ECAI 2016
G.A. Kaminka et al. (Eds.)

doi:10.3233/978-1-61499-672-9-1397

Exact Particle Filter Modularization Improves Runtime Performance

Padraic D. Edgington[1] and **Anthony S. Maida**[2]

Abstract. Bayesian filters provide a robust and powerful technique for integrating noisy information in dynamic environments. However, the computational cost of the filtering algorithm depends on the size of the problem, and an effective solution may be constrained by execution time. This paper applies basic concepts of clustering and message passing to particle filters making them substantially faster to compute, while still maintaining the original accuracy. An example from vehicle state estimation is provided to illustrate how to implement the technique. Our results indicate that modularization can produce a speed up of over 28 times even on this small problem.

1 Introduction

Bayesian filters, such as the Kalman filter, offer a robust method for addressing problems that can be modeled with a dynamic Bayesian network where we are only interested in the current state [7, 8, 12, 23, 22, 9, 5, 6]. These types of problems tend to have sensors that provide noisy observations about the state of a dynamic environment. In general, the goal is to continuously monitor the state of a set of variables, which are indirectly related to the sensor measurements. However, the computational cost of the filtering algorithm is dependent on the size of the problem and an effective solution may be constrained by execution time. This paper introduces a new technique which reorganizes a Bayesian filter into a set of modules. Each module addresses a conditionally independent subset of the original state variables. Each module uses a single Bayesian filter to process its set of state variables and passes the results to any neighboring modules to distribute the sensor information throughout the network. The primary benefit of this modularization lies in reducing the execution time by reducing the size of individual problems. Specifically, it reduces the problem of supralinear growth in time complexity as a function of problem size. This also has the side effect of requiring less memory to represent the smaller problems.

While Bayesian filters provide effective solutions for dynamic Bayesian networks (DBNs) with continuous variables, there are several other areas of research that are closely related [14]. Three examples are dynamic Bayesian networks with discrete variables, hidden Markov models (HMMs) and Bayesian networks. Dynamic Bayesian networks with discrete variables generally use different techniques than if they only had continuous variables, instead emphasizing building and parsing tables of conditional probabilities. Hidden Markov models focus on a frequentist view of probability rather than a causal model with noise. Bayesian networks are generally simpler forms of dynamic Bayesian networks: they generally have discrete variables, and their environment does not change as time elapses.

Each of these areas of research are sufficiently similar mathematically that they have enjoyed fertile cross-pollination in recent years. Many popular techniques are derived from Pearl's [19] algorithms for exact computation in Bayesian networks. Loopy belief propagation (LBP) takes Pearl's belief propagation algorithm for singly connected networks and applies it to densely connected networks for both Bayesian networks and dynamic Bayesian networks [16, 15]. Other techniques, such as the Boyen-Koller (B-K) method [2, 3] have found their way from dynamic Bayesian networks to Bayesian networks and Hidden Markov models. Particle filters have been proposed under many names as solutions to many types problems and have been adopted into even more [22, 4, 5, 14, 17, 6].

The work solving these types of problems has fallen into two categories: exact inference methods and approximate inference methods. Pearl's [19] original work fell into the category of exact methods and some research has since been performed to improve upon those algorithms [11, 20, 24]. However, the majority of the research that has been done since then—the B-K method [2, 3], loopy belief propagation [16, 15], the factored frontier algorithm [15], assumed density filtering [13], thin junction tree filtering [18] and particle filtering [22, 4, 5, 14, 17, 6]—has focused on approximate inference.

Several approaches have been developed for improving the execution time of inference in Bayesian networks [19, 11, 24]. While none of those methods are directly applicable to Bayesian filters, the concepts of clustering from poly-tree methods ([11, 19] see also [1, 10, 21]) as well as using message-passing to perform local computations [19] can be applied to this problem as well. Thus, we will be creating statistically independent clusters that can interact with each other.

Many of these algorithms explore similar concepts—primarily clustering and message passing—and the present work is no different. Modularization fundamentally breaks a filtering problem into a series of independent clusters and passes information between them. Thus, modularization brings the clustering concepts popular in traditional inference algorithms and Pearl's message passing concepts to filtering. This paper focuses on replicating the results of a particle filter without adding any approximation. At the end, we will see how the modularized version of the algorithm compares to a traditional particle filter with the same parameters.

2 Modularizing Dynamic Bayesian Networks

This paper applies our modularization technique to particle filters. The modularization focuses on taking an existing dynamic Bayesian network (DBN) (e.g. Figure 1) and decomposing it into smaller mod-

[1] University of Connecticut, email: padraic@engr.uconn.edu
[2] University of Louisiana at Lafayette, email: maida@cacs.louisiana.edu

ules that are easier to solve than the original problem (e.g. Figure 2).

By *modularized*, we mean that the nodes representing model state variables that compose a Markovian dynamic Bayesian network are partitioned into subsets or clusters each forming an independent filter. Nodes in each cluster are generally correlated with each other, but are independent of nodes in other clusters. This makes it possible to assign a separate Bayesian filter to each cluster. Compared with a monolithic filter, module filters are smaller and can be targeted to the dynamics of each particular cluster. By *exact*, we mean that the modularized Bayesian network—with collective lower run time overhead across the modularized filters—preserves the accuracy of the original solution that used a monolithic filter for the entire network. In this work, we will use *Bayesian filters* to refer to the entire class of filtering algorithms, such as Kalman filters and extended Kalman filters (collectively termed *Gaussian filters*), as well as particle filters.

The primary benefit from modularization lies in the computation time. Since the execution time for a Bayesian filter is super-linearly dependent on the size of either the state variables or the information sources, reducing the number of elements that are processed at any one time will improve the overall execution time. While there is some overhead cost from reassembling the modular results, the final cost will generally be lower than the non-modularized execution time. The speedup of the resulting modular algorithm depends on the size and complexity of the original DBN. Large problems have more potential for improvement and the speedup is correlated with the number of modules produced; however, some improvement can be seen even on simple problems.

In modularizing a given DBN, we will create a set of modules, with each module designed to calculate a disjoint subset of the state variables. Each of these modules will have a separate Bayesian filter to calculate the probability distribution for the state variables contained in that cluster. The relationships between information sources and the modules will remain the same. However, since information sources will only be directly related to a single module, the computation for incorporating the belief about an information source can be limited to the related module. To ensure that the information is well distributed throughout the network, message passing is implemented to allow modules to act as information sources for each other.

As a side effect, modularization also has the benefit of simplifying much of the conceptual and mathematical structure of a Bayesian filter. Instead of having to understand or calculate the complex relationships between distantly related elements in the network, much of the work will be handled locally by the direct relationships. Thus, the message passing mechanism will alleviate the need to calculate all of the distant relationships; instead it uses the local relationships when passing information between modules. Since this information passing is cumulative, long distance relationships are effectively calculated through function composition.

3 Building a Modularized Bayesian Filter

The first step in creating a modularized Bayesian filter is to group the nodes (state variables) into conditionally independent clusters. For the purposes of this paper, we will assume that this can be done. For a small problem, it is possible to do this by hand, but for a larger problem, we will want to use a clustering algorithm. As can be seen in Figure 2, each cluster is present in every time step and they serve to isolate their component nodes from other information sources in the network.

Once the dynamic Bayesian network has been modularized, work can begin on constructing the modularized Bayesian filter to solve the problem. Constructing a filter has two major steps. First, filters will be selected to process each cluster that was generated by clustering. Once these filters have been defined, the message passing structures that allow information to be propagated can be described.

3.1 Selecting Bayesian Filters

Selecting Bayesian filters for a modular dynamic Bayesian network is quite similar to selecting a Bayesian filter for a regular dynamic Bayesian network. The primary difference is that instead of selecting a single filter to handle all of the calculations of the network, one filter will be selected for each cluster in the modular dynamic Bayesian network.

This set of filters should be selected by the same criteria that are used traditionally. In this case, instead of looking at the problem as a whole, each cluster is examined independently of the rest of the network. The only state variables that are of importance are those within the cluster currently being examined. All of the adjacent nodes in the graph can be considered as if they were static information sources, regardless of whether they consist of a traditional information source or another node in the original graph.

Thus, standard criteria such as linearity and the shape of the probability distribution are only considered within the scope of a single cluster at a time. That is, a modularized filter can be decomposed into a set of heterogeneous Bayesian filters. One caveat to this simplicity is in the case where a cluster with a simple probability distribution is adjacent to a cluster with a complex probability distribution. While it is possible to manipulate the complex probability distribution so that it can be related to the simple distribution, in some cases it may be preferable to use a more complicated Bayesian filter for the simple cluster to make the comparisons easier.

3.2 Message Passing

As stated earlier, the modularized filters communicate by message passing. Message passing is—for the most part—a straightforward process: messages will go in both directions, from a causal cluster to a resultant cluster and vice-versa. Message passing utilizes the existing Bayesian filter algorithms for most of the work. Causal messages improve the update information that is passed to a Bayesian filter. Resultant messages act as an additional observation to be incorporated. The major complication comes when information that was passed causally may be returned in a resultant message.

Often causal links in a DBN indicate how a variable changes based on the state of another variable. This is encapsulated in the notion of updating the previous state with the changes that have occurred since the last update. In this case, message passing works to provide a more accurate update command for the receiving filter. The other possibility is that the receiving filter only uses the information to set the state of the variables. In this case, the message would not be used to modify the previous state of the variables, but as the predicted value for those variables. The basic difference between a traditional Bayesian filter and a modular one here is that the state variables in the causal cluster will be updated with the available information before being used as a control action by the receiving filter.

In the case of resultant messages, it is expected that the receiving filter has already predicted the state of the variables for this cycle. As such, the message should be framed in terms of a measurement of the state variables. Since the message is being viewed as a measurement, it can be processed as such by the receiving Bayesian filter. This means running just the measurement section of the receiving

Bayesian filter to integrate the message with the existing probabilistic belief.

It is hard to overstate the simplicity of the basic message passing method, as it primarily involves passing the results of one Bayesian filter to a related Bayesian filter using the existing relations defined by the equations underlying the Bayesian network. This simplicity will be shown more explicitly in our example. However, the message passing method has one complication: there is the potential for information to be duplicated.

Since problems will require passing information in both directions along graph edges, we should expect that information which is passed as a causal message would also be returned as part of a resultant message. To solve this problem, a novel inverse Bayesian filter is used to remove the information in the causal message from the information in the resultant message, thus only passing novel information to the causal module. This can be seen in Figure 3 with precise descriptions of the messages being passed. Since Bayesian filters work in a fixed manner, it is easy to work backwards through a filter to retrieve a piece of information given the other pieces. This method will also ensure a relatively fixed cost for the operation that is independent of the number of information sources that the resultant module has incorporated into its state variables.

3.3 Non-Standard Information Representations

The simplest information sources are represented as Gaussian functions when they are used as parameters for a Bayesian filter. Since message passing is a major part of modularized Bayesian filters, it needs to be possible to pass the state of a particle filter as a message to other filters.

There are two potential problems with using anything besides a unimodal Gaussian function as an information source for other Bayesian filters. The first—and most obvious—is that existing Bayesian filters have all been designed to process Gaussian functions as their information sources. Thus, either the probability distribution must be represented as a Gaussian function or the Bayesian filter needs to be modified to handle the complex probability distribution directly. The second problem is more subtle: complex probability representations are often used to represent complex probability distributions. Attempting to shoehorn them into a filter designed to handle simple representations is likely to require some approximation. In the case that approximation is needed, it is beneficial to minimize the amount of error added, while still being mindful of the execution time of the approximation methods.

In general, when using a complex probability representation in a simpler Bayesian filter, the probability representation will need to be converted to something that the filter can process. An example of this is shown in Section 3.3.1 in the form of converting particle sets to Gaussian distributions for Kalman filters and extended Kalman filters. In contrast, if the receiving Bayesian filter is sufficiently robust, then it may be possible to modify the filter to accept the complex probability representation directly. An example of this is shown in Section 3.3.2 where the particle filter will be modified to use particle sets as updates and measurements. Conceptually, these ideas should generalize well to other complex probability representations such as mixtures of Gaussians or kernel functions.

3.3.1 *Particle Sets as Information Sources for Kalman-Filter Based Bayesian Filters*

Particle sets have traditionally been limited to representing the probability density function for the state variables in a particle filter [5, 6, 22]. Since the Kalman filter and extended Kalman filter perform operations on the Gaussian distributions directly, modifying the algorithm would be overly complicated. Instead, it is easier to convert a particle set to a Gaussian distribution.

The simplest method involves calculating the mean and covariance of the particle set and using it as the parameter for the Bayesian filter. The equations are similar whether calculating the Gaussian form for an unweighted particle set or a weighted particle set. Equations (1) and (2) show the formulas to calculate the mean and covariance for an unweighted particle set. Equations (3) and (4) show the formulas to calculate the mean and covariance for a weighted particle set. In both cases the equations evaluate all n particles in the set $\mathbf{x}$. In general, the individual particles, $\mathbf{x}_i$s, are composed of multiple state variables and thus are processed as a vector.

This method can work quite well for many problems. However, this is the simplest method for converting a particle set to a Gaussian representation. For distributions that are more complex, alternative manipulations may be required. Additionally, for the more complicated variants of the Kalman filter, converting the particle set to a mixture of Gaussians or a kernel function can be used to provide a more accurate approximation of the particle set.

$$\hat{\boldsymbol{\mu}} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i \tag{1}$$

$$\hat{\boldsymbol{\Sigma}} = \frac{1}{n-1}\sum_{i=1}^{n}\left(\mathbf{x}_i - \hat{\boldsymbol{\mu}}\right)\left(\mathbf{x}_i - \hat{\boldsymbol{\mu}}\right)^{\mathrm{T}} \tag{2}$$

$$\hat{\boldsymbol{\mu}} = \frac{1}{\left(\sum_{i=1}^{n} w_i\right)}\sum_{i=1}^{n} w_i\mathbf{x}_i \tag{3}$$

$$\hat{\boldsymbol{\Sigma}} = \frac{1}{\left(\frac{n-1}{n}\sum_{i=1}^{n} w_i\right)}\sum_{i=1}^{n} w_i\left(\mathbf{x}_i - \hat{\boldsymbol{\mu}}\right)\left(\mathbf{x}_i - \hat{\boldsymbol{\mu}}\right)^{\mathrm{T}} \tag{4}$$

3.3.2 *Particle Sets as Information Sources for Particle Filters*

Using particle sets as information sources for particle filters provides a more interesting case to consider. Since the particle filter itself is quite robust, it is actually possible to use the particle sets as information sources directly. There is some additional complexity in that the methods for using a particle set as the control action will differ from the methods for using a particle set as an observation. However, both of these cases provide an interesting look at how complex probability representations can be manipulated within a Bayesian filter.

The first information source that a particle filter uses is the control action, which represents the belief about how the state of the environment has changed since the previous time step. Traditionally, the algorithm would take samples from the provided Gaussian distribution and apply them to selected particles from the previous time step to arrive at a prediction about the current state of the environment. Thus, the goal in modifying the particle filter to use a particle set as the representation of the update command involves finding a way to sample from the particle set. Those samples can then be used to

update selected particles representing possible previous states of the environment.

A particle set represents the true probability distribution as a set of particles. If the true probability density function is well represented by the set of particles, then it is sufficient to sample from the particle set. If the particle set is unweighted, then simply selecting particles randomly with equal weight is sufficient to provide a high quality sampling function for a particle set. However, if the particle set is weighted, then a discrete pseudo-random number generator is used to select particles with probability proportional to their weights. Both of these techniques are generally easy to implement, as they are already part of handling the main particle set in a particle filter.

Unfortunately, particle sets are not well suited for grading other particles. While the weights on weighted particles are readily accessible to a particle filter, the distribution described by the density of the particle set is not readily accessible. As such, using a particle set as the information provided as an observation is similar to using a particle set with a Gaussian filter. The main difference in working with a particle filter is that the particle set does not need to be converted to a specific type of probability representation. Instead, any representation that can quickly produce an estimation of the full distribution and then be used to grade particles will be effective for this task.

However, there is one case—which is of particular interest in modularized Bayesian filters—in which particle sets can be used by a particle filter directly. In modularized Bayesian filters, messages will often be passed circularly. In these cases, the information passed as the update command needs to be removed from the particle set. Normally, this would be done using an inverse Bayesian filter; however, in this case the inverse filter only needs to ignore the information about the distribution of the particles. the weights on the individual particles can be readily used as the probability of that particle based on other observations. As such, a particle can be graded simply by applying the weight of the relevant part of the particle set. The exact weight applied can be obtained by any of a number of simple techniques, such as interpolation, or just selecting the nearest point in the particle set and using its weight directly.

4 Computation Time for Modularized Bayesian Filters

This section will examine a few simple cases for particle filters that will show how the algorithm is capable of scaling for large problems. The actual speedup obtained from applying these techniques depends on the specific problem being addressed. There are two types of particle filters: one that uses a fixed number of particles and one that varies the number of particles based on the complexity of the probability distribution. The original particle filter, which uses a fixed number of particles at each time step, has a fixed execution time based on that number of particles. Since the number of particles produced is chosen by hand, looking at this algorithm is not terribly interesting. Instead, the KLD sampling variant provides a more robust and generalizable view of the cost of using a particle filter.

In the KLD sampling particle filter, the Kullback-Leibler divergence ensures that the generated probability distribution represents the true distribution within a specified tolerance. As long as the accuracy of the information provided to the KLD sampling particle filter is constant, the number of particles generated remains reasonably constant between iterations of the algorithm. For a single particle filter, the time complexity is $\mathrm{O}(p^n)$; where p is the number of particles required to represent a single dimension of the state space adequately and n is the number of dimensions in the state space. For this example, we will assume that p is constant across dimensions, though it does not have to be. As an illustration, if the particle set was representing a unimodal distribution, the particle distribution could be seen as forming a hyper-ellipse, where the number of particles would roughly correspond to the volume of the hyper-ellipse.

Dividing the problem into a set of m equally sized modules produces a time complexity of $\mathrm{O}(mp^{n/m})$ to calculate all of the modules. The time complexity of the inverse particle filter is based on the method used to obtain a weight for each particle of interest. For this example, we will assume that a *k*-d tree is used to select the nearest particle. *k*-d trees are efficient, having a time complexity of $\mathrm{O}(n \log n)$ to create the tree and $\mathrm{O}(\log n)$ to find the nearest element in the tree, where n is the number of particles in the set. For this example, that translates into a time complexity of $\mathrm{O}(p^{n/m} \log p^{n/m})$ to construct each k-d tree and $\mathrm{O}(p^{n/m} \log p^{n/m})$ to grade all the particles in a set. Summing these together yields a full time complexity of

$$\mathrm{O}\left(mp^{n/m} + 2\,(m-1)\,p^{n/m} \log p^{n/m}\right) = \mathrm{O}\left(mp^{n/m} \log p^{n/m}\right). \quad (5)$$

For additional clarification, let us consider two test cases: in the first case, there will be two modules with state variables evenly distributed between them; in the second case, there will be n modules, each processing one state variable. In the first case, substituting $m = 2$ into (5) produces a time complexity of $\mathrm{O}(2p^{n/2} \log p^{n/2}) = \mathrm{O}(p^{n/2} \log p^{n/2})$, which will clearly provide a healthy speedup as compared to the original $\mathrm{O}(p^n)$. In the second case, $m = n$ is substituted into (5). This produces the more attractive result of $\mathrm{O}(np^{n/n} \log p^{n/n}) = \mathrm{O}(np \log p)$, which can be further reduced to $\mathrm{O}(n)$.

For a particle filter, the effects on memory usage are the same as with time complexity. Since the number of particles generated directly influences the computation time and the amount of space required to hold the result, reducing the number of particles affects both complexities the same way.

5 Vehicle State Estimation Example

Next, we will look at a complete example of building a modularized Bayesian filter. This example considers a hypothetical automobile moving in a planar environment. It respects the basic physical concepts of motion, but ignores more complex aspects such as slip and drag. The formulation of the problem provides a set of equations relating acceleration, velocity and position variables. These equations can then be used to create a dynamic Bayesian network that can be clustered. Bayesian filters will then be applied to each cluster and their interactions can be described explicitly.

The vehicle described here operates in a two-dimensional plane, so it needs an x and y variable to represent its position in the plane. Since it is a non-holonomic vehicle, its orientation also affects the motion of the vehicle, so an angular pose variable θ is also necessary. This example is modeled in terms of position, velocity and acceleration so a set of three variables are needed for each level. This produces nine state variables to estimate. The equations of motion

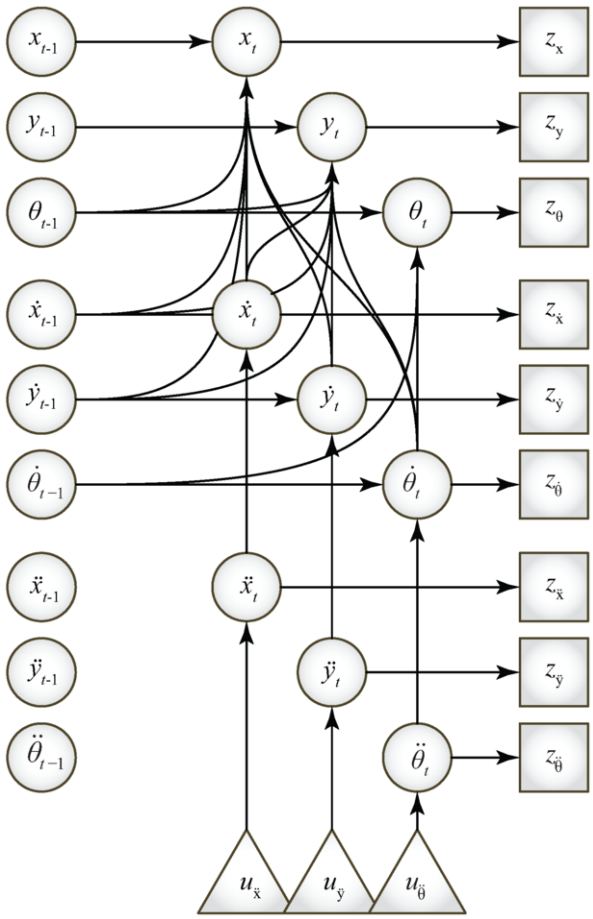

Figure 1. The initial dynamic Bayesian network for the vehicle state estimation example. Circles indicate state variables, triangles denote control values, and squares indicate observations or measurements.

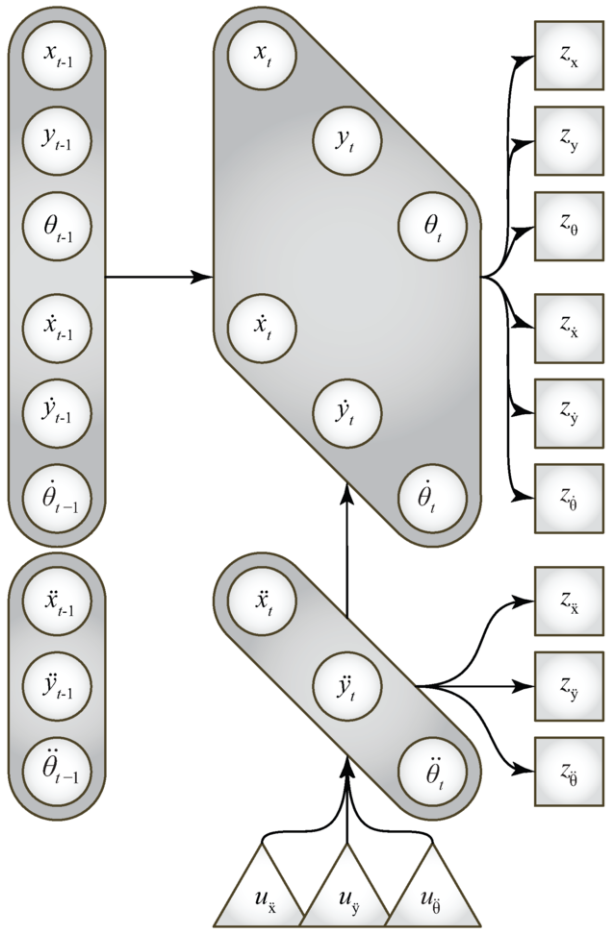

Figure 2. The clustered dynamic Bayesian network for the vehicle state estimation example. The original problem with 9 state variables reduced to subproblems of size 3 and 6.

for this vehicle are as follows:

$$\dot{x}_t = \dot{x}_{t-1} + \ddot{x}_t \Delta t \tag{6}$$

$$\dot{y}_t = \dot{y}_{t-1} + \ddot{y}_t \Delta t \tag{7}$$

$$\dot{\theta}_t = \dot{\theta}_{t-1} + \ddot{\theta}_t \Delta t \tag{8}$$

$$v_{\mathrm{m}} = \frac{1}{2}\left(\sqrt{\dot{x}_{t-1}^2 + \dot{y}_{t-1}^2} + \sqrt{\dot{x}_t^2 + \dot{y}_t^2}\right) \tag{9}$$

$$v_\theta = \frac{1}{2}\left(\dot{\theta}_{t-1} + \dot{\theta}_t\right) \tag{10}$$

$$x_t = x_{t-1} - \frac{v_{\mathrm{m}}}{v_\theta}\sin\left(\theta_{t-1}\right) + \frac{v_{\mathrm{m}}}{v_\theta}\sin\left(\theta_{t-1} + v_\theta \Delta t\right) \tag{11}$$

$$y_t = y_{t-1} + \frac{v_{\mathrm{m}}}{v_\theta}\cos\left(\theta_{t-1}\right) - \frac{v_{\mathrm{m}}}{v_\theta}\cos\left(\theta_{t-1} + v_\theta \Delta t\right) \tag{12}$$

$$\theta_t = \theta_{t-1} + v_\theta \Delta t \tag{13}$$

These equations can then be used to define a dynamic Bayesian network. With the addition of a set of singly connected control (u) and measurement (z) nodes, the graph will look like Figure 1.

This problem can be separated into two conditionally independent clusters. The first cluster is comprised of the position and velocity nodes, with the acceleration nodes in the second cluster. This clustered network is shown in Figure 2. By collapsing all of the related nodes and focusing on the information flow, we arrive at Figure 3. The graph shows the two clusters as a pair of circles. Both of the clusters have a set of related measurement variables. However, the control variables only affect the acceleration cluster directly. The results from the acceleration cluster will be fed to the position/velocity cluster. The results from the position/velocity cluster are also fed back to the acceleration cluster via an inverse Bayesian filter. Since the inverse filter will be decomposing the results of the position/velocity cluster, it requires several of the position/velocity filter's parameters as well.

Most of the relationships in Figure 3 are directly derived from Figure 2, where the clusters and their relationships to the other parts of the graph are fully defined. These relationships have been carried over into Figure 3. The addition here is in the information flow. The basic flow of causality to and from the information sources is unchanged as indicated by the thin-bodied arrows. However, in order for the information in these elements to be applied to all of the state variables in the graph, the information must be able to pass between clusters.

Since the state of the acceleration cluster causes the position and velocity variables to change, the state of the acceleration cluster is passed directly to the position/velocity cluster as its update. This allows the information in the external update (control) command and the information in the acceleration sensors to be applied to the position and velocity state variables. Since we also want the information from the position and velocity sensors to be applied to the acceleration calculations, this information also needs to be passed from the position/velocity cluster to the acceleration cluster. How-

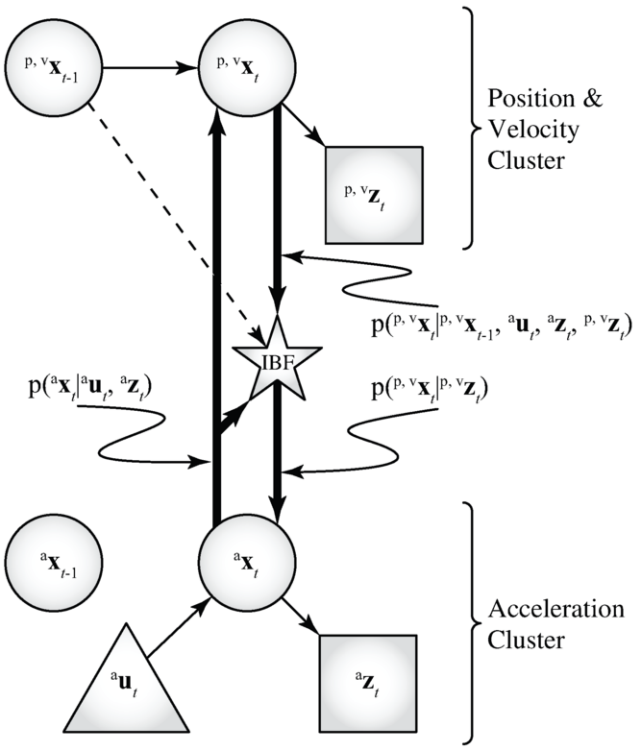

Figure 3. The clustered dynamic Bayesian network with information flow made explicit. Within cluster nodes are collapsed. The star-shaped node denotes an inverse filter.

ever, since the position and velocity calculations have already incorporated the information about the acceleration, the results of the position/velocity cluster cannot be passed back to the acceleration cluster directly. Instead, the current state of both the acceleration state variables, as well as the position and velocity state variables are passed to an inverse Bayesian filter, which can separate the acceleration information from the position and velocity information, allowing the remaining position and velocity measurements to be used to update the acceleration information.

5.1 Selecting Bayesian Filters

Having defined two clusters for the problem, the next step is to select a Bayesian filter for each individual cluster. Since the individual clusters do not need to employ the same filters, we can select a Bayesian filter that is appropriate for each cluster separately. As defined in Equations 6–8, the three acceleration nodes have a simple linear relationship with the velocity nodes. Since the control and measurement nodes are directly related for this example, the acceleration cluster is well suited to a simple Kalman filter. The position and velocity cluster on the other hand has several non-linear relationships between its nodes and other nodes in the graph. As such, it would be better suited to an extended Kalman filter (EKF) or particle filter. In this paper, we will show how this case unfolds with a particle filter.

5.2 Assembling the Modular Particle Filter Algorithm

These concepts can now be translated into a modularized Bayesian filter algorithm. The algorithm for the modular particle filter is shown

Algorithm 1 A modular particle filter algorithm for the vehicle state estimation example.

1: **procedure** MODULAR PARTICLE FILTER(...)
2: $\begin{bmatrix} {}^{\mathrm{a}}\hat{\boldsymbol{\mu}}_t & {}^{\mathrm{a}}\hat{\boldsymbol{\Sigma}}_t \end{bmatrix} \leftarrow \mathrm{KF}\left(\begin{matrix} {}^{\mathrm{a}}\boldsymbol{\mu}_{t-1}, {}^{\mathrm{a}}\boldsymbol{\Sigma}_{t-1}, \mathbf{u}_t, \boldsymbol{M}_t, {}^{\mathrm{a}}\mathbf{z}_t, \\ {}^{\mathrm{a}}\boldsymbol{Q}_t, {}^{\mathrm{a}}\boldsymbol{A}_t, {}^{\mathrm{a}}\boldsymbol{B}_t, {}^{\mathrm{a}}\boldsymbol{C}_t \end{matrix} \right)$
3: ${}^{\mathrm{p,v}}\boldsymbol{P}_t \leftarrow \text{PARTICLE FILTER}\left(\begin{matrix} {}^{\mathrm{p,v}}\boldsymbol{P}_{t-1}, \begin{bmatrix} {}^{\mathrm{a}}\hat{\boldsymbol{\mu}}_t & {}^{\mathrm{a}}\hat{\boldsymbol{\Sigma}}_t \end{bmatrix}, \\ \begin{bmatrix} {}^{\mathrm{p,v}}\mathbf{z}_t & {}^{\mathrm{p,v}}\boldsymbol{Q}_t \end{bmatrix} \end{matrix} \right)$
4: $\begin{bmatrix} \hat{\mathbf{z}}_{\mathrm{IBF}} & \hat{\boldsymbol{Q}}_{\mathrm{IBF}} \end{bmatrix} \leftarrow \text{INVERSE PARTICLE FILTER}({}^{\mathrm{p,v}}\boldsymbol{P}_t)$
5: $\begin{bmatrix} {}^{\mathrm{p,v}}\boldsymbol{\mu}_{t-1} & {}^{\mathrm{p,v}}\boldsymbol{\Sigma}_{t-1} \end{bmatrix} \leftarrow \text{PS2GAUSSIAN}({}^{\mathrm{p,v}}\boldsymbol{P}_{t-1})$
6: $\mathbf{z}_{\mathrm{IBF}} \leftarrow \text{PREDICTION}^{-1}\left({}^{\mathrm{p,v}}\boldsymbol{\mu}_{t-1}, \hat{\mathbf{z}}_{\mathrm{IBF}}\right)$
7: $\boldsymbol{Q}_{\mathrm{IBF}} \leftarrow \text{PREDICTION}^{-1}\left(\hat{\boldsymbol{Q}}_{\mathrm{IBF}}\right)$
8: $\begin{bmatrix} {}^{\mathrm{a}}\boldsymbol{\mu}_t & {}^{\mathrm{a}}\boldsymbol{\Sigma}_t \end{bmatrix} \leftarrow \mathrm{KF}_{\mathrm{m}}({}^{\mathrm{a}}\hat{\boldsymbol{\mu}}_t, {}^{\mathrm{a}}\hat{\boldsymbol{\Sigma}}_t, \mathbf{z}_{\mathrm{IBF}}, \boldsymbol{Q}_{\mathrm{IBF}}, \mathbf{I})$
9: **return** $\begin{bmatrix} {}^{\mathrm{p,v}}\boldsymbol{P}_t & {}^{\mathrm{a}}\boldsymbol{\mu}_t & {}^{\mathrm{a}}\boldsymbol{\Sigma}_t \end{bmatrix}$
10: **end procedure**

Algorithm 2 An inverse particle filter algorithm for the vehicle state estimation example.

procedure INVERSE PARTICLE FILTER($\boldsymbol{P}$)
$\begin{bmatrix} \boldsymbol{\mu}_t & \boldsymbol{\Sigma}_t \end{bmatrix} \leftarrow \text{WEIGHTED PS2GAUSSIAN}(\boldsymbol{P})$
$\begin{bmatrix} \boldsymbol{\psi}_t & \boldsymbol{\Psi}_t \end{bmatrix} \leftarrow \text{UNWEIGHTED PS2GAUSSIAN}(\boldsymbol{P})$
$\boldsymbol{K}_t^{-1} \leftarrow \boldsymbol{\Psi}_t \left(\boldsymbol{\Psi}_t - \boldsymbol{\Sigma}_t\right)^{-1}$
$\boldsymbol{Q}_t \leftarrow \boldsymbol{K}_t^{-1} \boldsymbol{\Sigma}_t$
$\mathbf{z}_t \leftarrow \boldsymbol{K}_t^{-1} \left(\boldsymbol{\mu}_t - \boldsymbol{\psi}_t\right) + \boldsymbol{\psi}_t$
return $\begin{bmatrix} \mathbf{z}_t & \boldsymbol{Q}_t \end{bmatrix}$
end procedure

in Algorithm 1. The modular particle filter largely serves as a meta-algorithm to call other algorithms based on the required parameters.

$\mathbf{x}$	a vector
$\boldsymbol{X}$	a matrix
$f(\cdot)$	a function
KF	Kalman filter
${}^{\mathrm{a}}x$	an element related to the acceleration cluster
${}^{\mathrm{p,v}}x$	an element related to the position and velocity cluster
x_{IBF}	an element related to the inverse Bayesian filter
$\hat{x}$	a temporary estimate of an element

Table 1. List of symbols used.

Line 2 begins by using a Kalman filter to integrate the previous acceleration state $({}^{\mathrm{a}}\boldsymbol{\mu}_{t-1}, {}^{\mathrm{a}}\boldsymbol{\Sigma}_{t-1})$ with the update information $(\mathbf{u}_t, \boldsymbol{M}_t)$ and the acceleration measurements $({}^{\mathrm{a}}\mathbf{z}_t, {}^{\mathrm{a}}\boldsymbol{Q}_t)$. For consistency, the previous acceleration state and the $\boldsymbol{A}_t$ matrix are both passed to the Kalman filter. In this example, the $\boldsymbol{A}_t$ matrix is also the zero matrix, so these variables do not actually contribute anything to the calculations. As such, a tightly optimized algorithm would omit these.

Since the update commands are only related to the acceleration variables, there is no need to qualify any of these variables with the cluster name. This is contrasted with the measurements, where only the variables related to acceleration are used directly in the Kalman filter. The result of the Kalman filter is another Gaussian distribution reflecting the belief about the current state of the vehicle's acceleration $({}^{\mathrm{a}}\hat{\boldsymbol{\mu}}_t, {}^{\mathrm{a}}\hat{\boldsymbol{\Sigma}}_t)$.

Line 3 uses a particle filter to integrate the previous state of the position and velocity variables $({}^{\mathrm{p,v}}\boldsymbol{P}_{t-1})$ with the control action $({}^{\mathrm{a}}\hat{\boldsymbol{\mu}}_t, {}^{\mathrm{a}}\hat{\boldsymbol{\Sigma}}_t)$ and the related sensor measurements $({}^{\mathrm{p,v}}\mathbf{z}_t, {}^{\mathrm{p,v}}\boldsymbol{Q}_t)$. The difference between a regular particle filter and the one used here, is that

it uses the results from line 2 as the control action, instead of something defined externally. The result of the particle filter is a particle set that contains all of the available information about the position and velocity variables (${}^{\mathrm{P,V}}\boldsymbol{P}_t$).

Line 4 calls an inverse particle filter to decompose the particle set generated in line 3. If we assume that the particle filter produces a weighted particle set, then the decomposition is simple. If the particle filter returns an unweighted particle set, then the prediction needs to be calculated directly. Since this is a simple problem, the particle set is just converted into a pair of Gaussian distributions.

Algorithm 2 shows the algorithm used for the inverse particle filter for this example. This allows the the prediction distribution to be removed from the result distribution. Line 2 converts the particle set into a Gaussian distribution taking the particle weights into account. This produces a Gaussian representation of the current state distribution ($\boldsymbol{\mu}_t$, $\boldsymbol{\Sigma}_t$). Line 3 extracts the predicted state information ($\boldsymbol{\psi}_t \quad \boldsymbol{\Psi}_t$) by not using the weight information. Lines 4–6 calculate the change in the distribution as though an EKF had been used. Line 4 calculates the inverse Kalman gain. Lines 5 and 6 then remove the predicted state from the final state to calculate the information that was added by the particle filter algorithm. The measurement information is left in terms of the position and state velocity variables, which makes it relatable to the acceleration state variables.

Back in the modular particle filter, the goal of lines 5–7 is to transform the measurement from the velocity space to a space that the Kalman filter can access. There is a linear relationship between the acceleration and the velocity, which enables this transformation to be applied easily. In some complicated problems, it may be easier to move this transformation into the inverse particle filter so that it can be addressed at the same time. Furthermore, a couple of simplifications could be made for this problem. Since the velocity variables are the only ones directly related to the acceleration variables, line 5 of the modular particle filter and lines 2 and 3 of the IPF algorithm could be modified to only calculate the Gaussian distribution for the velocity variables and ignore the position state variables. In addition, since the inverse particle filter calculates a Gaussian distribution for the particle set in line 2, this result could be held to be used directly in line 5 of the modular particle filter instead of recalculating it.

The measurement part of the Kalman filter used in line 8 begins with the assumption that there is an existing distribution about the acceleration state variables that merely needs a measurement integrated with it. The results from line 2 satisfy the first criterion. The measurement data calculated in lines 4–7 satisfy the second part. Thus, we can use the Kalman filter to integrate the new information into the existing distribution. The measurement part of the Kalman filter starts by calculating the Kalman gain and then integrates the new measurement.

6 Methods

To test this example and compare it to a traditional particle filter, a simulated test bed was created to match the environment described by (6)–(13) and Figure 1. Each of the information sources is calculated independently by adding noise to the true state of the related state variable, where the noise was sampled from a zero mean Gaussian distribution with a standard deviation of 0.10.

A standard KF was used for the simulation; however, the KLD sampling variant of the particle filter was used for the particle filters. This allowed the particle set to adapt to the complexity of the distribution for each situation. The KLD-sampling particle filter has some extra parameters which control the precision of the particle set; these parameters were fixed for both algorithms: $\epsilon = 0.01$, z-score of 1.645 (95%), and bin sizes of 0.025 in their respective measurements. The algorithms were initialized with a single particle representing the true state of the vehicle.

This simulator was built using Java and tested on Java version 1.6.29. EJML version 0.17 was used as the linear algebra package to calculate the Kalman filter equations. The code was not subjected to substantial optimization. Instead, code reuse was the focus of algorithm implementation so that none of the algorithms had an unfair advantage due to coding strategies. Likewise, the code was not multi-threaded during testing. The algorithms were timed by starting a nanosecond scale timer before the simulator entered into the respective modularized Bayesian filter procedure and stopped immediately after it exited the procedure. The results were then reported. The simulator repeated this process at every time step until the list of commands for the vehicle had been exhausted.

Each algorithm was run 50 times on each of three control plans. The control plans described the motion the vehicle should take for five minutes. Each plan was created separately. However, they were each similar in that the vehicle always started in the same position and ended at a stop after the five minutes. During the simulation the vehicle varied its velocity or its turning radius every few seconds.

The simulations were performed on the Big Red cluster at the University of Louisiana at Lafayette. Big Red was a homogeneous Beowulf cluster that consisted of up to 70 compute nodes connected to a primary distribution node. The cluster was built at the end of 2008 using relatively modern hardware for the time. Each of the machines had a dual-processor motherboard with two Intel® Xeon™ 2.8 GHz dual core processors and 1 GB of main memory. The nodes ran CentOS 4.4 with a 32-bit x86 SMP Linux kernel.

The simulations were scheduled so that only one simulation was run on an individual machine at a time. This helped limit memory issues with the particle filters; however, the particle filters were still limited to one million particles per particle set to keep the algorithms within the 1 GB RAM limit so that paging was not an issue. As we will see, this was only marginally successful, but did substantially reduce the overall time to produce results.

7 Results

Averaging all of the simulations at each time step produces the charts shown in Figures 4 and 5. Figure 4 shows the average amount of time required to process the Bayesian filter (vertical axis) at each time step (horizontal axis). Both algorithms are plotted as they progress through the simulation. A numerical representation of the overall averages is presented in Table 2. The table contains the mean and standard deviation for the execution time of each algorithm, as well as the size of their particle sets.

When the traditional particle filter was not being delayed by paging issues, it took about 18 s to compute; however, the modular particle filter consistently required about 1 s for its computations. As seen in Table 2, just comparing the averages directly, the modularized particle filter was about 29 times faster than the traditional particle filter under these conditions. The memory usage was substantially decreased due to the smaller particle set size. The modular particle filter required less than 9% as many particles as the traditional algorithm. The actual results are far better than this comparison shows. In this case, the traditional particle filter was explicitly limited to using 1 000 000 particles, and it did so every time. Without this limit, the particle set size continues to grow; limited testing on 16 GB machines showed that it would hit this increased limit within the first

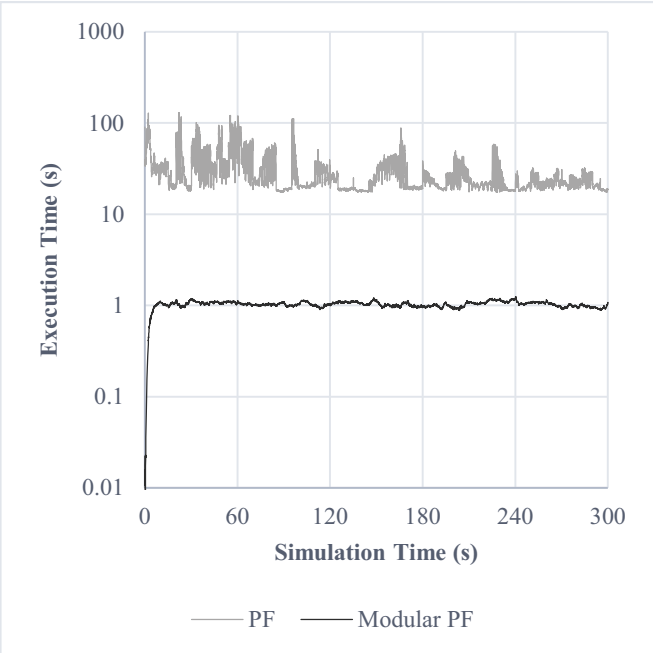

Figure 4. A chart of the simulation results showing the amount of time required at each time step throughout the simulation.

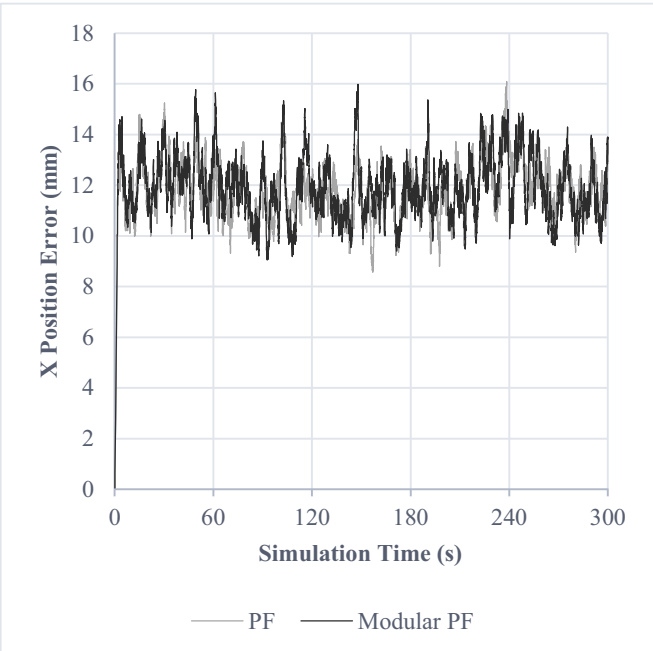

Figure 5. Mean absolute error in the x dimension over the 150 simulated runs for each algorithm. The other dimensions showed similar behavior.

couple seconds of simulation time.

Using the $\mathrm{O}()$ time complexity analysis above, we would expect the traditional particle filter to require time commensurate with $\mathrm{O}(p^9)$. This modular filter would require $\mathrm{O}(p^6)$ for the particle filter, and $\mathrm{O}(2p^6)$ to convert the particle set into Gaussian distributions for the inverse filter. The Kalman filter is cheap compared to the processing for the particle sets, and so adds a negligible amount of time to the computation. Thus, the overall computation time should be $\mathrm{O}(3p^6)$. Given that we find 87 087 particles were required to satisfy the conditions for the 6-state-variable particle filter, p should be 6.658 and we would expect the traditional particle filter to want about 25 700 000 particles. Thus we would expect about $25\,700\,000/(3 \cdot 87\,087) = 98.37$ times speedup on this problem. However, the traditional particle filter was limited to 1 000 000 particles. This would reduce the speedup to 3.83 times. Excluding the paging issues seen in the traditional particle filter, the speedup observed was around 18 times, which is close to the predicted speedup.

Table 2. A numerical presentation of the simulation results with EKF error for comparison.

Algorithm	$\hat{\mu}$	$\hat{\sigma}$
Traditional PF	29.12 s	14.51
Particle Count	1 000 000	0.000
x-Dimension MAE	11.79 mm	1.242
Modular PF	1.024 s	0.1042
Particle Count	87 087	9 445
x-Dimension MAE	11.90 mm	1.316
Traditional EKF MAE	13.07 mm	1.463

Figure 5 shows that modularization does not affect the accuracy of the filter on this problem. The mean error for both algorithms track very closely together, and any discrepancy is clearly being dwarfed by the noise from the measurements.

8 Conclusion

This research applied modularization to particle filters as a means of improving execution time. Modularization works by dividing the problem into a set of simpler sub-problems and recombining the results to maintain the accuracy of the traditional method. These modules then pass messages between them so that incoming information can be distributed throughout the network. This work also addressed a number of issues with filter construction with complex filters. An example was given to show how modularization can be applied to a practical problem. Finally, the example was tested in a simulator to show that the method is effective, even on small problems.

The results from the simulation showed that modularization can be effective even for small particle filters. On larger problems, modularization would allow implementers to choose between improving execution speed and improving the precision of the results.

9 Further Research

The most obvious place for further research is in application. We have shown results for vehicle state estimation here, but it is likely that other Bayesian filter applications would also benefit. When applied to an existing application, a relative performance comparison could be made. Modularization could also be applied to other types of Bayesian filters. We have shown how modularization works with particle filters, but there are several other filtering algorithms in use. Some of those also use specialized probability distributions which would require additional work to use as messages.

REFERENCES

[1] S.K. Andersen, K.G. Olesen, F.V. Jensen, and F. Jensen, 'HUGIN*– a shell for building Bayesian belief universes for expert systems', in *International Joint Conference on Artificial Intelligence*, volume 2, pp. 1080–1085, Detroit, MI, (1989). Morgan Kaufmann Publishers Inc.

[2] X. Boyen and D. Koller, 'Tractable inference for complex stochastic processes', in *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, UAI'98, pp. 33–42, San Francisco, CA, USA, (1998). Morgan Kaufmann Publishers Inc.

[3] X. Boyen and D. Koller, 'Exploiting the architecture of dynamic systems', in *Proceedings of the Sixteenth National Conference on Artificial Intelligence and the Eleventh Innovative Applications of Artificial Intelligence Conference Innovative Applications of Artificial Intelligence*, AAAI '99/IAAI '99, pp. 313–320, Menlo Park, CA, USA, (1999). American Association for Artificial Intelligence.
[4] A. Doucet, N. de Freitas, K. Murphy, and S. Russell, 'Rao-blackwellised particle filtering for dynamic Bayesian networks', in *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, UAI'00, pp. 176–183, San Francisco, CA, USA, (2000). Morgan Kaufmann Publishers Inc.
[5] D. Fox, 'KLD-sampling: adaptive particle filters', *Advances in Neural Information Processing Systems*, **1**(14), 713–720, (2002).
[6] D. Fox, 'Adapting the sample size in particle filters through KLD-sampling', *International Journal of Robotics Research*, **22**(12), 985–1004, (2003).
[7] R.E. Kalman, 'A new approach to linear filtering and prediction problems', *Transactions of the ASME–Journal of Basic Engineering*, **82**, 35–45, (1960).
[8] R.E. Kalman and R.S. Bucy, 'New results in linear filtering and prediction theory', *Journal of Basic Engineering*, **83**, 95–108, (March 1961).
[9] U. Kjærulff, 'dHugin: A computational system for dynamic time-sliced Bayesian networks', *International Journal of Forecasting*, **11**, 89–111, (1995).
[10] D. Koller and N. Friedman, *Probabilistic Graphical Models: Principles and Techniques*, The MIT Press, Cambridge, MA, 2009.
[11] S.L. Lauritzen and D.J. Spiegelhalter, 'Local computations with probabilities on graphical structures and their application to expert systems', *Journal of the Royal Statistical Society. Series B (Methodological)*, **50**(2), 157–224, (1988).
[12] J.D. McLean, S.F. Schmidt, and L.D. McGee, 'Optimal filtering and linear prediction applied to a midcourse navigation system for the circumlunar mission', Technical Report NASA-TN-D-1208, NASA, Ames Research Center, Moffett Field, CA, (March 1962).
[13] T.P. Minka, 'Expectation propagation for approximate Bayesian inference', in *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, eds., Jack Breese and Daphne Koller, UAI'01, pp. 362–369, San Francisco, CA, USA, (2001). Morgan Kaufmann Publishers Inc.
[14] K. Murphy, *Dynamic Bayesian Networks: Representation, Inference and Learning*, Ph.D. dissertation, University of California, Berkeley, Berkeley, CA, USA, 2002.
[15] K. Murphy and Y. Weiss, 'The factored frontier algorithm for approximate inference in DBNs', in *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, UAI'01, pp. 378–385, San Francisco, CA, USA, (2001). Morgan Kaufmann Publishers Inc.
[16] K.P. Murphy, Y. Weiss, and M.I. Jordan, 'Loopy belief propagation for approximate inference: an empirical study', in *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, UAI'99, pp. 467–475, San Francisco, CA, USA, (1999). Morgan Kaufmann Publishers Inc.
[17] B. Ng, L. Peshkin, and A. Pfeffer, 'Factored particles for scalable monitoring', in *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*, UAI'02, pp. 370–377, San Francisco, CA, USA, (2002). Morgan Kaufmann Publishers Inc.
[18] M.A. Paskin, *Exploiting locality in probabilistic inference*, Ph.D. dissertation, University of California at Berkeley, Berkeley, CA, USA, 2004. AAI3165519.
[19] J. Pearl, 'Fusion, propagation and structuring in belief networks', *Artificial Intelligence*, **29**(3), 241–288, (September 1986).
[20] M.A. Peot and R.D. Shachter, 'Fusion and propagation with multiple observations in belief networks', *Artificial Intelligence*, **48**(3), 299–318, (1991).
[21] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, Pearson Education Inc., Upper Saddle River, NJ, 2nd edn., 2003.
[22] A.F.M. Smith and A.E. Gelfand, 'Bayesian statistics without tears: a sampling - resampling perspective', *The American Statistician*, **46**(2), 84–88, (May 1992).
[23] G.L. Smith, S.F. Schmidt, and L.A. McGee, 'Application of statistical filter theory to the optimal estimation of position and velocity on board a circumlunar vehicle', Technical report, NASA Ames Research Center, Moffett Field, CA, (1962).
[24] N.L. Zhang and D. Poole, 'Exploiting causal independence in bayesian network inference', *Journal of Artificial Intelligence Research*, **5**, 301–328, (December 1996).