18

The Computational Structure of Spatial AI Systems

Andrew J. Davison

SLAM algorithms, which map an environment and estimate a device's position within it, have enabled a range of important new products and capabilities which are already having an impact on the world, in areas such as industrial automation, home robotics, drones and virtual/augmented reality (VR/AR). However, we believe that this is just the start of something much bigger: the capability for artificial devices to build genuinely rich but efficient representations of their surroundings which enable them to interact with them in generally intelligent ways. We define this capability as *Spatial AI*, a term first coined in 'FutureMapping' [254].

In this chapter, we argue that Spatial AI capabilities are an ongoing evolution of the SLAM research that has been presented throughout this handbook. We predict and analyse the capabilities and computational structure of Spatial AI systems over the coming years, and in particular analyse the impact that machine learning and new sensing and computing hardware will have.

18.1 From SLAM to Spatial AI

18.1.1 Intelligent Embodied Devices Need Spatial AI

The goal of a Spatial AI system is not abstract scene understanding, but continuously to build the right representations to enable real-time interpretation and general, intelligent action. The design of such a system will be framed at one end by task requirements on its performance, and at the other end by constraints imposed by the setting of the device in which it is to be used.

Let us consider the example of a mass market household robot product of the future, which is set the tasks of monitoring, cleaning, and tidying a set of rooms. The robot may be a humanoid, or other platform with general movement and manipulation capabilities. Its task requirements will include cleaning or sweeping surfaces and knowing when they are clean; recognising, tidying, moving and manipulating objects; and dealing promptly and respectfully with humans by moving out of the way or assisting them. Meanwhile, its Spatial AI system, comprising sensors, proces-

sors and algorithms, will be constrained by factors including price, aesthetics, size, safety, and power usage, which must fit within the range of a consumer product.

As a second example, this time from the domain not of autonomous robotics but wearable assistive devices for humans (which with pleasing symmetry we may refer to as "IA", Intelligence Augmentation), we imagine a future augmented reality system which can provide its wearer with a robust spatial memory of all of the places, objects and people they have encountered, enabling things such as easily finding lost objects, and the placing of virtual notes or other annotations on any world entity. This capability could enable the full context of a person's life to be available automatically to an always-on Large Language Model assistant, and perhaps general amplification of that person's intelligence capabilities. To achieve wide adoption, the device should have the size, weight,v and form factor of a standard pair of spectacles (65g), and operate all day without needing a battery charge (<1W power usage).

There is currently a big gap between what such powerful Spatial AI systems need to do in useful applications, and what can be achieved with current technology under real-world constraints. There is much promising research on the algorithms and technology needed, but robust performance is still difficult even when expensive, bulky sensors and unlimited computing resources are available. The gap between reality and desired performance becomes much more significant still when the constraints on real products are taken into account. In particular, the size, cost, and power requirements of the computer processors currently enabling advanced robot vision are very far from fitting the constraints imposed by these applications we envisage.

18.1.2 Scene Representation and World Models

We believe that the key to Spatial AI is *representation*. A representation is a set of values held in a computer's memory which represent the state of an embodied device and the world around it in a way that is persistent, but adaptable, and always changing as the device acquires new information about the world, or the state of the world itself changes. Also, the device may choose to abstract or even "forget" parts of its representation for reasons of efficiency.

The term 'world model' has recently become popular in machine learning research [430], and this term has very much the same meaning as scene representation. Many in machine learning consider world models a rather new domain of research, because most current trained neural network architectures are stateless, simply processing inputs and turning them deterministically into outputs. There is significant current research in robot learning on training such networks to perform tasks in an "end-to-end" manner, where inputs from sensors are turned directly into actions, in domains from autonomous driving to the manipulation of objects. These approaches are often extremely effective, but struggle to achieve long-horizon tasks.

World model research in neural networks aims to equip them with persistent memory modules which represent the changing state of a system and its environment, allowing simulation and longer-term planning.

Despite this recent new interest, the ideas of world models and scene representation of course have a long history in the more general world of AI, and SLAM research in robotics is the most clear example of where aiming to build persistent scene representations enables consistent, long-horizon behaviour. Most obviously, SLAM maps enable in reliable long-term navigation to multiple waypoints, or guaranteed area coverage, in applications such as autonomous drones or robot floor cleaners.

As SLAM expands beyond building minimal scene representations for localisation towards dense, semantic and even object-level and physically predictive scene models, it will enable devices to simulate and plan in general ways. Model-predictive control (MPC) can take place in an incrementally built but accurate, up-to-date simulation of a robot's environment, allowing optimal and creative behaviour, even for object manipulation.

Many machine learning researchers would assume that a 'world model' is something much more abstract then an explicit 3D map of a scene that is the typical output of a SLAM system, and we agree that there is much research to be done here on which world representations will enable true spatial intelligence while remaining efficient and practical. However, there are many advantages to representations which are rather explicit about the geometry and objects in a scene, in terms of efficiency, composability, interpretability, and generality.

We therefore make the following hypothesis: *When a device must operate for an extended period of time, carry out a wide variety of tasks (not all of which are necessarily known at design time), and communicate with other entities including humans, its Spatial AI system should build a general and persistent scene representation which is close to metric 3D geometry, at least locally, and is human understandable.* To be clear, this definition leaves a lot of space for many choices about scene representation, with both learned and designed elements, but rules out algorithms which make use of very specific task-focused representations, or completely abstract "black box" world models.

Our second hypothesis is that: *The usefulness of a Spatial AI system for a wide range of tasks is well represented by a relatively small number of performance measures.* That is to say that whether the system is to be used to guide the autonomous flight of a delivery drone in a tight space, or a household robot to tidy a room, or to enable an augmented reality display to add synthetic objects to a scene, then even though these applications certainly have different requirements and constraints, their Spatial AI systems will be largely similar, with differences specified by a small number of performance parameters. The obvious parameters describe aspects like global device localisation accuracy and update latency, but we believe that there are other metrics which will be more meaningful for applications, like distance to

surface contact prediction accuracy, object identification accuracy, or tracking robustness. We will discuss performance metrics further in Section 18.8.

For the purposes of the rest of this chapter, we will therefore call such a module which incrementally builds and maintains a generally useful, close to metric scene representation, in real-time and from primarily visual input, and with quantifiable performance metrics, a *Spatial AI system*.

Judea Pearl, recently discussing efficient situated learning and the need to reason about causation [863], argued that "what humans possessed that other species lacked was a mental representation, a blue-print of their environment which they could manipulate at will to imagine alternative hypothetical environments for planning and learning". We believe that Spatial AI is similarly what will set apart advanced robots and devices which can take embodied intelligence to the next level. Further, there is evidence that the spatial reasoning capabilities of humans are used more generally as a fundamental tool for intelligence, by organising thinking in spatial ways, even for abstract concepts [78]; this may also be true for artificial systems.

18.1.3 SLAM is Evolving into Spatial AI

Research in SLAM, and especially visual SLAM, has long been driven by live demonstrations, with real-time visualisations, and the release of open source code. These live demos (of systems such as MonoSLAM, PTAM, KinectFusion, LSD-SLAM, SVO, ORB-SLAM, iMAP, Gaussian Splatting SLAM) have arguably been the most important markers of progress rather than the dataset results prominent in computer vision. This is because datasets often only capture limited aspects of a system's performance (especially accuracy) while not highlighting robustness, efficiency, or flexibility, which are equally important and are readily assessed from a short real-time demo.

The level of scene representation that has been possible in real-time visual SLAM has gradually improved, from sparse features to dense maps and now increasingly semantic labels. Commercial SLAM provider Slamcore has referred to sparse localisation, dense mapping, and semantic labelling as Levels 1, 2 and 3 capabilities respectively. Beyond this is perhaps the final destination of advances in scene representation, scene graphs with accurately segmented and physically simulated object instances making up geometry hierarchically (*cf.* Chapter 16 for further discussion). These are ongoing steps along SLAM's evolution towards Spatial AI. Observing the steady and consistent progress of SLAM over more than 30 years, we have become confident that the operation of current and foreseeable SLAM systems is the best guide we have for the algorithmic structure of future Spatial AI.

We would like to emphasize the despite this level-based interpretation, the way that SLAM systems progress is usually not by the pure addition of layers to an already-working system. Rather, each change of representation causes the whole

system to be re-designed. Once a dense scene map is available, for instance, it can be used to implement accurate and robust direct tracking for camera localisation; or semantic scene labelling can improve dense reconstruction quality. This kind of thinking is inherent to real-time systems research, where every part of a system affects every other part in a closed loop.

We therefore believe strongly that ongoing SLAM research is the best model we have for the development of Spatial AI.

18.2 Overall Computational Structure

As a sensor platform carrying at least one camera and other sensors moves through the world, its motion either under active control or provided by another agent, the essential way that a Spatial AI system of any variety works can be summarised as follows:

1 Our system will comprise outward looking sensors such as cameras, and supporting sensors such as an IMU, closely integrated with a processing architecture in a low-power package which is embedded in a mobile entity such as a robot or AR system.
2 In real-time, the system must maintain and update a world model, with geometric and semantic information, and estimate its position within that model, from primarily or only measurements from its on-board sensors.
3 The system should provide a wide variety of task-useful information about 'what' is 'where' in the scene. Ideally, it will provide a full semantic level model of the identities, positions, shapes and motion of all of the objects and other entities in the surroundings.
4 The representation of the world model will be close to metric, at least locally, to enable rapid reasoning about arbitrary predictions and measurements of interest to an AI or IA system.
5 It will probably retain a maximum quality representation of geometry and semantics only in a focused manner; most obviously for the part of the scene currently observed and relevant for near-future interaction. The rest of the model will be stored at a hierarchy of residual quality levels, which can be rapidly upgraded when revisited.
6 The system will be generally alert, in the sense that every incoming piece of visual data is checked against a forward predictive scene model: for tracking, and for detecting changes in the environment and independent motion. The system will be able to respond to changes in its environment.

The key computational quality of this approach is its *closed loop* nature, where the world model is persistent and incrementally updated, representing in an abstracted form all of the useful data which has been acquired to date, and is used in the real-time loop for data association and tracking. This is in contrast with vision systems

which perform incremental estimation (such as pure visual odometry, where camera motion is estimated from frame to frame but long term data structures are not retained), or which can only achieve global consistency with off-line, after-the-fact, batch computation. That is not to say that in a closed loop Spatial AI system every computation should happen at a fixed rate, but more importantly that it should be available when needed to allow real-time operation of the whole system to continue without pauses.

The next element of our high-level thinking is to identify the core ways that we can achieve all of this with high performance but low power requirements. We believe that the key to efficient processing in Spatial AI is to identify the graphs of computation and data movement in the algorithms required, and as far as possible to make use of or design processing hardware which has the same properties, with the particular goal of minimising data movement around the architecture. We need to identify the following things: What is stored where? What is processed where? What is transmitted where, and when?

We will not attempt here to draw strong parallels with neuroscience, but clearly there is much scope for relating the ideas and designs we discuss here in artificial Spatial AI systems with the vision and spatial reasoning capabilities and structures of biological brains. The human brain apparently achieves high performance, fully 'embedded' semantic and geometric vision using less than 10 Watts of power, and certainly its structures have properties which mirror some of the concepts we discuss. We will leave it to other authors to analyse the relationships further; this is mostly due to our lack of expertise in neuroscience, but also partly to a belief that while artificial vision systems clearly still have a great deal to learn from biology, they need not be designed to replicate the performance or structure of brains. We consider the Spatial AI computation problem purely from the engineering point of view, with the goal of achieving the performance we need for applications while minimising resources. It should surely not be surprising that some aspects of our solutions should mimic those discovered by biological evolution, while in other respects we might find quite different methods due to two contrasts: firstly between the incremental 'has to work all of the time' design route of evolution and the increased freedom we have in AI design; and secondly between the wetware and hardware available as a computational substrate.

Before making this computational analysis more concrete, in terms of proposing a generic design for Spatial AI implementations, we need to consider two important topics: (i) state estimation vs. machine learning, and (ii) the landscape of future processor and sensor hardware. We will discuss these in the next two sections.

18.3 State Estimation and Machine Learning in Spatial AI

Much of this handbook covers approaches to SLAM which are based on human-designed algorithms and representations; predominantly 'state estimation' methods

based on probability theory. In the later chapters, we see the rise of machine learning techniques of many types. The key machinery of machine learning in SLAM is the artificial neural network. SLAM is notable within the broader picture of artificial intelligence in that it has not yet been completely dominated by ML. The best performing systems for localisation and sparse mapping are still mostly hand-designed, and dense and semantic SLAM systems use a balance, where ML is used to replace part of or add to the functionality of state estimation. However, the gradual rise of ML techniques to cover more of the functionality of Spatial AI has been undeniable.

In some of the first uses of ML in SLAM, a learned module was used to add a layer of functionality but did not interact directly with the state estimation components. An example is SemanticFusion [754], which used CNN-based semantic image segmentation to add labels to a the dense geometric map produced by ElasticFusion [1181]. Soon though, it became clear that learned components could play important roles in the core geometric estimation loop of SLAM, and this has taken several key forms, including:

1 Networks trained to predict geometric properties from single images, such as depth maps, normal maps, or object segmentation.
2 Components for updating standard geometric estimates, like poses and point clouds, as used in DROID-SLAM; these networks often perform operations such as matching or geometric optimisation.
3 The latest networks which produce higher-quality geometric predictions from multi-image input, such as DUSt3R and VGGT.

Although neural networks can directly predict geometry, such as depth maps from a single image, it has proven difficult to use these outputs in SLAM because single-view predictions often contain large systematic errors, and have poorly understood uncertainty. A promising branch of this approach is to train networks to predict somewhat weaker properties than raw depth, such as coded depth CodeSLAM [97] or depth covariance [277], allowing low-dimensional multi-view optimisation to perform consistent dense geometry fusion.

Perhaps the end goal of this single-view research is to develop a network which can turn an input image into a reliable set of semantic, object-like entities which can be then used to populate an efficient, abstract 3D object map or scene graph which is subject to ongoing, dynamic multi-view optimisation and updating. This approach was pioneered in SLAM by SLAM++ [969], which built an efficient scene graph map directly at the level of recognised 3D objects, though with the limitation that a pre-defined 3D model of every object was needed. A modern generalisation of this approach is SuperPrimitives [751] which uses generic single-image object segmentation and local reconstruction to enable a long-term map of arbitrary 3D object chunks.

The appeal of machine learning in Spatial AI has perhaps two key aspects. First,

neural networks can be trained to carry out tasks have proven very difficult to design by hand, such as scene labelling or depth prediction. Second, they compress iterative computation into an efficient, computationally simple form, which can run feed-forward and often fast on modern processors.

We should naturally ask the question of whether the whole of SLAM and Spatial AI will ultimately be performed end-to-end by a neural network, with no hand-designed representations or state estimation at all. Of course many researchers believe in this idea, and there are increasingly impressive pieces of work from the early demonstrations of end-to-end visual odometry DeepVO [1161, 1294] to networks such as VGGT which outputs 3D point clouds directly from a set of unposed images.

This work will surely continue to improve, but we believe that hybrid methods combining learning and state estimation will continue to be preferable for the foreseeable future, for several reasons. First, the issue of understanding the uncertainty of network predictions remains, and we will need to keep the ability to update and fuse data into models. If we have a network which can produce a 3D scene model from 100 images, how should we update it when one more image is available? Surely we should not need to run the whole network again with 101 images. As soon as we accept that long-term representation and fusion is needed, then we need the tools of probabilistic state estimation, and there is a strong motivation towards modular scene representations.

Ultimately, the Spatial AI systems of the mid-term future will surely have learned and designed components so tightly integrated that it is hard to tell which is which; and this position may be reached either by adding more and more learned modules to a designed system, or by adding structure to a purely learned system. What still remains important in either case, and at the centre of our interest in this chapter, is the storage and computational structure of the whole system.

18.4 The Future Landscape of Processor and Sensor Hardware

18.4.1 Processors

SLAM research was for many years conducted in the era when single core CPU processors could reliably be counted on to double in clock speed, and therefore serial processing capability, every 1–2 years. In recent years this has stopped being true. The strict definition of Moore's Law, describing the rate of doubling of transistor density in integrated circuits, has continued to hold into the current era. What has changed is that this can no longer be proportionally translated into serial CPU performance, due to the breakdown of another less well known rule of thumb called Dennard Scaling, which states that as transistors get smaller their power density stays constant. When transistors are reduced down to today's nanometre sizes, not so far from the size of the atoms they are made from, they leak current and heat

up. This 'power wall' limits the clock speed at which they can reasonably be run without overheating uncontrollably to something around 4GHz.

Processor designers must therefore increasingly look towards alternative means than simply faster clocks to improve computation performance. The processor landscape is becoming much more complex, parallel and specialised, as described well in Sutter's online article 'Welcome to the Jungle' [1067]. Processor design is becoming more varied and complex even in 'cloud' data centres. Pressure to move away from CPUs is even stronger in embedded applications like Spatial AI, because here power usage is a critical issue, and parallel, heteregeneous, specialised processors seem to be the only route to achieving the computational performance Spatial AI needs within power restrictions which will fit real products. So while current embedded vision systems, *e.g.,* for drones, often use CPU-only implementations of SLAM algorithms (rather than requiring GPUs), we believe that this is not the right approach for the longer term. While current desktop GPUs are certainly power-hungry, Spatial AI must fully embrace parallelism. However, GPUs are only the beginning of the wide space of processor designs that will emerge over the coming years. We highly recommend the recent PhD thesis of Julien Martel for ambitious thinking about this whole area [741].

Mainstream geometric computer vision started to take advantage of parallel processing in the form of GPUs nearly 20 years ago (*e.g.,* [887]), and in Spatial AI this led to breakthroughs in dense SLAM [806, 807]. The SIMT (Single Instruction, Multiple Threads) parallelism that GPUs provide is well suited to elements of real-time vision where the same operation needs to be applied to every element of a regular array in image or map space. Concurrently, GPUs were central to the emergence of deep learning in computer vision [615], by providing the computational resource to enable neural networks of sufficient scale to be trained to finally prove their worth in significant tasks such as image classification.

The move from CPUs to GPUs as the main processing workhorse for computer vision is only the beginning of how processing technology is going to evolve. We foresee a future where an embedded Spatial AI system will have a heterogeneous, multi-element, specialised architecture, where low power operation must be achieved together with high performance. A standard SoC (system-on-chip) for embedded vision ten years from now, which might be used in a personal mobile device, consumer robot or AR headset, will be likely to still have elements which are similar in design to today's CPUs and GPUs, due to their flexibility and the huge amount of useful software they can run. However, it is also likely to have a number of specialised processors optimised for low power real-time vision.

The key to efficient processing which is both fast and consumes little power is *to divide computation between a large number of relatively low clock-rate or otherwise simple cores, and to minimise the movement of data between them.* A CPU pulls and pushes small pieces of data one by one from and to a separate main memory store as it performs computation, with local caching of regularly used data the only

mechanism for reducing the flow. Programming for a single CPU is straightforward, because any type of algorithm can be broken down into sequential steps with access to a single central memory store, but the piece by piece flow of data to and from central memory has a huge power cost.

More efficient processor designs aim to keep processing and the data operated on close together, and to limit the transmission of intermediate results. The ideal way to achieve this is a close match between the design of a processor/storage architecture and the algorithm it must run. A GPU certainly has large advantages over a CPU for many computer vision processing tasks, but in the end a GPU is a processor designed originally for computer graphics rather than vision and AI. Its SIMT architecture can efficiently run algorithms where the same operation is carried out simultaneously on many different data elements. In a full Spatial AI system, there are still many aspects which do not fit well with this, and a joint CPU/GPU architecture is currently needed with substantial data transfer between the two.

While it is relatively accessible to design custom 'accelerator' processors which could implement certain specific low-level algorithms with high efficiency, there has been relatively little work until recently on thinking about the whole computational structure of closed-loop embedded systems like Spatial AI. It is certainly true now that low power vision is seen as an increasingly important aim in industry, and custom processors to achieve this have been developed such as Movidius' Myriad series. These processors combine low power CPU-like, DSP-like and custom elements in a complete package. The 'HPU' custom-designed by Microsoft for their original Hololens AR headset is rather similar in design. More recently, the Apple Vision Pro uses a custom Apple R1 co-processor for real-time sensor input processing. Meta's ARIA Gen 2 smart glasses research device has custom silicon for 'ultra low power and on-device machine perception', so that SLAM, eye tracking, hand-tracking and speech recognition can all operate on-device without the need for an external battery. The details of these commercial chips and the algorithms that run on them are confidential, but we can imagine that they are custom designed for efficient operation of small neural networks, other image processing and some level of general probabilistic optimisation.

If we try to look further ahead, we can conceive of processor designs which offer the possibility of a much closer match between architecture and algorithms. Highly relevant to our aims are major efforts which are now taking place on new ways of doing large-scale processing by being made up from large numbers of independent and relatively low-spec cores with the emphasis on communication. SpiNNaker [356] is a major research project from the University of Manchester which aims to build machines to emulate biological brains. It has produced a prototype machine made up from boards which each have hundreds of ARM cores, and with up to a million cores in total. With the rather different commercial aim of providing an important new type of processor for AI, Graphcore is a UK company developing 'Intelligence

Processing Unit' (IPU) processors which comprise thousands cores on a single chip, each of which with which is able to run an independent program on its own local memory, with an efficient programmable interconnect structure. Other companies such as Tenstorrent and Cerebras have interesting novel chip designs.

The focus of these projects has largely been efficient implementation of neural networks, in the case of both SpiNNaker and Graphcore with the belief that the important matter is the overall topology of a large number of cores, each performing different operations but highly and efficiently inter-connected in a graph configuration adapted to the use case. These designs have not taken strong decisions about the type of processing carried out at each core, or the type of messages they can exchange, with the desire to leave these matters to the choice of future programmers. This is as opposed to more some explicitly neuromorphic architectures aiming to implement particular models of the operation of biological brains, such as IBM's Truenorth project.

Such architectures are not yet close to ready for embedded vision, but seem to offer great long term potential for the design of Spatial AI systems where the graph structure of the algorithms and memory stores we use can be matched to the implementation on the processor in a custom and potentially highly efficient way. We will consider this in more detail later on.

But we also believe that we should go further than thinking of mapping Spatial AI to a single processor, even when it has an internal graph architecture. A more general concept of a graph applies to communication to cameras and other sensors, actuators and other outputs; to other independent robots and devices sharing the same environment; and potentially entirely off-board computing resources in the cloud.

18.4.2 Sensors

Cameras are the most important sensors in Spatial AI, and the concept of a camera is today becoming increasingly broad with ongoing innovation by sensor designers; for our purposes we consider any device which essentially captures an array of light measurements to be a visual sensor. Most are passive in that they record and measure the ambient light which reaches them from their surroundings, while another large class of cameras emit their own light in a more or less controlled fashion. In Spatial AI, many types of camera have been used, with the most common being passive monocular and stereo camera rigs, and depth cameras based on structured light or time of flight concepts. Every camera design represents a choice in terms of the quality of information it provides (measured in such ways as spatial and temporal resolution and dynamic range), and the constraints it places on the system it is used in such as size and power usage. In previous work [435] we studied some of the trade-offs possible between performance and computational cost in the Spatial AI sub-problem of real-time tracking.

A particularly promising sensor for Spatial AI, as already discussed in this handbook, is the event camera, which removes redundancy by sensing and transmitting only changes in intensity. An event stream encodes the information content of video at a much lower bit-rate, while offering advantages in temporal and intensity sensitivity. This is surely only the starting point for coming rapid changes in image sensor technology, where low power computer vision will be an increasingly important driver.

One significant ongoing academic project in this space is the SCAMP series of vision chips with in-plane processing from the University of Manchester (see [742] for an introduction). The SCAMP5 chip runs at 1.2W and has an image resolution of 256×256, with each pixel controlled by and processable by per-pixel processing. Using analog current-mode circuits, summation, subtraction, division, squaring, and communication of values with neighbouring pixels can be achieved extremely rapidly and efficiently to permit a significant level of real-time vision processing completely on-chip. Ultra-low power operation can alternatively already be achieved in applications where low update rates are sufficient.

As we will discuss later on, the range of vision processing which could eventually be performed by such an image plane processor is still be to fully discovered. The obvious use is in front-end pre-processing such as feature detection, local motion tracking, or segmentation. We believe that the longer term potential is that while a central processor will be required for full model-based Spatial AI, close-to-sensor processing can interact fully with this via two-way low communication, with the main aim of reducing the bit-rate needed between the sensor and main processor and therefore the communication power requirements.

Finally, when considering the evolution of the computing resources for Spatial AI, we should never forget that cloud computing resources will continue to expand in capacity and reduce in cost. All future Spatial AI systems will likely be cloud-connected most of the time, and from their point of view the processing and memory resources of the cloud can be assumed to be close to infinite and free. What is not free is communication between an embedded device and the cloud, which can be expensive in power terms, particularly if high bandwidth data such as video is transmitted. The other important consideration is the time delay, typically of significant fractions of a second, in sending data to the cloud for processing. There may also be applications where an always-on, high-bandwidth cloud connection is unfeasible (*e.g.,* space, undersea, underground).

18.5 Mapping Spatial AI Graphs to Hardware

We now turn more specifically to a design for the architecture of a Spatial AI device. Despite the clear potential for cloud-connected shared mapping, here we choose to focus purely on a single device which needs to operate in a space with only on-board

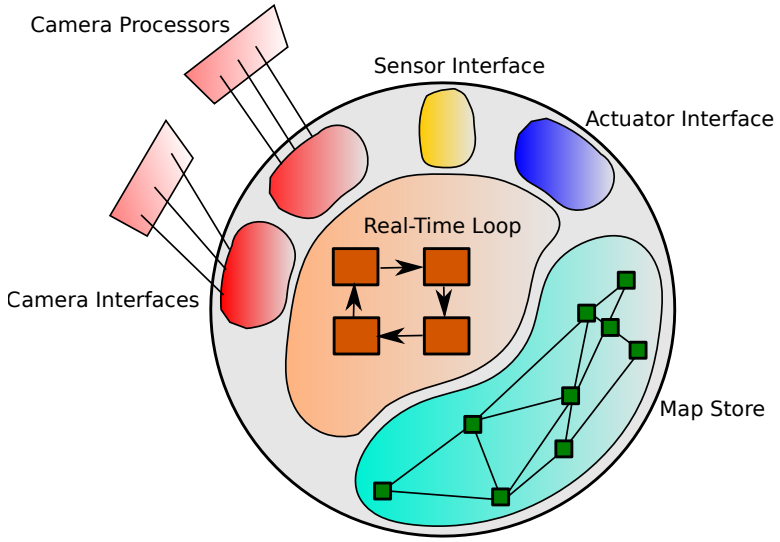

Figure 18.1 Spatial AI brain: how the representation and processing graph structures of a general Spatial AI system might map to a graph processor. The key elements we identify are the real-time processing loop, the graph-based map store, and blocks which interface with sensors and output actuators. Note that we envision additional 'close to the sensor' processing built into visual sensors, aiming to reduce the data bandwidth (eventually in two directions) between the main processor and cameras, which will generally be located some distance away.

resources, because this is the most generally capable setup which could be useful in any application and not require additional infrastructure.

We argue that in the overall architecture of our Spatial AI computation system, the long-term key to efficient performance is to match up the natural graph structures of our algorithms to the configuration of physical hardware. As we have seen, this reasoning leads to the use of 'close to the sensor' processing such as in-plane image processing, and the attempt to minimise data transfer from sensors and towards actuators or other outputs using principles such as events.

However, we still believe that the bulk of computation in an embedded Spatial AI system is best carried out by a relatively centralised processing resource. The key reason for this is the essential and ever-present role of an incrementally built and used world model representing the system's knowledge of its state and that of its environment. Every new piece of data from possibly multiple cameras and sensors is ultimately compared with this model, and either used to update it or discarded if the data is not relevant to device's short or long-term goals.

18.5.1 World Model Processing

What we are anticipating for this central processing resource is however far from the model of a CPU and RAM-like memory store. A CPU can access any contents of its RAM with a similar cost, but in Spatial AI there is much more structure present, both in the locality of the data representing knowledge of the world and in the organisation of the computation workload involved in incorporating new data.

The graph structures of processing and storage should be built into the design of the central world model computation unit, which should combine storage and processing in a fully integrated way. There are already significant efforts on new ways to design architectures which are explicitly and flexibly graphlike.

To focus on one processor concept, Graphcore's IPU or graph processor is designed to efficiently carry out AI workloads which are well modelled as operations on sparse graphs, and in particular when all of the required storage is itself also held *within* the same graph rather than in an external memory store. An IPU chip has a large number of independent cores (in the thousands), each of which can run its own arbitrary program and has its own local memory store, and then a powerful communication substrate such that the cores can efficiently send messages between themselves. When a program is to be run on the IPU, it is first compiled into a suitable form by analysis of the patterns of computation and communication it requires. A suitable graph topology of optimised locations of operations, data, and channels of communication is generated.

In Figure 18.1 we have made a first attempt at drawing a 'Spatial AI brain' model which is analogous to the way Graphcore compiles a computation graph onto the IPU. The disc contains the modules we anticipate running inside the main processor. One of the two main areas is the map store, which is where the current world model is stored. This has an internal graph structure relating to the geometry of the world. It will also contain significant internal processing capability to operate locally on the data in the model, and we will discuss the role of this shortly. The second main area is the real-time loop, which is where the main real-time computation connecting the input image stream to the world model is carried out. This has a processing graph structure and must support large real-time data flows and parallel computation on image/map structures so is designed to optimise this.

The main processor also has additional modules. There are camera interfaces, the job of each of which is to model and predict the data arriving at the sensor to which it is connected. This will then be connected to the camera itself, which the physical design of a robot or other device may force to be relatively far from the main processor. The connection may be serial or along multiple parallel lines.

We then imagine that each camera will have its own 'close-to-sensor' processing capability built in, separated from the main processor by a data link. The goal of modelling the input within the main processor is to minimise actual data transfer to the close-to-sensor processors. It could be that the close-to-sensor processor

performs purely image-driven computation, in a manner similar to the SCAMP project, and delivers an abstracted representation to the main processor. Or, there could be bi-directional data transmission between the camera and main processor. By sending model predictions to the close-to-sensor processors, they know what is already available in the main processor and should report only differences. This is a generalisation of the event camera concept. An event camera reports only changes in intensity, whereas a future optimally efficient camera should report places where the received data is *different from what was predicted*. We will discuss close-to-sensor processing further in Section 18.5.3.

18.5.1.1 World Model

The graph structure of world models has been recognized and made use of by many important SLAM methods (*e.g.,* [541, 603, 969, 315, 761]).

As a robot moves through and observes the world, a SLAM algorithm detects, tracks, and inserts into its map features which are extracted from the image data. Note that we use the word 'feature' here in a general sense to mean some abstraction of a scene entity, and that we are not confining our thinking to sparse point-like landmarks. As abstraction in Spatial AI increases, these features are likely to be objects or other semantic entities. Each feature in the scene has a region of camera positions from which it is measurable. A feature will not be measurable if it is outside of the sensor's field of view; if it is occluded; or for other reasons such that its distance from the camera or angle of observation are very different from when it was first observed.

This means that as the robot moves through a scene, features become observable in variable overlapping patterns. As measurements are made of the currently visible features, estimates of their locations are improved, and the measurements are also used to estimate the camera's motion. The estimates of the locations of features which are observed at the same time become strongly correlated with each other via the uncertain camera state. Features which are 'nearby' in terms of the amount of camera motion between observing them are still correlated but somewhat less strongly; and features which are 'distant' in that a lot of motion (and SLAM based on intermediate features) happens between their observation are only weakly correlated.

The probabilistic joint density over feature locations which is the output of SLAM algorithms can be efficiently represented by a graph where 'nearby' features are joined by strong edges, and 'distant' ones by weak edges. A threshold can be chosen on the accuracy of probabilistic representation which leads to the cutting of weaker edges, and therefore a sparse graph where only 'nearby' features are joined.

One way to do SLAM is not to explicitly estimate the state of scene features, but instead to construct a map of a subset of the historical poses that the moving camera has been in, and to keep the scene map implicit. This is usually called pose-graph SLAM (*cf.* Chapter I), and within this kind of map the graph structure

is obvious because we join together poses between which we have been able to get sensor correspondence. Poses which are consecutive in time are joined; and poses where we are able to detect a revisit after a longer period of time are also joined (*i.e.,* loop closures). Whether the graph is of historic poses, or of scene features, its structure is very similar, in that it connects either 'nearby' poses or the features measured from those poses, and there is not a fundamental difference between the two approaches.

This leads us to conclude that *the most likely representation for Spatial AI is to represent 3D space by a graph of features, which are linked in multi-scale patterns relating to camera motion and together are able of generating dense scene predictions.* Within the main processor, a major area will be devoted to storing this map, in a manner which is distributed around potentially a large number of individual cores which are connected in a topology to mirror the map graph topology. In SLAM, the map is defined and grown dynamically, so the graph within the processor must either be able to change dynamically as well, or must be initially defined with a large unused capacity which is filled as SLAM progresses.

Importantly, a significant portion of the processing associated with large-scale SLAM can be built directly into this graph. This is mainly the sort of 'maintenance' processing via which the map optimises, refines, and abstracts itself; including:

1 Feature clustering; object segmentation and identification.

2 Loop closure detection.

3 Loop closure optimisation.

4 Map regularisation (smoothing).

5 Unsupervised clustering to discover new semantic categories.

With time, data, and processing, a map which starts off as dense geometry and low-level features can be refined towards an efficient object-level map. These operations will run with very high parallelism and data locality. Smoothing and segmentation for instance take place at each part of the graph separately, while global map optimisation can be implemented via message passing, as we will explain in Section 18.6. Most importantly, all of these processes can take place in a manner which is internal to the map store itself.

The on-chip memory of current graph processors like Graphcore's IPU is fully distributed among the processing tiles, and the total capacity is large (on the order of around 1MB per tile, or around 1GB total over 1000 tiles), but not huge (certainly when compared with standard off-processor RAM), and therefore there should be an emphasis on rapidly abstracting the map store towards an efficient long-term form. Future processors may however have much more distributed storage.

18.5.2 Real-Time Loop

We make a hypothesis that the core computation graph for the tightest real-time loop of future SLAM systems will have many elements which are familiar from today's systems. Specifically the dense SLAM paradigm introduced by Newcombe *et al.* [805, 807, 806] is at the centre of this, because this approach aims to model the world in dense, generative detail such that every new pixel of data from a camera can be compared against a model-based prediction. This allows systems which are generally 'aware', since as they continually model the state of the world in front of the camera, they can detect when something is out of place with respect to this model, and therefore dense SLAM systems are now being developed which are moving beyond static scenes to reconstruct and track dynamic scenes [808, 962]. Dense SLAM systems can also make the best possible use of scene priors, which will increasingly come from learning rather than being hand-designed.

Some of the main computational elements in the real-time loop are:

1 Empirical labelling of images to features, usually via a neural network.
2 Rendering: producing a dense prediction from the world map into image space.
3 Tracking: aligning a prediction with new image data, including finding outliers and detecting independent movement.
4 Fusion: fusing updated geometry and labels back into the map.
5 Self-supervised learning from the running system.

Architecturally, this processing depends on both the live input from cameras and other sensors, and the world model, and must therefore in some sense take place 'between' them. Data from the map store is needed for *rendering*, when a predicted view of the scene is needed for tracking against new image data, and for *fusion*, when information (geometry and labels) acquired from the new data is used to update the map contents. This is generally massively parallel processing which is familiar from GPU-accelerated dense SLAM systems, and these functions can be defined as fixed elements in a computational graph which use a number of nodes and access a significant fraction of the main computational resource. Functions such as segmentation and labelling of input images could also be implemented here, though as we will discuss shortly these would be more sensibly located outside of the main processor by close-to-sensor processing.

The most difficult issue in applying graph processing to the real-time loop is the fact that correspondence with the relevant part of the graph-based map store for these operations changes continuously due to motion. This seems to preclude defining an efficient, fixed data path to the distributed memory where map information will be stored. Although there will be internal processing happening in the map store, this will be focused on maintenance and it does not seem appropriate to move data rapidly around the map store, for instance such that the currently observed part of the map is always available in a particular graph location.

Instead, a possible solution is to define special interface nodes which sit between the real-time loop block and the map store. These are nodes focused on communication, which are connected to the relevant components of real-time loop processing and then also to various sites in the map graph, and may have some analogue in the hippocampus of mammal brains. If the map store is organised such that it has a 'small world' topology, meaning that any part is joined to any other part by a small number of edge hops, then the interface nodes should be able to access (copy) any relevant map data in a small number of operations and serve them up to the real-time loop.

Each node in the map store will also have to play some part in this communication procedure, where it will sometimes be used as part of the route for copying map data backwards and forwards.

18.5.3 Processing Close to the Image Plane

A robot or other device will have one or more cameras which interface with the main processor, and we believe that the technology will develop to allow a significant amount of processing to occur either within the sensors themselves or nearby, with the key aim of reducing the amount of redundant data which flows from the cameras.

The pixels which make up the image sensor of a camera have a regular, usually rectangular geometry. While each of these pixels is normally independently sensitive to light intensity, many vision algorithms make use of the fact that the output of nearby pixels tends to be strongly correlated. This is because nearby pixels often observe parts of the same scene objects and structures. Most commonly, a regular graph in which each pixel is connected to its four (up, down, left, right) neighbours is used as the basis for smoothing operations in many early vision problems such as dense matching or optical flow estimation (e.g. [886]). The regular graph structure of images is also taken advantage of by the early convolutional layers of CNNs for all kinds of computer vision tasks. Here the multiple levels of convolutions also acknowledge the typical hierarchical nature of local regularity in image data.

Most straightforwardly, a sensor with in-plane processing similar to SCAMP5 [742] could be used to carry out purely bottom-up processing of the input image stream; abstracting, simplifying, and detecting features to reduce it to a more compact, data-rich form. Calculations such as local tracking (*e.g.,* optical flow estimation), segmentation and simple labelling could also be performed. This has biological analogies with the 'early vision' processing carried out by the retina and optic nerve.

Tracking using in-plane processing is an interesting problem. In-plane processing is good for problems where data access can be kept very local, so estimating local image motion (optical flow), where the output is a motion vector at every pixel, is well suited. At each update, the amount of image change locally can be augmented using local regularisation where smoothness is applied based on the differences of

neighbouring pixels. If we look at the parallel implementation of an algorithm like Pock's TV-L1 optical flow [886], we see that it involves pixelwise-parallel operations, where purely parallel steps relating to the data term alternate with regularisation steps involving gradient computation, which could be achieved using message passing between adjacent neighbours. So such an algorithm is an excellent candidate for implementation on an in-plane processor.

More challenging is the tracking usually required in SLAM, where from local image changes we wish to estimate consistent global motion parameters relating to a model. This could be instantaneous camera motion estimation, where we wish to estimate the amount of global rotation for instance between one frame and the next via whole image alignment [715], or tracking against a persistent scene model as in dense SLAM [705, 805]. When dense tracking is implemented on standard processors, it involves alternation of purely parallel steps for error term computation across all pixels with reduction steps where all errors are summed and the global motion parameters are updated. The reduction step, where a global model is imposed, plays the role of regularisation, but the big difference is that the regularisation here is global rather than local.

In a modern system, such global tracking is usually best achieved by a combination of GPU and CPU, and therefore a regular (and expensive) transfer of data between the two. But we do not have this option if we wish to use in-plane processing and keep all computations and memory local. Our main option for not giving up on data locality is to give up on guaranteed global consistency of our tracking solution, but to aim to converge towards this via local message passing. Each pixel could keep its own estimate of the global motion parameters of interest, and after each iteration share these estimates with local neighbours. We would expect that global convergence would eventually be reached, but that after a certain number of iterations that the values held by each pixel would be close enough that any single pixel could be queried for a usable estimate. Convergence would be much faster if the in-plane processor also featured some longer data-passing links between pixels, or more generally had a 'small world' pattern of interconnections.

Turning to the questions of labelling using local processing, this is certainly feasible but a problem with sophisticated labelling is that current in-plane processor designs have very small amounts of memory, which makes it difficult to store learned convolutional masks or similar. Future processors for bottom-up processing may move beyond operating purely in the image plane, and use a 3D stacking approach which could be well suited to implementing the layers of a CNN. Currently 3D silicon stacks are hard to manufacture, and there are particular challenges around heat dissipation and cost, but the time will surely come when extracting a feature hierarchy is a built-in capability of a computer vision camera. Work such as *featuremetric* tracking [243] shows the wide general use this would have. We should investigate the full range of outputs that a single purely feed-forward CNN could produce when trained with a multi-task learning approach.

An interesting question is whether processing close to the image plane will remain purely bottom-up, or whether two-way communication between cameras and the main processor will be worthwhile. This would enable model predictions from the world map to be delivered to the camera at some rate, and therefore for higher level processing to be carried out there such as model-based tracking of the camera's own motion or known objects.

In the limit, if a fully model-based prediction is able to be communicated to the camera, then the camera need only return information which is *different* from the prediction. This is in some sense a limit of the way that an event camera works. An event camera outputs data if a pixel changes in brightness — which is like assuming that the default is that the camera's view of the scene will stay the same. A general 'model-based event camera' will output data if something happens which differs from its prediction. These questions come down to key issues of 'bottom-up vs. top-down' processing, and we will consider them further in the next section.

As a final note here, any Spatial AI system must ultimately deliver a task-determined output. This could be the commands sent to robot actuators, communication to be sent to another robot or annotations and displays to be sent to a human operator in an IA setting. Just as 'close to the sensor' processing is efficient, there should alse be a role for 'close to the actuator' processing, particularly because actuators or communication channels have their own types of sensing (torque feedback for actuators; perhaps eye tracking or other measures for an AR display) which need to be taken account of and fused in tight loops.

18.6 Convergent Distributed Computation with Gaussian Belief Propagation

We have argued that efficiency and low power demand a graph design for Spatial AI systems, where processing and local storage is divided into modules and spread potentially across several hardware processors and sensors, but so far said little about the algorithms that will keep the stored representations up to date. As explained throughout this handbook, probability and estimation theory provide the fundamental background for state estimation in SLAM, and in particular factor graphs [259] have been universally useful as a flexible representation of the structure of the complex structure of SLAM systems.

Sutton's famous 'Bitter Lesson' of AI [1068] is that 'general methods that leverage computation are ultimately the most effective, and by a large margin'. In Spatial AI, if we truly believe that low-power computation and storage resources will become more and more distributed, both within and across processors, the right algorithms to bet on should be ready to scale with these conditions. The purest representation of the knowledge in a Spatial AI problem is the factor graph itself, rather than probability distributions derived from it, which will always have to be stored with some approximation. What we are really seeking is an algorithm which implements

Spatial AI by storing the factor graph as the master representation and operating on it in place using local computation and message passing to implement estimation of variables as needed but taking account of global influence. Messages should continually 'bubble' 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 high quality to currently important areas or aspects, which then are allowed to fade to a less up-to-date state once attention moves on, in a 'just-in-time' style.

If we wish estimation on factor graphs to have the properties of purely local computation and data storage, we must get away from the idea that a 'god's eye view' of the whole structure of the graph will ever be available. We are guided towards methods where each node of a processing and storage graph can operate with minimum knowledge of the whole graph structure — at a minimum, only purely local information about itself and its near neighbours.

This is the character of the Belief Propagation (BP) family of algorithms for inference of probabilistic factor graphs. In BP, 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 particular, the most important techniques in current geometric Spatial AI estimation are all Gaussian-based techniques such as Extended Kalman Filtering and Bundle Adjustment. 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. Gaussian Belief Propagation (GBP) is the specific form of BP when all factors and variables in a graph have Gaussian form, and it has particularly strong properties of convergence to optimal solutions in loopy graphs.

Davison and Ortiz gave a general introduction to GBP in FutureMapping 2 [251], and an interactive tutorial is presented in the online article 'A Visual Introduction to Gaussian Belief Propagation' [839]. These articles showed that GBP is fully compatible with the non-linear and robust factors which are ubiquitous in Spatial AI problems, and can be applied to problems including pose graph optimisation, sparse SLAM, surface fitting and image smoothing.

There had been a previous practical implementation of GBP to SLAM in [913], showing that it has some useful properties even when implemented on a single CPU, but Ortiz *et al.* [838] demonstrated its longer-term potential by using it to implement high performance bundle adjustment on a Graphcore IPU chip, with a fully distributed implementation solving small real vision optimisation problems 30X faster than on a CPU. Figure 18.2 shows the simple way that a bundle ad-

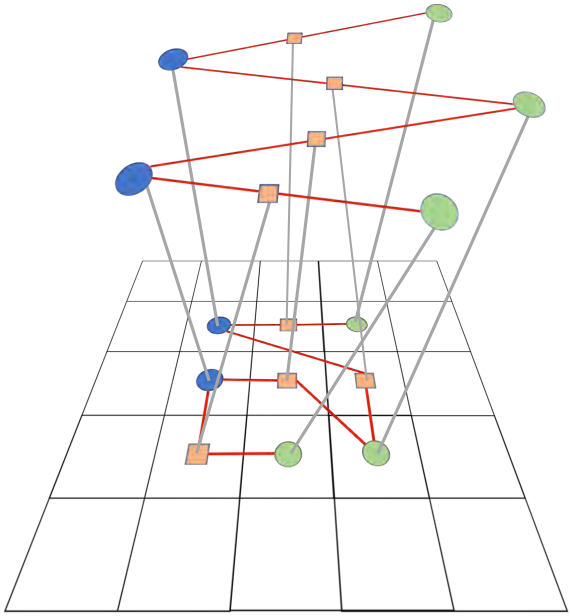

Figure 18.2 Mapping a bundle adjustment factor graph onto the tiles (cores) of a graph processor, allowing rapid, distributed, in-place inference via Gaussian Belief Propagation, from [838] (©IEEE). Here we display the most simple mapping in which each node in the factor graph is mapped onto a single arbitrary tile. Keyframe nodes are blue, landmark nodes are green and measurement factor nodes are orange. Tiles process synchronously, alternating compute and exchange steps.

justment factor graph was laid down on the tiles of a graph processor to take full advantage of its parallelism.

The generality of distributed of computation on graphs with GBP was taken even further by Murai *et al.* in Robot Web [794], showing that it can also be used for many-robot Spatial AI problems. Specifically, multi-robot localisation was formulated as a single factor graph as shown in Figure 18.3, which was divided up and stored across a network of robots connected by ad-hoc communication such as Wi-Fi. Robot Web showed GBP can achieve convergent, accurate estimation even with asynchronous message passing and in the presence of failing sensors and dropped communication. These properties also promise much for single robot Spatial AI when we envision the very long-term prospects for robot processors, which may achieve even higher performance per unit energy by becoming more brain-like — with perhaps low-precision number representation and asynchronous or event-driven parallelism.

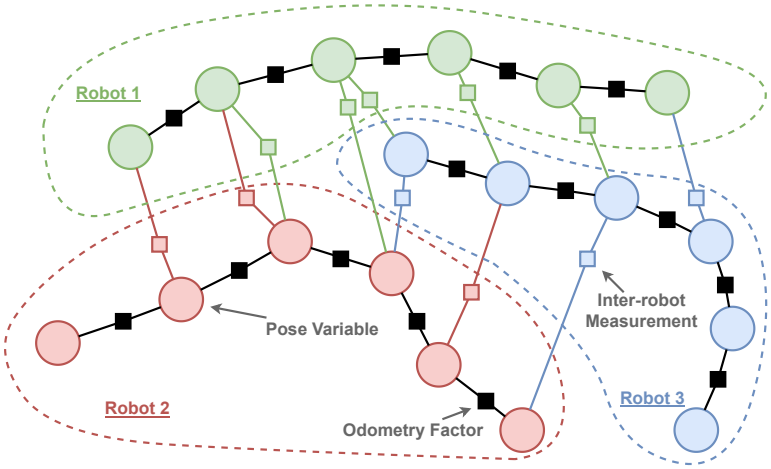

Figure 18.3 A factor graph for multi-robot localisation, taken from Robot Web [794] (©IEEE). Responsibility for storing and updating it is divided up between the multiple robots participating, as shown by the coloured regions separated by dotted lines. Each robot maintains its own pose variable nodes, 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 bound arises is via Robot Web Pages published and updated by each robot, and happens on an asynchronous and ad-hoc basis.

18.7 Continual Learning within Factor Graphs

We discussed the increasing importance of learned components in Spatial AI in Section 18.3, but interfacing these with modular state estimation continues to be clunky and with many limitations. Looking further ahead, we believe that the following ambitious properties are important for learning in Spatial AI:

1 Learning will be a key part of Spatial AI systems, but should run seamlessly alongside existing tried-and-tested hand-designed probabilistic inference algorithms which can correctly weigh and combine multiple uncertain information sources.
2 The primary mode of learning should be self-supervised, meaning that a running system can learn useful abstractions without external help, such as from a human providing labelled datasets.
3 Learning in Spatial AI should be continual, taking place as part of a running system whose performance is gradually improving in consequence. We should move away from the current artificial divide between 'training time' and 'test time'.
4 Learned components should ideally be interpretable, or human understandable, at least at some level of modules and how they connect together.

Most learning algorithms, and in particular standard neural networks, are trained and used separately. Training finds values for the parameters (weights) of a model by gradient descent using empirical numbers to determine learning rates and stopping criteria, and ad-hoc concepts are used such as dividing data into training and validation sets. It is very hard to imagine how such methods could be used properly for continual, self-supervised learning alongside probabilistic estimation.

We first argue that the points above lead us to employ a probabilistic approach to machine learning for Spatial AI. In probabilistic, or Bayesian ML, the parameters which represent a learned model are estimated using the same probabilistic inference laws as in standard probabilistic inference. There is no fundamental difference between state estimation in probabilistic inference, and model learning in Bayesian ML. It is simply a matter of structure and scale. Using the same machinery for both means that we can correctly attack self-supervised, continual learning alongside standard Bayesian inference.

Efficient and scalable Bayesian ML is a challenging topic, but Nabarro *et al.* [799] have recently made a proposal and preliminary demonstrations of a GBP Learning approach which is fully compatible with the factor graph Spatial AI vision of this chapter. The key idea is that overparameterised learning structures are built into factor graphs themselves, and that learning takes place as Bayesian inference using standard Gaussian Belief Propagation machinery. Instead of the neurons and weights of a neural networks, we build a graph of variables and factors with a similar structure. Variables infer values in a manner equivalent to weight learning, but this can happen continuously and with no explicit difference between training and inference time. Non-linear factors play the role of neurons, able to soft-switch and therefore combine in varied patterns to represent complex functions. Figure 18.4 from [799] shows a factor graph block with convolutional structure which has was used in a system able to to continual learning of image denoising and classification.

This approach or something similar could unlock general continual learning in Spatial AI. For instance, a robot could learn a smoothing or segmentation model for a scene while reconstructing it. The unified factor graph structure of both learning and estimations components of its systems would allow many choices of efficient or distributed estimation, or the easy synchronisation of learned elements across multiple devices.

18.8 Performance Metrics

One of the hypotheses we made at the start of this chapter is that the usefulness of a Spatial AI system for a wide range of tasks is well represented by a relatively small number of performance measures which have general importance.

The most common focus in performance measurement in SLAM is on localisation accuracy, and there have been several efforts to create benchmark datasets for

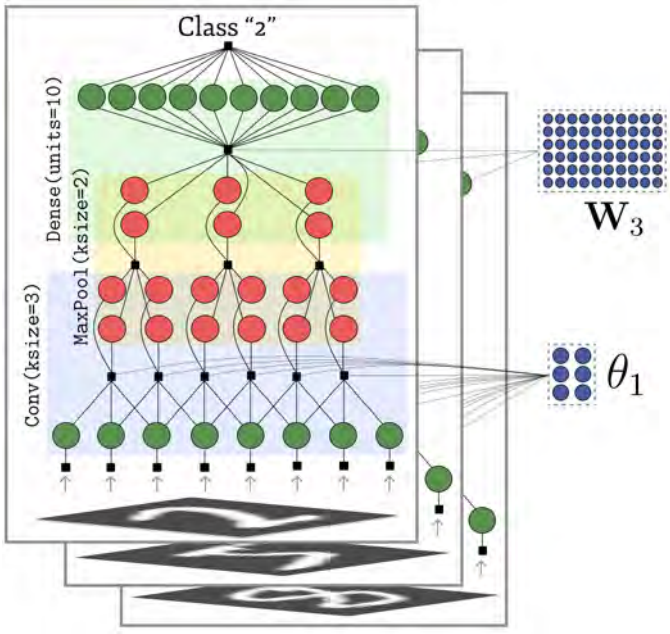

Figure 18.4 In GBP Learning [799], factor graphs are designed with structure which mirrors common NN architectures, enabling distributed training and prediction with GBP. Learnable parameters are included as random variables (circles), as are inputs, outputs and activations. The parameters are shared over across all observations, where the other variables are copied once per observation. Factors (black squares) between layers constrain their representations to be locally consistent, while those attached to inputs and outputs encourage compatibility with observation. The inter-layer factors are non-linear to enable soft-switching behaviour. This example architecture for image classification comprises convolutional, max pooling and dense projection layers. The same architecture could be trained without supervision by removing the output observation factor.

this (e.g. [1050]). An external pose measurement from a motion capture system is considered as the ground truth against which a SLAM algorithm is compared.

We have argued that Spatial AI is about much more than pose estimation, and more recent datasets have tried to broaden the scope of what can be evaluated. Dense scene modelling is difficult to evaluate because it requires an expensive and time-consuming process such as detailed laser scanning to capture a complete model of a real scene which is accurate and complete enough to be considered ground truth. The alternative, for this and other axes of evaluation, is to generate synthetic test data using computer graphics, such as the ICL-NUIM dataset [436]. Recently this approach has been extended also to provide ground truth for semantic labelling (*e.g.,* SceneNet RGB-D [753]), another output where it is difficult to get good ground truth for real data. It is natural to be suspicious of the value of evalua-

tion against synthetic test data, and there are many new approaches to gathering large scale real mapping data, such as crowdsourcing (e.g. ScanNet [247]). However, synthetic data is getting better all the time and we believe will only grow in importance.

Stepping back to a bigger point, we can question the value of using benchmarks at all for Spatial AI systems. It is an often heard comment that computer vision researchers are hooked on dataset evaluation, and that far too much effort has been spent on optimising and combining algorithms to achieve a few more percentage points on benchmarks rather than working on new ideas and techniques. SLAM, due to its real-time nature, and wide range of useful outputs and performance levels for different applications, has been particularly difficult to capture by meaningful benchmarks. We have usually felt that more can be learned about the usefulness of a visual SLAM system by playing with it for a minute or so, adaptively and qualitatively checking its behaviour via live visualisations, than from its measured performance against any benchmark available. Progress has therefore been more meaningfully represented by the progress of high quality real-time open source SLAM research systems (*e.g.,* MonoSLAM [253, 252], PTAM [591], KinectFusion [807], LSD-SLAM [315], ORB-SLAM [793], OKVIS [652], SVO [341], ElasticFusion [1181], DROID-SLAM [1086], Gaussian Splatting SLAM [747], MASt3R-SLAM [795]) that people can experiment with, rather than benchmarks.

Benchmarks for SLAM have been unsatisfactory because they make assumptions about the scene type and shape, camera and other sensor choices and placement, frame-rate and resolution, etc., and focus in on certain evaluation aspects such as accuracy while downplaying other arguably more important ones such as efficiency or robustness. For instance, many papers evaluating algorithms against accuracy benchmarks make choices among the test sequences available in a dataset such as [1050] and report performance only on those where they basically 'work'.

This brings us to ask whether we should build benchmarks and aim to evaluate and compare SLAM systems at all? We still argue yes, but the focus on broadening what is meant by a benchmark for a Spatial AI system, and an acknowledgement that we should not put too much faith in what they tell us.

The SLAMBench framework released by the PAMELA research project [801] represented initial work on looking at the performance of a whole Spatial AI system. In SLAMBench, a SLAM algorithm (specifically KinectFusion [807]) is measured in terms of both accuracy and computational cost across a range of processor platforms and using different language implementations. Over the longer term, we believe that benchmarking should move towards measures which have the general ability to predict performance on tasks that a Spatial AI might need to perform. This will clearly be a multi-objective set of metrics, and analysis of Pareto fronts [802] will permit choices to be made for a particular application.

A possible set of metrics includes:

1 Local pose accuracy in newly explored areas (visual odometry drift rate).
2 Long-term metric pose repeatability in well mapped areas.
3 Tracking robustness percentage.
4 Relocalisation robustness percentage.
5 SLAM system latency.
6 Dense distance prediction accuracy at every pixel.
7 Object segmentation accuracy.
8 Object classification accuracy.
9 Pixel registration accuracy for augmented reality.
10 Scene change detection accuracy.
11 Power usage.
12 On-device data movement, measured in bits×millimetres.

18.9 Conclusions

To conclude, the research area of Spatial AI and SLAM will continue to gain in importance, and evolve towards the general 3D perception capability needed for many different types of application with the full fruition of the combination of hand-designed estimation and machine learning techniques. However, wide use in real applications will require this advance in capability to be accompanied by driving down the resources required, and this needs joined-up thinking about algorithms, processors, and sensors. The key to efficient systems will be to identify their computational structure, and to design for sparse graph patterns in algorithms and data storage, and to fit this as closely as possible to the new types of hardware which will soon gain in importance.

Both hand-designed state estimation and machine learning will continue to be important, and must be fully integrated to enable flexible and continual learning. We believe we will witnessing an interesting convergence, as state estimation systems add more and more learned elements, and learned networks introduce increased domain knowledge and structure. Either route should lead to the same place, and the same research questions about computational structure will need to be tackled.

Notes

References

[1] *FastTriggs*. https://pypose.org/docs/main/generated/pypose.optim.corrector.FastTriggs/.

[2] *LieTensor*. https://pypose.org/docs/main/generated/pypose.LieTensor.

[3] *PyPose Documents*. https://pypose.org/docs/.

[4] *PyPose Tutorial*. https://github.com/pypose/tutorials.

[5] *Reality Labs Chief Scientist Outlines a New Compute Architecture for True AR Glasses*. https://www.roadtovr.com/michael-abrash-iedm-2021-compute-architecture-for-ar-glasses/.

[6] 2015. *You Only Look Once: Unified, Real-Time Object Detection*.

[7] 2025 (Mar.). *Meta Quest 3: Technical Specifications*.

[8] A, Zeng, Attarian, M., Ichter, B., Choromanski, K., Wong, A., Welker, S., Tombari, F., Purohit, A., Ryoo, M., Sindhwani, V., Lee, J., Vanhoucke, V., and Florence, P. 2022. Socratic Models: Composing Zero-Shot Multimodal Reasoning with Language. *arXiv*.

[9] Abadi, Martín, Barham, Paul, Chen, Jianmin, Chen, Zhifeng, Davis, Andy, Dean, Jeffrey, Devin, Matthieu, Ghemawat, Sanjay, Irving, Geoffrey, Isard, Michael, et al. 2016. TensorFlow: a system for Large-Scale machine learning. Pages 265–283 of: *12th USENIX symposium on operating systems design and implementation (OSDI 16)*.

[10] Abate, M., Chang, Y., Hughes, N., and Carlone, L. 2023a. Kimera2: Robust and Accurate Metric-Semantic SLAM in the Real World. In: *Intl. Sym. on Experimental Robotics (ISER)*.

[11] Abate, M., Schwartz, A., Wong, X.I., Luo, W., Littman, R., Klinger, M., Kuhnert, L., Blue, D., and Carlone, L. 2023b. Multi-Camera Visual-Inertial Simultaneous Localization and Mapping for Autonomous Valet Parking. In: *Intl. Sym. on Experimental Robotics (ISER)*. ,.

[12] Achiam, Josh, Adler, Steven, Agarwal, Sandhini, Ahmad, Lama, Akkaya, Ilge, Aleman, Florencia Leoni, Almeida, Diogo, Altenschmidt, Janko, Altman, Sam, Anadkat, Shyamal, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

[13] Adolfsson, Daniel, Magnusson, Martin, Alhashimi, Anas, Lilienthal, Achim J., and Andreasson, Henrik. 2021. CFEAR Radarodometry – Conservative Filtering for Efficient and Accurate Radar Odometry. Pages 5462–5469 of: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.

[14] Adolfsson, Daniel, Castellano-Quero, Manuel, Magnusson, Martin, Lilienthal, Achim J., and Andreasson, Henrik. 2022. CorAl: Introspection for robust radar and lidar perception in diverse environments using differential entropy. *Robotics and Autonomous Systems*, May, 104136.

[15] Adolfsson, Daniel, Magnusson, Martin, Alhashimi, Anas, Lilienthal, Achim J., and Andreasson, Henrik. 2023a. Lidar-Level Localization With Radar? The CFEAR Approach to Accurate, Fast, and Robust Large-Scale Radar Odometry in Diverse Environments. *IEEE Trans. Robotics*, **39**(2), 1476–1495.

[16] Adolfsson, Daniel, Karlsson, Mattias, Kubelka, Vladimír, Magnusson, Martin, and Andreasson, Henrik. 2023b. TBV Radar SLAM – trust but verify loop candidates. *IEEE Robotics and Automation Letters*, **8**, 3613–3620.

[17] Aftab, Khurrum, and Hartley, Richard. 2015. Convergence of iteratively re-weighted least squares to robust m-estimators. Pages 480–487 of: *2015 IEEE Winter Conference on Applications of Computer Vision*. IEEE.

[18] Agarwal, P., Grisetti, G., Tipaldi, G. D., Spinello, L., Burgard, W., and Stachniss, C. 2014. Experimental Analysis of Dynamic Covariance Scaling for Robust Map Optimization Under Bad Initial Estimates. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.

[19] Agarwal, Pratik, Tipaldi, Gian Diego, Spinello, Luciano, Stachniss, Cyrill, and Burgard, Wolfram. 2013. Robust map optimization using dynamic covariance scaling. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.

[20] Agarwal, Sameer, Snavely, Noah, Simon, Ian, Seitz, Steven M, and Szeliski, Richard. 2009. Building Rome in a day. Pages 72–79 of: *Intl. Conf. on Computer Vision (ICCV)*. IEEE.

[21] Agarwal, Sameer, Furukawa, Yasutaka, Snavely, Noah, Simon, Ian, Curless, Brian, Seitz, Steven M, and Szeliski, Richard. 2011. Building Rome in a day. *Communications of the ACM*, **54**(10), 105–112.

[22] Agarwal, Sameer, Mierle, Keir, and Team, The Ceres Solver. 2022 (3). *Ceres Solver*.

[23] Agate, M., Grimsdale, R. L., and Lister, P. F. 1991. The HERO Algorithm for Ray-tracing Octrees. Pages 61–73 of: *Advances in Computer Graphics Hardware IV*.

[24] Agia, Christopher, Jatavallabhula, Krishna Murthy, Khodeir, Mohamed, Miksik, Ondrej, Vineet, Vibhav, Mukadam, Mustafa, Paull, Liam, and Shkurti, Florian. 2022. Taskography: Evaluating Robot Task Planning over Large 3D Scene Graphs. Pages 46–58 of: *Conf. on Robot Learning (CoRL)*.

[25] Agrawal, A., Amos, B., Barratt, S., Boyd, S., Diamond, S., and Kolter, Z. 2019. Differentiable Convex Optimization Layers. In: *Advances in Neural Information Processing Systems*.

[26] Agrawal, Varun, Bertrand, Sylvain, Griffin, Robert J., and Dellaert, Frank. 2022. Proprioceptive State Estimation of Legged Robots with Kinematic Chain Modeling. Pages 178–185 of: *IEEE Intl. Conf. on Humanoid Robots*.

[27] Agudo, Antonio, and Moreno-Noguer, Francesc. 2015. Simultaneous pose and non-rigid shape with particle dynamics. In: *CVPR*.

[28] Agudo, Antonio, Agapito, Lourdes, Calvo, Begona, and Montiel, J.M.M. 2014. Good vibrations: A modal analysis approach for sequential non-rigid structure from motion. In: *CVPR*.