25. Multisensor Data Fusion

Multisensor data fusion is the process of combining observations from a number of different sensors to provide a robust and complete description of an environment or process of interest. Data fusion finds wide application in many areas of robotics such as object recognition, environment mapping, and localisation.

This chapter has three parts: methods, architectures and applications. Most current data fusion methods employ probabilistic descriptions of observations and processes and use Bayes' rule to combine this information. This chapter surveys the main probabilistic modeling and fusion techniques including grid-based models, Kalman filtering and sequential Monte Carlo techniques. This chapter also briefly reviews a number of non-probabilistic data fusion methods. Data fusion systems are often complex combinations of sensor devices, processing and fusion algorithms. This chapter provides an overview of key principles in data fusion architectures from both a hardware and algorithmic viewpoint. The applications of data fusion are pervasive in robotics and underly the core problem of sensing, estimation and perception. We highlight two example applications that bring out these features. The first describes a navigation or self-tracking application for an autonomous vehicle. The second describes an application in mapping and environment modeling.

The essential algorithmic tools of data fusion are reasonably well established. However, the development and use of these tools in realistic robotics applications is still developing.

25.1 Multisensor Data Fusion Methods

The most widely used data fusion methods employed in robotics originate in the fields of statistics, estimation and control. However, the application of these methods in robotics has a number of unique features and challenges. In particular, most often autonomy is the goal and so results must be presented and interpreted in a form from which autonomous decisions can be made; for recognition or navigation, for example.

In this section we review the main data fusion methods employed in robotics. These are very often based on probabilistic methods, and indeed probabilistic methods are now considered the standard approach to data fusion in all robotics applications [25.1]. Probabilistic data fusion methods are generally based on Bayes' rule for combining prior and observation information. Practically, this may be implemented in a number of ways: through the use of the Kalman and extended

Kalman filters, through sequential Monte Carlo methods, or through the use of functional density estimates. Each of these is reviewed. There are a number of alternatives to probabilistic methods. These include the theory of evidence and interval methods. Such alternative techniques are not as widely used as they once were, however they have some special features that can be advantageous in specific problems. These, too, are briefly reviewed.

25.1.1 Bayes' Rule

Bayes' rule lies at the heart of most data fusion methods. In general, Bayes' rule provides a means to make inferences about an object or environment of interest described by a state $\boldsymbol{x}$, given an observation $\boldsymbol{z}$.

Bayesian Inference

Bayes' rule requires that the relationship between $\boldsymbol{x}$ and $\boldsymbol{z}$ be encoded as a joint probability or joint probability distribution $P(\boldsymbol{x}, \boldsymbol{z})$ for discrete and continuous variables respectively. The chain-rule of conditional probabilities can be used to expand a joint probability in two ways

$$P(\boldsymbol{x}, \boldsymbol{z}) = P(\boldsymbol{x} \mid \boldsymbol{z})P(\boldsymbol{z}) = P(\boldsymbol{z} \mid \boldsymbol{x})P(\boldsymbol{x}). \tag{25.1}$$

Rearranging in terms of one of the conditionals, Bayes' rule is obtained

$$P(\boldsymbol{x} \mid \boldsymbol{z}) = \frac{P(\boldsymbol{z} \mid \boldsymbol{x})P(\boldsymbol{x})}{P(\boldsymbol{z})}. \tag{25.2}$$

The value of this result lies in the interpretation of the probabilities $P(\boldsymbol{x} \mid \boldsymbol{z})$, $P(\boldsymbol{z} \mid \boldsymbol{x})$, and $P(\boldsymbol{x})$. Suppose it is necessary to determine the various likelihoods of different values of an unknown state $\boldsymbol{x}$. There may be prior beliefs about what values of $\boldsymbol{x}$ might be expected, encoded in the form of relative likelihoods in the *prior probability* $P(\boldsymbol{x})$. To obtain more information about the state $\boldsymbol{x}$ an observation $\boldsymbol{z}$ is made. These observations are modeled in the form of a conditional probability $P(\boldsymbol{z} \mid \boldsymbol{x})$ which describes, for each fixed state $\boldsymbol{x}$, the probability that the observation $\boldsymbol{z}$ will be made; i. e., the probability of $\boldsymbol{z}$ given $\boldsymbol{x}$. The new likelihoods associated with the state $\boldsymbol{x}$ are computed from the product of the original prior information and the information gained by observation. This is encoded in the *posterior probability* $P(\boldsymbol{x} \mid \boldsymbol{z})$ which describes the likelihoods associated with $\boldsymbol{x}$ given the observation $\boldsymbol{z}$. In this fusion process, the marginal probability $P(\boldsymbol{z})$ simply serves to normalize the posterior and is not generally computed. The marginal $P(\boldsymbol{z})$ plays an important role in model validation or data association as it provides a measure of how well the observation is predicted by the prior. This is because $P(\boldsymbol{z}) = \int P(\boldsymbol{z} \mid \boldsymbol{x})P(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}$. The value of Bayes' rule is that it provides a principled means of combining observed information with prior beliefs about the state of the world.

Sensor Models and Multisensor Bayesian Inference

The conditional probability $P(\boldsymbol{z} \mid \boldsymbol{x})$ serves the role of a *sensor model* and can be thought of in two ways. First, in building a sensor model, the probability is constructed by fixing the value of $\boldsymbol{x} = x$ and then asking what probability density $P(\boldsymbol{z} \mid \boldsymbol{x} = x)$ on $\boldsymbol{z}$ results. Conversely, when this sensor model is used and observations are made, $\boldsymbol{z} = z$ is fixed and a *likelihood function* $P(\boldsymbol{z} = z \mid \boldsymbol{x})$ on $\boldsymbol{x}$ is inferred. The likelihood function, while not strictly a probability density, models the relative likelihood that different values of $\boldsymbol{x}$ gave rise to the observed value of $\boldsymbol{z}$. The product of this likelihood with the prior, both defined on $\boldsymbol{x}$, gives the posterior or observation update $P(\boldsymbol{x} \mid \boldsymbol{z})$. In a practical implementation of Equation025-bayeseq, $P(\boldsymbol{z} \mid \boldsymbol{x})$ is constructed as a function of both variables (or a matrix in discrete form). For each fixed value of $\boldsymbol{x}$, a probability density on $\boldsymbol{z}$ is defined. Therefore as $\boldsymbol{x}$ varies, a family of likelihoods on $\boldsymbol{z}$ is created.

The multisensor form of Bayes' rule requires conditional independence

$$\begin{aligned} P(\boldsymbol{z}_1, \cdots, \boldsymbol{z}_n \mid \boldsymbol{x}) &= P(\boldsymbol{z}_1 \mid \boldsymbol{x}) \cdots P(\boldsymbol{z}_n \mid \boldsymbol{x}) \\ &= \prod_{i=1}^{n} P(\boldsymbol{z}_i \mid \boldsymbol{x})\,. \end{aligned} \tag{25.3}$$

qso that

$$P(\boldsymbol{x} \mid \boldsymbol{Z}^n) = CP(\boldsymbol{x}) \prod_{i=1}^{n} P(\boldsymbol{z}_i \mid \boldsymbol{x})\,, \tag{25.4}$$

where C is a normalising constant. Equation (25.4) is known as the *independent likelihood pool* [25.2]. This states that the posterior probability on $\boldsymbol{x}$ given all observations $\boldsymbol{Z}^n$, is simply proportional to the product of prior probability and individual likelihoods from each information source.

The recursive form of Bayes' rule is

$$P(\boldsymbol{x} \mid \boldsymbol{Z}^k) = \frac{P(\boldsymbol{z}_k \mid \boldsymbol{x})P(\boldsymbol{x} \mid \boldsymbol{Z}^{k-1})}{P\left(\boldsymbol{z}_k \mid \boldsymbol{Z}^{k-1}\right)}\,. \tag{25.5}$$

The advantage of (25.5) is that we need compute and store only the posterior density $P(\boldsymbol{x} \mid \boldsymbol{Z}^{k-1})$ which contains a complete summary of all past information. When the next piece of information $P(\boldsymbol{z}_k \mid \boldsymbol{x})$ arrives, the previous posterior takes on the role of the current prior and

the product of the two becomes, when normalised, the new posterior.

Bayesian Filtering

Filtering is concerned with the sequential process of maintaining a probabilistic model for a state which evolves over time and which is periodically observed by a sensor. Filtering forms the basis for many problems in tracking and navigation. The general filtering problem can be formulated in Bayesian form. This is significant because it provides a common representation for a range of discrete and continuous data fusion problems without recourse to specific target or observation models.

Define $\boldsymbol{x}_t$ as the value of a state of interest at time t. This may, for example, describe a feature to be tracked, the state of a process being monitored, or the location

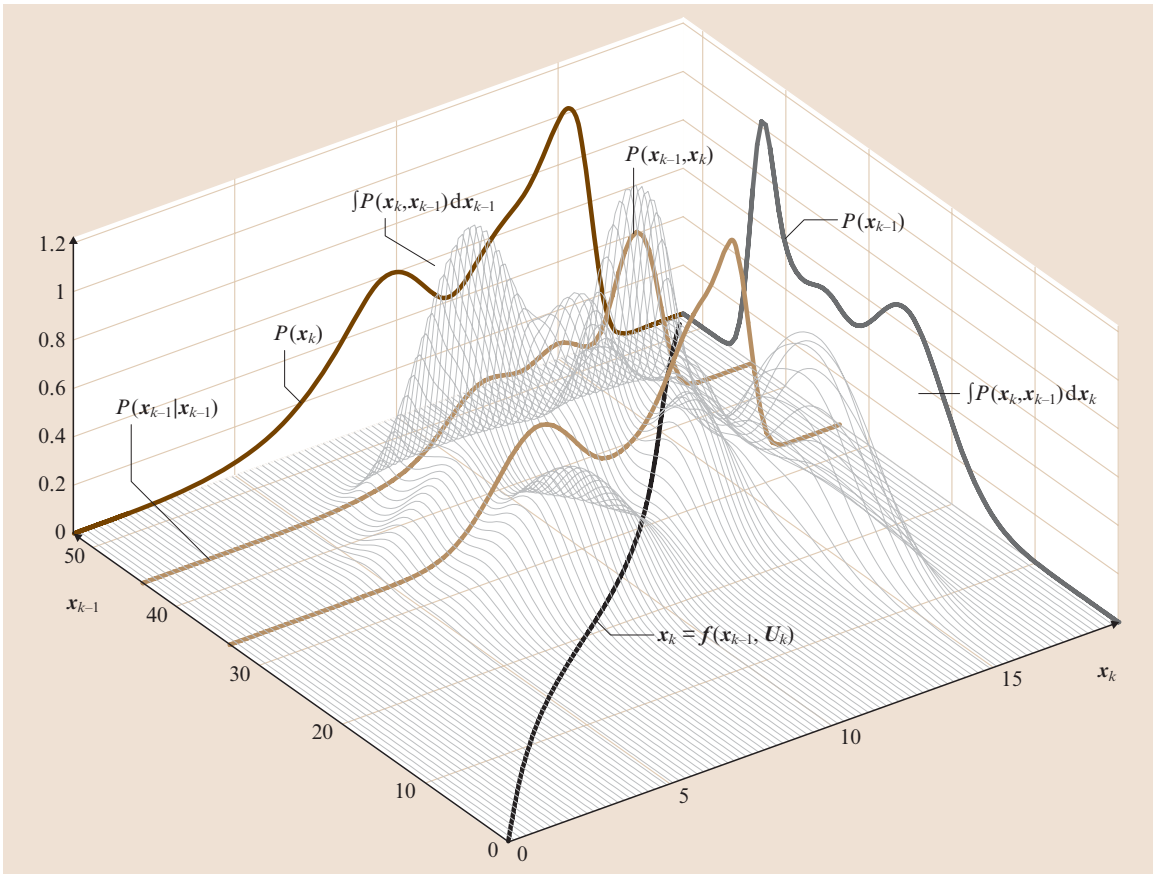

Fig. 25.1 Time update step for the full Bayes filter. At a time $k-1$, knowledge of the state $\boldsymbol{x}_{k-1}$ is summarised in a probability distribution $P(\boldsymbol{x}_{k-1})$. A vehicle model, in the form of a conditional probability density $P(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1})$, then describes the stochastic transition of the vehicle from a state $\boldsymbol{x}_{k-1}$ at a time $k-1$ to a state $\boldsymbol{x}_k$ at a time k. Functionally, this state transition may be related to an underlying kinematic state model in the form $\boldsymbol{x}_k = \boldsymbol{f}(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k)$. The figure shows two typical conditional probability distributions $P(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1})$ on the state $\boldsymbol{x}_k$ given fixed values of $\boldsymbol{x}_{k-1}$. The product of this conditional distribution with the marginal distribution $P(\boldsymbol{x}_{k-1})$, describing the prior likelihood of values of $\boldsymbol{x}_k$, gives the the joint distribution $P(\boldsymbol{x}_k, \boldsymbol{x}_{k-1})$ shown as the surface in the figure. The total marginal density $P(\boldsymbol{x}_k)$ describes knowledge of $\boldsymbol{x}_k$ after state transition has occurred. The marginal density $P(\boldsymbol{x}_k)$ is obtained by integrating (projecting) the joint distribution $P(\boldsymbol{x}_k, \boldsymbol{x}_{k-1})$ over all $\boldsymbol{x}_{k-1}$. Equivalently, using the total probability theorem, the marginal density can be obtained by integrating (summing) all conditional densities $P(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1})$ weighted by the prior probability $P(\boldsymbol{x}_{k-1})$ of each $\boldsymbol{x}_{k-1}$. The process can equally be run in reverse (a retroverse motion model) to obtain $P(\boldsymbol{x}_{k-1})$ from $P(\boldsymbol{x}_k)$ given a model $P(\boldsymbol{x}_{k-1} \mid \boldsymbol{x}_k)$

of a platform for which navigation data is required. For convenience, and without loss of generality, time is defined at discrete (asynchronous) times $t_k \triangleq k$. At a time instant k, the following quantities are defined:

$\boldsymbol{x}_k$: The state vector to be estimated at time k,
$\boldsymbol{u}_k$: A control vector, assumed known, and applied at time $k-1$ to drive the state from $\boldsymbol{x}_{k-1}$ to $\boldsymbol{x}_k$ at time k,
z_k: An observation taken of the state $\boldsymbol{x}_k$ at time k.

In addition, the following sets are also defined.

- The history of states: $\boldsymbol{X}^k = \{\boldsymbol{x}_0, \boldsymbol{x}_1, \cdots, \boldsymbol{x}_k\} = \{\boldsymbol{X}^{k-1}, \boldsymbol{x}_k\}$.
- The history of control inputs: $\boldsymbol{U}^k = \{\boldsymbol{u}_1, \boldsymbol{u}_2, \cdots, \boldsymbol{u}_k\} = \{\boldsymbol{U}^{k-1}, \boldsymbol{u}_k\}$.
- The history of state observations: $\boldsymbol{Z}^k = \{z_1, z_2, \cdots, z_k\} = \{\boldsymbol{Z}^{k-1}, z_k\}$.

In probabilistic form, the general data fusion problem is to find the posterior density

$$P\left(\boldsymbol{x}_k \mid \boldsymbol{Z}^k, \boldsymbol{U}^k, \boldsymbol{x}_0\right) \tag{25.6}$$

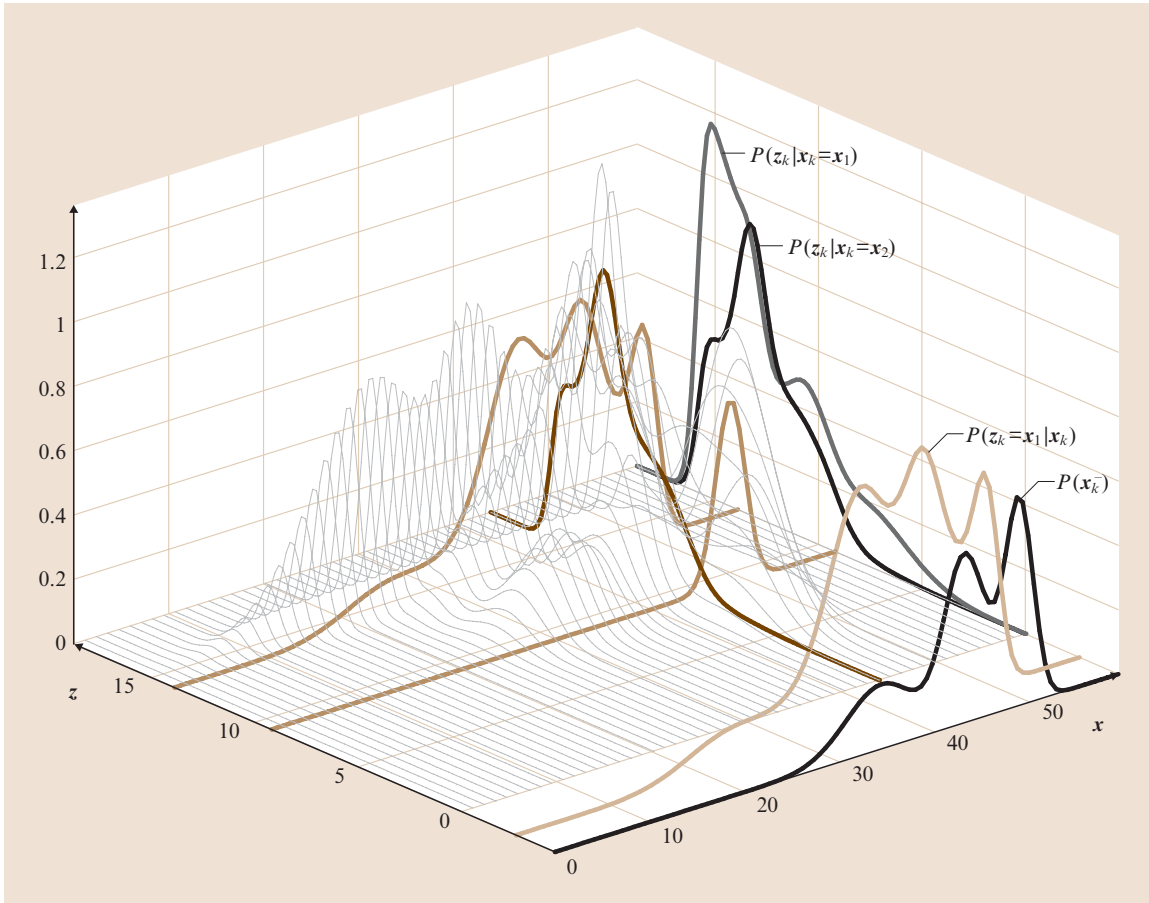

Fig. 25.2 Observation update for the full Bayes filter. Prior to observation, an observation model in the form of the conditional density $P(z_k \mid \boldsymbol{x}_k)$ is established. For a fixed value of $\boldsymbol{x}_k$, equal to $\boldsymbol{x}_1$ or $\boldsymbol{x}_2$ for example, a density function $P(z_k \mid \boldsymbol{x}_k = \boldsymbol{x}_1)$ or $P(z_k \mid \boldsymbol{x}_k = \boldsymbol{x}_2)$ is defined describing the likelihood of making the observation z_k. Together the density $P(z_k \mid \boldsymbol{x}_k)$ is then a function of both z_k and $\boldsymbol{x}_k$. This conditional density then defines the observation model. Now, in operation, a specific observation $z_k = \boldsymbol{x}_1$ is made and the resulting distribution $P(z_k = \boldsymbol{x}_1 \mid \boldsymbol{x}_k)$ defines a density function (now termed the likelihood function) on $\boldsymbol{x}_k$. This density is then multiplied by the prior density $P(\boldsymbol{x}_k^-)$ and normalised to obtain the posterior distribution $P(\boldsymbol{x}_k \mid z_k)$ describing knowledge in the state after observation

for all times k given the recorded observations and control inputs up to and including time k together (possibly) with knowledge of the initial state $\boldsymbol{x}_0$. Bayes' rule can be used to write (25.6) in terms of a sensor model $P(\boldsymbol{z}_k \mid \boldsymbol{x}_k)$ and a predicted probability density $P(\boldsymbol{x}_k \mid \boldsymbol{Z}^{k-1}, \boldsymbol{U}^k, \boldsymbol{x}_0)$ based on observations up to time $k-1$ as

$$P\left(\boldsymbol{x}_k \mid \boldsymbol{Z}^k, \boldsymbol{U}^k, \boldsymbol{x}_0\right) = \frac{P(\boldsymbol{z}_k \mid \boldsymbol{x}_k) P\left(\boldsymbol{x}_k \mid \boldsymbol{Z}^{k-1}, \boldsymbol{U}^k, \boldsymbol{x}_0\right)}{P\left(\boldsymbol{z}_k \mid \boldsymbol{Z}^{k-1}, \boldsymbol{U}^k\right)} . \tag{25.7}$$

The denominator in (25.7) is independent of the state and following (25.4) can be set to some normalising constant C. The sensor model makes use of the conditional independence assumption from (25.3).

The total probability theorem can be used to rewrite the second term in the numerator of (25.7) in terms of the state transition model and the joint posterior from time-step $k-1$ as

$$\begin{aligned} & P\left(\boldsymbol{x}_k \mid \boldsymbol{Z}^{k-1}, \boldsymbol{U}^k, \boldsymbol{x}_0\right) \\ &= \int P\left(\boldsymbol{x}_k, \boldsymbol{x}_{k-1} \mid \boldsymbol{Z}^{k-1}, \boldsymbol{U}^k, \boldsymbol{x}_0\right) \mathrm{d}\boldsymbol{x}_{k-1} \\ &= \int P\left(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1}, \boldsymbol{Z}^{k-1}, \boldsymbol{U}^k, \boldsymbol{x}_0\right) \\ &\quad \times P\left(\boldsymbol{x}_{k-1} \mid \boldsymbol{Z}^{k-1}, \boldsymbol{U}^k, \boldsymbol{x}_0\right) \mathrm{d}\boldsymbol{x}_{k-1} \\ &= \int P(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1}, \boldsymbol{u}_k) \\ &\quad \times P\left(\boldsymbol{x}_{k-1} \mid \boldsymbol{Z}^{k-1}, \boldsymbol{U}^{k-1}, \boldsymbol{x}_0\right) \mathrm{d}\boldsymbol{x}_{k-1} , \end{aligned} \tag{25.8}$$

where the last equality implies that the future state depends only on the current state and the control exerted at this time. The state transition model is described in terms of a probability distribution in the form $P(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1}, \boldsymbol{u}_k)$. That is, the state transition may reasonably be assumed to be a Markov process in which the next state $\boldsymbol{x}_k$ depends only on the immediately proceeding state $\boldsymbol{x}_{k-1}$ and the applied control $\boldsymbol{u}_k$, and is independent of both the observations and preceding states.

Equations (25.7) and (25.8) define a recursive solution to (25.6). Equation (25.8) is the *time update* or prediction step for the full Bayes data fusion algorithm. A graphical description of this equation is shown in Fig. 25.1. Equation (25.7) is the *observation update* step for the full Bayes data fusion algorithm. A graphical description of this equation is shown in Fig. 25.2. The Kalman filter, grid-based methods and sequential Monte Carlo methods, to be described, are specific implementations of these general equations.

25.1.2 Probabilistic Grids

Probabilistic grids are conceptually the simplest way of implementing Bayesian data fusion methods. They can be applied both to problems in mapping [25.3, 4] and tracking [25.5].

In mapping applications, the environment of interest is divided into a grid of equal sized spatial cells. Each cell is indexed and labeled with a property, thus the state $\boldsymbol{x}_{ij}$ may describe a two dimensional world indexed by ij and having the property x. Interest is focused on maintaining a probability distribution on possible state values $P(\boldsymbol{x}_{ij})$ *at each grid cell*. Typically, in navigation and mapping problems, the property of interest has only two values O and E, *occupied* and *empty*, respectively, and it is then usual to assume that $P(\boldsymbol{x}_{ij} = O) = 1 - P(\boldsymbol{x}_{ij} = E)$. However, there is no particular constraint on the property encoded by the state $\boldsymbol{x}_{ij}$ which could have many values (green, red, blue, for example) and indeed be continuous (the temperature at a cell for example).

Once the state has been defined, Bayesian methods require that a sensor model or likelihood function for the sensor be established. In theory, this requires specification of a probability distribution $P(\boldsymbol{z} \mid \boldsymbol{x}_{ij} = x_{ij})$ mapping each possible grid state x_{ij} to a distribution on observations. Practically, however, this is implemented simply as another *observation* grid so that for a specific observation $\boldsymbol{z} = z$ (taken from a specific location), a grid of likelihoods on the states $\boldsymbol{x}_{ij}$ is produced in the form $P(\boldsymbol{z} = z \mid \boldsymbol{x}_{ij}) = \Lambda(\boldsymbol{x}_{ij})$. It is then trivial to apply Bayes' rule to update the property value at each grid cell as

$$P^+(\boldsymbol{x}_{ij}) = C\Lambda(\boldsymbol{x}_{ij}) P(\boldsymbol{x}_{ij}) , \quad \forall i, j , \tag{25.9}$$

where C us a normalising constant obtained by summing posterior probabilities to one *at node ij only*. Computationally, this is a simple point-wise multiplication of two grids. Some care needs to be taken that the two grids appropriately overlap and align with each other at the right scale. In some instances it is also valuable to encode the fact that spatially adjacent cells will influence each other; that is, if we knew the value of the property (occupancy, temperature for example) at ij we will have some belief also of the value of this property at adjacent nodes $i+1$, j, i, $j+1$, etc. Different sensors and the fusion of different sensor outputs is accommodated simply by building appropriate sensor models $\Lambda(\boldsymbol{x}_{ij})$.

Grids can also be used for tracking and self-tracking (localisation). The state $\boldsymbol{x}_{ij}$ in this case is the location of the entity being tracked. This is a qualitatively different definition of state from that used in mapping. The probability $P(\boldsymbol{x}_{ij})$ must now be interpreted as the probability

that the object being tracked occupies the grid cell ij. In the case of mapping, the sum of property probabilities at each grid cell is one, whereas in the case of tracking, the sum of location probabilities over *the whole grid* must sum to one. Otherwise, the procedure for updating is very similar. An observation grid is constructed which when instantiated with an observation value provides a location likelihood grid $P(z = z \mid \boldsymbol{x}_{ij}) = \Lambda(\boldsymbol{x}_{ij})$. Bayes' rule is then applied to update the location probability at each grid cell in the same form as (25.9) except that now the normalisation constant C is obtained by summing posterior probabilities over *all* ij grid cells. This can become computationally expensive, especially if the grid has three or more dimensions. One major advantage of grid-based tracking is that it is easy to incorporate quite complex prior information. For example, if it is known that the object being tracked is on a road, then the probability location values for all off-road grid cells can simply be set to zero.

Grid based fusion is appropriate to situations where the domain size and dimension are modest. In such cases, grid based methods provide straightforward and effective fusion algorithms. Grid based methods can be extended in a number of ways; to hierarchical (quadtree) grids, or to irregular (triangular, pentagonal) grids. These can help reduce computation in larger spaces. Monte Carlo and particle filtering methods (Sect. 25.1.4) may be considered as grid-based methods, where the grid cells themselves are sample of the underlying probability density for the state.

25.1.3 The Kalman Filter

The Kalman filter is a recursive linear estimator which successively calculates an estimate for a continuous valued state, that evolves over time, on the basis of periodic observations of the state. The Kalman filter employs an explicit statistical model of how the parameter of interest $\boldsymbol{x}(t)$ evolves over time and an explicit statistical model of how the observations $\boldsymbol{z}(t)$ that are made are related to this parameter. The gains employed in a Kalman filter are chosen to ensure that, with certain assumptions about the observation and process models used, the resulting estimate $\hat{\boldsymbol{x}}(t)$ minimises mean-squared error and is thus the conditional mean $\hat{\boldsymbol{x}}(t) = \mathrm{E}[\boldsymbol{x}(t) \mid \boldsymbol{Z}^t]$; an average, rather than a most likely value.

The Kalman filter has a number of features which make it ideally suited to dealing with complex multi-sensor estimation and data fusion problems. In particular, the explicit description of process and observations allows a wide variety of different sensor models to be incorporated within the basic algorithm. In addition, the consistent use of statistical measures of uncertainty makes it possible to quantitatively evaluate the role each sensor plays in overall system performance. Further, the linear recursive nature of the algorithm ensures that its application is simple and efficient. For these reasons, the Kalman filter has found wide-spread application in many different data fusion problems [25.6–9].

In robotics, the Kalman filter is most suited to problems in tracking, localisation and navigation; and less so to problems in mapping. This is because the algorithm works best with well defined state descriptions (positions, velocities, for example), and for states where observation and time-propagation models are also well understood.

Observation and Transition Models

The Kalman filter may be considered a specific instance of the recursive Bayesian filter of (25.7,25.8) for the case where the probability densities on states are Gaussian. The starting point for the Kalman Filter algorithm is to define a model for the states to be estimated in the standard state-space form:

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{F}(t)\boldsymbol{x}(t) + \boldsymbol{B}(t)\boldsymbol{u}(t) + \boldsymbol{G}(t)\boldsymbol{v}(t)\,, \tag{25.10}$$

where $\boldsymbol{x}(t)$ is the state vector of interest, $\boldsymbol{u}(t)$ is a known control input, $\boldsymbol{v}(t)$ is a random variable describing uncertainty in the evolution of the state, and where $\boldsymbol{F}(t)$, $\boldsymbol{B}(t)$, and $\boldsymbol{G}(t)$ are matrices describing the contribution of states, controls and noise to state transition [25.7]. An observation (output) model is also defined in standard state-space form:

$$\boldsymbol{z}(t) = \boldsymbol{H}(t)\boldsymbol{x}(t) + \boldsymbol{D}(t)\boldsymbol{w}(t)\,, \tag{25.11}$$

where $\boldsymbol{z}(t)$ is the observation vector, $\boldsymbol{w}(t)$ is a random variable describing uncertainty in the observation, and where $\boldsymbol{H}(t)$ and $\boldsymbol{D}(t)$ are matrices describing the contribution of state and noise to the observation.

These equations define the evolution of a continuous-time system with continuous observations being made of the state. However, the Kalman Filter is almost always implemented in discrete-time $t_k = k$. It is straightforward [25.8] to obtain a discrete-time version of (25.10) and (25.11) in the form

$$\boldsymbol{x}(k) = \boldsymbol{F}(k)\boldsymbol{x}(k-1) + \boldsymbol{B}(k)\boldsymbol{u}(k) + \boldsymbol{G}(k)\boldsymbol{v}(k)\,, \tag{25.12}$$

$$\boldsymbol{z}(k) = \boldsymbol{H}(k)\boldsymbol{x}(k) + \boldsymbol{D}(k)\boldsymbol{w}(k)\,. \tag{25.13}$$

A basic assumption in the derivation of the Kalman filter is that the random sequences $\boldsymbol{v}(k)$ and $\boldsymbol{w}(k)$ describing process and observation noise are all Gaussian,

temporally uncorrelated and zero-mean

$$\mathrm{E}[\boldsymbol{v}(k)] = \mathrm{E}[\boldsymbol{w}(k)] = \boldsymbol{0}\,, \quad \forall k\,, \tag{25.14}$$

with known covariance

$$\mathrm{E}[\boldsymbol{v}(i)\boldsymbol{v}^{\mathrm{T}}(j)] = \delta_{ij}\boldsymbol{Q}(i)\,, \quad \mathrm{E}[\boldsymbol{w}(i)\boldsymbol{w}^{\mathrm{T}}(j)] = \delta_{ij}\boldsymbol{R}(i)\,. \tag{25.15}$$

It is also generally assumed that the process and observation noises are also uncorrelated

$$\mathrm{E}[\boldsymbol{v}(i)\boldsymbol{w}^{\mathrm{T}}(j)] = \boldsymbol{0}\,, \quad \forall i, j\,. \tag{25.16}$$

These are equivalent to a Markov property requiring observations and successive states to be conditionally independent. If the sequences $\boldsymbol{v}(k)$ and $\boldsymbol{w}(k)$ are temporally correlated, a shaping filter can be used to whiten the observations, again making the assumptions required for the Kalman filter valid [25.8]. If the process and observation noise sequences are correlated, then this correlation can also be accounted for in the Kalman filter algorithm [25.10]. If the sequence is not Gaussian, but is symmetric with finite moments, then the Kalman filter will still produce good estimates. If however, the sequence has a distribution which is skewed or otherwise pathological, results produced by the Kalman filter will be misleading and there will be a good case for using a more sophisticated Bayesian filter [25.5].

Filtering Algorithm

The Kalman filter algorithm produces estimates that minimise mean-squared estimation error conditioned on a given observation sequence and so is the conditional mean

$$\hat{\boldsymbol{x}}(i \mid j) \triangleq \mathrm{E}[\boldsymbol{x}(i) \mid \boldsymbol{z}(1), \cdots, \boldsymbol{z}(j)] \triangleq \mathrm{E}[\boldsymbol{x}(i) \mid \boldsymbol{Z}^j]\,. \tag{25.17}$$

The estimate variance is defined as the mean-squared error in this estimate

$$\boldsymbol{P}(i \mid j) \triangleq \mathrm{E}\{[\boldsymbol{x}(i) - \hat{\boldsymbol{x}}(i \mid j)][\boldsymbol{x}(i) - \hat{\boldsymbol{x}}(i \mid j)]^T \mid \boldsymbol{Z}^j\}\,. \tag{25.18}$$

The estimate of the state at a time k given all information up to time k is written as $\hat{\boldsymbol{x}}(k \mid k)$. The estimate of the state at a time k given only information up to time $k-1$ is called a one-step-ahead prediction (or just a prediction) and is written as $\hat{\boldsymbol{x}}(k \mid k-1)$.

The Kalman filter algorithm is now stated without proof. Detailed derivations can be found in many books on the subject, [25.7,8] for example. The state is assumed to evolve in time according to (25.12). Observations of this state are made at regular time intervals according to (25.13). The noise processes entering the system are assumed to obey (25.14), (25.15) and (25.16). It is also assumed that an estimate $\hat{\boldsymbol{x}}(k-1 \mid k-1)$ of the state $\boldsymbol{x}(k-1)$ at time $k-1$ based on all observations made up to and including time $k-1$ is available, and that this estimate is equal to the conditional mean of the true state $\boldsymbol{x}(k-1)$ conditioned on these observations. The conditional variance $\boldsymbol{P}(k-1 \mid k-1)$ in this estimate is also assumed known. The Kalman filter then proceeds recursively in two stages (Fig. 25.3).

Prediction. A prediction $\hat{\boldsymbol{x}}(k \mid \text{k-1})$ of the state at time k and its covariance $\boldsymbol{P}(k \mid k-1)$ is computed according to

$$\hat{\boldsymbol{x}}(k \mid k-1) = \boldsymbol{F}(k)\hat{\boldsymbol{x}}(k-1 \mid k-1) + \boldsymbol{B}(k)\boldsymbol{u}(k)\,, \tag{25.19}$$

$$\boldsymbol{P}(k \mid k-1) = \boldsymbol{F}(k)\boldsymbol{P}(k-1 \mid k-1)\boldsymbol{F}^{\mathrm{T}}(k) + \boldsymbol{G}(k)\boldsymbol{Q}(k)\boldsymbol{G}^{\mathrm{T}}(k)\,. \tag{25.20}$$

Update. At time k an observation $\boldsymbol{z}(k)$ is made and the updated estimate $\hat{\boldsymbol{x}}(k \mid k)$ of the state $\boldsymbol{x}(k)$, together with the updated estimate covariance $\boldsymbol{P}(k \mid k)$ is computed from the state prediction and observation according to

$$\hat{\boldsymbol{x}}(k \mid k) = \hat{\boldsymbol{x}}(k \mid k-1) + \boldsymbol{W}(k)[\boldsymbol{z}(k) - \boldsymbol{H}(k)\hat{\boldsymbol{x}}(k \mid k-1)]\,, \tag{25.21}$$

$$\boldsymbol{P}(k \mid k) = \boldsymbol{P}(k \mid k-1) - \boldsymbol{W}(k)\boldsymbol{S}(k)\boldsymbol{W}^{\mathrm{T}}(k)\,, \tag{25.22}$$

where the gain matrix $\boldsymbol{W}(k)$ is given by

$$\boldsymbol{W}(k) = \boldsymbol{P}(k \mid k-1)\boldsymbol{H}(k)\boldsymbol{S}^{-1}(k)\,, \tag{25.23}$$

where

$$\boldsymbol{S}(k) = \boldsymbol{R}(k) + \boldsymbol{H}(k)\boldsymbol{P}(k \mid k-1)\boldsymbol{H}(k) \tag{25.24}$$

is the innovation covariance. The difference between the observation $\boldsymbol{z}(k)$ and the predicted observation $\boldsymbol{H}(k)\hat{\boldsymbol{x}}(k \mid k-1)$ is termed the *innovation* or residual $\boldsymbol{v}(k)$:

$$\boldsymbol{v}(k) = \boldsymbol{z}(k) - \boldsymbol{H}(k)\hat{\boldsymbol{x}}(k \mid k-1)\,. \tag{25.25}$$

The innovation is an important measure of the deviation between the filter estimates and the observation sequence. Indeed, because the true states are not usually available for comparison with the estimated states, the innovation is often the only measure of how well the estimator is performing. The innovation is particularly important in data association.

The Extended Kalman Filter

The extended Kalman filter (EKF) is a form of the Kalman filter that can be employed when the state model

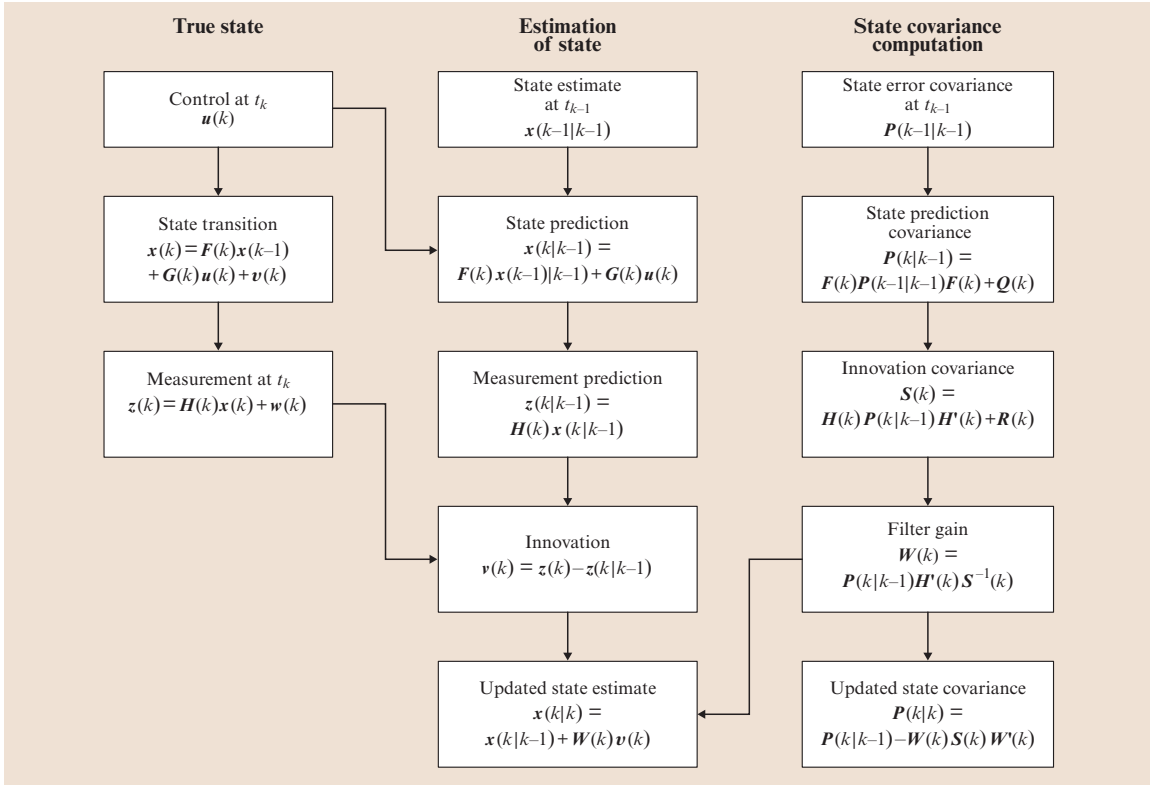

Fig. 25.3 Block diagram of the Kalman filter cycle (after *Bar-Shalom* and *Fortmann* 1988 [25.7])

and/or the observation model are nonlinear. The EKF is briefly described in this section.

The state models considered by the EKF are described in state-space notation by a first order nonlinear vector differential equation or state model of the form

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}[\boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{v}(t), t] \,, \tag{25.26}$$

where $\boldsymbol{f}[\cdot, \cdot, \cdot, \cdot]$ is now a general nonlinear mapping of state and control input to state transition. The observation models considered by the EKF are described in state-space notation by a nonlinear vector function in the form

$$\boldsymbol{z}(t) = \boldsymbol{h}[\boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{w}(t), t] \,, \tag{25.27}$$

where $\boldsymbol{h}[\cdot, \cdot, \cdot, \cdot]$ is now a general nonlinear mapping of state and control input to observations.

The EKF, like the Kalman filter, is almost always implemented in discrete-time. By integration and with appropriate identification of discrete time states and observations, the state model is written as

$$\boldsymbol{x}(k) = \boldsymbol{f}[\boldsymbol{x}(k-1), \boldsymbol{u}(k), \boldsymbol{v}(k), k] \,, \tag{25.28}$$

and the observation model as

$$\boldsymbol{z}(k) = \boldsymbol{h}[\boldsymbol{x}(k), \boldsymbol{w}(k)] \,. \tag{25.29}$$

Like the Kalman filter, it is assumed that the noises $\boldsymbol{v}(k)$ and $\boldsymbol{w}(k)$ are all Gaussian, temporally uncorrelated and zero-mean with known variance as defined in (25.14–25.16). The EKF aims to minimise mean-squared error and therefore compute an approximation to the conditional mean. It is assumed therefore that an estimate of the state at time $k-1$ is available which is approximately equal to the conditional mean, $\hat{\boldsymbol{x}}(k-1 \mid k-1) \approx \mathrm{E}[\boldsymbol{x}(k-1) \mid Z^{k-1}]$. The EKF algorithm will now be stated without proof. Detailed derivations may be found in any number of books on the subject. The principle stages in the derivation of the EKF follow directly from those of the linear Kalman filter with the additional step that the process and observation models are linearised as a Taylor series about the estimate and prediction, respectively. The algorithm has two stages:

Prediction. A prediction $\hat{\boldsymbol{x}}(k \mid k-1)$ of the state at time k and its covariance $\boldsymbol{P}(k \mid k-1)$ is computed according to

$$\hat{\boldsymbol{x}}(k \mid k-1) = \boldsymbol{f}[\hat{\boldsymbol{x}}(k-1 \mid k-1), \boldsymbol{u}(k)] , \quad (25.30)$$

$$\boldsymbol{P}(k \mid k-1) = \nabla \boldsymbol{f}_{\boldsymbol{x}}(k)\boldsymbol{P}(k-1 \mid k-1)\nabla^{\mathrm{T}} \boldsymbol{f}_{\boldsymbol{x}}(k) + \nabla \boldsymbol{f}_{\boldsymbol{v}}(k)\boldsymbol{Q}(k)\nabla^{\mathrm{T}} \boldsymbol{f}_{\boldsymbol{v}}(k) . \quad (25.31)$$

Update. At time k an observation $\boldsymbol{z}(k)$ is made and the updated estimate $\hat{\boldsymbol{x}}(k \mid k)$ of the state $\boldsymbol{x}(k)$, together with the updated estimate covariance $\boldsymbol{P}(k \mid k)$ is computed from the state prediction and observation according to

$$\hat{\boldsymbol{x}}(k \mid k) = \hat{\boldsymbol{x}}(k \mid k-1) + \boldsymbol{W}(k)\{\boldsymbol{z}(k) - \boldsymbol{h}[\hat{\boldsymbol{x}}(k \mid k-1)]\} , \quad (25.32)$$

$$\boldsymbol{P}(k \mid k) = \boldsymbol{P}(k \mid k-1) - \boldsymbol{W}(k)\boldsymbol{S}(k)\boldsymbol{W}^{\mathrm{T}}, (k) , \quad (25.33)$$

where

$$\boldsymbol{W}(k) = \boldsymbol{P}(k \mid k-1)\nabla^{\mathrm{T}} \boldsymbol{h}_{\boldsymbol{x}}(k)\boldsymbol{S}^{-1}(k) \quad (25.34)$$

and

$$\boldsymbol{S}(k) = \nabla \boldsymbol{h}_{\boldsymbol{w}}(k)\boldsymbol{R}(k)\nabla^{\mathrm{T}} \boldsymbol{h}_{\boldsymbol{w}}(k) + \nabla \boldsymbol{h}_{\boldsymbol{x}}(k)\boldsymbol{P}(k \mid k-1)\nabla^{\mathrm{T}} \boldsymbol{h}_{\boldsymbol{x}}(k) \quad (25.35)$$

and where the Jacobian $\nabla \boldsymbol{f}.(k)$ is evaluated at $\boldsymbol{x}(k-1) = \hat{\boldsymbol{x}}(k-1 \mid k-1)$ and $\nabla \boldsymbol{h}.(k)$ is evaluated at and $\boldsymbol{x}(k) = \hat{\boldsymbol{x}}(k \mid k-1)$.

A comparison of (25.19–25.24) with (25.30–25.35) makes it clear that the EKF algorithm is very similar to the linear Kalman filter algorithm, with the substitutions $\boldsymbol{F}(k) \rightarrow \nabla \boldsymbol{f}_{\boldsymbol{x}}(k)$ and $\boldsymbol{H}(k) \rightarrow \nabla \boldsymbol{h}_{\boldsymbol{x}}(k)$ being made in the equations for the variance and gain propagation. Thus, the EKF is, in effect, a linear estimator for a state error which is described by a *linear* equation and which is being observed according to a *linear* equation of the form of (25.13).

The EKF works in much the same way as the linear Kalman filter with some notable caveats.

- The Jacobians $\nabla \boldsymbol{f}_{\boldsymbol{x}}(k)$ and $\nabla \boldsymbol{h}_{\boldsymbol{x}}(k)$ are typically not constant, being functions of both state and timestep. This means that unlike the linear filter, the covariances and gain matrix must be computed on-line as estimates and predictions are made available, and will not in general tend to constant values. This significantly increases the amount of computation which must be performed on-line by the algorithm.
- As the linearised model is derived by perturbing the true state and observation models around a predicted or nominal trajectory, great care must be taken to ensure that these predictions are always close enough to the true state that second order terms in the linearisation are indeed insignificant. If the nominal trajectory is too far away from the true trajectory then the true covariance will be much larger than the estimated covariance and the filter will become poorly matched. In extreme cases the filter may also become unstable.
- The EKF employs a linearised model which must be computed from an approximate knowledge of the state. Unlike the linear algorithm, this means that the filter must be accurately initialized at the start of operation to ensure that the linearised models obtained are valid. If this is not done, the estimates computed by the filter will simply be meaningless.

The Information Filter

The information filter is mathematically equivalent to a Kalman filter. However, rather than generating state estimates $\hat{\boldsymbol{x}}(i \mid j)$ and covariances $\boldsymbol{P}(i \mid j)$ it uses information state variables $\hat{\boldsymbol{y}}(i \mid j)$ and information matrices $\boldsymbol{Y}(i \mid j)$ which are related to each other through the relationships

$$\hat{\boldsymbol{y}}(i \mid j) = \boldsymbol{P}^{-1}(i \mid j)\hat{\boldsymbol{x}}(i \mid j) , \quad \boldsymbol{Y}(i \mid j) = \boldsymbol{P}^{-1}(i \mid j) . \quad (25.36)$$

The information filter has the same prediction-update structure as the Kalman filter.

Prediction. A prediction $\hat{\boldsymbol{y}}(k \mid k-1)$ of the information state at time k and its information matrix $\boldsymbol{Y}(k \mid k-1)$ is computed according to (Joseph form [25.8]):

$$\hat{\boldsymbol{y}}(k \mid k-1) = (\boldsymbol{I} - \boldsymbol{\Omega}\boldsymbol{G}^{\mathrm{T}})\boldsymbol{F}^{-T}\hat{\boldsymbol{y}}(k-1 \mid k-1) + \boldsymbol{Y}(k \mid k-1)\boldsymbol{B}\boldsymbol{u}(k) , \quad (25.37)$$

$$\boldsymbol{Y}(k \mid k-1) = \boldsymbol{M}(k) - \boldsymbol{\Omega}\boldsymbol{\Sigma}\boldsymbol{\Omega}^{\mathrm{T}} , \quad (25.38)$$

respectively, where

$$\boldsymbol{M}(k) = \boldsymbol{F}^{-T}\boldsymbol{Y}(k-1 \mid k-1)\boldsymbol{F}^{-1} ,$$

$$\boldsymbol{\Sigma} = \boldsymbol{G}^{\mathrm{T}}\boldsymbol{M}(k)\boldsymbol{G} + \boldsymbol{Q}^{-1} ,$$

and

$$\boldsymbol{\Omega} = \boldsymbol{M}(t_k)\boldsymbol{G}\,\boldsymbol{\Sigma}^{-1} .$$

It should be noted that $\boldsymbol{\Sigma}$, whose inverse is required to compute $\boldsymbol{\Omega}$, is only of dimension of the process driving noise which is normally considerably smaller than the state dimension. Further, the matrix $\boldsymbol{F}^{-1}$ is the state-transition matrix evaluated backwards in time and so must always exist.

Update. At time k an observation $\boldsymbol{z}(k)$ is made and the updated information state estimate $\hat{\boldsymbol{y}}(k \mid k)$ together with

the updated information matrix $\boldsymbol{Y}(k \mid k)$ is computed from

$$\hat{\boldsymbol{y}}(k \mid k) = \hat{\boldsymbol{y}}(k \mid k-1) + \boldsymbol{H}(k)\boldsymbol{R}^{-1}(k)\boldsymbol{z}(k)\,, \quad (25.39)$$

$$\boldsymbol{Y}(k \mid k) = \boldsymbol{Y}(k \mid k-1) + \boldsymbol{H}(k)\boldsymbol{R}^{-1}(k)\boldsymbol{H}^{\mathrm{T}}(k)\,. \quad (25.40)$$

We emphasise that (25.38) and (25.37) are mathematically identical to (25.19) and (25.20), and that (25.39) and (25.40) are mathematically identical to (25.21) and (25.22). It will be noted that there is a duality between information and state space forms [25.10]. This duality is evident from the fact that $\boldsymbol{\Omega}$ and $\boldsymbol{\Sigma}$ in the prediction stage of the information filter play an equivalent role to the gain matrix $\boldsymbol{W}$ and innovation covariance $\boldsymbol{S}$ in the update stage of the Kalman filter. Further, the simple linear update step for the information filter is mirrored in the simple linear prediction step for the Kalman filter.

The main advantage of the information filter over the Kalman filter in data fusion problems is the relative simplicity of the update stage. For a system with n sensors, the fused information state update is *exactly* the linear sum of information contributions from all sensors as

$$\hat{\boldsymbol{y}}(k \mid k) = \hat{\boldsymbol{y}}(k \mid k-1) + \sum_{i=1}^{n} \boldsymbol{H}_i(k)\boldsymbol{R}_i^{-1}(k)\boldsymbol{z}_i(k)\,,$$

$$\boldsymbol{Y}(k \mid k) = \boldsymbol{Y}(k \mid k-1) + \sum_{i=1}^{n} \boldsymbol{H}_i(k)\boldsymbol{R}_i^{-1}(k)\boldsymbol{H}_i^{\mathrm{T}}(k)\,. \quad (25.41)$$

The reason such an expression exists in this form is that the information filter is essentially a log-likelihood expression of Bayes' rule, where products of likelihoods (25.4) are turned into sums. No such simple expression for multi-sensor updates exists for the Kalman Filter. This property of the information filter has been exploited for data fusion in robotic networks [25.11, 12] and more recently in robot navigation and localisation problems [25.1]. One substantial disadvantage of the information filter is the coding of nonlinear models, especially for the prediction step.

When to Use a Kalman or Information Filter

Kalman or information filters are appropriate to data fusion problems where the entity of interest is well defined by a continuous parametric state. This would include estimation of the position, attitude and velocity of a robot or other object, or the tracking of a simple geometric feature such as a point, line or curve. Kalman and information filters are inappropriate for estimating properties such as spatial occupancy, discrete labels, or processes whose error characteristics are not easily parametrised.

25.1.4 Sequential Monte Carlo Methods

Monte Carlo (MC) filter methods describe probability distributions as a set of weighted samples of an underlying state space. MC filtering then uses these samples to simulate probabilistic inference usually through Bayes' rule. Many samples or simulations are performed. By studying the statistics of these samples as they progress through the inference process, a probabilistic picture of the process being simulated can be built up.

Representing Probability Distributions

In sequential Monte Carlo methods, probability distributions are described in terms of a set of support points (state space values) $\boldsymbol{x}^i$, $i = 1, \cdots, N$, together with a corresponding set of normalised weights w^i, $i = 1, \cdots, N$, where $\sum_i w^i = 1$. The support points and weights can be used to define a probability density function in the form

$$P(\boldsymbol{x}) \approx \sum_{i=1}^{N} w^i \delta(\boldsymbol{x} - \boldsymbol{x}^i)\,. \quad (25.42)$$

A key question is how these support points and weights are selected to obtain a faithful representation of the probability density $P(\boldsymbol{x})$. The most general way of selecting support values is to use an *importance density* $q(\boldsymbol{x})$. The support values $\boldsymbol{x}^i$ are drawn as samples from this density; where the density has high probability, more support values are chosen, and where the density has low probability, few support support vectors are selected. The weights in (25.42) are then computed from

$$w^i \propto \frac{P(\boldsymbol{x}^i)}{q(\boldsymbol{x}^i)}\,. \quad (25.43)$$

Practically, a sample $\boldsymbol{x}^i$ is drawn from the importance distribution. The sample is then instantiated in the underlying probability distribution to yield the value $P(\boldsymbol{x} = \boldsymbol{x}^i)$. The ratio of the two probability values, appropriately normalised, then becomes the weight.

There are two instructive extremes of the importance sampling method.

1. At one extreme, the importance density could be taken to be a uniform distribution and so the support values $\boldsymbol{x}^i$ are uniformly distributed on the state space in a close approximation to a grid. The probabilities $q(\boldsymbol{x}^i)$ are also therefore equal. The weights computed from (25.43) are then simply proportional to the probabilities $w^i \propto P(\boldsymbol{x} = \boldsymbol{x}^i)$. The result is a model for the distribution which looks very like the regular grid model.

2. At the other extreme, we could choose an importance density equal to the probability model $q(\boldsymbol{x}) = P(\boldsymbol{x})$. Samples of the support values $\boldsymbol{x}^i$ are now drawn from this density. Where the density is high there will be many samples, where the density is low there will be few samples. However, if we substitute $q(\boldsymbol{x}^i) = P(\boldsymbol{x}^i)$ into (25.43), it is clear that the weights all become equal $w^i = 1/N$. A set of samples with equal weights is known as a *particle* distribution.

It is, of course, possible to mix these two representations to describe a probability distribution both in terms of a set of weights and in terms of a set of support values. The complete set of samples and weights describing a probability distribution $\{\boldsymbol{x}^i, w^i\}_{i=1}^N$ is termed a *random measure*.

The Sequential Monte Carlo Method

Sequential Monte Carlo (SMC) filtering is a simulation of the recursive Bayes update equations using sample support values and weights to describe the underlying probability distributions.

The starting point is the recursive or sequential Bayes observation update given in (25.7) and (25.8). The SMC recursion begins with a posterior probability density represented by a set of support values and weights $\{\boldsymbol{x}_{k-1}^i, w_{k-1|k-1}^i\}_{i=1}^{N_{k-1}}$ in the form

$$P\left(\boldsymbol{x}_{k-1} \mid \boldsymbol{Z}^{k-1}\right) = \sum_{i=1}^{N_{k-1}} w_{k-1}^i \delta\left(\boldsymbol{x}_{k-1} - \boldsymbol{x}_{k-1}^i\right) . \quad (25.44)$$

The prediction step requires that (25.44) is substituted into (25.8) where the joint density is marginalised. Practically however, this complex step is avoided by implicitly assuming that the importance density is exactly the transition model as

$$q_k\left(\boldsymbol{x}_k^i\right) = P\left(\boldsymbol{x}_k^i \mid \boldsymbol{x}_{k-1}^i\right) . \quad (25.45)$$

This allows new support values $\boldsymbol{x}_k^i$ to be drawn on the basis of old support values $\boldsymbol{x}_{k-1}^i$ while leaving the weights unchanged $w_k^i = w_{k-1}^i$. With this, the prediction becomes

$$P\left(\boldsymbol{x}_k \mid \boldsymbol{Z}^{k-1}\right) = \sum_{i=1}^{N_k} w_{k-1}^i \delta\left(\boldsymbol{x}_k - \boldsymbol{x}_k^i\right) . \quad (25.46)$$

The SMC observation update step is relatively straightforward. An observation model $P(\boldsymbol{z}_k \mid \boldsymbol{x}_k)$ is defined. This is a function on both variables, $\boldsymbol{z}_k$ and $\boldsymbol{x}_k$, and is a probability distribution on $\boldsymbol{z}_k$ (integrates to unity). When an observation or measurement is made, $\boldsymbol{z}_k = z_k$, the observation model becomes a function of state $\boldsymbol{x}_k$ only. If samples of the state are taken $\boldsymbol{x}_k = \boldsymbol{x}_k^i$, $i = 1 \cdots, N_k$, the observation model $P(\boldsymbol{z}_k = z_k \mid \boldsymbol{x}_k = \boldsymbol{x}_k^i)$ becomes a set of scalars describing the likelihood that the sample $\boldsymbol{x}_k^i$ could have given rise to the observation z_k. Substituting these likelihoods and (25.46) into (25.7) gives:

$$P\left(\boldsymbol{x}_k \mid \boldsymbol{Z}^k\right) = C \sum_{i=1}^{N_k} w_{k-1}^i P\left(\boldsymbol{z}_k = z_k \mid \boldsymbol{x}_k = \boldsymbol{x}_k^i\right) \delta\left(\boldsymbol{x}_k - \boldsymbol{x}_k^i\right) . \quad (25.47)$$

This is normally implemented in the form of an updated set of normalised weights

$$w_k^i = \frac{w_{k-1}^i P\left(\boldsymbol{z}_k = z_k \mid \boldsymbol{x}_k = \boldsymbol{x}_k^i\right)}{\sum_{j=1}^{N_k} w_{k-1}^j P\left(\boldsymbol{z}_k = z_k \mid \boldsymbol{x}_k = \boldsymbol{x}_k^j\right)} \quad (25.48)$$

and so

$$P\left(\boldsymbol{x}_k \mid \boldsymbol{Z}^k\right) = \sum_{i=1}^{N_k} w_k^i \delta\left(\boldsymbol{x}_k - \boldsymbol{x}_k^i\right) . \quad (25.49)$$

Note that the support values in (25.49) are the same as those in (25.46), only the weights have been changed by the observation update.

The implementation of the SMC method requires the enumeration of models for both state transition $P(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1})$ and the observation $P(\boldsymbol{z}_k \mid \boldsymbol{x}_k)$. These need to be presented in a form that allows instantiation of values for $\boldsymbol{z}_k$, $\boldsymbol{x}_k$ and $\boldsymbol{x}_{k-1}$. For low dimensional state spaces, interpolation in a lookup table is a viable representation. For high dimensional state spaces, the preferred method is to provide a representation in terms of a function.

Practically, (25.46) and (25.49) are implemented as follows:

Time Update. A process model is defined in the usual state-space form as $\boldsymbol{x}_k = \boldsymbol{f}(\boldsymbol{x}_{k-1}, \boldsymbol{w}_{k-1}, k)$, where $\boldsymbol{w}_k$ is an independent noise sequence with known probability density $P(\boldsymbol{w}_k)$. The prediction step is now implemented as follows: N_k samples $\boldsymbol{w}_k^i$, $i = 1, \cdots, N_k$ are drawn from the distribution $P(\boldsymbol{w}_k)$. The N_k support values $\boldsymbol{x}_{k-1}^i$ together with the samples $\boldsymbol{w}_k^i$ are passed through the process model as

$$\boldsymbol{x}_k^i = \boldsymbol{f}\left(\boldsymbol{x}_{k-1}^i, \boldsymbol{w}_{k-1}^i, k\right) \quad (25.50)$$

yielding a new set of support vectors $\boldsymbol{x}_k^i$. The weights for these support vectors w_{k-1}^i are not changed. In effect, the process model is simply used to do N_k simulations of state propagation.

Observation Update. The observation model is also defined in the usual state-space form as $\boldsymbol{z}_k = \boldsymbol{h}(\boldsymbol{x}_k, \boldsymbol{v}_k, k)$, where $\boldsymbol{v}_k$ is an independent noise sequence with known probability density $P(\boldsymbol{v}_k)$. The observation step is now implemented as follows: A measurement $z_k = z_k$ is made. For each support value $\boldsymbol{x}_k^i$, a likelihood is computed as

$$\Lambda(\boldsymbol{x}_k^i) = P(\boldsymbol{z}_k = z_k \mid \boldsymbol{x}_k = \boldsymbol{x}_k^i) \,. \tag{25.51}$$

Practically, this requires that the observation model be in an equational form (such as a Gaussian) which allows computation of the likelihood in the error between the measured value z_k and the observations predicted by each particle $\boldsymbol{h}(\boldsymbol{x}_k^i, k)$. The updated weights after observation are just

$$w_k^i \propto w_{k-1}^i P(\boldsymbol{z}_k = z_k \mid \boldsymbol{x}_k^i) \,. \tag{25.52}$$

Resampling

After the weights are updated it is usual to resample the measure $\{\boldsymbol{x}_k^i, w_k^i\}_{i=1}^N$. This focuses the samples in on those areas that have most probability density. The decision to resample is made on the basis of the *effective number*, N_{eff} of particles in the sample, approximately estimated from $N_{\text{eff}} = \frac{1}{\sum_i (w_k^i)^2}$. The Sampling Importance Resampling (SIR) algorithm resamples at every cycle so that the weights are always equal. One of the key problems with resampling is that the sample set fixates on a few highly likely samples. This problem of fixating on a few highly likely particles during resampling is known as *sample impoverishment*. Generally, it is good to resample when N_{eff} falls to some fraction of the actual samples (say 1/2).

When to Use Monte Carlo Methods

Monte Carlo (MC) methods are well suited to problems where state transition models and observation models are highly non-linear. This is because sample-based methods can represent very general probability densities. In particular multi-modal or multiple hypothesis density functions are well handled by Monte Carlo techniques. One caveat to note however is that the models $P(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1})$ and $P(\boldsymbol{z}_k \mid \boldsymbol{x}_k)$ must be enumerable in all cases and typically must be in a simple parametric form. MC methods also span the gap between parametric and grid-based data fusion methods.

Monte Carlo methods are inappropriate for problems where the state space is of high dimension. In general the number of samples required to faithfully model a given density increases exponentially with state space dimension. The effects of dimensionality can be limited by marginalising out states that can be modeled without sampling, a procedure known as Rao–Blackwellisation.

25.1.5 Alternatives to Probability

The representation of uncertainty is so important to the problem of information fusion that a number of alternative modeling techniques have been proposed to deal with perceived limitations in probabilistic methods.

There are three main perceived limitations of probabilistic modeling techniques.

1. Complexity: the need to specify a large number of probabilities to be able to apply probabilistic reasoning methods correctly.
2. Inconsistency: the difficulties involved in specifying a consistent set of beliefs in terms of probability and using these to obtain consistent deductions about states of interest.
3. Precision of models: the need to be precise in the specification of probabilities for quantities about which little is known.
4. Uncertainty about uncertainty: the difficulty in assigning probability in the face of uncertainty, or ignorance about the source of information.

There are three main techniques put forward to address these issues; interval calculus, fuzzy logic, and the theory of evidence (Dempster-Shafer methods). We briefly discuss each of these in turn.

Interval Calculus

The representation of uncertainty using an interval to bound true parameter values has a number of potential advantages over probabilistic techniques. In particular, intervals provide a good measure of uncertainty in situations where there is a lack of probabilistic information, but in which sensor and parameter error is known to be bounded. In interval techniques, the uncertainty in a parameter x is simply described by a statement that the true value of the state x is known to be bounded from below by a, and from above by b; $x \in [a, b]$. It is important that no other additional probabilistic structure is implied, in particular the statement $x \in [a, b]$ does not necessarily imply that x is equally probable (uniformly distributed) over the interval $[a, b]$.

There are a number of simple and basic rules for the manipulation of interval errors. These are described in detail in the book by *Moore* [25.13] (whose analysis was originally aimed at understanding limited precision computer arithmetic). Briefly, with $a, b, c, d \in \mathbb{R}$, addition, subtraction, multiplication and division are defined

by the following algebraic relations

$$[a,b]+[c,d]=[a+c,b+d]\,,$$
$$[a,b]-[c,d]=[a-d,b-c]\,, \tag{25.53}$$
$$[a,b]\times[c,d]=[\min(ac,ad,bc,bd),\ \max(ac,ad,bc,bd)]\,, \tag{25.54}$$
$$\frac{[a,b]}{[c,d]}=[a,b]\times\left[\frac{1}{d},\frac{1}{c}\right]\,,\quad 0\notin[c,d]\,. \tag{25.55}$$

It can be shown that interval addition and multiplication are both associative and commutative. Interval arithmetic admits an obvious metric distance measure;

$$d([a,b],[c,d])=\max(|a-c|,|b-d|)\,. \tag{25.56}$$

Matrix arithmetic using intervals is also possible, but substantially more complex, particularly when matrix inversion is required.

Interval calculus methods are sometimes used for detection. However, they are not generally used in data fusion problems since:

1. it is difficult to get results that converge to anything of value (it is too pessimistic), and
2. it is hard to encode dependencies between variables which are at the core of many data fusion problems.

Fuzzy Logic

Fuzzy logic has found wide-spread popularity as a method for representing uncertainty particularly in applications such as supervisory control and high-level data fusion tasks. It is often claimed that fuzzy logic provides an ideal tool for inexact reasoning, particularly in rule-based systems. Certainly, fuzzy logic has had some notable success in practical application.

A great deal has been written about fuzzy sets and fuzzy logic (see for example [25.14] and the discussion in [25.15, Chap. 11]). Here we briefly describe the main definitions and operations without any attempt to consider the more advanced features of fuzzy logic methods.

Consider a universal set consisting of the elements x; $\mathcal{X}=\{x\}$. Consider a proper subset $\mathcal{A}\subseteq\mathcal{X}$ such that

$$\mathcal{A}=\{x\mid x \quad \text{has some specific property}\}\,.$$

In conventional logic systems, we can define a membership function $\mu_A(x)$ (also called the *characteristic function* which reports if a specific element $x\in\mathcal{X}$ is a member of this set:

$$\mathcal{A}\rightleftharpoons\mu_A(x)=\begin{cases}1 & \text{if } x\in A\\ 0 & \text{if } x\notin A\end{cases}\,.$$

For example $\mathcal{X}$ may be the set of all aircraft. The set $\mathcal{A}$ may be the set of all supersonic aircraft. In the fuzzy logic literature, this is known as a *crisp* set. In contrast, a fuzzy set is one in which there is a *degree of membership*, ranging between 0 and 1. A fuzzy membership function $\mu_A(x)$ then defines the degree of membership of an element $x\in\mathcal{X}$ in the set $\mathcal{A}$. For example, if $\mathcal{X}$ is again the set of all aircraft, $\mathcal{A}$ may be the set of all *fast* aircraft. Then the fuzzy membership function $\mu_A(x)$ assigns a value between 0 and 1 indicating the degree of membership of every aircraft x to this set. Formally

$$\mathcal{A}\rightleftharpoons\mu_A\mapsto[0,1]\,.$$

Composition rules for fuzzy sets follow the composition processes for normal crisp sets, for example

$$\mathcal{A}\cap\mathcal{B}\rightleftharpoons\mu_{A\cap B}(x)=\min[\mu_A(x),\mu_B(x)]\,,$$
$$\mathcal{A}\cup\mathcal{B}\rightleftharpoons\mu_{A\cup B}(x)=\max[\mu_A(x),\mu_B(x)]\,.$$

The normal properties associated with binary logic now hold; commutativity, associativity, idempotence, distributivity, De Morgan's law and absorption. The only exception is that the law of the excluded middle is no longer true

$$\mathcal{A}\cup\overline{\mathcal{A}}\neq\mathcal{X}\,,\quad \mathcal{A}\cap\overline{\mathcal{A}}\neq\emptyset\,.$$

Together these definitions and laws provide a systematic means of reasoning about inexact values.

The relationship between fuzzy set theory and probability is still hotly debated.

Evidential Reasoning

Evidential reasoning (often called the Dempster–Shafer theory of evidence after the originators of these ideas) has seen intermittent success particularly in automated reasoning applications. Evidential reasoning is qualitatively different from either probabilistic methods or fuzzy set theory in the following sense: Consider a universal set $\mathcal{X}$. In probability theory or fuzzy set theory, a belief mass may be placed on any element $x_i\in\mathcal{X}$ and indeed on any subset $\mathcal{A}\subseteq\mathcal{X}$. In evidential reasoning, belief mass can not only be placed on elements and sets, but also sets of sets. Specifically, while the domain of probabilistic methods is all possible subsets $\mathcal{X}$, the domain of evidential reasoning is the power set $2^{\mathcal{X}}$.

As an example, consider the mutually exclusive set $\mathcal{X} = \{\text{occupied, empty}\}$. In probability theory we might assign a probability to each possible event. For example, $P(\text{occupied}) = 0.3$, and thus $P(\text{empty}) = 0.7$. In evidential reasoning, we construct the set of all subsets

$$2^{\mathcal{X}} = \{\{\text{occupied, empty}\}, \{\text{occupied}\}, \{\text{empty}\}, \emptyset\} ,$$

and belief mass is assigned to all elements of this set as

$$\begin{aligned} m(\{\text{occupied, empty}\}) &= 0.5 , \\ m(\{\text{occupied}\}) &= 0.3 , \\ m(\{\text{empty}\}) &= 0.2 , \\ m(\emptyset) &= 0.0 , \end{aligned}$$

(the null set $\emptyset$ is assigned a belief mass of zero for normalisation purposes). The interpretation of this is that there is a 30% chance of occupied, a 20% chance of empty and a 50% chance of either occupied or empty. In effect, the measure placed on the set containing both occupied and empty, is a measure of ignorance or inability to distinguish between the two alternatives. See [25.16] for a more detailed example of applying the evidential method to certainty-grid navigation.

Evidential reasoning thus provides a method of capturing ignorance or an inability to distinguish between alternatives. In probability theory, this would be dealt with in a very different manner by assigning an equal or uniform probability to each alternative. Yet, stating that there is a 50% chance of occupancy is clearly *not* the same as saying that it is unknown if it will occupied or not. The use of the power set as the *frame of discernment* allows a far richer representation of beliefs. However, this comes at the cost of a substantial increase in complexity. If there are n elements in the original set $\mathcal{X}$, then there will be 2^n possible subsets on which a belief mass will be assigned. For large n, this is clearly intractable. Further, when the set is continuous, the set of all subsets is not even measurable.

Evidential reasoning methods provide a means of assigning, and combing belief masses on sets. Methods also exist for obtaining related measures called *support* and *plausibility* which, in effect, provide upper and lower probability bounds in agreement with Dempster's original formulation of this method.

Evidential reasoning can play an important role in discrete data fusion systems, particularly in areas such as attribute fusion, and situation assessment, where information may be unknown or ambiguous. It's use in lower-level data fusion problems is challenging as the assignment of belief mass to the power set scales exponentially with state cardinality.

25.2 Multisensor Fusion Architectures

The multisensor fusion methods described in the previous section provide the algorithmic means by which sensor data and their associated uncertainty models can be used to construct either implicit or explicit models of the environment. However, a multisensor fusion system must include many other functional components to manage and control the fusion process. The organisation of these is termed a *multisensor fusion architecture*.

25.2.1 Architectural Taxonomy

Multisensor systems architectures can be organized in various ways. The military community has developed a layout of functional architectures based on the joint directors of the laboratories (JDL) model for multisensor systems. This approach views multisensor fusion in terms of signal, feature, threat and situation analysis levels (so-called JDL levels). The assessment of such systems is specified in terms of tracking performance, survivability, efficiency and bandwidth. Such measures are not generally appropriate in robotics applications and so the JDL model is not discuss this further here (see [25.17,18] for details). Other classification schemes distinguish between low and high level fusion [25.19], or centralised versus decentralised processing or data versus variable [25.20].

A general architectural framework for multisensor robotic systems has been developed and described in detail by *Makarenko* [25.21], and we will base our discussion on his approach. A system architecture is defined in terms of:

Meta Architecture. A set of high-level considerations that strongly characterize the system structure. The selection and organization of the system elements may be guided by aesthetics, efficiency, or other design criteria and goals (for example, system and component comprehensibility, modularity, scalability, portability, interoperability, (de)centralization, robustness, fault tolerance).

Algorithmic Architecture. A specific set of information fusion and decision making methods. These methods address data heterogeneity, registration, calibration, consistency, information content, independence, time interval and scale, and relationships between models and uncertainty.

Conceptual Architecture. The granularity and functional roles of components (specifically, mappings from algorithmic elements to functional structures).

Logical Architecture. Detailed canonical component types (i. e., object-oriented specifications) and interfaces to formalise intercomponent services. Components may be ad hoc or regimented, and other concerns include granularity, modularity, reuse, verification, data structures, semantics, etc. Communication issues include hierarchical versus heterarchical organization, shared memory versus message passing, information-based characterizations of subcomponent interactions, pull/push mechanisms, subscribe-publish mechanisms, etc. Control involves both the control of actuation systems within the multisensor fusion system, as well as control of information requests and dissemination within the system, and any external control decisions and commands.

Execution Architecture. Defines mapping of components to execution elements. This includes internal or external methods of ensuring correctness of the code (i. e., that the environment and sensor models have been correctly transformed from mathematical or other formal descriptions into computer implementations), and also validation of the models (i. e., ensure that the formal descriptions match physical reality to the required extent).

In any closed-loop control system, sensors are used to provide the feedback information describing the current status of the system and its uncertainties. Building a sensor system for a given application is a system engineering process that includes the analysis of system requirements, a model of the environment, the determination of system behavior under different conditions, and the selection of suitable sensors [25.22]. The next step in building the sensor system is to assemble the hardware components and develop the necessary software modules for data fusion and interpretation. Finally, the system is tested, and the performance is analysed. Once the system is built, it is necessary to monitor the different components of the system for the purpose of testing, debugging, and analysis. The system also requires quantitative measures in terms of time complexity, space complexity, robustness, and efficiency.

In addition, designing and implementing real-time systems are becoming increasingly complex owing to many added features such as graphical user interfaces (GUIs), visualization capabilities, and the use of many sensors of different types. Therefore, many software engineering issues such as reusability and the use of COTS (commercial off-the-shelf) components [25.23], real time issues [25.24–26], sensor selection [25.27], reliability [25.28–30], and embedded testing [25.31] are now getting more attention from system developers.

Each sensor type has different characteristics and functional descriptions. Consequently, some approaches aim to develop general methods of modeling sensor systems in a manner that is independent of the physical sensors used. In turn, this enables the performance and robustness of multisensor systems to be studied in a general way. There have been many attempts to provide *the* general model, along with its mathematical basis and description. Some of these modeling techniques concern error analysis and fault tolerance of multisensor systems [25.32–37]. Other techniques are model based, and require *a priori* knowledge of the sensed object and its environment [25.38–40]. These help fit data to a model, but do not always provide the means to compare alternatives. Task-directed sensing is another approach to devising sensing strategies [25.41–43]. General sensor modeling work has had a considerable influence on the evolution of multisensor fusion architectures.

Another approach to modeling sensor systems is to define sensori-computational systems associated with each sensor to allow design, comparison, transformation, and reduction of any sensory system [25.44]. In this approach, the concept of an information invariant is used to define a measure of information complexity. This provides a computational theory allowing analysis, comparison and reduction of sensor systems.

In general terms, multisensor fusion architectures may be classified according to the choice along four independent design dimensions:

1. centralized – decentralized,
2. local – global interaction of components,
3. modular – monolithic, and
4. heterarchical – hierarchical.

The most prevalent combinations are:

- centralized, global interaction, and hierarchical,
- decentralized, global interaction, and heterarchical,

- decentralized, local interaction, and hierarchical,
- decentralized, local interaction, and heterarchical.

In some cases explicit modularity is also desirable. Most existing multisensor architectures fit reasonably well into one of these categories. These categories make no general commitment to the algorithmic architecture; if the algorithmic architecture is the predominant feature of a system, then it will be characterized as part of multisensor fusion theory in Sect. 25.1; otherwise, it merely differentiates methods within one of the four meta architectures.

25.2.2 Centralized, Local Interaction, and Hierarchical

Centralized, local interaction and hierarchical architectures encompass a number of system philosophies. Least representationally demanding is the *subsumption architecture* initially proposed by *Braitenberg* [25.45] and popularized by *Brooks* [25.46]. The subsumption multisensor architecture defines behaviors as the basic components, and employs a layered set of behaviors to embody one program (monolithic). Any behavior may utilize the output of other behaviors, and may also inhibit other behaviors. The hierarchy is defined by the layers, although this is not always clear-cut. The major design philosophy is to develop behaviors directly from perception-action loops without recourse to brittle, environment representations. This leads to robustness in operation, but a lack of composite behavior predictability.

A more sophisticated (representationally) behavior-based system is the *distributed field robot architecture* (DFRA) [25.47]. This is a generalization of the sensor fusion effects (SFX) architecture [25.48]. This approach exploits modularity, and aims to achieve both behavior-based and deliberative action, reconfigurability and interoperability through the use of Java, Jini and XML, fault tolerance, adaptability, longevity, consistent interfaces and dynamic components. The algorithmic architecture is based on fuzzy logic controllers. Experiments have been demonstrated on outdoor mobile robot navigation.

Other similar architectures of this type include perception action networks *Lee* [25.49, 50], while *Draper* [25.51] focuses on types of information needed to perform tasks (higher-level integration); see also [25.52].

Another approach to this type of sensor fusion is to use artificial neural networks. The advantage is that the user, at least in principle, does not need to understand how sensor modalities relate, nor model the uncertainties, nor, in fact, determine the structure of the system more than to specify the number of layers in the network and the number of nodes per layer. The neural network is presented with a set of training examples, and must determine through the weights on the neuron connections, the optimal mapping from inputs to desired outputs (e.g., classifications, control signals, etc.) [25.53, 54].

Various other methods exist; for example, *Hager* [25.42, 43] defines a task-oriented approach to sensor fusion based on Bayesian decision theory and develops an object oriented programming framework. *Joshi* and *Sanderson* [25.55] describe a

> *"methodology for addressing model selection and multisensor fusion issues using representation size (description length) to choose (1) model class and number of parameters, (2) model parameter resolution (3) subset of observed features to model, and (4) correspondence to map features to models."*

Their approach is broader than an architecture and uses a minimization criterion to synthesize a multisensor fusion system to solve specific 2D and 3D object recognition problems.

25.2.3 Decentralized, Global Interaction, and Heterarchical

The major example of of the decentralised, global interaction meta-architecture is the blackboard system. There have been many examples of blackboard systems developed for data fusion applications. For example, the SEPIA system of *Cherfaoui* and *Vachon* [25.56] uses logical sensors (see below) in the form of modular agents which post results to a blackboard. The overall architectural goals for blackboards include; efficient collaboration and dynamic configuration. Experiments are reported on an indoor robot moving from room to room.

The MESSIE system [25.57] is a scene interpretation system based on multisensor fusion; it has been applied to the interpretation of remotely sensed images. A typology of the multisensor fusion concepts is presented, and the consequences of modeling problems for objects, scene and strategy are derived. The proposed multi-specialist architecture generalized the ideas of their previous work by taking into account the knowledge of sensors, the multiple viewing notion (shot), and the uncertainty and imprecision of models and data modeled with possibility theory. In

particular, generic models of objects are represented by concepts independent of sensors (geometry, materials, and spatial context). Three kinds of specialists are present in the architecture: generic specialists (scene and conflict), semantic object specialists, and low level specialists. A blackboard structure with a centralized control is used. The interpreted scene is implemented as a matrix of pointers enabling conflicts to be detected very easily. Under the control of the scene specialist, the conflict specialist resolves conflicts using the spatial context knowledge of objects. Finally, an interpretation system with SAR/SPOT sensors is described, and an example of a session concerned with bridge, urban area and road detection is shown.

25.2.4 Decentralized, Local Interaction, and Hierarchical

One of the earliest proposals for this type of architecture is the RCS (realtime control system) [25.58]. RCS is presented as a cognitive architecture for intelligent control, but essentially uses multisensor fusion to achieve complex control. RCS focuses on task decomposition as the fundamental organizing principle. It defines a set of nodes, each comprised of a sensor processor, a world model, and a behavior generation component. Nodes communicate with other nodes, generally in a hierarchical manner, although across layer connections are allowed. The system supports a wide variety of algorithmic architectures, from reactive behavior to semantic networks. Moreover, it maintains signals, images, and maps, and allows tight coupling between iconic and symbolic representations. The architecture does not generally allow dynamic reconfiguration, but maintains the static module connectivity structure of the specification. RCS has been demonstrated in unmanned ground vehicles [25.59]. Other object-oriented approaches include [25.34, 60].

An early architectural approach which advocated strong programming semantics for multisensor systems is the logical sensor system (LSS). This approach exploits functional (or applicative) language theory to achieve that.

The most developed version of LSS is instrumented LSS [25.22]. The ILLS approach is based on LSS introduced by *Shilcrat* and *Henderson* [25.61]. The LSS methodology is designed to specify any sensor in such a way that hides its physical nature. The main goal behind LSS was to develop a coherent and efficient presentation of the information provided by many sensors of different types. This representation provides a means for recovery from sensor failure, and also facilitates reconfiguration of the sensor system when adding or replacing sensors [25.62].

ILSS is defined as an extension to LSS, and it is comprised of the following components (Fig. 25.4):

1. *ILS name*: uniquely identifies a module;
2. *Characteristic output vector (COV)*: strongly typed output structure, with one output vector and zero or more input vectors;
3. *Commands*: input commands to the module, and output commands to the other modules;
4. *Select function*: selector that detects the failure of an alternate and switches to another alternate if possible;
5. *Alternate subnets*: alternative ways of producing the COV_{out}; it is these implementations of one or more algorithms that carry the main functions of the module;
6. *Control command interpreter (CCI)*: interpreter of the commands to the module;
7. *Embedded tests*: self-testing routines that increase robustness and facilitate debugging;
8. *Monitors*: modules that check the validity of the resulting COVs; and
9. *Taps*: hooks on the output lines to view different COV values.

These components identify the system behavior and provide mechanisms for on-line monitoring and

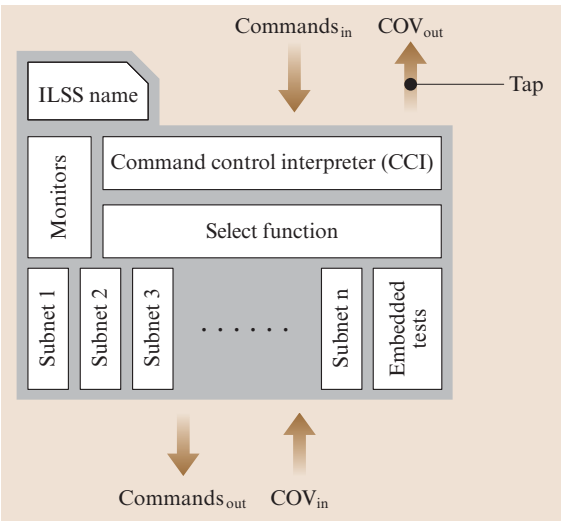

Fig. 25.4 Instrumented logical sensor module

debugging. In addition, they provide handles for measuring the run-time performance of the system. Monitors are validity check stations that filter the output and alert the user to any undesired results. Each monitor is equipped with a set of rules (or constraints) that governs the behavior of the COV under different conditions.

Embedded testing is used for on-line checking and debugging purposes. Weller proposed a sensor-processing model with the ability to detect measurement errors and to recover from these errors [25.31]. This method is based on providing each system module with verification tests to verify certain characteristics in the measured data, and to verify the internal and output data resulting from the sensor-module algorithm. The recovery strategy is based on rules that are local to the different sensor modules. ILSS uses a similar approach called *local embedded testing*, in which each module is equipped with a set of tests based on the semantic definition of that module. These tests generate input data to check different aspects of the module, then examine the output of the module using a set of constraints and rules defined by the semantics. Also, these tests can take input from other modules to check the operation of a group of modules. Examples are given of a wall-pose estimation system comprised of a Labmate platform with a camera and sonars. Many extensions have been proposed for LSS [25.63, 64].

25.2.5 Decentralized, Local Interaction, and Heterarchical

The best example of this meta architecture is the active sensor network (ASN) framework for distributed data fusion developed by *Makarenko* [25.21, 65]. The distinguishing features of the various architectures are now described.

Meta Architecture

The distinguishing features of ASN are its commitment to decentralization, modularity, and strictly local interactions (this may be physical or by type). Thus, these are communicating processes. By decentralized is meant that no component is central to operation of the system, and the communication is peer to peer. Also, there are no central facilities or services (e.g., for communication, name and service lookup or timing). These features lead to a system that is scalable, fault tolerant, and reconfigurable.

Local interactions mean that the number of communication links does not change with the network size. Moreover, the number of messages should also remain constant. This makes the system scalable and reconfigurable as well.

Modularity leads to interoperability derived from interface protocols, reconfigurability, and fault tolerance: failure may be confined to individual modules.

Algorithmic Architecture

There are three main algorithmic components: belief fusion, utility fusion and policy selection. Belief fusion is achieved by communicating all beliefs to neighboring platforms. A belief is defined as a probability distribution of the world state space.

Utility fusion is handled by separating the individual platform's partial utility into the team utility of belief quality and local utilities of action and communication. The downside is that the potential coupling between individual actions and messages is ignored because the utilities of action and communication remain local.

The communication and action policies are chosen by maximizing expected values. The selected approach is to achieve point maximization for one particular state and follows the work of *Manyika* and *Grochol-sky* [25.11, 66].

Table 25.1 Canonical components and the roles they play. Multiple X's in the same row indicate that some inter-role relationships are internalized within a component. FRAME does not participate in information fusion or decision making but is required for localization and other platform-sepcific tasks (from [25.21])

Component Type	Source	Belief Fuse/Dist	Sink	Source	Plan Fuse/Dist	Sink	Action Source	Sink
Sensor	×							
Node		×			×			
Actuator							×	
Planner			×	×		×		
UI	×		×			×		
Frame								

Conceptual Architecture

The data types of the system include

1. *beliefs*: current world beliefs,
2. *plans*: future planned world beliefs, and
3. *actions*: future planned actions.

The definition of component roles leads to a natural partition of the system.

The information fusion task is achieved through the definition of four component roles for each data type; these are: *source*, *sink*, *fuser*, and *distributor*. (Note that the data type action does not have fuser or distributor component roles.)

Connections between distributors form the backbone of the ASN framework, and the information exchanged is in the form of their local beliefs. Similar considerations are used to determine component roles for the decision making and the system configuration tasks.

Logical Architecture

A detailed architecture specification is determined from the conceptual architecture. It is comprised of six canonical component types as described in Table 25.1 [25.21].

Makarenko then describes how to combine the components and interfaces to realize the use cases of the problem domain in ASN.

Execution Architecture

The execution architecture traces the mapping of logical components to runtime elements, such as processes and shared libraries. The deployment view shows the mapping of physical components onto the nodes of the physical system. The source code view explains how the software implementing the system is organized. At the architectural level, three items are addressed: execution, deployment, and source code organization.

The experimental implementation of the ASN framework has proven to be flexible enough to accommodate a variety of system topologies, platform and sensor hardware, and environment representations. Several examples are given with a variety of sensors, processors and hardware platforms.

25.3 Applications

Multisensor fusion systems have been applied to a wide variety of problems in robotics (see references for this chapter), but the two most general areas are *dynamic system control* and *environment modeling*. Although there is some overlap in these, they may be generally characterized as

- *dynamic system control*: the problem is to use appropriate models and sensors to control the state of a dynamic system (e.g., industrial robot, mobile robot, autonomous vehicle, surgical robot, etc.). Usually such systems involve real-time feedback control loops for steering, acceleration, and behavior selection. In addition to state estimation, uncertainty models are required. Sensors may include, force/torque sensors, gyros, GPS, position encoders, cameras, range finders, etc.;
- *environment modeling*: the problem is to use appropriate sensors to construct a model of some aspect of the physical environment. This may be a particular object, e.g., a cup, a physical part, a face, etc., or a larger part of the surroundings: e.g., the interior of a building, part of a city or an extended remote or underground area. Typical sensors include cameras, radar, 3-D range finders, IR, tactile sensors and touch probes (CMMs), etc. The result is usually expressed as geometry (points, lines, surfaces), features (holes, sinks, corners, etc.), or physical properties. Part of the problem includes the determination of optimal sensor placement.

25.3.1 Dynamic System Control

The EMS-Vision system [25.67] is an outstanding exemplar of this application domain. The goal is to develop a robust and reliable perceptual system for autonomous vehicles. The development goals as stated by the EMS-Vision team are:

- COTS components,
- wide variety of objects modeled and incorporated into behaviors,
- inertial sensors for ego-state estimation,
- peripheral/foveal/saccadic vision,
- knowledge and goal driven behavior,
- state tracking for objects,
- 25 Hz real-time update rate.

The approach has been in development since the 1980s; Fig. 25.5 shows the first vehicle to drive fully au-

Fig. 25.5 First fully autonomous vehicle on German autobahn

tonomously on the German autobahn for 20 km and at speeds up to 96 km/h.

Information from inertial and vision sensors is combined to produce a road scene tree (Fig. 25.6). A 4-D generic object representation is built which includes background knowledge of the object (e.g., roads), its behavioral capabilities, object state and variances, shape and aspect parameters. Figure 25.7 shows the 4-D inertial/vision multisensor guidance system, while Fig. 25.8 shows the hardware aspects.

In summary, the EMS-Vision system is an interesting and powerful demonstration of multisensor fusion for dynamic system control.

25.3.2 ANSER II: Decentralised Data Fusion

Decentralised data fusion (DDF) methods were initially motivated by the insight that the *information* or canonical form of the conventional Kalman filter data fusion algorithm could be implemented by simply adding information contributions from observations as shown in (25.41). As these (vector and matrix) additions are commutative, the update or data fusion process can be optimally distributed amongst a network of sensors [25.11, 12, 68]. The aim of the ANSER II project was to generalise the DDF method to deal with non-Gaussian probabilities for observations and states, and to incorporate information from a diversity of sources including uninhabited air and ground vehicles, terrain data-bases and human operatives.

The mathematical structure of a DDF sensor node is shown in Fig. 25.9. The sensor is modeled directly in the form of a likelihood function. Once instantiated with an observation, the likelihood function is input to a local fusion loop which implements a local form of the Bayesian

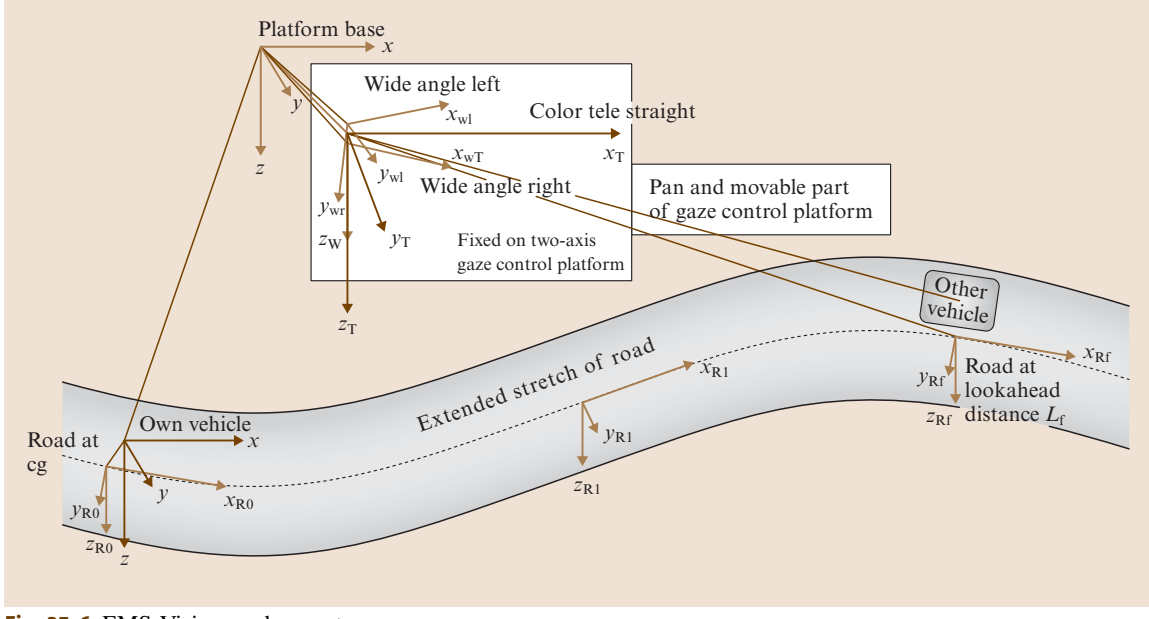

Fig. 25.6 EMS-Vision road scene tree

time and observation update of (25.7) and (25.8). Network nodes accumulate probabilistic information from observation or communication and exchange mutual information (information gain) with other nodes in the network [25.21]. This mutual information is transmitted to and assimilated by other nodes in the network in an *ad-hoc* manner. The result is that all nodes in the network obtain a single integrated posterior probability based all node observations.

The ANSER II system consists of a pair of autonomous air vehicles equipped with infra-red and visual sensors, a pair of unmanned ground vehicles equipped with visual and radar sensors, and additional information provided by geometric and hyper-spectral data bases, along with information input by human operatives [25.69]. The likelihood functions for single-sensor features are obtained through a semi-supervised machine learning method [25.70]. The resulting probabilities are modeled in the form of a mixture of Gaussians. Each platform then maintains a bank of decentralised, non-Gaussian Bayesian filters for the observed features, and transmits this information to all other platforms. The net result is that each platform maintains a complete map of all features observed by all nodes in the network. Multiple observations of the same feature, possibly by different platforms, results in an increasingly accurate estimate of the feature location for all nodes. A corresponding discrete probability measure is used for Fig. 25.10 shows a synopsis of the operation of the ANSER II system.

The ANSER II system demonstrates a number of general principles in Bayesian data fusion methods. Specifically the need to appropriately model sensors through the likelihood function, and the possibility of building very different data fusion architectures from the essential Bayesian form.

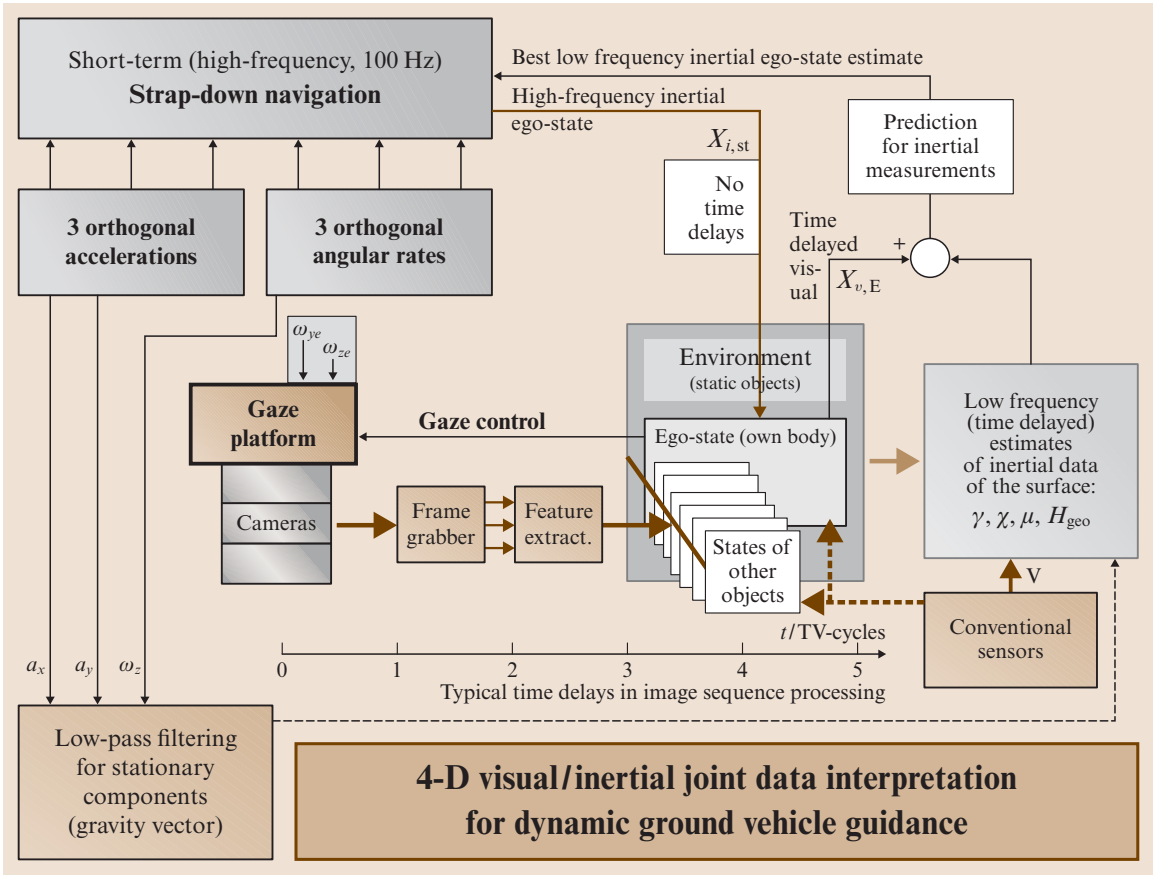

Fig. 25.7 EMS-Vision guidance system

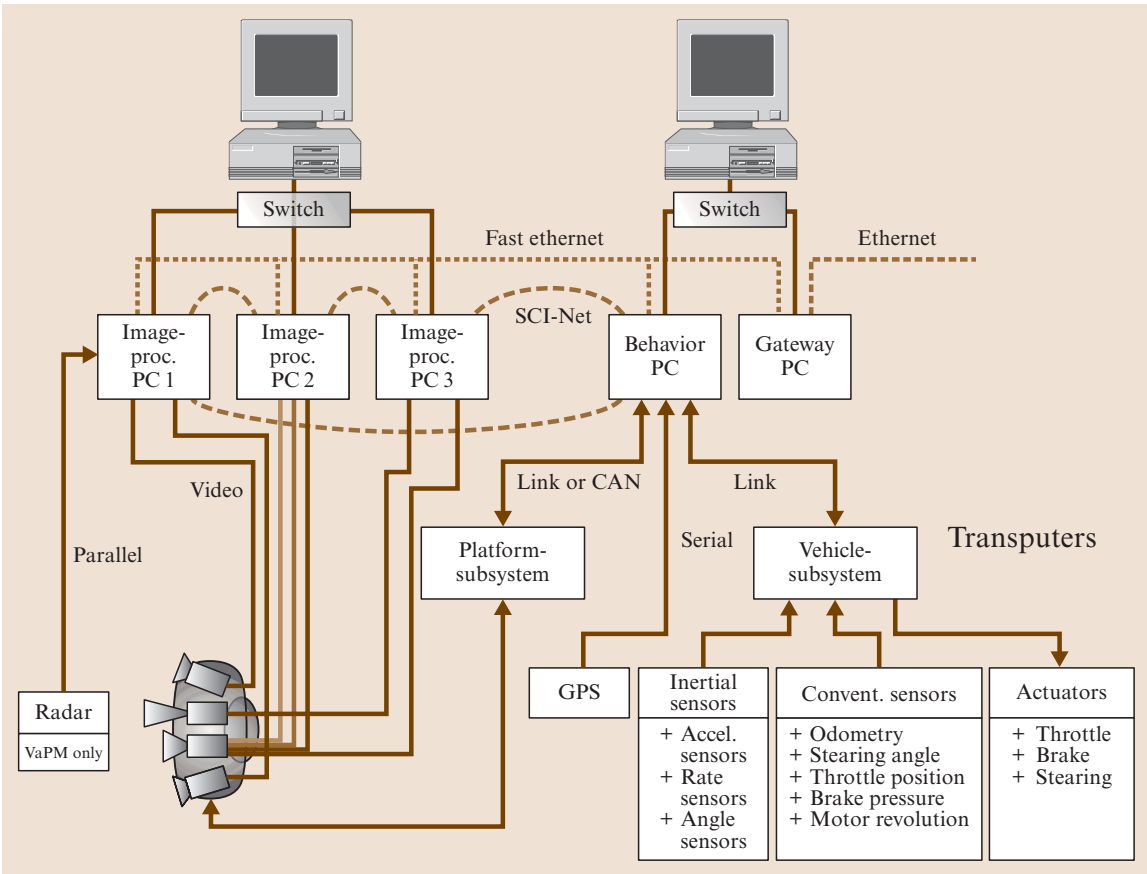

Fig. 25.8 EMS-Vision hardware layout

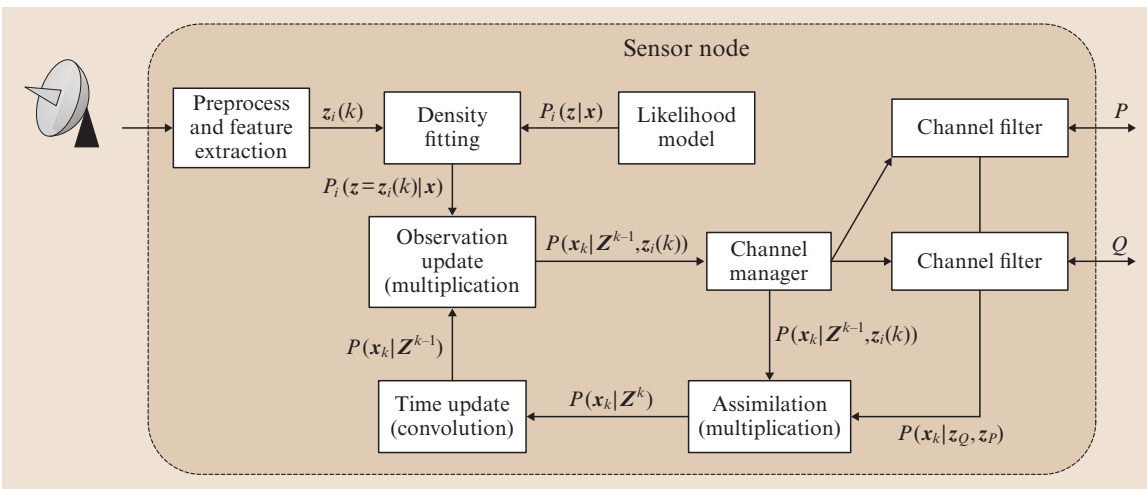

Fig. 25.9 Mathematical structure of a decentralised data fusion node

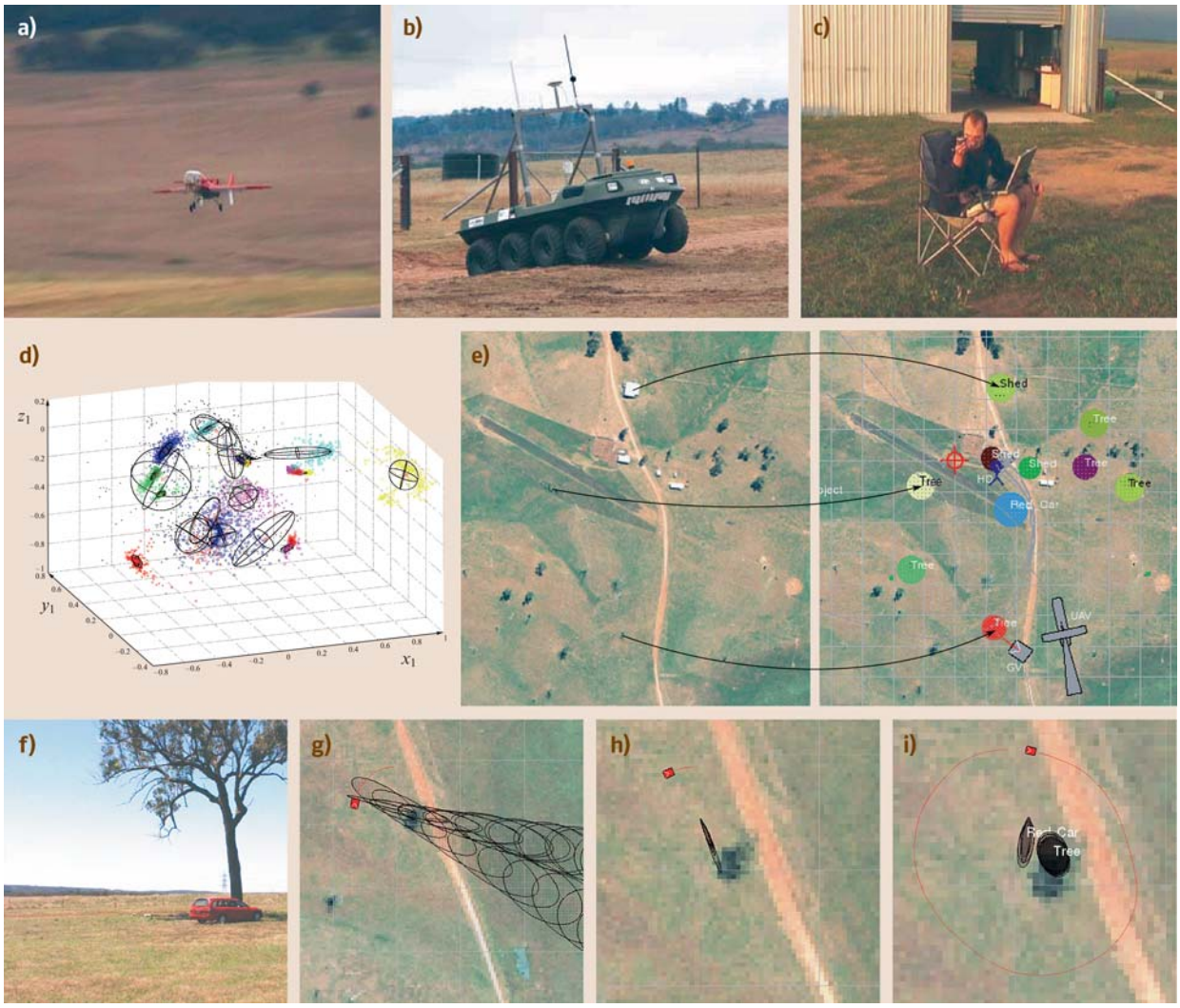

Fig. 25.10a–i A synopsis of the ANSER II autonomous network and its operation. (a–c) Main system components; **(a)** air vehicle, **(b)** ground vehicle, **(c)** human operative. (d–e) The perception process; **(d)** top three dimensions of features discovered from ground-based visual sensor data along with the derived mixture model describing these feature properties, **(e)** sector of the overall picture obtained from fusing air vehicle (UAV), ground vehicle (GV) and human operator (HO) information. Each set of ellipses corresponds to a particular feature and the labels represent the identity state with highest probability. (f–i) Sequential fusion process for two close landmarks: **(f)** a tree and a red car, **(g)** bearing-only visual observations of these landmarks are successively fused, **(h)** to determine location and identity **(i)**. Note the Gaussian mixture model for the bearing measurement likelihood

25.4 Conclusions and Further Reading

Multisensor data fusion has progressed much in the last few decades; further advances in the field will be documented in the robotics and multisensor fusion and integration conference and journal literature. Robust applications are being fielded based on the body of theory and experimental knowledge produced by

the research community. Current directions of interest include:

1. large-scale, ubiquitous sensor systems,
2. bio-based or biomimetic systems,
3. medical in situ applications, and
4. wireless sensor networks.

Representative large-scale examples include intelligent vehicle and road systems, as well as instrumented contexts such as cities. Biological principles may provide fundamentally distinct approaches to the exploitation of dense, redundant, correlated, noisy sensors, especially when considered as part of a Gibbsian framework for behavioral response to environmental stimuli. Another issue here is the development of a theoretical understanding of sensor system development, adaptivity and learning with respect to the particular context in which the system is deployed.

Further pushing the envelope of both technology and theory will permit the introduction of micro and nano sensors into the human body and allow the monitoring and locally adaptive treatment of various maladies. Finally, a more complete theoretical framework is still required which encompasses system models for wireless sensor networks. This should include models of the physical phenomena being monitored, as well as operational and network issues. Finally, numerical analysis of the algorithmic properties of data-driven systems with sensor data error sources which must be unified with the analysis of truncation, roundoff, and other errors.

A firm foundation exists upon which to build these new theories, systems, and applications. It will be a vibrant area of research for years to come!

References

25.1 S. Thrun, W. Burgard, D. Fox: *Probabilistic Robotics* (MIT Press, Cambridge 2005)

25.2 J.O. Berger: *Statistical Decision Theory and Bayesian Analysis* (Springer, Berlin, Heidelberg 1985)

25.3 A. Elfes: Sonar-based real-world mapping and navigation, IEEE Trans. Robot. Autom. **3**(3), 249–265 (1987)

25.4 A. Elfes: Integration of sonar and stereo range data using a grid-based representation, Proc. IEEE Int. Conf. Robot. Autom. (1988) 727–

25.5 L.D. Stone, C.A. Barlow, T.L. Corwin: *Bayesian Multiple Target Tracking* (Artech House, Norwood 1999)

25.6 Y. Bar-Shalom: *Multi-Target Multi-Sensor Tracking* (Artec House, Norwood 1990)

25.7 Y. Bar-Shalom, T.E. Fortmann: *Tracking and Data Association* (Academic, New York 1988)

25.8 P.S. Maybeck: *Stochastic Models, Estimaton and Control*, Vol. I (Academic, New York 1979)

25.9 W. Sorensen: Special issue on the applications of the Kalman filter, IEEE Trans. Autom. Control **28**(3), (1983)

25.10 B.D.O. Anderson, J.B. Moore: *Optimal Filtering* (Prentice Hall, Upper Saddle River 1979)

25.11 J. Manyika, H.F. Durrant-Whyte: *Data Fusion and Sensor Management: An Information-Theoretic Approach* (Prentice Hall, Upper Saddle River 1994)

25.12 S. Sukkarieh, E. Nettleton, J.H. Kim, M. Ridley, A. Goktogan, H. Durrant-Whyte: The ANSER project: Data fusion across multiple uninhabited air vehicles, Int. J. Robot. Res. **22**(7), 505–539 (2003)

25.13 R.E. Moore: *Interval Analysis* (Prentice Hall, Upper Saddle River 1966)

25.14 D. Dubois, H. Prade: *Fuzzy Sets and Systems: Theory and Applications* (Academic, New York 1980)

25.15 S. Blackman, R. Popoli: *Design and Analysis of Modern Tracking Systems* (Artec House, Norwood 1999)

25.16 D. Pagac, E.M. Nebot, H. Durrant-Whyte: An evidential approach to map-building for autonomous vehicles, IEEE Trans. Robot. Autom. **14**(4), 623–629 (1998)

25.17 D. Hall, J. Llinas: *Handbook of Multisensor Data Fusion* (CRC, Boca Raton 2001)

25.18 E.L. Waltz, J. Llinas: *Sensor Fusion* (Artec House, Norwood 1991)

25.19 M. Kam, Z. Zhu, P. Kalata: Sensor fusion for mobile robot navigation, IEEE Proc. **85**, 108–119 (1997)

25.20 H. Carvalho, W. Heinzelman, A. Murphy, C. Coelho: A general data fusion architecture, Proc. 6th Int. Conf. Inf. Fusion, Cairns (2003)

25.21 A. Makarenko: A Decentralized Architecture for Active Sensor Networks. Ph.D. Thesis (University of Sydney, Sydney 2004)

25.22 M. Dekhil, T. Henderson: Instrumented logical sensors systems, Int. J. Robot. Res. **17**(4), 402–417 (1998)

25.23 J.A. Profeta: Safety-critical systems built with COTS, IEEE Comput. **29**(11), 54–60 (1996)

25.24 H. Hu, J.M. Brady, F. Du, P. Probert: Distributed real-time control of a mobile robot, J. Intell. Autom. Soft Comput. **1**(1), 63–83 (1995)

25.25 S.A. Schneider, V. Chen, G. Pardo: ControlShell: A real-time software framework, AIAA Conf. Intell. Robot. Field Fact. Serv. Space (1994)

25.26 D. Simon, B. Espiau, E. Castillo, K. Kapellos: Computer-aided design of a generic robot con-

troller handling reactivity and real-time issues, IEEE Trans. Control Syst. Technol. **4**(1), (1993)
25.27 C. Giraud, B. Jouvencel: Sensor selection in a fusion process: a fuzzy approach, Proc. IEEE Int. Conf. Multisens. Fusion Integr., Las Vegas (1994) 599–606
25.28 R. Kapur, T.W. Williams, E.F. Miller: System testing and reliability techniques for avoiding failure, IEEE Comput. **29**(11), 28–30 (1996)
25.29 K.H. Kim, C. Subbaraman: Fault-tolerant real-time objects, Commun. ACM **40**(1), 75–82 (1997)
25.30 D.B. Stewart, P.K. Khosla: Mechanisms for detecting and handling timing errors, Commun. ACM **40**(1), 87–93 (1997)
25.31 G. Weller, F. Groen, L. Hertzberger: A sensor processing model incorporating error detection and recovery. In: *Traditional and Non-Traditional Robotic Sensors*, ed. by T. Henderson (Springer, Berlin, Heidelberg 1990) pp. 351–363
25.32 R.R. Brooks, S. Iyengar: *Averaging algorithm for multi-dimensional redundant sensor arrays: resolving sensor inconsistencies*, Tech. Rep. (Louisiana State University, 1993)
25.33 T.C. Henderson, M. Dekhil: *Visual target based wall pose estimation*, Tech. Rep. UUCS-97-010 (University of Utah, 1997)
25.34 S. Iyengar, D. Jayasimha, D. Nadig: A versatile architecture for the distributed sensor integration problem, IEEE Comput. **43**, 175–185 (1994)
25.35 D. Nadig, S. Iyengar, D. Jayasimha: New architecture for distributed sensor integration, IEEE SOUTHEASTCON Proc. (1993)
25.36 L. Prasad, S. Iyengar, R.L. Kashyap, R.N. Madan: Functional characterization of fault tolerant integration in distributed sensor networks, IEEE Trans. Syst. Man Cybern. , 1082–1087 (1991)
25.37 L. Prasad, S. Iyengar, R. Rao, R. Kashyap: Fault-tolerence sensor integration using multiresolution decomposition, Am. Phys. Soc. , 3452–3461 (1994)
25.38 H.F. Durrant-Whyte: *Integration, Coordination, and Control of Multi-Sensor Robot Systems* (Kluwer Academic, Boston 1987)
25.39 F. Groen, P. Antonissen, G. Weller: Model based robot vision, IEEE Instrum. Meas. Technol. Conf. (1993) 584–588
25.40 R. Joshi, A.C. Sanderson: Model-based multisensor data fusion: a minimal representation approach, Proc. IEEE Int. Conf. Robot. Autom. (1994)
25.41 A.J. Briggs, B.R. Donald: Automatic sensor configuration for task-directed planning, Proc. IEEE Int. Conf. Robot. Autom. (1994) 1345–1350
25.42 G. Hager: *Task Directed Sensor Fusion and Planning* (Kluwer Academic, Boston 1990)
25.43 G. Hager, M. Mintz: Computational methods for task-directed sensor data fusion and sensor planning, Int. J. Robot. Res. **10**(4), 285–313 (1991)
25.44 B. Donald: On information invariants in robotics, Artif. Intell. **72**, 217–304 (1995)
25.45 V. Braitenberg: *Vehicles: Experiments in Synthetic Psychology* (MIT Press, Cambridge 1984)
25.46 R.A. Brooks: A robust layered control system for a mobile robot, IEEE Trans. Robot. Autom. **2**(1), 14–23 (1986)
25.47 K.P. Valavanis, A.L. Nelson, L. Doitsidis, M. Long, R.R. Murphy: Validation of a distributed field robot architecture integrated with a matlab based control theoretic environment: A case study of fuzzy logic based robot navigation, CRASAR 25 (University of South Florida, 2004)
25.48 R.R. Murphy: *Introduction to AI Robotics* (MIT Press, Cambridge 2000)
25.49 S. Lee: Sensor fusion and planning with perception-action network, Proc. IEEE Conf. Multisens. Fusion Integr. Intell. Syst., Washington (1996)
25.50 S. Lee, S. Ro: Uncertainty self-management with perception net based geometric data fusion, Proc. IEEE Conf. Robot. Autom., Albuquerque (1997)
25.51 B.A. Draper, A.R. Hanson, S. Buluswar, E.M. Riseman: Information acquisition and fusion in the mobile perception laboratory, Proc. SPIE – Signal Processing, Sensor Fusion, and Target Recognition VI, Vol. 2059 (1996) 175–187
25.52 S.S. Shafer, A. Stentz, C.E. Thorpe: An architecture for sensor fusion in a mobile robot, Proc. IEEE Int. Conf. Robot. Autom. (1986) 2002–2007
25.53 S. Nagata, M. Sekiguchi, K. Asakawa: Mobile robot control by a structured hierarchical neural network, IEEE Control Syst. Mag. **10**(3), 69–76 (1990)
25.54 M. Pachter, P. Chandler: Challenges of autonomous control, IEEE Control Syst. Mag. **18**(4), 92–97 (1998)
25.55 R. Joshi, A.C. Sanderson: *Multisensor Fusion* (World Scientific, Singapore 1999)
25.56 V. Berge-Cherfaoui, B. Vachon: Dynamic configuration of mobile robot perceptual system, Proc. IEEE Conference on Multisensor Fusion and Integration for Intelligent Systems, Las Vegas (1994)
25.57 V. Clement, G. Giraudon, S. Houzelle, F. Sandakly: Interpretation of remotely sensed images in a context of mulrisensor fusion using a multi-specialist architecture, Rapport de Recherche: Programme 4 – Robotique, Image et Vision 1768, INRIA (1992)
25.58 J. Albus: RCS: A cognitive architecture for intelligent multi-agent systems, Proc. IFAC Symp. Intell. Auton. Veh., Lisbon (2004)
25.59 R. Camden, B. Bodt, S. Schipani, J. Bornstein, R. Phelps, T. Runyon, F. French: *Autonomous mobility technology assessment: Interim report*, ARL-MR 565 (Army Research Laboratory, 2003)
25.60 T. Queeney, E. Woods: A generic architecture for real-time multisensor fusion tracking algorithm development and evaluation, Proc. SPIE – Signal Processing, Sensor Fusion, and Target Recognition VII, Vol. 2355 (1994) 33–42
25.61 T. Henderson, E. Shilcrat: Logical sensor systems, J. Robot. Syst. , 169–193 (1984)

25.62 T. Henderson, C. Hansen, B. Bhanu: The specification of distributed sensing and control, J. Robot. Syst. , 387–396 (1985)
25.63 J.D. Elliott: Multisensor fusion within an encapsulated logical device architecture. Master's Thesis (University of Waterloo, Waterloo 2001)
25.64 M.D. Naish: Elsa: An intelligent multisensor integration architecture for industrial grading tasks. Master's thesis (University of Western Ontario, London 1998)
25.65 A. Makarenko, A. Brooks, S. Williams, H. Durrant-Whyte, B. Grocholsky: A decentralized architecture for active sensor networks, Proc. IEEE Int. Conf. Robot. Autom., New Orleans (2004) 1097–1102
25.66 B. Grocholsky, A. Makarenko, H. Durrant-Whyte: Information-theoretic coordinated control of multiple sensor platforms, Proc. IEEE Int. Conf. Robot. Autom., Taipei (2003) 1521–1527
25.67 R. Gregor, M. Lützeler, M. Pellkofer, K.-H. Siedersberger, E. Dickmanns: EMS-Vision: A perceptual system for autonomous vehicles, IEEE Trans. Intell. Transp. Syst. **3**(1), (2002)
25.68 B. Rao, H. Durrant-Whyte, A. Sheen: A fully decentralized multi-sensor system for tracking and surveillance, Int. J. Robot. Res. **12**(1), 20–44 (1993)
25.69 B. Upcroft: Non-gaussian state estimation in an outdoor decentralised sensor network, Proc. IEEE Conf. Decis. Control (CDC) (2006)
25.70 S. Kumar, F. Ramos, B. Upcroft, H. Durrant-Whyte: A statistical framework for natural feature representation, Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS), Edmonton (2005) 1–6

http://www.springer.com/978-3-540-23957-4

Springer Handbook of Robotics
Siciliano, B.; Khatib, O. (Eds.)
2008, LX, 1611 p. 1375 illus., 422 illus. in color. With DVD., Hardcover
ISBN: 978-3-540-23957-4