LaTeX Template

Benjamin Washington-Yule

September 19, 2011

Abstract

This paper details the particle filter. A particle filter is a class of sophisticated sequential estimation techniques for estimating the state of a dynamical system. To introduce a Particle filter, first the Bayes filter is introduced which encompasses both the Particle filter and the Kalman filter. The implementation of a particle filter is examined and compared to a Kalman filter where results are compared and concluded. Testing was preformed on a video of a spherical ball dropping and bouncing.

1 Introduction

The use of particle filters in signal processing is a comparatively new area of research. The particle filter is a type of *recursive Bayesian estimator*, also known as a *Bayes filter*, and is closely related to another type of Bayes filter called the *Kalman filter*. Bayes filters find applications in a wide range of areas, which includes among others,

- Robotics, to infer position and orientation.
- Computer vision, to track objects.
- Medical fields, to extract information from noisy data.

A Bayes filter is used in sensor and data fusion. It allows sensor measurements to be combined with a state model to accurately track the state of a system.

The particle filter differs from the Kalman in the assumptions that are made about the underlying system model and sensor noise. A more rigorous discussion of these differences is found in Section 2 but the particle filter can be generally described as having looser requirements, i.e., fewer assumptions must be made about the model and sensor noise for a particle filter. This may mean that in some cases a particle filter will succeed where a Kalman fails.

2 Background

Both particle and Kalman filters are types of *recursive Bayesian estimators*, or *Bayes filters*. A Bayes filter provides a probabilistic framework for estimation,

but cannot be instantiated itself. In other words, particle and Kalman filters are both implementations of the generic Bayes filter. As such, they share very similar properties, and a designer must go through the same initial steps for both.

To avoid confusion, the *system* refers to the system whose state we are attempting to estimate. The Bayes filter uses a *dynamical model* and *observations* to achieve this. Depending on the context, both *observations* and *sensor measurements* are used in the text, and mean the same thing.

2.1 State Space

A Bayes filter is used for estimating the *state* of a dynamic system from noisy sensor measurements. The dynamic system is described by the state vector **x**, and the following generic notation is used.

$\mathbf{x}_t$ state at time t
$\mathbf{z}_t$ observation at time t
$\mathbf{u}_t$ system input at time t

Note that all of the state variables are vectors, indicating that a system can have multiple states, observations and inputs. To illustrate how a system can be decomposed into its state space, consider the spring-mass-damper system shown in Figure 1.

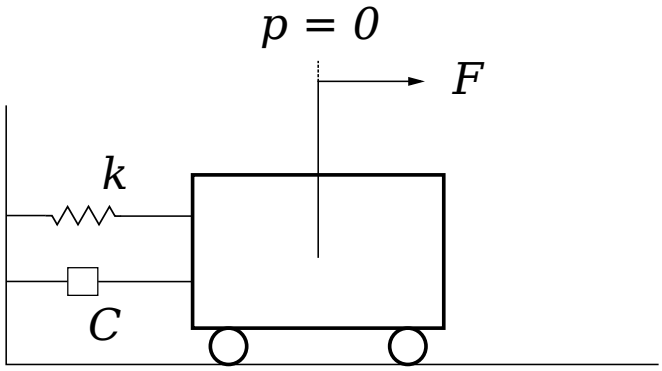


Figure 1: Spring-mass-damper system. k and C are the spring and damping coefficients respectively, p is the position referenced from zero and F the applied force.

The cart shown in the Figure is anchored to the wall by a spring and damper. It slides on a frictionless surface and has unity mass. There is an input $u = F$ which excites the system.

We arbitrarily decide that the state of the system is composed of the two variables[1] : position p and velocity $\dot{p}$. We can then use Newton's second law, $\sum F = ma$, to develop a force balance equation:

[1]Note that we could have chosen any property of the system as a state variable. Such a simple model offers few options however.

$$F_{spring} = -kp$$
$$F_{damper} = -C\dot{p}$$

$$\begin{aligned}\sum F &= ma \\ &= m\ddot{p} \\ &= -C\dot{p} - kp + u\end{aligned}$$

Which allows us to write a single differential equation to describe the dynamics of the system.

$$\implies m\ddot{p} + C\dot{p} + kp = u \tag{1}$$

The system state consists of position p and velocity $\dot{p}$, and we form the state vector thus:

$$\mathbf{x} = \begin{bmatrix} p \\ \dot{p} \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{2}$$

From which we can create the state space,

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{p} \\ \ddot{p} \end{bmatrix}$$

$$= \begin{bmatrix} \dot{x} \\ \frac{C}{M}\dot{p} - \frac{k}{M}p \end{bmatrix}$$

$$= \begin{bmatrix} x_1 \\ \frac{C}{M}x_2 - \frac{k}{M}x_1 - u \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 1 \\ \frac{-k}{M} & \frac{-C}{M} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u$$

Which has the usual state space form,

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}u \tag{3}$$

where

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ \frac{-k}{M} & \frac{-C}{M} \end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

We model the sensors similarly. The values of the matrices depend on which modes we are able to observe. In this example we have said that we are able to observe position, but not velocity. Our sensor vector is therefore simply a scalar value equal to the first mode (position) of the system:

$$\mathbf{z} = \begin{bmatrix} z \end{bmatrix} = x_1 \tag{4}$$

The state space observation equation is therefore:

$$z = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} u \tag{5}$$

which we write using the usual state space notation.

$$z = \mathbf{C}\mathbf{x} + \mathbf{D}u \tag{6}$$

These are the key equations used in both the particle and Kalman filtering algorithms.

Although the derivations of these state space forms are long-winded, the formation of the state space is an important step for a Bayes filter.

2.1.1 Discrete System Issues

Even though we have the system in state space form, it is not quite ready for use as a model for a Bayes filter. There is a subtle change in the form of equations 3 and 6 when used to model a discrete system, such as when modelling using Matlab. The changes are shown here without justification.[2]

$$\mathbf{x_{t+1}} = \mathbf{F}\mathbf{x_t} + \mathbf{G}u_t \tag{7}$$

$$z_t = \mathbf{H}\mathbf{x_t} + \mathbf{J}u_t \tag{8}$$

where

$$\mathbf{F} = \Delta \mathrm{t}\mathbf{A} + \mathbf{I_4} \qquad \mathbf{H} = \mathbf{C}$$
$$\mathbf{G} = \Delta \mathrm{t}\mathbf{B} \qquad \mathbf{J} = \mathbf{D}$$

The reason for these changes is beyond the scope of this discussion and does not affect any results.

[2] TODO

2.2 Estimation Process

The aim of the Bayes filter is to find the belief about the current state,

$$Bel(\mathbf{x}_t) = p(\mathbf{x}_t|\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_t) \tag{9}$$

that is, the probability of $\mathbf{x}_t$ given all the data we've seen so far. Roughly speaking, the belief answers the question: *What is the probability that the system's state is* $\mathbf{x}$ *if the history of sensor measurements is* $\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_t$?

The number of observations grows with time, and will get large very quickly (withing a few seconds) in any system where measurements are taken at a high rate. To make the computation tractable, Bayes filters assume that the dynamic system is a *Markov* process[3]. The result of this assumption is that we can assume that the current state $\mathbf{x}_t$ contains all the information we need. Using the cart example given previously, the Markov assumption implies that sensor measurements depend only on the carts current position and velocity, and that its position and velocity (stat) at time t depends only on the previous state $\mathbf{x}_{t-1}$. That is, states before $\mathbf{x}_{t-1}$ provide no additional information under this Markov assumption.

With this assumption made, the belief function simplifies to:

$$Bel(\mathbf{x}_t) = p(\mathbf{x}_t|\mathbf{z}_t) \tag{10}$$

2.2.1 Updating the Bayes Filter

Whenever a sensor provides a new observation[4] $\mathbf{z}_t$, the filter predicts state according to (11). Note that the new sensor observation is not used in this step.

$$p(\mathbf{x}_t|\mathbf{z}_{t-1}) = \int p(\mathbf{x}_t|\mathbf{x}_{t-1})p(\mathbf{x}_t|\mathbf{z}_{t-1})\mathrm{d}\mathbf{x}_{t-1} \tag{11}$$

The filter then corrects the predicted estimate using the new sensor observation according to[5] (12).

$$p(\mathbf{x}_t|\mathbf{z}_t) = \frac{p(\mathbf{z}_t|\mathbf{x}_t)p(\mathbf{x}_t|\mathbf{z}_{t-1})}{p(\mathbf{z}_t|\mathbf{z}_{t-1})} \tag{12}$$

[3] TODO: explanation of a Markov process

[4] Although written as a vector generally it is worth noting that in many systems, including the cart system above, there is only a single observation and the observation vector has only one element, i.e., it is a scalar

[5] We can write (12) as $\alpha p(\mathbf{z}_t|\mathbf{x}_t)p(\mathbf{x}_t|\mathbf{z}_{t-1})$ if we know that the sensor variance does not change with time.

It is often difficult to understand exactly what this all means just by looking at Equations (11) and (12). A more qualitative description is often more helpful for understanding what is happening.

- $p(\mathbf{z}_t|\mathbf{x}_t)$ is the perceptual model. It is the probability of seeing a particular observation given that the system is in some state $\mathbf{x}_t$ at time t. Using the previous example, it describes the likelihood of making observation z given that the cart is at location x. The distribution is a property of a given sensor.

- $p(\mathbf{x}_t|\mathbf{x}_{t-1})$ is sometimes referred to as the action model and describes the system dynamics. This is why it is essential to formulate a state space model of the system under examination, as this is what is used to formulate the distribution.

At this point it is worth iterating that Bayes filters only provides a probabilistic framework for recursive state estimation and that implementing a Bayes filter required specifying the perceptual model $p(\mathbf{z}_t|\mathbf{x}_t)$, the system dynamics $p(\mathbf{x}_t|\mathbf{x}_{t+1})$ and the representation of the Belief $p(\mathbf{x}_t|\mathbf{z}_t)$.

2.3 The Kalman Implementation

The Kalman filter implementation utilises the specified state space system model described in Section 2.1. It is assumed that process noise is added to the state update model thus:

$$\mathbf{x}_{t+1} = \mathbf{F}\mathbf{x}_t + \mathbf{B}\mathbf{u}_t + \mathbf{w}_t \tag{13}$$

Where $\mathbf{w}_t$ is zero mean, normally-distributed noise with convariance $\mathbf{Q}$, i.e., $\mathbf{w}_t \sim N(0, \mathbf{Q})$.

Measurements are made by observing system state thus,

$$\mathbf{z}_t = \mathbf{H}\mathbf{x}_t + \mathbf{v}_t \tag{14}$$

where $\mathbf{v}_t \sim N(0, \mathbf{R})$ and $\mathbf{R}$ is the noise convariance.

2.3.1 Prediciton

The next state is predicted blindly according to supplied state model,

$$\mathbf{x}_{t|t-1} = \mathbf{F}\mathbf{x}_{t-1|t-1} + \mathbf{B}\mathbf{u}_t \tag{15}$$

and the estimate covariance updated according to,

$$\mathbf{P}_{t|t-1} = \mathbf{F}\mathbf{P}_{\mathbf{t-1|t-1}}\mathbf{F}^{\mathbf{T}} + \mathbf{Q} \tag{16}$$

2.3.2 Updating

$\tilde{\mathbf{y}}_k = \mathbf{z}_k - \mathbf{H}_k\hat{\mathbf{x}}_{k|k-1}$
$\mathbf{S}_k = \mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^{\mathrm{T}} + \mathbf{R}_k$
$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}_k^{\mathrm{T}}\mathbf{S}_k^{-1}$
$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k\tilde{\mathbf{y}}_k$
$\mathbf{P}_{k|k} = (I - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k|k-1}$

2.4 The Particle Filter Implementation

A very commonly used algorithm used to implement the particle filter is Sequential Importance Resampling (SIR). The SIR is the original particle filter algorithm which approximates the filtering distribution

$$p(x_k|y_0, \ldots, y_k) \tag{17}$$

by a weighted set of P particles

$$\{(w_k^{(L)}, x_k^{(L)}) \ : \ L \in \{1, \ldots, P\}\}. \tag{18}$$

The *importance weights* $w_k^{(L)}$ are approximations to the relative posterior probabilities (or densities) of the particles such that $\sum_{L=1}^{P} w_k^{(L)} = 1$. The values of the *importance weights* is evaluated to give $w_k^{(L)}$. As in importance sampling, the expectation of a function $f(\cdot)$ can be approximated as a weighted average

$$\int f(x_k)p(x_k|y_0, \ldots, y_k)dx_k \approx \sum_{L=1}^{P} w^{(L)} f(x_k^{(L)}). \tag{19}$$

Because the $\int f(x_k)p(x_k|y_0, \ldots, y_k)dx_k$ is very hard to find, we use the approximation as a replacement.

2.4.1 Implementation

The implementation of the algorithm is rather simple (compared to other particle filtering algorithms) and it uses composition and rejection. To generate a single sample x at k from $p_{x_k|y_{1:k}}(x|y_{1:k})$

1. Set $p = 1$
2. Uniformly generate L from $\{1, ..., P\}$
3. Generate a test $\hat{x}$ from its distribution $p_{x_k|x_{k-1}}(x|x_{k-1|k-1}^{(L)})$
4. Generate the probability of $\hat{y}$ using $\hat{x}$ from $p_{y|x}(y_k|\hat{x})$ where y_k is the measured value

5. Generate another uniform u from $[0, m_k]$
6. Compare u and $p(\hat{y})$
7. (a) If u is larger then repeat from step 2

   (b) If u is smaller then save $\hat{x}$ as $x_{k|k}^{(p)}$ and increment p
8. If p ¿ P then quit

The goal is to generate P *particles* at k using only the particles from $k-1$. This requires that a Markov equation can be written (and computed) to generate a x_k based only upon x_{k-1}. This algorithm uses composition of the P particles from $k-1$ to generate a particle at k and repeats (steps 2-6) until P particles are generated at k.

This can be more easily visualized if x is viewed as a two-dimensional array. One dimension is k and the other dimensions is the particle number. For example, $x(k, L)$ would be the L^{th} particle at k and can also be written $x_k^{(L)}$ (as done above in the algorithm). Step 3 generates a "potential" x_k based on a randomly chosen particle ($x_{k-1}^{(L)}$) at time $k-1$ and rejects or accepts it in step 6. In other words, the x_k values are generated using the previously generated x_{k-1}.

3 Application One: Image Tracking

Image tracking is a common application for a Bayes filter. A certain aspect of, or object in an image is to be tracked, and the goal is to use a state model as well as sensor measurements, to estimate its state. In the case of image tracking the observations are usually generated by some sort of image processing operation that identifies objects in an image. The noise inherent in these measurements comes from the imperfect nature of this processing, i.e., we may pick up other objects beside what we are tracking and identify these as our object.

Two example video sequences were chosen to implement object tracking. Both consist of a single object (a ball) falling past a uniform background, hitting the ground and bouncing a certain number of times before falling out of view. Video sequences of this form are the easiest to track because:

- The object (ball) is visible (unobscured) for the entire duration.
- There is only a single moving object.
- There are a number of frames at the start of the sequence in which the ball is not visible.

The last point is important as it allows an accurate representation of the background to be stored by averaging the first n frames in which the object is not present. The measurements are then obtained by subtracting the background from this image.

3.1 Observations

The Bayes filter estimates state using (noisy) sensor measurements. In our example, the "sensor" is an image processing routine that attempts to identify the ball. No feedback is used in this operation so the sensor may end up simply identifying the largest, or most circular objects in some cases. Detection errors like these are what make up measurement noise.

The observation algorithm used to detect the ball in each frame is shown in pseudo-code in Listing 1.

Listing 1: Implementation of observation routine.

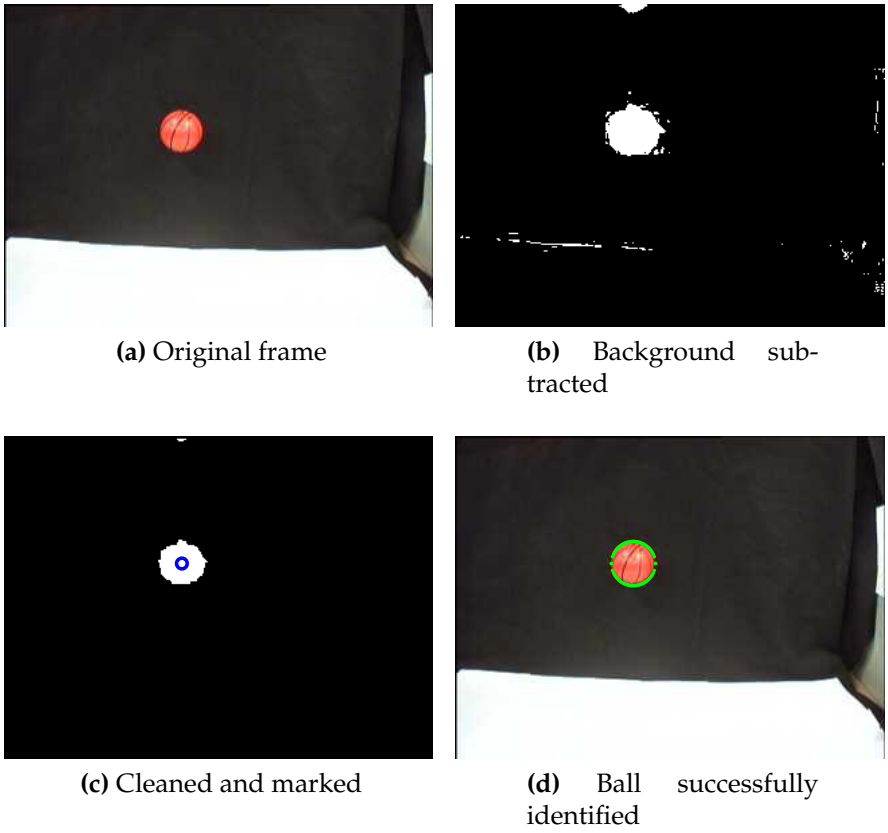
```
% Subtract background from frame
frame = frame - background;

% Background pixels = 0, Object pixels = 1
for pixel in frame:
    if pixel != 0 then pixel = 1;

% Clean up the shape of all objects.
frame = erode(frame);

our_object = largest_object(frame)

if object is None then exit

return object
```

The measurement routine is in essence a background-subtraction, followed by some inbuilt Matlab functions for identifying similar regions[6]. The result of a typical measurement for a particular frame is shown in Figure 2.

Figure 5a shows the original frame. The ball's motion is in a downward direction. Since we have already found the background for all frames, we can perform a subtraction to obtain the modified frame shown in Figure 5b. Note that after the background subtraction a simple routine was used to convert the image to black and white. It is then cleaned up using inbuilt Matlab image processing functions, and this same family of functions is also used to identify the contiguous pixels which form the ball. As a result of the eroded image's maintaining a circular geometry, we simply measure the ball's centre as the centre of the white area. Were the frame to be noisier, we may lose the circular outline and the measured centre may drift to a more incorrect value.

For illustration, the outline of the ball as measured is shown in Figure 2d. What appears to be a continuous green line is in fact a series of green points equidistant from the measured centre using a radial value returned from Matlab. Note again that even though the outline appears very accurate in the chosen frame, it is possible for it to be slightly, or completely off from the ball depending on the noise present in the image.

[6]TODO quick explanation of the functions bwmorph, bwlabel, regionprops



(a) Original frame

(b) Background subtracted

(c) Cleaned and marked

(d) Ball successfully identified

Figure 2: Sensor observation for a single frame.

It is worth pointing out at this stage that such an accurate sensor eliminates the need for a Bayes filter, as one can simply track the very accurate sensor measurements. However, only a small amount of additional noise, or the ball's being obstructed for several frames, would quickly necessitate a Bayes filter. To be able to implement and test Bayesian estimation, an additional ability was built into the measurement function to intentionally inject noise.

The entire measurement routine was implemented as a Matlab function which returned the coordinates (pixel locations) of the measured centre of the ball in a given frame. This value is then used by the state space model of the system.

3.2 State Space Model

State space models, while important, are often extremely difficult to formulate for real systems. However, because of the accurate sensor measurements we can get away with an overly simplistic model, at least initially. The state space that was chosen is shown in Equation (20). It consists of the 2D position of the centre of the ball and the velocity in each direction.

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix} \tag{20}$$

Where x and y are the x- and y-coordinates respectively, and the velocity in each direction is represented by their derivatives $\dot{x}$ and $\dot{y}$. The positional units are with respect the the size of the image, and the velocities in pixels per second. Having higher-order aspects of the system such acceleration in the state space serve little purpose in this example.

3.2.1 State Update Equation

The simplest non-trivial model that can be applied to this system is to model the affect of gravity. Having a dynamical model which *only* takes gravity into account will of course, break down very quickly and is not realistic. However, it serves as a simple test.

At each time step, the positions are updated as shown below,

$$x_{t+1} = x_t + \dot{x}_t \tag{21}$$
$$y_{t+1} = y_t + \dot{y}_t \tag{22}$$

That is, at each time step the positions increment by the current velocities. The velocities are themselves updated according [7] to,

$$\dot{x}_{t+1} = 0 \tag{23}$$
$$\dot{y}_{t+1} = \dot{y}_t + g \tag{24}$$

That is, the velocity in the y-direction is incremented by the value of gravity we choose and the velocity in the x direction remains unchanged. It is important to keep in mind that this is simply a model we are building to represent the behaviour of the system; even though there is almost certainly some sideways movement of the ball, it is probably suffice to model it as zero.

Using these equations we can begin to generate the state-space model for this example.

[7] The reader may notice a perceived discrepancy in the units at this point. It was stated that velocities are in pixels/sec, but the time steps occur much faster than this. This is precisely the reason for the modifications to the state matrices in Section 2.1.1.

$$\mathbf{x_{t+1}} = \begin{bmatrix} x_{t+1} \\ y_{t+1} \\ \dot{x}_{t+1} \\ \dot{y}_{t+1} \end{bmatrix} \tag{25}$$

$$= \begin{bmatrix} x_t + \dot{x}_t \\ y_t + \dot{y}_t \\ 0 \\ g \end{bmatrix} \tag{26}$$

$$= \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{x}_t + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} g \tag{27}$$

$$= \mathbf{A}\mathbf{x}_t + \mathbf{B}\mathbf{u}_t \tag{28}$$

Where (25) is the usual state-space form.

3.3 Filtering Steps and Noise

The Kalman filter implementation of the Bayes filter was used for testing. Under the conditions of the example, that is, a linear model and linear noise, the Kalman and particle filter methods are expected to perform identically. As stated in Section 2.2.1, the Bayes filter has two distinct steps: prediction and update. The implementation of the two steps differs for the Kalman and particle filter implementations, but are described generally below.

In the prediction step we use our simple state model to predict the state of the system in the next time step. Since we are using such a simplistic model, where the ball simply falls under a form of "pixel gravity", the predict step is not difficult to understand. It simply says that the ball will have moved downward by g pixels at each time step. This model performs well when the ball does in fact have a downward velocity, but as expected, breaks down when the ball hits the floor or has an upward velocity.

The update step is where we use the latest sensor measurement to correct our prediction. It incorporates the predicted state model and the observation. Naturally, if these values were identical (as they are when the ball is falling) then there is no change in the update step.

In both of the two steps the noise was assumed to be Gaussian. During testing, we simulated this noise by adding randomly distributed random numbers to both the sensor measurements and the state update equations.

3.4 Results

The results for this example were not groundbreaking. The Kalman filter implementation was first tested. As expected, the filter tracked the object very

closely while the ball had a downward trajectory, and even when it did not, the filter still managed to stay very close. We can explain this in terms of the (lack of) measurement noise; even when the state model broke down, (i.e., when the ball hit the ground), the measurements were so accurate that close tracking was still achieved.

3.4.1 Simulated Noise

To try and create a more realistic situation, we intentionally injected noise into the sensor measurements and process updates. Adding additional noise to the measurement step did not affect the tracking ability to a large extent. On the other hand, adding only a small amount of noise to the state update step caused tracking to become much worse. This is again entirely expected and came be easily explained by considering our simple model. Adding noise to the update model causes two detrimental effects, (a) the x position now changes due to noise and (b) the effect of "falling" under pixel gravity may now be exaggerated if a high or low noise value is added to the model while the ball is in free-fall or rebound respectively.

Even with the added noise and the inaccurate model, close tracking was still maintained. This can be put down to the simplicity of the actual system under examination. A ball bouncing under gravity has only a single degree of freedom and although the model is incorrect for half of duration of the video (while the ball is rebounding), the measurements are still so clear that we can track the ball accurately.

In order to prove that we were not simply following the measurements and disregarding the state update, a final test was constructed where the measurement noise was given an extremely high value ($\sigma = 10$) and the process noise set to zero. When the simulation was run, tracking was started as soon as the ball entered the frame and was maintained while the ball was moving downward, but was lost entirely when the ball hit the ground. This is an encouraging result and shows two things:

- Recursive Bayesian estimate relies on both the state model and the observations
- If either the measurement noise is high, or the model inaccurate, then the other must be even more accurate to compensate.

3.4.2 Obstructions

A second test of the filtering algorithm is to add an obstruction to the image. This means that the object (ball) that we are trying to track "disappears" for one or more frames before coming back into view. Unfortunately, the videos had already been taken by this stage and time did not permit a reshoot with an obstruction added. We were able to simulate this however, by simply dropping measurements for one or more frames.

Implementing this simulated obstruction was simple, and required only a small change to code. As a consequence of our overly simplistic model, the tracking was only expected to work when moving past the "obstruction" in the downward direction. This is indeed what was observed but it was interesting to note just how well tracking was maintained when moving thus. As soon as the ball disappeared, i.e., no measurement was available, the state model of the system took over and tracked the ball down though the obstruction. This highlights the strength of the Bayes filter, but also the importance of an accurate model.

3.5 Particle Filter Implementation

Unfortunately the algorithm developed for the particle filter method did not work. It is still not known why but we can make some assumptions about what would have happened:

- The measurements would have been identical as this was independent of the filter implementation.
- The tracking would have been as good or better then the Kalman implementation due to the particle filter's ability to deal with both Gaussian and non-Gaussian noise.

4 Application Two: Filtering

We developed a second example in order to further test the properties of Bayes filters. The image tracking example, while realistic, did not allow control of the system, as it was simply a collection of frames that could not be changed. Our second example involved developing a system that we completely specify, i.e., we have exact knowledge of its dynamical model because we created it.

The system used in the example is the spring-mass-damper described in Section 2.1. The state space derivation is already given. We can now analyse this system in a way we could not with the image tracking example. The goal of this application example can be stated thus: given a state-space model that is very accurate (by definition it is perfectly accurate), and a random input into this system, how much sensor noise can we tolerate before tracking is lost?

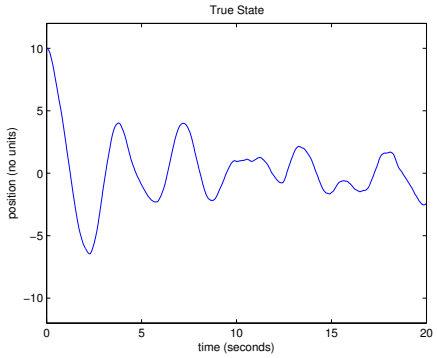
4.1 Noise

Measurement noise in the system is by definition zero, as we can observe the state without error. To simulate noise in the measurements, a normally-distributed random number was added to the measurement at each time step. Manually adding this error also means that we have knowledge of the noise variance (as we have specified it).

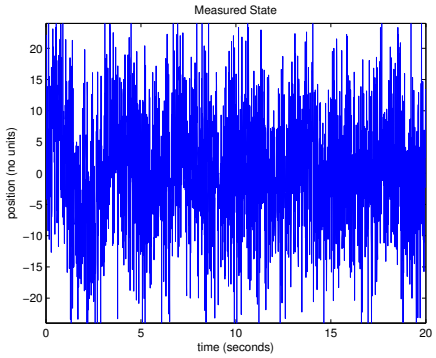
4.2 Results

The Bayes filter was implemented as a Kalman filter because of the difficulties obtaining results using the particle filter implementation.

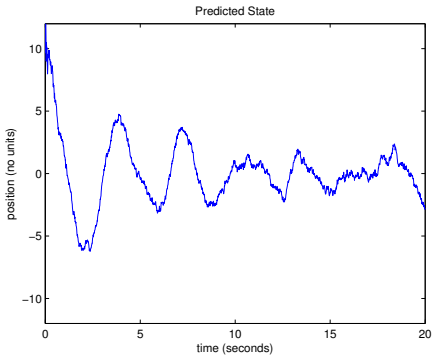
Figure 3 shows the results of the Bayes filter with extremely high measurement noise ($\sigma = 10$). The true state is shown in Figure 3a. It is a damped system starting at position $p = 10$ with an additional random input, which is *unknown to the sensor*. Figure 3b shows the measured state. Because of the amount of noise it is almost impossible to make out the original signal[8]. Finally, Figure 3c shows the output of the Bayes filter.



(a) True state



(b) Measured state



(c) Estimated state

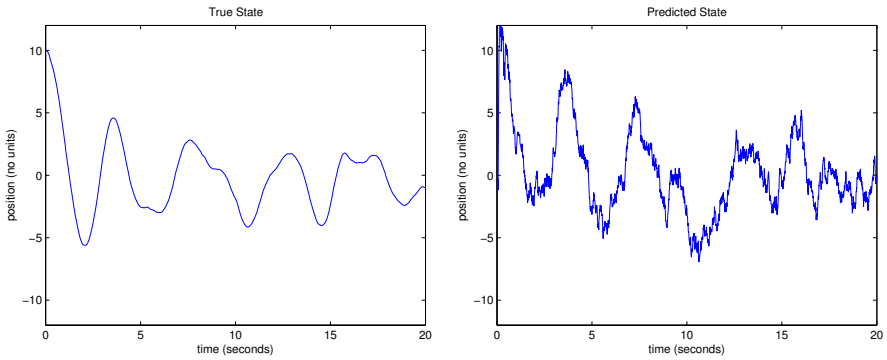
Figure 3: Tracking true state from extremely noisy ($\sigma = 10$) sensor measurements.

The output maintains remarkably close tracking of the input, or stated in a way more fitting for the example: the noise has been almost completely filtered from the measured signal. As remarkable as this example be, it is contrived and it is important to note the following:

- Measurement noise is randomly distributed and we are aware of this fact.
- We know the measurement noise variance *exactly*.
- We know the process noise variance *exactly*.

[8]This is typical of signals obtained from research into diabetes patients in which the blood-glucose level is to be tracked.

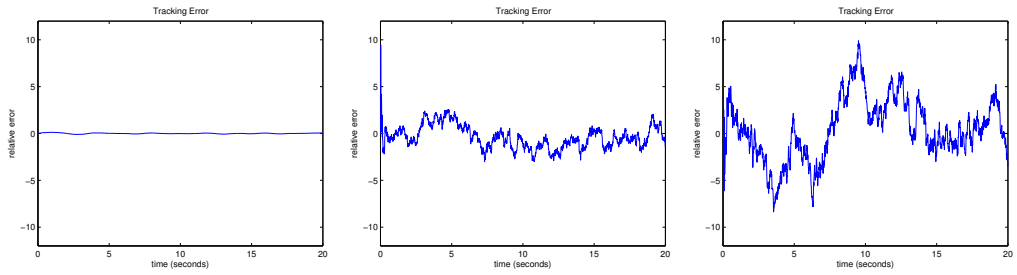
If either of these change, then tracking is expected to become less accurate, or at worst, lost completely. To test this hypothesis, the true noise variance was changed to ($\sigma = 5$) while our belief of the variance was set at ($\sigma = 2$), i.e., the sensors were poorer than the Bayes filter assumes. The result is shown in Figure 4. It is visually obvious that the state is not being tracked accurately. This highlights the importance of characterising the sensors so that the noise variance is known.



(a) True state **(b)** Estimated state

Figure 4: Output when we have characterised the sensor noise incorrectly.

The relative error between the true and estimated states for different values of noise variance error is shown in Figure 5. As expected, the more incorrect we are about the sensor noise variance, the worse tracking becomes.



(a) Error = 0 **(b)** Error = 10 **(c)** Error = 20

Figure 5: Changing measurement noise variance while maintaining a fixed value of assumed measurement noise variance (unity).

4.2.1 Particle Filter Implementation

Even though the particle filter implementation did not work, we attempted to create a non-linear state model, expecting the Kalman filter to fail because of its inability to handle non-linearity in the system model.

The state model was changed to a simple non-linear system modelling a pendulum. The derivation is omitted and only the final state-space equations are shown.

$$\mathbf{x}(t+1) = \begin{pmatrix} x_1(t+1) \\ x_2(t+1) \end{pmatrix} = \mathbf{f}(t, \mathbf{x}(t)) = \begin{pmatrix} x_2(t) \\ \alpha \sin x_1(t) - \beta x_2(t) \end{pmatrix} \tag{29}$$

The same procedure was used for this new non-linear system. Unexpectedly however, the Kalman implementation of the Bayes filter continued to work correctly. It is unknown why this happened, but one assumption to be made is that if the Kalman implementation functioned correctly, then the particle filter implementation would certainly function too.

5 Conclusions

The results observed for both application examples are in line with what was expected, with the exception of the Kalman filter implementation's not failing when presented with a non-linear system model.

The image tracking application was successful despite the overly simplistic model used. However, it is important not to draw unwarranted conclusions from this example and assume that an inaccurate system model will not affect the Bayes filter's ability to track an image. On the contrary, the results show that an inaccurate model is more detrimental than high measurement noise. If the first application example was even slightly more complex, e.g., giving the ball a second degree of freedom, then our overly simplistic model that only takes gravity into account would certainly not work as intended.

The second filtering application example showed that we are able to obtain very accurate tracking (filtering) even in the presence of atypically high sensor noise, but that this is only because we can characterize the noise accurately. We conclude that recursive Bayesian estimation is an extremely powerful tool, but that its power depends on

- The accuracy of the sensor model.
- The accuracy of the dynamical model.

i.e., a Bayes filter is only as accurate as the model and the characterization of the noise.

If either of these are below a certain threshold (dependent on the situation) then the other must be very accurate to compensate or tracking will be lost entirely.

5.1 Particle Filter Implementation

We conclude that the particle filter implementation of the Bayes filter is difficult to implement at best, especially in Matlab. We come to this conclusion after observing the following:

- There is very little reference material on the internet about particle filters in software, the few that do exist are implemented in C.

- Our Matlab implementation did not appear to work.
- Many papers that mention particle filtering, in fact write the implementation using an alternate (usually Kalman) implementation.

It should be pointed out that we are unsure whether our particle filter implementation was close to, or far from functioning. Given more time, it would be desirable to investigate this further.

Even without a working implementation, we have still built the framework for a particle filter implementation. The system, dynamical and noise models are all complete, and with further work, the particle filter predict and update routines could easily be inserted into the code.

From all evidence we conclude that for all examples presented in this report, the particle filter implementation would produce results equal to or better than our test Kalman implementation.

One point to note about the application described in Section **??** is that if were we to employ a more accurate model for the balls motion, e.g., a model that takes into account the impulse from hitting the ground and a sudden change in velocity, then it would likely become a very non-linear system which may eventually cause the Kalman implementation may break down, and a particle filter become necessary. However, we have also observed that the Kalman filter can handle non-linearities so the particular threshold is unknown.

It appears that for most applications, the Kalman filter performs satisfactorily, even in the presence of slight non-linearity in the system model. However, we assume that there exist models of sufficient complexity that will benefit from the particle filter method.

References

[1] Benjamin Washington-Yule, How to LaTex, 1999, Harper Collins