Sequential Localisation and Map-Building in Computer Vision and Robotics *

Andrew J. Davison and Nobuyuki Kita

Intelligent Systems Division, Electrotechnical Laboratory
1-1-4 Umezono, Tsukuba, Ibaraki 305–8568, Japan
`[davison,nkita]@etl.go.jp`

Abstract. Reviewing the important problem of sequential localisation and map-building, we emphasize its genericity and in particular draw parallels between the often divided fields of computer vision and robot navigation. We compare sequential techniques with the batch methodologies currently prevalent in computer vision, and explain the additional challenges presented by real-time constraints which mean that there is still much work to be done in the sequential case, which when solved will lead to impressive and useful applications. In a detailed tutorial on map-building using first-order error propagation, particular attention is drawn to the roles of modelling and an active methodology. Finally, recognising the critical role of software in tackling a generic problem such as this, we announce the distribution of a proven and carefully designed open-source software framework which is intended for use in a wide range of robot and vision applications: **http://www.robots.ox.ac.uk/~ajd/**

1 Introduction

Structure from motion in computer vision and simultaneous map building and localisation for mobile robots are two views of the same problem. The situation under consideration is that of a body which moves through a static environment about which it has little or no prior knowledge, and measurements from its sensors are used to provide information about its motion and the structure of the world. This body could be a single camera, a robot with various sensors, or any other moving object able to sense its surroundings. Nevertheless, in recent years research has taken many paths to solving this problem with a lack of acknowledgement of its general nature, with a particular divide arising between robotic and "pure vision" approaches.

Crucially, in this paper we are interested in the **sequential** case, where map-building and localisation are able to proceed in a step-by-step fashion as movement occurs. We will contrast this with situations where the **batch** methods currently prevalent in computer vision (and their cousins in robot map-building) can be applied, where measurements from different time steps are used in parallel after the event. Despite renewed interest in sequential map-building from

* The authors are grateful to Philip McLauchlan, Simon Julier and John Leonard for discussions.

the robotics community, in computer vision recent successful work in off-line reconstruction from image sequences has conspicuously not been accompanied by advances in real-time methods. Sequential map-building is a problem which is far from being solved, and we will look at the state of the art and its limitations.

1.1 Applications

In map-building applications where localisation or map estimates are needed quickly and successively, either to supply data to external processes in real-time or to feed back into determining future actions, **only sequential methods can be used**. For instance:

- Camera-based structure from motion methods that need to update in real-time, like in an inside-out head-tracking application, where an outward-looking camera attached to a head-mounted display user's head identifies and tracks arbitrary features in the surroundings to calculate head movement, or in live virtual studio applications, where the movement of television cameras needs to be known precisely so that live and computer-generated images can be fused to form composite output.
- Autonomous robot navigation in unknown environments, where sensor readings are used to build and update maps, and continually estimate the robot's movement through the world.

1.2 Aims of this Paper

1. To review and clarify the status of the sequential map-building problem, and emphasize its genericity within robotics and computer vision.
2. In a detailed tutorial on map-building using first-order error propagation, to discuss a number of details about implementing real sequential systems and explain the approaches our experience has led to.
3. To announce the distribution of an open-source software package for sequential localisation and map-building, designed with a realisation of the general nature of this class of problems and therefore readily applicable in many applications, and already proven in two research projects [7, 6],

2 The Challenges of Sequential Map-Building

2.1 The Main Point

Thinking first not of actively adding to a map, but of updating uncertain estimates of the locations of various features and that of a moving sensor platform measuring them in a sequential, real-time sense: **the amount of computation which can be carried out in each time-step is bounded by a constant**. This follows simply from thinking of implementing such a system: however fast the processor available, it can only do so much in a certain time-step.

A major implication of this is that **at a given time, we must express all our knowledge of the evolution of the system up to that time with an amount of information bounded by a constant.** The previous knowledge must be combined with any new information from the current time step to produce updated estimates within the finite processing time available.

In the following sections, we will look at the approach we are forced to take to fit this constraint and the difficulties this presents, since in sequential processing compromises must be made to maintain processing speed.

2.2 Approaching Sequential Map-Building

There would seem to be a difference between robot map-building for localisation, where the goal is to determine the robot's motion making use of arbitrary features in the environment as landmarks, and structure from motion, where the interest is in the scene structure and not in the arbitrary path of the camera used to study it. However, it is necessary, explicitly or implicitly, to estimate both sensor motion and scene structure together if either is to be determined.

Batch Methods: The optimal way to build maps from measurements from a moving robot or sensor is to take all the data obtained over a motion sequence and analyse it **at once**. Estimates of where the robot was at each measurement location are calculated altogether, along with the locations of the features observed, and then adjusted until they best fit all the data. This is the **batch** methodology which is used in state of the art geometrical vision to recover 3D structure maps from video sequences and auto-calibrate cameras (e.g. [20, 17]). In robot navigation, batch methods have a shorter history but have appeared recently under the banner **EM** [19], where maps of natural features were formed from a data set collected in an initial guided robot tour of an area; afterwards the robot could use the map during autonomous navigation. However, while batch methods can build optimal maps from previously acquired data sets, they do not offer a way to incrementally change maps in the way required by real-time applications as new data is acquired. The key to why not is that the processing effort needed to calculate an estimate for each robot or camera location on a trajectory depends on the **total number** of locations. If the robot or camera moves to a new location and we wish to combine new measurement data with an existing map, all previous estimates must be revised. This does not fit our requirement for constant processing cost for sequential applications.

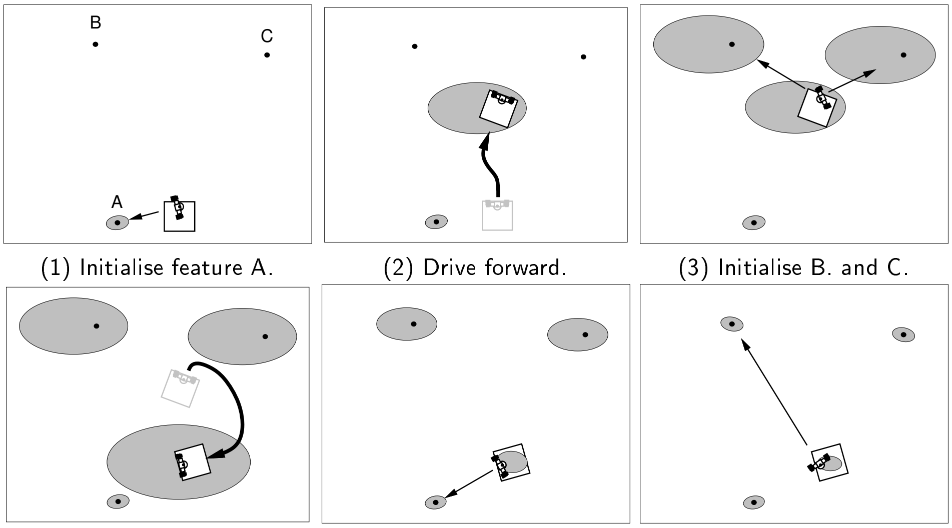
A comment on uncalibrated methods, which have become closely intertwined with batch estimation in computer vision: most advanced structure from motion approaches operate by assuming that certain parameters defining the camera's operation (such as the focal length) are completely unknown, and calculations take place in versions of the world which are warped in some unknown way relative to reality via the mathematics of projective geometry. Resolution to real Euclidean estimates only happens as a final step, often after an auto-calibration procedure. It should be remembered that estimating the unknown calibration parameters of a camera in this way is somewhat of a detail when it comes to the general problem of reconstructing the world from uncertain sensor measurements These warped worlds are of no use when we wish to incorporate other types of information; this could be data from other sensors, but most importantly we mean motion information. Motion models are inextricably tied to the real, Euclidean world — the argument may be made that things such as straight line motion are preserved under projective transformation, but physics is also about rotations and scales (such as that provided by the constant of gravitational acceleration). We argue that the best course for sequential methods is to place estimates in a Euclidean frame straight away. Of course some things may be uncertain, such as absolute scales or calibration parameters, but this uncertainty can be included explicitly in estimates and reduced as more data is acquired. We do not lose any deductive power by doing this: we only add it. As will be explained below, when the interdependence between estimates is propagated properly, we retain the ability to, for instance, estimate the ratio of the depths of two features with a high precision, even if either individually is poorly known.

Doing Things Sequentially: To tackle the sequential case, we need a representation of the current "state" of the system whose size does not vary over time. Then, this state can be updated in a constant time when new information arrives. Both the state and the new information will be accompanied by uncertainty, and we must take account of this when weighting the old and new data to produce updates. We are taken unavoidably into the domain of time-dependent **statistics**, whereas the optimisation approach used in batch methods permits a more lax handling of uncertainty.

Something to clarify early on is that when we talk here about a state being of constant size, we mean that for a map with a given number of features the state size does not change with **time**. The fact that the state size, and therefore processing burden, will increase as the number of features grows seems unavoidable. So to process maps in real-time, we will be limited to a finite number of features. How to deal with more features than this limit is the main challenge of sequential map-building research, and we will look at this further in Section 2.4.

Many early authors [8, 11, 1] took simple approaches to representing the state and its uncertainty; the locations of the moving robot in the world and features were stored and updated independently. However, if any type of long-term motion is attempted, these methods prove to be deficient: though they produce good estimates of instantaneous motion, they do not take account of the interdependence of the estimates of different quantities, and maps are seen to drift away from ground truth in a systematic way, as can be seen in the experiments of the authors referenced above. They are not able to produce sensible estimates for long runs where previously seen features may be revisited after periods of neglect, an action that allows drifting estimates to be corrected [7].

To give a flavour of the interdependence of estimates in sequential map-building, and emphasize that it is important to estimate robot and feature positions together, steps from a simple scenario are depicted in Figure 1. The



(1) Initialise feature A. (2) Drive forward. (3) Initialise B. and C.

(4) Drive back. (5) Re-measure A. (6) Re-measure B.

Fig. 1. Six steps in a example of sequential map-building, where a robot moving in two dimensions is assumed to have a fairly accurate sensor allowing it to detect the relative location of point features, and less accurate odometry for dead-reckoning motion estimation. Black points are the true locations of environmental features, and grey areas represent uncertain estimates of the feature and robot positions.

---

sequence of robot behaviour here is not intended to be optimal; the point is that a map-building algorithm should be able to cope with arbitrary actions and make use of all the information it obtains.

In (1), a robot is dropped into an environment of which it has no prior knowledge. Defining a coordinate frame at this starting position, it uses a sensor to identify feature A and measure its position. The sensor is quite accurate, but there is some uncertainty in this measurement which transposes into the small grey area representing the uncertainty in the estimate of the feature's position.

The robot drives forward in (2), during this time making an estimate of its motion using dead-reckoning (for instance counting the turns of its wheels). This type of motion estimation is notoriously inaccurate and causes motion uncertainties which grow without bound over time, and this is reflected in the large uncertainty region around the robot representing its estimate of its position. In (3), the robot makes initial measurements of features B and C. Since the robot's own position estimate is uncertain at this time, its estimates of the locations of B and C have large uncertainty regions, equivalent to the robot position uncertainty plus the smaller sensor measurement uncertainty. However, although it cannot be represented in the diagram, the estimates in the locations of the robot, B and C are all **coupled** at this point. Their relative positions are quite well known; what is uncertain is the position of the group as a whole.

The robot turns and drives back to near its starting position in (4). During this motion its estimate of its own position, again updated with dead-reckoning, grows even more uncertain. In (5) though, re-measuring feature A, whose absolute location is well known, allows the robot dramatically to improve its position estimate. The important thing to notice is that this measurement **also improves the estimate of the locations of features B and C**. Although the robot had driven farther since first measuring them, estimates of these feature positions were still partially coupled to the robot state, so improving the robot estimate also upgrades the feature estimates. The feature estimates are further improved in (6), where the robot directly re-measures feature B. This measurement, while of course improving the estimate of B, also improves C due to their interdependence (the relative locations of B and C are well known).

At this stage, all estimates are quite good and the robot has built a useful map. It is important to understand that this has happened with a small number of measurements because use has been made of the coupling between estimates.

2.3 Propagating Coupled Estimates

To work with coupled estimates, it is necessary to propagate not only each estimated quantity and its uncertainty, but also how this relates to the uncertainties of other estimates. Generally, a group of uncertain quantities is represented by a probability distribution in multiple dimensions, the form of which will depend on the specific agents of uncertainty in the system. Representing arbitrary probability distributions is not straightforward: one approach uses many random samples (sometimes called particles) to build up the shape, and has recently successfully been seen in vision in the form of the Condensation algorithm [12] for robust contour tracking. This approach has also been used in robot navigation for the problem of localisation using a known map [10], performing extremely well even for the difficult problem of re-localising a robot which is completely lost. However, these Monte Carlo methods are computationally expensive, and particularly are not applicable to the very high-dimensional map-building problem, since the number of particles N, and therefore the computational burden, needed to represent fairly a probability distribution in dimension d varies as:

$$N \geq \frac{D_{min}}{\alpha^d} \ ,$$

where D_{min} and α are constants with $\alpha << 1$. In modern Condensation applications, the number of dimensions under consideration is limited by computing power to perhaps something less than 10 if realtime operation is desired. This is of course sufficient for estimating the location of a robot with a known map, but not when we simultaneously need to estimate map parameters.

Currently more feasible is to propagate first order approximations to probability distributions. Each estimated parameter is accompanied by single numbers representing its variance and covariance with other parameters — a vector of parameters has a covariance matrix filled with these elements. The Kalman Filter is an optimal solution to linear problems in which all noise sources are gaussian in profile; however, most map-building scenarios are not linear so in these cases the Extended Kalman Filter provides an approximation which in general has been found to perform very well. Called "Stochastic Mapping" in its first correctly-formulated application to robot map-building [18], the EKF has been implemented successfully in different scenarios by other researchers [2, 4, 5, 7, 9, 14]. Its main weakness compared to the Monte Carlo methods is its inability to represent multi-modal distributions — where an estimate has two or more peak values that are most likely, with unlikely regions in between. The Monte Carlo methods gain greatly in robustness through this ability. In first order approaches, multiple hypotheses must be considered externally and explicitly.

2.4 The Problems

We consider that there are two main challenges in sequential map-building using first order uncertainty — they are both things we have already touched on:

1. The growth of computational complexity with the number of map features.
2. Coping with mismatches (sometimes known as data association).

We will see in Section 3 that correctly propagating the interdependence of map estimates requires a covariance matrix to be updated whose number of elements grows as the square of the number of estimated parameters. Clearly, the processing requirements quickly get out of hand as the map grows.

The first approach which can be taken is to keep the number of features low using sensible map management. With today's processing power, there need be little trouble with maintaining maps with features in the 50–100 range at decent update rates. This is quite sufficient for many localisation tasks within confined areas: the emphasis should be on using a small number of high-quality features, and on **map using** rather than wasteful map building. For the situation in question, the question "how many features need to be measurable at any given time?" can be asked. The answer will depend on the localising power of a single feature measurement, which depends on the sensor and feature types (for instance, for a camera moving in 3D, seeing just a single point feature will not improve estimates along the several degrees of freedom which that measurement does not constrain, whereas for a robot moving in one dimension with a range sensor, measuring one feature tells it all it needs to know), as well as potential desires for redundant measurements to improve robustness (see Section 3.5). A management algorithm can then add new features to the map only in places where less than this desired number is available.

For applications where the number of features must be larger, various authors have looked at ways to relieve the complexity of large maps. One simple approach, similar to the map management above, is to delete features from the map which do not provide much information[9]: for instance, if two features lie close together, and their estimates have become closely coupled, one can be deleted without sacrificing much information. This opens up the question of which features provide the best localisation information — something also looked at in [5] with regard to active choice among candidate measurements.

Other approaches split a large map into sub-maps, within each of which fully-coupled map-building goes on as normal but between which full coupling is not maintained [4, 15]. While in a certain area, the robot will only observe features in the current submap and only update estimates for this submap in real-time. This is an approximation, since in truth each feature estimate is related to every other in the world map, but can be effective when the coupling between submaps can be represented as a single parameter: all the estimates in submap A have a similar dependence on all of those in submap B. This is likely to be the case if the robot spends long periods of time confined to each submap region, rather than frequently moving between them. There are still many issues to settle with these approaches, such as how submap areas should be delineated.

McLauchlan's VSDF [16] is a powerful framework which marries sequential and batch methods, and has been used in several different vision applications. It is based on the propagation of inverse covariance matrices (called information matrices), a strategy which provides some computational advantages, and offers an efficient sequential mode, though this mode makes implicit approximations by ignoring some non-diagonal matrix elements and it is not clear how these approximations compare with other possibilities. Covariance Intersection [23], a general tool for distributed estimation, has also been touted as suitable for map-building, though its generalisations may be too forgiving to produce estimates as good as other more specific methods.

A final approach [5] suggests that while maintaining a full single map, current active parts can be kept fully up to date while currently uninteresting regions are kept on "the back burner" — or perhaps in a hierarchy of updatedness — and proves that it is possible to bring these back up to date at a later stage with no loss of information. This method is certainly interesting, particularly because the hierarchy idea provides possibilities for multiple-hypothesis branching at various levels, but presents large challenges in terms of management: deciding which parts should be updated or left to simmer.

We will look in Section 3.5 at how it is possible to use redundancy and methods like RANSAC to reduce the chance of falling prey to mismatches at the measurement stage, but more generally dealing with this problem requires a multiple-hypothesis approach where estimates fork and the decision on which branch is correct is postponed until more evidence is available. There have yet to be any convincing demonstrations of how this can be incorporated into rigorous sequential map-building. This should be a major focus of future research, since there is no point in improving efficiency with methods which are still prone to instant failure at a mismatch. In our experience, map-building of the type described in Section 3 often can survive a mismatch, though this is by luck since the method includes no model of these events.

3 Map-Building with First Order Uncertainty Propagation: Details and Insights

In the following section we will look in detail at sequential map-building using first-order uncertainty propagation. On its own this represents an obvious and rigorous approach to map-building, but it is also the backbone of the methods described in the previous section for improving efficiency. We will refer throughout to the moving sensor body as "the robot", but this can apply to a single camera or other unrobotlike object. "Feature" is also a general term referring to any object the robot is capable of observing. Features can be points, lines, planes, cylinders or any other type of geometrical object.

3.1 The State Vector and its Covariance

Current estimates of the state of the robot and the scene features which are known about are stored in the system state vector $\hat{\mathbf{x}}$, and the uncertainty of the estimates in the covariance matrix $\mathtt{P}$. $\hat{\mathbf{x}}$ and $\mathtt{P}$ will change in size dynamically as features are added to or deleted from the map. They are partitioned as follows:

$$\hat{\mathbf{x}} = \begin{pmatrix} \hat{\mathbf{x}}_v \\ \hat{\mathbf{y}}_1 \\ \hat{\mathbf{y}}_2 \\ \vdots \end{pmatrix} \quad , \quad \mathtt{P} = \begin{bmatrix} \mathtt{P}_{xx} & \mathtt{P}_{xy_1} & \mathtt{P}_{xy_2} & \dots \\ \mathtt{P}_{y_1x} & \mathtt{P}_{y_1y_1} & \mathtt{P}_{y_1y_2} & \dots \\ \mathtt{P}_{y_2x} & \mathtt{P}_{y_2y_1} & \mathtt{P}_{y_2y_2} & \dots \\ \vdots & \vdots & \vdots & \end{bmatrix} .$$

$\hat{\mathbf{x}}_v$ is the robot state estimate, and $\hat{\mathbf{y}}_i$ the estimated state of the ith feature. By the "state" of the robot and features, generally we mean a vector of all the parameters in which we are interested relating to those objects. Of course this means their positions, which can be defined by a number of parameters depending on the geometrical type of the object and dimensionality of the map; but also, there may be other parameters which we would like the estimate, usually because they will affect future motion or measurements.

For dynamically moving objects it is necessary to estimate higher-order motion parameters (velocity, acceleration, etc.). The number of derivatives needed depends on the expected motion (see Section 3.3 about Motion). As another example, a robot may have redundant axes of movement whose status is important but which are not uniquely defined by the robot's geometrical position. These extra parameters can also be used for **calibration constants**, which are initially only approximately known but whose accuracy will improve with the evolution of the system.

3.2 Coordinate Frames and Initialisation

When a robot moves in surroundings which are initially completely unknown, the choice of a world coordinate frame is arbitrary. The only things that can be reported on are the location of the robot relative to any features detected. Indeed, one possible approach is to do away with a world coordinate frame altogether and estimate just the locations of features in a frame which it fixed to the robot; robot motions appear simply as backward feature movements. However, there is not a large computational penalty in including an explicit robot state, and more importantly in most applications of map-building there will be some interaction with information from other sources, which could be in the form of some prior-known feature positions, or maybe simply metric way-points through which the robot is required to move — a world coordinate frame is necessary to interact with information of this kind.

If there is no prior knowledge of the environment, the coordinate frame can be defined to have its origin at the robot's starting position, and the initial uncertainty relating to the robot's position in $\mathtt{P}_{xx}$ is set to zero. If there is prior knowledge of some feature locations, this is put into the map explicitly and it is this which defines the coordinate frame. The robot's starting position relative to these features must also be input, and both robot and feature positions should be assigned suitable initial covariance values. It is not reasonable to set both robot and feature covariances to zero, because their relative locations can never be perfectly known; a typical initial situation would be to have several very well known feature positions with low covariance effectively pinning down the coordinate frame, with a more uncertain robot starting location.

3.3 Motion

What happens to the estimate of the robot's position during a movement? The answer is that we should model its movement as well as we can with a motion model $\mathbf{f}_v(\mathbf{x}_v, \mathbf{u})$, and add to the system covariance to account for our uncertainty in this motion estimate (the process noise $\mathtt{Q}$).

In batch structure from motion, there is typically no motion modelling. The assumption made is that at each new camera position, there is no prior location knowledge; that is to say there is infinite uncertainty (though there may be constraints on some movement dimensions in certain scenarios). In the quasi-static case that these methods are applied to this is sensible. However, when working in the time domain there is always extra information to be had by modelling motion. This model may be very simple or vague, but the best thing to do is to set it up as honestly as possible and make use of it. Quoting from Torr *et al.* [21], who in turn cite Jaynes [13]:

> Some will complain that to use Bayesian methods one must introduce arbitrary priors on the parameters. However, far from being a disadvantage, this is a tremendous advantage as it forces open acknowledgement of what assumptions were used in designing the algorithm, which all too often are hidden away beneath the veneer of equations.

There are many types of motion model depending on the level of our knowledge about the system. In the case that we have knowledge of the control parameters of a robot (such as "drive forward at 1ms^{-1} for one second with a steering angle of 5°"), which is the case for a robot which is controlling its own navigation, we can potentially make quite an accurate motion estimate and the process noise covariance will be small. However, if we want to estimate the motion of a camera strapped to an independently manoeuvring human head, we can make much less precise assumptions: for instance, that the head will keep moving more or less at it's current speed, or maybe that it is slightly more likely to slow down than speed up, given that the person is probably moving within a confined space (see the auto-regressive models of [12]).

Process noise accounts for the things that we don't attempt to model. There is of course no such thing as random noise in (classical!) physics: bodies move deterministically. It is just that we can't know all the details of what is happening to them, though in theory we could model everything (slipping wheels; human joints, muscles, chemical reactions!). Models have to stop somewhere: the rest of what is going on we call process noise, and estimate the size of its effect.

The robot's motion is discretised into finite steps, with an incrementing label k affixed to each. There is no need for the length of these steps to be equal in time, though this will often be the case. (One point we would like to highlight is that many navigation researchers have used unnecessarily simple motion models for their mobile robots; e.g. [8], where a model for car-like motion is used which is an approximation for small Δt: in this case it is quite straightforward to construct a motion models which does not require short time steps [5].) In a Jacobian calculation, we change the state and covariance as follows:

$$\hat{\mathbf{x}}(k+1|k) = \begin{pmatrix} \mathbf{f}_v(\mathbf{x}_v(k|k), \mathbf{u}(k)) \\ \hat{\mathbf{y}}_1(k|k) \\ \hat{\mathbf{y}}_2(k|k) \\ \vdots \end{pmatrix} \tag{1}$$

$$\mathtt{P}(k+1|k) = \begin{bmatrix} \frac{\partial \mathbf{f}_v}{\partial \mathbf{x}_v} \mathtt{P}_{xx}(k|k) \frac{\partial \mathbf{f}_v}{\partial \mathbf{x}_v}^\top + \mathtt{Q}(k) & \frac{\partial \mathbf{f}_v}{\partial \mathbf{x}_v} \mathtt{P}_{xy_1}(k|k) & \frac{\partial \mathbf{f}_v}{\partial \mathbf{x}_v} \mathtt{P}_{xy_2}(k|k) & \dots \\ \mathtt{P}_{y_1x}(k|k) \frac{\partial \mathbf{f}_v}{\partial \mathbf{x}_v}^\top & \mathtt{P}_{y_1y_1}(k|k) & \mathtt{P}_{y_1y_2}(k|k) & \dots \\ \mathtt{P}_{y_2x}(k|k) \frac{\partial \mathbf{f}_v}{\partial \mathbf{x}_v}^\top & \mathtt{P}_{y_2y_1}(k|k) & \mathtt{P}_{y_2y_2}(k|k) & \dots \\ \vdots & \vdots & \vdots & \end{bmatrix} \tag{2}$$

3.4 Measurements: Selection, Prediction and Searching

The way to measure a particular feature i is determined by its feature measurement model $\mathbf{h}_i(\mathbf{x}_v, \mathbf{y}_i)$, and the measurement noise $\mathtt{R}$. Analogous to process noise, measurement noise takes account of the things we don't model in the feature measurement model. Whenever we wish to measure a particular feature, the value of the measurement can be predicted by substituting current estimates $\hat{\mathbf{x}}_v$ and $\hat{\mathbf{y}}_i$ into the expression for $\mathbf{h}_i$. From the predicted value of a measurement, we can calculate, based on knowledge about the particular feature type and saved information on what the first initialisation measurement of this feature was, whether it is worth trying to measure it. For instance, when measuring point features visually with correlation, there is little chance of a successful match if the current viewpoint is far from the original viewpoint. In this way, regions of **measurability** can be defined for each feature, and aid robustness by only allowing match attempts from positions where the chances are good.

The innovation covariance $\mathtt{S}_i$ is how much the actual measurement $\mathbf{z}_i$ is expected to deviate from this prediction:

$$\mathtt{S}_i = \frac{\partial \mathbf{h}_i}{\partial \mathbf{x}_v} \mathtt{P}_{xx} \frac{\partial \mathbf{h}_i}{\partial \mathbf{x}_v}^\top + \frac{\partial \mathbf{h}_i}{\partial \mathbf{x}_v} \mathtt{P}_{xy_i} \frac{\partial \mathbf{h}_i}{\partial \mathbf{y}_i}^\top + \frac{\partial \mathbf{h}_i}{\partial \mathbf{y}_i} \mathtt{P}_{y_i x} \frac{\partial \mathbf{h}_i}{\partial \mathbf{x}_v}^\top + \frac{\partial \mathbf{h}_i}{\partial \mathbf{y}_i} \mathtt{P}_{y_i y_i} \frac{\partial \mathbf{h}_i}{\partial \mathbf{y}_i}^\top + \mathtt{R} \ . \quad (3)$$

Calculating $\mathtt{S}_i$ before making measurements allows us to form a **search region** in measurement space for each feature, at a chosen number of standard deviations. This is a large advantage because it allows the adoption of an **active** approach: we need only direct searching attention to the this area, maximising computational resources and minimising the chance of obtaining a mismatch.

It is also possible to make decisions on which of several potentially measurable features to observe as a priority based on $\mathtt{S}_i$: if the measurement cost for each candidate is similar, it is favourable to make a measurement where $\mathtt{S}_i$ has a high value because there is the most information to be gained here. There is no point in making a measurement when the result is predictable [7]. If measurements are continually chosen like this, the uncertainty in any particular part of the map can be stopped from getting out of control, a situation which would lead to large search regions and the high possibility of mismatches, and to the potential breaking of the linearisation approximations of the EKF.

3.5 Updating After a Measurement

After attempting measurements, those that were successful are used to update the state estimates. How do we know which were successful? Clearly there are some cases where failures are apparent, by matching scores below a given threshold for instance. However, in other cases we won't know: something has been found within the innovation covariance-bounded search region, but is it the feature we were looking for or just something else that looks the same?

One way to pick the good measurements from the bad is make a lot of measurements at the same time, and then look for sets among them which agree with each other: these are likely to be the correct matches, since no correlation is expected amongst the failures — this algorithm, RANSAC, is commonly used to lend robustness to batch methods [22]. To use RANSAC here, we try the update with randomly selected subsets of the measurements, and look for updated robot position estimates which agree. The bad matches can then be marked as such and the update performed with just the good ones.

To update the map based on a set of measurements $\mathbf{z}_i$, we perform EKF updates as below. Because the measurements are independent, these updates can be done in sequence, rather than stacking all the measurements into one large vector and doing everything at once (this is computationally beneficial because smaller $\mathtt{S}$ matrices are inverted). Note further that if a particular $\mathbf{z}_i$ has diagonal measurement noise $\mathtt{R}$ we can further subdivide to the individual measurement parameters for sequential updates.

For each independent measurement $\mathbf{h}_i$, the Kalman gain $\mathtt{W}$ can be calculated and the state updated as follows:

$$\mathtt{W} \;=\; \mathtt{P}\frac{\partial \mathbf{h}_i}{\partial \mathbf{x}}^\top \mathtt{S}_i^{-1} \;=\; \begin{pmatrix} \mathtt{P}_{xx} \\ \mathtt{P}_{y_1 x} \\ \mathtt{P}_{y_2 x} \\ \vdots \end{pmatrix} \frac{\partial \mathbf{h}_i}{\partial \mathbf{x}_v}^\top \mathtt{S}_i^{-1} + \begin{pmatrix} \mathtt{P}_{x y_i} \\ \mathtt{P}_{y_1 y_i} \\ \mathtt{P}_{y_2 y_i} \\ \vdots \end{pmatrix} \frac{\partial \mathbf{h}_i}{\partial \mathbf{y}_i}^\top \mathtt{S}_i^{-1} \tag{4}$$

$$\hat{\mathbf{x}}_{new} = \hat{\mathbf{x}}_{old} + \mathtt{W}(\mathbf{z}_i - \mathbf{h}_i) \tag{5}$$

$$\mathtt{P}_{new} = \mathtt{P}_{old} - \mathtt{W}\mathtt{S}_i\mathtt{W}^\top \; . \tag{6}$$

3.6 Initialising a New Feature

When an unknown feature is first observed, a measurement $\mathbf{z}_i$ is obtained of its position relative to the robot. If the measurement function $\mathbf{h}_i(\mathbf{x}_v, \mathbf{y}_i)$ is invertible to $\mathbf{y}_i(\mathbf{x}_v, \mathbf{h}_i)$, we can initialise the feature state as follows (assuming here that two features are known and the new one becomes the third):

$$\mathbf{x}_{new} = \begin{pmatrix} \mathbf{x}_v \\ \mathbf{y}_1 \\ \mathbf{y}_2 \\ \mathbf{y}_i \end{pmatrix} \tag{7}$$

$$\mathtt{P}_{new} = \begin{bmatrix} \mathtt{P}_{xx} & \mathtt{P}_{xy_1} & \mathtt{P}_{xy_2} & \mathtt{P}_{xx}\frac{\partial \mathbf{y}_i}{\partial \mathbf{x}_v}^\top \\ \mathtt{P}_{y_1 x} & \mathtt{P}_{y_1 y_1} & \mathtt{P}_{y_1 y_2} & \mathtt{P}_{y_1 x}\frac{\partial \mathbf{y}_i}{\partial \mathbf{x}_v}^\top \\ \mathtt{P}_{y_2 x} & \mathtt{P}_{y_2 y_1} & \mathtt{P}_{y_2 y_2} & \mathtt{P}_{y_2 x}\frac{\partial \mathbf{y}_i}{\partial \mathbf{x}_v}^\top \\ \frac{\partial \mathbf{y}_i}{\partial \mathbf{x}_v}\mathtt{P}_{xx} & \frac{\partial \mathbf{y}_i}{\partial \mathbf{x}_v}\mathtt{P}_{xy_1} & \frac{\partial \mathbf{y}_i}{\partial \mathbf{x}_v}\mathtt{P}_{xy_2} & \frac{\partial \mathbf{y}_i}{\partial \mathbf{x}_v}\mathtt{P}_{xx}\frac{\partial \mathbf{y}_i}{\partial \mathbf{x}_v}^\top + \frac{\partial \mathbf{y}_i}{\partial \mathbf{h}_i}\mathtt{R}\frac{\partial \mathbf{y}_i}{\partial \mathbf{h}_i}^\top \end{bmatrix} \tag{8}$$

It should be noted that some bias is introduced into the map in initialising features in this way if (as is usual) the measurement process is non-linear.

If $\mathbf{h}_i$ is not invertible, it means that a single measurement does not give enough information to pinpoint the feature location (for instance a single view of a point feature from a single camera only defines a ray on which it lies). The approach that must be followed here is to initialise it into the map as a **partially initialised feature** $\mathbf{y}_{pi}$, with a different geometrical type (e.g. a line feature to represent the ray we know a point must lie on), and wait until another measurement from a different viewpoint allows resolution. At this stage a special second initialisation function $\mathbf{y}_i(\mathbf{x}_v, \mathbf{y}_{pi}, \mathbf{z}_i)$ allows the actual state $\mathbf{y}_i$ to be determined from the partially initialised state and new measurement (feature types which require more than two steps for initialisation are also possible).

Once initialised, a feature has exactly the same status in the map as those whose positions may have been give as prior knowledge.

3.7 Deleting a Feature

Deleting a feature from the state vector and covariance matrix is a simple case of removing the rows and columns which contain it. An example in a system where the second of three known features is deleted would be:

$$\begin{pmatrix} \mathbf{x}_v \\ \mathbf{y}_1 \\ \mathbf{y}_2 \\ \mathbf{y}_3 \end{pmatrix} \rightarrow \begin{pmatrix} \mathbf{x}_v \\ \mathbf{y}_1 \\ \mathbf{y}_3 \end{pmatrix} \quad , \quad \begin{bmatrix} P_{xx} & P_{xy_1} & P_{xy_2} & P_{xy_3} \\ P_{y_1x} & P_{y_1y_1} & P_{y_1y_2} & P_{y_1y_3} \\ P_{y_2x} & P_{y_2y_1} & P_{y_2y_2} & P_{y_2y_3} \\ P_{y_3x} & P_{y_3y_1} & P_{y_3y_2} & P_{y_3y_3} \end{bmatrix} \rightarrow \begin{bmatrix} P_{xx} & P_{xy_1} & P_{xy_3} \\ P_{y_1x} & P_{y_1y_1} & P_{y_1y_3} \\ P_{y_3x} & P_{y_3y_1} & P_{y_3y_3} \end{bmatrix} . \quad (9)$$

In automated map-maintenance, features can be deleted if a large proportion of attempted measurements fail when the feature is expected to be measurable. This could be due to features not fitting the assumptions of their model (an assumed point feature which in fact contains regions of different depths and therefore appears very different from a new viewpoint for instance), or possibly occlusion — leading to the survival of features which do not suffer these fates.

4 Software and Implementations

Realisation of the generic properties of the sequential map-building problem and experience with different robot systems has led to the evolution of our original application-specific software into a general framework called *Scene*, efficiently implemented in C++ and designed with orthogonal axes of flexibility in mind:

1. Use in many different application domains; from multiple robots navigating in 1D, 2D or 3D with arbitrary sensing capabilities, to single cameras.
2. Implementation of different mapping algorithms and approaches to dealing with the complexity of sequential map-building.

Scene is now available with full source code (under the GNU Lesser General Public License), at **http://www.robots.ox.ac.uk/~ajd/** . The distribution package includes interactive simulations precompiled for Linux which allow immediate hands-on experience of sequential map-building in several real and simplified problem domains, the additional tools which turn these into systems operating with real hardware, and substantial documentation.

To give an impression of how the general framework can be applied to various systems, details of some current and planned implementations, differentiated by motion and feature measurement models plugged in as modules, are given in Table 1. The simplest is a one-dimensional test-bed, which is very useful for looking at what happens to robot and feature covariances in various situations and under different algorithms. Our main work to date has been on robot navigation using active vision [5,7,6], using mobile platforms which move in 2D and are equipped with steerable camera platforms which make measurements of point features in 3D with stereo vision. As detailed in [6], the software's motion model formulation is flexible enough to permit cooperative position estimation

	1D Test-bed	Nomad Robot with Stereo Active Vision	Camera Position Tracking
World Dimension	1	3	3
Position Dimension	1	2	3
Motion Model	Velocity Only	Steer and Drive	General Motion in 3D
State Size	1	5	12
Control Size	1	3	0
Feature Measurement Model	Point Range Measurement	3D Stereo Point Measurement	Single Image Point Measurement
Feature Dimension	1	3	3
Measurement Size	1	4	2

Table 1. Specifications for various implementations.

by cooperating robots, where one has stereo active vision and the other is blind, navigating primarily by odometry.

Current PC computers are powerful enough to perform correlation searches for many features at video frame rate. Our current goal is to apply the *Scene* library to real-time camera position tracking using just inside-out image measurements, potentially the "killer app" of sequential localisation and map-building, which would be useful in applications such as inside-out head tracking or the real-time virtual studio. The first demonstrations of this type have just started to appear [3]. To be successful, it will be necessary to make use of many of the details explained in this paper. For instance, RANSAC or similar must be used to detect failed matches fast because there will not be enough processing time available to propagate multiple hypothesis. A full 3D motion model must be used, and finally, it will be necessary to use partially initialised feature representations to bootstrap features in 3D from multiple views.

References

1. P. A. Beardsley, I. D. Reid, A. Zisserman, and D. W. Murray. Active visual navigation using non-metric structure. In *Proceedings of the 5th International Conference on Computer Vision, Boston*, pages 58–65. IEEE Computer Society Press, 1995.
2. J. A. Castellanos. *Mobile Robot Localization and Map Building: A Multisensor Fusion Approach.* PhD thesis, Universidad de Zaragoza, Spain, 1998.
3. A. Chiuso, P. Favaro, H. Jin, and S. Soatto. "mfm": 3-d motion from 2-d motion causally integrated over time. In *Proceedings of the 6th European Conference on Computer Vision, Dublin*, 2000.
4. K. S. Chong and L. Kleeman. Feature-based mapping in real, large scale environments using an ultrasonic array. *International Journal of Robotics Research*, 18(2):3–19, January 1999.
5. A. J. Davison. *Mobile Robot Navigation Using Active Vision.* PhD thesis, University of Oxford, 1998. Available at http://www.robots.ox.ac.uk/~ajd/.

6. A. J. Davison and N. Kita. Active visual localisation for cooperating inspection robots. In *In Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems*, 2000.
7. A. J. Davison and D. W. Murray. Mobile robot localisation using active vision. In *Proceedings of the 5th European Conference on Computer Vision, Freiburg*, pages 809–825, 1998.
8. H. F. Durrant-Whyte. Where am I? A tutorial on mobile vehicle localization. *Industrial Robot*, 21(2):11–16, 1994.
9. H. F. Durrant-Whyte, M. W. M. G. Dissanayake, and P. W. Gibbens. Toward deployments of large scale simultaneous localisation and map building (slam) systems. In *Proceedings of the 9th International Symposium of Robotics Research, Snowbird, Utah*, pages 121–127, 1999.
10. D. Fox, W. Burgard, H. Kruppa, and S. Thrun. Efficient multi-robot localization based on monte carlo approximation. In *Proceedings of the 9th International Symposium of Robotics Research, Snowbird, Utah*, pages 113–120, 1999.
11. C. G. Harris. Geometry from visual motion. In A. Blake and A. Yuille, editors, *Active Vision*. MIT Press, Cambridge, MA, 1992.
12. M. Isard and A. Blake. Contour tracking by stochastic propagation of conditional density. In *Proceedings of the 4th European Conference on Computer Vision, Cambridge*, pages 343–356, 1996.
13. E. T. Jaynes. Probability theory as extended logic. Technical report, Washington University in St. Louis, 1999. Web site: http://bayes.wustl.edu/.
14. Y. D. Kwon and J. S. Lee. A stochastic map building method for mobile robot using 2-d laser range finder. *Autonomous Robots*, 7:187–200, 1999.
15. J. J. Leonard and H. J. S. Feder. Decoupled stochastic mapping, part i: Theory. Preprint. Submitted to IEEE Transactions on Robotics and Automation, 1999.
16. P. F. McLauchlan and D. W. Murray. A unifying framework for structure and motion recovery from image sequences. In *Proceedings of the 5th International Conference on Computer Vision, Boston*. IEEE Computer Society Press, 1995.
17. M. Pollefeys, R. Koch, and L. Van Gool. Self-calibration and metric reconstruction in spite of varying and unknown internal camera parameters. In *Proceedings of the 6th International Conference on Computer Vision, Bombay*, pages 90–96, 1998.
18. R. Smith, M. Self, and P. Cheeseman. A stochastic map for uncertain spatial relationships. In *4th International Symposium on Robotics Research*, 1987.
19. S. Thrun, D. Fox, and W. Burgard. A probabilistic approach to concurrent mapping and localization for mobile robots. *Machine Learning*, 31, 1998.
20. P. H. S. Torr, A. W. Fitzgibbon, and A. Zisserman. Maintaining multiple motion model hypotheses over many views to recover matching and structure. In *Proceedings of the 6th International Conference on Computer Vision, Bombay*, pages 485–491, 1998.
21. P. H. S. Torr, R. Szeliski, and P. Anandan. An integrated bayesian approach to layer extraction from image sequences. In *Proceedings of the 7th International Conference on Computer Vision, Kerkyra*, pages 983–990, 1999.
22. P. H. S. Torr and A. Zisserman. Robust computation and parameterization of multiple view relations. In *Proceedings of the 6th International Conference on Computer Vision, Bombay*, pages 727–732, 1998.
23. J. K. Uhlmann, S. J. Julier, and M. Csorba. Nondivergent simultaneous map building and localisation using covariance intersection. In *The Proceedings of AeroSense: The 11th International Symposium on Aerospace/Defense Sensing, Simulation and Controls, Orlando, Florida*. SPIE, 1997. Navigation and Control Technologies for Unmanned Systems II.