Learning to Track a Random-Walk Target Motion Using Markov Chain Monte Carlo*

Stephen W. Thomas
Department of Computer Science
University of Arizona
Tucson, AZ, USA
sthomas@cs.arizona.edu

ABSTRACT

In this paper we present a generative model to track a random-walk target trajectory. We use Markov chain Monte Carlo (MCMC) sampling techniques to infer the most likely parameters for our model and we propose a data-driven technique to compute the proposal distribution.

Our generative model consists of a set of legs of motion that independently describe each sub-interval of the target trajectory. This approach allows a robust method for describing maneuvering ground-based targets that follow a trajectory that cannot be modeled by standard linear and Gaussian assumptions.

We evaluate our approach against a large number of sample trajectories obtained from a military tank simulation. We compare the performance of our model against the Kalman filter and show that our model achieves a better estimation over 75% of the time and is more tolerant of large and non-Gaussian sources of measurement noise.

Our results suggest that a generative model coupled with MCMC sampling provides an effective technique for tracking ground-based targets.

1. INTRODUCTION

In this paper we present a new method to track maneuvering ground targets that reduces the overall system uncertainty when compared to existing methods. Tracking ground targets is an important topic for both military and civilian applications (e.g., battlefield surveillance, target acquisition and engagement, traffic surveillance, computer vision, and signal processing). Reducing the uncertainty in this process can lead to increased situational awareness, increased effectiveness of weapon systems, and decreased collateral damage on the military side; and better traffic control and smarter vehicle knowledge systems on the civilian side.

The challenge in target tracking lies in dealing with noisy, sparse measurements of the target and trying to estimate the truth state of that target. When the trajectory of the target is non-linear or non-Gaussian, or when the measurement noise is non-linear or non-Gaussian, traditional tracking techniques are challenged. Further, although lots of work has been done to track air- and sea-based targets, less effort has been spent studying maneuvering ground-based targets (e.g, tanks) [10].

In this paper we consider the *random-walk* target motion model [5], which is a good model for describing ground-based maneuvering targets [2, 14]. The random-walk model differs from others in that it exhibits non-linear and non-Gaussian motion and can not accurately be described by any linear-dynamic system, as we will show. We obtain sample trajectories from a military tank simulation [14].

To accurately track a random-walk target trajectory in the face of noisy measurements, we propose a model that stochastically generates a trajectory to closely match the measurements. Our model generates a set of *legs* to comprise the trajectory, where each leg has a set of associated parameters to describe its behavior. Using Bayesian statistical inference and Markov chain Monte Carlo (MCMC) sampling, we can fit our model to the set of measurements and we can then describe the target trajectory and make predictions about its future behavior.

Using a leg-based model allows us to accurately estimate past and future locations of a ground-based target as well as learn the general behavior of other ground-based targets. Further, using MCMC sampling allows us to achieve better estimations of the target state when compared to traditional techniques, such as analytical solutions or filtering processes [8].

1.1 Target Tracking

The goal of *target tracking* is to estimate the dynamic state of an observed target. In the general setup, a set of noisy observations are generated from some sensing device and the task is to estimate the trajectory (past, present, and/or future) of that target. Depending on the type and accuracy of the sensing device, and the previous knowledge of the target motion, many different approaches can be taken to track the target. These include linear estimation processes,

*Produced for CSc 645 (Statistical modeling and inference), Spring 2009 (Dr. Kobus Barnard).

particle filtering, and various optimization techniques.

Common to all approaches is the idea that the motion of the target should be described by some mathematical *model* with sufficient accuracy [10]. These models usually take the form of a *state-space* model, although many others have also been proposed. The goal of all of the models, however, is to describe how the target state evolves over time, and thus how to *track* a target.

1.2 Previous Work

Li and Jilkov [10] provide an excellent survey of models that have been previously and are currently used in tracking maneuvering targets. They describe a diverse set of models for air- and sea-based targets, including the white-noise acceleration model, the Wiener-process acceleration model, the polynomial model, the semi-Markov jump process model, and various versions of curvilinear models. However, they admit that there currently exists a lack of models to accurately describe motions of ground based targets, such as the random-walk model that we are focused on here.

More simplistic methods, such as linear least squares and its variants, have been used in previous studies dealing with ground-based targets [14]. While this technique is useful when very few measurements are available, it lacks any ability to accurately model the details of the trajectory and is limited to very simple, linear trajectory paths.

Researchers have also employed the Kalman filter [7] and its variants to estimate the motion of a ground based target [8]. The *Kalman filter* is a stochastic estimation process that iteratively predicts and corrects estimations of a linear state-space model with Gaussian measurement noise. However, in the presence of non-linear systems or non-Gaussian measurement noise, the effectiveness of the Kalman filter decreases.

To the best of our knowledge, the definition of the random-walk motion model for target trajectories first appeared in a military technical report by Butterly [2]. In this work the author used this motion model and derived a series of closed-form probabilities to describe some future location of a target, given truth information about a target's trajectory for a specified amount of time. However, these closed-form solutions are of little use when we do not have such information or in the presence of noisy measurements.

The random-walk model has also been considered in other domains. It has been used in a technique for social group-based search algorithms amongst multiple robots [6]. A Kalman filter has also been used to approximate a random-walk model in the domain of stock pricing [13].

1.3 Contributions

The main contribution of this work is the development and evaluation of a generative model for ground-based target trajectories. We show that such an approach can provide good to excellent estimation of the truth target states, even in the presence of large, non-Gaussian observation noise, and is thus an effective approach for problems dealing with such target trajectories.

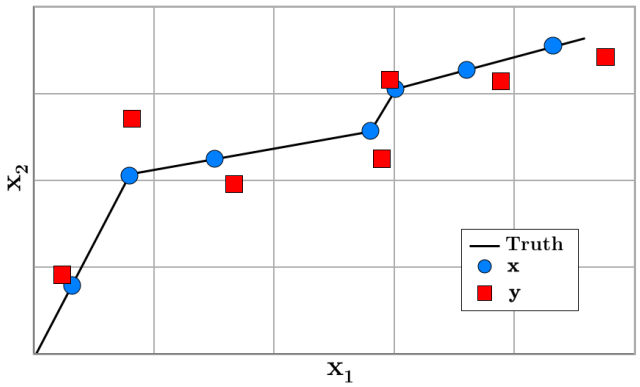

Figure 1: An example ground-based target trajectory in a 2-D plane. The blue dots represent the truth state vectors x while the red squares represent the estimated state measurements y.

We compare our approach with a variant of the Kalman filter and show that our method outperforms the Kalman filter 76% of the time and is more robust against increasing measurement noise variance, increasing measurement interval times (i.e., fewer measurements), and non-Gaussian measurement noise.

2. PROBLEM STATEMENT

We assume we are given a set of noisy measurements that collectively attempt to describe the trajectory of a given target and our goal is to use these measurements to estimate the actual (or *truth*) trajectory for that target. Our approach in this paper is to develop a generative model that describes the behavior of a random-walk target trajectory and use this model to estimate the truth trajectory. This generative model should take as input a parameter vector θ and use that vector to generate a random-walk trajectory[1]. We can then use various statistical techniques to find the θ that produces the target trajectory that best resembles the truth trajectory we desire.

To achieve the above, we need a way to evaluate how well a given parameter vector describes the input noisy measurements. We define a *likelohood* that computes the probability that the noisy measurements were produced by a given parameter vector. This function is described in Section 4.2.

2.1 Data Description

Consistent with existing approaches, we will consider target trajectories defined over a rectangular region $R = [0, L] \times [0, L] \in \mathbb{R}^2$. The truth target state at any time t is described by a vector $\mathbf{x}_t = [x_1, x_2, \dot{x_1}, \dot{x_2}]$ where (x_1, x_2) represents the Cartesian coordinates of the target and $(\dot{x_1}, \dot{x_2})$ is the corresponding velocity. At time t, a measurement $\mathbf{y}_t = \mathbf{C}\mathbf{x}_t + \mathbf{v}_t$ is created by some measurement process, where $\mathbf{C}$ is a projection matrix to extract certain features from the target state

[1]In this paper we use the phrase "θ generates a trajectory", but to be more accurate we could say "θ can be input into our generative model, and the generative model will produce a trajectory".

vector (e.g., location) and $\mathbf{v}_t$ is a possibly non-Gaussian process that models the noise of the measurements.

Figure 1 depicts the general setup for a single target. The continuous target trajectory is shown by the black line. The red squares represent the noisy target state measurements $\mathbf{y}$, while the blue circles represent the truth target location $\mathbf{x}$.

2.2 Assumptions

Often in the tracking literature, two assumptions are made [4].

1. Each state of the target is dependent only on the immediately previous state.
2. Each measurement of the target is independent from all other measurements and all previous states of the target.

Thus, our generative model will assume that each set of leg parameters is independent of all other sets of leg parameters, and each measurement is obtained only from the current state of the target.

3. MODEL DESCRIPTION

We present a generative model, **RW**, that generates a trajectory comprised of K legs of motion, where the i^{th} leg is described by the parameters

$$\text{leg}_i = (\omega_i, \alpha_i, \tau_i).$$

Here, ω_i represents the speed of the target during this leg, α_i represents the heading of the target for this leg, and τ_i represents the duration of this leg. We assume that each of these parameters remains constant for the duration of the leg, and thus the parameters are each scaler values. We denote the beginning location of leg_i as $\text{leg}_i^0 = (\text{leg}_i^{x_0}, \text{leg}_i^{y_0})$ and we denote the ending location as $\text{leg}_i^\tau = (\text{leg}_i^{x_\tau}, \text{leg}_i^{y_\tau})$.

Figure 2 shows a graphical example of a continuous random-walk trajectory broken into a set of four legs. By breaking the trajectory into legs we can accurately and independently describe each subinterval of the trajectory.

We impose the obvious constraints that each leg begins at the same time that the previous leg ended and that the beginning location of each leg is the same as the ending location of the previous leg. That is, we require

$$\text{time}_i = \text{time}_{i-1} + \tau_{i-1}$$

and

$$\text{leg}_i^{x_0} = \text{leg}_{i-1}^{x_\tau} = \text{leg}_{i-1}^{x_0} + \omega_{i-1} * \tau_{i-1} * \cos(\alpha_{i-1}),$$

$$\text{leg}_i^{y_0} = \text{leg}_{i-1}^{y_\tau} = \text{leg}_{i-1}^{y_0} + \omega_{i-1} * \tau_{i-1} * \sin(\alpha_{i-1}).$$

With these constraints **RW** generates trajectories that are consistent with realistic target trajectories and avoids nonsensical trajectories that have large gaps in time and location between successive legs.

We define the parameter vector $\theta \in \Omega$ for our model to be comprised of the number of legs K and the leg parameters $(\omega_i, \alpha_i, \tau_i)$ for $i = 1, \ldots, K$. The size of θ will thus be $1+3K$: one parameter for the number of legs K and three additional parameters per leg.

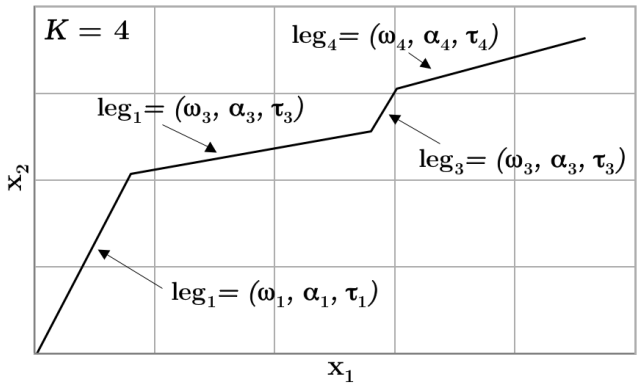

Figure 2: A set of legs compose a random-walk target motion. Here we show an example motion (comprised of four legs) moving in a 2-D plane.

Given an instance of these parameters, our model **RW** will generate a target trajectory. So we can create a vector $\mathbf{z} \in \mathbb{R}^2$ by inputting any θ into our model **RW**.

We summarize our notation in Table 1.

4. INFERENCE

Given a set of measurements, the inference task is to find the model parameters θ that best fit the measurements. We approach this task by using Bayesian statistical techniques. Specifically, we define a probability distribution over the solution space given the measurements and find an optimal assignment to θ. The probability distribution over the solution space Ω is defined as

$$P(\theta|\mathbf{y}) = \frac{L(\mathbf{y}|\theta)\pi(\theta)}{P(\mathbf{y})}. \quad (1)$$

$L(\mathbf{y}|\theta)$ is called the *likelihood* and represents how well the model with parameters θ describes the observed data $\mathbf{y}$. $\pi(\theta)$ is called the *prior distribution* and describes how plausible it is that θ came from the parameter space Ω. $P(\mathbf{y})$ is the *marginal likelihood* of $\mathbf{y}$ and describes how likely the observations are according to our model. Finally, $P(\theta|\mathbf{y})$ is the *posterior distribution* that we are after.

Table 1: Summary of Notation

Symbol	Possible Values	Units	Meaning
K	$\mathbb{Z}^+$	N/A	Number of legs
ω	$\mathbb{R}^+$	m/s	Speed of leg
α	$\mathbb{R} \in [-\pi, \pi]$	radians	Heading of leg
τ	$\mathbb{R}^+$	s	Duration of leg
θ	$\mathbb{Z}^+ \times \mathbb{R}^{3K}$	N/A	Model parameters
Ω	N/A	N/A	Solution space

As $P(\mathbf{y})$ is constant for all θ, we can reduce (1) to

$$P(\theta|\mathbf{y}) \propto P(\mathbf{y}|\theta)P(\theta). \quad (2)$$

We now describe our prior distribution and likelihood, followed by our data-driven technique for computing a proposal distribution for $P(\theta|\mathbf{y})$. We then conclude this section with a description of the sampling technique we use to compute the posterior distribution.

4.1 Prior

We define the prior distribution of the model parameters θ by using intuitions gained from manually inspecting a large set of truth trajectories. First, we note that the duration of each leg is fairly evenly spaced between 5 and 40 seconds, with values outside of this being extremely rare. Thus, we set the prior distribution for leg durations to be uniform in the range of 5 and 40 seconds.

Second, we note that the heading of a leg seemed to be dependent on the heading of the previous leg and never introduced a changed that was more than $\frac{\pi}{4}$ radians in either direction. Also, smaller changes seemed to be more common, and it seemed to be equally likely that the change would be either positive or negative. Thus, we set the prior distribution for the heading of each leg to be dependent on the heading of the previous leg, with the difference between them defined by a normal distribution with $\mu = 0$ and $\sigma = \frac{\pi}{4} \times \frac{1}{3} = \frac{\pi}{12}$ radians.

We take a similar approach for defining the prior distribution over the speed of each leg, and conclude that the speed of each leg can be approximated by a normal distribution with $\mu = 12.5$ units/s and $\sigma = 2.0$ units/s.

Finally, we require at least one leg of motion, so we set the prior distribution over K to 1 if θ has at least one leg and 0 otherwise.

4.2 Likelihood

We define a likelihood to determine how well a given parameter vector θ fits a given set of measurements $\mathbf{y}$. This likelihood uses the generative model **RW** and θ to create a trajectory $\mathbf{z}$. It then scores $\mathbf{z}$ against $\mathbf{y}$ by comparing the Euclidian distance between interpolated locations of the two trajectories at each time step and summing the error. The result will be a score $s \in \mathbb{R}$ with $s \geq 0$, with smaller values indicating a better fit.

4.3 Data-driven Proposal

We present a data-driven method for creating the proposal distribution in a way that is based on actual data and thus reduces the number of iterations needed by the MCMC sampler to converge.

Given the set of target measurements $\mathbf{y}$, we compute a parameter vector θ_0 by the following iterative process. First, we set k to a value in the sample set $\{5, 10, \ldots, 50\}$; this number k is a guess for the number of legs in the trajectory K and hence controls the number of elements in θ_0. We iterate through the entire sample set, so that k takes on every value in the set. For each value of k, we perform the following.

By dividing the total time by k, we obtain a guess for τ_i for $i = 1 \ldots K$. That is, we set

$$\tau_1 = \tau_2 = \ldots = \tau_K = \frac{T}{k}.$$

For each resulting leg, we guess the values for ω, α by selecting the two measurements that most closely correspond in time to the beginning and ending times of this leg. That is, we choose two measurements y_j and y_k where

$$y_j \in \mathbf{y} \; s.t. \; \forall u \neq j, \; \text{distance}(\text{leg}_i^0, y_j) < \text{distance}(\text{leg}_i^0, y_u)$$

and

$$y_k \in \mathbf{y} \; s.t. \; \forall v \neq k, \; \text{distance}(\text{leg}_i^\tau, y_k) < \text{distance}(\text{leg}_i^\tau, y_v)$$

(the function distance(x, y) is defined as the Euclidian distance between x and y). Then we compute each ω_i and α_i by

$$\alpha_i = \tan^{-1}\left(\frac{y_j^y - y_k^y}{y_j^x - y_k^x}\right),$$

$$\omega_i = \frac{\sqrt{(y_j^y - y_k^y)^2 + (y_j^x - y_k^x)^2}}{\tau_j}.$$

This yields a heading and speed estimate for each leg computed from the geometric relationship between the two measurements for the end points of that leg. As mentioned above, we repeat the process for each possible value of k in the sample set, each time saving θ_0. We then choose the k (and thus θ_0) that results in the best fit of the observations according to our likelihood.

Finally, we define distributions around each value of θ_0 as follows. For each leg i, we set the distributions for the three parameters duration, heading, and speed, respectively, to

$$\begin{gathered}\tau_i \sim \mathcal{U}(1, 40), \\ \alpha_i \sim \mathcal{N}(\theta_0, .025), \\ \omega_i \sim \mathcal{N}(\theta_0, 3).\end{gathered}$$

The distributions for each τ_i are based on Wald's intuition that each leg of the trajectory will last between 1 and 40 seconds, while the distributions for each α_i and ω_i are based on empirical experimentation and represent a balance between freedom for the sampler (which requires large variances) and fast convergence times (which requires smaller variances).

4.4 Sampling

The posterior distribution in Equation (2) is a complex distribution that is difficult or impossible to solve with analytical methods. Instead, we use MCMC sampling to search the solution space for a maximum under the posterior. MCMC methods provide a general, efficient way of providing a solution to such large-dimensional problems within a reasonable amount of time.

A thorough description of MCMC sampling can be found elsewhere (e.g., [1, 3, 12]). The basic strategy of a MCMC sampler is to iteratively and randomly generate samples for θ from the solution space Ω. On each new iteration, the sampler modifies the previous sample for θ by utilizing a set of moves (Markov chain) that dictate how to change the sample. (In our case, each move will either change the duration, speed, and/or heading of one or more of the legs.) Depending on how likely the new sample is, it is either accepted or rejected.

5. IMPLEMENTATION

We have implemented our approach using the MCMC toolbox for MATLAB [9]. The toolbox was created to allow a user to generate and analyze a Metropolis-Hastings (MH) MCMC chain using multivariate Gaussian proposal distributions. We have made changes to the toolbox to better meet our purposes, including the addition of non-Gaussian proposal distributions, additional constraints on the generative process, and visualization of the data as the sampler is running.

Our implementation works as follows.

1. We choose some values for T (total time), N (number of simulations), ΔT (timestep between measurements), and the errors associated with the sensor.
2. A truth trajectory is created using the military tank simulation [14], and noisy measurements are taken according to the input options (i.e., measurement variance and measurement interval).
3. Using the measurements, we compute the proposal distribution of the parameter vector as discussed in Section 4.3. We also compute the error associated with this proposal distribution, as specified by our likelihood.
4. We set the options for the sampling algorithm, such as the number of simulations, the burn-in time desired, and the covariance matrix associated with the measurements.
5. We run the MH algorithm by calling the `mcmcrun` method of the toolbox.
6. After the MH algorithm is complete, we can obtain the maximum likelihood estimate of the parameters by computing the mean of the chain produced by the `mcmcrun` method.
7. We compute the error associated with these maximum likelihood estimates, both against the measurements and truth data. We can then create plots and save statistics, and repeat to step 2 with a new random number seed.

In each iteration, we also use a variant of the Kalman filter to compute an estimate for the target trajectory. Here we use the Kalman filter toolbox for MATLAB [11] as a starting point for our implementation. The toolbox uses the EM algorithm to first learn the parameters of the linear dynamic system, and then it filters the measurements using these parameters. With this approach we are able to compare our generative model against the Kalman filter in a general way.

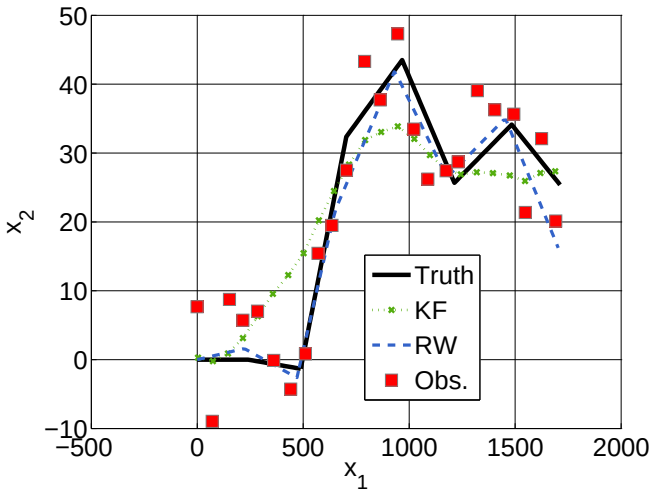

Figure 3: Results of a given run. The thick black line shows the truth target trajectory and the red squares show the noisy measurements. The dashed green line with x's shows the estimated trajectory produced by the Kalman filter while the dashed blue line shows the estimated trajectory produced by our generative model. Here, the measurement variance was 25 units and the measurement interval was 5 seconds.

We have empirically found that an effective burn-in time is a quarter of the total number of iterations N. This allows the sampler a sufficient amount of time to discard initial samples until the chain stabilizes.

6. EXPERIMENTS

To evaluate our approach, we will measure the *sum-of-squares* error between the estimated trajectory and the truth trajectory. This measure captures the difference of locations between the two trajectories over time. For an estimated trajectory $\mathbf{z}$ and a truth trajectory $\mathbf{x}$ we define the measure to be

$$SS = \sum_{t=1}^{T} \sqrt{(z_t^x - x_t^x)^2 + (z_t^y - x_t^y)^2} \ .$$

If we divide SS by the total amount of time T in the scenario we can obtain an estimate of our mean error at each time step; we call this the *mean SS Error / time* measure.

The experiments were conduced on a machine running Mac OS X 10.4.11 with a 2.2 GHz Intel Core 2 Duo processor and 2 GB main memory. The tank simulation, our **RW** model, and the Kalman filter were executed in MATLAB version 7.5.0 (R2007b). Unless indicated otherwise, we ran $N = 5000$ Monte Carlo simulations with 1250 iterations of burn-in, as this seemed to be a good trade-off of convergence versus run-time.

Figure 3 shows an example execution of our experiments. It

shows a comparison of our method versus the Kalman filter for a given target trajectory. Here, our method produced an SS measure of 547.76 compared to the Kalman filter's 1215.84. By noticing how well our method "matches" the form of the target trajectory, we can intuitively see that our method is much more capable of modeling the trajectory a ground-based target.

In each subsection below we explore different values–and their effect on the performance our model–to each the following independent variables.

- **Total time of scenario (T).** This value represents to number of seconds that the target is driving in the plane: the longer the total time, the longer the trajectory path. Default value is 200.
- **Measurement variance (σ^2).** This value represents the accuracy of the target state measurements: the larger the variance, the more error in the measurements and the more obscured the truth trajectory. Default value is 5.
- **Measurement interval (ΔT).** This value represents the number of seconds between each target state measurement. The larger the interval, the fewer reports available. Default value is 5.
- **Type of measurement noise.** This categorical variable captures the type of error distribution associated with the measurement device (e.g., Gaussian or Gamma distribution): different types may challenge assumptions in our models. Default type is Gaussian.
- **Number of Simulations (N).** This value represents the number of Monte Carlo simulations our sampler is run: the more simulations, the better the parameters are fit to the data. Default value is 5000.

6.1 Experiment 1: Fitting the Trajectory

We first investigated how well our method can fit to the truth trajectory with no measurement variance present; the idea was that this would give us an initial indication of the ability of our model to describe ground-based target trajectories. To do this, we used the military tank simulation to create a truth target trajectory and the corresponding noisy observations. We then input these observations to both our **RW** model and the Kalman filter, and both models learn parameters that create an estimated trajectory.

To create an ideal situation, we set the measurement variance to $\sigma^2 = 0$, the measurement interval to $\Delta T = 1$ second, and total time to $T = 100$ seconds. We ran our method for $n = 30$ different random seeds and the results show that both our method and the Kalman filter achieve an average SS measure of less than one unit in every iteration— a nearly perfect match of the target trajectory. Figure 4 shows the results of one such run: we see that both the Kalman filter and our method are able to exactly match the truth trajectory. Clearly, both approaches are capable of correctly modeling a ground-based target in ideal conditions.

6.2 Experiment 2: Measurement Error

We next turned focus to the robustness of our method to the magnitude and type of measurement errors. We first increased the variance from $\sigma^2 = 0$ up to $\sigma^2 = 1600$ in four steps and Figure 5 shows the results. We see that although the trends of the two methods are similar, our method is slightly more robust. This is especially noteworthy since the Kalman filter is optimized for Gaussian noise.

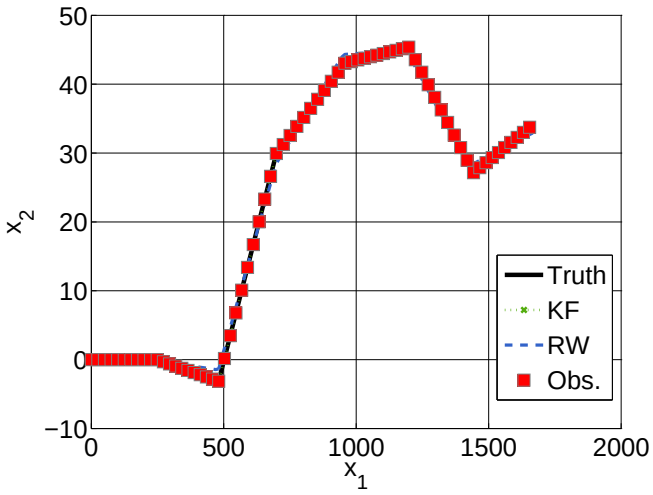

Figure 4: Fitting a trajectory in ideal conditions (i.e., no measurement noise and short measurement interval). We see that both our model and the Kalman filter produce an estimated trajectory that exactly corresponds with the truth trajectory, suggesting that both approaches are capable of correctly modeling a ground-based target in ideal conditions.

We then increased the amount of time between measurement intervals to determine the effect that fewer measurements would take on our approach. Figure 6 shows the results. We see that our method is much more capable of handling sparse reports than Kalman filter. This is because our model generates legs to describe the trajectory as a whole, while the Kalman filter only tries to match the (sparse) data directly.

We then considered measurement processes that followed a non-Gaussian distribution. Specifically, we obtained the measurements using Uniform, Gamma, and Beta distributions (each with a comparable variance to $\sigma^2 = 25$ units) and evaluated our approach. Figure 7 shows the results. We see that in each distribution type, our method achieves a lower error than the Kalman filter.

6.3 Experiment 3: Overall Performance

In this experiment our goal was to determine the average performance of our method in a variety of scenarios: small and large measurement noise, Gaussian and non-Gaussian measurement noise, short and long measurement intervals, and short and long total times. Our strategy was to run 200 repetitions, each time randomly selecting values for the above variables of interest. We choose from the following distributions for each variable.

$$\sigma^2 \sim \mathcal{U}(0, 1600),$$
$$T \sim \mathcal{U}(50, 500),$$
$$\Delta T \sim \mathcal{U}(1, 20).$$

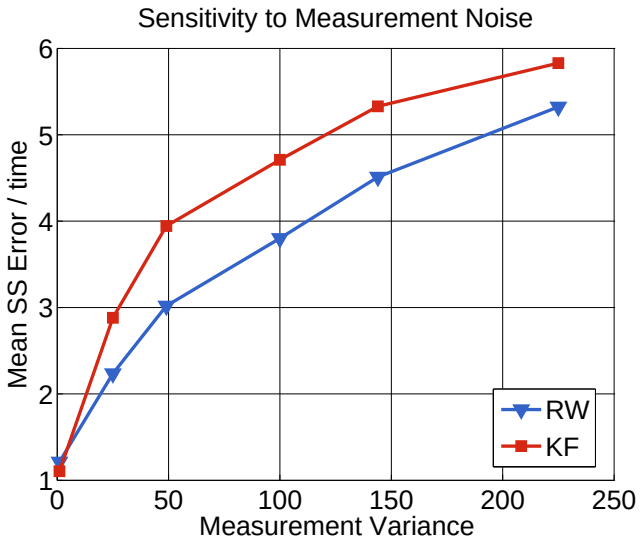

Figure 5: Sensitivity to measurement noise. The x-axis shows increasing variance in the measurement noise and the y-axis shows the mean SS error per time step. The blue line with triangles represents the performance of our RW model while the red line with squares represents the performance of the Kalman filter. These results suggest that our model is more robust to measurement variance than the Kalman filter.

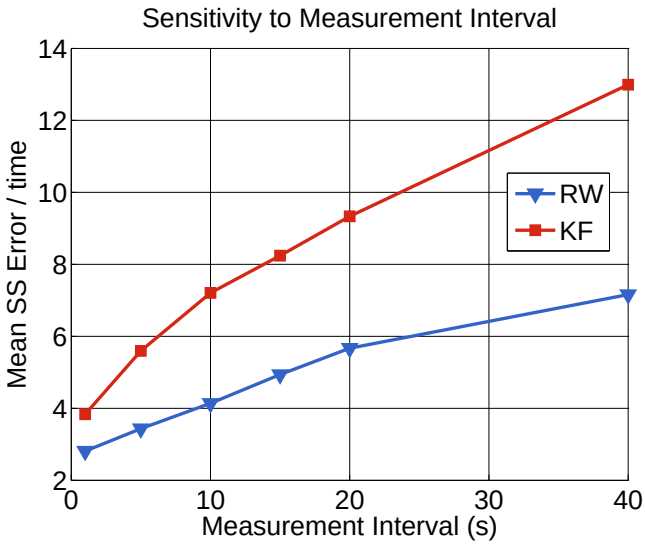

Figure 6: Sensitivity to measurement interval. The x-axis shows number of seconds between each measurement and the y-axis shows the mean SS error per time step. The blue line with triangles represents the performance of our RWmodel while the red line with squares represents the performance of the Kalman filter. Our model outperforms the Kalman filter as the measurement interval grows, suggesting that our model can be used in real-world scenarios where measurements are sparse.

Also, for each repetition we selected the distribution type for the measurement noise from the set {Gaussian, Uniform, Gamma, Beta} with equal probability.

Table 2 summarizes the results. We see that our method is more robust overall and achieves a smaller error than the Kalman filter 76% of the time. Additionally, when our method does achieve a smaller error, the error is on average 4.6 units less per time step than the Kalman filter's error. In contrast, when the Kalman filter achieved a smaller error, it did so by only 0.5 units.

	RW	KF
SS Error	2362.4	3189.7
SS Error / time	11.8	16.0
Win Percentage	76%	24%
Avg. Win Amount	4.6	0.5

Table 2: Comparison of our RW model and the Kalman filter in overall performance. Results show the average performance of each model in 200 randomly-generated scenarios. The "Win Percentage" row shows the percentage of those scenarios in which each model performed better than the other and the "Avg. Win Amount" row shows the average decrease in SS Error / time when that model did win. We see that our model is more robust overall and beats the Kalman filter in a significant fraction of scenarios.

6.4 Experiment 4: Number of Simulations

Here we look at how the SS measure of our model changes as we increase the number of simulations allowed to run. Of course we expect more simulations to result in a better fit, but here we quantify both how much better the fit is and how the execution time of our method is affected.

Figure 8 shows the results. As expected, we see that SS decreases as N increases, but started to level out after about 15,000 simulations. However, execution time is also increased linearly as N increases, so a balanced trade must be made between performance and execution time. The results suggest that choosing the number of Monte Carlo simulations to be $5,000 \leq N \leq 10,000$ might provide such a balance.

6.5 Conclusion

These experiments show that our method is capable of accurately tracking a ground-based target, even with high measurement variances, sparse measurement intervals, various noise distribution types, and scenario times. Further, our method is able to outperform the current most popular tracking method–the Kalman filter– in every type and magnitude of measurement variance.

These results suggest that a generative model, coupled with an MCMC sampling technique for parameter inference, can be used to provide an accurate ground-based target system in the real world.

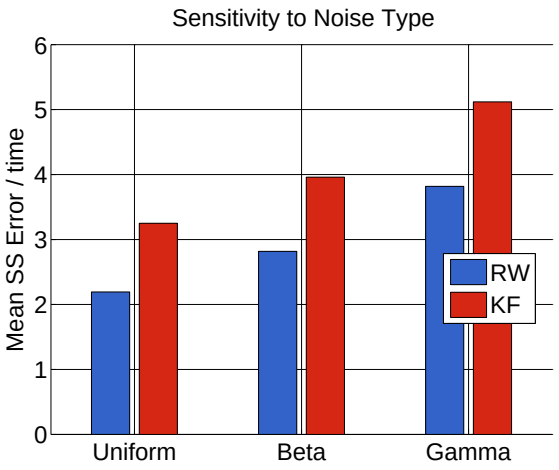

Figure 7: Sensitivity to measurement variance type. We set the measurement error distributions to uniform, beta, and gamma and measure the mean SS error per time step for our approach (blue) and the Kalman filter (red). Our approach has a smaller error in all three cases, suggesting that our approach is robust to non-Gaussian error distributions.

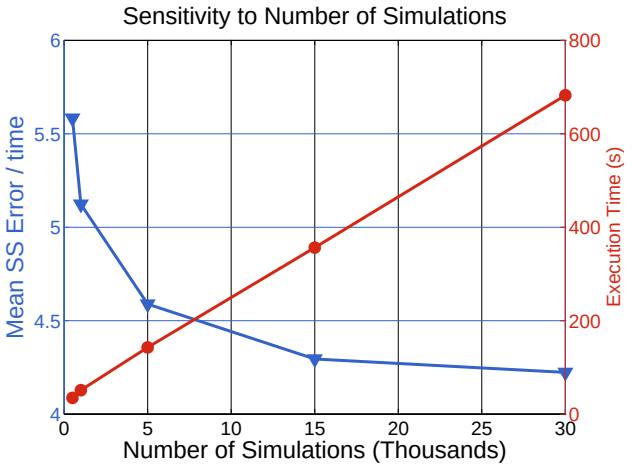

Figure 8: Sensitivity to the number of Monte Carlo simulations. The x-axis shows increasing number of simulations; the left y-axis shows the error achieved; and the right y-axis shows the execution time. The blue line with triangles shows the error and thus corresponds to the left axis while the red line with circles shows the execution time and thus corresponds to the right axis. We see that increasing the number of simulations decreasingly decreases the error and linearly increases the execution time.

7. REFERENCES

[1] C. Andrieu, N. De Freitas, A. Doucet, and M. Jordan. An introduction to MCMC for machine learning. *Machine learning*, 50:5–43, 2003.

[2] P. J. Butterly. Random walk models for target tracking. Technical Report CRC 287, Center for Naval Analysis, September 1975.

[3] P. Fearnhead. MCMC, sufficient statistics and particle filters. *Journal of Computational and Graphical Statistics*, 11(4):848–862, 2002.

[4] D. Forsyth and J. Ponce. *Computer Vision: A modern approach*. Prentice Hall Professional Technical Reference, 2002.

[5] H. Gould and J. Tobochnik. *An introduction to computer simulation methods*. Addison-Wesley New York, 1988.

[6] P. Ioannou and A. Pitsillides. *Modeling and Control of Complex Systems*. CRC Press, 2007.

[7] R. Kalman. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.

[8] R. Karlsson and U. i Linköping. *Particle filtering for positioning and tracking applications*. Department of Electrical Engineering, Linköping University, 2005.

[9] M. Laine. MCMC toolbox for Matlab, June 2008. `http://www.helsinki.fi/~mjlaine/mcmc/`, Viewed April 12, 2009.

[10] X. Li and V. Jilkov. Survey of maneuvering target tracking. Part I: Dynamic models. *IEEE Transactions on Aerospace and Electronic Systems*, 39(4):1333, 2003.

[11] K. Murphy. Kalman filter toolbox for Matlab, June 2004. `http://www.cs.ubc.ca/~murphyk/Software/Kalman/`, Viewed April 12, 2009.

[12] R. Neal and U. of Toronto. Dept. of Computer Science. *Probabilistic inference using Markov chain Monte Carlo methods*. Department of Computer Science, University of Toronto, 1993.

[13] G. Vidyamurthy. *Pairs trading: quantitative methods and analysis*. Wiley, 2004.

[14] J. Wald. The Timeline Analysis Model., 1988.