker than the reprojection error terms in the objective. We formulate this using the Jacobians and the measurement model which define the strength of measurement constraints. An example factor graph for a small BA problem is shown in Figure 4.2.

In bundle adjustment we want to perform maximum a posteriori (MAP) inference which computes the configuration of variables $\{X, L\}$ that maximises the joint probability $p(X, L|Z)$:

$$\begin{aligned} \{X^*, L^*\} &= \arg\max_{\{X,L\}} p(X, L|Z) && (4.1) \\ &= \arg\max_{\{X,L\}} p(Z|X, L)p(X, L)\ . && (4.2) \end{aligned}$$

In the second line we have used Bayes theorem and dropped the denominator $p(Z)$ as measurements are given quantities and do not affect the MAP solution. This leads to the factorisation of the probability distribution that we want to maximise (which we will call $p_{\text{obj}}(X, L)$) into the product of the likelihood of the measurements given the variables $p(Z|X, L)$ and priors on the variables $p(X, L)$. As $\mathbf{x}_i$ and $\mathbf{x}_j$ are independent in our formulation, $\mathbf{l}_i$ and $\mathbf{l}_j$ are independent and $\mathbf{x}_i$ and $\mathbf{l}_j$ are only conditionally dependent given a measurement $\mathbf{z}_{ij}$, these terms can be further factorised:

$$p_{\text{obj}}(X, L) = \prod_{i=1}^{N_k} \phi_i(\mathbf{x}_i) \prod_{j=1}^{N_l} \theta_j(\mathbf{l}_j) \prod_{k=1}^{N_k} \prod_{m, \mathbf{l}_m \in L_k} \psi_{km}(\mathbf{x}_k, \mathbf{l}_m)\ , \tag{4.3}$$

where L_k is the set of landmarks observed by keyframe $\mathbf{x}_k$.

The set of factors $\{\phi_i, \theta_j, \psi_{km}\}_{i=1:N_k, j=1:N_l, km \in O}$ can be interpreted as prior constraints on the keyframe poses, prior constraints on the landmark positions and measurement reprojection constraints respectively. The prior constraints have the form of Gaussians over the variables $\{\mathbf{x}_i\}_{i=1:N_k}$ and $\{\mathbf{l}_j\}_{j=1:N_l}$:

$$\begin{aligned} \phi_i(\mathbf{x}_i) &= p(\mathbf{x}_i|\mathbf{x}_{p_i}, \Sigma_{p,xi}) && (4.4) \\ &\propto \exp\left(-\frac{1}{2} \parallel \mathbf{x}_i - \mathbf{x}_{p,i} \parallel^2_{\Sigma_{p,xi}}\right), && (4.5) \end{aligned}$$

$$\theta_j(\mathbf{l}_j) = p(\mathbf{l}_j|\mathbf{l}_{p_j}, \Sigma_{p,lj}) \tag{4.6}$$

$$\propto \exp\left(-\frac{1}{2} \parallel \mathbf{l}_j - \mathbf{l}_{p,j} \parallel^2_{\Sigma_{p,lj}}\right). \tag{4.7}$$

Assuming a Gaussian measurement model, $\mathbf{z}_{km} = \mathbf{h}(\mathbf{x}_k, \mathbf{l}_m) + \eta$, with $\eta \sim \mathcal{N}(0, \Sigma_M)$ we can write out the form of the measurement factors:

$$\psi_{km}(\mathbf{x}_k, \mathbf{l}_m) = p(\mathbf{x}_k, \mathbf{l}_m|\mathbf{z}_{km}) \propto p(\mathbf{z}_{km}|\mathbf{x}_k, \mathbf{l}_m) \tag{4.8}$$

$$\propto \exp\left(-\frac{1}{2} \parallel \mathbf{z}_{km} - \mathbf{h}(\mathbf{x}_k, \mathbf{l}_m) \parallel^2_{\Sigma_M}\right). \tag{4.9}$$

The measurement factor ψ_{km} is Gaussian in $\mathbf{z}_{km}$ but is Gaussian in the variables $\mathbf{x}_k$ and $\mathbf{l}_m$ only if the measurement function $\mathbf{h}(\mathbf{x}_k, \mathbf{l}_m)$ is linear. In our case, we have a nonlinear measurement function, $\mathbf{h}(\mathbf{x}_k, \mathbf{l}_m) = \pi\left(R_k\,\mathbf{l}_m + \mathbf{t}_k\right)$, where π is the projection operator and R_k and $\mathbf{t}_k$ are the rotations and translations derived from $\mathbf{x}_k$. As a result, we must update the measurement factors by relinearising during optimisation.

After linearising about some fixed point $(\mathbf{x}_{k,0}, \mathbf{l}_{m,0})$, the measurement factors can be expressed as a Gaussian distribution using the information form which is parametrised by an information vector $\boldsymbol{\eta}$ and information matrix Λ:

$$\mathcal{N}^{-1}(\mathbf{x}; \boldsymbol{\eta}, \Lambda) \propto \exp\left(-\frac{1}{2}\mathbf{x}^\top \Lambda \mathbf{x} + \boldsymbol{\eta}^\top \mathbf{x}\right). \tag{4.10}$$

The information form is used as it can represent distributions with rank deficient covariances in which a variable is not constrained at all along a particular direction. With this at hand and after a small amount of work, we find that linearised measurement factors take the following form:

$$\psi_{km}(\mathbf{x}_k, \mathbf{l}_m) = \mathcal{N}^{-1}\left(\begin{bmatrix}\mathbf{x}_k \\ \mathbf{l}_m\end{bmatrix}; \boldsymbol{\eta}_{km}, \Lambda_{km}\right), \tag{4.11}$$

with information form parameters:

$$\boldsymbol{\eta}_{km} = \mathtt{J}^\top \Sigma_M^{-1}\left(\mathtt{J}\begin{bmatrix}\mathbf{x}_{k,0} \\ \mathbf{l}_{m,0}\end{bmatrix} + \mathbf{z}_{km} - \mathbf{h}(\mathbf{x}_{k,0}, \mathbf{l}_{m,0})\right), \tag{4.12}$$

$$\Lambda_{km} = \mathtt{J}^\top \Sigma_M^{-1} \mathtt{J}\,, \tag{4.13}$$

where $\mathtt{J} = \left[\frac{\partial \mathbf{h}}{\partial \mathbf{x}_k}, \frac{\partial \mathbf{h}}{\partial \mathbf{l}_m}\right]\Big|_{\mathbf{x}_k = \mathbf{x}_{k,0}, \mathbf{l}_m = \mathbf{l}_{m,0}}$ is the 2×9 Jacobian matrix.

Now that all of our constraints are in the Gaussian form, finding the MAP solution is equivalent to minimising the negative log likelihood which is a sum of squared residuals:

$$\{X^*, L^*\} = \underset{\{X,L\}}{\arg\min} \left[\sum_{i=1}^{N_k} \| \mathbf{x}_i - \mathbf{x}_{p,i} \|^2_{\Sigma_{p,xi}} + \sum_{j=1}^{N_l} \| \mathbf{l}_j - \mathbf{l}_{p,j} \|^2_{\Sigma_{p,lj}} + \sum_{k=1}^{N_k} \sum_{m,\mathbf{l}_m \in L_k} \| \mathbf{z}_{km} - \mathbf{h}(\mathbf{x}_k, \mathbf{l}_m) \|^2_{\Sigma_M} \right]. \tag{4.14}$$

4.4 Gaussian Belief Propagation for Bundle Adjustment

GBP can be used to solve bundle adjustment problems by computing the marginal distribution, with mean equal to the MAP solution, for all variables. In contrast, classical bundle adjustment methods compute a point estimate of the MAP solution using the Levenberg-Marquardt algorithm.

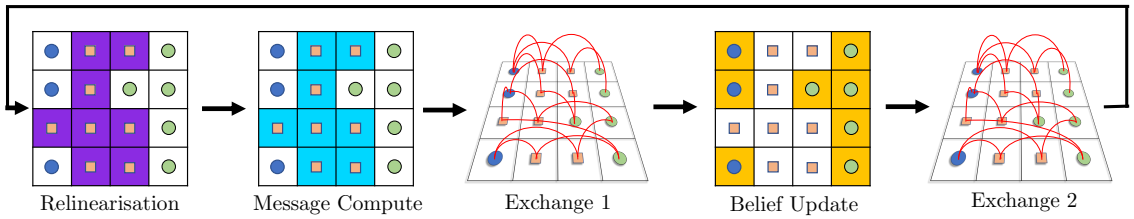
As the bundle adjustment factor graph is loopy, GBP stores a belief distribution at each variable node which converges to the marginal distribution after sufficient iterations of message passing. Prior factors send the same message to the variable node they connect to at all iterations. The beliefs are sent as messages from the variable nodes to the factor nodes as the true message can be recovered at the factor node using the previous factor to variable message.

We use a synchronous scheduling (see Section 3.2.2) in which, at each iteration, all factor nodes relinearise and send messages to adjacent variable nodes before all variable nodes update their belief and send back messages to adjacent factor nodes. Relinearisation is done in an entirely local manner and a measurement factor is relinearised when the distance between the current belief estimate and the linearisation point of the variables the factor connects to is greater than a threshold β. We use message damping (described in Section 3.4.6) which is commonly used to stabilise the convergence of Loopy GBP [Malioutov et al., 2006]. Lastly, we employ robust factors (described in Section 3.3) which are crucial in bundle adjustment to handle outlying measurements.

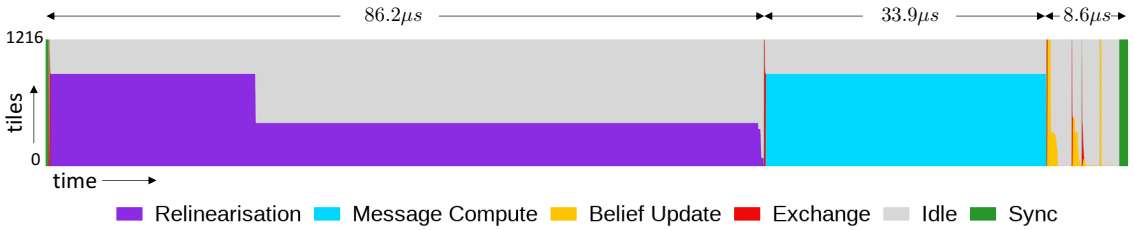
4.5 Implementation Details

4.5.1 IPU Implementation

An IPU chip is massively parallel and is composed of 1216 independent compute cores called *tiles*. Each tile has 256KB local memory and 6 hardware threads that can all execute independ-



(a) A schematic showing the compute on 16 tiles in a single iteration of GBP. Tiles are coloured when they are in a compute phase. In Exchange 1, factor nodes send messages to variable nodes and in Exchange 2 variable nodes send messages to factor nodes. Keyframe and landmark variable nodes are blue and green respectively and factor nodes are orange.



(b) Plot shows the activity of each tile during a single iteration of GBP for a factor graph with 1216 nodes mapped 1-to-1 onto the tiles. In the Relinearisation phase, all 929 factors compute the distance of the adjacent beliefs from their linearisation point and a subset of these factors subsequently relinearise. The Belief Update is implemented with Graphcore's Poplibs™library and so is significantly faster and is indicative of the speed-ups possible with a more specific implementation using an optimised linear algebra library.

Figure 4.3: IPU Phases.

ent programs. In contrast, a GPU has very limited cache on chip, all data must be fetched from off chip DRAM, and there is less flexibility for executing different programs on each thread. The IPU's distributed on-chip SRAM means that memory accesses consume approximately 1pJ per byte whereas external DRAM accesses on a GPU/CPU consume hundreds of pJ per byte. Embedded variants of the IPU will therefore have significant power advantages over existing processors [Graphcore, 2022].

To implement GBP on the IPU we must map each node in the factor graph onto a tile on the IPU. The tiles are connected all-to-all with similar latency between all pairs of tiles on a chip [Jia et al., 2019], meaning that nodes can be mapped to arbitrary tiles. The most simple mapping places exactly one factor or variable node per tile, as in Figure 4.1, but limits the size of the factor graph to 1216 nodes. Noting that variable and factor nodes alternate in compute and that there are 6 threads per tile, in all experiments we are able to map much larger graphs to a single chip by placing multiple nodes per tile without affecting speed.

In order to exploit this parallelism the IPU employs a *bulk synchronous parallel* execution model. In this model all tiles compute in parallel using their local memories. When each tile has finished

computing it enters a waiting phase (idle). When all tiles are finished, there is a short synchronisation phase (sync) across all tiles before data is copied between tiles with extremely high bandwidth in a predetermined schedule (exchange). This process then repeats as all tiles re-enter the compute phase. The period between syncs is not fixed but determined by the time taken for the computation.

GBP has three compute phases and two exchange phases in a single iteration. As shown in Figure 4.3a, factor nodes first relinearise and then compute their messages which are sent to adjacent variable nodes before the variable nodes update their beliefs which are sent back to adjacent factor nodes. Figure 4.3b shows that the total time for a single iteration of GBP is less than $125\mu s$ while factor relinearisation and message compute makes up the bulk of the total compute time.

4.5.2 GBP Implementation

In experiments, we set the relinearisation threshold $\beta = 0.01$ and allow a factor to relinearise at most every 10 iterations. The damping is set to $d = 0.4$ and messages from factors are undamped for 8 iterations after relinearisation. This damping schedule allows newly relinearised messages to propagate through the graph while also stabilising later iterations. As the IPU handles halves and floats but not doubles, we found that it was necessary for numerical stability to use the Jacobians to automatically set prior constraints to initially have the same scale as the measurement constraints. These priors are then weakened to a hundredth of the strength gradually over 10 iterations. GBP is not sensitive to the mean of the prior and displays the same behaviour on convergence as when implemented on a CPU with doubles when the stronger priors are not required.

4.6 Experimental Evaluation

For evaluation we use sections of sequences from the TUM [Sturm et al., 2012] and KITTI [Geiger et al., 2012] datasets. We use ORBSLAM [Mur-Artal et al., 2015] as the front-end to select keyframes, generate ORB features [Rublee et al., 2011] and handle correspondence. In all TUM experiments, landmarks are initialised at a depth of 1m from the keyframe which first observes them, while in KITTI experiments we initialise landmarks with Gaussian noise of standard deviation 0.5m.

We compare our implementation of GBP to Ceres [Agarwal and Mierle, 2012], a non-linear least squares optimisation library often used for bundle adjustment. In all comparisons Ceres is run on a 6 core i7-8700K CPU with 18 threads (which we found experimentally to maximise performance)

Table 4.1: The final two columns give the time in milliseconds to converge to ARE < 1.5 pixels for 10 sequences from the TUM data set (two testing sequences, 4 handheld camera sequences, 2 robot mounted sequences, 2 object reconstruction sequences) and 2 from the KITTI data set. k is the number of keyframes, p landmarks, m measurements.

Sequence	k	p	m	GBP	Ceres
fr1xyz	42	2194	12908	**37.2**	1180
fr1rpy	34	1999	8920	**130.3**	1030
fr1desk	63	2913	13514	**77.3**	2850
fr1room	20	1467	5388	**31.7**	779
fr2desk	40	892	3995	**20.8**	425
fr3loh	36	1140	5065	**44.6**	470
fr2robot360	40	333	1745	**51.5**	212
fr2robot2	20	567	4036	**8.6**	345
fr1plant	40	1824	6818	**31.8**	1450
fr3teddy	125	1919	9032	**40.0**	1450
KITTI00	30	2745	16304	**14.2**	342
KITTI08	30	3053	10480	**14.8**	394

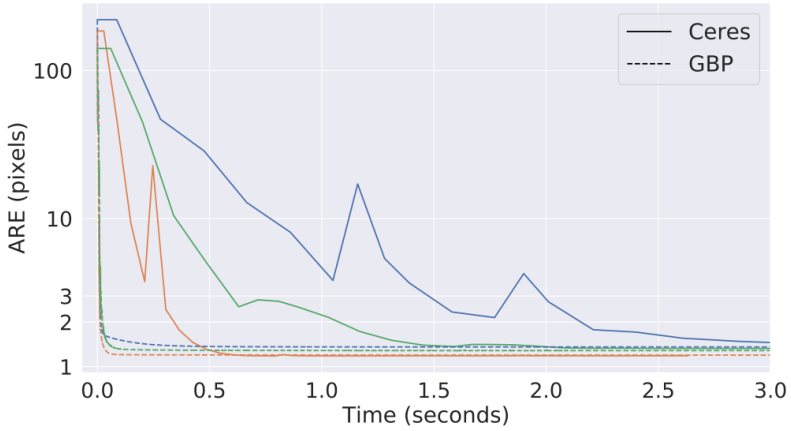
and uses Levenberg-Marquardt with Dense Schur and dense Cholesky on the reduced system, a Huber kernel and analytic derivatives.

Qualitative video results that show the refinement of points and cameras during bundle adjustment are available at: https://www.youtube.com/watch?v=TqeN8aQNgd0. We observe the typical property of local inference methods that local high frequency errors decay rapidly while it takes a large number of iterations for global low frequency errors to decay and produce a globally aligned reconstruction.

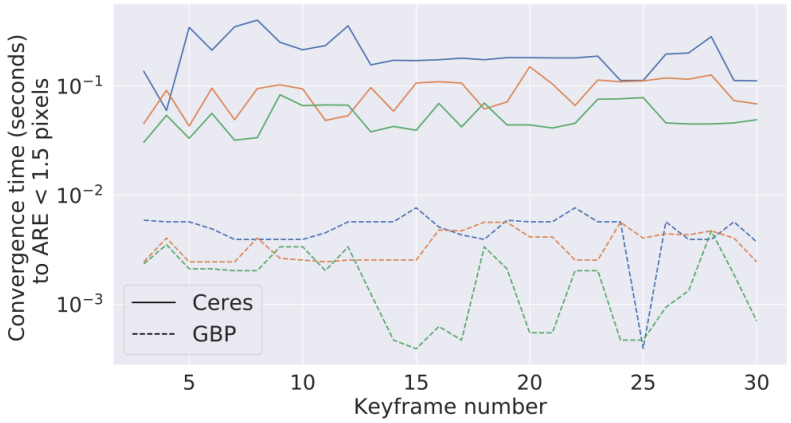
4.6.1 Bundle Adjustment Speed Evaluation

First we present results to show that our implementation of GBP can rapidly solve large bundle adjustment problems.

We evaluate the optimisation speed by tracking the average reprojection error (ARE) over all measurements in the graph. Table 4.1 shows the time to converge to ARE < 1.5 pixels for 10 sequences with diverse camera motion and co-observation of landmarks in which keyframe positions are initialised with Gaussian noise of standard deviation 7cm. The corresponding ARE curves for 3 of the sequences are plotted in Figure 4.4a. GBP reaches convergence an average of 24x faster than Ceres over the 10 sequences. Typically GBP takes between 50-300 iterations to converge and Ceres takes between 10-40 steps, however, due to the rapid in-place computation on the IPU, which operates at 120W, GBP is significantly faster.



(a) **Bundle adjustment.** ARE for 3 sequences fr1desk, fr2desk, fr3teddy. fr1desk is more difficult as it has the most measurements and the camera moves a large distance. fr3teddy has 125 keyframes but is easier to solve as fewer landmarks are densely observed in object reconstruction. Similar results were observed for the other TUM sequences whose convergence times are described in Table 4.1.



(b) **SLAM.** Time to converge to ARE $<$ 1.5 pixels after a new keyframe is added and initialised with the pose of the most recent keyframe. Results are for the first 30 keyframes of the sequences fr1desk, fr2desk, fr3teddy.

Figure 4.4: Speed comparison between GBP and Ceres. Note the logarithmic scale on the y axes.

4.6.2 SLAM Speed Evaluation

In GBP, the confidence in the belief estimations grows over iterations as the beliefs tend towards the marginal distributions. This Bayesian property is an inherent advantage over batch methods that make point estimates in the SLAM setting. For GBP, new variables are quickly snapped into a state that is consistent with the current estimates given the new constraints, while for batch methods, the full solution must be recomputed to refine just a few variables.

We go towards validating this advantage in incremental SLAM by comparing the time taken to converge to ARE $<$ 1.5 pixels after each new keyframe is added for 3 TUM sequences with 30 keyframes. New keyframes are initialised at the location of the most recent keyframe and new

landmarks at a depth of 1m. To aid Ceres and mimic the Bayesian approach, we fix the landmarks for the first 3 steps of Levenberg-Marquardt optimisation. Results are shown in Figure 4.4b for which on average, over the 90 keyframes added, GBP converges 36x faster than Ceres, often in fewer than 10 iterations.

4.6.3 Global vs Local Convergence Evaluation

As a local algorithm GBP exhibits fast local convergence followed by slower global convergence. This is because if two nodes are separated by k edges in the graph, it takes k iterations of GBP message passing for information to propagate between these two nodes. This behaviour is best observed in the video results accompanying this paper at https://www.youtube.com/watch?v=TqeN8aQNgd0, where the point cloud reconstruction rapidly becomes locally consistent while global structure is adjusted slowly over many iterations.

We quantitatively evaluate the difference between the local and global convergence speeds in Figure 4.5 by plotting both the average reprojection error (a local metric) and the camera loss (a global metric) for 3 synthetic medium sized bundle adjustment problems. The problems are generated with different random seeds and all have 40 cameras and 1000 points, making them of a similar size to the real sequences used in other evaluations. We observe that Levenberg-Marquardt, the global optimisation algorithm used by Ceres, is able to converge to a low reprojection error and camera loss in 10s of iterations. On the other hand, being a local algorithm, GBP can quickly converge to a low ARE, however it can take over 1000 iterations to convergence on accurate global camera estimates. This is because it can take hundreds of iterations for information to propagate from one side of the graph to another. This analysis suggests that if we want accurate global solutions rapidly with GBP, graph abstraction to produce more small world graphs will be important.

4.6.4 Robustness Evaluation

We compare the robustness of GBP and Ceres in solving BA problems by varying the noise added to the keyframe initialisation and counting the proportion of successful convergences over 100 trials at each noise level. Figure 4.6 shows that GBP has a comparable convergence radius to Ceres for these two TUM sequences.

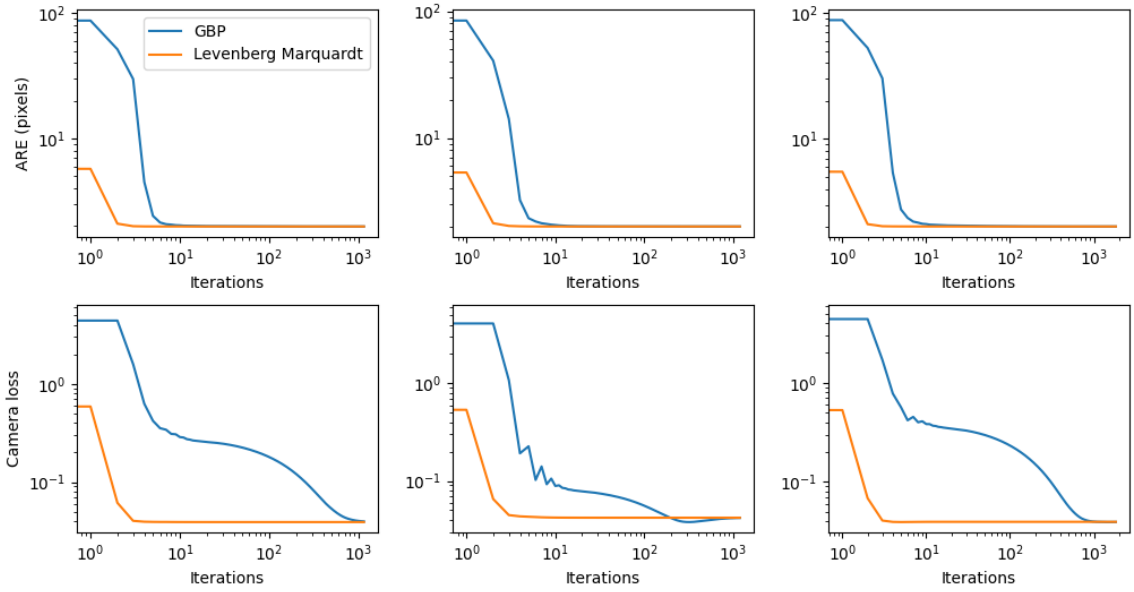


Figure 4.5: Local and global loss metrics comparing convergence of GBP (a local algorithm) and Levenberg-Marquardt (a global algorithm). The local metric is the average reprojection error (ARE), while the global metric is the camera loss between the estimated and ground truth camera poses. The 3 synthetic bundle adjustment problems have 40 cameras and 1000 points and are simulated with realistic noisy observations, initialisations and point track lengths. Note the log scale on both axes, and the x axis here is iterations rather than time.

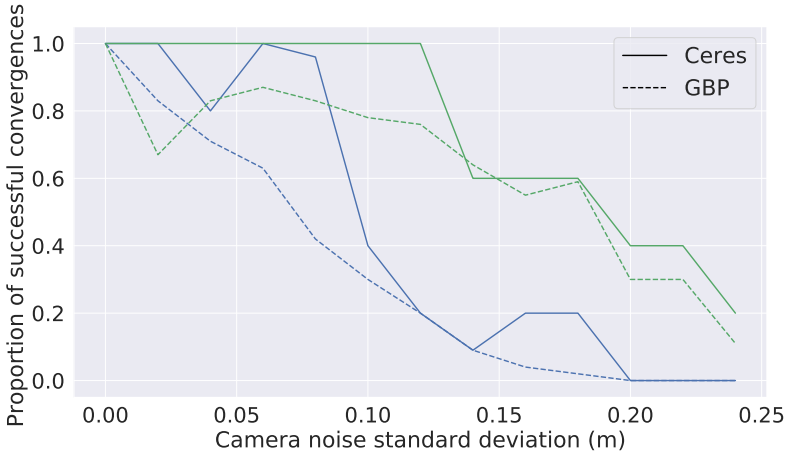


Figure 4.6: **Convergence basin comparison.** Proportion of successful convergences over 100 trials for different noise levels with the fr1desk and fr3teddy TUM 30-keyframe sequences. A successful convergence constitutes reaching ARE < 1.5 pixels.

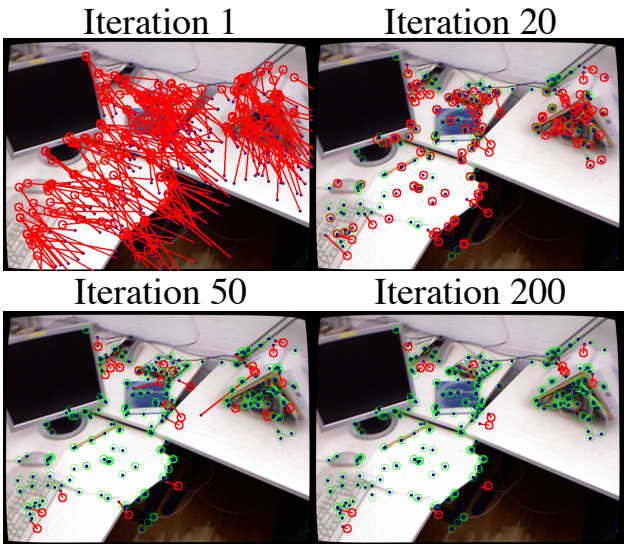
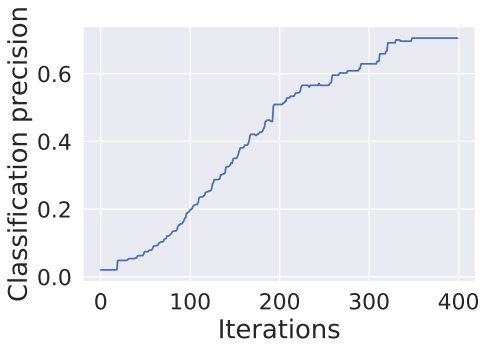


Figure 4.7: **GBP with Huber loss.** Landmark projections (blue points) and measurements (circles) are connected by lines. The lines and circles are red when the reprojection error exceeds the Huber threshold and the down-weighting of the message is proportional to the length of the red line.

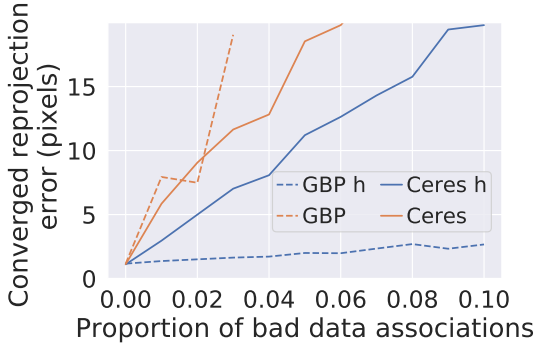
4.6.5 Huber Loss Evaluation

The Huber loss function has the effect of down-weighting messages from factors that may contain outlying measurements. We demonstrate this effect in Figure 4.7 in which we visualise the reprojection errors at iterations 1, 20, 50 and 200 of GBP in a chosen keyframe for which 10% of measurements are artificially added outliers. All measurements begin in the outlier regime and after 20 iterations a large proportion of the measurements remain in this regime as GBP has not yet worked out which measurements are inliers. By iteration 200, only the erroneous measurements are in the outlier regime as GBP has determined that these measurements are least consistent with other constraints in the graph. This behaviour of gradually removing false positive outlier classifications can be observed in Figure 4.8a, for a sequence in which 3% of data associations are incorrect.

To validate quantitatively the benefits of the Huber loss with both GBP and Ceres, we conduct an ablation study on a sequence with incorrect data associations and measure the converged reprojection error. Figure 4.8b shows that for GBP, the Huber loss is necessary and effective in handling incorrect data associations. For Ceres however, the same Huber loss is unable to identify the outliers and Ceres cannot arrive at a low ARE solution. This indicates that GBP's local consideration of outliers may be more effective than the global consideration in LM.



(a) Measurements are classified as outliers if they are in the linear loss regime. The recall is 1 over all iterations. ARE converges to < 1.5 pixels after 268 iterations while the precision is still increasing.



(b) h indicates Huber loss is used. For GBP, convergence is not reached without a Huber loss for more than 3% bad associations, while with a Huber loss GBP can down-weight the outliers and solve the bundle adjustment problem. For Ceres, the Huber loss improves the final ARE however it still cannot converge the solution.

Figure 4.8: Results for a 20 keyframe sequence from fr1desk in which bad data associations are artificially added.

4.7 Discussion / Conclusion

We have shown that with the emergence of new flexible computer architecture for AI, specifically Graph Processors like Graphcore's IPU, Gaussian belief propagation can be a flexible and efficient framework for inference in Spatial AI problems. By mapping the bundle adjustment factor graph onto the tiles of a single IPU, we demonstrated that GBP can rapidly solve a variety of bundle adjustment problems with a 24x speed advantage over Ceres. Additionally, we gave an indication of the framework's capacity to efficiently solve incremental SLAM problems and be robust to outlying measurements.

In the near term, we would like to apply GBP to very large bundle adjustment problems. Our framework scales arbitrarily to multiple chips, and Graphcore provide a custom interconnect for highly efficient inter-IPU message passing. An even more interesting direction which looks towards low power embedded Spatial AI would investigate how to fit large problems on a single chip by merging or replacing factors using a combination of network priors and marginalisation. We hope that our framework of flexible, in-place optimisation on a dynamically changing factor graph will be applied to a broad spectrum of AI tasks incorporating heterogeneous factors.

CHAPTER 5

Incremental Abstraction in Distributed Probabilistic SLAM Graphs

Contents

In this chapter, we describe work from our paper *Incremental Abstraction in Distributed Probabilistic SLAM Graphs* [Ortiz et al., 2022b] that was presented at the International Conference on Robotics and Automation (ICRA) in 2022.

Scene graphs represent the key components of a scene in a compact and semantically rich way, but are difficult to build during incremental SLAM operation because of the challenges of robustly identifying abstract scene elements and optimising continually changing, complex graphs. We present a distributed, graph-based SLAM framework for incrementally building scene graphs based on two novel components.

First, we propose an incremental abstraction framework in which a neural network proposes abstract scene elements that are incorporated into the factor graph of a feature-based monocular SLAM system. Scene elements are confirmed or rejected through optimisation and incrementally replace the points yielding a more dense, semantic and compact representation. Second, enabled by our novel routing procedure, we use Gaussian belief propagation for distributed inference on a graph processor. The time per iteration of GBP is structure-agnostic and we demonstrate the speed advantages over direct methods for inference of heterogeneous factor graphs. We run our system on real indoor datasets using planar abstractions and recover the major planes with significant compression.

A video presentation of our method and some qualitative results are available at: https://www.youtube.com/watch?v=ZoJ9ylb4Ss8.

5.1 Introduction

Abstract scene graphs of environments represent the key structures, objects and interactions in a semantically rich and compact way. Ideally, an intelligent embodied device should build a scene graph rapidly and on-the-fly in a new environment using on-board sensing and processing to enable immediate intelligent action. Identifying high-level abstractions is challenging and can require an expensive search-and-test over both the type of abstraction and the subset of elements to which it applies. Pre-trained neural networks can amortise this cost by directly proposing candidate abstractions and most algorithms for scene graph construction operate by either post-processing a low-level representation (e.g. semantic labelling of an occupancy volume [McCormac et al., 2017]) or by committing to abstractions of the low-level data at measurement time (e.g. detecting object instances [Salas-Moreno et al., 2013, Sucar et al., 2020]).

A more ambitious target is general *incremental abstraction*. Where abstract scene elements can be identified from single observations, they should be immediately added to a scene representation. More commonly, several observations may be needed to identify abstractions with high

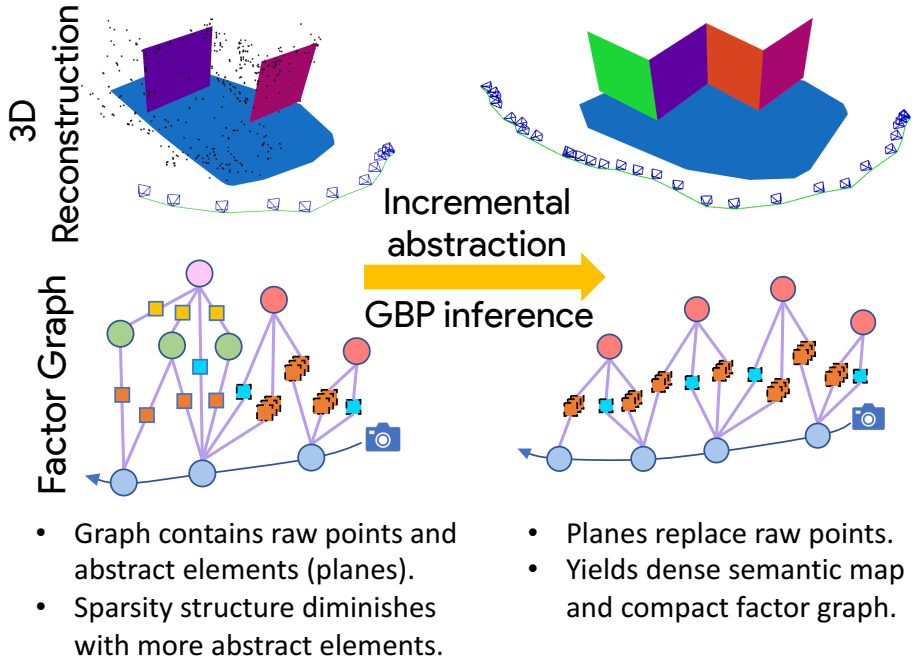


Figure 5.1: **Method Overview**. The time per iteration of Gaussian belief propagation is structure-agnostic so it can rapidly optimise factor graphs with little sparsity structure that appear in incremental scene abstraction. In the graph, green nodes are points and red nodes are planes. See the key in Figure 5.2 for the remaining nodes.

confidence, requiring the system to temporarily store low-level information (e.g. raw geometry as point clouds). As exploration continues, abstraction should operate continually and hierarchically on both stored and incoming observations, gradually replacing the raw elements in the map. A scene graph should therefore at any point in time be a hybrid mix of raw and abstract elements, potentially of many kinds.

The correct way to accumulate many different measurements and priors into coherent estimates is via probabilistic inference, and a factor graph represents the probabilistic structure of inference problems in SLAM [Dellaert and Kaess, 2017]. Abstract scene elements can be combined into this estimation framework and probabilistic inference can refine and confirm or reject the abstractions. A hybrid, incrementally abstracting map is therefore represented by a complicated, heterogeneous and dynamically changing factor graph, where new raw structure is continually added while abstractions are tested, and replace raw structure if those tests are passed.

There have been few attempts to solve the true, complex inference problems these factor graphs represent in real-time systems. Most SLAM systems that go beyond sparse point cloud processing make severe approximations to the true inference problem by artificially layering estimation [McCormac et al., 2017], baking in specific variable orderings [Zhou et al., 2020a], or by using alternation to avoid joint estimation [Newcombe et al., 2011]. We believe that these choices are often

related to the rigidity of existing optimisation algorithms that need to exploit the fixed structure of an optimisation problem to achieve efficient performance on standard processing hardware.

Gaussian belief propagation has recently been proposed as a strong candidate algorithm for real-time inference of arbitrary and dynamically changing factor graphs [Davison and Ortiz, 2019, Ortiz et al., 2021]. Its computational structure is node-wise parallel and it operates by local message passing on a factor graph. GBP trades the optimality of global updates for more flexible distribution of compute and memory, meaning it can better exploit parallel hardware and operate without assumptions about the global structure of an estimation problem. When implemented on a graph processor [Graphcore, 2022], GBP has already been shown to have speed advantages over global methods for inference on static bundle adjustment graphs [Ortiz et al., 2020].

In this work, we present a general method for incrementally constructing scene graphs in real-time based on two novel components:

1. An incremental abstraction framework that robustly identifies abstract scene elements and compresses the factor graph.

2. Distributed optimisation of dynamic heterogeneous graphs via GBP with dynamic routing.

First, our incremental abstraction framework combines amortised inference from an off-the-shelf network with probabilistic inference to robustly identify abstract scene elements. Additionally, upon accepting a scene element we linearise and join factors connecting common nodes to yield a more semantic, dense and compressed representation.

Second, enabled by our novel routing procedure, we are the first to use GBP on a graph processor for inference of dynamic heterogeneous factor graphs. The time per iteration of GBP is independent of the graph structure and we demonstrate the resulting advantages over direct methods.

While our framework is general and can be used with any abstract feature detector, we experiment with planar abstractions, which provide a compact way to densely represent geometry in many human-made environments. In evaluations, our framework reconstructs accurate planar scene graphs of real indoor environments at different levels of granularity, demonstrates significant graph compression and improves tracking compared to ablated baselines.

5.2 Related work

For long-term SLAM operation it is vital to manage computational cost by limiting factor graph growth to match the spatial extent of the map. Towards this goal, [Johannsson et al., 2013] re-uses existing keyframes for new measurements and [Ila et al., 2009] uses filtering to add only non-redundant nodes and edges. Alternatively, compression by node removal [Carlevaris-Bianco et al., 2014, Mazuran et al., 2016], attempts to recover the best non-linear and sparse factor graph that approximates the marginalised distribution. These methods target compression with minimal information loss, while we focus on semantic-guided compression.

Recent SLAM research has investigated incremental scene reconstruction with semantic elements included in the factor graph. Object-centric SLAM methods use object recognition and correspondence in the front-end to build scene graphs of objects [Salas-Moreno et al., 2013, Sucar et al., 2020]. Planar SLAM methods operate similarly, either detecting planes from an RGB-D input [Salas-Moreno et al., 2014, Kaess, 2015, Hsiao et al., 2018, Hosseinzadeh et al., 2018, Zhou et al., 2020a], or in the monocular case from CNN predictions [Yang et al., 2016, Yang and Scherer, 2019, Hosseinzadeh et al., 2019] or from the reconstructed points [Arndt et al., 2020]. All previous methods rely on accurate initial plane predictions and, unlike our method, cannot confirm or reject planes within the inference process, nor compress the factor graph. Additionally, for real-time operation, these methods often layer optimisation [Yang et al., 2016] or construct reduced systems to leverage specific problem structure [Zhou et al., 2020a].

Recent methods have built hierarchical 3D scene graphs containing layers of objects, rooms and buildings in which edges describe non-probabilistic relations [Armeni et al., 2019, Rosinol et al., 2020, Wald et al., 2020, Wu et al., 2021]. Armeni *et al.* [Armeni et al., 2019] constructs the graph from an annotated mesh model while Rosinol *et al.* [Rosinol et al., 2020] build an incremental system that also tracks human meshes and has shown impressive results on simulated datasets.

5.3 Incremental Planar Abstraction Framework

We combine amortised inference via a neural network with distributed probabilistic inference via GBP to incrementally abstract factor graphs in SLAM. We emphasise that the master representation of the scene is always the factor graph and during online operation GBP is continually performing inference on this graph.

Algorithm 1 System Overview.

```
Initialise factor graph with first keyframe.
n_iterations = 0
while in operation do
    Run iteration of GBP
    n_iterations += 1
    if new keyframe then
        Feature matching (Section 5.3.1, Figure 5.2 left)
        Add NN plane hypotheses (Section 5.3.2, Figure 5.2 middle)
    if n_iterations % N == 0 then
        for each plane hypothesis do
            if confirm criteria satisfied (Equation 5.5) then
                Create rigid body plane (Figure 5.2 right)
            else if reject criteria satisfied (Equation 5.4) then
                Remove plane hypothesis (Figure 5.2 left)
    if n_iterations % M == 0 then
        for each pair of planes do
            if merge criteria satisfied then
                Merge planes (Section 5.3.4)
```

GBP is interrupted to perform editing of the factor graph, which occurs when: i) adding a new keyframe (Sections 5.3.1 and 5.3.2) ii) testing plane hypotheses (Section 5.3.3) or iii) merging planes (Section 5.3.4). These components are detailed in the following sections, and a high-level overview of the system-level operation is described in Algorithm 1.

5.3.1 Feature Extraction from Keyframes

We construct the abstract scene graph on top of a feature-based monocular SLAM system. We choose a sparse front-end for experimental purposes while we concentrate on the graphical back-end in this work.

Given a stream of live images, we use the ORB-SLAM2 [Mur-Artal and Tardós, 2017] front-end to build the raw factor graph composed of keyframes, points and reprojection factors (Figure 5.2 *left*). Reprojection factors penalise the $L2$ distance between a matched feature at image coordinates $\mathbf{z}$ in keyframe $\mathbf{c}$ and the projection of the corresponding point $\mathbf{p}$ in the image plane. Reprojection factors have the form:

$$l_r(\mathbf{c}, \mathbf{p}\,; \mathbf{z}) \propto \exp\left(-\frac{1}{2} \parallel \mathbf{z} - \text{proj}\left(R_c\,\mathbf{p} + \mathbf{t}_c\right) \parallel^2_{\Sigma_r} \right), \tag{5.1}$$

where proj is the projection operator and R_c and $\mathbf{t}_c$ are the rotations and translations derived from keyframe pose $\mathbf{c}$.

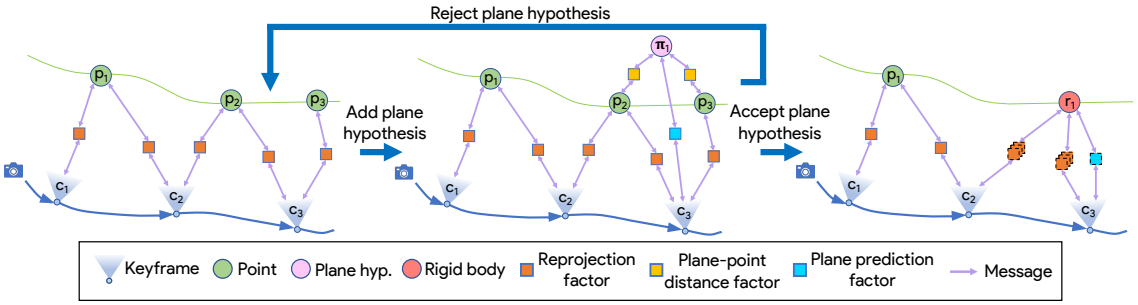


Figure 5.2: **Factor Graph.** *Left*: Raw factor graph with only keyframes and points. *Middle*: The network predicts plane $\boldsymbol{\pi}_1$ in keyframe 3 and points $\mathbf{p}_2$ and $\mathbf{p}_3$ lie inside the predicted segmentation mask. The plane hypothesis variable node is added to the graph along with 2 plane-point distance factors and a plane prediction factor. *Right*: The plane hypothesis is confirmed and the plane hypothesis and points are replaced by a rigid body plane node $\mathbf{r}_1$. The factors with a dashed border are factors that connect to a rigid body node and have a different functional form (Eq. 5.7 and 5.8). Linearised reprojection factors that connect to the same rigid body node are combined into a single factor which is represented by overlaying 3 reprojection factors. If the plane hypothesis is rejected the factor graph returns to the form on the left.

5.3.2 Integrating Plane Predictions

We denote the homogeneous plane vector $\boldsymbol{\pi} = (\pi_1, \pi_2, \pi_3, \pi_4)^\top \in \mathbb{P}^3$ [Hartley and Zisserman, 2004]. Points $\mathbf{p} \in \mathbb{R}^3$ lying in a plane satisfy $\hat{\mathbf{n}}^\top \mathbf{p} = d$, where $\hat{\mathbf{n}} = \frac{(\pi_1,\pi_2,\pi_3)^\top}{\sqrt{\pi_1^2+\pi_2^2+\pi_3^2}}$ is the normal vector and $d = \frac{-\pi_4}{\sqrt{\pi_1^2+\pi_2^2+\pi_3^2}}$ is the distance from the origin. The homogeneous plane vector is transformed between coordinate frames using the inverse transpose of the homogeneous point transform: $\boldsymbol{\pi}' = T^{-\top}\boldsymbol{\pi}$. For optimisation, we represent planes using the minimal parameterisation $\hat{\mathbf{n}} \cdot d$ and denote the $\boxminus$ operator to subtract the plane parameters in our chosen minimal form: $\boldsymbol{\pi_a} \boxminus \boldsymbol{\pi_b} \coloneqq \hat{\mathbf{n}}_a \cdot d_a - \hat{\mathbf{n}}_b \cdot d_b$.

For each keyframe, we run a forward pass of the PlaneRCNN model [Liu et al., 2019] to predict a set of plane parameters and corresponding segmentation masks. The model is based on Mask RCNN [He et al., 2017] with an additional head added to regress the plane normal. The predictions are filtered by removing planes with especially small or disconnected segmentation masks. The resulting *plane hypotheses* are then integrated into the existing factor graph as plane hypothesis nodes (Figure 5.2 *middle*).

Using the segmentation mask for each plane, we determine the map points that are predicted to lie in the plane and introduce plane-point distance factors, connecting the hypothesised plane $\boldsymbol{\pi}$ to each of these points $\mathbf{p}$. Plane-point distance factors penalise the perpendicular distance from a

point to a plane and have the form:

$$l_{pp}(\mathbf{p}, \boldsymbol{\pi}) \propto \exp\big(-\frac{1}{2} \parallel \hat{\mathbf{n}} \cdot \mathbf{p} - d \parallel^2_{\Sigma_{pp}}\big) . \tag{5.2}$$

Each plane hypothesis $\boldsymbol{\pi}$ is also connected to the keyframe $\mathbf{c}$ in which it was predicted via a plane prediction factor. Plane prediction factors treat the network prediction of the plane parameters $\boldsymbol{\pi}_z$ as a measurement and take the form:

$$l_{\pi p}(\boldsymbol{\pi}, \mathbf{c}\,; \boldsymbol{\pi}_z) \propto \exp\big(-\frac{1}{2} \parallel \boldsymbol{\pi}_z \boxminus T_{cw}^{-\top} \boldsymbol{\pi} \parallel^2_{\Sigma_{\pi p}}\big) , \tag{5.3}$$

where $T_{cw} \in SE(3)$ is the transformation from the global coordinate frame to the coordinate frame of the camera $\mathbf{c}$. In experiments we set $\Sigma_{\pi p}$ to be very large as the network plane parameter prediction can be unreliable.

We emphasise that our framework is not specific to planes. Any abstract scene element with an appropriate compatibility factor and inference model to generate hypotheses could be used.

5.3.3 Plane hypothesis confirmation and rejection

Having added the plane hypotheses to the raw factor graph, GBP carries out inference on the hybrid graph (Figure 5.2 *middle*) and converges to the configuration that minimises the energy of the constraints in the graph. To allow bad plane hypotheses to be treated as outlying measurements and have only a small contribution to the graph energy, we employ the robust Tukey loss function for all factors using covariance scaling as described in Section 3.3.

After convergence, for each plane hypothesis, we go through all connected points and read off from the factor graph the likelihood that the point lies in the plane. The likelihood is the plane-point distance factor density $l_{pp}(\mathbf{p}_{conv}, \boldsymbol{\pi}_{conv})$ evaluated at the converged belief means: $\mathbf{p}_{conv}$ and $\boldsymbol{\pi}_{conv}$. To determine whether to confirm or reject each plane hypothesis, we use the proportion of points y with likelihood $l_{pp}(\mathbf{p}_{conv}, \boldsymbol{\pi}_{conv}) > l_{thresh}$ and the number of iterations the hypothesis has been in the graph t.

$$\textbf{Reject criteria}: \quad y < y_{reject} \quad \textit{OR} \quad t > t_{max} \tag{5.4}$$

$$\textbf{Confirm criteria}: \quad y > y_{conf} \quad \textit{AND} \quad t > t_{min} \tag{5.5}$$

These criteria are based on the likelihoods, however a more rigorous approach would be to compare the marginal likelihood of the data with and without the plane as in Bayesian model selection [Mackay, 2003]. It remains an open question how to efficiently evaluate the marginal

likelihood in factor graphs and whether local approximations of the marginal likelihood could be used instead.

If rejected, the plane hypothesis node and all adjacent factors are removed from the graph, as in Figure 5.2. If confirmed, the plane hypothesis node and points with $l_{pp}(\mathbf{p}_{conv}, \boldsymbol{\pi}_{conv}) > l_{thresh}$ are replaced by a single rigid body variable node with just 6 degrees of freedom, as in Figure 5.2 *right*. We use a rigid body to represent the plane as we assume that the relative configuration of the planar points has now been determined and that they can be optimised as a single rigid planar body — this compression could also be used for small rigid objects.

The reprojection factors connected to the planar points, and the plane prediction factors connected to the plane hypothesis node are transferred to the new rigid body variable node. For these factors, the rigid body transformation $\mathbf{r}$ becomes an argument of the measurement function while $\boldsymbol{\pi}_{conv}$ and $\mathbf{p}_{conv}$ are parameters as we replace:

$$\boldsymbol{\pi} \rightarrow T_r^{-\top} \boldsymbol{\pi}_{conv} \quad \text{and} \quad \mathbf{p} \rightarrow R_r \mathbf{p}_{conv} + \mathbf{t}_r \ , \tag{5.6}$$

where T_r, R_r and $\mathbf{t}_r$ are the transformation matrix, rotation matrix and translation derived from the minimal $SE(3)$ vector $\mathbf{r}$. Plane prediction and reprojection factors become:

$$l_{\pi p}(\mathbf{r}, \mathbf{c}; \boldsymbol{\pi}_z, \boldsymbol{\pi}_{\boldsymbol{conv}}) \propto \exp\big(-\frac{1}{2} \parallel \boldsymbol{\pi}_z \boxminus T_{cw}^{-\top} T_r^{-\top} \boldsymbol{\pi}_{conv} \parallel^2_{\Sigma_{\pi p}} \big) \ , \tag{5.7}$$

$$l_r(\mathbf{c}, \mathbf{r}; \mathbf{z}, \mathbf{p}_{conv}) \propto \exp\big(-1/2 \parallel \mathbf{z} - proj(R_c(R_r \mathbf{p} + \mathbf{t}_r) + \mathbf{t}_c) \parallel^2_{\Sigma_r} \big) \tag{5.8}$$

As all transferred reprojection factors connect to the same pair of nodes, they can be linearised and combined into a single factor by taking the product of the factors. This further compresses the graph and reduces the computation at each iteration of GBP. Note that relinearising this combined factor requires linearising the contributions from all of the original reprojection factors, however this only occurs on a small proportion of GBP iterations.

5.3.4 Merging Planes

As there may be multiple plane hypotheses for the same geometric plane, we merge confirmed planes that have: i) aligned normal vectors, ii) small perpendicular separation, and iii) large overlap. The latter two criteria are estimated by sampling points in the plane. Once two planes are chosen to be merged, we replace the two rigid body planes with a single rigid body plane. The new plane normal is the average of the merged normals and the factors connecting to the merged planes are transferred to the new plane.

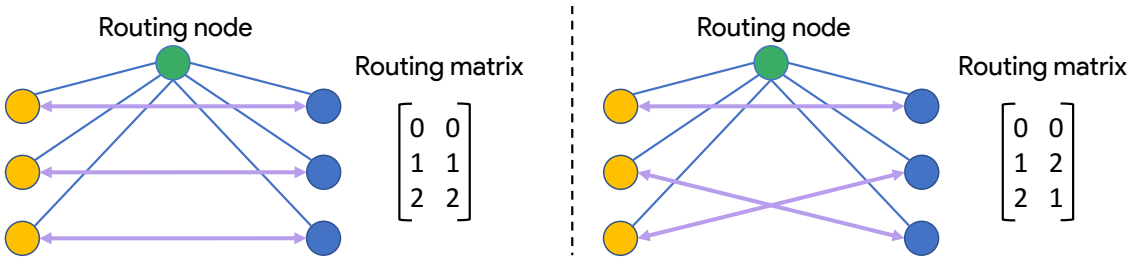


Figure 5.3: **Routing procedure**. The yellow and blue nodes are different node types. Purple lines represent edges in the factor graph and the blue lines represent communication edges between hardware cores. In this illustration, each node in the factor graph lives on a different hardware core.

5.4 Dynamic Routing on the IPU

We implement our incremental abstraction method on a single Graphcore MK1 IPU chip [Graphcore, 2022]. The IPU is a large chip composed of 1216 independent cores arranged in a fully connected graph structure. Like a GPU it is highly parallel, but due to its interconnect structure and the local memory on each core, it has breakthrough performance for algorithms with a sparse message passing character [Lacey, 2019].

We have already described in Section 4.5.1 how inference on static factor graphs is achieved by mapping the factor graph onto the IPU cores with the communication pattern matching the topology of the graph. This approach however cannot be applied to dynamic graphs, as the communication pattern must be precompiled and recompilation is expensive. To enable parallel inference of dynamic factor graphs, we develop a routing procedure that can manage arbitrary graph topologies while operating on a fixed precompiled communication pattern.

Our routing solution introduces densely connected routing nodes to mediate the transfer of messages through the factor graph. Routing nodes have knowledge about the structure of a part of the factor graph, stored in a routing matrix. As illustrated in Figure 5.3, when the factor graph is edited, the routing matrix can be updated to enable inference on the new factor graph without changing the compiled communication pattern.

To distribute the routing, we create a routing node for each factor type in the graph. One additional requirement is that we specify the maximum number of nodes of each type in the graph and the maximum number of edges each type of node can have — these are weak requirements and the limits can be set generously. For very large graphs we can simply create multiple routing nodes for each factor type to avoid routing becoming a bottleneck.

The routing procedure enables optimisation of arbitrary graph topologies at minimal time cost; only increasing the time per iteration from $200\mu s$ to $400\mu s$. As GBP typically converges in less than 100 iterations [Ortiz et al., 2020], inference remains comfortably within real-time constraints.

5.5 Implementation Settings

For all experiments, we use the following default parameters: $\Sigma_r = \sigma_r^2 I_2$, $\sigma_r = 2$ pixels, $\Sigma_{\pi p} = \sigma_{\pi p}^2 I_3$, $\sigma_{\pi p} = 20m$, $\Sigma_{pp} = \sigma_{pp}^2$, $\sigma_{pp} = 5cm$, $y_{reject} = 0.5$, $y_{conf} = 0.8$, $l_{thresh} = 0.8$, $t_{max} = 6000$ iterations, $t_{min} = 4000$ iterations, $N = M = 1000$ iterations and $\beta = 1e-4$. We use message dropout of 0.7 and message damping of 0.4 (see Section 3.4.6) to stabilise GBP [Bickson, 2008]. New keyframes are initialised with a constant velocity motion model and new points are initialised at an average depth from the camera.

5.6 Experimental Evaluation

For evaluation we use real-world sequences captured with the Kinect camera in varied indoor environments and sequences from the TUM dataset [Sturm et al., 2012]. In compression and tracking evaluations, we use Ortiz *et al.* [Ortiz et al., 2020] as a baseline and call the method GBP-BL. Similar to our system GBP-BL uses GBP for inference in SLAM but without planar abstractions, meaning that comparisons serve as ablations of our planar abstraction framework.

We first present convergence time evaluations and qualitative reconstructions before evaluating compression and tracking accuracy in ablation experiments.

5.6.1 Convergence Time Evaluation

We evaluate the convergence time for bundle adjustment problems based on the *sitting room* sequence for factor graphs with an increasing number of different factor types. We begin with a factor graph containing 35 keyframes, 3108 points and 10000 reprojection factors and measure the convergence time from noisy initialisations. We then add 200 additional factors of a new type to the graph along with any new variables nodes (for example we add plane hypotheses variable nodes when adding plane hypothesis prediction factors) and repeat the experiment. We conduct experiments adding 4 additional types of factors, meaning in the final experiment there are 10800 factors in the graph.

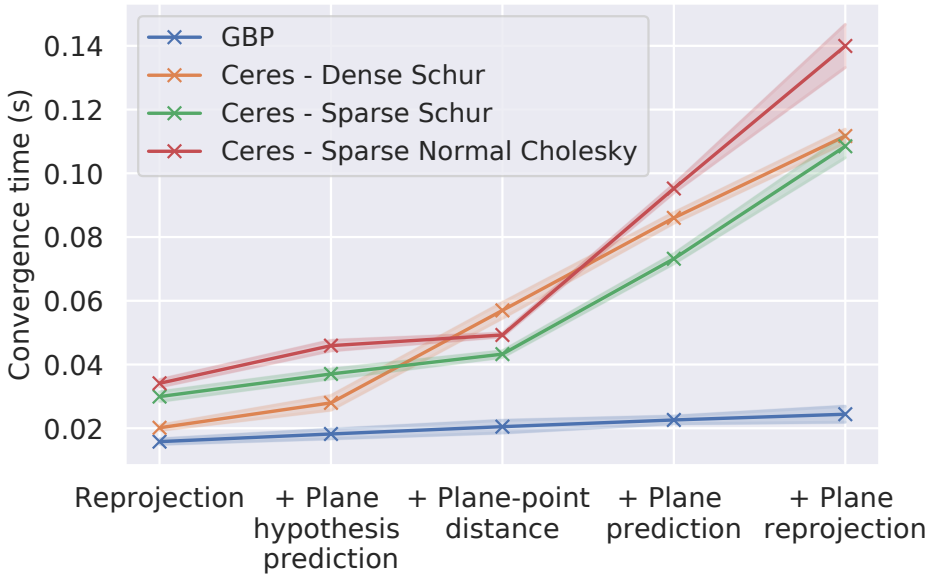


Figure 5.4: **Convergence time**. Mean and standard deviation error over 10 runs are plotted. We define convergence as reaching 1.5 pixels average reprojection error as in [Ortiz et al., 2020]. With the additional 800 factors, convergence time for Ceres with Dense Schur increases by roughly 5x from $20.1ms$ to $111.6ms$, while GBP increases from $15.6ms$ to $22.0ms$.

Following Ortiz *et al.* [Ortiz et al., 2020], we compare GBP implemented on a single IPU chip [Graphcore, 2022] with Ceres [Agarwal and Mierle, 2012], a non-linear least squares optimisation library, run on a 6 core i7-8700K CPU with 18 threads. Ceres uses Levenberg-Marquardt (LM), a Tukey kernel and analytic derivatives (which we found maximise performance). In Figure 5.4 we compare the convergence time of GBP with the 3 fastest Ceres linear solvers.

As different types of factors are added to the graph, convergence time for Ceres increases greatly while for GBP there is only a slight increase. To understand these curves, it is instructive to break down convergence time as the product of the time per iteration and the number of iterations to converge. Ceres makes optimal global updates through LM and, across all experiments, converges in 5-10 iterations while GBP requires 20-25 iterations. The time per iteration however is the more significant factor and where the two methods differ greatly in performance.

The local distributed nature of GBP makes its time per iteration *structure-agnostic*; in other words, the compute per iteration depends only on the number of factors and not their structure. Consequently, the time per iteration for GBP is approximately constant for all experiments and the convergence time only increases slightly, due to extra factors and more iterations required to converge.

In contrast, as different types of factors are added to the graph, solving the linear system or normal equations for the Ceres LM update becomes considerably more expensive, increasing the time per

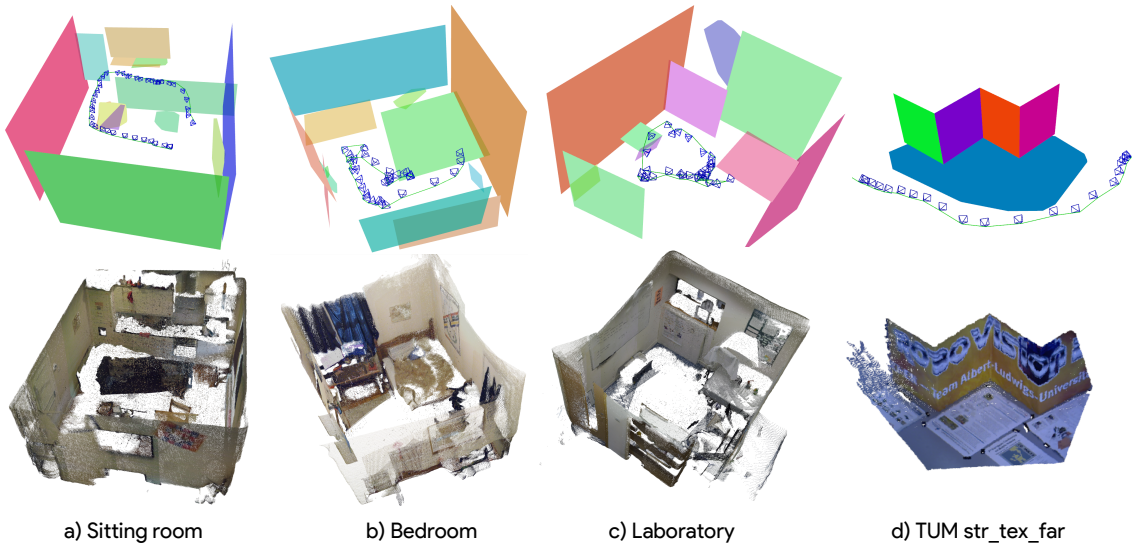


Figure 5.5: **Qualitative reconstructions**. *Top*: Planar reconstructions recovered by our system. Planes that are almost perpendicular or parallel to the largest planes in the reconstruction are displayed as rectangles, while the extent of other planes is the convex hull of the planar points. We do not include raw points that have not been abstracted in this visualisation. *Bottom*: Visualisation of the scene obtained using depth, shown for reference rather than comparison. Reconstructions *a)*, *b)* and *c)* are from sequences captured with the Kinect camera and *d)* is from the TUM sequence *structure texture far*. Perpendicular planes in *d)* are joined at their intersection for a watertight reconstruction. Note in sequence *c)*, the near side of the room is not abstracted into a plane as there is a cluttered bookshelf.

iteration. Ceres linear solvers compute the update by exploiting sparsity structure in the Fisher information matrix which depends on both the number of non-zero entries or equivalently factors in the graph and the variable ordering. In the base case, the Dense Schur solver is designed to leverage the large zero blocks in the information matrix to efficiently solve the normal equations without inverting the full information matrix. As different types of factors are added, even in very small numbers, these zero blocks are eroded and the time per iteration for Ceres increases by over 5x with only an additional 8% of factors.

These experiments expose the reliance of direct solvers on fixed sparsity structure and suggest that GBP is more efficient for optimising heterogeneous scene graphs without strong structure. Lastly, not only does GBP have the right computational properties, but it is also doing additional work by computing both the MAP and the marginal covariances while LM only computes the MAP with significant extra computation required to get the covariances.

5.6.2 Qualitative Reconstructions

We present planar reconstructions of 4 real-world sequences by our system in Figure 5.5. Our system captures the prominent planes in real scenes such as walls, beds (*b*), desks (*b*, *c*) and cupboards (*a*). In Figure 5.5 *d)* we verify that for a simple entirely planar scene, our system can achieve a complete reconstruction. Figure 5.1 shows an intermediate reconstruction of the same sequence with points. Qualitative video results that show the real-time reconstruction of the points and planar abstractions are available at: https://www.youtube.com/watch?v=ZoJ9ylb4Ss8.

The granularity of abstractions in our system is controlled by two interpretable parameters, which can be set according to the specific requirements of downstream processes which operate on the resulting reconstruction. For example, dense fine-grained geometry might be required for object manipulation, whereas coarse grained representations would be more efficient for path planning.

The value of σ_{pp} balances the contributions of the plane-point factors and the reprojection factors. As shown in Figure 5.6, weak plane-point factors (large σ_{pp}) prioritise minimising reprojection errors and consequently only very planar regions are abstracted. On the other hand, strong plane-point factors (small σ_{pp}) encourage points in hypothesised planes to be co-planar and can lead to the abstraction of regions that are approximately planar, such as the top of the shampoo bottle in the bottom right panel of Figure 5.6. Additionally, the merging criteria can be varied to determine the level of granularity. In Figure 5.6, coarse merging thresholds join the posters into a single wall plane and merge the planar objects into the table plane while finer thresholds leave these reconstructed planes separate.

5.6.3 Compression Evaluation

Graph compression yields a more dense, semantic and parameter-efficient representation and reduces the amount of computation per iteration of GBP. The computation is proportional to the number of edges or equivalently the number of factors when all factors are pairwise. To quantify the compression capabilities of our system, in Figure 5.7 we plot the number of factor nodes in the graph as a function of keyframes added and compare with GBP-BL [Ortiz et al., 2020].

For our planar system, there are initially more factors as many plane hypotheses are added, however the graph is quickly compressed as planes are confirmed. The compression is most significant in the *Laboratory* and *TUM str_tex_far* sequences in which there are large textured walls, while

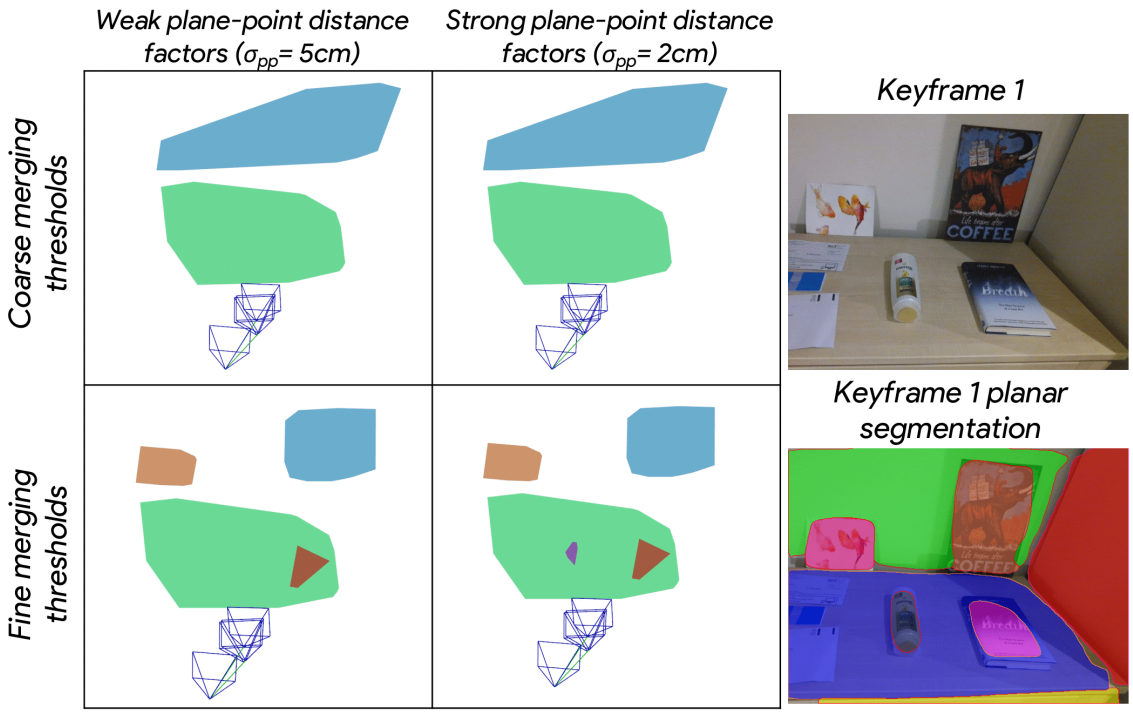


Figure 5.6: **Reconstruction granularity**. *Left*: Planar reconstruction for different parameter configurations. *Right*: RGB image and planar segmentation for the first keyframe.

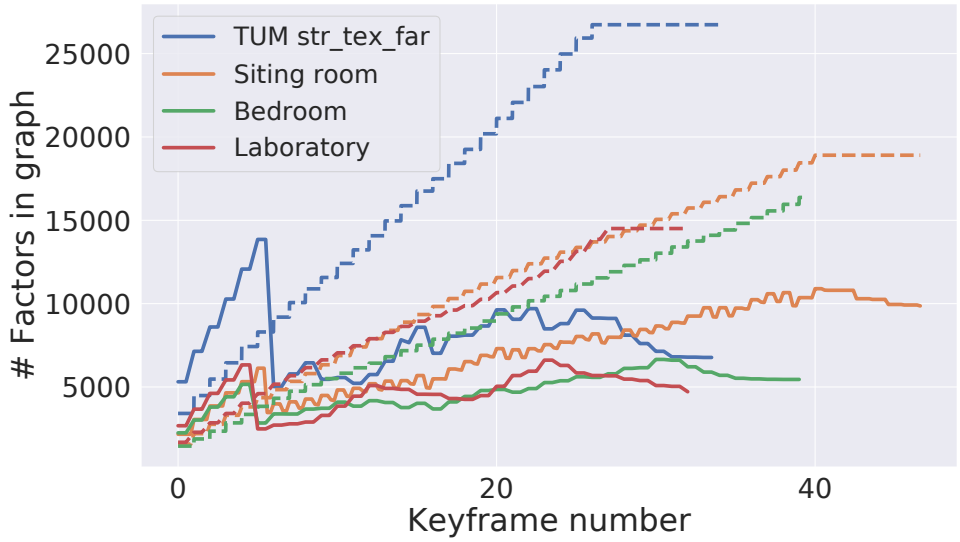


Figure 5.7: **Factor graph size.** Number of factor nodes in the graph as keyframes are added. Solid lines are our system, dashed lines are GBP-BL [Ortiz et al., 2020].

there is less compression in the *sitting room* sequence due to large curved objects such as the sofa.

Table 5.1: Average absolute trajectory error (ATE) across 16 runs for sections of TUM sequences [Sturm et al., 2012]. Ours-C is ours without compression.

ATE (cm)	Ours	Ours-C	GBP-BL	ORB-SLAM2
str_tex_far	1.204	1.186	1.384	**0.924**
cabinet	0.723	0.659	1.048	**0.601**
long_office	0.658	**0.648**	0.891	0.670

5.6.4 Tracking Evaluation

We evaluate the absolute trajectory error (ATE) of our method on 3 TUM sequences (chosen for their prominent planar regions) in Table 5.1. We compare our full method with two ablated systems: our planar method without compression (Ours-C) and GBP-BL which is a GBP based method for robust bundle adjustment with similar performance to Ceres.

Ours-C has lower ATE than GBP-BL demonstrating that additional planar constraints help tracking. As expected, our full method performs slightly worse than Ours-C because compression is lossy and bakes in errors. The fact that compression only comes with a small price in accuracy is further evidence that we are finding correct abstractions.

We also demonstrate comparable trajectory error to ORB-SLAM2 [Mur-Artal and Tardós, 2017], even achieving lower ATE for the *long office* sequence in which there are many planar points on desks and monitors and slower camera motion. Both our method and GBP-BL lack a tracking system so begin from a more difficult initialisation than ORB-SLAM2 when a new keyframe is added, likely explaining the performance difference on the first two sequences. Designing a distributed tracking system that would fit in with the GBP framework remains an important research direction.

5.7 Conclusions

We have proposed a method for efficient incremental construction of probabilistic scene graphs from monocular input by introducing two novel components. First, our incremental scene abstraction framework combines amortised inference via an off-the-shelf network with probabilistic inference to identify abstract scene elements and build a more semantic, dense and compact representation. Second, our routing procedure enables the first application of GBP on a graph processor to inference of dynamic heterogeneous factor graphs. We demonstrate the advantage of GBP over direct methods for optimising complex factor graphs due to the structure-agnostic time per

iteration. We implement our system with planar abstractions and show accurate reconstructions, significant compression and improved tracking over ablated systems.

In the near term, we would like to explore using our framework in a more practical SLAM system with an RGB-D input in which dense raw geometry is abstracted into a hierarchical factor graph including objects, planes and rooms. More generally, we hope that the SLAM community will begin to adopt more flexible and distributed inference frameworks, such as the system we present here. Such frameworks are vital to leverage novel parallel hardware and tackle the computational challenges brought about by slowing advancements in CPUs and GPUs and the move towards dynamic heterogeneous scene graphs for representing environments.

CHAPTER 6

Conclusions and Future Work

In this thesis, we have proposed, introduced and demonstrated Gaussian belief propagation as a strong and effective algorithm for real-time decentralised inference in Spatial AI. In Chapter 1, we argued that general, efficient and scalable inference in Spatial AI requires decentralised graph based algorithms that operate via local message passing on the factor graph with in-place local processing and data storage. This was inspired chiefly by current rapid developments in computing hardware and a move towards dynamic and heterogeneous scene representations that will require flexible and scalable inference algorithms. This line of reasoning led us to propose GBP as a strong candidate inference algorithm due to its unique properties of being: decentralised, probabilistic, iterative and convergent, and asynchronous.

Our broader vision of the role of distributed inference via GBP in Spatial AI systems is based on the non-linear factor graph being the master representation of all probabilistic constraints from multiple information sources. The factor graph can be efficiently stored and dynamically edited given incremental measurements and abstractions. Gaussian belief propagation is the simple but highly flexible inference tool which can compute the set of marginal probabilistic estimates with distributed processing and storage on the factor graph, suitable for graph processor chips or any distributed compute resource. Large real-time systems will operate with continual or attention-focused processing on their dynamic factor graphs, perhaps never reaching full convergence but with estimates always good enough to be useful, either locally or globally, or calculated on demand in a just-in-time manner. More specifically designed algorithms and specialised computing hardware will exist for focused uses, but GBP can serve as a general 'glue' which holds all of these together in a rigorous probabilistic framework.

Having presented this vision, in Chapter 2, we introduced the GBP algorithm, providing a derivation from first principles both for tree graphs and for loopy graphs from a variational inference

perspective. We also proposed a novel variant of the GBP algorithm for handling non-Euclidean variables using Lie Group theory. The key insight in this contribution is that all messages should take the form of a Lie Group point estimate and a precision matrix defined in the tangent space at this estimate. Although not presented in this thesis, GBP with Lie Groups was successfully applied to distributed multi-robot localisation and was shown to be tolerant to a high percentage of dropped messages [Murai et al., 2022].

In Chapter 3, we discussed a number of extra details and tricks that are required for solving real practical problems with GBP. With the help of examples, we demonstrated that GBP can converge with arbitrary asynchronous message schedules and that local robust factors can be used for effective outlier detection. Additionally, we discussed relinearisation and damping strategies that can improve both the chances and speed of convergence.

One key finding in this chapter was that accurate local convergence can be rapidly obtained long before full global convergence, particularly in SLAM-like problems with mainly relative measurements. In many applications, it is only the local relative information which is important (for instance a robot which needs to plan its next actions to avoid obstacles), and this raises the question of whether a different relative parameterisation could be more suitable. For example, Sibley *et al.* [Sibley et al., 2009] represent scene points relative to a camera pose and loop closure is a simple connection operation with the option for global adjustment only if needed.

In this final part of Chapter 3, we briefly presented the Robot Web protocol for ad-hoc peer-to-peer GBP communication. This develops our vision of GBP as the probabilistic inference glue between specialised inference modules for the problem of many-device localisation. Robot Web can be implemented by publishing and reading web pages or other asynchronous communication technologies without sharing any privileged local information, inspired by the original World Wide Web design. The protocol is closely related to ongoing work in clustered GBP, in which sub-problems are clustered into large nodes and message passing at the clusters is carried out with specialised centralised computation. In the longer term, we hope that developments in the Robot Web protocol may lead to GBP becoming a standard distributed estimation tool, rather than being instantiated in monolithic libraries like other estimation methods.

Chapter 4 presented the first application of GBP to bundle adjustment, implemented on graph processor hardware. Specifically, we leveraged Graphcore's IPU processor, an example of a novel flexible computer architecture for AI. By mapping the bundle adjustment factor graph onto the cores of the processor, GBP was able to rapidly solve a variety of challenging medium-sized bundle adjustment problems. In comparisons with the Ceres Solver on a CPU, we demonstrated

a 24x speed advantage and were able to solve most problems in under 50ms. We additionally presented some initial results for incremental SLAM problems and showed that GBP can be robust to outlying measurements.

One key limitation of this work is that for incremental problems, as exploration continues, the graph grows in an unbounded manner. As GBP operates via distributable message passing on the factor graph, when the size of the graph exceeds the number of available parallel threads, the implementation must involve some sequential computation therefore slowing down inference. The graph processor has several tens of thousands of parallel threads and this limit is reached for moderately large bundle adjustment problems. Although we could trivially scale to multiple chips, this would be infeasible for low-power embedded applications.

To retain the factor graph as our master representation and store and update it in real-time on an embedded processor, repeated simplification and abstraction is therefore essential to bound the size of a graph. We believe that the route towards simplification is continual detection and introspection, both in the front end and within the graph, to discover regions and structures which can be simplified by deleting, merging or compressing nodes. As outlined in the introduction, we hope the resulting hierarchical abstract scene graphs will represent the key structures, objects and interactions in a semantically rich and compact way, providing far more utility for downstream decision making than redundant low-level representations such as point clouds.

Graph simplification is challenging as robustly instantiating abstract scene elements can require an expensive search-and-test over both the abstraction and the subset of nodes to which is applies. In Chapter 5 we present a prototype system tackling this incremental abstraction problem in probabilistic SLAM graphs. Our system is based on an incremental abstraction framework that combines amortised inference via an off-the-shelf network with probabilistic inference to identify abstract scene elements and yield a more semantic, compact and dense representation. Our framework is designed to be general to arbitrary scene abstractions and we experiment with planar abstractions that are common in man-made environments. In experiments on real sequences, we demonstrate accurate planar reconstructions along with significant compression of the factor graph and improved camera tracking.

A second contribution in our incremental abstraction system is a routing procedure to enable GBP message passing on dynamic factor graphs given a fixed precompiled communication pattern on the IPU. Previous work in Chapter 4 assumed a static graph for bundle adjustment and routing provides a simple solution for dynamic graphs with limited additional overhead. In order to specify the precompiled graph, we analysed the typical graphs in an off-line experiment and used

the number of nodes and topology to create a large generic hierarchical structure that is filled dynamically at run-time.

With the ability to perform GBP on dynamic graphs on the IPU, we demonstrated the advantage of GBP over direct methods for inference on complex heterogeneous factor graphs due to the structure-agnostic time per iteration of GBP. We hope that as dynamic heterogeneous graphs become more common for representing environments, the community will begin to adopt more flexible distributed inference frameworks like GBP.

Our system is an early prototype for incremental reconstruction of hierarchical scene graphs and there remains a lot of work to be done to represent a richer set of scene elements such as objects and rooms. While we took the approach of directly predicting abstractions in the front-end via amortised inference, there are promising complementary directions. Recognition in the graph using Geometric Deep Learning [Bronstein et al., 2017b], which like GBP operates with in-place message passing, could enable fully graph-based semantic SLAM systems. A further promising alternative is to instantiate generic hierarchical structure and then learn the hierarchical features from data either in advance or at deployment time [George et al., 2017].

More generally, abstraction will play an important in maintaining 'small world' properties in the Spatial AI graph, such that a short path exists between any two nodes and therefore information flow and optimisation can always happen efficiently. Abstraction will also be crucial for attention based message passing in large graphs in which attention is actively focused around a region of interest for current decision making. Due to memory constraints for large graphs, as interest moves away from a region it may need to be abstracted into a low resolution approximate representation that can be brought back to high resolution with new incoming data and processing when revisited.

One important consideration is that it is difficult or impossible to reverse abstractions, unlike most other processing in GBP. We observed this in our experiments in Chapter 5 in which drift was baked into planar abstractions and could not be corrected in light of new measurements. Despite this, we maintain that making greedy assumptions and simplifying representations must ultimately be an essential part of efficient intelligence, and something that biological brains do as proven by optical illusions.

Looking forward, we see many exciting directions for future research around GBP for Spatial AI. Some directions we are most excited about are: improving theoretical guarantees, using learned factors [Czarnowski et al., 2020, Mukadam et al., 2018, Opipari et al., 2021], introducing discrete variables, combining GBP with GNNs [Satorras and Welling, 2021, Kuck et al., 2020], incre-

mentally abstracting factor graphs, investigating numerical precision for messages, using GBP for distributed learning in overparameterised networks and lastly unifying inference with test-time self-supervised learning.

Bibliography

[Absil et al., 2009] Absil, P.-A., Mahony, R., and Sepulchre, R. (2009). Optimization algorithms on matrix manifolds. In *Optimization Algorithms on Matrix Manifolds*. Princeton University Press. 49, 50

[Agarwal et al., 2012] Agarwal, P., Tipaldi, G. D., Spinello, L., Stachniss, C., and Burgard, W. (2012). Robust map optimization using dynamic covariance scaling. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 80, 81

[Agarwal and Mierle, 2012] Agarwal, S. and Mierle, K. (2012). Ceres solver: Tutorial & reference. *Google Inc*, 2(72):8. 7, 16, 80, 106, 124

[Agarwal et al., 2009] Agarwal, S., Snavely, N., Simon, I., Seitz, S. M., and Szeliski, R. (2009). Building Rome in a Day. In *Proceedings of the International Conference on Computer Vision (ICCV)*. 100

[Aragues et al., 2011] Aragues, R., Carlone, L., Calafiore, G., and Sagues, C. (2011). Multi-agent localization from noisy relative pose measurements. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 17

[Armeni et al., 2019] Armeni, I., He, Z., Gwak, J., Zamir, A., Fischer, M., Malik, J., and Savarese, S. (2019). 3d scene graph: A structure for unified semantics, 3d space, and camera. In *Proceedings of the International Conference on Computer Vision (ICCV)*. 12, 117

[Arndt et al., 2020] Arndt, C., Sabzevari, R., and Civera, J. (2020). From points to planes-adding planar constraints to monocular slam factor graphs. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*. 117

[Barfoot, 2020] Barfoot, T. D. (2020). Fundamental linear algebra problem of gaussian inference. *arXiv preprint arXiv:2010.08022*. 19

[Barooah and Hespanha, 2005] Barooah, P. and Hespanha, J. (2005). Distributed estimation from relative measurements in sensor networks. In *International Conference on Intelligent Sensing and Information Processing*. 17

[Battaglia et al., 2018] Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., et al. (2018). Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*. 13

[Bengio, 2017] Bengio, Y. (2017). The consciousness prior. *arXiv preprint arXiv:1709.08568*. 11

[Bengio, 2019] Bengio, Y. (2019). From System 1 Deep Learning to System 2 Deep Learning. https://www.youtube.com/watch?v=FtUbMG3rlFs. Keynote at NeurIPS 2019. 3

[Berners-Lee, 1999] Berners-Lee, T. (1999). *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web*. Harper. 94

[Bickson, 2008] Bickson, D. (2008). *Gaussian belief propagation: Theory and Application*. PhD thesis, PhD thesis, The Hebrew University of Jerusalem. 13, 41, 123

[Bishop, 2006] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer-Verlag New York, Inc. 13, 26

[Bloesch et al., 2018] Bloesch, M., Czarnowski, J., Clark, R., Leutenegger, S., and Davison, A. J. (2018). CodeSLAM — learning a compact, optimisable representation for dense visual SLAM. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 11, 100

[Boumal, 2023] Boumal, N. (2023). *An introduction to optimization on smooth manifolds*. Cambridge University Press. 49, 50

[Briggs et al., 2000] Briggs, W. L., Henson, V. E., and McCormick, S. F. (2000). *A multigrid tutorial*. SIAM. 89

[Bronstein et al., 2017a] Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., and Vandergheynst, P. (2017a). Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42. 3, 13

[Bronstein et al., 2017b] Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., and Vandergheynst, P. (2017b). Geometric Deep Learning: Going beyond Euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42. 134

[Carlevaris-Bianco et al., 2014] Carlevaris-Bianco, N., Kaess, M., and Eustice, R. (2014). Generic node removal for factor-graph slam. *IEEE Transactions on Robotics*, 30(6):1371–1385. 117

[Chambolle and Pock, 2011] Chambolle, A. and Pock, T. (2011). A First-Order Primal-Dual Algorithm for Convex Problems with Applications to Imaging. *Journal of Mathematical Imaging and Vision*, 40(1):120–145. 7

[Choudhary et al., 2017] Choudhary, S., Carlone, L., Nieto, C., Rogers, J., Christensen, H., and Dellaert, F. (2017). Distributed mapping with privacy and communication constraints: Lightweight algorithms and object-based models. *International Journal of Robotics Research (IJRR)*, 36(12):1286–1311. 18

[Cieslewski et al., 2018] Cieslewski, T., Choudhary, S., and Scaramuzza, D. (2018). Data-efficient decentralized visual SLAM. In *IEEE International Conference on Robotics and Automation (ICRA)*. 18

[Cohen and Welling, 2016] Cohen, T. and Welling, M. (2016). Group equivariant convolutional networks. In *International conference on machine learning*, pages 2990–2999. PMLR. 3

[Crandall et al., 2011] Crandall, D., Owens, A., Snavely, N., and Huttenlocher, D. (2011). Discrete-continuous optimization for large-scale structure from motion. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 13, 100

[Cristofalo et al., 2020] Cristofalo, E., Montijano, E., and Schwager, M. (2020). Geod: Consensus-based geodesic distributed pose graph optimization. *arXiv preprint arXiv:2010.00156*. 17

[Cunningham et al., 2010] Cunningham, A., Paluri, M., and Dellaert, F. (2010). DDF-SAM: Fully distributed SLAM using constrained factor graphs. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 19

[Czarnowski et al., 2020] Czarnowski, J., Laidlow, T., Clark, R., and Davison, A. J. (2020). Deepfactors: Real-time probabilistic dense monocular slam. *IEEE Robotics and Automation Letters*, 5(2):721–728. 44, 134

[Davison, 2005] Davison, A. J. (2005). Active Search for Real-Time Vision. In *Proceedings of the International Conference on Computer Vision (ICCV)*. 78

[Davison, 2018] Davison, A. J. (2018). FutureMapping: The computational structure of Spatial AI systems. *arXiv preprint arXiv:arXiv:1803.11288*. 7, 9, 79, 98

[Davison and Ortiz, 2019] Davison, A. J. and Ortiz, J. (2019). FutureMapping 2: Gaussian Belief Propagation for Spatial AI. *arXiv preprint arXiv:arXiv:1910.14139*. 11, 20, 21, 94, 99, 100, 116

[Dellaert, 2012] Dellaert, F. (2012). Factor graphs and GTSAM. Technical Report GT-RIM-CP&R-2012-002, Georgia Institute of Technology. 7, 17

[Dellaert, 2021] Dellaert, F. (2021). Factor graphs: Exploiting structure in robotics. *Annual Review of Control, Robotics, and Autonomous Systems*, 4. 5

[Dellaert and Kaess, 2017] Dellaert, F. and Kaess, M. (2017). Factor Graphs for Robot Perception. *Foundations and Trends in Robotics*, 6(1–2):1–139. 5, 25, 115

[Delouille et al., 2004] Delouille, V., Neelamani, R., and Baraniuk, R. (2004). Robust distributed estimation in sensor networks using the embedded polygons algorithm. In *International Symposium on Information Processing in Sensor Networks*. 18

[Demmel et al., 2020] Demmel, N., Gao, M., Laude, E., Wu, T., and Cremers, D. (2020). Distributed photometric bundle adjustment. In *2020 International Conference on 3D Vision (3DV)*, pages 140–149. IEEE. 18

[Dennis Jr, 1977] Dennis Jr, J. (1977). *Nonlinear Least Squares and Equations, in the State of the Art of Numerical Analysis*. PhD thesis, ed. D. Jacobs, Academic Press, London. 42

[DeVito et al., 2017] DeVito, Z., Mara, M., Zollhöfer, M., Bernstein, G., and Ragan-Kelley, J. (2017). Christian eobalt, pat hanrahan, ma hew fisher, and ma hias nießner. 2016. opt: A domain specific language for non-linear least squares optimization in graphics and imaging. In *ACM Transactions on Graphics (TOG)*. 101

[Diehl et al., 2018] Diehl, P. U., Martel, J., Buhmann, J., and Cook, M. (2018). Factorized computation: What the neocortex can tell us about the future of computing. *Frontiers in computational neuroscience*, 12:54. 74

[Du et al., 2017] Du, J., Ma, S., Wu, Y.-C., Kar, S., and Moura, J. M. (2017). Convergence analysis of distributed inference with vector-valued gaussian belief propagation. *Journal of Machine Learning Research*, 18:172–1. 88

[Du et al., 2018] Du, J., Ma, S., Wu, Y.-C., Kar, S., and Moura, J. M. (2018). Convergence analysis of belief propagation on gaussian graphical models. *arXiv preprint arXiv:1801.06430*. 41

[Duckett et al., 2000] Duckett, T., Marsland, S., and Shapiro, J. (2000). Learning globally consistent maps by relaxation. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 18

[Elidan et al., 2006] Elidan, G., McGraw, I., and Koller, D. (2006). Residual belief propagation: Informed scheduling for asynchronous message passing. In *In Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*. 72, 89

[Engel et al., 2017] Engel, J., Koltun, V., and Cremers, D. (2017). Direct sparse odometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*. 100

[Engel, 2017] Engel, J.-J. (2017). *Large-Scale Direct SLAM and 3D Reconstruction in Real-Time*. PhD thesis, PhD thesis, Technical University Munich. 7

[Eriksson et al., 2016] Eriksson, A., Bastian, J., Chin, T.-J., and Isaksson, M. (2016). A consensus-based framework for distributed bundle adjustment. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 18

[Eustice et al., 2005] Eustice, R. M., Singh, H., and Leonard, J. J. (2005). Exactly Sparse Delayed State Filters. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 38, 48

[Evans and Burgess, 2019] Evans, T. and Burgess, N. (2019). Coordinated hippocampal-entorhinal replay as structural inference. In *Neural Information Processing Systems (NeurIPS)*, volume 32. 71

[Fan and Murphey, 2020] Fan, T. and Murphey, T. (2020). Majorization minimization methods for distributed pose graph optimization with convergence guarantees. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 17

[Felzenszwalb and Huttenlocher, 2006] Felzenszwalb, P. F. and Huttenlocher, D. P. (2006). Efficient belief propagation for early vision. *International journal of computer vision*, 70(1):41–54. 13, 89

[Folkesson and Christensen, 2004] Folkesson, J. and Christensen, H. (2004). Graphical SLAM — a self-correcting map. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 100

[Geiger et al., 2012] Geiger, A., Lenz, P., and Urtasun, R. (2012). Are we ready for autonomous driving? The KITTI vision benchmark suite. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 106

[George et al., 2017] George, D., Lehrach, W., Kansky, K., Lázaro-Gredilla, M., Laan, C., Marthi, B., Lou, X., Meng, Z., Liu, Y., Wang, H., et al. (2017). A generative vision model that trains with high data efficiency and breaks text-based captchas. *Science*, 358(6368). 14, 90, 134

[Gervet et al., 2022] Gervet, T., Chintala, S., Batra, D., Malik, J., and Chaplot, D. S. (2022). Navigating to objects in the real world. *arXiv preprint arXiv:2212.00922*. 2

[Ghahramani, 2015] Ghahramani, Z. (2015). Probabilistic machine learning and artificial intelligence. *Nature*, 521(7553):452–459. 5

[Graphcore, 2022] Graphcore (2022). Graphcore. https://www.graphcore.ai/. 10, 99, 105, 116, 122, 124

[Grisetti et al., 2009] Grisetti, G., Stachniss, C., and Burgard, W. (2009). Non-linear constraint network optimization for efficient map learning. *IEEE Transactions on Intelligent Transportation Systems*, 10(3):428–439. 17

[Gui et al., 2019] Gui, C.-Y., Zheng, L., He, B., Liu, C., Chen, X.-Y., Liao, X.-F., and Jin, H. (2019). A survey on graph processing accelerators: Challenges and opportunities. *Journal of Computer Science and Technology*, 34(2):339–371. 10

[Gupta et al., 2010] Gupta, S., Choudhary, S., and P.J.Narayanan. (2010). Practical time bundle adjustment for 3D reconstruction on GPU. In *ECCV Workshop on Computer Vision on GPUs*. 101

[Halsted et al., 2021] Halsted, T., Shorinwa, O., Yu, J., and Schwager, M. (2021). A survey of distributed optimization methods for multi-robot systems. *ArXiv*, abs/2103.12840. 16

[Hartley and Zisserman, 2004] Hartley, R. and Zisserman, A. (2004). *Multiple View Geometry in Computer Vision*. Cambridge University Press, second edition. 119

[He et al., 2017] He, K., Gkioxari, G., Dollár, P., and Girshick, R. (2017). Mask R-CNN. In *Proceedings of the International Conference on Computer Vision (ICCV)*. 119

[Hosseinzadeh et al., 2018] Hosseinzadeh, M., Latif, Y., Pham, T., Suenderhauf, N., and Reid, I. (2018). Structure aware slam using quadrics and planes. In *Proceedings of the Asian Conference on Computer Vision (ACCV)*. 117

[Hosseinzadeh et al., 2019] Hosseinzadeh, M., Li, K., Latif, Y., and Reid, I. (2019). Real-time monocular object-model aware sparse slam. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 7123–7129. IEEE. 117

[Hsiao et al., 2018] Hsiao, M., Westman, E., and Kaess, M. (2018). Dense planar-inertial slam with structural constraints. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 6521–6528. IEEE. 117

[Huber, 1964] Huber, P. J. (1964). Robust Estimation of a Location Parameter. *The Annals of Mathematical Statistics*, 35(1):pp. 73–101. 81

[Huber, 1981] Huber, P. J. (1981). *Robust Statistics*. Wiley Series in Probability and Statistics. Wiley-Interscience. 81

[Ila et al., 2009] Ila, V., Porta, J., and Andrade-Cetto, J. (2009). Information-based compact pose SLAM. *IEEE Transactions on Robotics*, 26(1):78–93. 117

[Jaynes, 2003] Jaynes, E. T. (2003). *Probability Theory: The Logic of Science*. Cambridge University Press. 5, 6, 36

[Jeong et al., 2010] Jeong, Y., Nister, D., Steedly, D., Szeliski, R., and Kweon, I. (2010). Pushing the envelope of modern methods for bundle adjustment. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 100

[Jia et al., 2019] Jia, Z., Tillman, B., Maggioni, M., and Scarpazza, D. P. (2019). Dissecting the Graphcore IPU architecture via microbenchmarking. *arXiv preprint arXiv:1912.03413*. 105

[Johannsson et al., 2013] Johannsson, H., Kaess, M., Fallon, M., and Leonard, J. (2013). Temporally scalable visual SLAM using a reduced pose graph. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 117

[Kaess, 2015] Kaess, M. (2015). Simultaneous localization and mapping with infinite planes. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 117

[Kaess et al., 2012] Kaess, M., Johannsson, H., Roberts, R., Ila, V., Leonard, J., and Dellaert, F. (2012). iSAM2: Incremental Smoothing and Mapping Using the Bayes Tree. *International Journal of Robotics Research (IJRR)*. 7, 100

[Kaess et al., 2008] Kaess, M., Ranganathan, A., and Dellaert, F. (2008). iSAM: Incremental Smoothing and Mapping. *IEEE Transactions on Robotics (T-RO)*, 24(6):1365–1378. 100

[Kahneman, 2017] Kahneman, D. (2017). *Thinking, fast and slow*. Farrar, Straus and Giroux. 3

[Koller and Friedman, 2009] Koller, D. and Friedman, N. (2009). *Probabilistic graphical models: principles and techniques*. MIT press. 71

[Kschischang et al., 2001] Kschischang, F. R., Frey, B. J., and Loeliger, H.-A. (2001). Factor graphs and the sum-product algorithm. *IEEE Transactions on information theory*, 47(2):498–519. 13

[Kuck et al., 2020] Kuck, J., Chakraborty, S., Tang, H., Luo, R., Song, J., Sabharwal, A., and Ermon, S. (2020). Belief propagation neural networks. In *Neural Information Processing Systems (NeurIPS)*. 14, 134

[Kummerle et al., 2011] Kummerle, R., Grisetti, G., Strasdat, H., Konolige, K., and Burgard, W. (2011). g 2 o: A general framework for graph optimization. In *2011 IEEE International Conference on Robotics and Automation*, pages 3607–3613. IEEE. 7, 17

[Lacey, 2019] Lacey, D. (2019). New Graphcore IPU Benchmarks. https://www.graphcore.ai/posts/new-graphcore-ipu-benchmarks. 10, 99, 122

[Lajoie et al., 2020] Lajoie, P.-Y., Ramtoula, B., Chang, Y., Carlone, L., and Beltrame, G. (2020). Door-slam: Distributed, online, and outlier resilient slam for robotic teams. *IEEE Robotics and Automation Letters*, 5(2):1656–1663. 18

[Lázaro-Gredilla et al., 2021] Lázaro-Gredilla, M., Lehrach, W., Gothoskar, N., Zhou, G., Dedieu, A., and George, D. (2021). Query training: Learning a worse model to infer better marginals in undirected graphical models with hidden variables. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 8252–8260. 14

[LeCun et al., 2006] LeCun, Y., Chopra, S., Hadsell, R., Ranzato, M., and Huang, F. (2006). A tutorial on energy-based learning. *Predicting structured data MIT Press*, 1(0). 26

[Leung et al., 2012] Leung, K. Y., Barfoot, T. D., and Liu, H. H. (2012). Decentralized cooperative SLAM for sparsely-communicating robot networks: A centralized-equivalent approach. *Journal of Intelligent and Robotic Systems (JIRS)*, 66(3):3554–3561. 19

[Leung et al., 2010] Leung, K. Y. K., Barfoot, T. D., and Liu, H. H. T. (2010). Decentralized localization of sparsely-communicating robot networks: A centralized-equivalent approach. *IEEE Transactions on Robotics (T-RO)*, 26(1):62–77. 19

[Leutenegger et al., 2014] Leutenegger, S., Lynen, S., Bosse, M., Siegwart, R., and Furgale, P. (2014). Keyframe-based visual-inertial odometry using nonlinear optimization. *The International Journal of Robotics Research*, 34(3):314–334. 7

[Liu et al., 2019] Liu, C., Kim, K., Gu, J., Furukawa, Y., and Kautz, J. (2019). PlaneRCNN: 3D Plane Detection and Reconstruction from a Single Image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 119

[Lourakis, 2010] Lourakis, M. I. (2010). Sparse non-linear least squares optimization for geometric vision. In *Computer Vision–ECCV 2010: 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5-11, 2010, Proceedings, Part II 11*, pages 43–56. Springer. 42

[Lu and Milios, 1997] Lu, F. and Milios, E. (1997). Globally Consistent Range Scan Alignment for Environment Mapping. *Autonomous Robots*, 4(4):333–349. 100

[Mackay, 2003] Mackay, D. (2003). *Information Theory, Inference and Learning Algorithms*. Cambridge University Press. 120

[Malioutov et al., 2006] Malioutov, D. M., Johnson, J. K., and Willsky, A. S. (2006). Walk-sums and belief propagation in Gaussian graphical models. *Journal of Machine Learning Research*, 7(Oct):2031–2064. 93, 104

[Martel, 2019] Martel, J. (2019). *Unconventional Processing with Unconventional Visual Sensing*. PhD thesis, ETH Zurich. 9

[Martiros et al., 2022] Martiros, H., Miller, A., Bucki, N., Solliday, B., Kennedy, R., Zhu, J., Dang, T., Pattison, D., Zheng, H., Tomic, T., et al. (2022). Symforce: Symbolic computation and code generation for robotics. In *Robotics: Science and Systems (RSS)*. 17

[Mattamala, 2021] Mattamala, M. (2021). Reducing the uncertainty about the uncertainties, part 2: Frames and manifolds. https://gtsam.org/2021/02/23/uncertainties-part2.html. 49

[Mayer, 2019] Mayer, H. (2019). Rpba–robust parallel bundle adjustment based on covariance information. *arXiv preprint arXiv:1910.08138*. 18

[Mazuran et al., 2016] Mazuran, M., Burgard, W., and Tipaldi, G. D. (2016). Nonlinear factor recovery for long-term SLAM. *International Journal of Robotics Research (IJRR)*, 35(1):50–72. 117

[McCormac et al., 2017] McCormac, J., Handa, A., Davison, A. J., and Leutenegger, S. (2017). SemanticFusion: Dense 3D semantic mapping with convolutional neural networks. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 8, 114, 115

[McEliece et al., 1998] McEliece, R. J., MacKay, D. J. C., and Cheng, J.-F. (1998). Turbo decoding as an instance of pearl's "belief propagation" algorithm. *IEEE Journal on selected areas in communications*, 16(2):140–152. 13

[Mescheder et al., 2019] Mescheder, L., Oechsle, M., Niemeyer, M., Nowozin, S., and Geiger, A. (2019). Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4460–4470. 11

[Meta, 2019] Meta (2019). Facebook reality labs: Livemaps — oculus connect 6. https://www.youtube.com/watch?v=JTa8zn0RNVM. 12

[Micusik and Evangelidis, 2020] Micusik, B. and Evangelidis, G. (2020). Ego-motion alignment from face detections for collaborative augmented reality. *arXiv preprint arXiv:2010.02153*. 79

[Mildenhall et al., 2020] Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R., and Ng, R. (2020). NeRF: Representing scenes as neural radiance fields for view synthesis. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 405–421. Springer. 2, 11

[Minka et al., 2018] Minka, T., Cleven, R., and Zaykov, Y. (2018). Trueskill 2: An improved bayesian skill rating system. *Technical Report*. 19

[Minka, 2013] Minka, T. P. (2013). Expectation propagation for approximate bayesian inference. *arXiv preprint arXiv:1301.2294*. 18

[Mourikis and Roumeliotis, 2007] Mourikis, A. I. and Roumeliotis, S. I. (2007). A multi-state constraint Kalman filter for vision-aided inertial navigation. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 3565–3572. IEEE. 7

[Mukadam et al., 2018] Mukadam, M., Dong, J., Yan, X., Dellaert, F., and Boots, B. (2018). Continuous-time gaussian process motion planning via probabilistic inference. *The International Journal of Robotics Research*, 37(11):1319–1340. 44, 134

[Mur-Artal et al., 2015] Mur-Artal, R., Montiel, J. M. M., and Tardós, J. D. (2015). ORB-SLAM: a Versatile and Accurate Monocular SLAM System. *IEEE Transactions on Robotics (T-RO)*, 31(5):1147–1163. 100, 106

[Mur-Artal and Tardós, 2017] Mur-Artal, R. and Tardós, J. D. (2017). ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras. *IEEE Transactions on Robotics (T-RO)*, 33(5):1255–1262. 118, 128

[Murai et al., 2022] Murai, R., Ortiz, J., Saeedi, S., Kelly, P. H., and Davison, A. J. (2022). A robot web for distributed many-device localisation. *arXiv preprint arXiv:2202.03314*. 15, 20, 21, 49, 51, 87, 94, 95, 132

[Murphy, 2012] Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press. 88

[Murphy, 2023] Murphy, K. P. (2023). *Probabilistic Machine Learning: Advanced Topics*. MIT Press. 60

[Murphy et al., 1999] Murphy, K. P., Weiss, Y., and Jordan, M. I. (1999). Loopy belief propagation for approximate inference: An empirical study. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*. 13, 33, 93

[Nardi et al., 2015] Nardi, L., Bodin, B., Zia, M. Z., Mawer, J., Nisbet, A., Kelly, P. H., Davison, A. J., Lujan, M., OBoyle, M. F., Riley, G., Topham, N., and Furber, S. (2015). Introducing SLAMBench, a performance and accuracy benchmarking methodology for SLAM. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 98

[Newcombe, 2012] Newcombe, R. A. (2012). *Dense Visual SLAM*. PhD thesis, Imperial College London. 7

[Newcombe et al., 2011] Newcombe, R. A., Izadi, S., Hilliges, O., Molyneaux, D., Kim, D., Davison, A. J., Kohli, P., Shotton, J., Hodges, S., and Fitzgibbon, A. (2011). KinectFusion: Real-Time Dense Surface Mapping and Tracking. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*. 87, 115

[Olson and Agarwal, 2013] Olson, E. and Agarwal, P. (2013). Inference on networks of mixtures for robust robot mapping. *International Journal of Robotics Research (IJRR)*, 32(7):826–840. 84

[Olson et al., 2006] Olson, E., Leonard, J. J., and Teller, S. (2006). Fast iterative alignment of pose graphs with poor initial estimates. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 17

[Opipari et al., 2021] Opipari, A., Chen, C., Wang, S., Pavlasek, J., Desingh, K., and Jenkins, O. C. (2021). Differentiable nonparametric belief propagation. *arXiv preprint arXiv:2101.05948*. 14, 134

[Ortiz et al., 2022a] Ortiz, J., Clegg, A., Dong, J., Sucar, E., Novotny, D., Zollhoefer, M., and Mukadam, M. (2022a). iSDF: Real-Time Neural Signed Distance Fields for Robot Perception. *arXiv preprint arXiv:2204.02296*. 11, 20

[Ortiz et al., 2021] Ortiz, J., Evans, T., and Davison, A. J. (2021). A visual introduction to gaussian belief propagation. *arXiv preprint arXiv:2107.02308*. 20, 21, 63, 116

[Ortiz et al., 2022b] Ortiz, J., Evans, T., Sucar, E., and Davison, A. J. (2022b). Incremental abstraction in distributed probabilistic slam graphs. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 12, 20, 22, 87, 90, 94, 113

[Ortiz et al., 2020] Ortiz, J., Pupilli, M., Leutenegger, S., and Davison, A. (2020). Bundle Adjustment on a Graph Processor. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 14, 20, 21, 22, 87, 93, 94, 97, 116, 123, 124, 126, 127

[Park et al., 2019] Park, J. J., Florence, P., Straub, J., Newcombe, R., and Lovegrove, S. (2019). Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 165–174. 11

[Paskin, 2003] Paskin, M. A. (2003). Thin Junction Tree Filters for Simultaneous Localization and Mapping. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 13, 100

[Pearl, 1982] Pearl, J. (1982). Reverend Bayes on inference engines: A distributed hierarchical approach. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*. 26

[Pearl, 1988] Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann. 13

[Pineda et al., 2022] Pineda, L., Fan, T., Monge, M., Venkataraman, S., Sodhi, P., Chen, R. T., Ortiz, J., DeTone, D., Wang, A., Anderson, S., Dong, J., Amos, B., and Mukadam, M. (2022). Theseus: A Library for Differentiable Nonlinear Optimization. *Advances in Neural Information Processing Systems*. 17, 21

[Ramamurthy et al., 2020] Ramamurthy, K. N., Lin, C.-C., Aravkin, A., Pankanti, S., and Viguier, R. (2020). Distributed bundle adjustment. In *Proceedings of the International Conference on 3D Vision (3DV)*. 18

[Ranganathan et al., 2007] Ranganathan, A., Kaess, M., and Dellaert, F. (2007). Loopy SAM. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 13, 18, 46, 72, 78, 100

[Rockafellar, 1976] Rockafellar, R. T. (1976). Monotone operators and the proximal point algorithm. *SIAM journal on control and optimization*, 14(5):877–898. 18

[Rosinol et al., 2020] Rosinol, A., Gupta, A., Abate, M., Shi, J., and Carlone, L. (2020). 3D Dynamic Scene Graphs: Actionable Spatial Perception with Places, Objects, and Humans. *arXiv preprint arXiv:2002.06289*. 8, 12, 117

[Rublee et al., 2011] Rublee, E., Rabaud, V., Konolige, K., and Bradski, G. (2011). ORB: an efficient alternative to SIFT or SURF. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 2564–2571. IEEE. 106

[Saeedi et al., 2018] Saeedi, S., Bodin, B., Wagstaff, H., Nisbet, A., Nardi, L., Mawer, J., Melot, N., Palomar, O., Vespa, E., Spink, T., et al. (2018). Navigating the landscape for real-time localization and mapping for robotics and virtual and augmented reality. *Proceedings of the IEEE*. 100

[Salas-Moreno et al., 2014] Salas-Moreno, R. F., Glocker, B., Kelly, P. H. J., and Davison, A. J. (2014). Dense planar SLAM. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*. 117

[Salas-Moreno et al., 2013] Salas-Moreno, R. F., Newcombe, R. A., Strasdat, H., Kelly, P. H. J., and Davison, A. J. (2013). SLAM++: Simultaneous Localisation and Mapping at the Level of Objects. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 114, 117

[Satorras and Welling, 2021] Satorras, V. G. and Welling, M. (2021). Neural enhanced belief propagation on factor graphs. In *International Conference on Artificial Intelligence and Statistics*, pages 685–693. PMLR. 14, 134

[Scarselli et al., 2008] Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. (2008). The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80. 13

[Scona et al., 2022] Scona, R., Matsuki, H., and Davison, A. (2022). From scene flow to visual odometry through local and global regularisation in markov random fields. *IEEE Robotics and Automation Letters*, 7(2):4299–4306. 87

[Sibley et al., 2009] Sibley, G., Mei, C., Reid, I., and Newman, P. (2009). Adaptive Relative Bundle Adjustment. In *Proceedings of Robotics: Science and Systems (RSS)*. 132

[Solà et al., 2018] Solà, J., Deray, J., and Atchuthan, D. (2018). A micro Lie theory for state estimation in robotics. *arXiv:1812.01537*. 50

[Strasdat et al., 2010] Strasdat, H., Montiel, J. M. M., and Davison, A. J. (2010). Real-Time Monocular SLAM: Why Filter? In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 7

[Sturm et al., 2012] Sturm, J., Engelhard, N., Endres, F., Burgard, W., and Cremers, D. (2012). A Benchmark for the Evaluation of RGB-D SLAM Systems. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*. 106, 123, 128

[Su and Wu, 2015] Su, Q. and Wu, Y.-C. (2015). On convergence conditions of gaussian belief propagation. *IEEE Transactions on Signal Processing*, 63(5):1144–1155. 41, 93

[Sucar et al., 2021] Sucar, E., Liu, S., Ortiz, J., and Davison, A. J. (2021). iMAP: Implicit Mapping and Positioning in Real-Time. In *Proceedings of the International Conference on Computer Vision (ICCV)*. 8, 11, 20

[Sucar et al., 2020] Sucar, E., Wada, K., and Davison, A. (2020). NodeSLAM: Neural object descriptors for multi-view shape reconstruction. In *Proceedings of the International Conference on 3D Vision (3DV)*. 11, 114, 117

[Sutter, 2011] Sutter, H. (2011). Welcome to the jungle. `https://herbsutter.com/welcome-to-the-jungle`. 9

[Sutton and McCallum, 2007] Sutton, C. and McCallum, A. (2007). Improved dynamic schedules for belief propagation. In *In Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*. 72

[Sutton, 2019] Sutton, R. (2019). The bitter lesson. `http://www.incompleteideas.net/IncIdeas/BitterLesson.html`. 10

[Sutton and Barto, 2018] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press. 3

[Sze et al., 2017] Sze, V., Chen, Y.-H., Yang, T.-J., and Emer, J. S. (2017). Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329. 10

[Takahashi, 1973] Takahashi, K. (1973). Formation of sparse bus impedance matrix and its application to short circuit study. In *Proc. PICA Conference, June, 1973*. 19

[Tian et al., 2021] Tian, Y., Chang, Y., Arias, F. H., Nieto-Granda, C., How, J. P., and Carlone, L. (2021). Kimera-multi: Robust, distributed, dense metric-semantic slam for multi-robot systems. *arXiv preprint arXiv:2106.14386*. 17

[Tian et al., 2019] Tian, Y., Khosoussi, K., Rosen, D. M., and How, J. P. (2019). Distributed certifiably correct pose-graph optimization. *arXiv preprint arXiv:1911.03721*. 17

[Tian et al., 2020] Tian, Y., Koppel, A., Bedi, A. S., and How, J. P. (2020). Asynchronous and parallel distributed pose graph optimization. *IEEE Robotics and Automation Letters*, 5(4):5819–5826. 17

[Triggs et al., 1999] Triggs, B., McLauchlan, P., Hartley, R., and Fitzgibbon, A. (1999). Bundle Adjustment — A Modern Synthesis. In *Proceedings of the International Workshop on Vision Algorithms, in association with ICCV*. 7, 100

[Wainwright and Jordan, 2008] Wainwright, M. J. and Jordan, M. I. (2008). *Graphical models, exponential families, and variational inference*. Now Publishers Inc. 33, 157

[Wald et al., 2020] Wald, J., Dhamo, H., Navab, N., and Tombari, F. (2020). Learning 3D Semantic Scene Graphs From 3D Indoor Reconstructions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 12, 117

[Weerasekera et al., 2018] Weerasekera, C. S., Dharmasiri, T., Garg, R., Drummond, T., and Reid, I. D. (2018). Just-in-Time Reconstruction: Inpainting Sparse Maps Using Single View Depth Predictors as Priors. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 9

[Weiss and Freeman, 2000] Weiss, Y. and Freeman, W. T. (2000). Correctness of belief propagation in Gaussian graphical models of arbitrary topology. In *Neural Information Processing Systems (NIPS)*. 13, 33, 41, 74, 158, 159, 161, 167

[Welling, 2020] Welling, M. (2020). As far as the AI can see: What we still need to build human-level intelligence. https://www.qualcomm.com/news/onq/2020/05/far-ai-can-see-what-we-still-need-build-human-level-intelligence. 3

[Whelan et al., 2015] Whelan, T., Leutenegger, S., Salas-Moreno, R. F., Glocker, B., and Davison, A. J. (2015). ElasticFusion: Dense SLAM without a pose graph. In *Proceedings of Robotics: Science and Systems (RSS)*. 8

[Wu et al., 2011] Wu, C., Agarwal, S., Curless, B., and Seitz, S. M. (2011). Multicore Bundle Adjustment. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 7, 101

[Wu et al., 2021] Wu, S., Wald, J., Tateno, K., Navab, N., and Tombari, F. (2021). Scenegraphfusion: Incremental 3d scene graph prediction from rgb-d sequences. In *CVPR*. 12, 117

[Yang and Scherer, 2019] Yang, S. and Scherer, S. (2019). Monocular object and plane slam in structured environments. *IEEE Robotics and Automation Letters*. 117

[Yang et al., 2016] Yang, S., Song, Y., Kaess, M., and Scherer, S. (2016). Pop-up slam: Semantic monocular plane slam for low-texture environments. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*. 117

[Yedidia et al., 2000] Yedidia, J. S., Freeman, W. T., Weiss, Y., et al. (2000). Generalized belief propagation. In *Neural Information Processing Systems (NIPS)*, volume 13, pages 689–695. 33, 155, 157

[Zhang et al., 2017a] Zhang, R., Zhu, S., Fang, T., and Quan, L. (2017a). Distributed very large scale bundle adjustment by global camera consensus. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 18

[Zhang et al., 2017b] Zhang, Z., Suleiman, A., Carlone, L., Sze, V., and Karaman, S. (2017b). Visual-inertial odometry on chip: An algorithm-and-hardware co-design approach. In *Proceedings of Robotics: Science and Systems (RSS)*. 100

[Zhi et al., 2019] Zhi, S., Bloesch, M., Leutenegger, S., and Davison, A. J. (2019). SceneCode: Monocular dense semantic reconstruction using learned encoded scene representations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 11

[Zhi et al., 2021] Zhi, S., Sucar, E., Mouton, A., Haughton, I., Laidlow, T., and Davison, A. J. (2021). iLabel: Interactive Neural Scene Labelling. *arXiv preprint arXiv:2111.14637*. 8

[Zhou et al., 2020a] Zhou, L., Koppel, D., Ju, H., Steinbruecker, F., and Kaess, M. (2020a). An Efficient Planar Bundle Adjustment Algorithm. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*. 115, 117

[Zhou et al., 2020b] Zhou, L., Luo, Z., Zhen, M., Shen, T., Li, S., Huang, Z., Fang, T., and Quan, L. (2020b). Stochastic bundle adjustment for efficient and scalable 3D reconstruction. In *European Conference on Computer Vision*, pages 364–379. Springer. 18

[Zhu et al., 2017] Zhu, S., Shen, T., Zhou, L., Zhang, R., Wang, J., Fang, T., and Quan, L. (2017). Parallel structure from motion from local increment to global averaging. *arXiv preprint arXiv:1702.08601*. 18

[Zhu et al., 2022] Zhu, Z., Peng, S., Larsson, V., Xu, W., Bao, H., Cui, Z., Oswald, M. R., and Pollefeys, M. (2022). NICE-SLAM: Neural Implicit Scalable Encoding for SLAM. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 11

[Zienkiewicz, 2017] Zienkiewicz, J. (2017). *Dense Monocular Perception for Mobile Robotics*. PhD thesis, Imperial College London. 7

CHAPTER 7

Appendix

7.1 Derivation of Belief Propagation as Variational Inference

In this section, we provide a derivation of Belief Propagation from the perspective of variational inference, following Yedida *et al.* [Yedidia et al., 2000]. This derivation shows that the BP equations follow from constrained minimisation of an approximate free energy known as the Bethe free energy.

We begin by writing the posterior as a product of the factors:

$$p(\mathbf{x}) = \frac{1}{Z}\prod_a f_a(\mathbf{X}_a) = \frac{1}{Z}\prod_a e^{-E(\mathbf{X}_a)}\,, \tag{7.1}$$

where $Z = \int_{\mathbf{x}} \prod_a f_a(\mathbf{X}_a)$ is the partition function which we previously assumed to be 1.

The goal of variational inference is to find a variational distribution $q(\mathbf{x})$ that approximates the posterior well by minimising the Kullback-Leibler divergence between the variational distribution and the posterior. The KL divergence is a non-negative asymmetric similarity metric that has a minimum of 0 when $p = q$.

$$\begin{aligned} KL(q||p) &= \sum_{\mathbf{x}} q(\mathbf{x}) \log \frac{q(\mathbf{x})}{p(\mathbf{x})} \\ &= \sum_{\mathbf{x}} q(\mathbf{x}) \log q(\mathbf{x}) - \sum_{\mathbf{x}} q(\mathbf{x}) \log p(\mathbf{x}) \\ &= -H_q(\mathbf{x}) - \mathbb{E}_q[\log p(\mathbf{x})] \\ &= -H_q(\mathbf{x}) - \sum_a \mathbb{E}_q[\log f_a(\mathbf{X}_a)] + \log(Z) \\ &= F(p,q) + \log(Z) \end{aligned} \tag{7.2}$$

Above, we defined the free energy:

$$F(p,q) = -H_q(\mathbf{x}) - \sum_a \mathbb{E}_q[\log f_a(\mathbf{X}_a)] \,, \tag{7.3}$$

where the first term is the negative of the entropy and the second term is known as the average energy because $-\log f_a(\mathbf{X}_a) = E(\mathbf{X}_a)$. The free energy has a minimum value of $-\log(Z)$ when $p = q$ and by minimising this free energy we are also minimising the KL divergence.

We first consider the form of the free energy for a tree. For tree graphs, the distribution $q(X)$ can be written in the form:

$$q(\mathbf{x}) = \prod_i b_i(x_i)^{1-d_i} \prod_a b_a(\mathbf{X}_a) \,, \tag{7.4}$$

where the first product is over variables and the second is over the factors. $b_i(x_i)$ is the marginal belief distribution over variable x_i, $b_a(\mathbf{X}_a)$ is the joint marginal distribution over variables $\mathbf{X}_a$ that connect to factor f_a, and d_i is the degree of variable node i (the number of nodes neighbouring node i). Plugging this into the expression for the entropy, we get:

$$H_{tree}(\mathbf{x}) = -\sum_i (1-d_i) \sum_{x_i} b_i(x_i) \log b_i(x_i) - \sum_a \sum_{X_a} b_a(\mathbf{X}_a) \log b_a(\mathbf{X}_a) \,. \tag{7.5}$$

Similarly, the average energy can be written as:

$$-\sum_a \mathbb{E}_q[\log f_a(\mathbf{X}_a)] = -\sum_a b_a(\mathbf{X}_a) \log f_a(\mathbf{X}_a) \tag{7.6}$$

Putting this together gives the free energy for tree graphs:

$$F_{tree}(\mathbf{x}) = -\sum_i (d_i - 1) \sum_{x_i} b_i(x_i) \log b_i(x_i) + \sum_a \sum_{\mathbf{X}_a} b_a(\mathbf{X}_a) \log \frac{b_a(\mathbf{X}_a)}{f_a(\mathbf{X}_a)} \,. \tag{7.7}$$

For general factor graphs with loops, the free energy $F \neq F_{tree}$. The Bethe approximation is the choice to use the free energy for a tree to approximate the free energy for arbitrary loopy graphs. The resulting approximate free energy is known as the Bethe free energy.

Belief propagation can be derived via minimisation of the Bethe free energy subject to two constraints. The first is a marginalisation constraint: $b_i(x_i) = \sum_{\mathbf{X}_a \setminus x_i} b_a(\mathbf{X}_a)$, and the second is a normalisation constraint: $\sum_i b_i(x_i) = 1$. With these constraints, we can form the Lagrangian and then set the derivatives with respect to the parameters to zero:

$$L = F_{Bethe} + \sum_i \gamma_i \Big\{ 1 - \sum_{x_i} b_i(x_i) \Big\} + \sum_a \sum_{i \in n(f_a)} \sum_{x_i} \alpha_{ai}(x_i) \Big\{ b_i(x_i) - \sum_{\mathbf{X}_a \setminus i} b_a(\mathbf{X}_a) \Big\} \tag{7.8}$$

$$\frac{\partial L}{\partial b_i(x_i)} = 0 \qquad \Rightarrow b_i(x_i) \propto \prod_{a \in n(x_i)} \exp\left(\alpha_{ai}(x_i)\right) \tag{7.9}$$

$$\frac{\partial L}{\partial b_a(\mathbf{X}_a)} = 0 \qquad \Rightarrow b_a(\mathbf{X}_a) \propto f_a(\mathbf{X}_a) \prod_{i \in n(f_a)} \exp\left(\alpha_{ai}(x_i)\right) \tag{7.10}$$

We now choose the Lagrange multiplier to be:

$$\exp(\alpha_{ai}(x_i)) = \mu_{x_i \to f_a}(x_i) = \prod_{c \in n(x_i) \setminus a} \mu_{f_c \to x_i}(x_i) \,. \tag{7.11}$$

This is the familiar variable-to-factor message equation (Equation 2.21). Substituting this into the Equations 7.9 and 7.10 yields the belief propagation fixed point equations (the first of we recognise as the belief update rule from Equation 2.13).

$$b_i(x_i) \propto \prod_{a \in n(x_i)} \mu_{x_i \to f_a}(x_i) \propto \prod_{a \in n(x_i)} \mu_{f_a \to x_i}(x_i) \tag{7.12}$$

$$b_a(\mathbf{X}_a) \propto f_a(\mathbf{X}_a) \prod_{i \in n(f_a)} \mu_{x_i \to f_a}(x_i) = f_a(\mathbf{X}_a) \prod_{i \in n(f_a)} \prod_{c \in n(x_i) \setminus a} \mu_{f_c \to x_i}(x_i) \tag{7.13}$$

Using the marginalisation condition, we can derive an equation for the messages in terms of other messages and produce the factor-to-variable message equation (as derived for trees in Equation 2.17):

$$\begin{aligned} \mu_{f_a \to x_i}(x_i) &= \sum_{\mathbf{X}_a \setminus x_i} f_a(\mathbf{X}_a) \prod_{j \in n(f_a) \setminus i} \prod_{b \in n(x_j) \setminus a} \mu_{f_b \to x_j}(x_j) \\ &= \sum_{\mathbf{X}_a \setminus x_i} f_a(\mathbf{X}_a) \prod_{j \in n(f_a) \setminus i} \mu_{x_j \to f_a}(x_j) \end{aligned} \tag{7.14}$$

This result tells us that the fixed-points of loopy belief propagation are local stationary points of the Bethe free energy and because the Bethe energy is bounded from below, BP always has a fixed point.

BP variants have been developed using more accurate or convex approximations of the free energy [Yedidia et al., 2000], however a detailed discussion of the theory behind BP is beyond the scope of this article and we refer the reader to [Wainwright and Jordan, 2008] for a in depth review.

7.2 Proof of Exactness of Gaussian Belief Propagation

Here we prove that when Gaussian belief propagation converges, it gives correct posterior means for all graph topologies while the posterior covariances are in general inexact by a quantifiable offset. This proof is an adaptation of the proof in Weiss & Freeman [Weiss and Freeman, 2000] using notation and definitions consistent with the rest of this thesis.

7.2.1 Unwrapped Computation Tree

We begin by introducing the concept of an unwrapped computation tree for a loopy graph. The unwrapped tree is the tree graph that loopy GBP is solving exactly when applied to a given loopy graph. This tree has number of levels equal to the number of iterations of loopy GBP performed on the loopy graph.

The unwrapped tree is formed by first choosing an arbitrary root node. To form each level of the tree, corresponding to a time step t of message passing in the loopy graph we do the following:

- Find all leaves of the tree. At $t = 0$ start with the root node.
- For each leaf, find the k nodes in the loopy graph that are adjacent to the corresponding node in the loopy graph.
- Add $k - 1$ nodes as children to each leaf, corresponding to all neighbours apart from the parent. This is because to send a message to the parent, the node collects messages from all other neighbouring nodes.

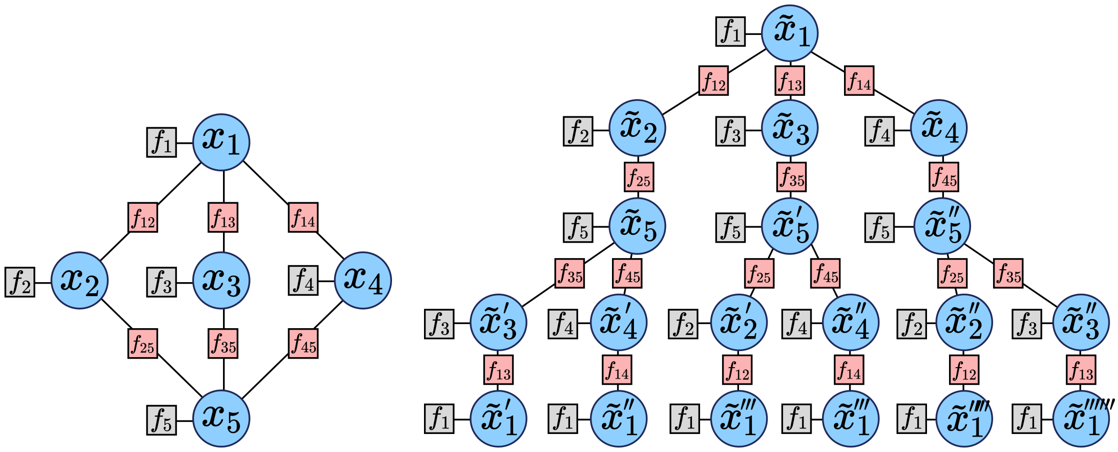
An example of a loopy graph and its corresponding unwrapped computation tree for 4 steps of message passing with x_1 chosen as the root node is shown in Figure 7.1. Using a breadth first ordering starting from node x_1 in the loopy graph and $\tilde{x_1}$ in the unwrapped tree, we can define the stacked vectors corresponding to all of the variables in the loopy graph:

$$\mathbf{x} = [x_1, x_2, x_3, x_4, x_5]^\top \in \mathbb{R}^{N_l} , \tag{7.15}$$

and in the first 3 levels of the unwrapped tree:

$$\tilde{\mathbf{x}} = [\tilde{x}_1, \tilde{x}_2, \tilde{x}_3, \tilde{x}_4, \tilde{x}_5, \tilde{x}_5', \tilde{x}_5'', \tilde{x}_3', \tilde{x}_4', \tilde{x}_2', \tilde{x}_4'', \tilde{x}_2'', \tilde{x}_3'', \ldots]^\top \in \mathbb{R}^{N_u} . \tag{7.16}$$

N_u and N_l are the number of nodes in the unwrapped and loopy graphs respectively.



(a) Loopy graph (b) Unwrapped tree

Figure 7.1: The loopy graph and the corresponding unwrapped computation tree for 4 steps of message passing. All nodes and factors in the unwrapped tree are copies of the nodes and factors in the loopy graph. All unary and binary factors are drawn for clarity. This figure is adapted from Weiss & Freeman [Weiss and Freeman, 2000].

We now introduce a matrix $\mathtt{O} \in \mathbb{R}^{N_u \times N_l}$ which relates nodes in the loopy graph to nodes in the unwrapped graph. In our example, filling in only the non-zero elements, the matrix $\mathtt{O}$ is:

$$\mathtt{O} = \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & 1 & \\ & & & & 1 \\ & & & & 1 \\ & & & & 1 \\ & & & 1 & \\ & & 1 & & \\ & 1 & & & \\ & & & 1 & \\ & & 1 & & \\ & 1 & & & \\ 1 & & & & \\ 1 & & & & \\ & & \vdots & & \end{bmatrix}. \tag{7.17}$$

7.2.2 Loopy and Unwrapped information parameters

We begin by defining the Gaussian information form parameters for the joint posterior in the loopy graph. In general, this Gaussian posterior can be expressed as:

$$p(\mathbf{x}) = \mathcal{N}^{-1}(\mathbf{x}; \boldsymbol{\eta}, \Lambda) \,, \tag{7.18}$$

where $\boldsymbol{\eta} \in \mathbb{R}^{N_l}$ and $\Lambda \in \mathbb{R}^{N_l \times N_l}$. In our example, the information vector and precision matrix can be divided up as:

$$\boldsymbol{\eta} = [\eta_1, \eta_2, \eta_3, \eta_4, \eta_5]^\top \tag{7.19}$$

$$\Lambda = \begin{bmatrix} \Lambda_{11} & \Lambda_{12} & \Lambda_{13} & \Lambda_{14} & 0 \\ \Lambda_{21} & \Lambda_{22} & 0 & 0 & \Lambda_{25} \\ \Lambda_{31} & 0 & \Lambda_{33} & 0 & \Lambda_{35} \\ \Lambda_{41} & 0 & 0 & \Lambda_{44} & \Lambda_{45} \\ 0 & \Lambda_{52} & \Lambda_{53} & \Lambda_{54} & \Lambda_{55} \end{bmatrix} . \tag{7.20}$$

Note that the precision matrix reflects the conditional independence structure of the loopy graph.

We can also define the covariance form parameters of the loopy graph, which for now we define by the relation to the information form:

$$\boldsymbol{\mu} = \Lambda^{-1}\boldsymbol{\eta} \tag{7.21}$$

$$\Sigma = \Lambda^{-1} \tag{7.22}$$

Let us now define the joint Gaussian distribution over all variables in the unwrapped tree:

$$p(\tilde{\mathbf{x}}) = \mathcal{N}^{-1}(\tilde{\mathbf{x}}; \tilde{\boldsymbol{\eta}}, \tilde{\Lambda}) \,, \tag{7.23}$$

where $\tilde{\boldsymbol{\eta}} \in \mathbb{R}^{N_u}$ and $\tilde{\Lambda} \in \mathbb{R}^{N_u \times N_u}$.

All nodes in the unwrapped tree apart from the leaf nodes have the same neighbours as their copied node in the loopy graph. As a result, elements in $\tilde{\boldsymbol{\eta}}$ corresponding to non-leaf nodes take the same value as the element in $\boldsymbol{\eta}$ of the loopy graph corresponding to the copied node. We can therefore relate $\tilde{\boldsymbol{\eta}}$ and $\boldsymbol{\eta}$ as:

$$\tilde{\boldsymbol{\eta}} + \mathbf{e} = \mathtt{O}\boldsymbol{\eta} \,. \tag{7.24}$$

Here the vector $\mathbf{e}$ describes the error between $\tilde{\boldsymbol{\eta}}$ and $\boldsymbol{\eta}$ and is non-zero apart from the final L elements, where L is the number of leaf nodes in the unwrapped tree. In our example, the first two

non-zero elements of this error vector are:

$$\mathbf{e}(N_u - L + 1) = -\boldsymbol{\eta}_{12}(1) - \boldsymbol{\eta}_{13}(1) \tag{7.25}$$

$$\mathbf{e}(N_u - L + 2) = -\boldsymbol{\eta}_{12}(1) - \boldsymbol{\eta}_{14}(1)\,, \tag{7.26}$$

where we are indexing from 1 so $\boldsymbol{\eta}_{12}(1)$ is the first element of the precision matrix of the Gaussian factor connecting x_1 and x_2.

Next, we would like to produce a similar relationship between the precision matrices in the loopy and unwrapped graphs, Λ and $\tilde{\Lambda}$. As before, all nodes in the unwrapped tree apart from the leaf nodes have the same non-zero elements in their corresponding row of the precision matrix $\tilde{\Lambda}$. These non-zero elements are equal to the corresponding values in Λ. The leaf nodes are missing some neighbours compared to the corresponding node in the loopy graph and so their row in $\tilde{\Lambda}$ has only 2 non-zero entries for the dependence of the variable on itself and its parent. We can relate Λ and $\tilde{\Lambda}$ as:

$$\tilde{\Lambda}\mathtt{O} + \mathtt{E} = \mathtt{O}\Lambda\,. \tag{7.27}$$

The matrix $\mathtt{E} \in \mathbb{R}^{N_u \times N_l}$ is non-zero only in the final L rows. In our example the first two non-zero rows or this matrix are:

$$E(N_u - L + 1) = \begin{bmatrix} 0 & -\Lambda_{12} & 0 & -\Lambda_{14} & 0 \end{bmatrix} \tag{7.28}$$

$$E(N_u - L + 2) = \begin{bmatrix} 0 & -\Lambda_{12} & -\Lambda_{13} & 0 & 0 \end{bmatrix}\,. \tag{7.29}$$

Summarising, we have the following relationships between the information parameters of the loopy and unwrapped graphs:

$$\tilde{\boldsymbol{\eta}} + \mathbf{e} = \mathtt{O}\boldsymbol{\eta} \tag{7.30}$$

$$\tilde{\Lambda}\mathtt{O} + \mathtt{E} = \mathtt{O}\Lambda \tag{7.31}$$

7.2.3 Loopy and Unwrapped marginal covariance parameters

Having derived the relationship between loopy and unwrapped information parameters in Equations 7.30 and 7.31, we now want to find a relationship between the marginal mean and covariance of node x_1 in the loopy graph and the root node $\tilde{x}_1$ in the unwrapped tree. Following Weiss & Freeman [Weiss and Freeman, 2000], we derive these relationships below.

Mean parameters

We would like to relate the marginal mean $\boldsymbol{\mu}(1)$ of node x_1 in loopy graph, to the marginal mean $\tilde{\boldsymbol{\mu}}(1)$ of node $\tilde{x}_1$ in the unwrapped tree. We begin with the relationship for the loopy graph:

$$\boldsymbol{\eta} = \Lambda\boldsymbol{\mu} \; . \tag{7.32}$$

We then left multiply by $\mathtt{O}$:

$$\mathtt{O}\boldsymbol{\eta} = \mathtt{O}\Lambda\boldsymbol{\mu} \tag{7.33}$$

and then substitute in unwrapped quantities using Equations 7.30 and 7.31:

$$\tilde{\boldsymbol{\eta}} + \mathbf{e} = (\tilde{\Lambda}\mathtt{O} + \mathtt{E})\boldsymbol{\mu} \; . \tag{7.34}$$

Lastly substituting in $\tilde{\boldsymbol{\eta}} = \tilde{\Lambda}\tilde{\boldsymbol{\mu}}$ and rearranging yields:

$$\begin{aligned} \tilde{\Lambda}\tilde{\boldsymbol{\mu}} + \mathbf{e} &= (\tilde{\Lambda}\mathtt{O} + \mathtt{E})\boldsymbol{\mu} && (7.35)\\ \tilde{\Lambda}\tilde{\boldsymbol{\mu}} &= \tilde{\Lambda}\mathtt{O}\boldsymbol{\mu} + \mathtt{E}\boldsymbol{\mu} - \mathbf{e} && (7.36)\\ \tilde{\boldsymbol{\mu}} &= \mathtt{O}\boldsymbol{\mu} + \tilde{\Lambda}^{-1}(\mathtt{E}\boldsymbol{\mu} - \mathbf{e}) \; . && (7.37) \end{aligned}$$

Taking the first row, and substituting $\tilde{\Sigma} = \tilde{\Lambda}^{-1}$ gives an equation relating the mean of $\tilde{x}_1$ to the mean of x_1:

$$\tilde{\boldsymbol{\mu}}(1) = \boldsymbol{\mu}(1) + \tilde{\Sigma}_{(1)}(\mathtt{E}\boldsymbol{\mu} - \mathbf{e}) \; , \tag{7.38}$$

where $\tilde{\Sigma}_{(1)}$ is the first row of $\tilde{\Sigma}$.

Looking at this relation, we see that the difference between the mean of $\tilde{x}_1$ in the unwrapped tree and the mean of x_1 in the loopy graph is the dot product of the first row in the covariance matrix and an error vector. As the $\mathtt{E}$ and $\mathbf{e}$ are non-zero only in the last L rows, we can express this difference as:

$$\begin{aligned} \delta\mu &= \tilde{\Sigma}_{(1)} \left(\mathtt{E}\boldsymbol{\mu} - \mathbf{e}\right) && (7.39)\\ &= \sum_{j > N_u - L} \tilde{\Sigma}_{1j} \left(\mathtt{E}_{(j)}\boldsymbol{\mu} - \mathbf{e}_j\right) \; . && (7.40) \end{aligned}$$

The difference therefore depends only on the covariances between the root node and the leaf nodes in the unwrapped tree. If these covariances tend to zero, then the mean of $\tilde{x}_1$ tends towards the mean of x_1 in the loopy graph. In other words, if the covariances between the root node and leaf nodes decrease rapidly to zero, then the belief propagation means converge and the means are exact. Intuitively, these covariances tend to zero when there are strong unary constraints in the graph that reduce the correlation between distant nodes in the graph.

Covariance parameters

Moving on to the covariance parameters, we would like to relate the marginal covariance Σ_{11} of node x_1 in loopy graph, to the marginal mean $\tilde{\Sigma}_{11}$ of node $\tilde{x}_1$ in the unwrapped tree. Starting with the identity for the loopy graph:

$$\Lambda\Sigma = I \,, \tag{7.41}$$

and again left multiplying by $\mathtt{O}$ and then substituting in Equation 7.31:

$$\mathtt{O}\Lambda\Sigma = \mathtt{O} \tag{7.42}$$

$$(\tilde{\Lambda}\mathtt{O} + \mathtt{E})\Sigma = \mathtt{O} \,. \tag{7.43}$$

Expanding the bracket, left multiplying by $\tilde{\Sigma}$ and then rearranging gives:

$$\tilde{\Sigma}^{-1}\,\mathtt{O}\,\Sigma + \mathtt{E}\,\Sigma = \mathtt{O} \tag{7.44}$$

$$\mathtt{O}\,\Sigma + \tilde{\Sigma}\,\mathtt{E}\,\Sigma = \tilde{\Sigma}\,\mathtt{O} \,. \tag{7.45}$$

Taking the first row and column entry in the above equation yields:

$$\Sigma_{11} + \sum_j \tilde{\Sigma}_{1j}\,(\mathtt{E}\,\Sigma)_{j1} = \sum_k \tilde{\Sigma}_{1k}\,\mathtt{O}_{k1} \,. \tag{7.46}$$

We can split the term on the right hand side as:

$$\sum_k \tilde{\Sigma}_{1k}\,\mathtt{O}_{k1} = \tilde{\Sigma}_{11} + \sum_{k\in C_1} \tilde{\Sigma}_{1k} \,, \tag{7.47}$$

where C_1 are the copies of node x_1 in the unwrapped tree apart from $\tilde{x}_1$. Rearranging we then find:

$$\tilde{\Sigma}_{11} = \Sigma_{11} + \sum_j \tilde{\Sigma}_{1j}\,(\mathtt{E}\,\Sigma)_{j1} - \sum_{k\in C_1} \tilde{\Sigma}_{1k} \,. \tag{7.48}$$

Now we see that the difference between the marginal covariance of x_1 in the loopy graph and $\tilde{x}_1$ in the unwrapped tree depends on two terms. The second term on the right hand side features the matrix $\mathtt{E}$ and so is non-zero only for the last L elements corresponding to the leaf nodes. Like the offset for the marginal means, this second term depends on the covariances between the root node and the leaf nodes. The third term on the right hand side is a sum of the covariances between the root node in the unwrapped tree $\tilde{x}_1$ and all other copies of x_1 in the unwrapped tree such as $\tilde{x}_1'$, $\tilde{x}_1''$ etc. We can therefore say that is the covariances between the root node and the leaf node decrease rapidly to zero, then the belief propagation covariances converge and are equal to the correct variances minus the summed covariances between $\tilde{x}_1$ and all other copies of x_1 in the unwrapped tree.

At this point, it is useful to consider exactly what we have shown. We have shown that if the covariances between the root node and the leaf nodes of the unwrapped true decrease rapidly to zero then GBP converges, the means are exact, and the variances are equal to the correct variances minus the summed covariances between $\tilde{x}_1$ and all other copies of x_1 in the unwrapped tree. Note that we have not described the conditions under which the covariances between the root node and the leaf nodes of the unwrapped true decrease rapidly to zero nor have we made any statements about the means and variances of GBP in general when it converges.

7.2.4 Fixed Points of Loopy Belief Propagation

We now aim to evaluate the properties of fixed points of GBP. We can view GBP as an operator F that transforms a list of all the messages and transforms it to another list of messages:

$$m^{(t+1)} = Fm^{(t)} \,, \tag{7.49}$$

where m the list of all factor-to-variable messages $\mu_{f_s \to x_j}$. In this light, GBP can be thought of as a way of finding a solution m^* to the fixed point equation:

$$m^* = Fm^* \,. \tag{7.50}$$

Here, we ask the question, if we find a fixed point of GBP with messages m^*, then how do the beliefs relate to the correct beliefs?

We begin with the **claim**: If m^* is a fixed point of GBP message passing then the means based on that fixed point are exact.

To prove this claim we first introduce the *modified* unwrapped tree for a given loopy graph and fixed point messages m^*. The modified unwrapped tree is constructed as the same way as previously such that all non-leaf nodes have the same statistical relationships with their neighbours as the corresponding nodes in the original graph. The difference is all nodes in the modified unwrapped tree have the same beliefs as the beliefs in the loopy graph derived from the fixed point m^*. This can be achieved with a simple modification to all the unary factors on the leaf nodes in the unwrapped tree. These unary factors are modified such that the message they send to the leaf node is equal to the product of the original message from the unary factor and all fixed point messages from its neighbours apart from its parent node. This process of modifying the unary factors for all leaf nodes is shown in Figure 7.2.

With this modification, all leaf nodes now send their parents a message from m^*. Since all non-leaf nodes have the same statistical relationship to their neighbours as the corresponding nodes in

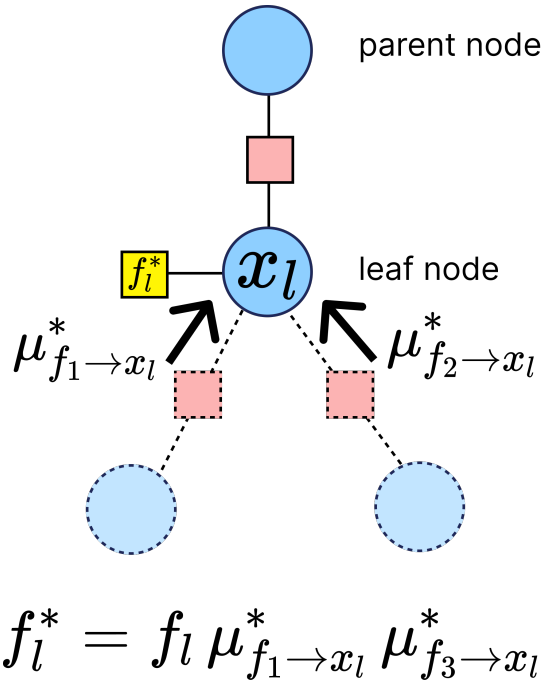


Figure 7.2: To produce the modified unwrapped computation tree the unary factor for each leaf node (highlighted in yellow) is modified. Each unary factor is modified such that the modified message it sends f_l^* is the product of the original message f_l and the fixed point messages $\mu^*_{f \to x_l}$ from all neighbouring nodes apart from its parent node. The dashed parts of the graph represent the messages that are represented in the modified yellow unary factor.

the loopy graph, all messages in this modified unwrapped tree are therefore identical to the fixed point messages m^*.

Having constructed this modified unwrapped tree in which all messages are the fixed point messages m^*, we now want to determine what the marginal means are in this tree. We begin by relating the means in the modified unwrapped tree to the marginal means in the loopy tree with fixed point messages:

$$\tilde{\boldsymbol{\mu}} = \mathsf{O}\boldsymbol{\mu}_0 \,, \tag{7.51}$$

where $\boldsymbol{\mu}_0(i)$ is the posterior mean at node i under messages m^*. This equality holds because all nodes in the modified unwrapped tree have identical incoming messages to the corresponding node in the loopy graph under m^*, meaning that the beliefs are equal.

We also know that $\tilde{\boldsymbol{\mu}}$ satisfies:

$$\tilde{\Lambda}\,\tilde{\boldsymbol{\mu}} = \tilde{\boldsymbol{\eta}} \,, \tag{7.52}$$

for the modified unwrapped tree. Substituting in Equation 7.51:

$$\tilde{\Lambda}\,\mathsf{O}\,\boldsymbol{\mu}_0 = \tilde{\boldsymbol{\eta}} \,. \tag{7.53}$$

We now use notation $[A]_m$ to denote taking the first m rows of a matrix A, and use the identity $[AB]_m = [A]_m B$. Taking the first m rows of Equation 7.53:

$$[\tilde{\Lambda}\ \mathtt{0}]_m\, \boldsymbol{\mu}_0 = [\tilde{\boldsymbol{\eta}}]_m\ . \tag{7.54}$$

Since all non-leaf nodes in the modified unwrapped graph have the same relationships with neighbouring nodes as the corresponding nodes in the loopy graph, we can write the following equations for all $m < N_u - L$:

$$[\tilde{\boldsymbol{\eta}}]_m = [\mathtt{0}\boldsymbol{\eta}]_m \tag{7.55}$$

$$[\tilde{\Lambda}\mathtt{0}]_m = [\mathtt{0}\Lambda]_m\ . \tag{7.56}$$

Note that these are the same as Equations 7.30 and 7.31, but only for the non-leaf nodes in the unwrapped tree. Substituting into Equation 7.54 and using our identity for the first m rows:

$$[\mathtt{0}\Lambda]_m\, \boldsymbol{\mu}_0 = [\mathtt{0}\boldsymbol{\eta}]_m \tag{7.57}$$

$$[\mathtt{0}]_m\, \Lambda\, \boldsymbol{\mu}_0 = [\mathtt{0}]_m \boldsymbol{\eta}\ . \tag{7.58}$$

This equality in Equation 7.58 holds for any $m < N_u - L$, but we can unwrap the tree to an arbitrarily large size such that $[\mathtt{0}]_m$ has N_l independent rows. In other words, we unwrap the tree until all all nodes in the loopy graph appear at least once in the non-leaf nodes of the modified unwrapped tree. At this point, Equation 7.58 is an equation with m separate equalities for all m non-leaf nodes in the unwrapped tree. If we ignore the repeated rows of $[\mathtt{0}]_m$, and choose only the rows such that we replace $[\mathtt{0}]_m$ with $I \in \mathbb{R}^{N_l \times N_l}$, then we see that the following must hold:

$$\Lambda\, \boldsymbol{\mu}_0 = \boldsymbol{\eta}\ . \tag{7.59}$$

This tells us that these means $\boldsymbol{\mu}_0$ derived from the fixed point messages are exact.

7.2.5 Summary of findings

To summarise our findings in this section:

- If the covariances between the root node and the leaf nodes in the unwrapped tree decrease to zero rapidly enough, then GBP converges, the belief means are exact and the belief variances are equal to the exact variances minus the summed covariances between the root node and copies of that same node in the unwrapped tree.
- If GBP converges, then the belief means are guaranteed to be exact.

- We have not shown that GBP always converges. We have not shown under what conditions the covariances in the unwrapped tree decrease rapidly to zero. We have not shown what the belief variance compute when GBP converges.

We refer the reader to Weiss & Freeman [Weiss and Freeman, 2000] for proof that these findings hold for vector valued nodes.