Methods for Object Tracking With Machine Vision

Zachary Stamler
Portland State University

Follow this and additional works at: https://pdxscholar.library.pdx.edu/open_access_etds

Part of the Electrical and Computer Engineering Commons

Let us know how access to this document benefits you.

Recommended Citation
https://doi.org/10.15760/etd.7507

This Thesis is brought to you for free and open access. It has been accepted for inclusion in Dissertations and Theses by an authorized administrator of PDXScholar. Please contact us if we can make this document more accessible: pdxscholar@pdx.edu.
Abstract

As machine learning and deep learning systems continue to find applications in science and engineering, the problem of providing these systems with high-quality data continues to increase in importance. Many of these systems utilize machine vision as their primary source of information, and in order to maximally leverage their abilities it is important to be able to provide them with high quality, accurate data. Unfortunately, many sets of tracking data extracted from video suffer from the problem of missing frames, which can arise from a multitude of causes depending on the system. These missing frames can result in confusion between object-track links, or even to the loss of tracking altogether.

This work is motivated by the specific problem of repairing flight tracking information in a pre-existing dataset involving large swarms of birds. These swarms can contain very large numbers of small birds, and the tracking information is derived from image segmentation of video, thus confusion between several objects in the data is a common and pervasive issue. The amount of data and the large number of tracked individuals necessitates efficient and trustworthy algorithms for interpolating between endpoints of known tracks.

We perform a literature review that discusses several approaches to track reconstruction and highlights the need for effective and accurate interpolation. We evaluate the effectiveness of a Kalman filter implementation on several tracks from such a dataset as well as synthetic tracks designed to simulate the flocking behavior of another species of interest. We also propose a novel interpolation algorithm, Stochastic Straight-Line Perturbation (SSLP), suitable for use on its own or as a pre-processing step for other algorithms. Using complete tracks, real and synthetic, with synthetically created gaps, we compare the accuracy of the Kalman filter to that of SSLP, as well as that of the Kalman filter operating on the same data pre-processed by SSLP. We find empirically
that not only does SSLP outperform the Kalman filter on gap reconstruction in a number of scenarios, but it can also improve the performance of a Kalman filter, or potentially another tracking algorithm, when used as a pre-processing step. We also discuss performance trade-offs and potential limitations for its application.

For future work, we propose an alternate state-space model that may have some potential to improve upon the performance of the Kalman filter versus the model employed. We also discuss several possible strategies for further improving upon the performance of the SSLP algorithm both in terms of prediction accuracy and computational load, as well as several possible future applications.
Acknowledgements

I’d like to thank my advisor Dr. John Lipor, for consistently pointing me in the right direction, advocating for me in the background, believing in me, and encouraging me to step up to the next level.

I’d like to thank both of the other members of my thesis committee. Dr. James McNames, for his enthusiasm, support, and approachability, and Dr. Martin Siderius, for his continuing support of the Avian Flocks project. And I’d like to thank everyone at Portland State from whom I have had the opportunity to learn from over the past five years.

I’d like to thank my office mates in the NEAR lab for tolerating my angry typing when simulations don’t behave as intended. Special thanks to Alex Higgins for his help with the project, and for always making sure everyone has the computing resources they needed.

I’d like to thank my colleague Andy Goodwin, whose enthusiasm and particular brand of meticulousness always kept me honest in the field. I’d also like to thank my Italian colleagues for opening up a world of opportunity for me, and especially Dr. Stefania Melillo for her tireless work and invaluable advice.

I’d like to thank my parents for their unwavering support, encouragement, and love, and my sister Jess, whose brilliance has always either inspired me or embarrassed me into trying harder.

Finally, I’d like to thank my cat Woody, for constantly forcing me to get up and open the door for him, thereby saving my posture from my office chair.
# Table of Contents

**Abstract**

**Acknowledgements**

**List of Tables**

**List of Figures**

1 Introduction 1
   1.1 Motivation .......................................................... 1
   1.2 Contributions ......................................................... 4
      1.2.1 Kalman Filter .................................................. 4
      1.2.2 Stochastic Straight-Line Perturbation ....................... 4

2 Review of Literature 6

3 The Kalman Filter 9
   3.1 Motivation .......................................................... 10
   3.2 Technical Background .............................................. 11
      3.2.1 The Kalman Recursions ....................................... 12
   3.3 Approach to Process and Measurement Model Design ............. 13
      3.3.1 Process Model .................................................. 14
   3.4 Measurement Model ................................................ 15
   3.5 Bi-Directional Kalman Filter: Approach and Justification .... 18
   3.6 The Kalman Smoothing Filter ..................................... 19

4 Stochastic Straight-Line Perturbation 22
   4.1 Motivation .......................................................... 22
   4.2 Theoretical Underpinnings ........................................ 24
   4.3 SSLP Outlined ....................................................... 26
A.4 Full Derivation of the Gradient of the Stepwise Gamma Matrix for SSLP
List of Tables

5.1 Mean incidences of row length over 300 processed tracks, real dataset. Maximum row length is greater than 100. Statistics from the synthetic set are similar. ................................................................. 47
5.2 Key for labeling scheme of SSLP configurations on plots ............... 48
5.3 Head to head performance of selected SSLP types vs Kalman filters on synthetic data for selected values of $\rho$. Uni-directional Kalman filter used as baseline. ................................................................. 50
5.4 Execution time, in seconds, statistics for a single track, Kalman filter versus SSLP. Baseline mean time for each dataset is the uni-directional Kalman filter. ................................................................. 57
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>A sample sequence of images of a Vaux’s Swift swarm from Chapman Elementary School in Portland, OR.</td>
<td>3</td>
</tr>
<tr>
<td>4.1</td>
<td>From left to right: A row with no orphans, a row with a 2-orphan (red highlight), a row with a 1-orphan (green highlight)</td>
<td>37</td>
</tr>
<tr>
<td>5.1</td>
<td>Real Track #3 (Trajectory: Bottom Right → Top Left), with no samples removed or additional processing.</td>
<td>44</td>
</tr>
<tr>
<td>5.2</td>
<td>Sample synthetic tracks</td>
<td>45</td>
</tr>
<tr>
<td>5.3</td>
<td>Mean and standard deviation of error, Kalman vs SSLP. In terms of predictive error, SSLP outperformed the Kalman filters on both datasets.</td>
<td>49</td>
</tr>
<tr>
<td>5.4</td>
<td>Direct comparison of Kalman Filter to SSLP, real track #3, same gaps. Both algorithms can be seen making good predictions.</td>
<td>51</td>
</tr>
<tr>
<td>5.5</td>
<td>SSLP results, sample synthetic track. Medium curve parameter ($\rho = 8$). Kalman filter predictions diverge wildly from the ground truth when a gap coincides with a curve. SSLP performs well.</td>
<td>53</td>
</tr>
<tr>
<td>5.6</td>
<td>Mean execution times across all runs, both algorithms. The Kalman filter is between 1 and 3 orders of magnitude faster than SSLP, with the problem exacerbated on the synthetic dataset.</td>
<td>54</td>
</tr>
<tr>
<td>5.7</td>
<td>SSLP on a sample synthetic run</td>
<td>56</td>
</tr>
<tr>
<td>5.8</td>
<td>Mean and standard deviation of MSE for both Kalman filter types. The uni-directional filter outperforms the bi-directional filter on the real dataset, but is significantly outperformed on the synthetic set.</td>
<td>58</td>
</tr>
<tr>
<td>5.9</td>
<td>Row length vs error norm, Kalman filter, all types. The filters are mostly insensitive to row length on the real dataset, but very sensitive on the synthetic dataset.</td>
<td>59</td>
</tr>
</tbody>
</table>
5.11 Mean and standard deviation of error, all SSLP types. SSLP modestly
outperformed the Kalman filter on the real dataset, and significantly
outperformed the Kalman filter on the synthetic set. On the synthetic
set there was divergent performance between the different SSLP types.

5.12 Row length vs error norm, SSLP, all types. Each dot represents the
mean results per filter type.

5.14 Combined system on synthetic track samples. SSLP type #10 (“Uni-
form” method), mixed Kalman types. The combined system performed
significantly better than the Kalman filter alone, but did not match the
performance of SSLP alone.

5.15 Mean and standard deviation of error, combined systems. The system
was marginally better than the Kalman filter on the real dataset, and
significantly better on the synthetic set. On both sets, the combined
system was outperformed by SSLP alone.

5.16 Row length vs error measure, combined systems. The combined system
appeared to inherit the worst characteristics of both algorithms on the
real dataset, but performs better on the synthetic set.

A.1 Normalized Blobsize counts found in the real dataset, and their best-fit
approximation.

A.2 Real blobsize sample sequences compared to synthetic blobsize sequences.
Chapter 1

Introduction

1.1 Motivation

Emergent systems, in recent years, have become a major focus of multidisciplinary study. If complex phenomena can arise from the collective behavior of a large number of leaderless, autonomous, individual agents, each acting on a simple rule set, it should be possible to create systems of individuals that incorporate these rules, engineering large-scale systems with previously unattainable performance. These principles, and the technology based on them, has wide-ranging potential for application, from medicine [1, 2, 3] to autonomous traffic control [4, 5] to economics [6, 7] to fluid dynamics [8, 9, 10] and beyond. One way to study the dynamics of emergent systems is by studying one of the many examples of such systems found in nature, namely, a large swarm of birds, colloquially known as a “birdcloud,” as depicted in Fig. [11].

The study of complex biological systems such as birdclouds is, at its core, the study of the instinctive algorithmic rules an individual member of the system, known as an agent, uses in order to navigate that system. In order learn these algorithms, it is necessary to develop some means of tracking the position of each individual agent in the cloud. One common approach is the use of stereoscopic camera systems, which use two or more cameras to capture a series of images of the birdcloud from several perspectives, then use well-established geometric principles to reconstruct the position of each recognized individual within the cloud [11]. These coordinates are
linked temporally across successive frames of the sequence [12, 13] in order to obtain a sequence of positions of the individual in question, which are referred to as tracks.

The 2D coordinates of each individual agent are typically obtained through image segmentation. Thus, difficulties can arise within this approach whenever a birdcloud is sufficiently dense to cause occlusions, which occur when one agent and another occupy the same 2D space within an individual image. Several of these are visible in Fig. 1.1. Since a small bird typically appears as a featureless dark mass of pixels in a captured image of a birdcloud, the segmentation algorithm cannot differentiate between individuals, instead interpreting one larger, relatively formless mass. One bird or the other is thus lost for a short sequence of frames, which we will refer to in this work as a “gap.” In addition to occlusions, gaps can also occur when, for example, an agent flies behind a stationary foreground object present in the frame, or flies out of a camera’s field of view entirely.

The ultimate objective of a 3D tracking and reconstruction algorithm is to generate unbroken tracks representing the path of each specific agent over the entire time frame of the data acquisition. Any time a gap is present, one or more complete tracks are separated into several fragments which must then be re-linked. There has been significant research into methods for linking these fragments by minimizing some cost function or maximizing some likelihood function related to assumptions about the motion patterns of the agents [14], or solving a linear network optimization or set cover problem [15]. This work seeks to augment these approaches by focusing on methods of interpolating likely track position given some candidate pair of track fragments, and therefore some gap with arbitrarily high sample length but with a well-defined start and end position. These interpolated tracks could then be evaluated in some way to determine the best candidate for a match between two track segments. Alternately, they could be employed to partially recover information about agent behavior when out of frame or obstructed by some object.
Figure 1.1: A sample sequence of images of a Vaux’s Swift swarm from Chapman Elementary School in Portland, OR.

The images are highly cropped, and are separated temporally by five frames each. Several occlusions are visible, as well as a foreground object.
1.2 Contributions

1.2.1 Kalman Filter

The Kalman filter is, of course, a well-established tracking algorithm with a number of established applications. It is known to be robust to additive noise and, since it incorporates a two-stage prediction/measurement process in order to generate estimates for each timestep, it is also to some extent robust to missing samples. This work devises an implementation of the Kalman filter, presented in Chapter 3, with the intent of analyzing the extent of its robustness to missing samples, and exploring the different circumstances in which it may succeed or fail. The specific state-space model employed is very similar to the one found in [16], but with some new innovations incorporated into the noise model in order to better leverage existing data. We present a simple method for improving the accuracy of the filter in some data regimes by incorporating backwards-time predictions into those of the standard forward-time filter. We explore the limits, for each version of the filter, of track inference ability in terms of both gap length and amount of curvature present in the track.

1.2.2 Stochastic Straight-Line Perturbation

Stochastic Straight-Line Perturbation (SSLP), presented in Chapter 4, is a novel algorithm that seeks to leverage first- and second-order characteristics of the positional information present in a track by interpolating acceleration values. These inferred values are guaranteed to link a starting positional value to an end value for any gap where these values are defined. They are additionally constrained by derived velocity values present in the data. Additionally, to ensure matching process noise statistics between the inferred data and the existing data, the noise variance of the predicted acceleration values is matched within some tolerance to the variance of the surrounding samples. SSLP is evaluated alongside the Kalman filter implementation according to
the same testing criteria and found to match up favorably in terms of accuracy of prediction in most cases. Additionally, it is tested as a pre-processing front end for the Kalman filter for use in situations where the filter’s noise-rejection characteristics are desirable. The execution times of both algorithms are also compared.
Chapter 2

Review of Literature

A number of studies exist featuring other methods for interpolation of positional data. In the field of fluid dynamics, these methods typically offer computational efficiency by solving trajectories for large systems of particles in fluid flow using global methods. The interpolation schemes discussed in [17] and [18] chiefly employ well-established interpolation techniques such as $n$th-order spline, Hermite, and linear interpolation, and are motivated by the problem of increasing spatiotemporal resolution of otherwise completely observed data in order to better estimate the Lagrangian specifications of the flow field in a given volume. In this sense they are chiefly concerned with wide-sense stationary statistics of the motion of a large number of particles. While some of these interpolation methods may be capable of filling in missing data sequences of arbitrary length, the complete systems that employ them are not well suited for predicting a dynamically changing system with amorphous borders such as a swarm of birds.

The topic of curve interpolation is also prevalent in video restoration. In [19, 20], a method is described of interpolating using “elastica,” also known as planar elastic curves. This concept, first described in the 18th century by Euler, involves a curve of a given length and with known endpoints and tangents at the line endpoints. The optimal curve is defined as the one that minimizes the integral of the curvature squared. In [19], the technique is applied to the problem of resolving the outline shape of an object when occluded by another object when the borders are well-defined in the captured
image. The work in [20] is motivated by “rationalizing” architectural CAD drawings, whereby shapes specified by an architect are adapted to an intended manufacturing process by approximating them in a manner that makes them maximally suited to the process in question. While both of these applications are concerned with spatial interpolation in two or three dimensions, there is no reason the technique could not be applied to spatio-temporal interpolation as with our track gaps. Interpolating using elastica will indeed yield solutions involving smooth turns as one might expect to see describing the trajectory of a bird in flight. However, it assumes that the physical length of the curve is known ahead of the problem formulation, as well as assuming that a given agent in a cloud will always choose the most efficient flight path irrespective of its neighbors. In our work, it is presumed that transient statistical characteristics of surrounding acceleration data may be an indicator of external factors influencing the maneuvering choices of the agent in question.

Several Kalman filter models are described in [21]; one employs a backwards-time information filter in conjunction with the forwards-time Kalman filter for the purpose of integrating future sequence values. This work is focused on audio de-noising rather than data interpolation, thus a key assumption in the application of the methods presented therein is that measurements exist for all sample points but suffer from low signal-to-noise ratio. In our case, some value might need to be assigned to missing data points to employ this method, thereby biasing the results by the choice of this value. In [22], the authors present an Autoregressive Integrated Moving Average (ARIMA) model, in which a summed series of weighted past values, along with error values of past predictions, are used as predictors.

Other approaches to track linking, such as in [14] and [15], describe methods of linking disparate track fragments, sometimes called “tracklets,” without detailing methods for re-creating the junctions between these incomplete tracks after a symbolic link has been established. In [14], the problem of occlusions is resolved through the
use of a track graph, a Bayesian network inference algorithm, and the assumption, not dissimilar to Kirchoff’s Current Law, that the same number of objects must leave a given node as enter it. This method also makes the assumption that each track is accompanied by a feature vector consisting of attributes such as color, shape, position and velocity, several of which will not be available for a birdcloud of homogeneous species as in our work, where every individual has similar appearance and velocities are also likely to be similar between close neighbors. In [15], the track linking method employed by [12], a track graph is also utilized, but track linking is instead treated as either a linear network optimization problem or a set cover problem, which allows individual problems to be solved either locally or globally, respectively. Our work is intended to complement these techniques by either exploring possible methods of recovering missing data once endpoints have been established, or by furnishing potential replacement track segments that could be evaluated by some cost function, particularly in cases where individuals are obscured by some environmental object or by field-of-view limitations.
Chapter 3

The Kalman Filter

As we have described in Chapter 1, a common difficulty when tracking objects, particularly in dense datasets, is that of mutual object occlusions. It may also happen that the object of interest goes temporarily out of the frame of the imaging sensor, or behind some stationary obstacle present in the scene. Additionally, it is possible that whatever means used to define object position, be it an image segmentation algorithm or something else, may fail to find a given object’s position for some number of frames. Broken trajectories present a major problem for tracking algorithms when trying to establish the identity of a given object across all available image frames, a key requirement for any useful analysis of an object or collection of objects [12].

We have several criteria in order for a given method to be considered useful in this application. First, the output data should reflect the ground truth as closely as possible. In other words, we would like to minimize the error between what the algorithm predicts as probable missing sample positions and what those positions actually were in reality. In order to allow for maximum flexibility in application, we would also like our method to be flexible enough to apply in either two or three dimensions with a minimum of adaptation. We would also like to be able to leverage any prior knowledge we may have about the motion characteristics of the objects of interest.

In this chapter and Chapter 4, we explore two possible methods for repairing broken tracks, including proposing and detailing a new algorithm for estimating an
object’s position by interpolation of its missing acceleration values from the available
data surrounding the region. We also explore the possibility of using these methods in
tandem. The methods under discussion for this work are applied in two dimensions,
but each of them can be extended to three dimensions, should this be desirable, to
allow for their application after a 3D reconstruction has been performed.

3.1 Motivation

The Kalman filter has long been a favored method for object tracking [23, 24, 25, 26].
Being based in a state-space representation of an object’s motion, it allows for great
flexibility in defining a process model for prediction and filtering. Provided we can
make certain assumptions about the process model and the noise present in the data,
we are able to get estimates that are theoretically optimal in the least-squares sense
in terms of error [21]. Additionally, as a recursive algorithm, and as one possible to
implement entirely in closed form with matrix and vector math, it is exceptionally
efficient computationally, making it an excellent choice for applications such as ours,
in which large numbers of objects may need to be resolved across a large number
of frames. Further, there is a wealth of material published [21, 22, 27, 28], making
the application of the Kalman filter here a relatively straightforward and time-tested
proposition.

Owing to its inherent recursiveness and associated computational advantage,
the Kalman filter is often applied in real-time tracking applications [29]. Clearly,
however, this is not our use case; we instead hope to make predictions on static data.
Ideally, we could devise some method of incorporating future data into our predictions.
Unfortunately however, unlike in [21] and [30], we are dealing not only with data
corrupted by noise with predictable characteristics, but also with data with arbitrarily
long rows of missing values. As we shall see in Section 3.5 this flexibility of row
lengths complicates our efforts to base our updates on future values.

3.2 Technical Background

We will begin with a brief summary of the Kalman filter recursions as implemented in our model. The actual recursions are found in Section 3.2.1. A Kalman filter takes as its input some information about the object to be tracked, as well as some parameters pertaining to the measurement and process noise believed present in the data. The parameters of this information are determined by the design of the process and noise models, which will be presented in detail in Section 3.3. Let $D$ be the dimensions of the dimension of the system state vector, to be specified as part of the state-space model. Then the output of the filter is an estimate $x_n \in \mathbb{R}^D$ for the state at timestep $n$, as well as an error covariance matrix, $P_n \in \mathbb{R}^{D \times D}$, which can be used to derive a confidence interval. Each timestep of the filter’s operation is divided into a prediction update and a measurement update.

The prediction update, shown in Eq. (3.1), takes $x_{n-1}$, the filter’s output from the previous timestep, and generates a prediction $x^-_n = \alpha F x_{n-1} + w_n$, which can be seen as a prior estimate in the Bayesian sense based on state transition matrix $F \in \mathbb{R}^{D \times D}$ and process noise $w \in \mathbb{R}^D$. The autoregression constant $\alpha$ is tuned with a grid search. This constant serves, in effect, as an infinite impulse response low-pass filter on the cumulative effects of data from past timesteps on the predictions made by the filter, improving the stability of the filter. For this reason, it is a requirement for stability of results that $0 \leq \alpha \leq 1$ to avoid runaway growth. The prediction update also generates $P^-_n$, the prior estimate of the state error covariance matrix $P_n$.

For the measurement update, shown in Eq. (3.3), the innovation $e_n$ is first calculated as the difference between the predicted and actual measurements. The filter then calculates a measure of uncertainty, here called $U_n$, based on $P^-_n$. The Kalman gain $K_n$
is then calculated based on the inverse of the uncertainty measure, the measurement transition matrix, and the prior state error covariance estimate. Crucially here, larger Kalman gain 2-norms will correlate with higher error estimates and with less uncertainty. Thus, when the posterior state estimate is generated, the Kalman gain times the innovation forces error correction based on the amount of error and the confidence of the filter, acting as a negative feedback mechanism \[28\]. Finally, the posterior error covariance estimate \( P_n \) is generated based on the additive inverse of the Kalman gain, \( H \).

For timesteps where input data is missing, the measurement update is skipped, and we have that \( P_n = P_n^- \) and \( x_n = x_n^- \). The trace of \( P_n \) represents the sum of the mean squared errors between final prediction \( x_n \) and \( \bar{x}_n \), which is the presumed actual state of the system at timestep \( n \) \[31\], where \( z_n = H_n \bar{x}_n + v_n \). Vectors \( z_n \) and \( v_n \) are the state measurement and the measurement noise, respectively. \( P_n \) is used to generate the 95% confidence interval for the prediction.

### 3.2.1 The Kalman Recursions

**Prediction update**

\[
\begin{align*}
\dot{x}_n^- &= \alpha F x_{n-1}^- + w_n, & \text{prior state estimate} \\
\dot{P}_n^- &= F P_{n-1}^- F^T + Q_n, & \text{prior state error covariance estimate}
\end{align*}
\]

(3.1)

(3.2)
Measurement update
\[
\begin{align*}
e_n &= z_n - Hx_n^-, & \text{innovation} \\
U_n &= HP_n^- H^T + R_n, & \text{uncertainty} \\
K_n &= P_n^- H^T (U_n)^{-1}, & \text{Kalman gain} \\
x_n &= x_n^- + K_n e_n, & \text{posterior state estimate} \\
P_n &= (I - K_nH) P_n^-, & \text{posterior state error covariance estimate}
\end{align*}
\]

(3.3)

The variable definitions are presented for reference in Chapter 7.1.

3.3 Approach to Process and Measurement Model Design

Velocity and acceleration at a given timestep \( n \) are derived from extant position data using the first- and second-order difference equations

\[
\begin{align*}
\tilde{v}_n &= f_s(p_n - p_{n-1}) \\
\tilde{a}_n &= \frac{f_s}{2}(\tilde{v}_n - \tilde{v}_{n-1}) \\
&= \frac{f_s^2}{2}((p_n - p_{n-1}) - (p_{n-1} - p_{n-2})) \\
&= \frac{f_s^2}{2}(p_n - 2p_{n-1} + p_{n-2}).
\end{align*}
\]

(3.4)

where \( p_n \) is the measured position, in pixel coordinates, of an object within an image at sample \( n \). Here, we use \( p_n \) to represent \( x \) and \( y \) position interchangeably; there is no difference between the handling of the two. We use \( \tilde{v}_n \) and \( \tilde{a}_n \) rather than \( v_n \) and \( a_n \) for reasons that will be made clear in Section 4.2. We will make use of the process model from \cite{16}, with some modifications. This is a constant velocity (CV) model, in which both velocity and acceleration are presumed constant for the entire period between samples. Thus, equations (3.4) and (3.5) represent Riemann approximations.
of the continuous-time expressions for velocity and acceleration as a function of time,  
where if \( \Delta t_s \) is our sampling period,  
\[
    p_n = p_t, \quad f_s = (\Delta t_s)^{-1}, \quad \lim_{f_s \to \infty} (\bar{v}_n, \bar{a}_n) = (v_t, a_t).
\]

This implies that, insofar as we require our velocity and acceleration values to match  
real-world values, we need only use a high enough sample rate. In our case, \( f_s = 155 \)  
Hz. As long as we are only interested in estimating position, the sampling frequency  
is irrelevant, provided it gives sufficient temporal resolution to allow for tracking the  
objects of interest.

### 3.3.1 Process Model

In our model, prediction update equation \( \mathbf{x}_n^+ = \alpha \mathbf{F} \mathbf{x}_n + \mathbf{w}_n \) is

\[
\begin{bmatrix}
    p_{x_n} \\
    p_{y_n} \\
    \bar{v}_{x_n} \\
    \bar{v}_{y_n} \\
\end{bmatrix}
\begin{bmatrix}
    1 & 0 & f_s & 0 \\
    0 & 1 & f_s & 0 \\
    0 & 0 & 1 & 0 \\
    0 & 0 & 0 & 1 \\
\end{bmatrix}
\begin{bmatrix}
    p_{x_n} \\
    p_{y_n} \\
    \bar{v}_{x_n} \\
    \bar{v}_{y_n} \\
\end{bmatrix}
\begin{bmatrix}
    0 & 0 & 0 & 0 \\
    0 & 0 & 0 & 0 \\
    0 & 0 & 2 f_s & 0 \\
    0 & 0 & 0 & 2 f_s \\
\end{bmatrix}
\begin{bmatrix}
    \bar{a}_x \\
    \bar{a}_y \\
    \bar{a}_x \\
    \bar{a}_y \\
\end{bmatrix}
\begin{bmatrix}
    0 \\
    0 \\
    0 \\
\end{bmatrix},
\]

where \( \mathbf{w}_n \triangleq F_w \bar{a}_n \). Thus, the acceleration is incorporated as process noise and is  
assumed to follow a zero-mean normal distribution with covariance \( Q_n \). Conversely,  
this means that for any given \( n \), we can obtain \( Q_n = \bar{a}_n \bar{a}_n^T \).

Excepting missing samples, we have data for every timestep, which means that  
when initializing our Kalman filter, we are able to define matrices for all sequential data.  
Specifically, for all \( n \in \{1, \ldots, N\} \) we can compile matrix \( \bar{\mathbf{A}} \), where \( \bar{\mathbf{A}} \triangleq [\bar{\mathbf{a}}_1 \ldots \bar{\mathbf{a}}_N] \)  
for all \( \bar{\mathbf{a}}_n \) from timesteps with defined values for \( \bar{a}_{x_n}, \bar{a}_{y_n} \). Then we simply calculate
the sample variances of each row of \( \tilde{A} \) and use them to generate \( \tilde{a}_n \) for any timesteps that are missing data. Then we use these vectors to generate corresponding sample covariance matrices \( Q_n \). In this way, when our filter is dealing with missing samples and thus making predictions based only on previous information, it is able to leverage some statistical properties of the available data.

### 3.4 Measurement Model

To explain the design of measurement transition matrix \( H \), we note that the measurement model assumes errors in measurement should only directly affect the position vector. Otherwise, since the velocity data is derived from the positional data, we would effectively be double penalizing any error in position. Thus, our measurement transition matrix is defined as

\[
H \triangleq \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0
\end{bmatrix}.
\]

In deriving our added noise, we must consider what information we have about how the given positional measurements have been calculated. An important parameter of the dataset under consideration is “blobsize,” a measure of the number of pixels comprising a given object, as interpreted by the segmentation algorithm. To facilitate discussion, we define the unit “Px” as representing square pixels, and “px” representing a linear unit one pixel’s edge in size. Each set of \( p_{x_n}, p_{y_n} \) coordinates in the dataset, describing some object’s position at a given sample \( n \), has an associated blobsize measurement \( z_{b_n} \), and these measurements range in natural numbers anywhere from 1 to roughly 50 Px (see Appendix A.3.2 for a more detailed discussion of the distribution.
of blob sizes in the real dataset). For now, suffice to say that the blob size is linked by real-world distance from the sensor and a number of system parameters to the actual size of the object’s projection onto a 2-dimensional plane. The $p_x, p_y$ coordinates in the data are based on the calculated centroids of individually segmented objects [12]. A larger object in our particular dataset is assumed either to correspond to a multi-bird occlusion, or to a bird with wings extended. Either of these is an object or objects whose true center of mass is more difficult to determine, based on the outline of its imaged representation, than that of a smaller object or objects. It follows that a larger value for blob size represents a measurement with more uncertainty. The advantage of this approach is that we are able to incorporate dynamically changing information based on real measurements to model our measurement uncertainty, rather than assuming wide sense stationarity in our measurement noise.

As has been noted in [32], the precision of the sensor must be considered as well as uncertainty stemming from the segmentation algorithm. A hypothetical point-sized object captured in an image as having a one-pixel blob size could, with some caveats, be reasonably assumed to be anywhere within said pixel. The 1D positional error, $\varepsilon_p$, for such an object can be modeled as $\varepsilon_p \sim U(-0.5, 0.5)$ for both $x$ and $y$. We can go on to imagine some circular object with $r = 0.5$px. The blob size of such an object can be anywhere from 1px to 4px depending on the actual location of its center, due to “pixel-splitting,” i.e., the true center of an object being located on the border between two or more pixels. Without more detailed information from the segmentation algorithm, it would not be possible to determine which scenario is representative of the truth. This ambiguity increases with increasing $r$.

Our noise model, therefore, requires some simplifying assumptions. We first say that $\varepsilon_p = \varepsilon_{p_x} = \varepsilon_{p_y} = \varepsilon_z + \varepsilon_q$, where $\varepsilon_z, \varepsilon_q$ are our error measures due to blob size $z_b$ and pixel quantization, respectively. Since we cannot differentiate between the effects of object size and pixel-splitting, we simply draw the pixel quantization component of
our error, $\varepsilon_q$ from the worst-case scenario, where $\varepsilon_q \sim \mathcal{U}(-0.5, 0.5)$. The variance on the uniform distribution with a range of 1 is $\frac{1}{12}$, and thus the standard deviation is $\frac{1}{\sqrt{12}}$. As for the blobsize, we know that, excluding occlusions, a significant amount of the variation in blobsize for a given object occurs as a result of the bird represented by the object flapping its wings. Having observed actual images from the dataset of birds in flight, we can also guess that the centroid of an object, disregarding any loss of precision from pixel quantization, should not typically deviate from the true center of mass in any given direction by much more than two pixels. To this end we introduce the tuning parameter $\varrho$, which is a scalar multiplier that allows us to control the effect of input blobsize $z_{bn}$ on our blobsize error measure $\varepsilon_{z_n}$. This error measure is a 1D measurement, representing a presumed standard deviation, that is derived from a 2D measure, so we have that $\varepsilon_{z_n} = \sqrt{\varrho z_{bn}}$.

In our actual noise model, our scaled blobsize measure for a given sample is used to generate the measurement noise covariance matrix $R_n$, where the noise is assumed to be uncorrelated between $x$ and $y$. Our assumptions are regarding the standard deviation for the positional measurement, so for our matrix to apply to the derived velocity measurements, as it must if the velocity is derived from uncertain position measurements, we have that $\varepsilon_{v_n} = f_s \varepsilon_{p_n}$. Finally, since $R_n$ is a covariance matrix, we must square our standard deviation measures. Therefore, for timestep $n$,

$$R_n = \begin{bmatrix}
\left(\sqrt{\varrho z_{bn}} + \frac{1}{\sqrt{12}}\right)^2 & 0 & 0 & 0 \\
0 & \left(\sqrt{\varrho z_{bn}} + \frac{1}{\sqrt{12}}\right)^2 & 0 & 0 \\
0 & 0 & f_s^2 \left(\sqrt{\varrho z_{bn}} + \frac{1}{\sqrt{12}}\right)^2 & 0 \\
0 & 0 & 0 & f_s^2 \left(\sqrt{\varrho z_{bn}} + \frac{1}{\sqrt{12}}\right)^2
\end{bmatrix}.$$ (3.7)

Fortunately, since acceleration is modeled as process noise, we do not need to account for it in $R_n$. To attain values for $\varrho$, we simply perform a grid search for the
value that results in desirable mean and standard deviation values across datasets under consideration.

Clearly, some art has been required to estimate the variance, but we consider this a reasonable approximation. Since we have directly calculated our noise covariances from measured information and the measurement noise vector \( v_n \) does not appear directly in the recursions, we can omit it from consideration altogether.

### 3.5 Bi-Directional Kalman Filter: Approach and Justification

As discussed in Section 3.1, a standard implementation of a Kalman filter is not able to incorporate future measurements. Though it may be possible to effect some non-standard implementation that somehow incorporates future estimates, perhaps weighted by number of timesteps away from present, such an approach would require alteration of the model depending upon the length of gaps present. Such a model would likely suffer from an overabundance of complexity and would not be guaranteed to work.

The main issue with the Kalman filter when dealing with gaps in data is not stability, or lack of ability to generate predictions. A standard implementation, which indeed describes ours, will continue to generate positional predictions regardless of whether measurements arrive for a given timestep; faced with this situation it simply outputs its prediction step estimates without updating them in the measurement step. In our model, this has the effect that the predicted objects simply continue in roughly the same direction they were moving in at roughly the same velocity, influenced only by “acceleration” noise generated using statistical properties of the intact acceleration data, as in Section 3.3. A potentially large error results when, for example, a gap occurs in the data during a period of significant curvature, or an object initiates a change of direction during a gap, or a gap begins during a particularly noisy event in
the data. The problem is exacerbated the longer the duration, in samples, of the gap.

Our bi-directional filter implementation is a lightweight attempt to alleviate these difficulties. The principle is simply that one standard filter run through the data is performed, then another performed on a time-reversed copy of the data. The output of the reversed filter is again reversed, and the results averaged by sample to obtain a smoothed estimate. At minimum, this approach allows the resultant predictions to reflect what happens on both sides of the gap, and could ultimately be refined to better reflect the influence of intact data in terms of number of samples in time separation from a given predicted step.

Empirically, it has been found that the bi-directional filter outperforms the uni-directional filter on the synthetic dataset, and is slightly worse on the real dataset. These results are presented in detail in Section 5.2.2.

3.6 The Kalman Smoothing Filter

One other option which, while not explored in depth in this work, bears mentioning is the Kalman smoothing filter. This is another, more mathematically elegant method of leveraging both forward and reverse-time predictions to obtain better results than are possible with a standard implementation of the Kalman filter. There are several variations on smoothing algorithms based on the inference methods developed for the Kalman filter. We will discuss one particular version, known as the Rauch-Tung-Striebel (RTS) smoother. We shall see that the RTS smoother is a mathematically rigorous estimation method that incorporates all available samples in a non-causal framework. Thus it would likely have been a better method for comparison than our own bi-directional Kalman filter, and would have been worthwhile to implement, given unlimited time. We include a description here for completeness.

An advantage of Kalman smoothing algorithms is that they are extensions of
the methods already developed for the Kalman filter; an RTS smoother, in other words, does not require any further development of the system model we have already developed. It has similar recursive calculation methods as the standard, forward-pass Kalman filter, and thus has similar requirements for execution time and memory usage. One critical advantage of the RTS smoother over our simple bi-directional algorithm is that it incorporates the smoothing process into its recursions, eliminating any need to scheme methods for weighting samples after the fact.

The RTS smoother works on the principle that, for a given track with $N$ total samples, any given sample $n$ should be able to incorporate every measured sample. So, while a Kalman filter makes its estimate based on the system state in sample range $[0, n]$, an RTS makes use of all system states in range $[0, N]$. Crucially, the RTS filter begins its smoothing pass at the last sample and continues to the first. This requires that all measured samples be available at runtime for the algorithm, which requirement would, naturally, not be a problem for our purposes. All system states before a given timestep $n$ are used implicitly since the input “measurements” for the reverse-time smoothing pass are the outputs, $x_n$ and $P_n$, of the Kalman filter. All system states after a given sample $n$ are incorporated implicitly in the recursive smoothing pass.

We will denote the final RTS output state estimate as $\hat{x}_n$, the state error covariance estimate as $\hat{P}_n$, and the gain parameter as $\hat{K}_n$. After initializing the filter with $x_N, P_N$, the RTS pass is performed starting at sample index $n = N - 1$, with the final system output calculated as
\[ \hat{K}_n = P_n F^T (P_n)^{-1} \quad (3.8) \]
\[ \hat{x}_n = x_n + \hat{K}_n (\hat{x}_{n+1} - x_{n+1}) \quad (3.9) \]
\[ \hat{P}_n = P_n - \hat{K}_n (P_{n+1}^{-1} - \hat{P}_{n+1}) \hat{K}_n^T. \quad (3.10) \]

As with the standard Kalman filter, we note that the RTS gain is chiefly dependent upon an uncertainty measure, specifically the one provided by the forward pass of the Kalman filter. We note that it is also a function of Kalman state transition matrix \( F \), which makes sense, since unlike the forward pass we are no longer differentiating between a prediction and measurement update. Final estimate \( \hat{x}_n \) is Kalman estimate \( x_n \) plus RTS gain \( \hat{K}_n \) times the difference between the Kalman and RTS outputs for the previous RTS step. Thus, “significance” of the update, in terms of magnitude, is proportional to the significance of the previous update. The RTS state error covariance matrix \( \hat{P}_n \) is calculated in a similar manner.
Chapter 4

Stochastic Straight-Line Perturbation

4.1 Motivation

While a well-designed and well-implemented Kalman filter may yield a good estimate for the actual position of a given tracked object, we shall see in Chapter 5 that, even when painstakingly optimized, it will have a difficult time adequately tracking certain types of data when that data is missing samples.

As we have noted, the Kalman filter is strictly causal, which makes it an excellent choice for real-time tracking applications, but not typically optimal for situations such as ours, where we have foreknowledge of the object’s measured position at later timesteps. The bi-directional Kalman filter implementation described above is a crude attempt to integrate this prior information into our estimate. While it does improve performance to some degree, it does not address the fundamental problem: a standard Kalman filter has no way to guide the sample-by-sample estimates based on known future values of the data.

We would like some way of interpolating between start- and endpoints of positional data that realistically models the underlying data. Specifically, we know that the motion of birds is highly constrained: the physics of flight demand that some forward momentum is conserved in order to remain aloft, which translates to trajectories with smoothly changing velocity values in both the $x$ and $y$ directions in a given 2D image. We propose a method, Stochastic Straight-Line Perturbation (SSLP), of interpolating
positional values using statistical estimates of the underlying acceleration values.

Why use acceleration values? The simplest means of interpolation is always linear interpolation, wherein a straight line is drawn from start to endpoint. Clearly, for objects that do not move in straight lines with instantaneous, discrete changes in trajectory, a linear interpolation of positional values will yield poor representations of object motion. But why not use velocity? By smoothly changing values of velocity between samples, we can yield positional estimates that conform to the types of trajectories we are trying to estimate. However, while the resultant velocity estimates can easily be designed to create positional estimates that start and end in the right place, it is far more difficult to constrain them so that the endpoints of the underlying velocity vector smoothly match the surrounding values.

Since discrete acceleration values are derived as the difference between velocity values, a linear interpolation of velocity can be exactly generated by a row of constant acceleration values. By using estimates based on a reconstructed acceleration vector, we gain several advantages. First, by perturbing such a constant vector we can generate positional vectors with similar statistical properties to the surrounding measurements, effectively emulating measurement and process noise present in the original data. This makes SSLP a suitable pre-processing step for data to be Kalman filtered, if desired. Second, using acceleration values makes it relatively simple to create estimates that not only match the start- and endpoints of the positional data, but of the implicit velocity data as well, effectively leveraging first- and second-order characteristics of the measured data to create more accurate estimates. Third, as we shall see, using acceleration greatly simplifies our estimation process in certain situations where one or two positional measurements are surrounded by several rows of missing data.

A possible bonus is that, while SSLP uses a constant vector of acceleration values with added noise, it is theoretically adaptable to more complex initial estimates of acceleration values, perhaps based on motion studies for a given subject, that can
then be perturbed with noise. Additionally, since it operates on one dimension at a
time, SSLP can be easily adapted to reconstruct data that has already been rendered
into three or more dimensions.

4.2 Theoretical Underpinnings

Suppose we have in the \( x \) component of our trajectory a gap of \( N - 1 \) samples. We
define the indices of this gap such that the last known position before the gap is \( p_0 \) and
the first known position after the gap is \( p_N \). As with our Kalman filter implementation,
velocity and acceleration are derived from position using first- and second-order
difference equations (3.4) and (3.5), with all accompanying assumptions. Again, \( \tilde{v}_n, \tilde{a}_n \)
here represent velocity and acceleration values that have not been normalized by
sample rate, as detailed below.

It is worth noting that, for our application, we are not especially concerned with
these values in an external sense anyway, as we are only interested in a method for
deriving probable missing positional values. Before proceeding, we also point out that,
while for our application we make the assumption that any missing positional value \( x \)
will likewise be missing in the \( y \) data, in principle this need not always be the case.

To simplify the derivations involved, before working with these velocity and
acceleration values, we normalize to avoid having to account for sample rate, thus for
our purposes,

\[
\begin{align*}
v_n &= p_n - p_{n-1} \\
a_n &= v_n - v_{n-1} \\
&= (p_n - p_{n-1}) - (p_{n-1} - p_{n-2}) \\
&= p_n - 2p_{n-1} + p_{n-2}.
\end{align*}
\]
Hereafter, the notation will reflect this fact.

Implicit in the above equations is the fact that, with \( N - 1 \) missing positional samples, we also have \( N \) missing velocity samples, and \( N + 1 \) missing acceleration samples. We would like to be able to express an estimate for acceleration in terms of our known start and end values for position and velocity. For example, suppose that \( N = 6 \). Then, we have the telescoping series

\[
p_6 = v_6 + p_5 = v_6 + v_5 + p_4 = \ldots = p_0 + \sum_{n=1}^{6} v_n,
\]

which in turn implies that

\[
p_N = p_0 + \sum_{n=1}^{N} v_n.
\]

Similarly, we have that

\[
v_n = v_0 + \sum_{m=1}^{n} a_m.
\]

Compositing these, we get that

\[
p_N = p_0 + \sum_{n=1}^{N} (v_0 + \sum_{m=1}^{n} a_m)
\]

\[
= p_0 + N v_0 + \sum_{n=1}^{N} \sum_{m=1}^{n} a_m.
\]

Further,

\[
p_N = p_0 + N v_0 + (a_1) + (a_1 + a_2) + \ldots
\]

\[
= p_0 + N v_0 + N a_1 + (N - 1) a_2 + \ldots + 2 a_{N-1} + a_N,
\]

which is in terms of \( p_n, p_0 \) as desired. For our example where \( N = 6 \), we have that \( p_N = p_0 + 6v_0 + 6a_1 + 5a_2 + 4a_3 + 3a_4 + 2a_5 + a_6 \). The weight vector we create from these coefficients will be labeled \( \ell \), and our final, constrained vector estimate of the
missing acceleration values \( \hat{a} \). Formally,

\[
\forall N \in \mathbb{N}, \quad \ell \in \mathbb{R}^N \triangleq \begin{bmatrix} N & (N-1) & \ldots & 1 \end{bmatrix}^T, \quad \hat{a} \in \mathbb{R}^{N+1} \triangleq \begin{bmatrix} \hat{a}_1 & \hat{a}_2 & \ldots & \hat{a}_{N+1} \end{bmatrix}^T
\]

Regarding the relationship between the missing acceleration samples and known \( v_{N+1}, v_0 \), we have that

\[
v_{N+1} = v_0 + \sum_{n=1}^{N+1} \hat{a}_n.
\]

We define \( \mathbf{1}_N \in \mathbb{R}^N \) whose elements are all ones, as well as all-ones vector \( \mathbf{1}_{N+1} \in \mathbb{R}^{N+1} \), then re-write \( v_{N+1} \) as

\[
v_{N+1} = v_0 + \hat{a}^T \mathbf{1}_{N+1}.
\]

Let \( \mathbf{x}_0 \in \mathbb{R}^{N+1} \) denote some vector \( \mathbf{x} \in \mathbb{R}^N \) with a zero appended to the end. We now have the two constraint equations

\[
\begin{align*}
p_N &= p_0 + Nv_0 + \hat{a}^T \ell_0, \quad \text{(4.1)}
v_{N+1} &= v_0 + \hat{a}^T \mathbf{1}_{N+1}.
\end{align*}
\]

To summarize, the above analysis indicates that our estimate \( \hat{a} \) must satisfy the following constraints in order to match our known positional and velocity values

\[
\begin{align*}
\hat{a}^T \ell_0 &= p_N - p_0 - Nv_0, \quad \text{(4.2)}
\hat{a}^T \mathbf{1}_{N+1} &= v_{N+1} - v_0. \quad \text{(4.3)}
\end{align*}
\]

### 4.3 SSLP Outlined

Before providing details of our method, we briefly outline the estimation procedure. Let \( \mathcal{A} \) be the set of all vectors in \( \mathbb{R}^{N+1} \) that satisfy constraint equations \( \text{(4.2)} \) and \( \text{(4.3)} \). Our goal is to find vector \( \hat{a} \in \mathcal{A} \) whose sample variance \( \sigma^2_{\hat{a}} \) matches the sample
variance $\sigma^2$ of the intact data surrounding our row of missing values. Broadly speaking, our strategy for estimating $\hat{a}$ will be to first generate a naive estimate $\bar{a}$ that satisfies Eq. (4.2), then perturb it using stochastic methods so that it satisfies Eq. (4.3), resulting in final estimate $\hat{a}$. We then use $\hat{a}$ to generate our positional estimates. We will now describe the procedure for dealing with $a$, the standard row of missing acceleration values.

4.4 Satisfying Equation (4.2)

Our goal in this step is to generate naive estimate $\bar{a}$ that satisfies Eq (4.2). In principle, there are many possible approaches to generating $\bar{a}$; it need only be a collection of acceleration values that, when summed in the correct manner, will generate a row of position estimates that link $p_0$ to $p_{N-1}$. Note that the constant vector $\bar{a} = \beta \mathbf{1}_{N+1}$ satisfies Eq. (4.2). Hence, our goal is to find the appropriate constant $\beta$.

With $\Delta_p \triangleq p_N - p_0$, the constant $\beta$ is calculated using Eq. (4.2), substituting $\bar{a}$ for $\hat{a}$:

$$\bar{a}^T \ell_0 = \Delta_p - Nv_0$$

$$\beta(\mathbf{1}_{N+1})^T \ell_0 = \Delta_p - Nv_0$$

$$\beta = \frac{\Delta_p - Nv_0}{(\mathbf{1}_{N+1})^T \ell_0}$$

So,

$$\beta = 2 \frac{\Delta_p - Nv_0}{N(N + 1)}.$$ (4.4)

The last step results from using the well-known triangle equation. Thus, $\bar{a}$ has a simple, closed-form solution.
4.5 Satisfying Equation (4.3)

In perturbing \( \mathbf{a} \) to obtain final estimate \( \hat{\mathbf{a}} \), we utilize the fact that, given any vector \( \mathbf{w} \in \mathbb{R}^N \),

\[
\sum_{i=1}^{N} \left( w_i - \frac{1}{N} \sum_{j=1}^{N} w_j \right) = \mathbf{1}_N^T (\mathbf{w} - \frac{1}{N} \mathbf{1}_N 1_N^T \mathbf{w})
= \mathbf{1}_N^T \mathbf{w} - \frac{1}{N} \mathbf{1}_N 1_N^T \mathbf{w}
= \mathbf{1}_N^T \mathbf{w} - \mathbf{1}_N^T \mathbf{w}
= 0.
\]

Now we can define \( \bar{\mathbf{w}}_0 \in \mathbb{R}^{N+1} \triangleq (\mathbf{w}_0 - \frac{1}{N} \mathbf{1}_{N|0} 1_{N|0}^T \mathbf{w}_0) \circ \ell_0^{o-1} \), where \( \circ \) denotes the element-wise (Hadamard) product and \( (\cdot)^{o-1} \) is the element-wise inverse of some vector.

Then we have \( (\hat{\mathbf{a}} + \bar{\mathbf{w}}_0)^T \ell_0 = \hat{\mathbf{a}}^T \ell_0 \). Proof:

\[
(\hat{\mathbf{a}} + \bar{\mathbf{w}}_0)^T \ell_0 = \hat{\mathbf{a}}^T \ell_0
\]

\[
\hat{\mathbf{a}}^T \ell_0 + \bar{\mathbf{w}}_0^T \ell_0 = \hat{\mathbf{a}}^T \ell_0
\]

\[
\hat{\mathbf{a}}^T \ell_0 + \left( (\mathbf{w}_0 - \frac{1}{N} \mathbf{1}_{N|0} 1_{N|0}^T \mathbf{w}_0) \circ \ell_0^{o-1} \right)^T \ell_0 = \hat{\mathbf{a}}^T \ell_0
\]

\[
\hat{\mathbf{a}}^T \ell_0 + \mathbf{w}_0^T \ell_0 \circ \ell_0^{o-1} - \frac{1}{N} \mathbf{w}_0^T \mathbf{1}_{N|0} 1_{N|0}^T \mathbf{1}_{N|0} \ell_0 \circ \ell_0^{o-1} = \hat{\mathbf{a}}^T \ell_0
\]

\[
\hat{\mathbf{a}}^T \ell_0 + \mathbf{w}_0^T \mathbf{1}_{N|0} - \frac{1}{N} \mathbf{w}_0^T \mathbf{1}_{N|0} 1_{N|0}^T \mathbf{1}_{N|0} = \hat{\mathbf{a}}^T \ell_0
\]

\[
\hat{\mathbf{a}}^T \ell_0 + \mathbf{w}_0^T \mathbf{1}_{N|0} - \mathbf{w}_0^T \mathbf{1}_{N|0} = \hat{\mathbf{a}}^T \ell_0
\]

Likewise, given some scalar \( \gamma \),

\[
(\hat{\mathbf{a}} + \gamma \bar{\mathbf{w}}_0)^T \ell_0 = \hat{\mathbf{a}}^T \ell_0.
\]
Thus, we can randomly generate \( \mathbf{w} \), then normalize by subtracting the sample mean \( \mu_{\mathbf{w}} \) (not the parameter \( \mu \) used to generate the vector, which in practice may as well be 0). Any resultant vector \( \mathbf{\bar{w}}_{|0} \triangleq \left[ ((\mathbf{w} - \mathbf{1}_N \mu_{\mathbf{w}}) \circ \ell^{e^{-1}})^{T} \ 0 \right]^{T} \) will still satisfy constraint Eq. (4.2) when added to \( \mathbf{\bar{a}} \).

Let \( \Delta_v \triangleq v_{N+1} - v_0 \). To satisfy constraint Eq. (4.3), we substitute \( (\mathbf{\bar{a}} + \gamma \mathbf{\bar{w}}_{|0}) \) for \( \hat{\mathbf{a}} \) and solve for \( \gamma \)

\[
(\mathbf{\bar{a}} + \gamma \mathbf{\bar{w}}_{|0})^{T} \mathbf{1}_{N+1} = \Delta_v
\]

\[
\mathbf{\bar{a}}^{T} \mathbf{1}_{N+1} + \gamma \mathbf{w}_{|0}^{T} \mathbf{1}_{N+1} = \Delta_v.
\]

Thus

\[
\gamma = \frac{\Delta_v - \mathbf{\bar{a}}^{T} \mathbf{1}_{N+1}}{\mathbf{w}_{|0}^{T} \mathbf{1}_{N+1}} = \frac{\Delta_v - \beta(N + 1)}{\mathbf{w}_{|0}^{T} \mathbf{1}_{N+1}}.
\]  

(4.5)

To avoid as much confusion as possible, it is worth noting that while equations for satisfying constraint Eq. (4.3) are dependent upon the last element of \( \hat{\mathbf{a}} \) (i.e., \( \hat{a}_{N+1} \)), those for satisfying Eq. (4.2) are not. Since, by definition, the last element of \( \mathbf{\bar{w}}_{|0} \) is 0, the scalar \( \gamma \) has no authority to modify \( \hat{a}_{N+1} \), which is still \( \bar{a}_{N+1} \). Accordingly, after obtaining \( \mathbf{\bar{w}}_{|0} \triangleq \gamma \mathbf{\bar{w}}_{|0} \), we must finish our final estimate \( \hat{\mathbf{a}} \in \mathbb{R}^{N+1} \) by calculating the last element of our additive noise vector such that \( \hat{\mathbf{a}} \) satisfies Eq. (4.3).

We will need \( \mathbf{0}_{N|1} \triangleq \left[ \mathbf{0}_{N}^{T} \ 1 \right]^{T} \in \mathbb{R}^{N+1} \), where \( \mathbf{0}_{N} \in \mathbb{R}^{N} \) is a vector of zeros. We
then wish to satisfy

$$
\sum_{i=1}^{N+1} \hat{a}_i = \Delta v
$$

$$
\hat{a}^T 0_{N|1} + \hat{a}^T 1_{N|0} = \Delta v
$$

$$
0 + \ldots + 0 + \hat{a}_{N+1} = \Delta v - \hat{a}^T 1_{N|0}.
$$

This leads to an interesting result when we evaluate the resulting vector \( \hat{a} \) using (4.3):

$$
\hat{a}^T 1_{N+1} = \Delta v
$$

$$
\hat{a}^T 1_{N|0} + \hat{a}^T 0_{N|1} = \Delta v
$$

$$
\hat{a}_{N+1} = \Delta v - \hat{a}^T 1_{N|0}
$$

$$
= \Delta v - (\bar{a} + \hat{\gamma} \bar{w}_{|0})^T 1_{N|0}
$$

$$
= \Delta v - (\bar{a} + \frac{\Delta v - \beta (N + 1)}{\bar{w}_{|0}^T 1_{N|0}} \bar{w}_{|0})^T 1_{N|0}
$$

$$
= \Delta v - N \beta - \Delta v + (N + 1) \beta
$$

$$
\hat{a}_{N+1} = \beta.
$$

Thus the full form for \( \hat{a} \) is

$$
\hat{a} = \bar{a} + \hat{\gamma} \bar{w}_{|0} = 2 \frac{\Delta p - N v_0 - \bar{a}^T \ell'_{N|0}}{N(N+1)} 1_{N|0} + \bar{w}_{|0} \frac{\Delta v - \beta (N + 1)}{\bar{w}_{|0}^T 1_{N|0}}.
$$

(4.6)

4.5.1 Sample Variance and Generating \( w \)

Since it will ultimately be normalized by its own sample mean, there are theoretically infinite ways to generate \( w \). The only real constraint is that we would like to minimize the difference between \( \sigma^2_{\hat{a}} \), the variance of \( \hat{a} \), and \( \sigma^2_{\bar{a}} \), the variance of the surrounding
samples. We can formalize this by defining the vector $s$, which is defined based on the values of the acceleration data present surrounding the missing values. There are likewise many possible ways to decide how many of these values to include. At present the algorithm operates according to the rule

$$s = \left[ s_\cdot^T \ s_+^T \right]^T,$$

where $s_+$ and $s_-$ are existing acceleration values in the data bordering $a$. Since $s_-$ and $s_+$ may have different variances, and we are trying to bridge the gap between the two rows, we would like $\sigma_a^2$ to be based on a suitable number of samples from $s$. For this reason, it is useful to define $n_{\text{min}}$, the minimum number of samples in $s$ on which to base our target variance, as

$$\min(N, n_{\pm}),$$

where $N$ is the sample length of $a$, and $n_{\pm}$ is the number of samples available in $s$.

It is also necessary to guarantee some minimum number of samples in both $s_+, s_-$ in order for the calculated variance to be valid, since, for instance, if only one sample is available, its variance will be zero. As of this writing, the algorithm uses a minimum value of 4, but it is possible this could be tuned as a hyperparameter in the future. If $n_{\text{min}}$ samples are not available on both sides of $a_k$, it is skipped until the algorithm has filled in other $a$ vectors, which can then be resampled to obtain a target variance.

For this thesis, we have evaluated four separate strategies for generating $w$:

1. Normal: $w \sim \mathcal{N}(0, \sigma^2_a)$

2. Split Rademacher: $w \in \{-1, 1\}$, called “split” because even number of positives and negatives is enforced. If $N$ is odd, the resultant vector is multiplied by $\pm 1$, as determined by a Rademacher random variable with $p = 0.5$ in order to avoid biasing towards positive or negative values.
3. Uniform: \( w \sim U(-1, 1) \)

4. Normal + Uniform: \( w = w_1 + w_2, w_1 \sim \mathcal{N}(0, \max\left(\left(\sqrt{\sigma_s^2} + 0.5\right)^2, 0.5^2\right)), w_2 \sim U(-0.5, 0.5) \). Matches the “normal plus uniform” noise model used in the Kalman filter, incorporating pixel quantization error.

The relative performance of these methods will be compared in Section 5.2.3.

4.5.2 Minimizing MSE of Variance

We are hoping to minimize the difference between \( \sigma_a^2 \triangleq \text{var}(\hat{a}) \) and \( \sigma_s^2 \) as much as possible and, since in our implementation \( \hat{a} \) is a constant vector plus scaled noise vector \( \hat{w} \), \( \sigma_a^2 \) is really \( \sigma_w^2 \). Although we may have generated \( \hat{w} \) using \( \sigma_s^2 \) as a parameter, we have also scaled it by \( \tilde{\gamma} \), which, depending on the composition of \( \hat{w} \), may result in suboptimal variance. Due to the Law of Large Numbers (LLN), this problem is exacerbated with smaller \( N \). Since we do not know \( \tilde{\gamma} \) until after we have generated \( \hat{w} \), we cannot simply compensate for the value of \( \tilde{\gamma} \) by adjusting a variance parameter for \( \hat{w} \).

We could try to minimize the MSE between \( \sigma_a^2 \) and \( \sigma_s^2 \) using some form of gradient descent. The gradient \( \nabla_w(\sigma_a^2 - \sigma_s^2)^2 \) turns out to be

\[
\nabla_w \left( \sigma_a^2 - \sigma_s^2 \right)^2 = \frac{1}{(N+1)^4} \left[ (N+1)(\tilde{\gamma}\hat{w}|_0 + \bar{a}) - 1_{N+1,N+1}(\tilde{\gamma}\hat{w}|_0 + \tilde{a}) \right] \\
\left[ \tilde{\gamma}^2 \left( 2\hat{w}_T|_0((N+1)(\tilde{\gamma}\hat{w}|_0 + \bar{a}) - 1_{N+1,N+1}(\tilde{\gamma}\hat{w}|_0 + \tilde{a})) \right) + \\
2\tilde{\gamma} \left( \bar{a}^T((N+1)\bar{a} - 1_{N+1,N+1}\tilde{a}) - (N+1)^2\sigma_s^2 \right) \right],
\]

where \( 1_{N+1,N+1} \in \mathbb{R}^{(N+1) \times (N+1)} \) is the matrix of all ones. This gradient descent turns
out to be unstable, which is not surprising, since it seeks to find a global minimum in a function space that has an infinite number of local minima. When it is able to find a solution, it is quite slow to converge, especially given its use case as a procedure that may need to be repeated thousands of times for a given dataset. Fortunately, there is a more efficient method of arriving at a satisfactory answer most of the time.

Assuming \( \hat{w} \) is stochastically generated, some guesses for \( \hat{w} \) are certain to yield better results for final acceleration vector estimate \( \hat{a} \) than others. We can leverage this by defining \( M \) as the number of guesses we would like to make in a single round, then define matrix \( \bar{W}_\gamma \in \mathbb{R}^{M \times (N+1)} \) whose rows are transposed guesses for \( w_{|0} \), i.e.,

\[
\bar{W}_\gamma \triangleq \begin{bmatrix}
    w_{T}^T_{|01} \\
    w_{T}^T_{|02} \\
    \vdots \\
    w_{T}^T_{|0M}
\end{bmatrix}.
\]

We also define \( L_\gamma \in \mathbb{R}^{(n+1)\times(n+1)} \triangleq \text{diag}(\ell^-_{|0} \cdot 1) \). Then we have matrix \( \bar{W}_p = \bar{W}_\gamma L_\gamma \), whose rows are candidates for \( \bar{w} \). From here, we generate \( \bar{g} \), a vector of \( \gamma \) corresponding to the rows in \( \bar{W}_p \), by substituting rows of \( \bar{W}_p \) into the denominator of the \( \gamma \) formula in place of \( \bar{w} \). Finally, we choose \( \hat{a}^* \):

\[
\hat{a}^* = \arg \min_i \left[ \text{var}(\bar{a} + \bar{g}, \bar{W}_{p(i,:)} - \sigma_s^2)^2 \right].
\] (4.7)

This is simple, fast, and can be run until an \( \hat{a}^* \) is generated that meets some stopping criterion for maximum MSE between \( \text{var}(\hat{a}) \) and \( \sigma_s^2 \). If a suitable vector is not generated after some number of iterations, a gradient descent can then be performed, if desired, or the best candidate can simply be chosen.
4.5.3 Accounting for $\hat{a}_{N+1}$

Allowing the last element of our estimate vector $\hat{a}$ to always be $\beta$ presents several problems. First, we would like our entire vector of estimated acceleration values to be subject to whatever stochastic rule generates $\bar{w}$, as this better models the data we are trying to interpolate. Second, when the last element of $\hat{a}$ is constrained to always be the simple offset value, it can force the rest of the vector to do some strange things as a result, particularly in situations where there are only three missing acceleration values in a row, i.e., when $N = 2$. This is, again, because of LLN; with fewer degrees of freedom available to satisfy constraint Eq. (4.2), it becomes increasingly unlikely that suitable values are generated. After scaling, this can result in a vector with higher-than-desired variance, resulting ultimately in positional estimates that lurch wildly off-track for one sample.

It has been shown that since $\hat{a}_{N+1}$ does not factor into the calculations for Eq. (4.2), $\hat{a}_{N+1} = \beta$ regardless of anything else that happens. In fact, $\hat{a}_{N+1} = \bar{a}_{N+1}$ no matter what its value. If we instead define $\bar{a}$ as $\bar{a} = \beta 1_{N|0} + \alpha 0_{N|1}$, where $\alpha$ is some constant,

$$\hat{a}_{n+1} = \Delta_v - (\bar{a} + \bar{\gamma} \hat{w}|0)^T 1_{N|0}$$

$$= \Delta_v - (\beta 1_{N|0} + \alpha 0_{N|1}) + \frac{\Delta_v - (\beta 1_{N|0} + \alpha 0_{N|1})^T 1_{N+1}}{\hat{w}|0} 1_{N|0}$$

$$= \Delta_v - (n\beta + \Delta_v - n\beta - \alpha)$$

$$= \alpha$$

Note that $0_{N|1}^T 1_{N|0} = 0$, while $0_{N|1}^T 1_{N+1} = 1$.

Since we want $\alpha$ to fit with the rest of our scheme, we simply assign $\alpha$ according to whatever rule has been used to generate $w$ (recall that $\bar{w} = w \circ \ell^o$). This slightly changes the method of calculating $\bar{\gamma}$, which we will re-define as $\bar{\gamma}_\alpha$ to reflect the change
in methodology. Given $\bar{w}_{| \alpha} \triangleq \begin{bmatrix} \bar{w} & \alpha \end{bmatrix}$ we can follow the same derivation as Eq. (4.5) to calculate $\bar{\gamma}_{\alpha}$ as

$$\bar{\gamma}_{\alpha} = \frac{\Delta_v - \hat{\alpha}^T \mathbf{1}_{N+1}}{(\bar{w}_{| \alpha})^T \mathbf{1}_{N+1}}.$$  

(4.8)

In order to differentiate between the two approaches, we define $\hat{\alpha}$ for the case where this method has been used. Comparative results for the two approaches are presented in Section 5.2.3.

4.6 Bi-Directional SSLP

As with the Kalman filter implementation, it is worth investigating the possibility that the SSLP algorithm also benefits from an averaged, bi-directional approach. However, again due to LLN, it is near-certain that averaging two row estimates together will reduce the sample variance of the resulting vector. Thus, in creating acceleration vector estimates to be averaged together, it is useful to adjust the target variance to compensate. Let $\bar{a} \in \mathbb{R}^n$ represent the result of an averaging operation between forwards- and backwards-estimated $\hat{a}_1$ and $\hat{a}_2$, let $\sigma_a^2$ represent the variance of $\bar{a}$ without any compensation, and $\mathbf{1}_{N,N} \in \mathbb{R}^{n \times n}$ be the matrix of all ones. We have that

$$\sigma_a^2 = \frac{1}{n} \bar{a}^T \bar{a} - \frac{1}{n^2} \bar{a}^T \mathbf{1}_{N,N} \bar{a}$$

$$= \frac{1}{n} (\frac{\hat{a}_1 + \hat{a}_2}{2})^T (\frac{\hat{a}_1 + \hat{a}_2}{2}) - \frac{1}{n^2} (\frac{\hat{a}_1 + \hat{a}_2}{2})^T \mathbf{1}_{N,N} (\frac{\hat{a}_1 + \hat{a}_2}{2})$$

$$= \frac{1}{4n} (\hat{a}_1 + \hat{a}_2)^T (\hat{a}_1 + \hat{a}_2) - \frac{1}{4n^2} (\hat{a}_1 + \hat{a}_2)^T \mathbf{1}_{N,N} (\hat{a}_1 + \hat{a}_2)$$

$$= \frac{1}{4n} \hat{a}_1^T \hat{a}_1 - \frac{1}{4n^2} \hat{a}_1^T \mathbf{1}_{N,N} \hat{a}_1 + \frac{1}{4n} \hat{a}_2^T \hat{a}_2 - \frac{1}{4n^2} \hat{a}_2^T \mathbf{1}_{N,N} \hat{a}_2$$

$$+ \frac{1}{4n} 2\hat{a}_1^T \hat{a}_2 - \frac{1}{4n^2} 2\hat{a}_1^T \mathbf{1}_{N,N} \hat{a}_2$$

35
\[ \frac{\sigma_s^2}{2} + \frac{1}{2n} \left( \hat{a}_1^T \hat{a}_2 + \frac{1}{n} \hat{a}_1^T \mathbf{1}_{N,N} \hat{a}_2 \right) \]

We will define \( \mathbf{I}_r \in \mathbb{R}^{n \times n} \) as an anti-diagonal identity matrix, and make the simplifying assumption that, since \( \hat{a}_1, \hat{a}_2 \) are designed to match the target variances of the corresponding intact data vectors \( \mathbf{s}_1 \) and \( \mathbf{s}_2 \), they will have equal variances to those vectors. This assumption is true to within arbitrary tolerance, as defined by the SSLP algorithm, as we have shown in Section 4.5.2. Furthermore, since \( \mathbf{s}_1 \) and \( \mathbf{s}_2 \) are simply time-reversed mirror images of one another, \( \mathbf{s}_1 = \mathbf{I}_r \mathbf{s}_2 \). So the correct value for \( \sigma_s^2 \) is

\[ \frac{\sigma_s^2}{2} + \frac{1}{2n} \left( \mathbf{s}^T (\mathbf{I} + \frac{1}{n} \mathbf{1}_{N,N}) \mathbf{I}_r \mathbf{s} \right). \quad (4.9) \]

### 4.7 Handling Orphans

#### 4.7.1 Orphans Defined

An orphan occurs when either one or two positional points exist within a row of missing acceleration points (see Fig. 4.1). Due to the dependence of velocity and acceleration on the positional data, when a single position point is missing, it generates a “shadow” in the velocity vector of one extra missing point, and a larger shadow in the acceleration vector of two extra missing points. Owing to this, it is possible for some row \( \mathbf{a} \) to contain an arbitrary number of one- or two-point position clusters in its corresponding positional data. We define these stray position points as orphans. Note that, if three were present, the acceleration point would be available and rows would obviously be separate. If two positional points are present, we call these points 2-orphans. This situation is the easier of the two to deal with, since with two position points there is still a velocity target present. Thus, with careful indexing, we can
simply split the missing acceleration vector into two separate vectors, and estimate
according to the same procedure as a typical vector.

A singular extra position point is defined as a 1-orphan, and presents a more
difficult estimation problem if the constraint equations are to be satisfied. The chief
problem is that, with only one sample, there are no velocity samples present in the row,
and thus it is not possible to constrain using Eq. (4.3). While it would be possible to
simply omit any 1-orphans and interpolate between the ultimate start- and endpoints
of the acceleration row as with normal rows, this would result in nontrivial error
between predicted and actual positional point in an overwhelming majority of cases.
Thus, in this scenario, it becomes necessary to adopt a slightly different strategy.

4.7.2 Satisfying Equation (4.2)

With 1-orphans present, generating naive estimate $\bar{a}$ based on the available position
data and Eq. (4.2) is similar to the standard method, but requires some new definitions.
The main difference between this case and the standard case is that the assumption
$\bar{a} = \beta 1_{N+1}$ no longer holds. The vector $\bar{a}$ will instead be split up into $K$ sections,
where $K$ is the number of subsections created by any 1-orphans and $k \in \{1, \ldots, K\}$ is
a specific subsection. If a row has two 1-orphans, for example, it will be split up into
$K = 3$ sections. Rather than a single constant $\beta$ for all $\bar{a}$, each subsection of $\bar{a}$ is a
row of ones multiplied by $\beta_k$. Then, the indices of the orphan points become $N_k$, and
the orphan points themselves become $p_{N_k}$; in our example, the final position point of
this entire example row would be $p_{N_3}$.
Define $\kappa$ as the indices set of $\bar{a}$, then let $E^{(k)}$ be the selector matrix that selects the entries of $\kappa$ corresponding to subsection $k$. Then,

$$I^{(k)} = \begin{bmatrix} 0 & E^{(k)} \\ E^{(k)} & 0 \end{bmatrix}.$$ 

Additionally, we define $f_\ell(n)$ as a function that, given integer $n$, generates $\ell_{0_{N+1-n}} \in \mathbb{R}^{N+1}$, i.e., an $\ell$ based on the integer $n$, concatenated with the appropriate number of trailing zeros.

We start by re-writing Eq. (4.2) in terms of $p_{N_1}$,

$$p_{N_1} = p_0 + N_1 v_0 + \bar{a}^T(f_\ell(N_1))$$

$$= p_0 + N_1 v_0 + \bar{a}^T(I^{(1)}f_\ell(N_1))$$

$$= p_0 + N_1 v_0 + \beta_1(I^{(1)}f_\ell(N_1))^T 1_{N+1}.$$ 

Now, extending to $p_{N_2}$,

$$p_{N_2} = p_0 + N_2 v_0 + \bar{a}^T(f_\ell(N_2))$$

$$= p_0 + N_2 v_0 + (\beta_1 I^{(1)} + \beta_2 I^{(2)})f_\ell(N_2)^T 1_{N+1}.$$ 

Generalizing to $k$ and solving for $\beta_k$, we get

$$p_0 + N_k v_0 + \left( \sum_{i=1}^{k} \beta_i I^{(i)}f_\ell(N_k) \right)^T 1_{N+1} = p_{N_k}$$

$$p_0 + N_k v_0 + \left( \beta_k I^{(k)} + \sum_{i=1}^{k-1} \beta_i I^{(i)} \right)f_\ell(N_k)^T 1_{N+1} = p_{N_k}$$

$$\beta_k \left( I^{(k)}f_\ell(N_k) \right)^T 1_{N+1} + \left( \sum_{i=1}^{k-1} \beta_i I^{(i)}f_\ell(N_k) \right)^T 1_{N+1} = p_{N_k} - p_0 - N_k v_0.$$
Thus, for 1-orphans,

\[ \beta_k = \frac{p_{N_k} - p_0 - N_k v_0 - \left( \sum_{i=1}^{k-1} \beta_i I^{(i)} f^\ell(N_k) \right)^T \mathbf{1}_{N+1}}{(I^{(k)} f^\ell(N_k))^T \mathbf{1}_{N+1}}. \] (4.10)

Note that, since each value of \( \beta_k \) depends on the previous value in the row, \( \bar{a} \) is obtained iteratively by sequentially solving for each value of \( \beta_k \), and that for the first iteration, the summation term in the numerator need not be evaluated, as it is zero.

We are now able to express \( \bar{a} \) for 1-orphan rows as a function of \( \beta_1, \ldots, \beta_K \) as

\[ \bar{a} = \sum_{k=1}^{K} \beta_k I^{(k)} \mathbf{1}_{N+1}. \] (4.11)

### 4.7.3 Satisfying Equation (4.3)

To obtain \( \hat{a} = \bar{a} + \hat{w} \), we must again find a solution for perturbation vector \( \hat{w} \) that satisfies (4.3). However, our task is complicated by several facts. In order to ensure that \( \hat{a} \) still satisfies (4.2), we can no longer generate a single noise vector \( \bar{w} = w \circ \ell^{\circ-1} \), since in general, for the random samples in a given subsection \( k \),

\[ I^{(k)} \bar{w} \neq 0. \]

To deal with this, we will first re-write Eq. (4.2) in terms of \( \hat{a} \) and \( N_k \), using what we learned deriving Eq. (4.10), as

\[ \hat{a}^T \left( \sum_{i=1}^{k} I^{(i)} f^\ell(N_k) \right) = \hat{a}^T f^\ell(N_k) = p_{N_k} - p_0 - N_k v_0 \] (4.12)

In order to guarantee that \( \hat{a} = \bar{a} + \hat{w}_{\mid 0} \) or \( \hat{a}_0 = \bar{a} + \hat{w}_{\mid 0} \) still satisfies Eq. (4.12) for each \( p_{N_k} \in \{p_{N_1}, \ldots, p_{N_K}\} \), it is necessary to split \( w \) into \( k \) sections as with \( \bar{a} \). These sections are then normalized by their individual sample means so that the net effect
of each segment of \( \hat{a} \) on the estimates of any given \( p_{N_k} \) is still constrained by Eq. (4.12). We will demonstrate the procedure for \( \hat{w}_0 \), but other than the appropriate substitution of the last element it is unchanged for \( \hat{w}_\alpha \). For 1-orphans, we generalize our definition of \( \bar{w}_0 \) as

\[
\bar{w}_0 \triangleq \sum_{k=1}^{K} I^{(k)} f_t(N_k)(w_0 - \frac{1}{N_k - N_{k-1}} 1_{N_0}^T 1_{N_0}^{-1} w_0).
\]

Note that \( I^{(k)} f_t(n) = I^{(k)} 1_{N_0} \circ f_t(n) \). As before, this will continue to satisfy Eq. (4.2) regardless of any scaling of \( \bar{w}_0 \). We need \( \bar{G} \triangleq \sum_{k=1}^{K} \bar{\gamma}_k I^{(k)} \), where \( \bar{\gamma}_k \) are the scalars that allow \( \hat{a} \) to satisfy Eq. (4.3). Thus we have the definition

\[
\hat{w}_0 \triangleq \bar{G} \bar{w}_0,
\]

which is simply an orphan-friendly generalization of the earlier definition of \( \hat{w}_0 \).

Deriving scalars \( \bar{\gamma}_k \in \{ \bar{\gamma}_1, \ldots, \bar{\gamma}_K \} \) is a special challenge, since unlike the derivations for \( \beta_k \), we do not have internal constraints \( v_{N_k} \) to guide us; in actuality this is the defining trait of the 1-orphan problem. Our only constraint is Eq. (4.3), and we may have any arbitrary \( K \geq 2 \) of unknowns. This means there is no closed-form solution, and to find our set of \( \bar{\gamma}_k \) we must turn to alternate techniques.

Fortunately, this is possible to formulate as a convex optimization problem. First, using our generalized \( \bar{w}_0 \), we re-write Eq. (4.3) as

\[
\hat{a}^T 1_{N+1} = \Delta_v \\
(\hat{a} + \hat{w}_0)^T 1_{N+1} = \Delta_v \\
(\hat{a} + \bar{G} \bar{w}_0)^T 1_{N+1} = \Delta_v
\]
We would like to minimize $\epsilon_v$, defined as the mean squared error (MSE) between $\hat{a}^T \mathbf{1}_{N+1}$ and $\Delta_v$, where our argument is the matrix $\bar{\mathbf{G}}$. Therefore we solve

\[
\nabla \bar{\mathbf{G}} \left( (\hat{a} + \bar{\mathbf{G}} \mathbf{w}_{|0})^T \mathbf{1}_{N+1} - \Delta_v \right)^2 
= \bar{\mathbf{G}} \left( (\bar{\mathbf{w}}_{|0}^T \mathbf{1}_{N+1})^2 + (\mathbf{1}_{N+1} \bar{\mathbf{w}}_{|0}^T)^2 \right) + 2 \left( \mathbf{1}_{N+1} (\hat{a}^T \mathbf{1}_{N+1} - \Delta_v) \right) \bar{\mathbf{w}}_{|0}^T.
\]

A full derivation is presented in Section A.4.

Using this gradient, a Stochastic Gradient Descent (SGD) with $\ell_2$-regularization is conducted. It is important to note that, since $\bar{\gamma}_k$ are our noise scaling parameters, we need to generate a solution for $\bar{\mathbf{G}}$ that has stepwise values for $\bar{\gamma}_k$ corresponding to the proper indices for each row $k$. If we simply perform an SGD using $\mathbf{\bar{w}}$, a solution for $\mathbf{\bar{G}}$ will be arrived at that has different values at each index; such a solution would not be useful for scaling our noise vector $\mathbf{\bar{w}}$. Fortunately, the answer is fairly straightforward. We define a stepwise means vector $\mathbf{\bar{w}}_{|0} \in \mathbb{R}^{N+1}$ as

\[
\mathbf{\bar{w}}_{|0} \triangleq \sum_{k=1}^K \mathbf{I}^{(k)} \mathbf{w}_{|0} - \frac{\mathbf{1}_{N+1} \mathbf{I}^{(k)} \mathbf{w}_{|0}}{N_k - N_{k-1}}.
\]

Note that these means are generally nonzero, as they are the individual zero-mean segments of random vector $\mathbf{w}$ multiplied by $\mathbf{I}^{(k)} f_{\ell}(N_k)$.

Replacing $\mathbf{\bar{w}}_{|0}$ with $\mathbf{\bar{w}}_{|0}$, the cost function $L(\mathbf{\bar{G}})$ to be minimized is

\[
L(\mathbf{\bar{G}}) = \left( (\hat{a} + \mathbf{\bar{G}} \mathbf{\bar{w}}_{|0})^T \mathbf{1}_{N+1} - \Delta_v \right)^2.
\]

The update equation is

\[
\mathbf{\bar{G}}_{t+1} = \mathbf{\bar{G}}_t - \eta^* \left( \mathbf{\bar{G}} \left( (\mathbf{\bar{w}}_{|0}^T \mathbf{1}_{N+1})^2 + (\mathbf{1}_{N+1} \mathbf{\bar{w}}_{|0}^T)^2 \right) + 2 \left( \mathbf{1}_{N+1} (\hat{a}^T \mathbf{1}_{N+1} - \Delta_v) \right) \mathbf{\bar{w}}_{|0}^T - 2\lambda \mathbf{\bar{G}} - \theta \mathbf{V} \right).
\]
where
\[
V \in \mathbb{R}^{N+1 \times N+1} = \sum_{k=1}^{K} X \mathbf{I}^{(k)}, \quad X \sim \mathcal{U}(-0.5, 0.5).
\]
The regularizer \(2\lambda \bar{G} = \nabla \bar{G} \left( \lambda \|\bar{G}\|_F^2 \right)\). \(\bar{G}\) is initialized to \(I \in \mathbb{R}^{N+1 \times N+1}\) at step \(t = 0\).

Hyperparameters \(\eta\), \(\lambda\), and \(\theta\) are found using a standard grid search.

The gradient descent is performed as part of the Monte Carlo procedure from Section 4.5.2 where the individual rows of \(W_p\), previously candidates for \(\bar{w}_0\), here are candidates for \(\hat{w}_0\). Instead of a vector \(\bar{g}\) of scalar \(\bar{\gamma}\) candidates, we generate a matrix \(\bar{G}_d\) where each row is \(\bar{G}_{1N+1}\) for the \(\bar{G}\) corresponding to each row of \(W_p\). Then,

\[
\hat{a}^* = \arg\min_i \left[ \left( \text{var}(\bar{a} + \bar{G}_{d(i, :)} \circ \bar{W}_{p(i, :)} - \sigma_s^2) \right)^2 \right]. \tag{4.13}
\]
Chapter 5

Comparative Performance of Methods

NOTE: Although the combined process using SSLP as a pre-processor for a Kalman filter is not technically a distinct algorithm, for the purposes of expediency it will be referred to as such within this chapter.

5.1 Description of Evaluation Procedures

5.1.1 Real Dataset

The real dataset consisted of three complete tracks of 600 frames each, taken from a dataset similar to the bird flocking dataset used in [12]. The positional and blobsize information for these three tracks was generated by the GReTA algorithm’s image segmentation pre-processing. The specific tracks used were chosen due to being entirely unbroken trajectories, as well as containing likely occlusions and other data features. One thing lacking in this dataset was trajectories with sufficient curvature; indeed, all three tracks here consisted of the tracked object moving in a mostly straight line with some incidental variation. Therefore, while the three tracks contain realistic features, especially pertaining to the blobsize parameter, by themselves they are not sufficient for fully evaluating the performance of our track repair algorithms.
Figure 5.1: Real Track #3 (Trajectory: Bottom Right → Top Left), with no samples removed or additional processing.
5.1.2 Synthetic Dataset

The synthetic dataset was created to augment the real dataset for the purpose of testing the performance of the algorithms on tracks with varying amounts of curvature. The individual tracks in the synthetic dataset all start at an origin point of \((p_x, p_y) = (0, 0)\), and are generated with a constantly varying but roughly constant velocity, and curvature depending on input curvature parameter \(\rho\). Details of the generation method are presented in Section [A.3.1](#) but the ultimate result of the procedure is that any synthetic track can, in principle, contain regions with a high turning rate, but those with higher values for curve parameter \(\rho\) are very likely to have many more of these regions than those with lower \(\rho\). This was designed to avoid tracks with arbitrarily sharp turns, since ultimately the birds we are interested in tracking are subject to physical constraints in terms of ability to turn. Modeling and generation of the blobsize data is detailed in Section [A.3.2](#).

Figure 5.2: Sample synthetic tracks

5.1.3 Procedure

Owing to differing structure of the two different datasets, the testing procedure for our three algorithms differed depending on which dataset was used. An algorithm
was designed to generate different lengths of missing data rows in the middle of each track by removing datapoints. The algorithm was configured in such a way that each processed track would have a different makeup in terms of row lengths, and that these row lengths would themselves fall in different locations. Each of the three real tracks was pre-processed by this algorithm 100 separate times. This batch of $3 \times 100$ processed tracks was then fed to each algorithm. For the synthetic tracks, 100 tracks were generated for each curvature parameter $\rho$ in range $[1, 16]$, and each track was pre-processed once by the gap creation algorithm. Each batch of 100 processed tracks, 16 in total, was then fed to into each algorithm.

To evaluate the performance of the algorithms, the values predicted by each algorithm for each row of missing positional samples were extracted from the overall sequence output by each algorithm, and a 2-norm was calculated between each predicted $\hat{p}_x, \hat{p}_y$ and original track values $p_x, p_y$. This error-norm value will be referred to in this chapter as $\varepsilon$, not to be confused with $\epsilon$, which is a measure of the MSE between sample variances in the SSLP algorithm. The mean of some defined set of samples shall be denoted as $\mu_\varepsilon$, and the standard deviation of such a set will be denoted as $\sigma_\varepsilon$. These sample-wise norms and their associated statistics form the basis for our evaluation of the algorithms. The comparison was done between missing samples, rather than all samples, to foster fair comparison between the algorithms, since SSLP only predicts on missing samples, while the Kalman Filter generates an estimate for every datapoint in the track. SSLP, therefore, cannot have any error on intact datapoints, while the Kalman filter typically has a small amount of error on every sample. See Section 5.2 for more on this point.

For both filter types, owing to the dropout generation method, shorter rows are far more common than longer rows. This was by design, as was meant to emulate the structure of a dataset where short occlusions are more common than long dropouts, but both occur with some regularity. The distribution is evident in the row incidence.
bar graphs in Figs. 5.9 and 5.12 and in Table 5.1. Thus $\mu_\varepsilon$, the overall error mean for each algorithm, is more reflective of the error values for shorter rows. The longest dropouts, it should be noted, are close to 200 samples, which is a significant fraction of 600, the sample length of each track.

<table>
<thead>
<tr>
<th>Row length</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>all others</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean incidences per track</td>
<td>10.71</td>
<td>2.78</td>
<td>2.13</td>
<td>1.62</td>
<td>1.43</td>
<td>1.22</td>
<td>1.06</td>
<td>7.82</td>
</tr>
</tbody>
</table>

Table 5.1: Mean incidences of row length over 300 processed tracks, real dataset. Maximum row length is greater than 100. Statistics from the synthetic set are similar.

In general, $\sigma_\varepsilon$, the standard deviation of some collection of error measures, has been considered rather than the variance thereof. The error measure $\varepsilon$ for prediction versus ground truth is a measure of absolute error. Owing to this, and since error values can range from sub-pixel to hundreds of pixels in the worst cases, using standard deviation as a metric fosters a more intuitive comparison of results between methods.

5.2 Discussion of Results

We begin with a high-level overview of the head-to-head performance of the Kalman filter versus the SSLP algorithm in Section 5.2.1. We then present more data-driven evaluations of the Kalman filter in Section 5.2.2 followed by SSLP in Section 5.2.3. We finish in Section 5.2.4 with a treatment of the performance of the system where SSLP is used as a pre-processing front end for a Kalman filter. Each section will include information on tuned hyerparameters, where applicable.

Kalman filter configuration labeling is straightforward, as there are only two possible configurations, i.e., uni-directional and bi-directional. SSLP, however, was tested with 16 different configurations. The numbering convention used for these configurations is detailed in Table 5.2, and detailed descriptions of the different methods are presented in Section 4.5.1.
### Table 5.2: Key for labeling scheme of SSLP configurations on plots

<table>
<thead>
<tr>
<th>Type #</th>
<th>Perturbation Method</th>
<th>Bi-Directional?</th>
<th>Adjusted N + 1?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>normal</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>2</td>
<td>normal</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>3</td>
<td>normal</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>4</td>
<td>normal</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>5</td>
<td>split_rad</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>6</td>
<td>split_rad</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>7</td>
<td>split_rad</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>8</td>
<td>split_rad</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>9</td>
<td>uniform</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>10</td>
<td>uniform</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>11</td>
<td>uniform</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>12</td>
<td>uniform</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>13</td>
<td>norm_uniform</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>14</td>
<td>norm_uniform</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>15</td>
<td>norm_uniform</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>16</td>
<td>norm_uniform</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

**5.2.1 Kalman vs SSLP**

Before proceeding, it should be mentioned that a head-to-head error comparison between the Kalman filter and SSLP can be somewhat misleading, since part of what a Kalman filter does is attempt to mitigate noise present in measured data or in the process itself. Thus it is at an inherent disadvantage when it comes to directly measuring the error between its output and the underlying measurements. However, it will become clear, particularly upon analysis of the performance of the Kalman filter on the synthetic dataset, that there is still some value in making the comparison. The head-to-head results are presented in Figure 5.3, with selected numbers presented in Table 5.3.
Figure 5.3: Mean and standard deviation of error, Kalman vs SSLP. In terms of predictive error, SSLP outperformed the Kalman filters on both datasets.
Table 5.3: Head to head performance of selected SSLP types vs Kalman filters on synthetic data for selected values of $\rho$. Uni-directional Kalman filter used as baseline.

5.2.1.1 Accuracy of Predictions

The performance of both algorithms on the real dataset was generally quite good in terms of accuracy of prediction, as can be seen in Figure 5.5. The Kalman filter predicted within roughly a 5-pixel radius of the baseline sample value on average, and even the worst-performing SSLP type predicting within an average radius of roughly 3 pixels. For each SSLP type, the bi-directional version outperformed the uni-directional filter of the same type both in terms of $\mu_\epsilon$ and $\sigma_\epsilon$. Perhaps surprisingly, the bi-directional Kalman filter generally performed slightly worse than the uni-directional version for $\mu_\epsilon$, albeit with a lower $\sigma_\epsilon$. An example run of both Kalman types alongside the best performing fully stochastic SSLP type (i.e. excluding types based upon the “Split Rademacher” method, types 5-8) is pictured in Fig. 5.4.

The advantages of SSLP over the Kalman filter, in terms of accuracy of prediction, become clear when we examine the performance on the synthetic dataset, seen in Fig. 5.5. For each portion of the pictured track in which samples are missing, we can see that the unidirectional Kalman filter simply proceeds in the last known direction of the trajectory for as many timesteps as there are missing samples. Once data becomes available again, it snaps back onto the trajectory in what appears to be an exponential decay in $\epsilon$. The bi-directional Kalman filter, of course, exhibits the
Figure 5.4: Direct comparison of Kalman Filter to SSLP, real track #3, same gaps. Both algorithms can be seen making good predictions.
same behavior, albeit with the effects on $\varepsilon$ mitigated by the averaging between the forwards and backwards prediction steps. With no new samples, the Kalman state error covariance matrix $P_n$ does not have a realistic sense of how far off-track it is getting, and thus continues to predict a narrow confidence interval that does not at all account for the error. Interestingly, as we can see on the right side of Subfigure 5.5d, the averaging of the bi-directional Kalman can actually detract from the accuracy of predictions on steps with intact sample values. This effect does not show up in our measures of $\mu_\varepsilon$ or $\sigma_\varepsilon$, but it must be accounted for when employing a bi-directional Kalman filter.

Where the Kalman filters fail in this example, we can clearly see SSLP succeeding. The predicted track sticks closely to the trajectory of the ground-truth track. It should be noted that any difference in forward velocity is not clearly visible in these plots, though in this dataset, as it would also be generally true with flying birds, the variation from the mean flying speed is not excessive, and thus likewise, this effect should be minimal. Depending on track curvature parameter $\rho$, the worst-performing variant of SSLP outperforms the best-performing Kalman filter by a factor of between 3 and 5 in terms of $\mu_\varepsilon$, though it should be noted that the gains are less dramatic in terms of $\sigma_\varepsilon$.

### 5.2.1.2 Execution Time

Execution times were measured using Python’s `time` module, with the timer started at the beginning of the prediction step and ended after the step has finished but before the algorithm instance has tabulated the data post-prediction. This was done to minimize overhead from implementation details, hopefully to provide as pure a measurement as possible for the algorithm itself.

As we can see in Fig. 5.6, the one area in which the Kalman filter greatly outperforms SSLP is in the typical execution times for one track. The Kalman filter operates
Figure 5.5: SSLP results, sample synthetic track. Medium curve parameter ($\rho = 8$). Kalman filter predictions diverge wildly from the ground truth when a gap coincides with a curve. SSLP performs well.
recursively, using the same number of matrix operations for each timestep with measurement data. This is reflected in the low deviation from the mean times; the Kalman filter takes close to the same amount of time for every track, regardless of which dataset is being predicted upon. For timesteps without measurement data, it is even faster, as it is able to skip the measurement update. Fascinatingly, the bi-directional Kalman filter does not require twice the execution time as the unidirectional one. This is likely due to optimizations performed by the Python engine, and as such is largely out of the control of the author.

![Execution Times, All Types, Real Data](image1)

![Execution Times, All Types, Synthetic Data](image2)

**Figure 5.6**: Mean execution times across all runs, both algorithms. The Kalman filter is between 1 and 3 orders of magnitude faster than SSLP, with the problem exacerbated on the synthetic dataset.

By contrast to the Kalman Filter, the number of calculations for SSLP varies greatly depending upon the composition of the data, and is therefore highly non-linear with respect to the number of samples present. While in the simplest cases it is able to make predictions based on some batch number of closed-form solutions, other scenarios create extra computational penalties; as we have seen, the procedure for calculating rows containing 1-orphans even includes a Stochastic Gradient Descent. We recall that a defining characteristic of SSLP is that it attempts to match sample variances...
between its row predictions and the surrounding data. The extra overhead involved in handling different types of scenarios can prove costly in terms of total computation, depending chiefly on the difficulty achieving this match within the specified tolerance for MSE, which in our case was set to $\epsilon = 0.1$. As expected, in each instance the bi-directional SSLP types took roughly twice as long as the unidirectional ones.

The difference in computational requirements between SSLP and the Kalman filter is clearly represented in terms of execution times. As we can see in Table 5.4, the mean execution time, expressed as a multiple of the Kalman filter’s execution time, varies between a factor of 29 in the best case and over 1800 in the worst. The times also have large measures for $\sigma_\epsilon$, supporting the assertion that execution time for SSLP is highly sensitive to the composition of the data present.

Particularly surprising is the markedly worse performance in terms of execution time of SSLP on the synthetic dataset compared to the real one. One possible explanation is that, while the real dataset has noisy data that is easy for the algorithm to match, the synthetic set was not created with any additive noise; any segments resembling noise are a result of random fluctuations in the generating algorithm, while the tracks generated are typically quite smooth, as pictured in Fig. 5.7. This could potentially make it difficult for the perturbation step of SSLP to match variances, particularly in more pathological cases where a gradient descent is required. The general lack of additive noise in the synthetic set may also conceivably affect the performance of the Kalman filter, which is “expecting” the noise in the source to match the measurement noise characteristics built in to the model. While the addition of noise to the synthetic set would require great care to avoid biasing the results in favor of one method or another, it might prove instructive.

It should be noted that there is a strong possibility that more 1-orphans are present, in both the synthetic dataset and the real dataset with samples artificially removed, than would be present in a real dataset whose missing samples arise organically as a
result of occlusions or other issues. If true, this would suggest that the performance hit of using SSLP in a realistic setting might not be as severe as indicated by these results.

5.2.2 Kalman Filter

5.2.2.1 Results

The Kalman filters were tested according to the procedure outlined in Section 5.1.3. One confounding aspect of the Kalman filter results is the worse performance, in terms of $\mu_\varepsilon$, of the bi-directional filter on the real dataset versus the uni-directional filter, particularly in light of its markedly better performance on the synthetic dataset, as seen in Fig. 5.8. It is difficult to posit an explanation for this fact, other than that the process model for the Kalman filter seems to be very well-suited to tracks that proceed in mostly straight lines. With a bi-directional filter, the chances increase that a gap is initiated during a noisy event, and thus one side of the track, with it half the averaged data, are thrown into question. Perhaps if the filter outputs of the forwards and backwards pass of the bi-directional filter were weighted according to some metric, better results might have been obtained.
<table>
<thead>
<tr>
<th>Real Data</th>
<th>Type</th>
<th>Mean</th>
<th>Mean factor × baseline</th>
<th>Std. Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Kalman-Uni</td>
<td>0.22</td>
<td>1.00</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>Kalman-Bi</td>
<td>0.36</td>
<td>1.65</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>SSLP #1</td>
<td>9.76</td>
<td>45.18</td>
<td>12.89</td>
</tr>
<tr>
<td></td>
<td>SSLP #2</td>
<td>18.63</td>
<td>86.27</td>
<td>22.66</td>
</tr>
<tr>
<td></td>
<td>SSLP #3</td>
<td>9.03</td>
<td>41.81</td>
<td>15.48</td>
</tr>
<tr>
<td></td>
<td>SSLP #4</td>
<td>16.49</td>
<td>76.37</td>
<td>23.34</td>
</tr>
<tr>
<td></td>
<td>SSLP #5</td>
<td>6.17</td>
<td>28.56</td>
<td>9.53</td>
</tr>
<tr>
<td></td>
<td>SSLP #6</td>
<td>11.65</td>
<td>53.95</td>
<td>16.12</td>
</tr>
<tr>
<td></td>
<td>SSLP #7</td>
<td>6.77</td>
<td>31.35</td>
<td>10.59</td>
</tr>
<tr>
<td></td>
<td>SSLP #8</td>
<td>13.06</td>
<td>60.50</td>
<td>18.04</td>
</tr>
<tr>
<td></td>
<td>SSLP #9</td>
<td>15.49</td>
<td>71.73</td>
<td>21.88</td>
</tr>
<tr>
<td></td>
<td>SSLP #10</td>
<td>28.57</td>
<td>132.31</td>
<td>37.28</td>
</tr>
<tr>
<td></td>
<td>SSLP #11</td>
<td>13.35</td>
<td>61.82</td>
<td>20.86</td>
</tr>
<tr>
<td></td>
<td>SSLP #12</td>
<td>23.14</td>
<td>107.15</td>
<td>30.01</td>
</tr>
<tr>
<td></td>
<td>SSLP #13</td>
<td>11.46</td>
<td>53.08</td>
<td>15.01</td>
</tr>
<tr>
<td></td>
<td>SSLP #14</td>
<td>20.19</td>
<td>93.48</td>
<td>24.67</td>
</tr>
<tr>
<td></td>
<td>SSLP #15</td>
<td>10.13</td>
<td>46.93</td>
<td>15.07</td>
</tr>
<tr>
<td></td>
<td>SSLP #16</td>
<td>17.06</td>
<td>79.00</td>
<td>21.77</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Synthetic Data</th>
<th>Type</th>
<th>Mean</th>
<th>Mean factor × baseline</th>
<th>Std. Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Kalman-Uni</td>
<td>0.28</td>
<td>1.00</td>
<td>0.06</td>
</tr>
<tr>
<td></td>
<td>Kalman-Bi</td>
<td>0.53</td>
<td>1.87</td>
<td>0.09</td>
</tr>
<tr>
<td></td>
<td>SSLP #1</td>
<td>158.26</td>
<td>561.82</td>
<td>964.87</td>
</tr>
<tr>
<td></td>
<td>SSLP #2</td>
<td>355.71</td>
<td>1262.72</td>
<td>1798.14</td>
</tr>
<tr>
<td></td>
<td>SSLP #3</td>
<td>208.86</td>
<td>741.43</td>
<td>1028.24</td>
</tr>
<tr>
<td></td>
<td>SSLP #4</td>
<td>481.74</td>
<td>1710.10</td>
<td>1635.36</td>
</tr>
<tr>
<td></td>
<td>SSLP #5</td>
<td>120.64</td>
<td>428.24</td>
<td>690.48</td>
</tr>
<tr>
<td></td>
<td>SSLP #6</td>
<td>279.16</td>
<td>990.98</td>
<td>1499.69</td>
</tr>
<tr>
<td></td>
<td>SSLP #7</td>
<td>120.42</td>
<td>427.46</td>
<td>833.87</td>
</tr>
<tr>
<td></td>
<td>SSLP #8</td>
<td>236.36</td>
<td>839.05</td>
<td>1173.18</td>
</tr>
<tr>
<td></td>
<td>SSLP #9</td>
<td>257.40</td>
<td>913.72</td>
<td>379.51</td>
</tr>
<tr>
<td></td>
<td>SSLP #10</td>
<td>514.50</td>
<td>1826.40</td>
<td>800.04</td>
</tr>
<tr>
<td></td>
<td>SSLP #11</td>
<td>123.19</td>
<td>437.31</td>
<td>450.96</td>
</tr>
<tr>
<td></td>
<td>SSLP #12</td>
<td>249.38</td>
<td>885.26</td>
<td>641.81</td>
</tr>
<tr>
<td></td>
<td>SSLP #13</td>
<td>139.75</td>
<td>496.08</td>
<td>659.53</td>
</tr>
<tr>
<td></td>
<td>SSLP #14</td>
<td>260.16</td>
<td>923.54</td>
<td>1024.55</td>
</tr>
<tr>
<td></td>
<td>SSLP #15</td>
<td>152.98</td>
<td>543.05</td>
<td>542.52</td>
</tr>
<tr>
<td></td>
<td>SSLP #16</td>
<td>271.65</td>
<td>964.31</td>
<td>712.16</td>
</tr>
</tbody>
</table>

Table 5.4: Execution time, in seconds, statistics for a single track, Kalman filter versus SSLP. Baseline mean time for each dataset is the uni-directional Kalman filter.
Figure 5.8: Mean and standard deviation of MSE for both Kalman filter types. The uni-directional filter outperforms the bi-directional filter on the real dataset, but is significantly outperformed on the synthetic set.

Uni-directional versus bi-directional Kalman filter performance is more in line with expectations on the synthetic dataset. While it should be noted that both versions of the filter perform rather poorly, the bi-directional filter outperforms the uni-directional filter for every value of $\rho$ tested. Additionally, a mostly monotonic increase, albeit a noisy one, is visible in both $\mu_\epsilon$ and $\sigma_\epsilon$ as $\rho$ is increased.

Another surprising trend can be seen in Fig. 5.9. One would expect the prediction error to worsen on average as the lengths of missing rows increase. However, in the real dataset, after a slight increase in the mean errors for rows 0-15, the error actually decreases slightly as the row lengths increase. This trend is clearly visible despite the increasing noise in the data for higher row length values, which happens due to the decrease in available incidences of each row length. A likely explanation for this is the fact that the process model for this particular Kalman filter appears to be optimal for tracking objects that tend to move in straight lines.

For the synthetic dataset, there is a definite upwards trend as the row lengths increase, apparently following some sort of curve. It is tempting to read into this, but it is also very possible that this simply reflects the straight lines of the Kalman filter.
predictions as the curves of the original, intact tracks pull away from them the longer a row of missing data continues. Of interest are the lines representing the mean values for data with different ranges for $\rho$. While there clearly is some positive correlation between higher values of $\rho$ and higher values of error, the effect is far from linear. Quite likely this reflects the fact that $\rho$ does not directly determine the maximum amount of curvature in a plot, but rather the likelihood that a track has many regions where the turn rate is changing. By way of further explanation, we can imagine a track that suddenly begins traveling in circles one sample after a row of missing values begins. As the Kalman prediction flies off in a straight line, $\varepsilon$ will continue to increase in a sinusoidally fluctuating manner. If we contrast this with a similar track that instead makes a sharp 180° turn then continues in a straight line in the other direction, we can begin to intuit why, past a certain point, the effect of higher $\rho$ values on $\mu_{\varepsilon}$ begins to plateau.
5.2.2.2 Hyperparameter Search

For the real dataset, values of $\varrho = 0.7$ and $\alpha = 1$ were chosen. For the synthetic set, $\varrho = 0.1$, and $\alpha$ was set to 0.998 and 0.999 for the uni-directional and bi-directional filters, respectively. Details of the parameter search are found in Section A.1.

5.2.3 SSLP

5.2.3.1 Results

The overall performance of SSLP on both datasets can be seen in Figure 5.11. In terms of $\mu_\varepsilon$, the performance was generally quite good. On the real dataset, the best-performing SSLP algorithms were able to beat the uni-directional Kalman filter by a factor of better than 4, while the worst-performing one, type #3, still performed nearly twice as well, although it did have worse figures for $\sigma_\varepsilon$. In every case, the bi-directional SSLP type outperformed the uni-directional version with otherwise identical parameters by a modest margin. Generally speaking, the versions without specially adjusted $N + 1$ outperformed the versions in which adjustments were made. The Split Rademacher variant in particular performed quite well in all configurations.

Predictive performance over the synthetic set was excellent when compared to that of the Kalman filter, with the worst-performing variant, #15, outperforming the bi-directional Kalman filter by a factor of between 4 and 5.5, depending on the value of $\rho$. Across all values of $\rho$, the best performing configurations outperformed the bi-directional Kalman filter by a factor of roughly 7. There was remarkable consistency between the two datasets of the performance of the different types; for the most part $\mu_\varepsilon$ followed the same pattern reading from left to right across the algorithm variants. Like the Kalman filters, there is a trend of increasing $\mu_\varepsilon$ and $\sigma_\varepsilon$ as $\rho$ increases, though the effect is not as consistent across all configurations as with the Kalman filter.

One surprising aspect of these results was the standout performance of all Split
Figure 5.11: Mean and standard deviation of error, all SSLP types. SSLP modestly outperformed the Kalman filter on the real dataset, and significantly outperformed the Kalman filter on the synthetic set. On the synthetic set there was divergent performance between the different SSLP types.
Rademacher variants of the algorithm (types 5-8). As detailed in Section 4.5.1, this perturbation method enforces an even split between $\pm 1$ in the $w$ vector before scaling. This means that, out of all the perturbation options detailed in this work, it is the least stochastic; the only thing random about $w$ is the specific order of positives and negatives. While this may be a disadvantage in settings where matching the intrinsic noise characteristics of the source track is important, it should be considered as an option in situations where this does not matter, particularly in light of its generally superior performance in terms of execution time.

Another surprising result was the comparatively poor performance, across all variants of the algorithm, of the types that adjusted for random values of $\hat{a}_{N+1}$ (types 3,4,7,8,11,12,15,16). Recall from Section 4.5.3 that this adjustment is performed in order to improve upon performance, particularly for $a$ vectors comprised of three samples, i.e., rows of missing positional data with a length of one. Since a large fraction of the row incidences were one sample in length (Figs. 5.9 and 5.12), this flies directly in the face of expectation. As of this writing, the cause of this phenomenon is unclear, and it merits further investigation.

On both datasets, SSLP exhibits what appears to be a nearly linear relationship between row length and $\mu_\varepsilon$, though it should again be noted that, with fewer incidences of longer rows, the data becomes increasingly noisy as the row length increases. On the real dataset, even the longest rows appear to have values for $\mu_\varepsilon$ of less than 10px on average, although there are enough outliers to suggest that some algorithms performed significantly better on these long rows than others. We recall that the sample rate utilized for this dataset was 155Hz, meaning that rows of $\geq 100$ samples correspond to two thirds of a second or longer, a significant amount time for a bird in flight or some other fast-moving object.

On the synthetic dataset, the $\mu_\varepsilon$ figures achieved for rows of $\geq 100$ samples are generally comparable to the numbers achieved by the Kalman filters for rows of
Figure 5.12: Row length vs error norm, SSLP, all types. Each dot represents the mean results per filter type.

Each dot in (a) represents the mean results per filter type, while each dot in (b) represents results averaged across all types for a particular value of $\rho$. For both datasets, SSLP exhibits a roughly linear correlation between row length and $\varepsilon$. SSLP is insensitive to $\rho$.

approximately 30 samples. Interestingly, there is no obvious relationship between the value of $\rho$ and the correlation between error and row length; all of the lines appear to have roughly the same slope. This indicates that the SSLP algorithm is, on average, insensitive to the amount of curvature present in a track. This linear relationship is not wholly unexpected, since SSLP, far more than our Kalman implementation, attempts to generate paths with an amount of curvature which, while not necessarily a perfect match for the missing data, will at least result in emergence at the other end of the row with the same heading. This is the theoretical guarantee made by satisfying SSLP constraint equation (4.3). On a shorter row, where all the curvature is likely to be in the same direction, this generally results in close fits, while on longer rows where curvature may potentially exist in several directions, or an object’s forwards velocity may change significantly, there is more potential for error to accumulate.
5.2.3.2 Hyperparameter Search

The values ultimately used for SGD hyperparameters were $\eta = 4 \times 10^{-2}$ for the learning rate, $\lambda = 10^{-4}$ for the regularization parameter, and $\theta = 5 \times 10^{-3}$ for the noise multiplier. The descent stopping criterion $\epsilon = \left( \hat{a}^T \mathbf{1}_{N+1} - \Delta v \right)^2$ was set to a value of $10^{-1}$. The general Monte Carlo MSE threshold $\epsilon$ was set to a value of $\epsilon = 10^{-3}$. Details of the parameter search are presented in Section A.2.

5.2.4 Combined System

For the combined system, with SSLP used as a pre-processing front-end for the Kalman filter, a similar procedure was used as for both systems individually. In fact, the results from the SSLP testing round were fed directly into both uni-directional and bi-directional filters, producing a total of 32 possible testing configurations. By inspection of Fig. 5.14, we can immediately see a marked improvement over the performance of either unaided Kalman filter. The predictions, as expected, no longer have the problem of veering well off-track upon hitting a gap, simply because there are no gaps to hit. It is evident, however, that the filter is generally unable to match some of the more pronounced rates of curvature present in the tracks. The bi-directional filter suffers less from this effect than the uni-directional filter.

In Figure 5.15, we see the error statistics for the combined system, where types 1-16 as displayed are the uni-directional Kalman filter paired with SSLP types 1-16, and 17-32 as displayed are the bi-directional Kalman filter paired with SSLP types 1-16. On the real dataset, the combined system achieved a modest improvement, if any, over the uni-directional filter alone with between 0.5-1px reductions in error across the range of SSLP configurations. The bi-directional Kalman filters achieved a slightly better improvement over the bi-directional filter alone, with error reductions between 1-1.5px. The values for $\sigma_\epsilon$ mostly match up with the variance values for
Figure 5.14: Combined system on synthetic track samples. SSLP type #10 (“Uniform” method), mixed Kalman types. The combined system performed significantly better than the Kalman filter alone, but did not match the performance of SSLP alone.
Figure 5.15: Mean and standard deviation of error, combined systems. The system was marginally better than the Kalman filter on the real dataset, and significantly better on the synthetic set. On both sets, the combined system was outperformed by SSLP alone.

SSLP, generally improving upon them slightly.

On the synthetic set, the combined system performed far better than the Kalman filter alone, and showed worse performance in general than SSLP alone. The best-performing version with the uni-directional Kalman, which used SSLP type #2, outperformed the corresponding Kalman filter alone by factors of between 2 and 2.4, depending on the value of \( \rho \). The best performing bi-directional system, which also used SSLP type #2, outperformed the corresponding Kalman filter by similar factors. In general, the \( \mu_\epsilon \) values of the combined system closely tracked those of the SSLP algorithm, which is not surprising. Interestingly, both versions of the Kalman tended to level off the differences in \( \mu_\epsilon \) between the various SSLP types to some degree.
Figure 5.16 shows the results of row length vs $\mu_\varepsilon$ for both the real and synthetic datasets. The combined system on the real dataset shows a similar trend to that of SSLP alone, but with an additional offset that starts in roughly the same place as the Kalman filter. On the real dataset, unfortunately, the combined system appears to inherit the worst characteristics of both systems; it shows a similar linear increase to the SSLP but without its excellent performance on short rows, and likewise it has similar performance over short rows as the Kalman filter but trends upwards in terms of error rather than downwards as the row lengths increase. On the synthetic dataset, the combined system fares better than on the real when compared to the other systems. It shows similar characteristics to the Kalman filter on short rows, including the saturated insensitivity to an increase in $\rho$. However, the error increases far slower as row lengths increase, where it appears to assume the characteristics of SSLP, but with slightly worse $\mu_\varepsilon$. That the combined system should look more like SSLP at high row lengths is hardly surprising; the error balloons with the unaided Kalman filter precisely because, with no inputs, it tends to continue in a straight line. In the combined system, the Kalman has input values that are at least somewhat tethered to the actual measurements.
Figure 5.16: Row length vs error measure, combined systems. The combined system appeared to inherit the worst characteristics of both algorithms on the real dataset, but performs better on the synthetic set.
Chapter 6

Conclusions and Future Directions

This thesis has presented several approaches to the problem of repairing trajectories in datasets with two-dimensional positional information, as well as providing frameworks that could be extended to data in a three-dimensional space with minimal effort. The ultimate goal of this work was to be able to use one or both of these approaches in order to match broken trajectories in existing datasets. The approaches detailed each require some further work before they can be effectively applied in this setting, however, they do represent important first steps towards this goal. We summarize the key contributions of this thesis, and propose future work for each.

6.1 The Kalman Filter

We have presented a novel application of a Kalman filter, using it chiefly as a predictor for rows of missing sample data. The implementation included a new state space model based on the Kalman filter presented in [16] and modified using ideas from [32]. We have also included a simple method, the bi-directional Kalman filter, for easily increasing accuracy of our standard Kalman filter in certain settings, and provided empirical data to demonstrate the effectiveness of the two different approaches depending upon the composition of the data. A key finding was that, given tracks from a dataset consisting of birds flying in uniform directions and a synthetic dataset in which objects are modeled that spend significant time turning, and assuming gaps in the data of any significant length, it is far less effective predicting on the synthetic set.
It is quite possible that a major source of weakness in this specific implementation of the Kalman filter was its reliance on Cartesian $x, y$ coordinates. A state-space model based on a velocity-heading representation of the data might conceivably prove to be a much more able model for diverse types of data, particularly as it would be able to keep track of turning rates and changes in turning rate. Another, possibly complementary, possibility is that of filling trajectory gaps after the data has been projected from the stereoscopic 2D space to the 3D space, as detailed in [11]. This should improve performance in cases where the 2D representation appears to demand rates of turn that do not match well to those of the 3D reality, for example if a bird is turning in a flat circle contained in a plane with one axis parallel to the major axis of a camera.

The bi-directional filter approach also has great potential for improvement. The simple averaging method employed does not take into account the number of samples from the previous intact datapoint, for example, nor does it make use of the Kalman filter’s built-in uncertainty measure or state error covariance matrix. An approach that weights the predictions of the forward-time filter run against those of the reverse-time filter using these metrics, analogous to the approach in [21], could potentially yield significant performance gains without sacrificing much computational efficiency.

Perhaps the main issue with the Kalman filter as implemented was that the specific implementation used was better suited to solving prediction problems, whereas our problem formulation really poses an interpolation problem. It is likely that a fairer comparison could have been made using a smoothing algorithm, such as the one described in Section 3.6.
6.2 SSLP

Built from basic characterizations of discrete-time velocity and acceleration that also form the basis of the Kalman filter implementation presented, we have developed a novel algorithm, Stochastic Straight-Line Perturbation (SSLP). The algorithm uses foreknowledge of the target endpoint of a given missing data row, as well as a measure of the variance of intact data surrounding the row, to interpolate an estimated sequence of positional points, filling in the gaps in the data. We have presented an empirical evaluation demonstrating that SSLP outperforms the Kalman filter implementation presented, in some cases by very large margins, particularly in tracks with significant curvature. In many cases, the predictions can be generated efficiently, using a combination of closed-form solutions and stochastic Monte Carlo methods implemented as matrix operations. However, there exist pathological cases requiring a stochastic gradient descent, which can prove quite costly in terms of execution time.

A fact of the application of SSLP for joining track fragments in a large dataset is that the endpoint of a specific object may not be known in advance. In fact, by the definition of this problem, there should be a number of possible candidates for possible endpoints. In cases such as these, it may be possible to employ SSLP combined with methods in [15]. Crucially, to use SSLP in the context of a dense dataset with many missing or ambiguous samples and several possible endpoints per object, its computational efficiency would need to be improved. The greatest gains would likely be in dropping the requirement that positional and velocity estimates for orphaned single positional points surrounded by missing data be approximated exactly. By tolerating a small amount of extra error, the need for a gradient descent could be elided, radically speeding up the algorithm and potentially making it computationally competitive with the Kalman filter. Short of this, it may be possible to refine the
hyperparameter tuning based on curvature present in the track, or to loosen the stopping criteria somewhat, without significant adverse effects on $\mu_\varepsilon$.

While the predictive performance of SSLP is promising, there is still room for improvement. Similarly to the Kalman filter, it should be possible to implement in a velocity-heading representation in two or three dimensions, with similar gains. One issue the current algorithm has is that it uses the assumption of constant acceleration to form the basis of its initial unperturbed acceleration row estimate. This could likely be improved upon by enforcing that both ends of the initial acceleration estimate match some idealized endpoint of the extant acceleration data, perhaps by low-pass filtering it in some way, or perhaps even through the use of another Kalman filter.

The noise-matching ability of SSLP could also be improved upon. At present, the algorithm does not account for situations in which $s$, the existing acceleration data surrounding a gap, may have slow moving, long-term trends underlying the noisy sample-to-sample measurements. This could affect the measured sample variance, $\sigma_s^2$, that the perturbation step is attempting to match. Applying some form of high-pass filtering to the data in $s$ may improve the noise-matching performance, and also conceivably open up some understanding of whether the noise added in the perturbation step should exhibit some form of heteroskedasticity in order to match differing variances on either side of the gap.

In the future, it would also likely be instructive to compare SSLP to existing interpolation methods, such as the elastica mentioned in Chapter 2 or the method of spline interpolation. The latter method may use polynomials of arbitrary order, and is optimal in the sense that it will minimize curvature of the resulting track output. Whether SSLP stacks up favorably to such a method or not, it should be an instructive step, as it may be a good way to characterize to what extent the flight behavior of the agents in question is optimal in this least-curvature sense.
Glossary of Kalman Filter Terms

7.1 Glossary of Terms for Kalman Filter Elements

- $x_{n-1}$: System state at previous timestep, i.e., the output of the Kalman filter at timestep $n - 1$
- $x_n^-$: Prior state estimate for timestep $n$
- $x_n$: Posterior state estimate for timestep $n$
- $\alpha$: Autoregression constant
- $\varrho$: Blobsize effect/uncertainty constant
- $F$: State transition matrix
- $H$: Measurement transition matrix
- $w_n$: Process noise for timestep $n$
- $P_n^-$: Prior state error covariance estimate
- $P_{n-1}$: State error covariance at previous timestep
- $P_n$: State error covariance for timestep $n$
- $Q_n$: Process noise covariance matrix
- $R_n$: Measurement noise covariance matrix
- $z_n$: State measurement for timestep $n$
- $e_n$: Innovation for timestep $n$
- $U_n$: State uncertainty measure
- $K_n$: Kalman gain
Bibliography


A.1 Tuning $\varrho$ and $\alpha$ For the Kalman Filter

Given the process and noise models as described in Sections 3.3.1 and 3.4, the Kalman filter was tuned individually to the real and synthetic datasets by finding optimal values for autoregression parameter $\alpha$ and blobsize/uncertainty effect parameter $\varrho$. First, $\varrho$ was tuned using the logic outlined in 3.4; namely, that the error norm on these small objects should probably not amount to much more than 2px most of the time. Therefore, for a given dataset, $\varrho$ was adjusted until, for all blobsizes $z_b$ in a dataset, the mean of $\varrho z_b$ plus the standard deviation of $\varrho z_b$ was approximately $\sqrt{2}$.

An interesting fact is that the error measure actually continues to decrease as the $\varrho$ approaches zero. It is thus tempting to simply set $\varrho = 0$. However, this is equivalent to telling the Kalman filter that every incoming measurement is perfect with no noise, and thus we lose an important effect of the Kalman filter, namely, its ability to predict on noisy data. For the real dataset, $\varrho$ was set to 0.7, and for the synthetic dataset, $\varrho$ was set to 0.1.

With $\varrho$ determined, $\alpha$ was adjusted. This was accomplished using a simple grid search approach, which we will call a line search as it was conducted along a single dimension. The value chosen for $\alpha$ was the value that minimized, in terms of priority, $\mu_\epsilon$ first. However, in some cases, it happened that a certain value for $\alpha$ resulted in an incrementally smaller error but a rapidly ballooning variance. In these situations $\alpha$ was chosen for minimal variance. Interestingly, for the real dataset, the optimal value for $\alpha$ was found to be 1, meaning it had no effect on the output. For the synthetic dataset, optimal $\alpha$ was found to be 0.998 and 0.999 for the uni-directional and bi-directional filters, respectively.

A.2 Tuning Error Tolerance and SGD Hyperparameters for SSLP

Like the Kalman filter, the SSLP algorithm relies on several hyperparameters, mostly pertaining to the gradient descent performed in the case on 1-orphans, as detailed in Section 4.5.2. Hyperparameters $\eta$, $\lambda$, and $\theta$ are found using a standard grid search for parameters that minimize descent time while maintaining stability. The values ultimately used were $\eta = 4 \times 10^{-2}$ for the learning rate, $\lambda = 10^{-4}$ for the regularization parameter, and $\theta = 5 \times 10^{-3}$ for the noise multiplier. Descent stopping criterion $\epsilon_v = \left(\hat{a}^T 1_{N+1} - \Delta_v\right)^2$ was set to a value of $10^{-1}$. This parameter is distinct from $\epsilon$, which is the target MSE between $\sigma^2_{\hat{a}}$ and $\sigma^2_{s}$, which was set to a more exacting threshold of $10^{-3}$.
These parameters were tuned while developing the algorithm for use on the real dataset. While they did reliably find solutions over the synthetic dataset, it is possible that better performance in terms of execution time might be achieved if the synthetic set was included in the parameter tuning.

A.3 Details on the Generation of the Synthetic Dataset

A.3.1 Generating Curves

This synthetic dataset was created for the purpose of evaluating the characteristics of the filters on a large number of tracks with varying curvature. Thus, a method was needed to generate a dataset with these characteristics. First, a row of velocity value is generated, distributed as $\mathcal{N}(\mu_v, (\frac{\mu_v}{5})^2)$, where $\mu_v$ is a constant value.

The 600 samples of the tracks are separated into several regions of random length $k$. The turning rate of the track, i.e. $\Delta \phi_n$, where $\phi_n$ rad/sample is the instantaneous heading of the object, is generated from turn rate change values $a_\phi$ according to the following algorithm. First, a row of $k$ random values $\mu_{a_\phi k}$ is generated, distributed as $\sim \mathcal{N}(\mu_v, (\rho \mu_v)^2)$, where $\rho$ is the curve parameter. These are in turn used to generate $k$ rows of actual $a_\phi$ values, distributed as $\sim \mathcal{N}(\mu_{a_\phi k}, 50^2)$. The mean of each row is removed, the values are normalized by the maximum, and then they are multiplied by a maximum value of turning rate for a given object is limited to $3\pi$ rad/s, or $\frac{3\pi}{f_s}$ rad/sample. Finally, they are concatenated together, and the cumulative sum of their cumulative sum becomes a row of values for $\phi_n$.

A.3.2 Assigning Synthetic Values for the “blobSize” Parameter

One major challenge for generating the synthetic dataset was the question of how to properly generate blobsize information, which, as detailed in Section 3.4, is reflective of the number of pixels for a given object as interpreted by the segmentation algorithm, and is thus a convenient, dynamic measure of uncertainty. Blobsize should be expected to change with three major factors, namely, the flapping of a bird’s wings, the distance of the bird from the observer, and the orientation of the major axis of the bird’s body with respect to the observer. We would expect the patterns of wing flapping to be related to the bird’s behavior; a change in direction might be expected to correlate with soaring, i.e., less flapping, while a period of straight-line acceleration might be accompanied by a period of vigorous flapping. Such close modeling of avian flight dynamics is, however, beyond the scope of this work.

Instead, we have focused on simply trying to match the general statistical patterns found in the blobsize data from the real dataset. First, we note that blobsize data is strictly limited to the natural numbers, as it is a measure of the number of pixels constituting an individual object. Thus we also exclude zero, since any segmentation algorithm should not be able to recognize an object of size zero as an object. We know that in our particular dataset, the flock of birds is flying in a roughly orthogonal direction to the main axis of the camera, and at a significant distance on the order of
∼ 100 meters. So, we will concentrate on effects related to wing flapping, ignoring effects due to changing orientation and distance from the observer.

Thus, if we are to find a statistical distribution for blob sizes found in the original data, we must find a distribution that is limited to positive numbers, and has a well-defined mean and a long tail in the positive direction. The Gamma distribution is a natural candidate for this. We get further confirmation of this hypothesis when examining the distribution of available blob size counts from the dataset. Once we have found this distribution, we can use it to generate new blob size values per track, rounding up to the nearest natural number.

The distribution was approximated using functions in the SciPy library. Rather than attempting to derive the Gamma function shape \((k)\) and scale \((\theta)\) parameters, we instead performed a simple grid search, generating pairs of \((k, \theta)\) and using them to generate a large random vector distributed according to \(\sim \Gamma(k, \theta)\). These results were binned according to discrete integer value, then evaluated using Scipy’s implementation of the \(\chi^2\) test against the counts from the data, which were normalized by the total number of counts in order to sum to unity. Using this approach, the parameters found were \(k = 3.0, \theta = 4.6\). These results can be seen in Figure A.1.

\[ \text{Figure A.1: Normalized Blobsize counts found in the real dataset, and their best-fit approximation.} \]

The synthetic blob size data generated by the distribution was initially highly disordered. In order for these generated and discretized values to resemble actual data, it was necessary to order them according to some pattern. To accomplish this, a sine wave was generated that approximated the frequency of the fluctuations in the actual data. This wave was modulated by a lower frequency sine wave. Additionally, the entire sequence was amplitude-modulated by a very low-frequency sine wave. The frequency and phase of every sine wave were randomized per instance in order to guarantee uniqueness across all generated tracks. Finally, Average Gaussian White Noise was added to the sinewave.

The synthetic blob size data was then sorted in order from minimum to maximum
prior to integer discretization. Meanwhile, the indices of the sinewave were sorted according to value of resulting sample, such that the resulting array first contained the index of the lowest actual value, followed by that of the second lowest, and so on up to the highest value. These indices were then used to select the sorted blobsize values. In this way, the sinewave shape was mapped onto the blobsize values, which were then discretized to generate the final vector of blobsizes. Several examples of the results of this operation, compared to the real blobsize vectors, are found in Figure A.2.

A.4 Full Derivation of the Gradient of the Stepwise Gamma Matrix for SSILP:

\[
\nabla \tilde{G} \left( (\bar{a} + \tilde{G}\tilde{w}_{[0]}^T \mathbf{1}_{N+1} - \Delta_v)^2 \right) \\
= \nabla \tilde{G} \left( (\bar{a} + \tilde{G}\tilde{w}_{[0]}^T \mathbf{1}_{N+1})^2 + \Delta_v^2 - 2\Delta_v (\bar{a} + \tilde{G}\tilde{w}_{[0]}^T \mathbf{1}_{N+1}) \right) \\
= \nabla \tilde{G} \left( \bar{a}^T \mathbf{1}_{N+1}^2 + (\tilde{G}\tilde{w}_{[0]}^T \mathbf{1}_{N+1})^2 \right. \\
\left. + 2\tilde{G}\tilde{w}_{[0]}^T \mathbf{1}_{N+1} \bar{a}^T \mathbf{1}_{N+1} + \Delta_v^2 - 2\Delta_v \bar{a}^T \mathbf{1}_{N+1} + 2\Delta_v \tilde{w}_{[0]} \tilde{G} \mathbf{1}_{N+1} \right) \\
= \nabla \tilde{G} \left( \bar{a}^T \mathbf{1}_{N+1}^2 + (\tilde{G}\tilde{w}_{[0]}^T \mathbf{1}_{N+1})^2 \right. \\
\left. + 2\tilde{G}\tilde{w}_{[0]}^T \mathbf{1}_{N+1} \bar{a}^T \mathbf{1}_{N+1} + \Delta_v^2 - 2\Delta_v \bar{a}^T \mathbf{1}_{N+1} + 2\Delta_v \tilde{w}_{[0]} \tilde{G} \mathbf{1}_{N+1} \right)
\]

Since \( \tilde{G} \) is square diagonal matrix, we rearrange to be able to use the identities \( \nabla X(a^TX^Tb) = ba^T \) and \( \nabla X(b^TX^TXc) = X(bc^T + cb^T) \), yielding

\[
= \nabla \tilde{G} \left[ \tilde{w}_{[0]}^T \tilde{G} \tilde{G} \mathbf{1}_{N+1} \tilde{w}_{[0]}^T \mathbf{1}_{N+1}^2 + 2\tilde{w}_{[0]}^T \tilde{G} \left( \mathbf{1}_{N+1} \bar{a}^T \mathbf{1}_{N+1} - \Delta_v \mathbf{1}_{N+1} \right) + C \right] \\
= \tilde{G} \left( \tilde{w}_{[0]}^T (\mathbf{1}_{N+1} \tilde{w}_{[0]}^T \mathbf{1}_{N+1})^T + \mathbf{1}_{N+1} \tilde{w}_{[0]}^T \mathbf{1}_{N+1}^T \right) + 2 \left( \mathbf{1}_{N+1} (\bar{a}^T \mathbf{1}_{N+1} - \Delta_v) \right) \tilde{w}_{[0]}^T \\
= \tilde{G} \left( \left( \tilde{w}_{[0]}^T \mathbf{1}_{N+1} \tilde{w}_{[0]}^T \mathbf{1}_{N+1}^T \right)^2 + (\mathbf{1}_{N+1} \tilde{w}_{[0]}^T \mathbf{1}_{N+1})^2 \right) + 2 \left( \mathbf{1}_{N+1} (\bar{a}^T \mathbf{1}_{N+1} - \Delta_v) \right) \tilde{w}_{[0]}^T
\]
Figure A.2: Real blobsize sample sequences compared to synthetic blobsize sequences.