Particle filtering for high dimensional autonomous aerial navigation (Tech Report)

Steven L. Scott
Daedalean AI

May 31, 2018

Abstract

These notes describe a Bayesian framework that could be used for autonomous flight. The fundamental problems in autonomous flight are object recognition and aircraft location, which are similar to the SLAM algorithm for locating a robot in an unknown map. The approach used here is to work with a pre-defined dictionary of object types, each parameterized in terms of location, shape, scale, and rotation. The objects are explored using Markov chain Monte Carlo to populate a set of particles in newly observed pixels, and a specialized particle filtering algorithm to update belief about the state of the system as objects are observed in consecutive image frames.

1 Introduction

These notes describe an attempt to imbue a sensor-based robot with Bayesian reasoning so that it can can navigate through unknown surroundings. The goal is for the robot to understand its surroundings so that it can better track obstacles through time and space, and reason about which obstacles would be least costly to crash into, should a controlled crash landing become necessary.

Navigating a robot through an unknown (or partially unknown) scene is a well studied problem in AI (see Thrun, 2002; Srivastava *et al.*, 2001, and many others). A common approach is to model the robot's position in an uncertain environment using a general hidden Markov model (HMM). A general HMM takes the form

$$\begin{aligned} y_t &\sim f(y|\theta_t) \\ \theta_{t+1} &\sim g(\theta|\theta_t), \end{aligned} \tag{1}$$

where y_t is observed data at time t and θ_t is a vector of unobserved state variables. Here y_t contains all sensor measurements taken at time t, and the unobserved θ_t describes the robot's pose and surrounding environment at time t. Inferring the robot's pose in an uncertain environment using the general HMM framework is known at the "simultaneous location and mapping" (SLAM) problem (e.g. Montemerlo *et al.*, 2002).

Let $Y_t = (y_1, \ldots, y_t)$ denote the sample path of y_t up to time t. *Filtering* the data means sequentially computing $p(\theta_t|Y_t)$ for each t as new data becomes available. Closed form solutions to the filtering problem are available in two cases. When both y_t and θ_t are Gaussian, and both f and g are linear, the solution to the filtering problem is given by the Kalman filter. If θ_t is discrete and finite dimensional, then the solution is known as the *forward recursion* for HMM's. Approximate filtering solutions are available in other cases (e.g. the extended Kalman filter).

A *particle filter* (Doucet *et al.*, 2001) is a Monte Carlo approximation to the filtering distribution $p(\theta_t|Y_t)$ through a Monte Carlo ensemble $\Theta_t = \{\theta_{it}\}$ for $i = 1, \ldots, N$. Some versions of particle filters attach a weight w_i to each particle, defined so that for an arbitrary function $h(\theta)$, the integral $\int h(\theta)p(\theta|Y_t)\ d\theta \approx \sum_i w_i h(\theta_{it})$. A weighted Monte Carlo ensemble can be turned into an unweighted (or equally weighted) ensemble by sampling elements θ_{it} with replacement using the w_i as sampling weights.

The simplest particle filter is known variously as the "SIR filter" or "bootstrap filter" (Gordon *et al.*, 1993). It updates $\Theta_t \to \Theta_{t+1}$ in two steps. First, each particles θ_{it} moves to an intermediate stage $\tilde{\theta}_{it+1} \sim g(\cdot|\theta_{it})$. The intermediate ensemble $\tilde{\Theta}_{t+1} = \{\tilde{\theta}_{it+1}\}$ represents the distribution $p(\theta_{t+1}|Y_t)$. The second step reweights each particle, with weight

$w_i \propto p(y_{t+1}|\tilde{\theta}_{it+1})$. The new ensemble Θ_{t+1} is obtained by resampling $\tilde{\theta}_{it+1}$ with replacement using weights w_i.

The SIR filter is the canonical filter in the particle filtering literature, and the starting point for many other algorithms. There are two problems that prevent its use in the SLAM problem. The first is that the dimension of the particle θ_t can be very large, easily stretching into hundreds of dimensions for a particular scene, and to the tens or hundreds of thousands for long journeys comprising many scenes. The SIR filter performs well when θ_t is low dimensional, but as the dimension of θ_t grows the number of particles required to maintain an effective Monte Carlo ensemble (i.e. without all the weight collapsing onto a single particle) grows exponentially (Bengtsson *et al.*, 2008). The second problem is that some elements of θ_t represent fixed features that don't vary over time. Learning these fixed elements is analogous to learning the parameters of a statistical model, which is a challenge for the SIR filter (Liu and West, 2001; Carvalho *et al.*, 2010).

The dimensionality problem can be overcome by assuming additional structure in equation (1). Problem-specific particle filters can sometimes be constructed to take advantage of certain conditional independence relationships between elements of θ_t, and between elements of θ_t and elements of y_t. For example, Rebeschini and van Handel (2015) showed that if the elements of θ_t evolved independently, and if each element of θ_t was associated with a distinct element of y_t, then one can construct a local particle filter independent of dimension. Thrun (2002) describes a similar result in the context of the SLAM problem, in which a Rao-Blackwellized particle filter is applied in upwards of 100,000 dimensions. Rao-Blackwellization is a technique in which some components of the state space are marginalized out, conditional on a component set drawn using a particle filter (Doucet *et al.*, 2000; Murphy and Russell, 2001).

The remainder of this report is structured as follows. Section 2 describes the model used to locate the aircraft and describe the universe of obstacles. Section 3 describes Markov chain Monte Carlo and particle filtering strategies for describing the state of the aircraft. Section 4 describes some preliminary computer experiments with the computational techniques from Section 3. Section 5 summarizes conclusions and lays out a potential framework to extend this preliminary research to a method that could be used for actual flight. These notes describe roughly 6 weeks of work, so they are quite preliminary.

2 Flight as a hidden Markov model

The data to be tracked, y_t, is a two-dimensional array of pixels as seen by the camera at time t. In principle y_t could contain other sensor measurements as well. Let $Y_t = y_1, \ldots, y_t$ denote the sequence of images (or more generally, sensor readings) observed thus far. The belief state of an aircraft is a high dimensional vector containing subcomponents describing the aircraft's pose α_t as well as parameterized objects in the environment, such as trees, houses, and other aircraft. A posterior distribution over the belief state at time t can be represented by an ensemble of particles

$$\theta_{it} = (\alpha_{it}, \nu_{i1}, \ldots, \nu_{iK_i}) \tag{2}$$

where θ_{it} is a draw from $p(\theta|Y_t)$. In equation (2), ν_{ij} represents a fixed object to be tracked. Object ν_{ij} has subcomponents $\nu_{ij} = (\delta_{ij}, \phi_{ij}, \chi_{ij})$, where $\delta_{ij} \in \{1, \ldots, M\}$ is a class label for the different types of objects being tracked (e.g. $\delta_{ij} = 1$ means the object is a house, while $\delta_{ij} = 2$ means it is a tree), and ϕ_{ij} is a vector of real values describing the object's shape, such as its height, width, orientation, etc. The dimension and distribution of ϕ_{ij} depends on δ_{ij}. Finally, $\chi_{ij} = \mu_{ij}, V_{ij}$ is the mean and variance of the distribution describing the colors of the pixels contained in the object.

Let y_{tij} denote the i, j pixel in the image y_t. For grayscale images, pixel is a value between 0 and 255. For color images a pixel is a triple of such values describing the red, green, and blue contributions to the pixel color. Our preliminary model for pixel values can be thought of as a type of finite mixture model, where the mixture components describe the objects to be tracked. The conditional distribution of $y_t|\theta_t$ is

$$p(y_t|\theta_t) = \prod_{k=0}^{K} \prod_{i,j \in \nu_k(\alpha_t)} \mathcal{N}\left(y_{tij}|\mu_k, V_k\right), \tag{3}$$

where $\chi_k = \{\mu_k, V_k\}$ are the mean and variance of the color distribution for object k, and where we sloppily write $i, j \in \nu_k(\alpha)$ to mean that pixels i and j are in the shape produced by object ν_k when viewed from aircraft pose α_t, and we let ν_0 denote the background object, which is defined as the absence of any other object. We assume the objects are ordered so that if

$i < j$ then ν_i obscures ν_j. The potential for obscuring induces a correlation between colocated objects in the observation equation, but the objects remain independent in the transition equation.

Equation (3) is a simple model that could easily be extended by embedding the color distribution in a hierarchy that allows for object specific random effects, or by a more sophisticated model that incorporated texture, etc. The key feature of equation (3) is that pixel colors in different objects are conditionally independent (given the objects). This feature, which would be preserved in the more complex models described above, allows us to evaluate the likelihood of different regions of the image independently. Conditional independence is the key to updating particles in a piecewise manner and for parallelizing otherwise expensive Monte Carlo computations required to work with the posterior distribution.

Now consider plausible prior distributions. To begin, assume that objects occur according to a homogeneous Poisson process in two dimensions. The homogeneity assumption could be easily generalized to an inhomogeneous Poisson process with trivial modifications to the mathematical and computational procedures described below. An inhomogeneous Poisson process would be useful if a detailed map was available *a priori* , so that buildings tended to occur more frequently in cities and towns, for example. The homogeneous Poisson process induces a Poisson prior on the total number of objects in a area, a conditionally multinomial prior on the object types (with prior category rates proportional to Poisson intensity), and a uniform prior on object locations. Let B denote the region (the new pixels) over which the algorithm is to be run. Let $|B|$ denote the volume of B. Let N_m denote the number of objects of type m found in B. Because objects are modeled as a Poisson process, $N_m \sim Po(\lambda_m|B|)$. Organize the shape and color parameters within an object family, so ϕ_{mj} and χ_{mj} are the ϕ and χ parameters for object j in object family m. Then the prior distribution is

$$p(\theta) = \prod_{m=1}^{M} p(N_m) \prod_{j=1}^{N_m} p_m(\phi_{mj}) p_m(\chi_{mj}). \qquad (4)$$

Equation (4) assumes that an object's shape is independent of its position, and that an object's color distribution is independent of both shape and position. We furthermore assume that $p_m(\chi_j)$ is a normal-inverse gamma prior (or normal-inverse Wishart in the case of color images). The conditional conjugacy resulting from the normal inverse gamma prior allows χ to be analytically integrated out of most posterior computation, which both mathematically convenient and computationally efficient. The remaining priors over the object's size and rotation are assumed to be uniform over a defined range.

3 Posterior exploration and updating

With each observed frame in the image sequence there are two related tasks that must be accomplished. First, the new frame may contain pixels showing a previously unobserved part of a scene. Particles must be updated with a Monte Carlo ensemble describing the state of the new scene. This is most effectively done using Markov chain Monte Carlo.

Obtaining a reasonable representation initial of scene elements is critical for subsequent particle filter updates, which are used to firm up belief states as objects are repeatedly viewed in subsequent image frames. For stationary objects, the initialization is the only time the prior distribution comes into play.

3.1 Initializing new pixels

A set of pixels corresponding to a newly observed region of physical space is first scanned with an MCMC algorithm. To account for the uncertainty in the number, type, size, and location of the objects in the new region one must consider several types of MCMC moves that should be considered. These include

1. Moving an object through parameter space by changing its location, size, or rotation.
2. Adding or deleting a new object at a random location.
3. Splitting one object into two, or merging two objects into one.
4. Morphing an object from one type to another (e.g. a circle changes to a square or vice versa).

In principle, the MCMC algorithm can be initialized by saturating the scene with objects and letting the add/delete move remove the ones that don't belong. Quickly identifying that there is an object in a location is more important than getting the correct object type, because the type can be corrected later by the morph move.

All the above moves can be implemented using variants of the Metropolis-Hastings (MH) algorithm.

The MH algorithm simulates from a target distribution $\pi(\theta)$ by proposing a candidate value $\theta' \sim q(\theta|\theta^{(t)})$, where $\theta^{(t)}$ is the current parameter value. The candidate is compared to the current value using the MH acceptance ratio

$$\alpha_M = \frac{\pi(\theta')/q(\theta'|\theta^{(t)})}{\pi(\theta^{(t)}|\theta')/q(\theta^{(t)}|\theta')}. \quad (5)$$

With probability $\min(\alpha_M, 1)$ the candidate is promoted to the next value (meaning $\theta^{(t+1)} = \theta'$). Otherwise $\theta^{(t+1)} = \theta^{(t)}$ and the algorithm does not move. The sequence of draws from the MH algorithm forms a Markov chain with stationary distribution π. The rate of convergence is determined by the choice of proposal distribution q. The different moves described above are thus implemented using different choices for q.

In our case, the target distribution $\pi(\theta)$ comes from Bayes' rule so that $\pi(\theta) \propto p(y_t|\theta)p(\theta)$, the product of equations (3) and (4). The unknown normalizing constant in $\pi(\theta)$ is irrelevant because it cancels in the MH ratio. In fact, most of the likelihood cancels as well. Each of the moves we consider operates on one object at a time. Because of the conditional independence assumptions in equation (3), components of the image that are unaffected by the proposed change can be ignored because they will cancel in equation (5).

3.1.1 Moving, resizing, and rotating objects

Each object has location parameters (x, y) that determine its center on the map, a size parameter r, and a rotation parameter ζ. The location, size, and rotation moves are implemented as three separate MH moves.

The location move proposal must balance between large scale moves, which are needed when an object model is searching for an object to which to bind, and fine scale moves, for when an object has been found and minor adjustments are needed to optimize its location.

In one version of the location move the proposal distribution is a mixture, with probability .5 of proposing from a two dimensional uniform distribution on a small box centered on the current object. The remaining probability is on a uniform distribution over the space of new pixels.

In experiments the fine tuning portion of the algorithm worked extremely well once the object had been located, but the large scale move was unreliable, sometimes taking several hundred iterations to find the right neighborhood. We developed two alternatives to address this issue.

The first is a MH proposal based on a grid search. Partition the region to be sampled into rectangles of the same size and shape as the bounding box containing the object. Sample a grid cell from the multinomial distribution with probability proportional to the target density that would obtain if the object were moved to the center of each grid cell. Given

In experiments, each call to the slice sampler was found to be considerably more expensive than a MH update, but the slice sampler did a better job of finding the correct location, size, and orientation of objects in simulated data.

3.1.2 Splitting and Merging Objects

For the split move, let $\theta = (\alpha, \nu_1, \ldots, \nu_K)$ denote the current state, and assume α is known (and thus ignored in subsequent notation).

1. Choose a random object ν_j from the current set of objects.
2. Randomly generate a second object ν_{K+1} from the prior distribution, centered at a random location in the bounding box of ν_j.
3. Evaluate the Metropolis-Hastings ratio and decide whether to accept the move.

Let $\tilde{\theta}$ denote θ with ν_{K+1} appended. The Metropolis-Hastings ratio is

$$\alpha_M = \frac{p(\mathbf{y}|\tilde{\theta})p(\tilde{\theta})/q(\tilde{\theta}|\theta)}{p(\mathbf{y}|\theta)p(\theta)/q(\theta|\tilde{\theta})}. \quad (6)$$

The ratio of likelihood functions cancels in all pixels not covered by either ν_j or ν_{K+1}, so only those pixels need to be considered. Let δ indicate the object type of ν_{K+1}, and let N_δ be the number of objects of type δ prior to ν_{K+1} being added. The ratio of priors is

$$\frac{p(\tilde{\theta})}{p(\theta)} = \frac{\lambda_\delta}{N_\delta + 1} p_\delta(\phi_{K+1}). \quad (7)$$

Note that χ is absent from equation (7) because the color distribution has been integrated out. Let $\Lambda = \sum_k \lambda_k$. The proposal density $q(\tilde{\theta}|\theta)$

$$q(\tilde{\theta}|\theta) = \frac{1}{K} \frac{\lambda_\delta}{\Lambda} p_\delta(\phi_{K+1}). \quad (8)$$

contains a component $1/K$ for choosing the object to split, a component proportional to λ_δ for choosing the type of the second object, and a component for the ϕ parameters that were simulated from the prior.

The last piece needed to evaluate equation (6) is the distribution of the reverse move $q(\theta|\tilde{\theta})$. To get that we need to define the merge move. To preserve reversibility, the merge move must mirror the split move. Therefore the merge move is

1. Choose a uniformly random object ν_j from the current set of objects.
2. Choose a second object uniformly randomly from among the objects centered within the bounding box of ν_j. If there are no other objects in the bounding box then reject the move.
3. Remove the second object.
4. Evaluate the MH ratio and decide whether to accept the move. The likelihood need only be evaluated within the union of bounding boxes of the original two objects.

Let K denote the number of objects present after the merge (so that K has the same meaning in both the split and merge discussions), and let b_j denote the number of objects present in the bounding box of ν_j.

$$q(\tilde{\theta}|\theta) = \frac{1}{K+1}\frac{1}{b_j}. \tag{9}$$

Equation (9) completes the specification of equation (10), and equations (7), (8), and (9) collectively define the MH ratio for the merge move as well.

The MH ratio for the split move is thus

$$\begin{aligned}\alpha_M &= \frac{p(\mathbf{y}|\tilde{\theta})}{p(\mathbf{y}|\theta)}\frac{\lambda_\delta}{N_\delta+1}\frac{K\Lambda}{\lambda_\delta}\frac{1}{(K+1)b_j} \\ &= \frac{p(\mathbf{y}|\tilde{\theta})}{p(\mathbf{y}|\theta)}\frac{K}{K+1}\frac{\Lambda}{(N_\delta+1)b_j}.\end{aligned} \tag{10}$$

3.1.3 Morphing moves

The morph move is as follows

1. Choose a random object ν_j from the set of objects.
2. Choose a random object type δ_j' with probability proportional to λ_k, for $k \neq \delta_j$.
3. Simulate a set of object parameters $\phi_j' \sim p_{\delta_j'}(\phi_j')$. Scale the object so that it contains the same area as ϕ_j.
4. Swap ν_j' for ν_j, evaluate the MH ratio and decide whether to accept the proposal.

As above, let $\tilde{\theta}$ denote the candidate value, which in this case is θ with ν_j' replacing ν_j. Let N_δ and $N_{\delta'}$ denote the number of objects of type δ_j and δ_j' before the morph, so the class sizes after the morph are $N_\delta - 1$ and $N_{\delta'} + 1$. The proposal distribution for the morph move contains a factor of $1/K$ for selecting which object to change, a factor of $\lambda_{\delta'}/(\Lambda - \lambda_\delta)$ for selecting which object type to morph to, and a factor of $p_{\delta'}(\phi_j')$ for the new values of the parameters.

The ratio of the prior and proposal distributions for the morph move is

$$\frac{p(\tilde{\theta})}{q(\tilde{\theta}|\theta)} = C_0\frac{\lambda_\delta^{N_\delta-1}\lambda_{\delta'}^{N_{\delta'}+1}}{(N_\delta-1)!(N_{\delta'}+1)!}\frac{K(\Lambda-\lambda_\delta)}{\lambda_{\delta'}}. \tag{11}$$

Factors of $p(\phi_j)$ in equation (11) cancel because the proposal distribution simulates from the prior. The constant C_0 contains factors from the prior distribution unrelated to ν_j or ν_j'.

The corresponding ratio for the reverse move is symmetric.

$$\frac{p(\theta)}{q(\theta|\tilde{\theta})} = C_0\frac{\lambda_\delta^{N_\delta}\lambda_{\delta'}^{N_{\delta'}}}{(N_\delta)!(N_{\delta'})!}\frac{K(\Lambda-\lambda_{\delta'})}{\lambda_\delta}. \tag{12}$$

Dividing equation (11) by equation (12) and multiplying by the likelihood ratio gives the MH ratio for the morph move.

$$\alpha = \frac{p(\mathbf{y}|\tilde{\theta})}{p(\mathbf{y}|\theta)}\frac{\Lambda-\lambda_\delta}{\Lambda-\lambda_{\delta'}}\frac{N_\delta}{N_{\delta'}+1}. \tag{13}$$

The likelihood ratio only needs to be evaluated on the union of the bounding boxes for ν_j and ν_j'.

3.1.4 Add/Delete moves

The add move is

1. Choose a random object from among the set of object types, with probability proportional to λ_k.
2. Simulate its parameters from the prior, and locate it at a random point in the bounding box of the new pixels.

Let δ be the type of the new simulated object. The proposal distribution is

$$q(\tilde{\theta}|\theta) = \frac{\lambda_\delta}{\Lambda}p_\delta(\phi_{K+1}) \tag{14}$$

The delete move is

1. Choose a random object from the set of objects and delete it.

The proposal distribution for the delete move is simply $p(\theta|\tilde{\theta}) = 1/(K+1)$.

Let $\tilde{\theta}$ denote the proposal after an addition has been proposed. The Metropolis-Hastings ratio for the add move is

$$\begin{aligned}\alpha &= \frac{p(\mathbf{y}|\tilde{\theta})}{p(\mathbf{y}|\theta)} \frac{p(\tilde{\theta})}{p(\theta)} \frac{\Lambda}{\lambda_\delta} \frac{K+1}{p_\delta(\phi_{K+1})} \\ &= \frac{p(\mathbf{y}|\tilde{\theta})}{p(\mathbf{y}|\theta)} \frac{\lambda_\delta^{N_\delta+1} N_\delta! \Lambda (K+1)}{\lambda_\delta^{N_\delta+1} (N_\delta+1)!} \\ &= \frac{p(\mathbf{y}|\tilde{\theta})}{p(\mathbf{y}|\theta)} \frac{\Lambda(K+1)}{N_\delta+1}.\end{aligned} \tag{15}$$

The factor of Λ in equation (15) makes sense, because an add move will tend to be accepted if the overall object frequency is high. The ratio $(K+1)/(N_\delta+1)$ is the proportion of objects of type δ out of all objects, which also make sense.

The MH ratio for the deletion move is the inverse of equation (15), where $K+1$ and $N_\delta+1$ are the sizes relative to the original θ.

3.2 Particle filtering

The envisioned particle filtering algorithm has two components: an initialization step where previously unobserved pixels are filled with objects, and a filtering step where the ensemble of potential scenes is updated as the scenes are repeatedly observed.

4 Simulation Examples

4.1 One dimension, no motion

The simplest example considered is a 1-d simulation with no motion. The goal is to determine an effective method of identifying simple objects. The object to be located in this section is an interval.

The problem setup is a one-dimensional image where the pixels are gray (and thus a single number). The image contains an interval observed in background noise. Background pixels are modeled as independent Gaussian noise with a large standard deviation. Pixels inside the interval to be detected are modeled as independent Gaussian noise with a different mean and standard deviation. The location and size of the interval are to be discovered by the detector. The upper left panel of Figure 2 shows a simulated example. In this case the interval we wish to detect is centered at 50 with a radius of 5.

If we treat the background mean and standard deviation as known, then this is a 4 dimensional problem, with the unknowns being interval center, interval radius, and the mean and standard deviation of the color distribution in the interval. We compare three methods of building an ensemble over this distribution. The first is straight particle filtering, where four-dimensional particles are simulated from a uniform prior and weighted according to equation (3). The second method simulates two-dimensional particles representing the interval's center and radius, with the mean and variance of the color distribution integrated out with respect to a normal-inverse gamma prior. The third method uses Markov chain Monte Carlo to create a particle ensemble.

Figure 2 compares the two particle filtering methods. The top left plot shows the grayscale intensities for the pixels in a 1-d image. The central plot in the top row of Figure 2 shows the posterior ensemble of intervals produced by method 1. Each interval in the ensemble is colored according to its weight, with the most heavily weighted interval being black, and with colors grading to white as the weights degrade to zero. Intervals with weights numerically indistinguishable from zero are not shown. The position of each interval on the vertical axis is random, and thus conveys no useful information. The center-top plot in Figure 2 only shows a single interval, because even with 10,000 particles all the weight collapses to a single particle, which gets the interval's center about right, but fails to get the radius. As a single particle it definitely fails to represent any posterior uncertainty.

The plot in the top right of Figure 2 shows the posterior distribution of the intervals when the pixel color mean and variance are integrated out with respect to a prior. Integrating out the color distribution is a form of Rao-Blackwellization that reduces the effective dimension of the problem from 4 to 2. The large number of nearly white intervals obscure many of the image points, showing that the ensemble is maintaining (small) weight in the tails of the distribution.

The bottom left panel of Figure 2 compares the posterior distribution of the weights under the two likelihoods. The left boxplot shows the weights under method 1 (where a single point has all the weight), and the right boxplot shows the weights under method 2, which integrates out the color distribution parameters. The final two scatterplots describe

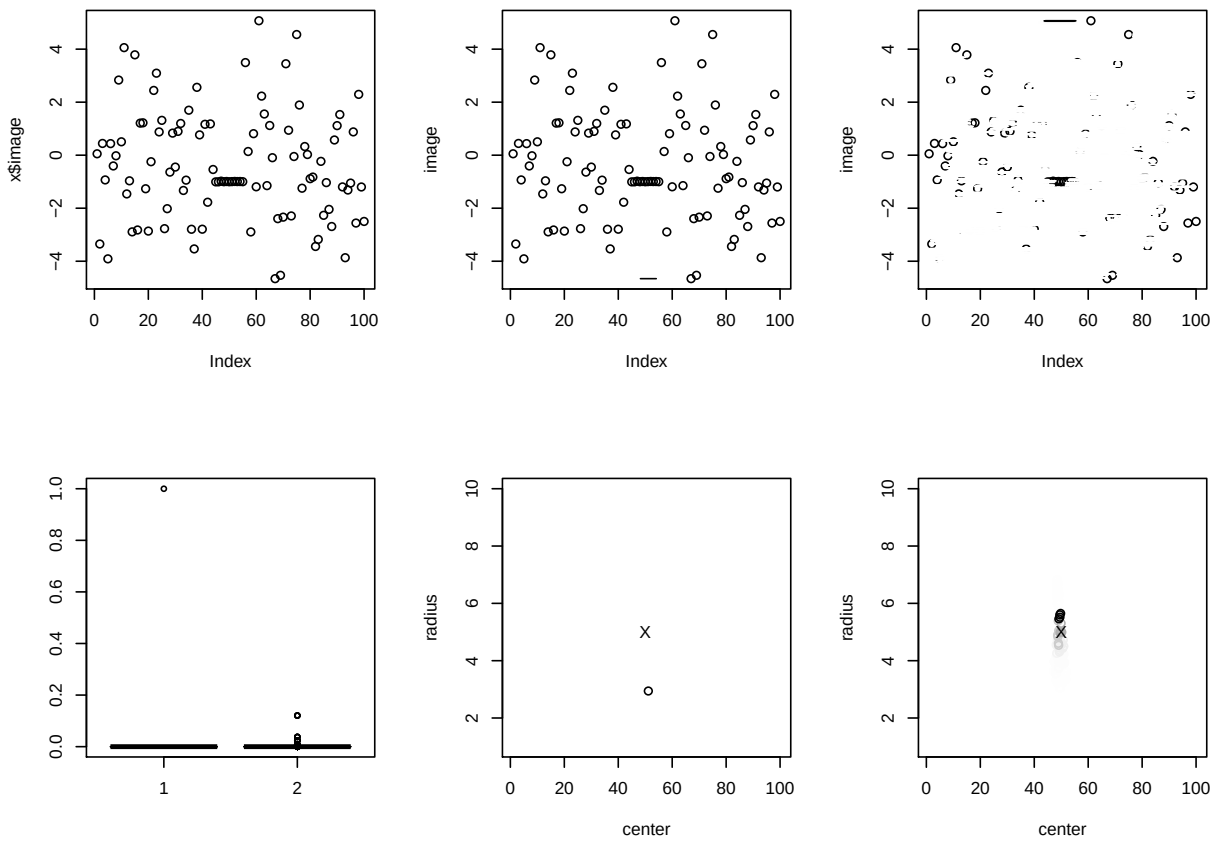

Figure 2: *Particle filtering results with 10,000 particles. Top row: (left) the test image, (center) the particles generated by method 1 (the single black interval near the bottom of the plot), (right) the particles generated by method 2. Bottom row: (left) boxplots comparing the distribution of weights, (middle) weighted distribution of interval centers and radii from method 1 (truth is shown by an 'X'), (right), weighted distribution of interval centers and radii from method 2. See the text of Section 4.1 for further explanation.*

the joint distributions of the center and the radius of the interval. The center-bottom plot shows the Monte Carlo estimate from the particles based on equation (3), while the plot in the bottom right shows the distribution after integrating out the color parameters. Like the intervals in the top row, the points in the bottom row are colored with darker points having higher weight. There is only one point to show in the plot based on method 1, but the weights based on integrated likelihood are spread out enough to get a reasonable view of the posterior uncertainty.

Figure 2 demonstrates that integrating out the color distribution is a clear win. The next comparison is between the particle filter from method 2 and an MCMC algorithm. The target distribution in the MCMC algorithm is the integrated likelihood used in method 2, assuming a uniform distribution over the interval's center and radius. We can use a slice sampler (Neal, 2003) to quickly sample from this distribution. Several hundred MCMC draws take less than 1 second in R, and would be much faster in a compiled language. Figure 3 plots the results of an MCMC run of 500 iterations on the same data from Figure 2. The top panel in Figure 3 shows the integrated log likelihood associated with each MCMC draw, which can be used to assess convergence. The bottom two panels show MCMC sample paths of the interval's center and radius parameters, which are well estimated once convergence is achieved.

The MCMC algorithm with 500 draws locates the

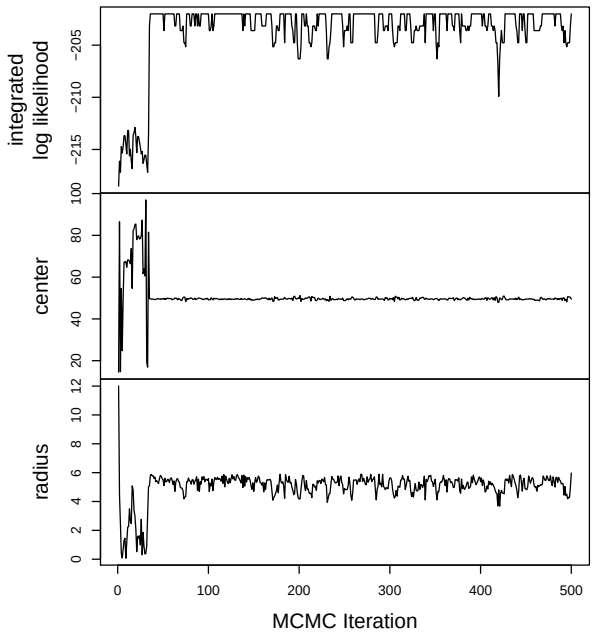

Figure 3: *Posterior distribution obtained from an MCMC run of 200 iterations.*

interval significantly better (and faster) than the particle filter with 10,000 particles. It is impractical to run an MCMC algorithm for each frame of a moving image, but these results suggest using MCMC initialize the particle filter when a new portion of an image is observed.

4.2 A simulation in 2-dimensions

Several simulations were done to validate the methods described in Section 3.1. The simulations sought to identify the correct number of circles and squares in an image like the one shown in Figure 4. The simulations were primarily implemented as unit tests in the C++ code implementing the algorithm. Each step was validated and found to perform adequately, but funding considerations put a halt to this research around this time.

The next steps milestones were to have been putting the individual MH moves together and optimizing them to process an entire image, and then marrying the MH initialization with a Rao-Blackwellized particle filter to model image sequences.

5 Conclusion

As mentioned in the introduction, this research project lasted only about 6 weeks, which wasn't enough time to get definitive results. A considerable amount of time was spent on software engineering issues: debugging and unit testing. The following lessons were learned during the programming and software engineering part of this exercise.

- Searching for an object location is harder in two dimensions than in one dimension because it can take substantially longer for an object randomly walking in two dimensions to stumble upon a high likelihood region.

- The slice sampling algorithm worked extremely quickly in one dimension, but was much slower in two dimensions, primarily because the likelihood function involves many more pixels. Economizing log likelihood calculations proved vital to getting an algorithm that ran anywhere near efficiently. Quite a bit of programming time was spent computing with unions and intersections of bounding boxes enclosing various shapes, and these optimizations sped up the compute time of the algorithm considerably. Likewise, the fact that the slice sampler required so many likelihood evaluations led me to

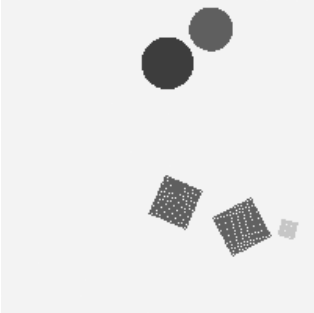

Figure 4: *An example of an image from a simulated flight. The obstacles are circles and squares in different orientations, sizes, and colors.*

explore several different MH updating schemes for two-dimensional location updates.

- It is a shame that this work stopped just prior the point where the full MCMC algorithm would get unleashed on an image. The combination of several moves would hopefully have compensated for some of the deficiencies found in individual move types. For example, if the algorithm had been initialized with several objects, then the ensemble of MCMC objects would have been able to find "real" objects faster than the single MCMC object that was tracked in the unit tests.

In terms of future directions, once the "circles and squares" problem is solved, the next step is to move to more complex three dimensional objects. That is a conceptually small step, but it involves large practical challenges. In our two dimensional example there is an obvious equivalence between an object's location in pixel space and its location in physical space. Moving to three dimensions complicates that relationship. A physical model is needed to describe each object's expected appearance as a function of its relationship to the aircraft's position. However, I presume this is a solved problem and (e.g.) video game simulation software could be used to provide this kind of mapping. A somewhat more sophisticated MCMC algorithm is also needed for morphing between objects. For example, the proposal distribution should favor morphs between objects of similar size. That would help speed things up as the object dictionary grew in size.

And finally, I never got around to working out the details of the RBPF. Because the model for the pixels contains so much conditional independence, there are opportunities to shuffle objects between particles in ways that would hopefully prevent particle collapse.

References

Bengtsson, T., Bickel, P., and Li, B. (2008). *Curse-of-dimensionality revisited: Collapse of the particle filter in very large scale systems*, vol. Volume 2 of *Collections*, 316–334. Institute of Mathematical Statistics.

Carvalho, C. M., Johannes, M. S., Lopes, H. F., and Polson, N. G. (2010). Particle learning and smoothing. *Statistical Science* **25**, 88–106.

Doucet, A., De Frietas, N., and Gordon, N. (2001). *Sequential Monte Carlo in Practice*. Springer.

Doucet, A., de Frietas, N., Murphy, K., and R (2000). Rao-blackwellised particle filtering for dynamic bayesian networks. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, 176–183.

Gordon, N. J., Salmond, D. J., and Smith, A. F. M. (1993). Novel approach to nonlinear/non-Gaussian Bayesian state estimation. *IEE Proceedings Part F: Communications, Radar and Signal Processing* **140**, 107–113.

Liu, J. and West, M. (2001). *Combined Parameter and State Estimation in Simulation-Based Filtering*, chap. 10, 197–223. Springer.

Montemerlo, M., Thrun, S., Koller, D., and Wegbreit, B. (2002). Fastslam: A factored solution to the simultaneous localization and mapping problem. In *Eighteenth National Conference on Artificial Intelligence*, 593–598, Menlo Park, CA, USA. American Association for Artificial Intelligence.

Murphy, K. and Russell, S. (2001). *Rao-Blackwellised Particle Filtering for Dynamic Bayesian Networks*, chap. 24, 499–515. Springer.

Neal, R. M. (2003). Slice sampling. *The Annals of Statistics* **31**, 705–767.

Rebeschini, P. and van Handel, R. (2015). Can local particle filters beat the curse of dimensionality? *The Annals of Applied Probability* **25**, 2809–2866.

Srivastava, A., Lanterman, A. D., Grenander, U., Loizeaux, M., and Miller, M. I. (2001). *Monte Carlo Techniques for Automated Target Recognition*, chap. 26, 533–552. Springer New York, New York, NY.

Thrun, S. (2002). Particle filters in robotics. In *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*, UAI'02, 511–518, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.