A Probabilistic Framework for Tracking

by

Navid Sabbaghi

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

March 2001

Author ....[illegible]..
Department of Electrical Engineering and Computer Science
March 5, 2001

Certified by ....
Sanjoy K. Mitter
Professor of Electrical Engineering
Thesis Supervisor

Accepted by ....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

A Probabilistic Framework for Tracking

by

Navid Sabbaghi

Submitted to the Department of Electrical Engineering and Computer Science
on March 5, 2001, in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering and Computer Science

Abstract

In this thesis, I study the problem of offline tracking for a moving object in space and time. I setup a prior "motion probability model" describing how the object we wish to track moves at each time step, and a noisy "measurement probability model" describing the possible pixel values that can be measured as a function of the object's location. Given measurements over time, I am interested in finding the MAP estimate of the true path and the MAP estimate of a δ-neighborhood that contains the true path for all time.

It is not clear how to obtain these estimates in polynomial time in the parameters of the problem if there are an exponential number of paths and neighborhoods[1] to optimize over. Therefore, in this thesis I analyze the complexity of such problems and do the appropriate experiments. I focus on the role of the prior "motion probability model" in reducing the computational complexity of the problem.

keywords: motion estimation, random walk, markov process, two dimensional lattice, bayesian analysis, phase transition(s), prior probability models, optimal neighborhoods, confidence intervals.

Thesis Supervisor: Sanjoy K. Mitter
Title: Professor of Electrical Engineering

[1]Exponential in the parameters of the problem

Acknowledgments

First of all, I thank my advisor, Professor Sanjoy Mitter, for his patience and guidance. He has shown me the importance of a research methodology that I plan to utilize in my research path. Essentially, he has shown me that open-loop research can be unstable and that closed-loop feedback is essential for productive and meaningful research. My advisor has also shown me the efficiency that mathematically precise problem forumulations offer to a person's mental processes when solving and comparing problems. I thank him for his valuable lessons and look forward to learning more from him.

Also I thank my colleagues and friends Louay Bazzi, Paul-Peter Sotiriadis, Stefano Casadei, Chahriar Assad, Ali Rahimi, Anand Ganti, Michael Neely, Keith Duggar, Fardad Hashemi, Farid Ganji, Carol Frederick, Taher Dehkhoda, Ziad Nejmeldeen, Rain, Tiger, and Sun for their willingness to listen to my ideas and to share theirs.

Lastly, I thank my family: my brothers Arman, Azad, and Omid, and my mother Manijeh and my father Asghar. Without their love and encouragement throughout life, my life would be very different.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivational background

1.1.1 How did I choose this research problem?

Bayesian tracking is an extremely active area of research that I stumbled upon through my interest in online handwritten signature verification [11, 12, 13, 18, 1]. My research in online handwritten signature verification led me to conjecture several new invariants in people's handwritten signatures [16]. However, these invariants required real-time measurements of the kinematic state of the hand. Initially, I wanted to build a marker-based vision tracking system in order to track features of the hand as it was signing and demonstrate my conjectured invariants [14, 8]. In modelling the tracking problem, I treated the features of the hand as separate ping pong balls in three dimensional space that I wanted to track. However, the more I abstracted away the physical details of the problem, the more I realized the complexity of the tracking problem I confronted and tracking problems in general [7, 19, 4]. In this thesis, I analyze different formulations for the abstract tracking problem and analyze how complexity increases in the abstract tracking problem as one changes the prior "motion probabilty model" that describes how the object we wish to track moves at each time step.

1.1.2 The Bayesian Tracking Problem and its intricate questions

In order to visualize the abstract tracking problem, imagine the following scenario. A white ping pong ball slides on a uniformly black table as a human tugs at it via an attached invisible string. A camera, oriented such that its image plane is roughly parallel to the table top and such that it is suspended above the table top, records the motion of the ball. The camera has only the black table top and the white ping pong ball in its field of view at all instances of time. Given the sequence of images that the camera records, how can we best determine the location of the ping pong ball in the sequence of images for all time? How meaningful is such an estimate? How robust is such an estimate to noise? What would be a more robust estimator? What is the complexity of such estimates? How do the answers to these questions change if we change the prior "motion probability model"? These are the questions that bolstered my interest in the abstract tracking problem. I will address these questions throughout the thesis.

For the sake of notational simplicity and building intuition, I will analyze these questions for the case where the ping pong ball is one pixel wide and its movement is limited to one dimension, a line. Then after having built some intuition, I will look to extend my results to the case where the ball can move in two or more dimensions. Therefore, imagine a one pixel wide object moving on a line. Let us assume a prior "motion probability model" for the motion of the object, and a noisy "measurement probability model" for measurement of pixel values in the camera with the addition of shot noise. Within that framework, what are good formulations for tracking the object's path? How do we devise efficient algorithms to find estimates of the object's path as defined by the formulation? Are those algorithms generalizable to prior "motion probability models" that allow motion in more than one dimension? What changes in our prior "motion probability model" can alter the optimality of simple efficient algorithms? These are the main questions that drive the organization of this thesis.

1.1.3 Further details on the organization of this thesis

In this chapter, I propose different optimization formulations for the abstract tracking problem, discuss the advantages and disadvantages of the different formulations, look at their connections, choose a formulation to explore deeply in this thesis, and analyze the complexity of that formulation. Furthermore I consider several shot-noise "measurement probability models" and choose one to utilize throughout the thesis. In addition, I consider several "motion probability models" and choose a class of random walks to utilize throughout the thesis. Lastly on this class of random walks, I define three prior "motion probability model" regimes in order to analyze the optimality and efficiency of deterministic algorithms that aim to solve the optimization problem exactly for the class of motion models that I shall define. The analysis and realizability of optimal and efficient algorithms that solve the formulation in these three "operating regimes" will be the basis for our phase transition analysis at the end of the thesis.

In chapter two, three, and four I will look at each of these "operating regimes" separately. In chapter five I will assess the results and offer the "bigger picture".

1.2 Setting up the mathematical infrastructure

In order to offer different formulations for the Bayesian Tracking Problem, we need to setup some mathematical infrastructure to refer to pixels, measurements on those pixels as a function of time, and the object's possible location as a function of time.

1.2.1 Defining the lowest level of abstraction: the available data, the labelling

Therefore let

$$\mathtt{S} \in \mathbb{N}$$

$$\mathcal{S} = \{n \mid 1 \le n \le \mathtt{S}, n \in \mathbb{N}\}$$

$$\mathtt{T} \in \mathbb{N}$$

$$\mathcal{T} = \{t \mid 1 \le t \le \mathtt{T}, t \in \mathbb{N}\}$$

$$\mathcal{Q} \subset \mathbb{N}, |\mathcal{Q}| \in \mathbb{N}$$

$$i \colon \mathcal{S} \times \mathcal{T} \to \mathcal{Q} \subset \mathbb{N}$$

$$l \colon \mathcal{S} \times \mathcal{T} \to \{0, 1\}$$

The set $\mathcal{S}$ represents the pixels in *space* from which we obtain image intensity measurements. Throughout this thesis, let us assume that this set of pixels lies on a line as opposed to a plane, in order to initially analyze the one dimensional tracking problem. The set $\mathcal{T}$ represents the *time* instances over which we obtain image measurements. The function $i(s,t)$ returns the *quantized* image *intensity* measurement for pixel s at time t. And the function $l(s,t)$ *labels* pixel s at time t as either a background pixel (represented by 0) or an object pixel (represented by 1).

Given the sets $\mathcal{S}, \mathcal{T}, \mathcal{Q}$, and measurements $i(s,t)$ $\forall (s,t) \in \mathcal{S} \times \mathcal{T}$, the Bayesian Tracking Problem is related to estimating the unknown labelling $l(s,t)$ $\forall (s,t) \in \mathcal{S} \times \mathcal{T}$.

In order to finish setting up the problem, we have to decide on a model for how the measurements (represented by the function i evaluated on $\mathcal{S} \times \mathcal{T}$) are related to the unknown labelling (represented by the function l evaluated on $\mathcal{S} \times \mathcal{T}$).

1.2.2 A higher level of abstraction: the measurement matrix, the object's path

Let us simplify the problem and assume the labelling, l, has a special form: $\exists \mathtt{L}$ $\forall (s,t) \in \mathcal{S} \times \mathcal{T}$ such that,

$$l(s,t) = \left\{ \begin{array}{lll} 1 & \text{if} & X_t - \frac{\mathtt{L}}{2} \leq s \leq X_t + \frac{\mathtt{L}}{2} \\ 0 & \text{otherwise} & \end{array} \right\}.$$

$X_t \in \mathcal{S}$ represents the centroid in image intensity of the object we wish to track at time t. $\mathtt{L}$ is the length of the tracking window whose center is the centroid of the object and which encloses the object and a few background pixels. Assuming the labelling has a fixed size window which contains most of the object for all time is a realistic assumption because the object moves in a plane parallel to the camera's image plane and hence its size doesn't change much. But for the purposes of this thesis we will simplify the labelling l even further by assuming the object is one pixel wide, so that $\mathtt{L} = 0$. If we estimate the vector $\underline{X} \triangleq (X_1, ..., X_{\mathtt{T}}) \in \mathcal{S}^{\mathtt{T}}$, we will have estimated l. Let us assume throughout this thesis that we know the object's initial location at time $t = 0$ and let X_0 denote the object's initial location.

Let us define another way to refer to the image intensity measurements, $Y_s(t) \colon \mathcal{T} \rightarrow \mathcal{Q}$, $\forall t \in \mathcal{T}, \forall s \in \mathcal{S}$ such that $Y_s(t) \triangleq i(s,t)$ over the domain $\mathcal{T}$. In order to finish setting up the problem, let us define a prior *motion probability model*, $P(\underline{X})$, that acts as a prior for the likely positions (and hence motions), $\underline{X}$, and a noisy *measurement probability model*, $P(\underline{\underline{Y}} \mid \underline{X})$, that relates the data we measure, the $\mathtt{T} \times \mathtt{S}$ matrix $\underline{\underline{Y}}$ whose $\mathtt{t}^{th}$ row equals $\underline{Y}(t) \triangleq (Y_1(t), ..., Y_{\mathtt{S}}(t)) \colon \mathcal{T} \rightarrow \mathcal{Q}^{\mathtt{S}}$ or equivalently whose (t,s) element $Y_{t,s}$ equals $Y_s(t)$, to what we wish to estimate, $\underline{X}$. I will present examples of different motion and measurement probability models in the following sections. Later we will choose amongst different models and analyze the complexity and realizability of an "optimal" deterministic algorithm in solving a formulation that we shall soon define.

1.2.3 The highest level of abstraction: sets of possible paths, and sets of possible neighborhoods for paths

The prior "motion probability model" will aid us in defining a set of possible paths for the object. Given a prior "motion probability model", M, let us define the set of possible paths for the object as $\mathcal{P}(M) \subset \mathcal{S}^{\mathrm{T}}$.

In our formulation we will be interested in describing neighborhoods of paths as opposed to paths. These neighborhoods will be parameterized by vectors that lie in $\mathcal{S}^{\mathrm{T}}$. For future reference, for a given prior "motion probability model", M, let us define the set of possible neighborhood parameters (each parameter corresponding to a unique neighborhood with δ pixels at each time instant) as $\mathcal{A}(M, \delta) \subset \mathcal{S}^{\mathrm{T}}$.

We have set up the mathematical infrastructure for the problem. We now have a probability distribution on the set of problem instances that any algorithm is likely to encounter, namely $P(\underline{X}, \underline{\underline{Y}})$, because we have defined the prior "motion probability model", $P(\underline{X})$, and the noisy "measurement probability model", $P(\underline{\underline{Y}} \mid \underline{X})$. All that remains is a mathematical formulation for the problem we wish to solve.

1.3 Different mathematical problems with the same stated goal

For the meantime let us assume that we fix a prior "motion probability model", M, and a noisy "measurement probability model."

We want to know the location of the object for all time, $\underline{X}$, given measurements, $\underline{\underline{Y}}$. There are many different ways of obtaining an estimate of the true path $\underline{X}$, $\underline{\widehat{X}} \triangleq (\widehat{X}_1, ..., \widehat{X}_{\mathrm{T}}) \in \mathcal{P}(M)$. Some such estimates of the true path, $\underline{X}$, are

1. $\underline{\widehat{X}}^{(1)} = arg\max_{\underline{K} \in \mathcal{P}(M)} P[\underline{K} = \underline{X} \mid \underline{\underline{Y}}]$,

2. $\underline{\widehat{X}}^{(2)} = arg\max_{\underline{K}\in\mathcal{P}(M)} P[(|\sum_{i=1}^{t} K_i - \sum_{i=1}^{t} X_i| \leq \delta), \forall t \in \mathcal{T} \mid \underline{\underline{Y}}]$,

3. $\underline{\widehat{X}}^{(3)} = arg\max_{\underline{K}\in\mathcal{P}(M)} P[(|\underline{K} - \underline{X}|_\infty \leq \delta) \mid \underline{\underline{Y}}]$,

4. $\underline{\widehat{X}}^{(4)} = arg_{\underline{K}\in\mathcal{P}(M)}(P[\underline{K} = \underline{X} \mid \underline{\underline{Y}}] > 1 - \epsilon)$,

5. $\underline{\widehat{X}}^{(5)} = arg_{\underline{K}\in\mathcal{P}(M)}(P[(|\sum_{i=1}^{t} K_i - \sum_{i=1}^{t} X_i| \leq \delta), \forall t \in \mathcal{T} \mid \underline{\underline{Y}}] > 1 - \epsilon)$,

6. $\underline{\widehat{X}}^{(6)} = arg_{\underline{K}\in\mathcal{P}(M)}(P[(|\underline{K} - \underline{X}|_\infty \leq \delta) \mid \underline{\underline{Y}}] > 1 - \epsilon)$.

What do these estimates mean? What are the connections between them? What are their differences? Which ones are robust to noise? First note that the last three estimates are relaxations of the first three estimates. So we might expect it to be more difficult in terms of complexity to solve the first three problems than their corresponding relaxed versions. In the following subsections, I analyze each of the estimates and the conditions they satisfy in order to determine the connections between the estimates.

1.3.1 Estimates that maximize the probabilty of exact matches

Initially, one might hypothesize that finding the maximum a posteriori estimate

$$arg \max_{\underline{K}\in\mathcal{P}(M)} P(\underline{K} = \underline{X} \mid \underline{\underline{Y}})$$

and its relaxation

$$arg_{\underline{K}\in\mathcal{P}(M)}(P[\underline{K} = \underline{X} \mid \underline{\underline{Y}}] > 1 - \epsilon)$$

are the best ways to formulate tracking problems in the sense described in the introduction. However, such estimates might not be unique. In fact, there may be an exponential number (in the parameters $\mathtt{S}$ and $\mathtt{T}$) of paths that all have the same maximizing value. Secondly, such an estimate need not be robust; a small perturbation in the noise measurements may lead to a completely different path estimate. Thirdly, for the chosen estimate $\underline{\widehat{X}} \triangleq (\widehat{X}_1, ..., \widehat{X}_{\mathtt{T}}) \in \mathcal{P}(M)$ the value $P(\underline{\widehat{X}} = \underline{X} \mid \underline{\underline{Y}})$ might be small and close to the a posteriori probability value for other possible estimates, leading to an ambiguity in the quality of $\underline{X}$ as an estimate for the problem described in the introduction even though at first glance in terms of mathematical symbols it seems to formalize the problem we described. The ambiguity might be resolved if we knew the distribution of possible paths and their probabilities but this could be a large computational cost [17].

1.3.2 Estimates that maximize the probability of having a small absolute value of the integral of signed error

So consider estimating the object's path $\{X_t \mid 1 \leq t \leq \mathtt{T}\}$ with

$$\{\widehat{X}_t \mid 1 \leq t \leq \mathtt{T}, P[(|\sum_{i=1}^{k} \widehat{X}_i(\underline{\underline{Y}}) - \sum_{i=1}^{k} X_i| \leq \delta), \forall k \in \mathcal{T} \mid \underline{\underline{Y}}] > 1 - \epsilon\}$$

where $\underline{\underline{\mathrm{Y}}}$ is the $\mathtt{T} \times \mathtt{S}$ matrix of pixel value measurements whose $\mathtt{t}^{th}$ row equals $\underline{Y}(t) \triangleq (Y_1(t), ..., Y_{\mathtt{S}}(t))\colon \mathcal{T} \to \mathcal{Q}^{\mathtt{S}}$. Since $\underline{\widehat{X}}$ is a function of $\underline{\underline{Y}}$, I write $\underline{\widehat{X}}$ as $\underline{\widehat{X}}(\underline{\underline{Y}})$ and call it a decision rule. I shall not always explicitly denote the estimate as a function of the measurements, $\underline{\underline{Y}}$, because we always will fix the measurements before conducting any analysis. Now how do the estimates, the $\widehat{X}_t$'s, relate geometrically to the object's actual position at each point in time, the X_t's? The following conditions are equivalent:

$$(|\sum_{i=1}^{k} \widehat{X}_i(\underline{\underline{Y}}) - \sum_{i=1}^{k} X_i| \leq \delta), \forall k \in \mathcal{T}$$

$$\Longleftrightarrow$$

$$(|\sum_{i=1}^{k} (\widehat{X}_i(\underline{\underline{Y}}) - X_i)| \leq \delta), \forall k \in \mathcal{T}$$

$$\Longleftrightarrow$$

$$\max_{k \in \mathcal{T}} |\sum_{i=1}^{k} (\widehat{X}_i(\underline{\underline{Y}}) - X_i)| \leq \delta.$$

Note that $(\widehat{X}_i(\underline{\underline{Y}}) - X_i)$ is the signed error in length at time instant i. So $|\sum_{i=1}^{k}(\widehat{X}_i(\underline{\underline{Y}}) - X_i)|$ is the absolute value of the integral of signed error over the time instances $\{1, 2, ..., k\}$. We would like this discrete integral to be small $\forall k \in \mathcal{T}$. That would imply that every estimated motion subpath, $\underline{\widehat{P}_k} \triangleq (\widehat{X}_1, ..., \widehat{X}_k)$, wiggles around the true motion subpath, $\underline{P_k} \triangleq (X_1, ..., X_k)$ $\forall k \in \mathcal{T}$. Does this condition constrain the estimated path $\underline{\widehat{X}} \triangleq (\widehat{X}_1, ..., \widehat{X}_{\mathrm{T}})$to be within some distance of the true path for all time?

1.3.3 Estimates that define "high probability" neighborhoods

Now instead of looking for estimates that have a high probability of having a small absolute value of the integral of signed error, we could look for estimates that have a high probability of having a small absolute error at all time instants. In that spirit, a different criterion is one that states that the estimated path $\underline{\widehat{X}} \triangleq (\widehat{X}_1, ..., \widehat{X}_{\mathrm{T}})$should be within a distance δ of the true path for all time:

$$(|\widehat{X}_k(\underline{\underline{Y}}) - X_k| \leq \delta), \forall k \in \mathcal{T}$$

$$\Longleftrightarrow$$

$$\max_{k \in \mathcal{T}} |\widehat{X}_k(\underline{\underline{Y}}) - X_k| \leq \delta$$

At this point let us define an (ϵ, δ)-optimal algorithm, as an algorithm that outputs such an estimate $\underline{A} \triangleq (A_1, ..., A_{\mathrm{T}})$for $\underline{X} \triangleq (X_1, ..., X_{\mathrm{T}})$with (ϵ, δ)-error, such that $\underline{A} \triangleq (A_1, ..., A_{\mathrm{T}})$= $arg_{\underline{K} \in \mathcal{P}(M)}(P[(|\underline{K} - \underline{X}|_\infty \leq \delta) \mid \underline{\underline{Y}}] > 1 - \epsilon)$. We can think of this vector $\underline{A}$ as the index or parameter for a neighborhood or tube that includes the actual path $\underline{X}$ with high probability.

Now we still may have the problem that the estimate may not be unique but since we are utilizing many paths that collectively form a neighborhood or tube in order to determine the solution to our optimization problem, the axis for the best neighborhood $\underline{A} \triangleq (A_1, ..., A_{\mathrm{T}})$, the solution should be more robust to noise and have much higher probability values associated with it making it a higher "quality" estimate and setting it apart from other possible axes and hence neighborhoods that aim to optimize this criterion.

In order to quantify the error between the estimated path and the true path using the geometrical idea of "closeness" at every point in time, we should choose a variant of this criterion as the optimiziation criterion for this thesis.

Relationships between the absolute value of the integral of signed error and the absolute value of error

For completeness and to answer the previous question, how are the conditions

$$(|\sum_{i=1}^{k} \widehat{X}_i(\underline{\underline{Y}}) - \sum_{i=1}^{k} X_i| \leq \delta), \forall k \in \mathcal{T}$$

$$(|\widehat{X}_k(\underline{\underline{Y}}) - X_k| \leq \delta), \forall k \in \mathcal{T}$$

related? Lets look at the implication of each condition:

1. $(|\widehat{X}_k(\underline{\underline{Y}}) - X_k| \leq \delta), \forall k \in \mathcal{T}$
   $\Longrightarrow (|\sum_{i=1}^{k} \widehat{X}_i(\underline{\underline{Y}}) - \sum_{i=1}^{k} X_i| \leq \sum_{i=1}^{k} |\widehat{X}_i(\underline{\underline{Y}}) - X_i| \leq \mathtt{T}\delta), \forall k \in \mathcal{T}$.
2. $(|\sum_{i=1}^{k} \widehat{X}_i(\underline{\underline{Y}}) - \sum_{i=1}^{k} X_i| = |\sum_{i=1}^{k}(\widehat{X}_i(\underline{\underline{Y}}) - X_i)| \leq \delta), \forall k \in \mathcal{T}$
   $\Longrightarrow (-2\delta \leq \sum_{i=m}^{n}(\widehat{X}_i(\underline{\underline{Y}}) - X_i) \leq 2\delta), \forall m \leq n; m, n \in \mathcal{T}$
   $\Longrightarrow (|\widehat{X}_i(\underline{\underline{Y}}) - X_i)| \leq 2\delta), \forall i \in \mathcal{T}$.

The above implications illustrate that the condition $(|\sum_{i=1}^{k} \widehat{X}_i(\underline{\underline{Y}}) - \sum_{i=1}^{k} X_i| \leq \delta)\forall k \in \mathcal{T}$ is more restrictive over time than the condition $(|\widehat{X}_k(\underline{\underline{Y}}) - X_k| \leq \delta), \forall k \in \mathcal{T}$. This result agrees with our intuition that the absolute value of the integral of signed error can increase over time even though the absolute value of error at each time instant is bounded.

If we let $\delta = 0$ then the optimization problem associated with both of these criterions reduces to finding our other estimate that maximizes the probability of equalling the true path $\underline{X}$ at all time instants. So the optimization problem $arg \max_{\underline{K} \in \mathcal{P}(M)} P[\underline{K} = \underline{X} \mid \underline{\underline{Y}}]$ is a simplified version of the optimization problems associated with the criterions discussed in this segment.

Finding the neighborhood that maximizes the probability that the actual path $\underline{X}$ will stay within the neighborhood for all time instants is not necessarily easier than finding estimates that satisfy the other two main criterions. But given that such a neighborhood-based estimator[1] aggregates many paths [2] that are spatially close together in order to make a probabilistic statement about the true path lying within the associated neighborhood, a neighborhood-based estimator seems more robust to noisy measurements.

1.3.4 The route to the mathematical problem addressed in this thesis

The potential robustness to noise, the real world physical interpretation associated with the optimization statement, and the possibility that the best neighborhood would have a relatively larger probability of containing the true path then the next less optimal neighborhood bolstered my interest in the estimate $\underline{A} \triangleq (A_1, ..., A_{\mathrm{T}})= arg_{\underline{\widehat{X}} \in \mathcal{P}(M)}(P[(|\underline{\widehat{X}}(\underline{\underline{Y}}) - \underline{X}|_\infty \leq \delta) \mid \underline{\underline{Y}}] > 1 - \epsilon)$ and algorithms that can find such an estimate.

Why is the estimate and its associated neighborhood meaningful?

Exact tracking is not always necessary and noisy measurements can restrict any algorithm to low certainty estimates for an object's path. For some applications it suffices to know that an object's actual space-time trajectory, $\underline{X}$, lies within a δ-radius cylinder (tube) with probability at least $1 - \epsilon$. Let us refer to the process of finding such a tube as approximate tracking or (ϵ, δ)-optimal tube tracking. For example, in collision avoidance systems (ie automated highways) the system tracks an object and is only interested in knowing that its path lies in some "region" (ie cylinder) that doesn't contain obstacles. In surveillance (ie aerial) applications, one is interested in approximate paths/locations of an object in order to determine if an object enters a designated "region". In real-time data acquisition systems for

[1]namely the estimate that indexes or parametrizes the "optimal" neighborhood

[2]Note that the number of paths in a neighborhood could be exponential in the parameters of the problem and hence on the same order as the number of possible paths.

controlling experimental setups (ie wafer polishing sensor systems that collect sensor data in order to decide when to stop polishing a silicon wafer), the system needs to act differently when the system's state trajectory enters a critical data "region" for the first time. Instead of finding an exact path with low certainty, we can aggregate the possible paths and find a δ-radius cylinder (tube) that contains the true path with high certainty. Hence finding a δ-cylinder may benefit many engineering systems and it should prove to be a robust estimator with meaningful mathematical properties.

Initially it seemed that finding a rough cylindrical boundary for the true path would be computationally less expensive than estimating a single path for the actual path hence making such a formulation ideal for real-time systems. However at the end of this thesis we discovered that the complexity for the most efficient algorithms for these formulations has the same order of growth in terms of the parameters of the problem. This does not necessarily imply that there is not a faster solution for a neighborhood formulation compared to an exact path formulation.

For real-time applications, where we also have to quantize our measurement values and sample them in space, finding a δ-cylinder may be more robust; the exact path may become partially "sampled" or "quantized" out, whereas aggregate collections of paths that are near one another are more resistant to these alterations.

This forumlation helps us model away some of the complex realities of real world systems such as camera jitter and object jitter since localization noise shouldn't perturb our δ-cylinder too much if our neighborhood size's are larger than the jitter sizes. Instead we can focus on modelling the physics of the problem.

In terms of error analysis, we would like to know how likely it is that an object's space-time trajectory lies outside a cylinder of interest at any point in time. This formulation would allow us to make such exact statements.

Hence with that motivation, the formulation of the tracking problem as the problem of finding these high probability neighborhoods (I also refer to them as tubes and cylinders) under different circumstances (probability models) and finding a "reasonable" algorithm (in terms of complexity) to solve that problem commenced the work of this research.

Therefore, I started studying the unrelaxed version of this problem. Namely, how do we find an efficient (in space and time), deterministic, offline algorithm that given sensor input matrix , $\underline{\underline{Y}}$, will output $\underline{A} \triangleq (A_1, ..., A_{\mathtt{T}})$ such that $\underline{A} \triangleq (A_1, ..., A_{\mathtt{T}})$= $arg\max_{\underline{K}\in\mathcal{P}(M)} P[(|\underline{K} - \underline{X}|_\infty \leq \delta) \mid \underline{\underline{Y}}]$.

Since $\underline{A}$ is a function of $\underline{\underline{Y}}$, I will occasionally write $\underline{A}$ as $\underline{A}(\underline{\underline{Y}})$ and call it a decision rule. Therefore, more compactly, I wanted the output $\underline{A}(\underline{\underline{Y}})$ to satisfy

$$\underline{A}(\underline{\underline{Y}}) = arg \max_{\underline{a}\in\mathcal{P}(M)} P[\underline{X} \in C_\delta(\underline{a}) \mid \underline{\underline{Y}}],$$

where the δ-cylinder with center $\underline{a}$ is defined as $C_\delta(\underline{a}) \triangleq \{\underline{\gamma} \in \mathcal{S}^{\mathtt{T}} \mid |\underline{\gamma} - \underline{a}|_\infty \leq \delta\}$.

Then if such an $\underline{A}$ satisfies

$$P[(|\underline{A}(\underline{\underline{Y}}) - \underline{X}|_\infty \leq \delta) \mid \underline{\underline{Y}}] > 1 - \epsilon,$$

I have a decision instance that is an (ϵ, δ)-optimal estimate.

Finding (ϵ, δ) estimates efficiently in terms of time and space complexity is not obvious. For any reasonable motion model there are an exponential number (in $\mathtt{T}$) possible paths. For example, if the motion model allowed the object to move from any element in $\mathcal{S}$ to k elements in $\mathcal{S}$, then there are $k^{\mathtt{T}}$ possible motion paths. In general, without a motion model there are $\mathtt{S}^{\mathtt{T}}$ possible motion paths. Can (ϵ, δ)-optimal estimates be found in polynomial time and space for our problem? How efficiently can the unrelaxed version be solved for different probability models? This unrelaxed problem is the problem that is tackled in this thesis. In the next sections, we explore the probability models that will completely define the optimization problem that we aim to solve.

1.3.5 Introducing the δ-neighborhood

Let us first fix a motion model, M. There are many ways to parametrize the optimal-neighborhoods that we seek. The δ-cylinder is parameterized by the center of of the cylinder and has a δ-radius at each point in time. Instead we will define and utilize a

δ-neighborhood, $N_\delta(\underline{a})$, throughout the rest of the thesis. A δ-neighborhood ,$N_\delta(\underline{a})$, is a neighborhood with δ pixels at each point in time parameterized by the left most path that lies inside the neighborhood for all time or more formally,

$$N_\delta(\underline{a}) \triangleq \{\underline{\gamma} \in \mathcal{P}(M) \mid 0 \leq \gamma_t - a_t \leq \delta - 1, \forall t \in \mathcal{T}\}.$$

We will use this formalization of neighborhood throughout the rest of the thesis because it is defined by the number of space pixels a neighborhood contains. This concept will be useful for our phase transition analysis later.

Now what are the possible parameters $\underline{a}$ for our neighborhoods $N_\delta(\underline{a})$? They are defined to be the elements of the set $\mathcal{A}(M, \delta)$.

In terms of this formalization, we wish to find

$$\underline{A}(\underline{\underline{Y}}) = arg \max_{\underline{a} \in \mathcal{A}(M,\delta)} P[\underline{X} \in N_\delta(\underline{a}) \mid \underline{\underline{Y}}] \bigstar.$$

1.4 Possible motion models

Now, let us explore various prior "motion probability models", $P(\underline{X})$. The need for a prior for the path, $\underline{X} \triangleq (X_1, ..., X_\mathtt{T})$, surfaces in this problem domain because there are $\mathtt{S}^\mathtt{T}$ possible motion paths in general. However, realistically all those paths are not likely. Therefore as a mechanism to reduce any algorithm's time complexity in estimating $\underline{X}$, it makes sense for an algorithm to use a prior on the motions, $P(\underline{X})$, to "prioritize" the search space.

One realistic probabilistic motion model defines a probability mass function on the next likely position conditioned on an object's previous position, velocity, acceleration, and jerk. For example, if an object's current acceleration is greater than zero, and its current velocity is greater than zero, then we know that at the next time instant the object will most likely be to the right of its previous position with a variance related to its jerk, acceleration, and velocity. This intuition translates into a probability model $P(X_{n+1} \mid X_n, V_n, A_n, J_n)$, where X_n is the position at time instant n, V_n is the velocity at time n, A_n is the acceleration at time n, J_n is the time difference of acceleration between times n and $n+1$ (also referred to as the jerk). In essence, the above motion probability model translates into a Markov chain with $|\mathcal{S}| \times q_V \times q_A \times q_J$ states ($q_K \triangleq$ the cardinality of the set of possible quantized values that can be assigned to $K_i, \forall i \in \mathcal{T}$). This Markov chain and initial conditions define $P(\underline{X})$.

A simpler yet less realistic probabilistic motion model allows an object to move left with probability p, right with probability p, and stay in the same position with probability $1 - 2p$. The number of paths with nonzero probability is 3^T. Despite the fact that the model is unrealistic, it is useful; the conditional probability mass function can be an analyzable decision mechanism in the first part of a "divide-and-conquer" search heuristic used in answering the question, "Did the object move? If so, did it move left or right?" If we add memory (velocity, acceleration, ... of the object) to this model, then it will resemble the first model more closely and hence become more realistic.

This latter model will be the starting point in our definition of a class of prior "motion probability models" that we will continually refer to throughout the thesis. Let this class of motion models be parameterized by a parameter, Δ, that defines the maximum step size in space pixels that the object can move at each point in time. More precisely, a prior "motion probability model" M_Δ is defined in the following way:

- $X_0 \in \mathcal{S}$ is the known initial location of the object at time t=0,
- the motion of the object is described by the following first order markov chain:

$$P(X_{n+1} \mid X_n) = \left\{ \begin{array}{lll} \frac{1}{2\Delta+1} & \text{if} & X_{n+1} = X_n - i, i \in \{1, 2, ..., \Delta\} \\ \frac{1}{2\Delta+1} & \text{if} & X_{n+1} = X_n \\ \frac{1}{2\Delta+1} & \text{if} & X_{n+1} = X_n + i, i \in \{1, 2, ..., \Delta\} \\ 0 & \text{otherwise} & \end{array} \right\},$$

$$1 \leq n \leq \mathrm{T}.$$

Therefore M_1 denotes the prior "motion probabillity model" that describes the ability of an object to stay in the same position, move one step left, or move one step right with equal probability at each time instant.

1.5 Possible measurement models

The noisy "measurement probability model", $P(\underline{\underline{Y}} \mid \underline{X})$, describes how an object and its trajectory are related to what a camera senses. In their least filtered form, our measurements are the elements of a $\mathtt{T} \times \mathtt{S}$ matrix $\underline{\underline{Y}}$ whose (t, s) element is $Y_s(t) \triangleq i(s, t)$.

For example, a model for noisy measurements that are minimally filtered is $Y_s(t) = o(s - X_t) + n_s(t)$ where $o(s) \colon \mathbb{Z} \to \mathcal{Q}$ is a template of the object that an algorithm is tracking. Modelling the image of a ping pong ball, we could define the template

$$o(s) = \left\{ \begin{array}{lll} A(1 + \cos \frac{2\pi s}{\mathtt{L}_\circ}) & \text{if} & -\frac{\mathtt{L}_\circ}{2} \leq s \leq \frac{\mathtt{L}_\circ}{2} \\ 0 & \text{otherwise} & \end{array} \right\}$$

where A is the brightness intensity at the center of the one dimensional ping pong ball and $\mathtt{L}_\circ$ is the length (in pixels) of the object. Simply rewriting $o(s)$ as $o(\underline{s})$ and replacing s in the definition with $||\underline{s}||^2$, we could define $o(s)$ for a two dimensional ping pong ball. Note that this particular template $o(s)$ implies the fact that brightness measurements tend to be concave near the maximum intensity and convex near the "edges". A and $\mathtt{L}_\circ$ are unknowns that can be learned from training data. $n_s(t)$ is a discrete-space-time noise process where $n_{s_1}(t_1)$ and $n_{s_2}(t_2)$ are iid random variables $\forall (s_1, t_1), (s_2, t_2) \in \mathcal{S} \times \mathcal{T}$ whose distribution is determined empirically from training data.

An alternative noisy "measurement probability model" relates measurement data filtered through system H, $H(\underline{\underline{Y}})$, to the object's positions over time, $\underline{X}$. For example, system H with input $Y_s(t) \forall s \in C \subseteq \mathcal{S}$ may output a quantized scalar $y_{C,t}$ (or more generally, a lower dimensional vector) that represents a feature. The set C represents the fact that an algorithm may not need all the intensity measurements in space to accurately determine the location of the object and that certain data combinations may be more important for tracking. Using a scalar measurement for each time instant may lead to a simpler estimator $\underline{A}(\underline{\underline{Y}})$ and the ability to design filters whose scalars inform a "local" algorithm which direction to search for the object in order to locate the object with high probability. Also, learning a scalar (or lower dimensional vector) measurement's distribution is easier than learning the distribution for a vector of measurements. So for example, a filter whose impulse response is h(s)=$(\delta(s) - \delta(s - \frac{L}{2})) + (\delta(s) - \delta(s + \frac{L}{2}))$ gives us scalar measurements for the object. The filter will usually give the highest response for the ping pong ball image at pixel s if s corresponds to the point of maximum intensity (the center of the ball). Thus the filter's response at different time instants can be calculated and used for determining the ball's position. This filter can be generalized and visualized as an octopus that is centered at the center of the ball with tentacles that reach out. The statistics of these "tentacle" measurements are the outputs of this filter.

Matched filters are another method that can be used to determine an object's path in a noisy image. Different matched filters will cost different amounts in terms of time and space complexity, and probability of error for an (ϵ, δ)-optimal estimate or an approximation of one.

In this thesis we will utilize a simple template for our noisy "measurement probability model" assuming that the object is one pixel wide with uniform intensity. We do this because we want to isolate the effect of different motion models on the complexity of the problem, and hence altering the noise models is not our prime concern. The measurement model employed throughout this thesis will be formally described in the next section.

1.6 Tracking algorithms

Now that we have explored different models for motion and measurement, the question arises: How do we use these models to come up with an efficient, deterministic, offline algorithm that given sensor input $\underline{\underline{Y}}$, will output $\underline{A} \triangleq (A_1, ..., A_\mathtt{T})$such that

$$\underline{A}(\underline{\underline{Y}}) = arg \max_{\underline{a} \in \mathcal{A}(M,\delta)} P[\underline{X} \in N_\delta(\underline{a}) \mid \underline{\underline{Y}}](\bigstar)$$

1.6.1 Brute force analysis

Any deterministic decision rule, $\bar{\underline{A}}(\underline{\underline{Y}})\colon \mathcal{Q}^{\mathtt{ST}} \to \mathcal{S}^{\mathtt{T}}$, has $|\mathcal{Q}|^{\mathtt{ST}}$ elements in its domain. So unless the decision rule has an analytic form, in terms of space complexity an algorithm would have to carry around a large table to define the decision rule.

How long would a brute force algorithm take to find a deterministic decision rule, $\underline{A}(\underline{\underline{Y}})$, that satisfies condition ($\bigstar$)? There are $\mathtt{S}^{T|\mathcal{Q}|^{\mathtt{ST}}}$ different decision rules. So in the worst case, a brute force algorithm would need to go through each of those decision rules and for each decision rule, $\bar{\underline{A}}$, it would need to calculate $P[\underline{X} \in N_\delta(\bar{\underline{A}}) \mid \underline{\underline{Y}}]$, which could also be expensive depending on the form of $P(\underline{X}, \underline{\underline{Y}})$.

1.6.2 Simplifying model assumptions

For the purposes of devising an efficient algorithm, I will simplify the problem with some assumptions about the formulation setup which can be relaxed later:

- First, throughout the thesis let us assume our measurements are binary such that the quantized image intensity measurements come from the set $\mathcal{Q} = \{0, 1\}$.
- Next, let us assume the object's length is one pixel wide ($\mathtt{L} = 1$).
- Also let us assume the initial position X_0 is known and choose the M_1 prior "motion probability model" that states that the object moves left, right, or stays in the same position with equal probability:

$$P(X_{n+1} \mid X_n) = \left\{ \begin{array}{lll} \frac{1}{3} & \text{if} & X_{n+1} = X_n - 1 \\ \frac{1}{3} & \text{if} & X_{n+1} = X_n \\ \frac{1}{3} & \text{if} & X_{n+1} = X_n + 1 \\ 0 & \text{otherwise} & \end{array} \right\}, 1 \leq n \leq (\mathtt{T} - 1).$$

Therefore, the true path $\underline{X}$ is a member of the set of possible paths

$$\mathcal{P}(M_1) = \{\underline{\gamma} \in \mathcal{S}^{\mathrm{T}} \mid \gamma_0 = X_0, |\gamma_j - \gamma_{j-1}| \leq 1, \forall j \in \mathcal{T}\} \subset \mathcal{S}^{\mathrm{T}}.$$

In the next chapter of this thesis, we will utilize the M_1 prior but in later chapters we will change the prior "motion probability model" to M_Δ for general Δ. In that case,

$$\mathcal{P}(M_\Delta) = \{\underline{\gamma} \in \mathcal{S}^{\mathrm{T}} \mid \gamma_0 = X_0; |\gamma_j - \gamma_{j-1}| \leq \Delta, j \in \mathcal{T}\} \subset \mathcal{S}^{\mathrm{T}}.$$

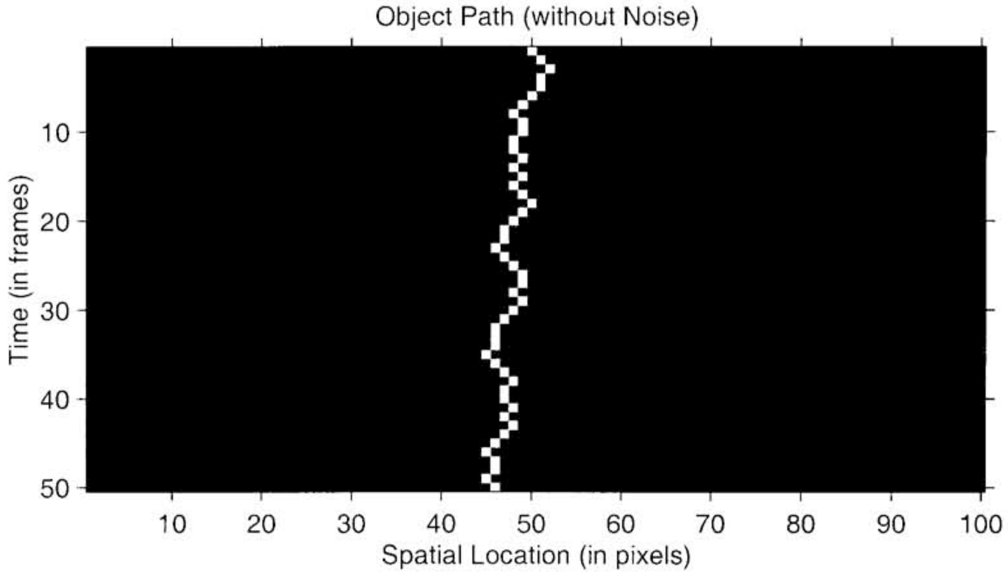
- Initially, let us assume a noise-free measurement model that states that if the object is present at position X_t at time t, then we should measure an image intensity of 1 at that location in space-time and for all other locations we should measure a 0 at time t. So let us define a simple object template, $o(s)\colon \mathbb{Z} \to \mathcal{Q}$, where

$$o(s) \triangleq \left\{ \begin{array}{lll} 1 & \text{if} & s = 0 \\ 0 & \text{otherwise} & \end{array} \right\}$$

Figure 1-1 shows a noise-free measurement, $\underline{\underline{Y}}$, in space-time sampled from our motion model.

Let us assume a simple noisy "measurement probability model" that states that the noise corrupts the measurements by flipping bits. Relating the template to our measurements and the positions, $Y_s(t) = (o(s - X_t) + n_s(t)) \bmod 2$, where $n_s(t)\colon \mathcal{T} \to \mathcal{Q}$ is discrete-time noise process such that $n_{s_1}(t_1)$ and $n_{s_2}(t_2)$ are iid Bernoulli random variables $\forall (s_1, t_1), (s_2, t_2) \in \mathcal{S} \times \mathcal{T}$. More specifically let $P(n_s(t) = 1) = \alpha$ and $P(n_s(t) = 0) = 1 - \alpha$.

Figure 1-1: The noise-free measurement matrix associated with an object's sample trajectory.



This implies that

$$P(Y_s(t)|X_t) = \left\{ \begin{array}{lll} 1-\alpha & \text{if} & X_t = s \text{ and } Y_s(t) = 1 \\ \alpha & \text{if} & X_t = s \text{ and } Y_s(t) = 0 \\ 1-\alpha & \text{if} & X_t \neq s \text{ and } Y_s(t) = 0 \\ \alpha & \text{if} & X_t \neq s \text{ and } Y_s(t) = 1 \end{array} \right\}, \forall (s,t) \in \mathcal{S} \times \mathcal{T}.$$

Lastly in order to fully write out the measurement model, $P(\underline{\underline{Y}} \mid \underline{X})$, lets look at the consequences of the probabilistic model we chose:

(A) The image measurements at time t only depend on the position measurements at time t and not at any other times. Formally, $\underline{Y}(t_1)$ and X_{t_2} are conditionally independent given X_{t_1} if $t_1 \neq t_2$.

(B) Also $\underline{Y}(t_1)$ and $\underline{Y}(t_2)$ are conditionally independent given X_{t_1} if $t_1 \neq t_2$.

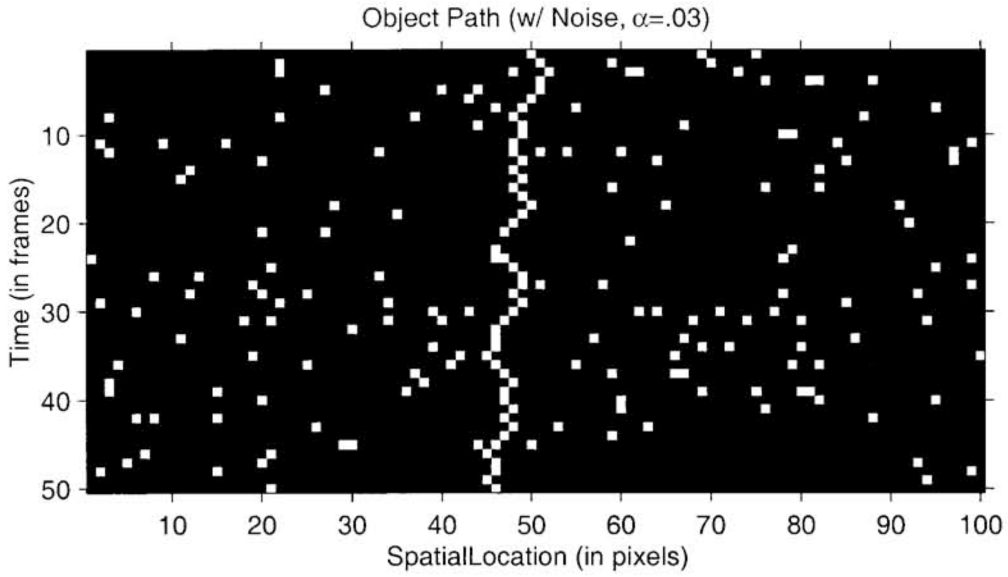
(C) And $Y_{s_1}(t)$ and $Y_{s_2}(t)$ are conditionally independent given X_t.

This implies

$$\begin{aligned}
P(\underline{\underline{Y}} \mid \underline{X}) &= P(\underline{Y}(\mathrm{T}), ..., \underline{Y}(1) \mid \underline{X}) \\
&= [\textstyle\prod_{t=2}^{\mathrm{T}} P(\underline{Y}(t) \mid \underline{Y}(t-1), ..., \underline{Y}(1), \underline{X})] P(\underline{Y}(1) \mid \underline{X}) \\
&= [\textstyle\prod_{t=2}^{\mathrm{T}} P(\underline{Y}(t) \mid \underline{Y}(t-1), ..., \underline{Y}(1), X_1, ..., X_{\mathrm{T}})] P(\underline{Y}(1) \mid X_1, ..., X_{\mathrm{T}}) \\
&= [\textstyle\prod_{t=2}^{\mathrm{T}} P(\underline{Y}(t) \mid \underline{Y}(t-1), ..., \underline{Y}(1), X_t)] P(\underline{Y}(1) \mid X_1) && \mathrm{A} \\
&= \textstyle\prod_{t=1}^{\mathrm{T}} P(\underline{Y}(t) \mid X_t) && \mathrm{B} \\
&= \textstyle\prod_{t=1}^{\mathrm{T}} P(Y_1(t), ..., Y_{\mathrm{S}}(t) \mid X_t) \\
&= \textstyle\prod_{t=1}^{\mathrm{T}} \prod_{s=1}^{\mathrm{S}} P(Y_s(t) \mid X_t) && \mathrm{C}
\end{aligned}$$

Figure 1-2 shows our space-time measurement data, $\underline{\underline{Y}}$, for the trajectory depicted in the previous figure but now corrupted with our noisy "measurement probability model".

Figure 1-2: The noisy measurement matrix associated with an object's sample trajectory.



- Throughout the thesis we will also place a continuity constraint on the neighborhood parameters, $\underline{a} \in \mathcal{A}(M, \delta)$ that parameterize/index the neighborhoods $N_\delta(\underline{a})$. Namely we will impose the constraint

$$\mathcal{A}(M_\Delta, \delta) \triangleq \{\underline{\gamma} \in \mathcal{S}^{\mathrm{T}} \mid |\gamma_j - \gamma_{j-1}| \leq 1, \forall j \in \mathcal{T}; \gamma_0 = X_0 - \lfloor \frac{\delta}{2} \rfloor\} \subset \mathcal{S}^{\mathrm{T}}.$$

1.6.3 Mathematical simplification

With our simplifying assumptions we are interested in devising an efficient, deterministic, offline algorithm that given sensor input $\underline{\underline{Y}}$, will output $\underline{A} \triangleq (A_1, ..., A_{\mathsf{T}})$such that

$$\underline{A}(\underline{\underline{Y}}) = arg \max_{\underline{a} \in \mathcal{A}(M,\delta)} P[\underline{X} \in N_\delta(\underline{a}) \mid \underline{\underline{Y}}](\bigstar)$$

So using Bayes Rule let us rewrite $P[\underline{X} \in N_\delta(\underline{a}) \mid \underline{\underline{Y}}]$:

$$\begin{aligned} P(\underline{X} \in N_\delta(\underline{a}) \mid \underline{\underline{Y}}) &= \frac{\sum_{\underline{\gamma} \in N_\delta(\underline{a})} P(\gamma, \underline{\underline{Y}})}{P(\underline{\underline{Y}})} \\ &= \frac{\sum_{\underline{\gamma} \in N_\delta(\underline{a})} P(\underline{\underline{Y}}|\underline{\gamma})P(\underline{\gamma})}{\sum_{\underline{\gamma} \in \mathcal{P}(M)} P(\underline{\underline{Y}}|\underline{\gamma})P(\underline{\gamma})} \\ &= \frac{\sum_{\underline{\gamma} \in N_\delta(a)} P(\underline{\underline{Y}}|\underline{\gamma})}{\sum_{\underline{\gamma} \in \mathcal{P}(M)} P(\underline{\underline{Y}}|\underline{\gamma})} \qquad \text{(motion model} \rightarrow \text{constant } P(\underline{\gamma})) \end{aligned}$$

Therefore

$$\begin{aligned} \underline{A}(\underline{\underline{Y}}) &= arg \max\nolimits_{\underline{a} \in \mathcal{A}(M,\delta)} P[\underline{X} \in N_\delta(\underline{a}) \mid \underline{\underline{Y}}] \\ &= arg \max\nolimits_{\underline{a} \in \mathcal{A}(M,\delta)} \sum\nolimits_{\underline{\gamma} \in N_\delta(a)} P(\underline{\underline{Y}} \mid \underline{\gamma}) \\ &= arg \max\nolimits_{\underline{a} \in \mathcal{A}(M,\delta)} \sum\nolimits_{\underline{\gamma} \in N_\delta(a)} \prod\nolimits_{t=1}^{\mathsf{T}} \prod\nolimits_{s=1}^{\mathsf{S}} P(Y_s(t) \mid \gamma_t) \end{aligned}$$

So how do we find $\underline{A}(\underline{\underline{Y}})$? And once we do that, how do we compute $P(\underline{X} \in N_\delta(\underline{A}) \mid \underline{\underline{Y}})$ given that its denominator in the Bayes Rule manipulation is a sum over an exponential number of paths? These questions will be addressed for the special case of $\delta = 0$ in the next chapter.

Further mathematical simplication using the simplified model

In this section, we would like to rewrite the term $\sum_{\underline{\gamma} \in N_\delta(a)} \prod_{t=1}^{\mathsf{T}} \prod_{s=1}^{\mathsf{S}} P(Y_s(t) \mid \gamma_t)$ stated in the last section more explicitly in terms of the noise level α and the individual measurements $Y_{t,s}$.

So let us start by defining a function that takes a time t, set of measurements at time t, say $\underline{Y}(t) = (Y_{t,1}, Y_{t,2}, ..., Y_{t,\mathsf{S}})$ and a possible location for the object at time t, say $l_t \in \mathcal{S}$, and outputs the number of local (in time) undamaged measurement bits at time t.

Therefore I define

$$N(t, \underline{Y}(t), l_t) \triangleq \mathtt{S} - \sum_{l=1}^{\mathtt{S}} Y_{t,l} + I(t, Y_{t,l_t}, l_t) - O(t, Y_{t,l_t}, l_t)$$

where $I(t, Y_{t,l_t}, l_t)$ is the indicator function that returns 1 if the measurement Y_{t,l_t} is a one and 0 otherwise. Likewise $O(t, Y_{t,l_t}, l_t)$ is the indicator function that returns 1 if the measurement Y_{t,l_t} is a zero and 0 otherwise.

Therefore we can rewrite

$$\begin{aligned} N(t, \underline{Y}(t), l_t) &= \mathtt{S} - \textstyle\sum_{l=1}^{\mathtt{S}} Y_{t,l} + Y_{t,l_t} - (1 - Y_{t,l_t}) \\ &= \mathtt{S} - \textstyle\sum_{l=1}^{\mathtt{S}} Y_{t,l} + 2Y_{t,l_t} - 1 \end{aligned}$$

So for example say $\mathtt{S} = 9$ and $\mathtt{T} = 5$. Then we have a 5×9 matrix of measurements which are 0s and 1s. Say the initial location is $X_0 = 5$. Then at time $t = 1$, the object can be either at position $4, 5$, or 6. If the measurements at time $t = 1$ are $\underline{Y}(1) = (0, 0, 1, 0, 1, 1, 0, 0, 0)$ then the respective number of local undamaged bits for a path going through location $s = 4$ is $N(1, \underline{Y}(1), 4) = 1+1+0+0+0+0+1+1+1 = 5$. Similarly, for the same measurements at time $t = 1$, the number of local undamaged bits for a path going though location $s = 5$ is $N(1, \underline{Y}(1), 5) = 1 + 1 + 0 + 1 + 1 + 0 + 1 + 1 + 1 = 7$.

Now let us use the above formalism to simplify the optimization problem even further:

$$\begin{aligned} \underline{A}(\underline{\underline{Y}}) &= arg\max\nolimits_{\underline{a} \in \mathcal{A}(M,\delta)} P[\underline{X} \in N_\delta(\underline{a}) \mid \underline{\underline{Y}}] \\ &= arg\max\nolimits_{\underline{a} \in \mathcal{A}(M,\delta)} \textstyle\sum_{\underline{\gamma} \in N_\delta(\underline{a})} P(\underline{\underline{Y}} \mid \underline{\gamma}) \\ &= arg\max\nolimits_{\underline{a} \in \mathcal{A}(M,\delta)} \textstyle\sum_{\underline{\gamma} \in N_\delta(\underline{a})} \prod_{t=1}^{\mathtt{T}} \prod_{s=1}^{\mathtt{S}} P(Y_s(t) \mid \gamma_t) \\ &= arg\max\nolimits_{\underline{a} \in \mathcal{A}(M,\delta)} \textstyle\sum_{\underline{\gamma} \in N_\delta(\underline{a})} \prod_{t=1}^{\mathtt{T}} (1-\alpha)^{N(t,\underline{Y}(t),\gamma_t)} \alpha^{(S - N(t,\underline{Y}(t),\gamma_t))} \\ &= arg\max\nolimits_{\underline{a} \in \mathcal{A}(M,\delta)} \textstyle\sum_{\underline{\gamma} \in N_\delta(\underline{a})} (1-\alpha)^{\sum_{t=1}^{\mathtt{T}} N(t,\underline{Y}(t),\gamma_t)} \alpha^{\sum_{t=1}^{\mathtt{T}} (S - N(t,\underline{Y}(t),\gamma_t))} \\ &= arg\max\nolimits_{\underline{a} \in \mathcal{A}(M,\delta)} \textstyle\sum_{\underline{\gamma} \in N_\delta(\underline{a})} (1-\alpha)^{2\sum_{t=1}^{\mathtt{T}} Y_{t,\gamma_t}} \alpha^{-2\sum_{t=1}^{\mathtt{T}} Y_{t,\gamma_t}} \\ &= arg\max\nolimits_{\underline{a} \in \mathcal{A}(M,\delta)} \textstyle\sum_{\underline{\gamma} \in N_\delta(\underline{a})} (\frac{1-\alpha}{\alpha})^{2\sum_{t=1}^{\mathtt{T}} Y_{t,\gamma_t}} \end{aligned}$$

This simplified version is friendlier to work with because the individual measurement values on the paths of interest in a given cylinder show up directly in the optimization problem.

For example, if the noise paramater that flips bits is $\alpha = \frac{1}{3}$, then $\underline{A}(\underline{\underline{Y}}) = arg\max_{\underline{a} \in \mathcal{A}(M,\delta)} \sum_{\underline{\gamma} \in N_\delta(\underline{a})} 4^{\sum_{t=1}^{\mathtt{T}} Y_{t,\gamma_t}}$.

So notice that if the noise level $\alpha \leq .5$ then we roughly want to find the region with the most number of $1s$ but more precisely the region parametrized by $\underline{\mathrm{A}}$ with many paths with $1s$. And if $.5 \leq \alpha$ we roughly want to find the region with the most number of $0s$ but more precisely the region with parametrized by $\underline{\mathrm{A}}$ with many paths with $0s$.

1.6.4 The combinatorial optimization problem

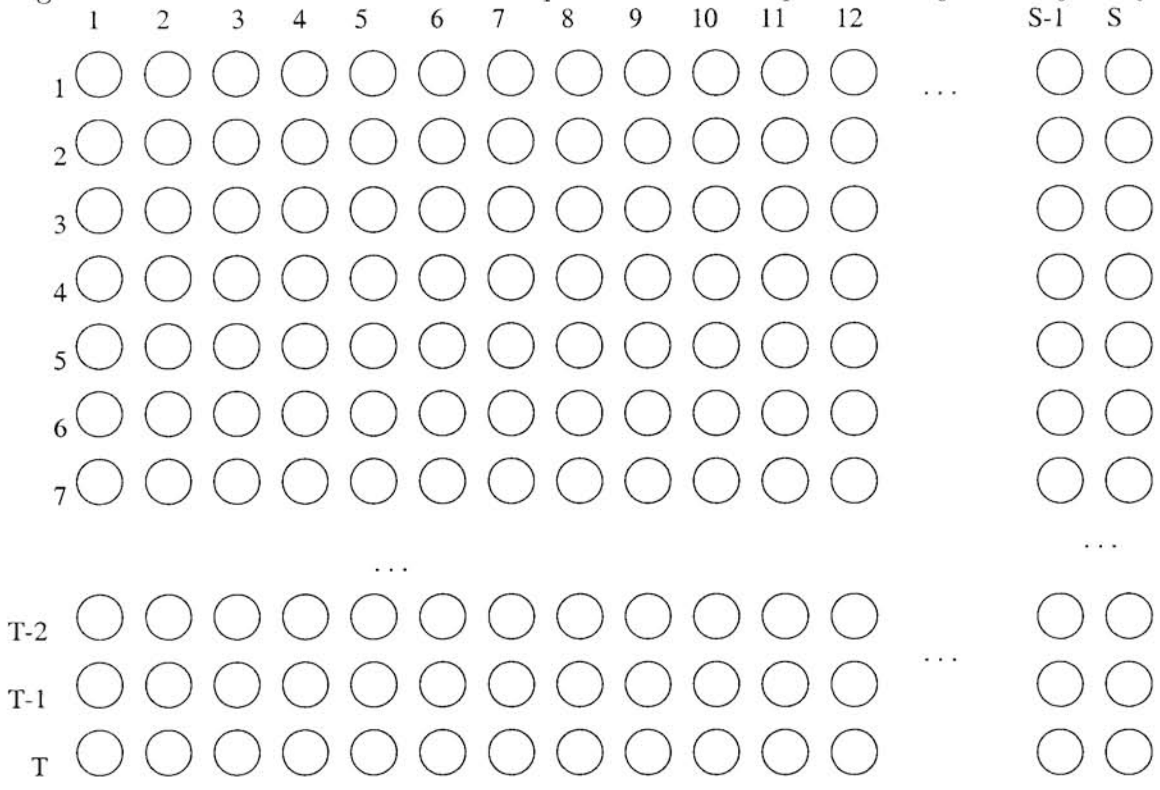
In this section, for completeness I will summarize the combinatorial optimization problem we confront without reference to probability models. So I will discuss the pure optimization problem with minimal reference to any tracking problem. This section is therefore completely self contained although its purpose is to solve the problem set out. Therefore, the notation may be a bit different than the rest of the thesis.

We have a set of points $\mathcal{S} = \{1, 2, ..., \mathtt{S}\}$ and a set of points $\mathcal{T} = \{1, 2, ..., \mathtt{T}\}$. Their cross product $\mathcal{T} \times \mathcal{S}$ defines a lattice of points. We can spatially order and hence visualize these points on the lattice as shown in figure 1-3. The lattice point indexed by (t, s) can be found by looking at the circle located in the t^{th} row and the s^{th} column.

On this lattice there are measurements. For each lattice point $(t, s) \in \mathcal{T} \times \mathcal{S}$ we have a one bit measurement $Y_{t,s}$ which is either a 0 or a 1. Fix these measurements $Y_{t,s} \forall (t, s) \in \mathcal{T} \times \mathcal{S}$. Let us collect these measurements into a $\mathtt{T} \times \mathtt{S}$ matrix $\underline{\underline{Y}}$ such that $(\underline{\underline{Y}})_{t,s} = Y_{t,s}$.

In this optimization problem we will be interested in finding a vector $\underline{A} \in \mathcal{S}^{\mathtt{T}}$ that maximizes some cumulative value function. The cumulative value function takes as input the vector $\underline{A}$ and outputs a score based on the number of lattice points in the

Figure 1-3: The lattice that defines the space-time landscape for an object's trajectory.



enclosed region defined by $\underline{A}$ that have a 1 measurement bit associated with them and the pattern of those $1s$.

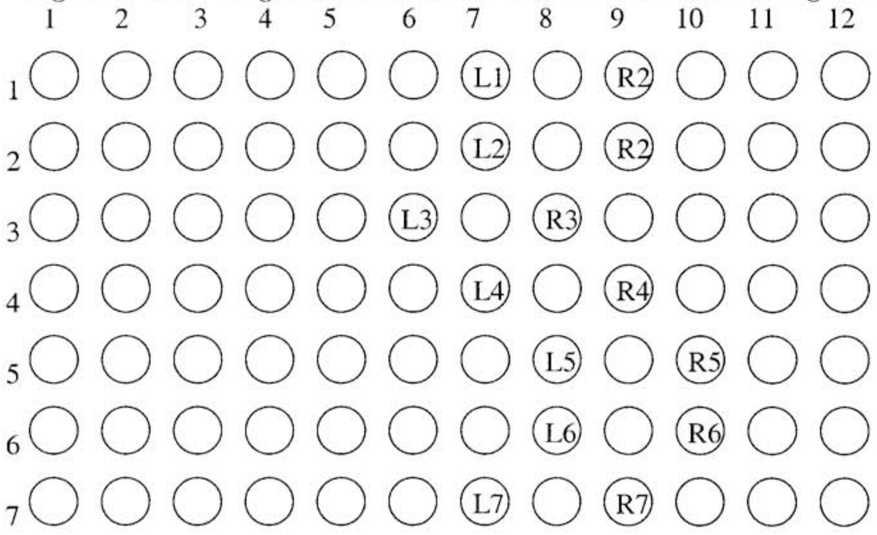
I will describe the closed region that $\underline{A}$ parameterizes on the lattice and then I will later explain the cumulative value function. But first let us look at the space of possible $\underline{A}s$. First fix any point $X_0 \in \mathcal{S}$ such that it is sufficiently far from the boundaries 1 and $\mathtt{S}$. $\underline{A}$ is constrained to be a "continuous" vector in

$$\mathcal{A}(\delta) \triangleq \{\underline{\gamma} \in \mathcal{S}^{\mathrm{T}} \mid |\gamma_j - \gamma_{j-1}| \leq 1, \forall j \in \mathcal{T}; \gamma_0 = X_0 - \lfloor\frac{\delta}{2}\rfloor\} \subset \mathcal{S}^{\mathrm{T}}.$$

So let us define the closed region that $\underline{A} \in \mathcal{A}(\delta)$ parameterizes via a left and right boundary. There is a left boundary, $\underline{L} \in \mathcal{S}^{\mathrm{T}}$, associated with the closed region that $\underline{A}$ parameterizes where $L_t = A_t, \forall t \in \mathcal{T}$. And there is also a right boundary, $\underline{R} \in \mathcal{S}^{\mathrm{T}}$, associated with that region where $R_t = A_t + \delta - 1, \forall t \in \mathcal{T}$.

Now we can visualize the region defined by these left and right boundaries on the lattice in the following way for the given $\underline{A}$ and the subsequently defined left boundary, $\underline{L}$, and right boundary, $\underline{R}$. In figure 1-4 we illustrate with an example where $\delta = 1$, the region has a one unit wide radius. Also we fix $\mathtt{T} = 7$, $\mathtt{S} = 12$, and $\underline{A} = (7, 7, 6, 7, 8, 8, 7)$.

Figure 1-4: A neighborhood in the lattice with left and right boundaries depicted.



At this point I will focus on describing paths that follow the M_1 motion probability model as described earlier in the thesis to avoid complications for people that want to understand the main difficulty of the optimization problem we confront.

Now I shall describe the cumulative value function. First we shall define a path in the region parameterized by $\underline{A}$ in the following way. The set of possible paths contains paths $\underline{p} \triangleq (p_1, p_2, ..., p_\mathrm{T})$ that are defined via these constraints:

$$p_1 = X_0 - 1 \text{ or } p_1 = X_0 \text{ or } p_1 = X_0 + 1 \text{ such that } L_1 \leq p_1 \leq R_2$$

$$p_2 = p_1 - 1 \text{ or } p_2 = p_1 \text{ or } p_2 = p_1 + 1 \text{ such that } L_2 \leq p_2 \leq R_2$$

$$p_3 = p_2 - 1 \text{ or } p_3 = p_2 \text{ or } p_3 = p_2 + 1 \text{ such that } L_3 \leq p_3 \leq R_3$$

$$\ldots$$

$$p_\mathrm{T} = p_{\mathrm{T}-1} - 1 \text{ or } p_\mathrm{T} = p_{\mathrm{T}-1} \text{ or } p_\mathrm{T} = p_{\mathrm{T}-1} + 1 \text{ such that } L_\mathrm{T} \leq p_\mathrm{T} \leq R_\mathrm{T}$$

Thus a region of the lattice parameterized by the vector $\underline{A}$ contains a large subset of the set of all possible paths. Let us define the set of all such possible paths $\underline{p}$ in the region parameterized by $\underline{A}$ or equivalently $\underline{L}$ and $\underline{R}$ as $N_\delta(\underline{A})$. More formally,

$$N_\delta(\underline{A}) = \{\underline{p} \in \mathcal{S}^{\mathrm{T}} \mid |p_1 - X_0| \leq 1, |p_2 - p_3| \leq 1, ..., |p_{\mathrm{T}} - p_{\mathrm{T}-1}| \leq 1;$$

$$L_t = A_t, R_t = A_t + \delta - 1, L_t \leq p_t \leq R_t, \forall t \in \mathcal{T}\}.$$

Now let us define a value function $v(\underline{p}) : \mathcal{S}^{\mathrm{T}} \rightarrow \mathcal{Q}$. $v(\underline{p})$ takes as input a path $\underline{p}$ and outputs a number that is exponentially related to the number of 1s that lie on the path, $\underline{\mathrm{p}}$. Namely,

$$v(\underline{p}) = (\frac{1-\alpha}{\alpha})^{2\sum_{t=1}^{\mathrm{T}} Y_{t,p_t}}.$$

Now the cumulative value function, $c(\underline{A}) : \mathcal{A}(\delta) \rightarrow \mathcal{Q}$, cumulates the value functions of all paths that lie in the region defined by $N_\delta(\underline{A})$ in the following way.

$$c(\underline{A}) = \sum_{\underline{p} \in N_\delta(\underline{A})} v(\underline{p})$$

Now the problem can be stated precisely as finding the vector $\underline{A}^*$, which parameterizes a neighborhood region, from the set $\mathcal{A}(\delta)$, that maximizes the cumulative value function $c(\underline{A})$.

Therefore, $\underline{A}^* = arg\max_{\underline{A} \in \mathcal{A}(\delta)} c(\underline{A})$.

Chapter 2

Movement Range=Neighborhood Size=1

2.1 Introduction

In this chapter we will utilize the M_1 prior "motion probability model" which allows the object to move one unit to the left or right, or to stay in the same position at each point in time and additionally we will fix our neighborhood size to be $\delta = 1$. As with many reductionist engineering approaches, understanding this special case and developing an algorithm for it will lead us to algorithms that will generalize to other prior "motion probability models" M_Δ and neighborhood sizes, δ.

Under these conditions, let us find an efficient algorithm that will calculate the parameter $\underline{A}(\underline{\underline{Y}}) \in \mathcal{A}(M_1, \delta = 1)$ for the MAP neighborhood $N_\delta(\underline{A})$ where

$$\underline{A}(\underline{\underline{Y}}) = arg \max_{\underline{a} \in \mathcal{A}(M_1,\delta)} P[\underline{X} \in N_\delta(\underline{a}) \mid \underline{\underline{Y}}] = arg \max_{\underline{a} \in \mathcal{A}(M_1,1)} \sum_{\underline{\gamma} \in N_1(\underline{a})} \prod_{t=1}^{\mathtt{T}} \prod_{s=1}^{\mathtt{S}} P(Y_s(t) \mid \gamma_t).$$

We will find an efficient algorithm that finds $\underline{A}(\underline{\underline{Y}})$ and analyze the algorithm's complexity. Then we will discuss how to calculate

$$P[\underline{X} \in N_{\delta=1}(\underline{A}) \mid \underline{\underline{Y}}] = \frac{\sum_{\underline{\gamma} \in N_1(\underline{A})} P(\underline{\underline{Y}} \mid \underline{\gamma})}{\sum_{\underline{\gamma} \in \mathcal{P}(M_1)} P(\underline{\underline{Y}} \mid \underline{\gamma})}$$

which at first glance looks like it is not computable in polynomial time since the denominator is a sum over an exponential number of paths (in T) regardless of the neighborhood parameter, $\underline{A}$, or the neighborhood, $N_\delta(\underline{A})$.

Then we will end this chapter with some experimental results and a generalization of the algorithm to motion in more than one dimension so that we can begin to consider tracking two dimensional motions such as a ping pong ball moving against the black background.

2.2 Algorithm for finding a MAP neighborhood

2.2.1 Mathematical simplification resulting from $\delta = 1$

If the neighborhood size at each point in time is $\delta = 1$, then the neighborhood $N_1(\underline{a})$ parameterized by $\underline{a} \in \mathcal{A}(M_1, 1)$ has only one path, namely the parameter $\underline{a}$. That is $N_1(\underline{a}) = \{\underline{a}\}$. So our optimization problem simplifies in the following manner:

$$\underline{A}(\underline{\underline{Y}}) = arg \max_{\underline{a} \in \mathcal{A}(M_1,1)} \prod_{t=1}^{\mathtt{T}} \prod_{s=1}^{\mathtt{S}} P(Y_s(t) \mid a_t) = arg \max_{\underline{a} \in \mathcal{P}(M_1)} (\frac{1-\alpha}{\alpha})^{2\sum_{t=1}^{\mathtt{T}} Y_{t,a_t}}$$

$$= arg \max_{\underline{a} \in \mathcal{P}(M_1)} (\sum_{t=1}^{\mathtt{T}} Y_{t,a_t}).$$

The above simplification assumes that $\alpha \leq .5$, which we can assume without loss of generality. How do we find the best path, $\underline{A}(\underline{\underline{Y}})$ amongst the exponentially many (in T) possible paths in $\mathcal{P}(M_1)$?

Can the optimization problem be separated in time so that we find the best path up to time n and use the results to efficiently find the best path up to time $n+1$? In order to investigate this possibility let us formally define the set of paths starting at time step 1 and stopping at time step t that follow motion model M_Δ as

$$\mathcal{P}_t(M_\Delta) = \{\underline{\gamma} \in \mathcal{S}^t \mid \gamma_0 = X_0; |\gamma_j - \gamma_{j-1}| \leq \Delta, 1 \leq j \leq t\} \subset \mathcal{S}^t.$$

The ability to separate the optimization problem in time means that in order to find the best path for time steps 1 through $n+1$, $arg \max_{\underline{a} \in \mathcal{P}_{n+1}(M_1)}(\sum_{t=1}^{n+1} Y_{t,a_t})$, we would use the results from finding the best path for time steps 1 through n, $arg \max_{\underline{a} \in \mathcal{P}_n(M_1)}(\sum_{t=1}^{n} Y_{t,a_t})$.

Indeed the problem can be separated in time in such a way and that ability to divide up the problem in time is what allows us to devise an efficient algorithm to solve the problem. For example, without loss of generality assume that $\alpha \leq .5$. In that case, we want to find the path with the most number of $1s$ on its path. We can divide up the problem in time in the following manner. At time step n, if for each reachable location in space, we know the best path that goes through it taking into consideration only measurements up to time n (ie the path with the most number of $1s$ on its path up to time step n), then we can calculate the best path going through each of the reachable locations in space at time step $n+1$ using measurements at time step $n+1$ and the best path going through each reachable location at time step n using only measurements up to time step n. This is due to the fact that the above sum can be broken up in time. This will all be made more formal, but the idea behind the algorithm can be summarized as optimizing for each reachable location in space at each time step using only the previous time steps optimization results. Keep this in mind as you read the following formalization.

2.2.2 Defining the Algorithm

First, let us define the set of possible paths that start at position X_0 and end at position s at time t obeying the proposed motion model M_1 as

$$\mathcal{P}(t,s) = \{\underline{\gamma} \in \mathcal{S}^t \mid \gamma_0 = X_0; |\gamma_j - \gamma_{j-1}| \leq 1, 1 \leq j \leq t; \gamma_t = s\}.$$

Therefore, an object whose trajectory commences at position X_0 and ends up at position s at time step t could have taken any path $\underline{p} \in \mathcal{P}(t,s) \subseteq \mathcal{S}^t$ to reach that position. Similarly let us remind ourselves of the definition for the set of possible paths that start at position X_0 and terminate at time t under motion model M_1:

$$\mathcal{P}_t(M_\Delta) = \{\underline{\gamma} \in \mathcal{S}^t \mid \gamma_0 = X_0; |\gamma_j - \gamma_{j-1}| \leq \Delta, 1 \leq j \leq t\} \subset \mathcal{S}^t.$$

Also let us define the set of possible locations for the object at time $1 \leq t \leq \mathtt{T} - 1$ given the object's position at time $t+1$ is l as

$$\mathcal{L}(t+1, l) = \{s \in \mathcal{S} \mid |s - X_0| \leq t, |s - l| \leq 1\}.$$

For example, an object that is at position l_4 at time step 4 could have come from any position $l_3 \in \mathcal{L}(4, l)$ in the previous time step (as dictated by the motion model M_1). Finally, let us define the set of possible locations at final time $\mathtt{T}$ as $\mathcal{L}_\mathtt{T} = \{s \in \mathcal{S} \mid |s - X_0| \leq \mathtt{T}\}$ and let us define the set of possible locations at time t as $\mathcal{L}_t = \{s \in \mathcal{S} \mid |s - X_0| \leq t\}$.

We now need to define some functions to keep track of the results of our optimization at each time step with the aim of reusing these results in the next time step of the optimization (which will have been "separated in time"). Let $M_n(s) : \mathcal{S} \to \mathcal{S}^n$ be a sequence of functions for $1 \leq n \leq \mathtt{T}$ such that

$$M_n(s) = arg \max_{\underline{a} \in \mathcal{P}(n,s)} P[\underline{a} \mid \underline{\underline{Y}}] = arg \max_{\underline{a} \in \mathcal{P}(n,s)} (\sum_{t=1}^{n} Y_{t,a_t})$$

$M_n(s)$ represents a path that maximizes the objective function $f_n(\underline{a}) \triangleq (\sum_{t=1}^{n} Y_{t,a_t})$ over all $\underline{a} \in \mathcal{P}(n, s)$. We are interested in searching amongst the paths in the set $\{\underline{\gamma} : \underline{\gamma} = M_\mathtt{T}(s), s \in \mathcal{L}_\mathtt{T}\}$ and outputting a path that maximizes $f(\underline{a}) = (\sum_{t=1}^{\mathtt{T}} Y_{t,a_t})$. In order to devise an efficient algorithm to calculate $M_\mathtt{T}(s) \forall s \in \mathcal{L}_\mathtt{T}$, we will use use $M_t(s) \forall s \in \mathcal{L}_t$ in order to determine $M_{t+1}(s) \forall s \in \mathcal{L}_{t+1}$ for all times $t \in \mathcal{T}$. That is the idea behind separating the optimization problem in time. Now why and how can this be done?

It can be accomplished because of the mathematical structure of the objective function $f_n(\underline{a}) = (\sum_{t=1}^{n} Y_{t,a_t})$ and the fact that the measurements are non-negative.

More formally, it can be shown that

$$M_n(s) = CONCAT[arg \max_{M_{n-1}(l), l \in \mathcal{L}(n,s)} f_{n-1}(M_{n-1}(l)), s]$$

where $CONCAT$ is a function that takes two finite vectors and outputs a vector that is the concatentation of the two. The validity of the above step can be ascertained via a proof by induction.

So the pseudo code for the algorithm that will solve the optimization problem can be written down concisely:

```
$M_1(X_0 - 1) = (X_0 - 1);$
$M_1(X_0) = (X_0);$
$M_1(X_0 + 1) = (X_0 + 1);$
for $t = 2$ to T
begin
        for $s \in \mathcal{L}_t \subseteq \mathcal{S}$
        begin
                $M_t(s) = CONCAT(arg\max_{M_{t-1}(l), l \in \mathcal{L}(t,s)} f_{t-1}(M_{t-1}(l)), s);$
        end
end
return $arg\max_{M_T(l), l \in \mathcal{L}_T} f_T(M_T(l));$
```

The above algorithm finds the optimizing route through every node at each point in time and works its way down in time, reusing the work from the previous time step. In that sense we have been able to "separate the problem in time". An actual implementation of the algorithm is coded in MATLAB and included in Appendix A. Note that in order to achieve maximal performance with that implementation the for loops must be transformed to matrix operations. However the code serves the purpose of defining the pseudo code in more detail.

2.2.3 Complexity Calculations

Again we are assuming $\mathtt{S}$ is much larger than $\mathtt{T}$. If this is the case then the above algorithm looks at only a pyramid of points in the lattice (which are reachable from X_0). Therefore, there are $O(\frac{\mathtt{T}(\mathtt{T}+1)}{2}) = O(\mathtt{T}^2)$ points. For any given point in the pyramid the algorithm does at most three comparisons of objective function values for different paths that could have reached the given point. Therefore the algorithm takes $O(\mathtt{T}^2)$ time.

In terms of space complexity, at any given time step t we only need to keep track of $M_{t-1}(l) \forall l \in \mathcal{L}_{t-1}$. Since the cardinality of the set $\mathcal{L}_{t-1}$ is $2(t-1)+1 = O(t)$ and $M_{t-1}(l)$ is just a $(t-1)$-vector of numbers in $\mathcal{S}$, we have that the space requirements are $O(\mathtt{T}^2 log(\mathtt{S}))$.

2.3 Calculating the probability in polynomial time

Now we have an algorithm that calculates the parameter, $\underline{A}$, of the MAP neighborhood for the special case when $\delta = 1$. But how do we calculate

$$P[\underline{X} \in N_\delta(\underline{A}) \mid \underline{\underline{Y}}] = \frac{\sum_{\underline{\gamma} \in N_\delta(\underline{A})} P(\underline{\underline{Y}} \mid \underline{\gamma})}{\sum_{\underline{\gamma} \in \mathcal{P}(M_1)} P(\underline{\underline{Y}} \mid \underline{\gamma})}$$

for the special case of $\delta = 1$ which initially appears to not be computable in polynomial time since the denominator is a sum over an exponential number of paths (in $\mathtt{T}$) regardless of the parameter, $\underline{A}$, or the neighborhood, $N_\delta(\underline{A})$.

Is the numerator computable in polynomial time? It should be because it is the probability of a single path. And indeed the numerator simplifies since $\delta = 1$ so that the δ-neighborhood with parameter $\underline{A}$ is precisely the vector $\underline{A}$ and therefore,

$$\begin{aligned}
\textstyle\sum_{\underline{\gamma}\in N_1(\underline{A})} P(\underline{\underline{Y}} \mid \underline{\gamma}) &= P(\underline{\underline{Y}} \mid \underline{A}) \\
&= \textstyle\prod_{t=1}^{\mathsf{T}} \prod_{s=1}^{\mathsf{S}} P(Y_s(t) \mid A_t) \\
&= \textstyle\prod_{t=1}^{\mathsf{T}} (1-\alpha)^{N(t,\underline{Y}(t),A_t)} \alpha^{(S-N(t,\underline{Y}(t),A_t))} \\
&= (1-\alpha)^{\sum_{t=1}^{\mathsf{T}} N(t,\underline{Y}(t),A_t)} \alpha^{\sum_{t=1}^{\mathsf{T}}(S-N(t,\underline{Y}(t),A_t))} \\
&= (1-\alpha)^{T(S-1)-sum(\underline{\underline{Y}})+2\sum_{t=1}^{\mathsf{T}} Y_{t,A_t}} \alpha^{T+sum(\underline{\underline{Y}})-2\sum_{t=1}^{\mathsf{T}} Y_{t,A_t}} \\
&= (1-\alpha)^{TS} \left(\tfrac{1-\alpha}{\alpha}\right)^{-T-sum(\underline{\underline{Y}})} \left(\tfrac{1-\alpha}{\alpha}\right)^{2\sum_{t=1}^{\mathsf{T}} Y_{t,A_t}}
\end{aligned}$$

where $sum(\underline{\underline{Y}}) \triangleq \sum_{t=1}^{\mathsf{T}} \sum_{s=1}^{\mathsf{S}} Y_{t,s}$. So indeed the numerator is computable in polynomial time.

However, is the denominator computable in polynomial time? How can an algorithm efficiently count and determine $\sum_{\underline{\gamma}\in\mathcal{P}(M_1)} P(\underline{\underline{Y}} \mid \underline{\gamma})$ without going through all the possible paths $\underline{\gamma} \in \mathcal{P}(M_1)$?

If we can change the sum over paths into a sum over equivalence classes for paths where the number of equivalence classes is small (namely polynomial in T and S) and each path, $\underline{x}$, in an equivalence class has the same conditional probability, $P(\underline{\underline{Y}} \mid \underline{x})$, (so that the sum over the equivalence classes is easy to calculate) then we may be closer to an answer. But what equivalence relation (property) do we need to have a "favorable" structure for our equivalence classes? We need an equivalence relation (property) such that the number of equivalence classes for paths is polynomial (in S and T) and such that it is easy to count the number of paths in each class.

So if we find a property for a path $\underline{x}$ such that

- two paths $\underline{x}^{(1)}$ and $\underline{x}^{(2)}$ are in the same equivalence class if they have the same property,
- $P(\underline{\underline{Y}} \mid \underline{x})$ is the same for all paths $\underline{x}$ in the same equivalence class and efficient to calculate for any path in any equivalence class,
- there are a polynomial number of equivalence classes (in S and T),
- and the number of paths in each equivalence class can be calculated efficiently,

for example "recursively" or "iteratively" in polynomial time (in $\mathtt{S}$ and $\mathtt{T}$), then the transformation of the sum will allow us to efficiently calculate the denominator.

There does exist such a property (that induces an equivalence relation) for any path $\underline{x} \triangleq (x_1, x_2, ..., x_{\mathcal{T}}) \in \mathcal{P}(M_1)$. That property relates the path, $\underline{x}$, to the measurement data, $\underline{\underline{Y}}$. Specifically, define

$$I(\underline{x}) \triangleq \sum_{t=1}^{\mathtt{T}} Y_{t,x_t}$$

so that $I(\underline{x})$ is the number of 1s on the path $\underline{x}$.

Then we can define the equivalence relation:

$$\underline{x}^{(1)} \sim \underline{x}^{(2)} \Leftrightarrow I(\underline{x}^{(1)}) = I(\underline{x}^{(2)}).$$

Intuitively the number of 1s on the path $\underline{x}$ is related to the number of undamaged measurement bits, $\sum_{t=1}^{\mathtt{T}} N(t, \underline{Y}(t), x_t)$, if $\underline{x}$ was indeed the true path. Namely,

$$\sum_{t=1}^{\mathtt{T}} N(t, \underline{Y}(t), x_t) = T(S-1) - sum(\underline{\underline{Y}}) + 2\sum_{t=1}^{\mathtt{T}} Y_{t,x_t}.$$

Therefore the number of 1s is related to the number of undamaged measurement bits via the addition of a constant term that is independent of the assumed path, $\underline{x}$. Now we can index the equivalence classes uniquely in the following way: take a path $\underline{x}$ in any equivalence class, then the equivalence class $[\underline{x}]$ will have an integer index $k = \sum_{t=1}^{\mathtt{T}} Y_{t,x_t} \in \{1, 2, 3, ..., \mathtt{T}\}$. And I define the cardinality of the equivalence class with index k, $[k]$, to be $C(k)$.

So my point is that given $\underline{\underline{Y}}$ and $I(\underline{x})$, we can rewrite $\sum_{\underline{x}\in\mathcal{P}(M_1)} P(\underline{\underline{Y}} \mid \underline{x})$ as

$$\sum_{\underline{x}\in\mathcal{P}(M_1)} P(\underline{\underline{Y}} \mid \underline{x}) = \sum_{k=0}^{\mathtt{T}} C(k)(1-\alpha)^{TS}(\frac{1-\alpha}{\alpha})^{-T-sum(\underline{\underline{Y}})}(\frac{1-\alpha}{\alpha})^{2k}, \forall t \in \mathcal{T}$$

Note that our first three requirements for the equivalence relation were needed in order to transform the sum over an exponential number of objects to a sum over a polynomial number of objects (in $\mathtt{T}$). It only remains to show that our fourth requirement can be met so that $C(k)$ can be calculated efficiently for all $k \in \{0, 1, 2, 3, ..., \mathtt{T}\}$, hence allowing us to calculate $\sum_{\underline{x} \in \mathcal{P}(M_1)} P(\underline{\underline{Y}} \mid \underline{x})$ efficiently.

Is there an efficient way to calculate $C(k)$ for all $k \in \{0, 1, 2, 3, ..., \mathtt{T}\}$ using the set of all paths $\underline{X} \in \mathcal{P}(M_1)$? Again that seems difficult due to the fact that there are an exponential number of possible paths in $\mathcal{P}(M_1)$. Nonetheless, the following algorithm is inspired from the algorithm for calculating Pascal's triangle. First let me recursively define

$$C^t(s,k) \triangleq \sum_{l=-1}^{1} C^{(t-1)}(s+l, k - Y_{t,s})$$

$$\text{where } C^t(s,k) = 0, \forall s \in \mathcal{S}, \forall k \in \{0, 1, ..., \mathtt{T}\} \text{ if } s \geq (X_0 + t) \text{ or } s \leq (X_0 - t)$$

$$\text{and } C^0(s,k) = \left\{ \begin{array}{lll} 1 & \text{if} & k = 0, s = X_0 \\ 0 & \text{otherwise} & \end{array} \right\}$$

$$\text{such that } C(k) = \sum_{s \in \mathcal{S}} C^{\mathtt{T}}(s,k).$$

Note that $C^t(s,k)$ is the number of paths through time t that go through pixel s and have colored k pixels as $1s$ along their paths.

2.3.1 Complexity calculations

We can do the counting iteratively at each time step so that calculating $C^{\mathtt{T}}(s,k) \forall s \in \mathcal{S}, \forall k \in \{0, 1, ..., \mathtt{T}\}$ can be done in time $O(\mathtt{T}^3)$. This is possible because at each time step t we update order t^2 function values corresponding to the set $\{C^t(s,k) : s \in \{X_0 - t, X_0 - t + 1, ..., X_0 + t\}; k \in \{0, 1, 2, ..., t\}$.

The space requirement of this algorithm is $O(\mathtt{T}^2)$. This can be seen because at each time t the algorithm only needs to keep track of order t pixels (the ones that may have paths going through them) and for each of those pixels indexed by (t, s) we need to keep track of the values $\{C^t(s,k) : k \in \{0, 1, 2, ..., t\}$ which sum to at most 3^t

(which requires at most t bits).

So indeed, $C(k) \forall k \in \{0, 1, ..., \mathtt{T}\}$ can be calculated efficiently. And we therefore have an efficient algorithm that calculates $\sum_{\underline{x} \in \mathcal{P}(M_1)} P(\underline{\underline{Y}} \mid \underline{x})$ without enumerating the probabilities associated with each of the possible paths $\underline{x}$ in the large set $\mathcal{P}(M_1)$.

2.4 Simulation

In figure 2-1 the top subplot shows the object's path, the middle subplot shows the measurement matrix corrupted with shot noise, and the bottom subplot shows the estimate for the object's path. Regardless of the high noise level, in this particular instance the estimate nearly tracked the object's actual trajectory.

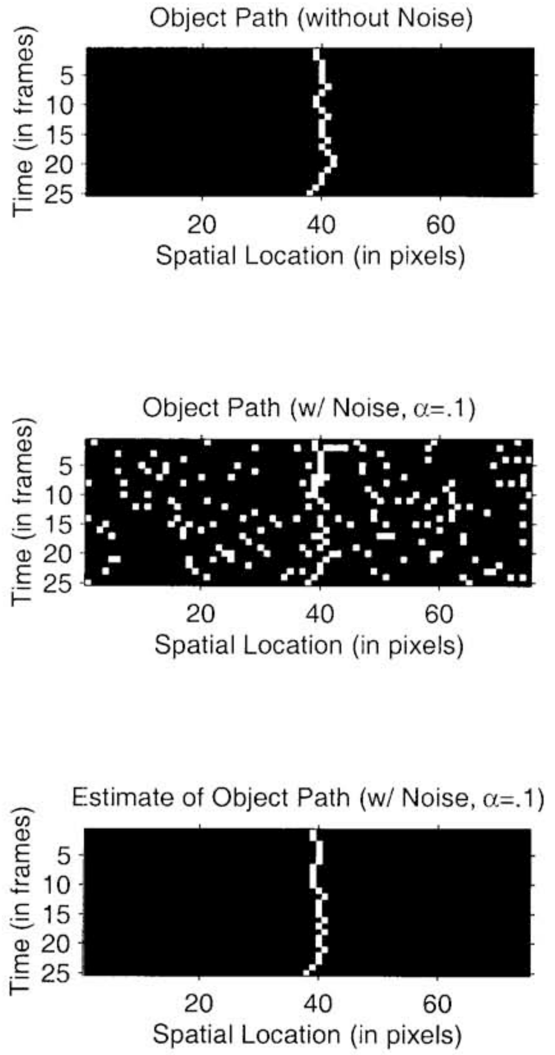
However, the probability of the trajectory equalling the object's actual path given the noisy measurements is only 0.0167 and many other trajectories have a relatively close probabilty associated with them. But if we put a neighborhood that is centered about this estimate and that contains five pixels at each point in time, the probabilty that this particular tube contains the object's actual trajectory for all time is 0.9655. Note that this probability is a lower bound for the probability associated with the best tube that has a five pixel neighborhood size at each point in time.

Now in one simulation, I ran the problem independently 500 times with noise $\alpha = .1$ and found the best paths and the their associated probabilities. Let us look at a histogram of those values on the bottom subplot of figure 2-2. Notice that the values have a rather wide experimental variance compared to the histogram of the probabilities associated with the tubes (with $\delta = 5$) that are placed around those paths. The histogram of the probabilities associated with the tubes is illustrated in the top subplot of figure 2-2.

2.5 A generalization to motion in two dimensions

The results we have obtained thus far can be extended to an object moving in more than one dimension. For notational simplicity, I will show how to extend the results

Figure 2-1: The noise-free and noisy measurements along with the associated estimate for the object's sample trajectory.

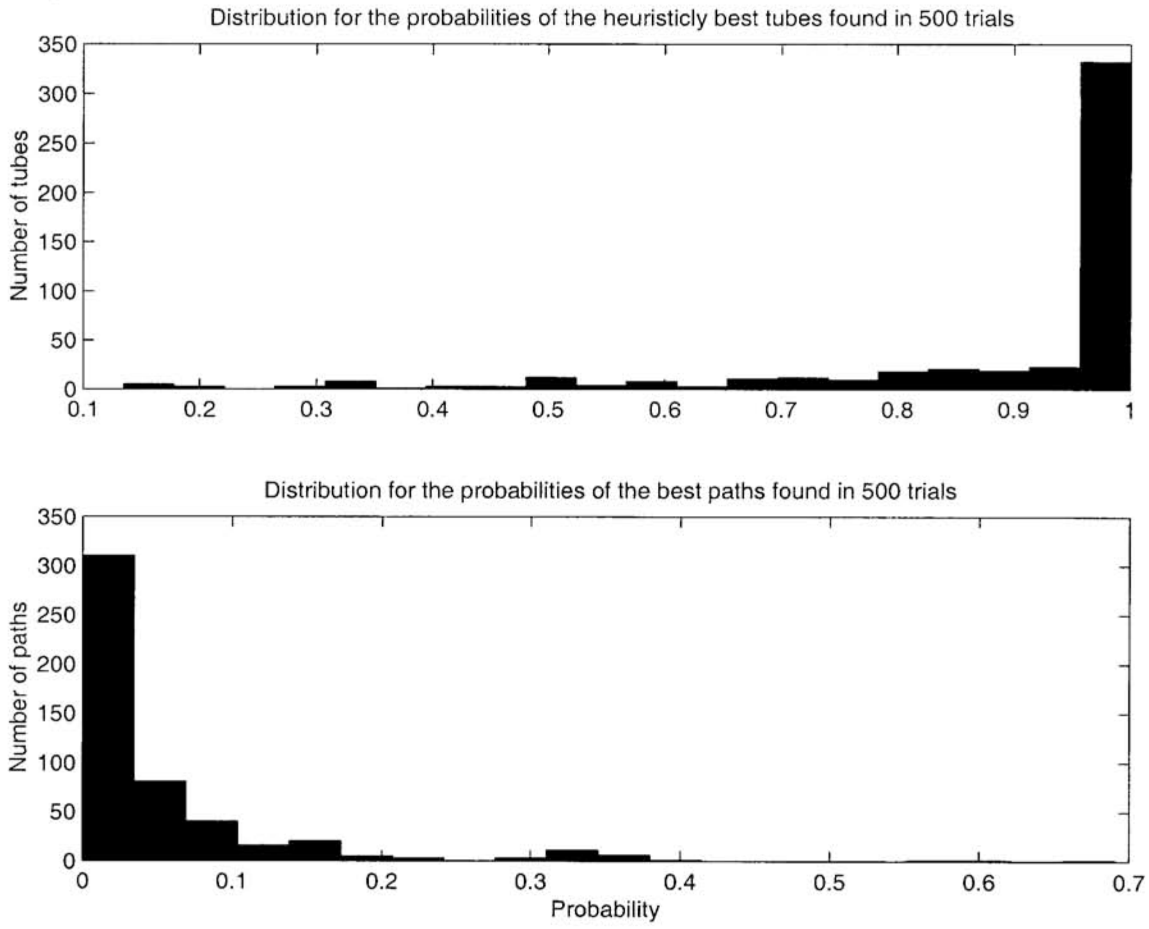


for an object moving in two dimensional euclidean space. Other dimensions and spaces can be approached in a similar fashion.

2.5.1 Setting up the problem

Let us use the formalism we have already setup and simply change the object $\mathcal{S}$ and the objects that pertain to that set. We will still use a set $\mathcal{S}$ of points. But these points will now be in some bounded region of two dimensional euclidean space as opposed to some bounded region of one dimensional euclidean space. Therefore let

Figure 2-2: The histogram of probability values associated with the best paths and neighborhood placed around them.



us formally redefine $\mathcal{S} = \{(m,n) \mid 1 \leq m,n \leq \mathtt{S}, m,n \in \mathbb{N}\}$. An object's location at time t is represented by (m_t, n_t). And the path of the object is stored in the vector $\underline{X} \triangleq (\underline{X_1}, \underline{X_2}, ..., \underline{X_\mathtt{T}})$, where $\underline{X_t} \triangleq (m_t, n_t)$.

We have a three dimensional cube of measurements. Namely because for each time instant, we have a two dimensional lattice of measurements, corresponding to the two dimensional pixel value measurements. Let us denote our measurements by the $\mathtt{T} \times \mathtt{S} \times \mathtt{S}$ matrix $\underline{\underline{\underline{Y}}}$.

Now we make our usual simplifying assumptions with slight twists:

- Let us assume the initial position $\underline{X_0} \in \mathcal{S}$ is known and choose the two dimensional prior "motion probability model", $M_1^{(2D)}$, that states that the object moves west, southwest, south, southeast, east, northeast, north, northwest, or stays in the same location with equal probability:

$$P(\underline{X_{t+1}} \mid \underline{X_t}) = \left\{ \begin{array}{lll} \frac{1}{9} & \text{if} & m_{t+1} = m_t - 1, n_{t+1} = n_t \\ \frac{1}{9} & \text{if} & m_{t+1} = m_t - 1, n_{t+1} = n_t - 1 \\ \frac{1}{9} & \text{if} & m_{t+1} = m_t, n_{t+1} = n_t - 1 \\ \frac{1}{9} & \text{if} & m_{t+1} = m_t + 1, n_{t+1} = n_t - 1 \\ \frac{1}{9} & \text{if} & m_{t+1} = m_t + 1, n_{t+1} = n_t \\ \frac{1}{9} & \text{if} & m_{t+1} = m_t + 1, n_{t+1} = n_t + 1 \\ \frac{1}{9} & \text{if} & m_{t+1} = m_t, n_{t+1} = n_t + 1 \\ \frac{1}{9} & \text{if} & m_{t+1} = m_t - 1, n_{t+1} = n_t + 1 \\ \frac{1}{9} & \text{if} & m_{t+1} = m_t, n_{t+1} = n_t \\ 0 & \text{otherwise} & \end{array} \right\}, 1 \leq t \leq (\mathtt{T}{-}1).$$

Therefore, the true path $\underline{X}$ is a member of the set of possible paths

$$\mathcal{P}(M_1^{(2D)}) = \{\underline{\gamma} \in \mathcal{S}^{\mathtt{T}} \mid \underline{\gamma_0} = \underline{X_0}, |\underline{\gamma_t} - \underline{\gamma_{t-1}}| \leq 1, \forall t \in \mathcal{T}\} \subset \mathcal{S}^{\mathtt{T}}.$$

Note that the norm used in the definition for the set of possible paths is a discrete norm whose isocontours are discrete points on the set of squares that surround the origin.

- Also let us formalize the two dimensional notion of δ-neighborhoods. A two dimensional δ-neighborhood, $N_\delta(\underline{a})$, is a neighborhood with δ pixels in the x-direction and δ pixels in the y-direction, at each time instant, parameterized by the path that occupies the lower left corner of the neighborhood (inside the neighborhood) for all time or more formally,

$$N_\delta(\underline{a}) \triangleq \{\underline{\gamma} \in \mathcal{P}(M_1^{(2D)}) \mid 0 \leq |\underline{\gamma_t} {-} \underline{a_t}| \leq \delta{-}1, \forall t \in \mathcal{T}; (\underline{a_t})_x \geq (\underline{\gamma_t})_x, (\underline{a_t})_y \geq (\underline{\gamma_t})_y\}.$$

- We shall impose the continuity constraint on the neighborhood parameters, $\underline{a}$, contained in the set $\mathcal{A}(M_1^{(2D)}, \delta)$ that parameterizes the neighborhoods $N_\delta(\underline{a})$.

Namely we will impose the constraint

$$\mathcal{A}(M_1^{(2D)},\delta) \triangleq \{\underline{\gamma} \in \mathcal{S}^{\mathrm{T}} \mid |\underline{\gamma_j} - \underline{\gamma_{j-1}}| \leq 1, \forall j \in \mathcal{T}; \underline{\gamma_0} = \underline{X_0} - (\lfloor \frac{\delta}{2} \rfloor, \lfloor \frac{\delta}{2} \rfloor)\} \subset \mathcal{S}^{\mathrm{T}}.$$

- As usual let us assume our measurements are binary such that the quantized image intensity measurements come from the set $\mathcal{Q} = \{0, 1\}$.
- Next, let us assume the object's length is one pixel wide ($\mathtt{L} = 1$).
- Now, let us assume a noise-free measurement model that states that if the object is present at position $\underline{X_t}$ at time t, then we should measure an image intensity of 1 at that location in space-time and for all other locations at time t we should measure a 0. So let us define a simple object template, $o(\underline{s})\colon \mathbb{Z} \times \mathbb{Z} \to \mathcal{Q}$, where

$$o(\underline{s}) \triangleq \left\{ \begin{array}{lll} 1 & \text{if} & \underline{s} = (s_x, s_y) = (0,0) \\ 0 & \text{otherwise} & \end{array} \right\}$$

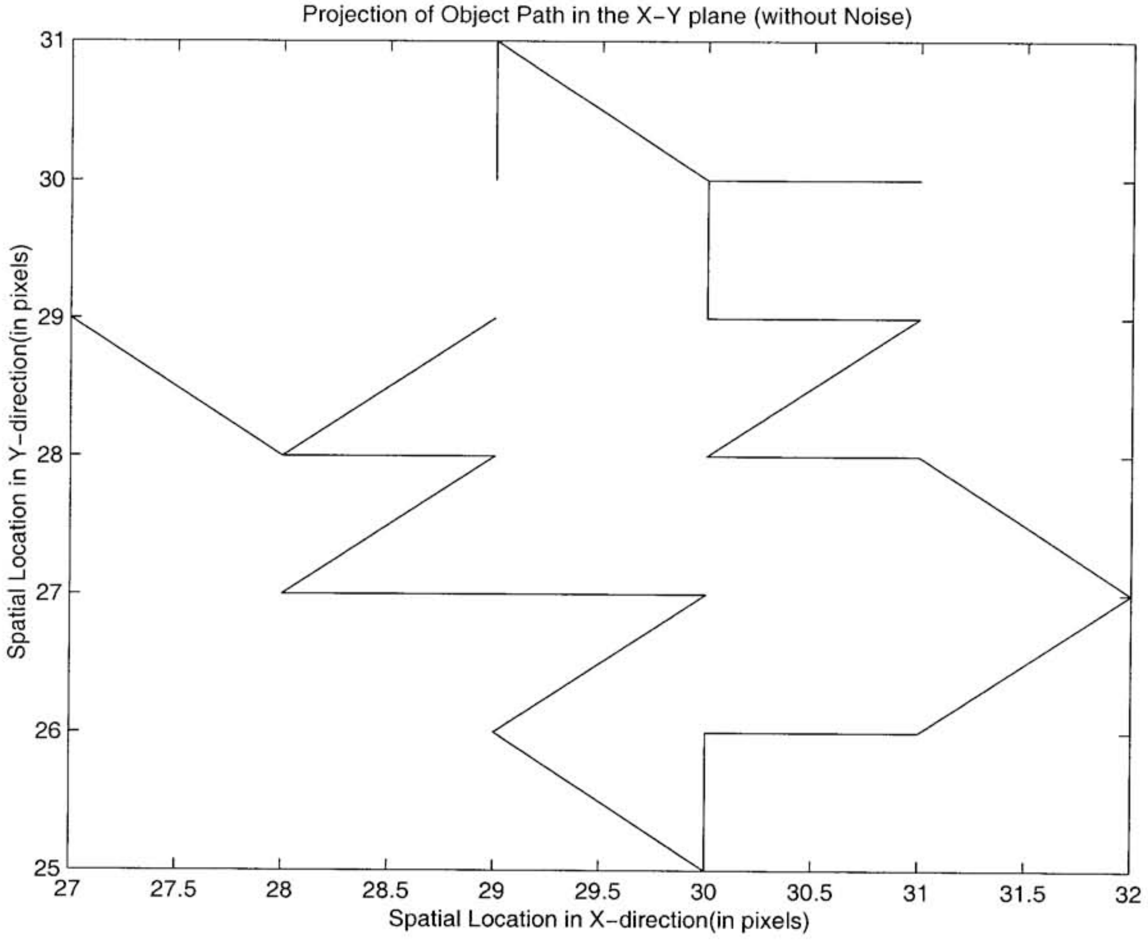
  Figure 2-3 shows the noise-free measurements associated with an object's path as projected into the XY plane (without keeping track of time).

  Figure 2-4 shows the noise-free measurements associated with an object's path projected into the X-time plane.

  And similarly figure 2-5 shows the noise-free measurements associated with an object's path projected into the Y-time plane.

Let us utilize a simple noisy "measurement probability model", generalized for two dimensions, that states that the noise corrupts the measurements by flipping bits. Relating the template to our measurements and the object's trajectory, $Y_{\underline{s}}(t) = (o(\underline{s} - \underline{X_t}) + n_{\underline{s}}(t)) \bmod 2$, such that $n_{\underline{s}}(t)\colon \mathcal{T} \to \mathcal{Q}$ is a discrete-time noise process where $n_{\underline{s_1}}(t_1)$ and $n_{\underline{s_2}}(t_2)$ are iid Bernoulli random variables $\forall(\underline{s_1}, t_1), (\underline{s_2}, t_2) \in \mathcal{S} \times \mathcal{T}$. More specifically let $P(n_{\underline{s}}(t) = 1) = \alpha$ and $P(n_{\underline{s}}(t) = 0) = 1 - \alpha$.

Figure 2-3: The noise-free measurement matrix associated with an object's sample trajectory as projected onto the XY plane.



This implies that

$$P(Y_{\underline{s}}(t)|X_t) = \left\{ \begin{array}{lll} 1-\alpha & \text{if} & \underline{X_t} = \underline{s} \text{ and } Y_{\underline{s}}(t) = 1 \\ \alpha & \text{if} & \underline{X_t} = \underline{s} \text{ and } Y_{\underline{s}}(t) = 0 \\ 1-\alpha & \text{if} & \underline{X_t} \neq \underline{s} \text{ and } Y_{\underline{s}}(t) = 0 \\ \alpha & \text{if} & \underline{X_t} \neq \underline{s} \text{ and } Y_{\underline{s}}(t) = 1 \end{array} \right\}, \forall (\underline{s}, t) \in \mathcal{S} \times \mathcal{T}.$$

Note that we still have a one dimensional signal. Only the dimension of the space it is embedded in has more dimensions and hence the number of pixels in which noise can occur has increased. Yet the number of "signal" pixels has remained constant (namely `T`). However if we had used a noise model that wasn't merely shot noise, but also localization noise, depending on the radius of the the localization noise, we

Figure 2-4: The noise-free measurement matrix associated with an object's sample trajectory as projected onto the XY plane.

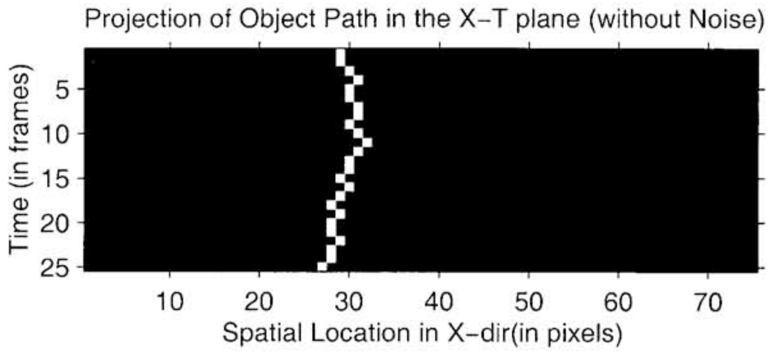
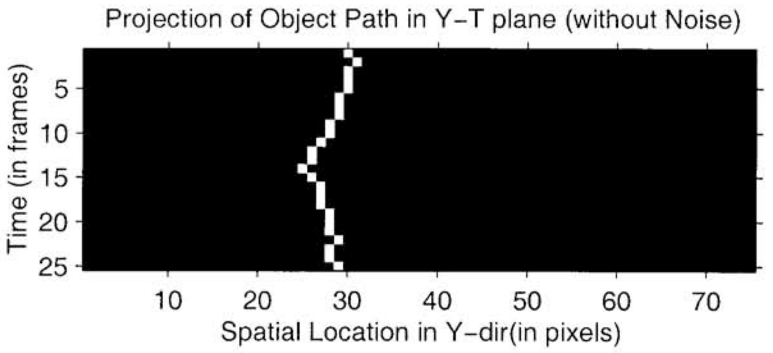


Figure 2-5: The noise-free measurement matrix associated with an object's sample trajectory as projected onto the XY plane.



would have many more signal pixels that would increase with each dimension we add to the problem.

Lastly in order to fully write out the measurement model, $P(\underline{\underline{Y}} \mid \underline{X})$, lets look at the consequences of the probabilistic model we chose:

(A) The image measurements at time t only depend on the position measurements at time t and not on any other times. Formally, $\underline{\underline{Y}}(t_1)$ and $\underline{X_{t_2}}$ are conditionally independent given $\underline{X_{t_1}}$ if $t_1 \neq t_2$.

(B) Also $\underline{\underline{Y}}(t_1)$ and $\underline{\underline{Y}}(t_2)$ are conditionally independent given $\underline{X_{t_1}}$ if $t_1 \neq t_2$.

(C) And $Y_{\underline{s_1}}(t)$ and $Y_{\underline{s_2}}(t)$ are conditionally independent given $\underline{X_t}$.

This implies

$$\begin{aligned}
P(\underline{\underline{Y}} \mid \underline{X}) &= P(\underline{\underline{Y}}(\mathtt{T}), ..., \underline{\underline{Y}}(1) \mid \underline{X}) \\
&= [\textstyle\prod_{t=2}^{\mathtt{T}} P(\underline{\underline{Y}}(t) \mid \underline{\underline{Y}}(t-1), ..., \underline{\underline{Y}}(1), \underline{X})] P(\underline{\underline{Y}}(1) \mid \underline{X}) \\
&= [\textstyle\prod_{t=2}^{\mathtt{T}} P(\underline{\underline{Y}}(t) \mid \underline{\underline{Y}}(t-1), ..., \underline{\underline{Y}}(1), \underline{X_1}, ..., \underline{X_{\mathtt{T}}})] P(\underline{\underline{Y}}(1) \mid \underline{X_1}, ..., \underline{X_{\mathtt{T}}}) \\
&= [\textstyle\prod_{t=2}^{\mathtt{T}} P(\underline{\underline{Y}}(t) \mid \underline{\underline{Y}}(t-1), ..., \underline{\underline{Y}}(1), \underline{X_t})] P(\underline{\underline{Y}}(1) \mid \underline{X_1}) & \text{A} \\
&= \textstyle\prod_{t=1}^{\mathtt{T}} P(\underline{\underline{Y}}(t) \mid \underline{X_t}) & \text{B} \\
&= \textstyle\prod_{t=1}^{\mathtt{T}} P(Y_{(1,1)}(t), ..., Y_{(\mathsf{S},\mathsf{S})}(t) \mid \underline{X_t}) \\
&= \textstyle\prod_{t=1}^{\mathtt{T}} \prod_{m=1}^{\mathsf{S}} \prod_{n=1}^{\mathsf{S}} P(Y_{(m,n)}(t) \mid \underline{X_t}) & \text{C}
\end{aligned}$$

2.5.2 Mathematical objective

With our simplifying assumptions we are interested in devising an efficient, deterministic, offline algorithm that given sensor input $\underline{\underline{Y}}$, will output $\underline{A} \triangleq (\underline{A_1}, ..., \underline{A_{\mathtt{T}}})$ such that

$$\underline{A}(\underline{\underline{Y}}) = arg \max_{\underline{a} \in \mathcal{A}(M_1^{(2D)}, \delta)} P[\underline{X} \in N_\delta(\underline{a}) \mid \underline{\underline{Y}}](\bigstar\bigstar)$$

So using Bayes Rule let us rewrite $P[\underline{X} \in N_\delta(\underline{a}) \mid \underline{\underline{Y}}]$:

$$\begin{aligned}
P(\underline{X} \in N_\delta(\underline{a}) \mid \underline{\underline{Y}}) &= \frac{\sum_{\underline{\gamma} \in N_\delta(\underline{a})} P(\underline{\gamma}, \underline{\underline{Y}})}{P(\underline{\underline{Y}})} \\
&= \frac{\sum_{\underline{\gamma} \in N_\delta(\underline{a})} P(\underline{\underline{Y}} | \underline{\gamma}) P(\underline{\gamma})}{\sum_{\underline{\gamma} \in \mathcal{P}(M_1^{(2D)})} P(\underline{\underline{Y}} | \underline{\gamma}) P(\underline{\gamma})} \\
&= \frac{\sum_{\underline{\gamma} \in N_\delta(\underline{a})} P(\underline{\underline{Y}} | \underline{\gamma})}{\sum_{\underline{\gamma} \in \mathcal{P}(M_1^{(2D)})} P(\underline{\underline{Y}} | \underline{\gamma})} \qquad \text{(motion model} \rightarrow \text{constant } P(\underline{\gamma}))
\end{aligned}$$

Therefore

$$\begin{aligned}
\underline{A}(\underline{\underline{Y}}) &= arg \max_{\underline{a} \in \mathcal{P}(M_1^{(2D)})} P[\underline{X} \in N_\delta(\underline{a}) \mid \underline{\underline{Y}}] \\
&= arg \max_{\underline{a} \in \mathcal{P}(M_1^{(2D)})} \textstyle\sum_{\underline{\gamma} \in N_\delta(\underline{a})} P(\underline{\underline{Y}} \mid \underline{\gamma}) \\
&= arg \max_{\underline{a} \in \mathcal{P}(M_1^{(2D)})} \textstyle\sum_{\underline{\gamma} \in N_\delta(\underline{a})} \prod_{t=1}^{\mathtt{T}} \prod_{m=1}^{\mathsf{S}} \prod_{n=1}^{\mathsf{S}} P(Y_{(m,n)}(t) \mid \underline{\gamma_t})
\end{aligned}$$

Now let us rewrite the term $\sum_{\underline{\gamma} \in N_\delta(\underline{a})} \prod_{t=1}^{\mathtt{T}} \prod_{m=1}^{\mathsf{S}} \prod_{n=1}^{\mathsf{S}} P(Y_{(m,n)}(t) \mid \underline{\gamma_t})$ more explicitly in terms of the noise level α and the individual measurements $Y_{t,\underline{s}}$.

So let us start by defining a function that takes a time t, set of measurements at time t, say $\underline{\underline{Y}}(\mathrm{t})$=

$$\begin{pmatrix} Y_{t,(1,1)} & Y_{t,(1,2)} & \dots & Y_{t,(1,\mathtt{T})} \\ Y_{t,(2,1)} & Y_{t,(2,2)} & \dots & Y_{t,(2,\mathtt{T})} \\ \dots & \dots & \dots & \dots \\ Y_{t,(\mathtt{T},1)} & Y_{t,(\mathtt{T},2)} & \dots & Y_{t,(\mathtt{T},\mathtt{T})} \end{pmatrix},$$

and a possible location for the object at time t, say $\underline{l_t} = (m_t, n_t) \in \mathcal{S}$, and outputs the number of local (in time) undamaged measurement bits at time t.

Therefore I define

$$N(t, \underline{\underline{Y}}(t), (m_t, n_t)) \triangleq \mathtt{S}^2 - \sum_{m=1}^{\mathtt{S}} \sum_{n=1}^{\mathtt{S}} Y_{t,(m,n)} + I(t, Y_{t,(m_t,n_t)}, (m_t, n_t)) - O(t, Y_{t,(m_t,n_t)}, (m_t, n_t))$$

where $I(t, Y_{t,(m_t,n_t)}, (m_t, n_t))$ is the indicator function that returns 1 if the measurement $Y_{t,(m_t,n_t)}$ is a one and 0 otherwise. Likewise $O(t, Y_{t,(m_t,n_t)}, (m_t, n_t))$ is the indicator function that returns 1 if the measurement $Y_{t,(m_t,n_t)}$ is a zero and 0 otherwise.

Therefore we can rewrite

$$\begin{aligned} N(t, \underline{\underline{Y}}(t), (m_t, n_t)) &= \mathtt{S}^2 - \textstyle\sum_{m=1}^{\mathtt{S}} \sum_{n=1}^{\mathtt{S}} Y_{t,(m,n)} + Y_{t,(m_t,n_t)} - (1 - Y_{t,(m_t,n_t)}) \\ &= \mathtt{S}^2 - \textstyle\sum_{m=1}^{\mathtt{S}} \sum_{n=1}^{\mathtt{S}} Y_{t,(m,n)} + 2Y_{t,(m_t,n_t)} - 1 \end{aligned}$$

So for example say $\mathtt{S} = 9$ and $\mathtt{T} = 5$. Then we have five 9×9 matrices of measurements which are 0s and 1s.

Now let us use the above formalism to simplify the optimization problem even further:

$$\begin{aligned}
\underline{A}(\underline{\underline{Y}}) &= arg\max{}_{\underline{a}\in\mathcal{P}(M_1^{(2D)})} P[\underline{X} \in N_\delta(\underline{a}) \mid \underline{\underline{Y}}] \\
&= arg\max{}_{\underline{a}\in\mathcal{P}(M_1^{(2D)})} \sum_{\underline{\gamma}\in N_\delta(\underline{a})} P(\underline{\underline{Y}} \mid \underline{\gamma}) \\
&= arg\max{}_{\underline{a}\in\mathcal{P}(M_1^{(2D)})} \sum_{\underline{\gamma}\in N_\delta(\underline{a})} \prod_{t=1}^{\mathtt{T}} \prod_{m=1}^{\mathtt{S}} \prod_{n=1}^{\mathtt{S}} P(Y_{(m,n)}(t) \mid \underline{\gamma_t}) \\
&= arg\max{}_{\underline{a}\in\mathcal{P}(M_1^{(2D)})} \sum_{\underline{\gamma}\in N_\delta(\underline{a})} \prod_{t=1}^{\mathtt{T}} (1-\alpha)^{N(t,\underline{\underline{Y}}(t),\underline{\gamma_t})} \alpha^{(S^2-N(t,\underline{\underline{Y}}(t),\underline{\gamma_t}))} \\
&= arg\max{}_{\underline{a}\in\mathcal{P}(M_1^{(2D)})} \sum_{\underline{\gamma}\in N_\delta(\underline{a})} (1-\alpha)^{\sum_{t=1}^{\mathtt{T}} N(t,\underline{\underline{Y}}(t),\underline{\gamma_t})} \alpha^{\sum_{t=1}^{\mathtt{T}}(S^2-N(t,\underline{\underline{Y}}(t),\underline{\gamma_t}))} \\
&= arg\max{}_{\underline{a}\in\mathcal{P}(M_1^{(2D)})} \sum_{\underline{\gamma}\in N_\delta(\underline{a})} (1-\alpha)^{2\sum_{t=1}^{\mathtt{T}} Y_{t,\underline{\gamma_t}}} \alpha^{-2\sum_{t=1}^{\mathtt{T}} Y_{t,\underline{\gamma_t}}} \\
&= arg\max{}_{\underline{a}\in\mathcal{P}(M_1^{(2D)})} \sum_{\underline{\gamma}\in N_\delta(\underline{a})} (\tfrac{1-\alpha}{\alpha})^{2\sum_{t=1}^{\mathtt{T}} Y_{t,\underline{\gamma_t}}}
\end{aligned}$$

This simplified version is friendlier to work with because the individual measurement values on the paths of interest in a given cylinder show up directly in the optimization problem.

For example, if the noise paramater that flips bits is $\alpha = \frac{1}{3}$, then $\underline{A}(\underline{\underline{Y}}) = arg\max{}_{\underline{a}\in\mathcal{P}(M_1^{(2D)})} \sum_{\underline{\gamma}\in N_\delta(\underline{a})} 4^{\sum_{t=1}^{\mathtt{T}} Y_{t,\underline{\gamma_t}}}$.

So notice that if the noise level $\alpha \leq .5$ then we roughly want to find the region with the most number of $1s$ but more precisely the region parametrized by $\underline{\text{A}}$ with many paths each with many $1s$. And if $.5 \leq \alpha$ we roughly want to find the region with the most number of $0s$ but more precisely the region with parametrized by $\underline{\text{A}}$ with many paths each with many $0s$. So we have generalized the solution to the problem for motion in two dimensions. However, the problem becomes more difficult because we now have to search for a one dimensional signal in three dimensions of noise as opposed to two in our lower dimensional version of the problem.

2.5.3 Algorithm for finding a MAP neighborhood

Now assuming without loss of generalization that our noise level $\alpha \leq .5$, our optimization problem is reduced to finding $\underline{A}$ such that:

$$\underline{A}(\underline{\underline{Y}}) = arg \max_{\underline{a}\in\mathcal{P}(M_1^{(2D)})} (\frac{1-\alpha}{\alpha})^{2\sum_{t=1}^{\mathtt{T}} Y_{t,\underline{a_t}}} = arg \max_{\underline{a}\in\mathcal{P}(M_1^{(2D)})} (\sum_{t=1}^{\mathtt{T}} Y_{t,\underline{a_t}}).$$

Let us now generalize our one dimensional definitions to two dimensional definitions because we will show that our one dimensional algorithm generalizes to solve the two dimensional problem.

First, let us define the set of possible paths that start at position $\underline{X_0}$ and end at position $\underline{s}$ at time t obeying the proposed motion model M_1 as

$$\mathcal{P}(t,\underline{s}) = \{\underline{\gamma} \in \mathcal{S}^t \mid \underline{\gamma_0} = \underline{X_0}; |\underline{\gamma_j} - \underline{\gamma_{j-1}}| \leq 1, 1 \leq j \leq t; \underline{\gamma_t} = \underline{s}\}.$$

Also let us define the set of possible locations at time $1 \leq t \leq \mathtt{T} - 1$ given the position at time $t+1$ is $\underline{l}$ as

$$\mathcal{L}(t+1,\underline{l}) = \{\underline{s} \in \mathcal{S} \mid |\underline{s} - \underline{X_0}| \leq t, |\underline{s} - \underline{l}| \leq 1\}.$$

Finally, let us define the set of possible locations at final time $\mathtt{T}$ as $\mathcal{L}_{\mathtt{T}} = \{\underline{s} \in \mathcal{S} \mid |\underline{s} - \underline{X_0}| \leq \mathtt{T}\}$ and let us define the set of possible locations at time t as $\mathcal{L}_t = \{\underline{s} \in \mathcal{S} \mid |\underline{s} - \underline{X_0}| \leq t\}$.

In order to keep track of the results of our optimization at each time step with the aim of reusing these results in the next step of the optimization (which will have been "separated in time"). Let $M_n(\underline{s}) : \mathcal{S} \to \mathcal{S}^n$ be a sequence of functions for $1 \leq n \leq \mathtt{T}$ such that

$$M_n(\underline{s}) = arg \max_{\underline{a} \in \mathcal{P}(n,\underline{s})} P[\underline{a} \mid \underline{\underline{Y}}] = arg \max_{\underline{a} \in \mathcal{P}(n,\underline{s})} (\sum_{t=1}^{n} Y_{t,\underline{a_t}})$$

And again it can be shown that

$$M_n(\underline{s}) = CONCAT[arg \max_{M_{n-1}(\underline{l}), \underline{l} \in \mathcal{L}(n,s)} f_{n-1}(M_{n-1}(\underline{l})), \underline{s}]$$

where $CONCAT$ is a function that takes two finite vectors and outputs a vector that is the concatentation of the two. The validity of the above step can be ascertained via a proof by induction. This leads us to our generalization of the one dimensional algorithm for the two dimensional problem:

for i=-1 to 1

```
begin
        for j=-1 to 1
        begin
        $M_1(\underline{X_0} - (i,j)) = (\underline{X_0} - (i,j));$
        end
end
for $t = 2$ to T
begin
        for $\underline{s} \in \mathcal{L}_t \subseteq \mathcal{S}$
        begin
                $M_t(\underline{s}) = CONCAT(arg\max_{M_{t-1}(\underline{l}), \underline{l} \in \mathcal{L}(t,\underline{s})} f_{t-1}(M_{t-1}(\underline{l})), \underline{s});$
        end
end
return $arg\max_{M_{\mathtt{T}}(\underline{l}), \underline{l} \in \mathcal{L}_{\mathtt{T}}} f_{\mathtt{T}}(M_{\mathtt{T}}(\underline{l}));$
```

Complexity calculations

Again we are assuming each dimension of S is much larger than T. If this is the case then the above algorithm looks at only a pyramid of points in the lattice (which are reachable from $\underline{X_0}$). Therefore, there are $O(\frac{\mathtt{T}(\mathtt{T}+1)}{2}^2) = O(\mathtt{T}^4)$ points. For any given point in the pyramid the algorithm does at most nine comparisons of objective function values for different paths that could have reached the given point. Therefore the algorithm takes $O(\mathtt{T}^4)$ time. Note that the running time increases with the dimension of the problem.

In terms of space complexity, at any given time step t we only need to keep track of $M_{t-1}(l) \forall l \in \mathcal{L}_{t-1}$. Since the cardinality of the set $\mathcal{L}_{t-1}$ is $(2(t-1)+1)^2 = O(t^2)$ and $M_{t-1}(l)$ is just a $(t-1)$-vector of numbers in $\mathcal{S}$, we have that the space requirements are $O(\mathtt{T}^3 log(\mathtt{S}))$. Note that the space required increased too. This may not be practical but it is interesting from a complexity point of view.

2.5.4 Calculating the probability in polynomial time

Now we have an algorithm that calculates the parameter, $\underline{A}$, of the MAP neighborhood for the special case when $\delta = 1$. But how do we calculate

$$P[\underline{X} \in N_\delta(\underline{A}) \mid \underline{\underline{Y}}] = \frac{\sum_{\underline{\gamma} \in N_\delta(\underline{A})} P(\underline{\underline{Y}} \mid \underline{\gamma})}{\sum_{\underline{\gamma} \in \mathcal{P}(M_1^{(2D)})} P(\underline{\underline{Y}} \mid \underline{\gamma})}$$

for the special case of $\delta = 1$ which initially appears to not be computable in polynomial time since the denominator is a sum over an exponential number of paths (in $\mathtt{T}$) regardless of the parameter, $\underline{A}$, or the neighborhood, $N_\delta(\underline{A})$.

Is the numerator computable in polynomial time? Yes because it is the probability of a single path. Indeed the numerator simplifies since $\delta = 1$ so that the δ-neighborhood with parameter $\underline{A}$ is precisely the vector $\underline{A}$ and therefore,

$$\begin{aligned}
\textstyle\sum_{\underline{\gamma} \in N_1(\underline{A})} P(\underline{\underline{Y}} \mid \underline{\gamma}) &= P(\underline{\underline{Y}} \mid \underline{A}) \\
&= \textstyle\prod_{t=1}^{\mathtt{T}} \prod_{m=1}^{\mathtt{S}} \prod_{n=1}^{\mathtt{S}} P(Y_{(m,n)}(t) \mid \underline{A_t}) \\
&= \textstyle\prod_{t=1}^{\mathtt{T}} (1-\alpha)^{N(t,\underline{Y}(t),\underline{A_t})} \alpha^{(S^2 - N(t,\underline{Y}(t),\underline{A_t}))} \\
&= (1-\alpha)^{\sum_{t=1}^{\mathtt{T}} N(t,\underline{Y}(t),\underline{A_t})} \alpha^{\sum_{t=1}^{\mathtt{T}} (S^2 - N(t,\underline{Y}(t),\underline{A_t}))} \\
&= (1-\alpha)^{T(S^2-1) - sum(\underline{\underline{Y}}) + 2\sum_{t=1}^{\mathtt{T}} Y_{t,\underline{A_t}}} \alpha^{T + sum(\underline{\underline{Y}}) - 2\sum_{t=1}^{\mathtt{T}} Y_{t,\underline{A_t}}} \\
&= (1-\alpha)^{TS^2} \left(\tfrac{1-\alpha}{\alpha}\right)^{-T - sum(\underline{\underline{Y}})} \left(\tfrac{1-\alpha}{\alpha}\right)^{2\sum_{t=1}^{\mathtt{T}} Y_{t,\underline{A_t}}}
\end{aligned}$$

where $sum(\underline{\underline{Y}}) \triangleq \sum_{t=1}^{\mathtt{T}} \sum_{m=1}^{\mathtt{S}} \sum_{n=1}^{\mathtt{S}} Y_{t,(m,n)}$. So indeed the numerator is computable in polynomial time.

However, is the denominator computable in polynomial time? How can an algorithm efficiently count and determine $\sum_{\underline{X} \in \mathcal{P}(M_1)} P(\underline{\underline{Y}} \mid \underline{X})$ without going through all the possible paths $\underline{X} \in \mathcal{P}(M_1)$?

The equivalence class approach naturally generalizes to solve this higher dimensional problem. Define

$$I(\underline{x}) \triangleq \sum_{t=1}^{\mathtt{T}} Y_{t,\underline{x_t}}$$

so that $I(\underline{x})$ is the number of 1s on the path $\underline{x}$.

Then we can define the equivalence relation:

$$\underline{x}^{(1)} \sim \underline{x}^{(2)} \Leftrightarrow I(\underline{x}^{(1)}) = I(\underline{x}^{(2)}).$$

Intuitively the number of 1s on the path $\underline{x}$ is related to the number of undamaged measurement bits, $\sum_{t=1}^{\mathrm{T}} N(t, \underline{Y}(t), \underline{x_t})$, if $\underline{x}$ was indeed the actual path. Namely,

$$\sum_{t=1}^{\mathrm{T}} N(t, \underline{Y}(t), \underline{x_t}) = T(S^2 - 1) - sum(\underline{\underline{Y}}) + 2\sum_{t=1}^{\mathrm{T}} Y_{t,\underline{x_t}}.$$

Therefore for two dimensional motion, the number of 1s is related to the number of undamaged measurement bits via the addition of a constant term that is independent of the assumed path, $\underline{x}$. Now we can index the equivalence classes uniquely in the following way: take a path $\underline{x}$ in any equivalence class, then the equivalence class $[\underline{x}]$ will have an integer index $k = \sum_{t=1}^{\mathrm{T}} Y_{t,\underline{x_t}} \in \{1, 2, 3, ..., \mathrm{T}\}$. And I define the cardinality of the equivalence class with index k, $[k]$, to be $C(k)$.

So my point is that given $\underline{\underline{Y}}$ and $I(\underline{x})$, we can rewrite $\sum_{\underline{x} \in \mathcal{P}(M_1^{(2D)})} P(\underline{\underline{Y}} \mid \underline{x})$ as

$$\sum_{\underline{x} \in \mathcal{P}(M_1^{(2D)})} P(\underline{\underline{Y}} \mid \underline{x}) = \sum_{k=0}^{\mathrm{T}} C(k)(1-\alpha)^{TS^2}(\frac{1-\alpha}{\alpha})^{-T-sum(\underline{\underline{Y}})}(\frac{1-\alpha}{\alpha})^{2k}$$

Is there an efficient way to calculate $C(k)$ for all $k \in \{0, 1, 2, 3, ..., \mathrm{T}\}$ using the set of all paths $\underline{X} \in \mathcal{P}(M_1^{(2D)})$? Again that seems difficult due to the fact that there are an exponential number of possible paths in $\mathcal{P}(M_1^{(2D)})$. Nonetheless, the following algorithm is inspired from the algorithm for calculating Pascal's triangle. First let me recursively define

$$C^t(\underline{s}, k) \triangleq \sum_{l=-1}^{1} \sum_{m=-1}^{1} C^{(t-1)}(\underline{s} + (l, m), k - Y_{t,\underline{s}})$$

$$\text{where } C^t(\underline{s}, k) = 0, \forall s \in \mathcal{S}, \forall k \in \{0, 1, ..., \mathrm{T}\} \text{ if } s_x \geq ((\underline{X_0})_x + t)$$

$$\text{or } s_x \leq ((\underline{X_0})_x - t) \text{ or } s_y \geq ((\underline{X_0})_y + t) \text{ or } s_y \leq ((\underline{X_0})_y - t)$$

$$\text{and } C^0(\underline{s}, k) = \left\{ \begin{array}{lll} 1 & \text{if} & k = 0, \underline{s} = \underline{X_0} \\ 0 & \text{otherwise} & \end{array} \right\}$$

$$\text{such that } C(k) = \sum_{\underline{s} \in \mathcal{S}} C^{\mathtt{T}}(\underline{s}, k).$$

Note that $C^t(\underline{s}, k)$ is the number of paths through time t that go through pixel $\underline{s}$ and have colored k pixels as 1s along their paths.

Complexity Calculations

We can do the counting iteratively at each time step so that calculating $C^{\mathtt{T}}(\underline{s}, k) \forall \underline{s} \in \mathcal{S}, \forall k \in \{0, 1, ..., \mathtt{T}\}$ can be done in time $O(\mathtt{T}^4)$. This is possible because at each time step t we update order t^3 function values corresponding to the set $\{C^t(\underline{s}, k) : s_x \in \{(\underline{X_0})_x - t, (\underline{X_0})_x - t + 1, ..., (\underline{X_0})_x + t\}; s_y \in \{(\underline{X_0})_y - t, (\underline{X_0})_y - t + 1, ..., (\underline{X_0})_y + t\}; k \in \{0, 1, 2, ..., t\}\}$.

The space requirement of this algorithm is $O(\mathtt{T}^3)$. This can be seen because at each time t the algorithm only needs to keep track of order t^2 pixels (the ones that may have paths going through them) and for each of those pixels indexed by (t, s) we need to keep track of the values $\{C^t(s, k) : k \in \{0, 1, 2, ..., t\}$ which sum to at most 9^t (which requires at most t bits).

So indeed, $C(k) \forall k \in \{0, 1, ..., \mathtt{T}\}$ can be calculated efficiently in terms of time. We therefore have an efficient algorithm that calculates $\sum_{\underline{x} \in \mathcal{P}(M_1^{(2D)})} P(\underline{\underline{Y}} \mid \underline{x})$ without enumerating the probabilities associated with each of the possible paths $\underline{x}$ in the large set $\mathcal{P}(M_1^{(2D)})$.

Nonetheless as our calculations illustrate, the complexity of our algorithm increases with the dimension of movement that is allowed in the motion model.

2.6 Summary

In this chapter, we have discovered that for the prior "motion probability model" M_1 (and its two dimensional equivalent, $M_1^{(2D)}$) and for a neighborhood size $\delta = 1$ instead of a brute force search over an exponential number of paths we can devise an algorithm that finds the MAP neighborhood in polynomial time. After conversing with colleagues and reading the relevant articles, I found out that this algorithm that I discovered is actually the Viterbi Algorithm or a special case of dynamic programming used to solve several hidden markov modelling problems [2]. Our problem can also be cast as a hidden markov model because the state (the actual path) is hidden from us and instead we have a matrix of measurements from which we are asked to infer the best neighborhood for the state variable.

Furthermore, I discovered a polynomial time algorithm to calculate the denomiator in the probability calculation. At first glance this is a surprise because the sum is a sum over an exponential number of objects, but through equivalence classes a polynomial-time algorithm manifested itself.

In the next chapter, I will change the motion model to M_Δ and seek a MAP neighborhood with size δ allowed to be values other than 1, but with the restriction that $\delta \leq \Delta$. Can we can achieve the same efficiency in finding the optimal MAP neighborhood and calculating its probability for those parameters of the problem?

Chapter 3

Movement Range $\geq$ Neighborhood Size $\geq$ 1

3.1 Introduction

In this chapter, we look at another operating regime for our one dimensional tracking problem. Namely, we will analyze the complexity of our optimization problem if the prior "motion probability model" we utilize is restricted to have a motion range, Δ, that is greater than or equal to the neighborhood size, δ. Therefore, for a motion model M_Δ and a neighborhood size δ where $\Delta \geq \delta \geq 1$, we seek the MAP neighborhood, $N_\delta(\underline{A})$, parameterized by $\underline{A}(\underline{\underline{Y}}) \in \mathcal{A}(M_\Delta, \delta)$ where

$$\underline{A}(\underline{\underline{Y}}) = arg \max_{\underline{a} \in \mathcal{A}(M_\Delta, \delta)} P[\underline{X} \in N_\delta(\underline{a}) \mid \underline{\underline{Y}}] = arg \max_{\underline{a} \in \mathcal{A}(M_\Delta, \delta)} \sum_{\underline{\gamma} \in N_\delta(\underline{a})} \prod_{t=1}^{\mathtt{T}} \prod_{s=1}^{\mathtt{S}} P(Y_s(t) \mid \gamma_t).$$

We will find an efficient algorithm that finds $\underline{A}(\underline{\underline{Y}})$ and analyze the algorithm's complexity. It will turn out that the algorithm will be similar to the algorithm we found for the operating regime where $\Delta = \delta = 1$. Then we will discuss how to calculate

$$P[\underline{X} \in N_\delta(\underline{A}) \mid \underline{\underline{Y}}] = \frac{\sum_{\underline{\gamma} \in N_\delta(\underline{A})} P(\underline{\underline{Y}} \mid \underline{\gamma})}{\sum_{\underline{\gamma} \in \mathcal{P}(M_\Delta)} P(\underline{\underline{Y}} \mid \underline{\gamma})}$$

which, again, at first glance looks like it is not computable in polynomial time since the denominator is a sum over an exponential number of paths (in T) and in addition for $\delta \geq 2$ the numerator is a sum over an exponential number of paths.

Then we will end this chapter with some insights regarding this operating regime for the tracking problem.

In this chapter, when we fix δ to be greater than one we gain what we had been seeking in a formulation: a formulation that sought out optimal regions as opposed to optimal paths. At the same time we will allow our prior "motion probability model" to have a wider range of motion modelling the idea that our object can travel to more pixels at each point in time. With this increased motion range and neighborhood size will the problem become harder to solve?

3.2 Algorithm for finding a MAP neighborhood

3.2.1 Some mathematical simplifications

In the previous chapter we had a sum of products that we had to optimize which in general is a difficult problem depending on the domain that the indices are summing and multiplying over and the connection between the domains. However, because $\delta = 1$ our "external" summand over the paths contained in a δ-neighborhood disappeared. What mathematical simplifications can we do for this regime of the problem, where the external "summand" need not disappear?

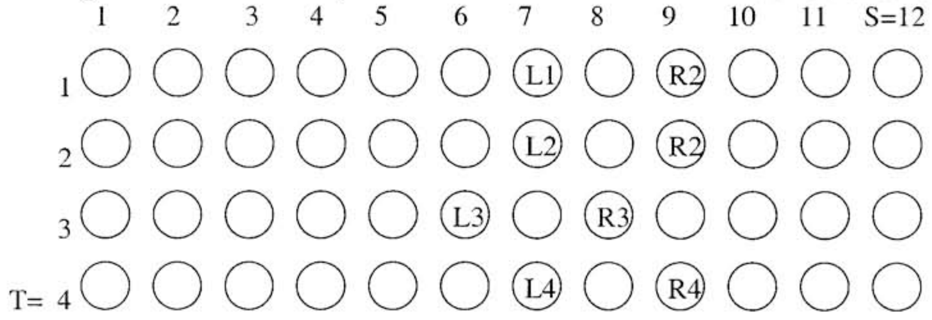
Recalling our previously defined notion of number of local (in time) undamaged measurement bits at time t, $N(t, \underline{Y}(t), l_t)$, we can make the following simplifications:

$$\begin{array}{rcl}
\underline{A}(\underline{\underline{Y}}) & = & arg\max_{\underline{a} \in \mathcal{A}(M,\delta)} P[\underline{X} \in N_\delta(\underline{a}) \mid \underline{\underline{Y}}] \\
& = & arg\max_{\underline{a} \in \mathcal{A}(M,\delta)} \sum_{\underline{\gamma} \in N_\delta(\underline{a})} P(\underline{\underline{Y}} \mid \underline{\gamma}) \\
& = & arg\max_{\underline{a} \in \mathcal{A}(M,\delta)} \sum_{\underline{\gamma} \in N_\delta(\underline{a})} \prod_{t=1}^{\mathtt{T}} \prod_{s=1}^{\mathtt{S}} P(Y_s(t) \mid \gamma_t) \\
& = & arg\max_{\underline{a} \in \mathcal{A}(M,\delta)} \sum_{\underline{\gamma} \in N_\delta(\underline{a})} \prod_{t=1}^{\mathtt{T}} (1-\alpha)^{N(t,\underline{Y}(t),\gamma_t)} \alpha^{(S-N(t,\underline{Y}(t),\gamma_t))} \\
& = & arg\max_{\underline{a} \in \mathcal{A}(M,\delta)} \prod_{t=1}^{\mathtt{T}} \sum_{k=0}^{\delta-1} (1-\alpha)^{N(t,\underline{Y}(t),a_t+k)} \alpha^{(S-N(t,\underline{Y}(t),a_t+k))} (\blacktriangleleft) \\
& = & arg\max_{\underline{a} \in \mathcal{A}(M,\delta)} \sum_{t=1}^{\mathtt{T}} log(\sum_{k=0}^{\delta-1} (1-\alpha)^{N(t,\underline{Y}(t),a_t+k)} \alpha^{(S-N(t,\underline{Y}(t),a_t+k))})
\end{array}$$

Why does the simplification happen?

The step in the above simplification labelled with (◀) was feasible because the motion model allowed enough movement in comparison to the size of the δ-neighborhood. We can make this more clear with an example. Let us fix $\mathtt{T} = 4$, $\mathtt{S} = 12$, $X_0 = 8$, and $\underline{a} = (7, 7, 6, 7)$. Then we have a region of the lattice bounded by the left and right boundaries, $\underline{L}$ and $\underline{R}$ respectively as we see in figure 3-1. Note that we have picked a

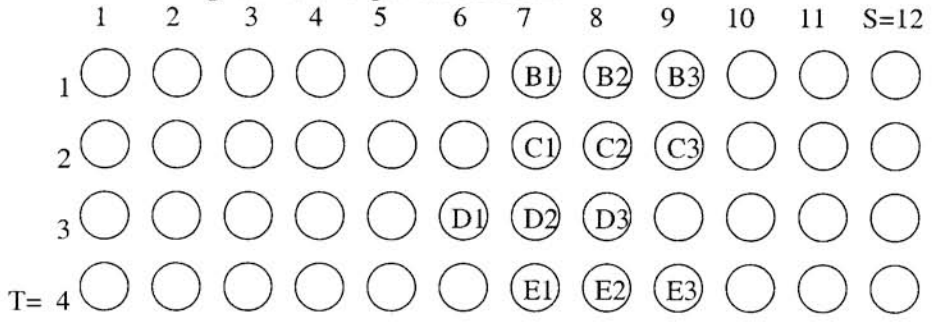
Figure 3-1: An example of the lattice and the boundaries that define a region.



$\delta = 3$ size neighborhood. So let us assume a motion model M_Δ such that $\Delta \geq \delta = 3$. In particular let us assume $\Delta = 3$.

Now we can assign every pixel (t, s) in that neighborhood a value that only depends on the position of the pixel and its local measurement value, $Y_{(t,s)}$. Let us assign pixel (t, s) with value $(1-\alpha)^{N(t,\underline{Y}(t),s)}\alpha^{(S-N(t,\underline{Y}(t),s))}$. Using simpler notation, let us write the values for each of the pixels in the δ-neighborhood on top of the pixels in figure 3-2.

Figure 3-2: An example of the lattice and the boundaries that define a region and the values assigned to the pixels therein.



Now we see that $\sum_{\underline{\gamma} \in N_\delta(\underline{a})} \prod_{t=1}^{\mathtt{T}} (1-\alpha)^{N(t,\underline{Y}(t),\gamma_t)}\alpha^{(S-N(t,\underline{Y}(t),\gamma_t))}$ is just the sum over the set $\{B_i * C_j * D_k * E_l : 1 \leq i, j, k, l \leq 3\}$ which can also be rewritten as

$$(B_1 + B_2 + B_3) * (C_1 + C_2 + C_3) * (D_1 + D_2 + D_3) * (E_1 + E_2 + E_3)$$

which is precisely $\prod_{t=1}^{\mathtt{T}} \sum_{k=0}^{\delta-1} (1-\alpha)^{N(t,\underline{Y}(t),a_t+k)} \alpha^{(S-N(t,\underline{Y}(t),a_t+k))} (\blacktriangleleft)$. This sketch can be expanded to a proof, but this sketch is meant to give intuition for the step labelled with a $(\blacktriangleleft)$ symbol.

3.2.2 Defining the algorithm

In this section, we will give various definitions with the aim of devising an algorithm where we can separate the optimization in time as we accomplished in chapter two.

First, let us define the set of possible parameters for δ-neighborhoods that start at position $X_0 - \lfloor \frac{\delta}{2} \rfloor$ and end at position s at time t obeying the "continuity" constraint imposed on δ-neighborhoods:

$$\mathcal{A}(M_\Delta, \delta, t, s) = \{\underline{\gamma} \in \mathcal{S}^t \mid |\gamma_j - \gamma_{j-1}| \leq 1, 1 \leq j \leq t; \gamma_0 = X_0 - \lfloor \frac{\delta}{2} \rfloor; \gamma_t = s\}.$$

Also let us define the set of possible values for the t^{th} component of the parameter vector, $1 \leq t \leq \mathtt{T} - 1$, given the value of the $(t+1)^{th}$ component is l as

$$\mathcal{L}(t+1, l) = \{s \in \mathcal{S} \mid |s - \gamma_0| \leq t, |s - l| \leq 1\}.$$

Finally, let us define the set of possible locations at the final time $\mathtt{T}$ for the parameter vector $\underline{a}$, parameterizing the neighborhood $N_\delta(\underline{a})$, as $\mathcal{L}_{\mathtt{T}} = \{s \in \mathcal{S} \mid |s - \gamma_0| \leq \mathtt{T}\}$ and let us define the set of possible locations for the parameter vector at time t as $\mathcal{L}_t = \{s \in \mathcal{S} \mid |s - \gamma_0| \leq t\}$.

We now need to define some functions to keep track of the results of our optimization at each time step with the aim of reusing these results in the next step of the optimization (which will have been "separated in time"). Let $M_n(s) : \mathcal{S} \to \mathcal{S}^n$ be a sequence of functions for $1 \leq n \leq \mathtt{T}$ such that

$$M_n(s) = arg \max_{\underline{a} \in \mathcal{A}(M_\Delta, \delta, n, s)} P[\underline{X} \in N_\delta(\underline{a}) \mid \underline{\underline{Y}}] = arg \max_{\underline{a} \in \mathcal{A}(M_\Delta, \delta, n, s)} (\sum_{t=1}^{n} g(t, a_t))$$

where $g(t, a_t) \triangleq log(\sum_{k=0}^{\delta-1} (1-\alpha)^{N(t, \underline{Y}(t), a_t+k)} \alpha^{(S-N(t,\underline{Y}(t),a_t+k))})$

Therefore, $M_n(s)$ represents a parameter that maximizes the objective function $f_n(\underline{a}) \triangleq (\sum_{t=1}^{n} g(t, a_t))$ over all $\underline{a} \in \mathcal{A}(M_\Delta, \delta, n, s)$. We are interested in searching amongst the parameters in the set $\{\underline{\gamma} : \underline{\gamma} = M_{\mathtt{T}}(s), s \in \mathcal{L}_{\mathtt{T}}\}$ and outputting a path that maximizes $f(\underline{a}) = (\sum_{t=1}^{\mathtt{T}} g(t, a_t))$. And in order to devise an efficient algorithm to calculate $M_{\mathtt{T}}(s) \forall s \in \mathcal{L}_{\mathtt{T}}$, we will use use $M_t(s) \forall s \in \mathcal{L}_t$ in order to determine $M_{t+1}(s) \forall s \in \mathcal{L}_{t+1}$ for all times $t \in \mathcal{T}$. That is the idea behind separating the optimization problem in time. Now why and how can this be done?

It can be accomplished because of the mathematical structure of the objective function $f_n(\underline{a}) = (\sum_{t=1}^{n} g(t, a_t)$ and the fact that the measurements are non-negative. More formally, it can be shown that

$$M_n(s) = CONCAT[arg \max_{M_{n-1}(l), l \in \mathcal{L}(n,s)} f_{n-1}(M_{n-1}(l)), s]$$

where $CONCAT$ is a function that takes two finite vectors and outputs a vector that is the concatenation of the two. The validity of the above step can be ascertained via a proof by induction.

So the pseudo code for the algorithm that will solve the optimization problem can be written down concisely:

```
$M_1(X_0 - 1) = (X_0 - \lfloor \frac{\delta}{2} \rfloor) - 1;$
$M_1(X_0) = X_0 - \lfloor \frac{\delta}{2} \rfloor;$
$M_1(X_0 + 1) = (X_0 - \lfloor \frac{\delta}{2} \rfloor) + 1;$
for $t = 2$ to $\mathtt{T}$
begin
    for $s \in \mathcal{L}_t \subseteq \mathcal{S}$
```

```
        begin
            $M_t(s) = CONCAT(arg\max_{M_{t-1}(l), l\in\mathcal{L}(t,s)} f_{t-1}(M_{t-1}(l)), s);$
        end
    end
    return $arg\max_{M_{\mathtt{T}}(l), l\in\mathcal{L}_{\mathtt{T}}} f_{\mathtt{T}}(M_{\mathtt{T}}(l));$
```

The above algorithm finds the optimizing route through every node at each point in time and works its way down in time, reusing the work from the previous time step. In that sense we have been able to "separate the problem in time".

3.2.3 Complexity calculations

Again we are assuming $\mathtt{S}$ is much larger than $\mathtt{T}$. If this is the case then the above algorithm looks at only a pyramid of points in the lattice (which are reachable from $X_0 - \lfloor\frac{\delta}{2}\rfloor$). Therefore there are $O(\frac{\mathtt{T}(\mathtt{T}+1)}{2}) = O(\mathtt{T}^2)$ points. For any given point in the pyramid the algorithm does at most three comparisons of objective function values for different parameter trajectories that could have reached the given point. Therefore the algorithm takes $O(\mathtt{T}^2)$ time.

In terms of space complexity, at any given time step t we only need to keep track of $M_{t-1}(l)\forall l \in \mathcal{L}_{t-1}$. Since the cardinality of the set $\mathcal{L}_{t-1}$ is $2(t-1)+1 = O(t)$ and $M_{t-1}(l)$ is just a $(t-1)$-vector of numbers in $\mathcal{S}$, we have that the space requirements are $O(\mathtt{T}^2 log(\mathtt{S}))$.

3.3 Calculating the probability in polynomial time

Now we have an algorithm that calculates the parameter, $\underline{A}$, of the MAP neighborhood for the special case when $\Delta \geq \delta \geq 1$. But how do we calculate

$$P[\underline{X} \in N_\delta(\underline{A}) \mid \underline{\underline{Y}}] = \frac{\sum_{\underline{\gamma}\in N_\delta(\underline{A})} P(\underline{\underline{Y}} \mid \underline{\gamma})}{\sum_{\underline{\gamma}\in\mathcal{P}(M_\Delta)} P(\underline{\underline{Y}} \mid \underline{\gamma})}$$

for that special case? Well from chapter two, we know that we can calculate these at first glance "difficult" enumeration problems with equivalence classes. So let us reuse our results with modifications to account for the fact that the paths now are contained in neighborhoods and the paths have a general M_Δ prior "motion probability model".

3.3.1 Calculating the numerator

The numerator is now a sum over an exponential number of paths in $N_\delta(\underline{A})$. Can that be calculated in polynomial time? The answer turns out to be yes if we utilize our equivalence classes in a different fashion.

How can an algorithm efficiently count and determine $\sum_{\underline{\gamma}\in N_\delta(\underline{A})} P(\underline{\underline{Y}} \mid \underline{\gamma})$ without going through all the possible paths $\underline{\gamma} \in N_\delta(\underline{A})$.

If we can change the sum over paths into a sum over equivalence classes for paths where the number of equivalence classes is small (namely polynomial in $\mathtt{T}$ and $\mathtt{S}$) and each path, $\underline{x}$, in an equivalence class has the same conditional probability, $P(\underline{\underline{Y}} \mid \underline{x})$, (so that the sum over the equivalence classes is easy to calculate) then will find an answer as we did in chapter two. But what equivalence relation (property) do we need?

We use the property discussed in chapter two. Namely, define

$$I(\underline{x}) \triangleq \sum_{t=1}^{\mathtt{T}} Y_{t,x_t}$$

so that $I(\underline{x})$ is the number of 1s on the path $\underline{x}$.

Therefore, we define the equivalence relation:

$$\underline{x}^{(1)} \sim \underline{x}^{(2)} \Leftrightarrow I(\underline{x}^{(1)}) = I(\underline{x}^{(2)}).$$

Now we can index the equivalence classes uniquely in the following way: take a path $\underline{x}$ in any equivalence class, then the equivalence class $[\underline{x}]$ will have an integer index $k = \sum_{t=1}^{\mathtt{T}} Y_{t,x_t} \in \{1, 2, 3, ..., \mathtt{T}\}$. And I define the cardinality of the equivalence class with index k, $[k]$, to be $C(k)$.

So my point is that given $\underline{\underline{Y}}$ and $I(\underline{x})$, we can rewrite $\sum_{\underline{x}\in N_\delta(\underline{A})} P(\underline{\underline{Y}} \mid \underline{x})$ as

$$\sum_{\underline{x}\in N_\delta(\underline{A})} P(\underline{\underline{Y}} \mid \underline{x}) = \sum_{k=0}^{\mathtt{T}} C(k)(1-\alpha)^{TS}(\frac{1-\alpha}{\alpha})^{-T-sum(\underline{\underline{Y}})}(\frac{1-\alpha}{\alpha})^{2k}$$

We do require that $C(k)$ only count paths that lie in the δ-neighborhood. And this will be shown as we demonstrate how $C(k)$ can be calculated efficiently for all $k \in \{0,1,2,3,...,\mathtt{T}\}$, hence allowing us to calculate $\sum_{\underline{x}\in N_\delta(\underline{A})} P(\underline{\underline{Y}} \mid \underline{x})$ efficiently.

First let me recursively define

$$C^t(s,k) \triangleq \sum_{l=-\Delta}^{\Delta} C^{(t-1)}(s+l, k-Y_{t,s})$$

$$\text{where } C^t(s,k) = 0, \forall k \in \{0,1,...,\mathtt{T}\} \text{ if } s \geq (X_0 + t\Delta) \text{ or } s \leq (X_0 - t\Delta)$$

$$\text{or } s \leq (A_t - 1) \text{ or } s \geq (A_t + \delta)$$

$$\text{and } C^0(s,k) = \left\{ \begin{array}{lll} 1 & \text{if} & k=0, s=X_0 \\ 0 & \text{otherwise} & \end{array} \right\}$$

$$\text{such that } C(k) = \sum_{s\in\mathcal{S}} C^{\mathtt{T}}(s,k).$$

Note that $C^t(s,k)$ is the number of paths through time t that go through pixel s and have colored k pixels as $1s$ along their paths.

3.3.2 Calculating the denominator

How can an algorithm efficiently count and determine $\sum_{\underline{X}\in\mathcal{P}(M_\Delta)} P(\underline{\underline{Y}} \mid \underline{X})$ without going through all the possible paths $\underline{X} \in \mathcal{P}(M_\Delta)$?

Using an equivalence relation resembling the equivalence relation used in the previous section, the following recursively defined value

$$D^t(s,k) \triangleq \sum_{l=-\Delta}^{\Delta} D^{(t-1)}(s+l, k-Y_{t,s})$$

$$\text{where } D^t(s,k) = 0, \forall k \in \{0,1,...,\mathtt{T}\} \text{ if } s \geq (X_0 + t\Delta) \text{ or } s \leq (X_0 - t\Delta)$$

$$\text{and } D^0(s,k) = \left\{ \begin{array}{lll} 1 & \text{if} & k=0, s=X_0 \\ 0 & \text{otherwise} & \end{array} \right\}$$

$$\text{such that } D(k) \triangleq \sum_{s \in \mathcal{S}} D^{\mathtt{T}}(s,k).$$

then we will have transformed $\sum_{\underline{x} \in \mathcal{P}(M_\Delta)} P(\underline{\underline{Y}} \mid \underline{x})$ into

$$\sum_{k=0}^{\mathtt{T}} D(k)(1-\alpha)^{TS}(\frac{1-\alpha}{\alpha})^{-T-sum(\underline{\underline{Y}})}(\frac{1-\alpha}{\alpha})^{2k}$$

hence calculating the denominator.

Complexity Calculations

We can do the counting iteratively at each time step so that calculating $C^{\mathtt{T}}(s,k) \forall s \in \mathcal{S}, \forall k \in \{0, 1, ..., \mathtt{T}\}$ can be done in time $O(\mathtt{T}^3 \Delta^2)$. This is possible because at each time step t we update order $t^2 \Delta$ function values corresponding to the set $\{C^t(s,k) : s \in \{X_0 - t\Delta, X_0 - t\Delta + 1, ..., X_0 + t\Delta\}; k \in \{0, 1, 2, ..., t\}$ by looking at $O(\Delta)$ tables at the previous time step in order to define $C^t(s,k)$.

The space requirement of this algorithm is $O(\mathtt{T}^2 \Delta log(\Delta))$. This can be seen because at each time t the algorithm only needs to keep track of order $t\Delta$ pixels (the ones that may have paths going through them) and for each of those pixels indexed by (t,s) we need to keep track of the values $\{C^t(s,k) : k \in \{0, 1, 2, ..., t\}$ which sum to at most $O(\Delta^t)$ (which requires at most $tlog(\Delta)$ bits).

So indeed, $C(k) \forall k \in \{0, 1, ..., \mathtt{T}\}$ can be calculated efficiently. And we therefore have an efficient algorithm that calculates $P[\underline{X} \in N_\delta(\underline{A}) \mid \underline{\underline{Y}}]$ for a given $\underline{A}$.

3.4 A Generalization to motion in higher dimensions

The generalization for higher dimensional motion follows the generalization that was done in chapter two very closely. Therefore, it will not be repeated here.

3.5 Summary

Allowing prior "motion probability models", M_Δ, in our formulation, where $\Delta \geq \delta \geq 1$ did not alter our ability to come up with an efficient algorithm to solve the optimization problem. In a sense we have seen that if we make the motion model more flexible so that the object can move at least the size of the neighborhood in pixels in either direction, then we can still solve this problem of finding the "best" neighborhood. In chapter two we saw that if the object moves 1 step in any direction at each point in time then we can find the "best" neighborhood that is 1 pixel wide at each time instant. Therefore chapter two was a special case of chapter three. Likewise we found that if the maximum allowable movement for an object at each time instant is greater than or equal to the number of pixels in the neighborhood at each time instant then the problem can be efficiently solved in a similar manner. This accounts for the most common cases in which this formulation would be useful. However for theoretical interest what if the motion model was not as flexible and instead the motion at each time step was confined to step sizes that were smaller than the cardinality of the number of pixels in the neighborhood at a fixed point in time? Would the problem still be "separable in time" and efficiently solvable? We shall investigate this problem regime in the next chapter.

Chapter 4

Neighborhood Size $>$ Movement Range $\geq$ 1

4.1 Introduction

In this chapter, we look at another operating regime for our one dimensional tracking problem. Namely, we will analyze the complexity of our optimization problem if the prior "motion probability model" we utilize is restricted to have a motion range, Δ, that is strictly smaller than the neighborhood size, δ. Therefore, for a motion model M_Δ and a neighborhood size δ where $\delta > \Delta \geq 1$, we seek the MAP neighborhood, $N_\delta(\underline{A})$, parameterized by $\underline{A}(\underline{\underline{Y}}) \in \mathcal{A}(M_\Delta, \delta)$ where

$$\underline{A}(\underline{\underline{Y}}) = arg \max_{\underline{a} \in \mathcal{A}(M_\Delta,\delta)} P[\underline{X} \in N_\delta(\underline{a}) \mid \underline{\underline{Y}}] = arg \max_{\underline{a} \in \mathcal{A}(M_\Delta,\delta)} \sum_{\underline{\gamma} \in N_\delta(\underline{a})} \prod_{t=1}^{\mathtt{T}} \prod_{s=1}^{\mathtt{S}} P(Y_s(t) \mid \gamma_t).$$

Now we would hope that the algorithm in chapter two and three could be further generalized to solve the problem formulated with our less flexible [1] motion model. But how would this work? Let us look at the mathematics and see if this is possible.

[1]relative to our neighborhood size

Let us start with our usual simplification steps:

$$\begin{aligned}
\underline{A}(\underline{\underline{Y}}) &= arg\max\nolimits_{\underline{a}\in\mathcal{A}(M_\Delta,\delta)} P[\underline{X} \in N_\delta(\underline{a}) \mid \underline{\underline{Y}}] \\
&= arg\max\nolimits_{\underline{a}\in\mathcal{A}(M_\Delta,\delta)} \sum\nolimits_{\underline{\gamma}\in N_\delta(\underline{a})} P(\underline{\underline{Y}} \mid \underline{\gamma}) \\
&= arg\max\nolimits_{\underline{a}\in\mathcal{A}(M_\Delta,\delta)} \sum\nolimits_{\underline{\gamma}\in N_\delta(\underline{a})} \prod\nolimits_{t=1}^{\mathtt{T}} \prod\nolimits_{s=1}^{\mathtt{S}} P(Y_s(t) \mid \gamma_t) \\
&= arg\max\nolimits_{\underline{a}\in\mathcal{A}(M_\Delta,\delta)} \sum\nolimits_{\underline{\gamma}\in N_\delta(\underline{a})} \prod\nolimits_{t=1}^{\mathtt{T}} (1-\alpha)^{N(t,\underline{Y}(t),\gamma_t)} \alpha^{(S-N(t,\underline{Y}(t),\gamma_t))} \\
&\neq arg\max\nolimits_{\underline{a}\in\mathcal{A}(M_\Delta,\delta)} \prod\nolimits_{t=1}^{\mathtt{T}} \sum\nolimits_{k=0}^{\delta-1} (1-\alpha)^{N(t,\underline{Y}(t),a_t-k)} \alpha^{(S-N(t,\underline{Y}(t),a_t-k))} \\
&= arg\max\nolimits_{\underline{a}\in\mathcal{A}(M_\Delta,\delta)} \sum\nolimits_{t=1}^{\mathtt{T}} log\big(\sum\nolimits_{k=0}^{\delta-1} (1-\alpha)^{N(t,\underline{Y}(t),a_t-k)} \alpha^{(S-N(t,\underline{Y}(t),a_t-k))}\big)
\end{aligned}$$

Because of the inequality we can not generalize our previous algorithm for the flexible motion model. Intuitively, at time t, our algorithm kept track of the states for each "optimal" tube going through each pixel at time t. These "optimal" tubes were found by selectively extending "optimal" tubes from the previous time $t-1$ based on their states. Each pixel from the previous time $t-1$ only had one path to offer as a candidate path for the reachable pixels at time t. The state space we used does not generalize to the long range time dependencies inherent to this problem regime. Therefore an algorithm similar to the algorithms of chapter two and chapter three will not solve the problem in this operating regime. Thus, if we seek an optimization algorithm, it can not be accomplished by separating the problem in time and using only the past to make a decision at the current point in time. In fact this long range dependance property hints at the use of matrices and graphs (as constraints) to reformulate the optimization problem.

4.2 Reformulation as a Graph Problem

So let us reformulate the problem with $\Delta = 1$, and $\delta = 3$ in a different language, namely matrices and graphs. An object is probabilistically moving from the set of discrete points $\mathcal{S} = \{1, 2, ..., \mathtt{S}\}$ to $\mathcal{S}$ at each discrete point in time $\mathcal{T} = \{1, 2, ..., \mathtt{T}\}$.

The object's location is described by a random vector $\underline{X} = (X_1, X_2, ..., X_{\mathtt{T}})$.

I assume the initial position X_0 is known and choose a simple probabilistic motion model, M_1, that states that the object moves left, right, or straight ahead with equal probability,

$$P(X_{n+1} \mid X_n) = \left\{ \begin{array}{lll} \frac{1}{3} & \text{if} & X_{n+1} = X_n - 1 \\ \frac{1}{3} & \text{if} & X_{n+1} = X_n \\ \frac{1}{3} & \text{if} & X_{n+1} = X_n + 1 \\ 0 & \text{otherwise} & \end{array} \right\}, 1 \leq n \leq (\mathrm{T} - 1).$$

Therefore, the true path $\underline{X}$ is a member of the set of possible paths

$$\mathcal{P} = \{\underline{\gamma} \in \mathcal{S}^{\mathrm{T}} \mid |\gamma_i - X_0| \leq i - 1, |\gamma_j - \gamma_{j+1}| \leq 1\} \subset \mathcal{S}^{\mathrm{T}}.$$

We do not observe the object's path $\underline{X}$. Instead we observe a matrix of measurements, $\underline{\underline{Y}}$. These measurements depend probabilistically on the object's path $\underline{X}$ in the following way:

$$P(Y_{t,s} | X_t) = \left\{ \begin{array}{lll} 1 - \alpha & \text{if} & X_t = s \text{ and } Y_{t,s} = 1 \\ \alpha & \text{if} & X_t = s \text{ and } Y_{t,s} = 0 \\ 1 - \alpha & \text{if} & X_t \neq s \text{ and } Y_{t,s} = 0 \\ \alpha & \text{if} & X_t \neq s \text{ and } Y_{t,s} = 1 \end{array} \right\}, \forall (s,t) \in \mathcal{S} \times \mathcal{T}.$$

For notational convenience let us define the sequence of functions: $Y_t(s) : \mathcal{S} \to \{0,1\}, t \in \mathcal{T}$ defined such that $Y_t(s) = Y_{t,s}$.

4.2.1 Optimization Problem

Fix a parameter called $\delta = 3$ that is a positive integer. I am interested in finding a parameter $\underline{A}$ from the set $\mathcal{P}$ that best parametrizes a cylinder $C_\delta(\underline{A})$, hence finding the best cylinder. Intuitively, a cylinder is a set of paths that are spatially close to each other. More formally, $C_\delta(\underline{a}) \triangleq \{\underline{\gamma} \in \mathcal{P} \mid |\underline{\gamma} - \underline{a}|_\infty \leq \delta\}$.

So the problem is to devise an efficient, deterministic, offline algorithm that given sensor input $\underline{\underline{Y}}$, will output $\underline{A} \triangleq (A_1, ..., A_{\mathtt{T}})$such that

$$\underline{A}(\underline{\underline{Y}}) = arg \max_{\underline{a} \in \mathcal{P}} P[\underline{X} \in C_\delta(\underline{a}) \mid \underline{\underline{Y}}](\clubsuit)$$

4.2.2 Problem Simplification

Let us define a function that takes a time t, set of measurements at time t, say $\underline{Y}_t = (Y_{t,1}, Y_{t,2}, ..., Y_{t,\mathtt{S}})$ and a possible location for the object at time t, say $l_t \in \mathcal{S}$, and outputs the number of local (in time) undamaged measurement bits at time t.

Therefore I define

$$N(t, \underline{Y}_t, l_t) \triangleq \mathtt{S} - \sum_{l=1}^{\mathtt{S}} Y_{t,l} + I(t, Y_{t,l_t}, l_t) - O(t, Y_{t,l_t}, l_t)$$

where $I(t, Y_{t,l_t}, l_t)$ is the indicator function that returns 1 if the measurement Y_{t,l_t} is a one and 0 otherwise. Likewise $O(t, Y_{t,l_t}, l_t)$ is the indicator function that returns 1 if the measurement Y_{t,l_t} is a zero and 0 otherwise.

Therefore we can rewrite

$$\begin{aligned} N(t, \underline{Y}_t, l_t) &= \mathtt{S} - \textstyle\sum_{l=1}^{\mathtt{S}} Y_t(l) + Y_t(l_t) - (1 - Y_t(l_t)) \\ &= \mathtt{S} - \textstyle\sum_{l=1}^{\mathtt{S}} Y_t(l) + 2Y_t(l_t) - 1 \end{aligned}$$

So for example say $\mathtt{S} = 9$ and $\mathtt{T} = 5$. Then we have a 5×9 matrix of measurements which are 0s and 1s. Set the initial location to be $X_0 = 5$. At time $t = 1$, the object can be either at position $4, 5$, or 6. If the measurements at time $t = 1$ are $\underline{Y}(1) = (0, 0, 1, 0, 1, 1, 0, 0, 0)$ then the respective number of local undamaged bits for a path going through location $s = 4$ is $N(1, \underline{Y}(1), 4) = 1+1+0+0+0+0+1+1+1 = 5$. Similarly, for the same measurements at time $t = 1$, the number of local undamaged bits for a path going though location $s = 5$ is $N(1, \underline{Y}(1), 5) = 1 + 1 + 0 + 1 + 1 + 0 + 1 + 1 + 1 = 7$.

Now let us use the above formalism to simplify the optimization problem even further:

$$\begin{aligned}
\underline{A}(\underline{\underline{Y}}) &= arg\max\nolimits_{\underline{a}\in\mathcal{P}} P[\underline{X} \in C_\delta(\underline{a}) \mid \underline{\underline{Y}}] \\
&= arg\max\nolimits_{\underline{a}\in\mathcal{P}} \sum\nolimits_{\underline{\gamma}\in C_\delta(\underline{a})} P(\underline{\underline{Y}} \mid \underline{\gamma}) \\
&= arg\max\nolimits_{\underline{a}\in\mathcal{P}} \sum\nolimits_{\underline{\gamma}\in C_\delta(\underline{a})} \prod\nolimits_{t=1}^{\mathtt{T}} \prod\nolimits_{s=1}^{\mathtt{S}} P(Y_t(s) \mid \gamma_t) \\
&= arg\max\nolimits_{\underline{a}\in\mathcal{P}} \sum\nolimits_{\underline{\gamma}\in C_\delta(\underline{a})} \prod\nolimits_{t=1}^{\mathtt{T}} (1-\alpha)^{N(t,\underline{Y}_t,\gamma_t)} \alpha^{(S-N(t,\underline{Y}_t,\gamma_t))} \\
&= arg\max\nolimits_{\underline{a}\in\mathcal{P}} \sum\nolimits_{\underline{\gamma}\in C_\delta(\underline{a})} (1-\alpha)^{\sum_{t=1}^{\mathtt{T}} N(t,\underline{Y}_t,\gamma_t)} \alpha^{\sum_{t=1}^{\mathtt{T}}(S-N(t,\underline{Y}_t,\gamma_t))} \\
&= arg\max\nolimits_{\underline{a}\in\mathcal{P}} \sum\nolimits_{\underline{\gamma}\in C_\delta(\underline{a})} (1-\alpha)^{2\sum_{t=1}^{\mathtt{T}} Y_t(\gamma_t)} \alpha^{-2\sum_{t=1}^{\mathtt{T}} Y_t(\gamma_t)} \\
&= arg\max\nolimits_{\underline{a}\in\mathcal{P}} \sum\nolimits_{\underline{\gamma}\in C_\delta(\underline{a})} (\tfrac{1-\alpha}{\alpha})^{2\sum_{t=1}^{\mathtt{T}} Y_t(\gamma_t)}
\end{aligned}$$

This simplified version is friendlier to work with because the individual measurement values on the paths of interest in a given cylinder show up directly in the optimization problem

For example, if the noise paramater that flips bits $\alpha = \frac{1}{3}$, then

$$\underline{A}(\underline{\underline{Y}}) = arg\max_{\underline{a}\in\mathcal{P}} \sum_{\underline{\gamma}\in C_\delta(\underline{a})} 4^{\sum_{t=1}^{\mathtt{T}} Y_t(\gamma_t)}.$$

4.2.3 A Shortest/Longest Path Problem

Remember that the measurement values, $Y_{t,l}$ can only take on values in the set $\{0,1\}$. Fix the measurements. For simplification of notation, without loss of generality let us assume $\alpha = \frac{1}{3}$. And also let us assume $\delta = 3$ so that at any point in time, all of the cylinder's paths go through three consecutive points in $\mathcal{S}$. Also to avoid worries about the boundary cases, let us assume $\mathtt{S}$ is much bigger than $\mathtt{T}$ and that the initial location, X_0, of the object is somewhere in the middle of $\{1,2,3,...,\mathtt{S}\}$.

Now let us create a directed graph $\mathcal{G}$. Consider the lattice of points $\mathcal{T}\times\mathcal{S}$. Let each of those points $(t,s)\in\mathcal{T}\times\mathcal{S}$ define a vertex in graph $\mathcal{G}$. Define the edges of $\mathcal{G}$ in the following manner:

- add one node to the lattice and label it $(t=0, s=X_0)$.
- create edge $e_{-1}(0,X_0)$ that leaves vertex $(0,X_0)$ and points left to vertex $(1,X_0-1)$, edge $e_0(0,X_0)$ that leaves vertex $(0,X_0)$ and points straight to vertex $(1,X_0)$,

edge $e_1(0, X_0)$ that leaves vertex $(0, X_0)$ and points right to vertex $(1, X_0 + 1)$.

- now for every vertex (s, t) whose time index $t \leq \mathtt{T} - 1$ and that has an edge pointing to it, create three edges pointing away from it. Namely create edge $e_{-1}(t, s)$ that leaves vertex (t, s) and points to $(t + 1, s - 1)$, create edge $e_0(t, s)$ that leaves vertex (t, s) and points to $(t + 1, s)$, and create edge $e_1(t, s)$ that leaves vertex (t, s) and points to $(t + 1, s + 1)$.

In the formulation that is about to take place, we will be looking to find the path in the graph from $(1, X_0)$ to $(\mathtt{T}, s)$ that maximizes some value function for all s's that are "reachable" from X_1. Then we will say that the path amongst all those paths with the maximal value is the $\underline{A}(\underline{\underline{Y}})$ we were seeking in the first place.

So let us associate a 3×3 matrix $E_i(t, s, \underline{\underline{Y}})$ with each edge $e_i(t, s)$ in the following manner:

With edge $e_{-1}(t, s)$ we associate matrix

$$E_{-1}(t, s, \underline{\underline{Y}}) = \begin{pmatrix} 4^{Y_{t+1,s-2}} & 0 & 0 \\ 4^{Y_{t+1,s-1}} & 4^{Y_{t+1,s-1}} & 0 \\ 4^{Y_{t+1,s}} & 4^{Y_{t+1,s}} & 4^{Y_{t+1,s}} \end{pmatrix}$$

With edge $e_0(t, s)$ we associate matrix

$$E_0(t, s, \underline{\underline{Y}}) = \begin{pmatrix} 4^{Y_{t+1,s-1}} & 4^{Y_{t+1,s-1}} & 0 \\ 4^{Y_{t+1,s}} & 4^{Y_{t+1,s}} & 4^{Y_{t+1,s}} \\ 0 & 4^{Y_{t+1,s+1}} & 4^{Y_{t+1,s+1}} \end{pmatrix}$$

With edge $e_1(t, s)$ we associate matrix

$$E_1(t, s, \underline{\underline{Y}}) = \begin{pmatrix} 4^{Y_{t+1,s}} & 4^{Y_{t+1,s}} & 4^{Y_{t+1,s}} \\ 0 & 4^{Y_{t+1,s+1}} & 4^{Y_{t+1,s+1}} \\ 0 & 0 & 4^{Y_{t+1,s+2}} \end{pmatrix}$$

Now let us say that we are at some node (t, s) and have some taken some path

$(t-1, s_{t-1}), ...(1, X_0)$ to get there then there is a 3×1 value vector v_t associated with the path so far. I will explain how to obtain this value vector for a path inductively:

- If at node (t, s) we decide to go to node $(t+1, s-1)$ then the value vector $v_{t+1} = E_{-1}(t, s, \underline{\underline{Y}})v_t$.
- If at node (t, s) we decide to go to node $(t+1, s)$ then the value vector $v_{t+1} = E_0(t, s, \underline{\underline{Y}})v_t$.
- If at node (t, s) we decide to go to node $(t+1, s+1)$ then the value vector $v_{t+1} = E_1(t, s, \underline{\underline{Y}})v_t$.
- The value vector v_1 for the initial subpath $(1, X_0)$ is $v_1 = (4^{Y_{1,X_1-1}}, 4^{Y_{1,X_1}}, 4^{Y_{1,X_1+1}})$.

Now the value of a path that has an associated value vector $v_\mathrm{T} = (a, b, c)$ is $(a+b+c)$. So looking at the last result of the last section, finding $A(\underline{\underline{Y}})$ amounts to finding the ordered set of nodes on the path from $(1, X_0)$ to a node indexed at time T such that the path has highest value amongst all such paths.

4.3 Difficulty of Problem

This problem is much tougher than the previous two regimes of the problem that were discussed in chapter two and chapter three. Iteration of matrices in general is NP-hard. I have not shown this difficulty for this problem but finding an optimal sequence of matrices has only been studied in terms of joint spectral radius and control theory and in both cases very limited problems have been solved [6]. This regime of the problem however is interesting precisely because it reduces to a problem of picking an optimal sequence of matrices.

4.4 Heuristics

The heuristics described in this section rely on finding a δ' sized nieghborhood and then modifying the neighborhood so that the result is a δ-neighborhood that hopes to approximate the probability of the MAP δ-neighborhood. These heuristics could experimentally be shown to work well for small values of α.

One such heuristic is to fix $\delta' = 1$ and solve the problem exactly as in chapter two to find an $\underline{A}'$. The heuristic outputs the δ-neighborhood centered about $\underline{A}'$. The probability that the true path is contained within that region may be a very good lower bound for the probability that the true path is contained in the MAP δ-neighborhood depending on the level of noise, α.

Another heuristic is to set $\delta' = \Delta$ and to solve the problem exactly as in chapter three and obtain a δ'-neighborhood. The heuristic would output as an answer a δ-neighborhood that results from growing the δ'-neighborhood uniformly in both directions until it is a *delta* sized neighborhood.

4.5 Summary

Let us explain why this problem regime is difficult using linear algebra. We showed that our problem was equivalent to finding an "optimal" sequence of matrices. Imagine you were working with the space of 2x2 rotation matrices and try analyzing the effect of a rotation transformation on the norm of a state vector. The norm changes in manner that is difficult to write down in closed form. If we could find a decomposition for matrices into a product of matrices where the effect of the matrices on the norm is either invariant or multiplicative by a scalar [2], then we could make further progress on this problem.

Without knowing about future rotations one would have to resort to a randomized strategy to have any hope of achieving a value close to the maximum. Our problem is easier than the rotation matrices problem because our matrices have a special form. Nonetheless it is more much difficult than the problem regimes in chapters two and three.

[2]that is solely defined by the matrix

Chapter 5

Conclusions

In this thesis we have seen that we can devise an efficient algorithm for the problem of finding the δ-neighborhood that contains the object's true trajectory with maximal probability for a class of prior "motion probability models", M_Δ, given noisy sensor measurements. We found that this optimal algorithm generalizes for problems involving higher dimensional motion with only polynomial growth in complexity. We also witnessed a phase transition in the problem when the movement range, Δ, is strictly less than the neighborhood size, δ. This phase transition made that problem regime a more difficult combinatorial optimization problem.

5.1 The problem's phase transition

What does all of this mean?

Well it has to do with model flexibility. If your motion model is very restrictive on a micro-scale relative to the macro-scale optimization problem, then the optimization problem becomes difficult due to their dependencies. In this particular problem, we realized this lesson with M_Δ when we varied Δ in relation to δ.

This lesson is very similar to the importance of the signal to noise ratio in estimation. There has been work done on analyzing phase transitions that are based on the signal to noise ratio [20]. In nonlinear estimation problems it turns out that the signal to noise ratio tells you how good an estimator you can have [9]. Similar phase transition findings in percolation theory are still being discovered [17]. In our work it turns out that the flexibility of our motion model in comparison to the neighborhood size is the signal strength. This notion can be formalized further.

5.2 Lessons in algorithmic design

Algorithmic design is an art. Devising an efficient algorithm for a problem requires constant feedback to the problem formulation. You can visualize an algorithmic design box as a black box. As input it has the parameters of the problem formulation [1] and it outputs an algorithm that you design that you hypothesize finds some value efficiently. If you find that for certain input parameter values your black box doesn't provide you with an efficient algorithm you have to redefine your box to accept different parameters or you have to give your black box more tools to work with. After enough iterations of this process, interesting results come about.

5.3 Open problems

Now let us discuss some questions and hence open problems in this work.

5.3.1 The formulation

Initially when formulating the mathematical optimization problem it seemed that finding a neighborhood that was highly likely to include the object's actual trajectory and hence "approximately" defined the object's path would be computationally less expensive than finding the exact path. In general this was not the case and depended heavily on the probability models employed. That is not a problem since there were more important motivating factors for our formulation and the complexity was approximately the same.

However an interesting extension of our formulation would be to allow neighborhoods that varied in size according to the level of noise at each time instant. It seems

[1]for example, δ and Δ

that one could find non-trivial useful regions that contained the actual path with extremely high probability.

5.3.2 Motion probability models

Also using a probability model where all paths were equally likely was not very constraining because we could have accounted for the case where they had different probabilities. However, the notation would have become messy.

But it would be an interesting extension to solve our problem assuming other prior "motion probability models". For example, if one uses the the prior "motion probability model" that is a markov chain that relates position, velocity, acceleration, and jerk, more real world tracking problems could be solved.

5.3.3 Another possible noise model

In the area of algorithmic design an open problem is to change our noise model and see how well any algorithm can still do. For example, we could restate the problem with a "shot noise" model and a "localization noise" model. Effectively the "shot noise" model will be the same source of noise as described in our formulation and the "localization noise" model will say that bits are less likely to get flipped as you move away from the object's actual location. So let us describe the problem with this model.

An object is probabilistically moving from the set of discrete points $\mathcal{S} = \{1, 2, ..., \mathtt{S}\}$ to $\mathcal{S}$ at each discrete instant of time $\mathcal{T} = \{1, 2, ..., \mathtt{T}\}$.

The object's location is described by a random vector $\underline{X} = (X_1, X_2, ..., X_{\mathtt{T}})$.

I assume the initial position X_0 is known and choose a simple probabilistic motion model that states that the object moves left, right, or straight ahead with equal probability,

$$P(X_{n+1} \mid X_n) = \left\{ \begin{array}{lll} \frac{1}{3} & \text{if} & X_{n+1} = X_n - 1 \\ \frac{1}{3} & \text{if} & X_{n+1} = X_n \\ \frac{1}{3} & \text{if} & X_{n+1} = X_n + 1 \\ 0 & \text{otherwise} & \end{array} \right\}, 1 \leq n \leq (\mathtt{T} - 1).$$

Therefore, the true path $\underline{X}$ is a member of the set of possible paths

$$\mathcal{P} = \{\underline{\gamma} \in \mathcal{S}^{\mathtt{T}} \mid |\gamma_i - X_0| \leq i, |\gamma_j - \gamma_{j+1}| \leq 1\} \subset \mathcal{S}^{\mathtt{T}}.$$

We do not observe the object's path $\underline{X}$. Instead we observe a matrix of measurements, $\underline{\underline{Y}}$. These measurements depend probabilistically on the object's path $\underline{X}$ in the following way:

$$Y_{t,s} = (I_{s,X_t} + Q_t(s - X_t) + W_{t,s}) \bmod 2$$

where the indicator function

$$I_{s,X_t} = \left\{ \begin{array}{lll} 1 & \text{if} & X_t = s \\ 0 & \text{otherwise} & \end{array} \right\} \forall (s,t) \in \mathcal{S} \times \mathcal{T},$$

and the space localization noise

$$Q_t(l) = \left\{ \begin{array}{ll} 1 & \text{if } |l| \leq 3\delta \text{ with probability } \ \beta^l \\ 0 & \text{otherwise} \end{array} \right\} \forall l \in \mathcal{S},$$

and the shot noise

$$W_{t,s} = \left\{ \begin{array}{ll} 1 & \text{with probability } \ \alpha \\ 0 & \text{otherwise} \end{array} \right\}.$$

Let all the usual iid assumptions be made.

For notational convenience let us define the sequence of functions: $Y_t(s) : \mathcal{S} \to \{0,1\}, t \in \mathcal{T}$ defined such that $Y_t(s) = Y_{t,s}$.

Optimization Problem Associated with that Change

Fix a parameter called δ that is a positive integer. It would be interesting to find a parameter $\underline{A}$ from the set $\mathcal{P}$ that best parametrizes a cylinder $C_\delta(\underline{A})$, a set of paths that are spatially close to each other. More formally, $C_\delta(\underline{a}) \triangleq \{\underline{\gamma} \in \mathcal{P} \mid |\underline{\gamma} - \underline{a}|_\infty \leq \delta\}$.

So the problem is to devise an efficient, deterministic, offline algorithm that given sensor input $\underline{\underline{Y}}$, will output $\underline{A} \triangleq (A_1, ..., A_T)$such that

$$\underline{A}(\underline{\underline{Y}}) = arg \max_{\underline{a} \in \mathcal{P}} P[\underline{X} \in C_\delta(\underline{a}) \mid \underline{\underline{Y}}](\clubsuit)$$

It should be possible to use "statistical" differences in the noise in order to find out where the object would not be with high probability so that we could find the tube that contains the true path more efficiently. Hence with this change in the noise model we may be able to find a more efficient optimal algorithm.

5.3.4 Finding optimal sequences of matrices

The final problem regime that we explored in this thesis is related to another open problem of how one can find optimal sequences of matrices that maximize some norm or more formally how one can optimize over semi-groups.

Appendix A

Source Code

```
%clear workspace...you can also use who to see what is defined.
clear


% ----------------------------------SETUP
% set the space and time variables
S=100;
T=50;
InitialPosition=50;


% ----------------------------------GENERATE THE DATA
% initialize the SpaceTime data Y, and the path to the init pos
X=zeros(T);
X0=InitialPosition;
Y_noise_free=zeros(T,S);


% generate the random path according to the prior and generate the noise matrix
```

```
path=random('Uniform',0,1,T,1);
noise=random('Uniform',0,1,T,S);
if path(1) > 2/3
  X(1)=X0 + 1;
  Y_noise_free(1,X(1))=1;
elseif path(1) > 1/3
  X(1)=X0;
  Y_noise_free(1,X(1))=1;
elseif path(1) > 0
  X(1)=X0 -1;
  Y_noise_free(1,X(1))=1;
end
for t=2:T
   if path(t) > 2/3
        X(t)=X(t-1) + 1;
        Y_noise_free(t,X(t))=1;
   elseif path(t) > 1/3
        X(t)=X(t-1);
        Y_noise_free(t,X(t))=1;
   elseif path(t) > 0
        X(t)=X(t-1) -1;
        Y_noise_free(t,X(t))=1;
   end
end

% display the random path without noise, note that gtext is cool
colormap(gray);
figure(1);
imshow(Y_noise_free);
axis on;
```

```
title('Object Path (without Noise)');
xlabel('Spatial Location (in pixels)');
ylabel('Time (in frames)');

% add the noise, and display the random path immersed in the noise
% let alpha=.03
alpha=.03;
Y=zeros(T,S);
for t=1:T
  for s=1:S
    if noise(t,s) < alpha
      Y(t,s)=mod(Y_noise_free(t,s)+1,2);
    else
      Y(t,s)=Y_noise_free(t,s);
    end
  end
end
figure(2);
imshow(Y);
axis on;
title('Object Path (w/ Noise, \alpha=.03)');
xlabel('Spatial Location (in pixels)');
ylabel('Time (in frames)');


% ------------------------------------------------- START ALGORITHM
% now lets start find the best path.  formally lets find
% argmax_{X \in SetOfPossiblePaths} P(X | Y)

% since P(X | Y) = P(Y |X) P(X) / sum_{x in SetOfPossiblePaths} [P(Y | x) P(x)]
%                P(Y |X) / sum_{x in SetOfPossiblePaths} P(Y | x)
% the algorithm we need will have two sub-algorithms:
% 1) we need DenomTable to calculate the denominator (like pascals triangle)
% 2) we will use a breadth-like algorithm to maximize the numerator
%    for this part we will need NumTable
% note for MATLAB code only: DenomTable and NumTable indices will be offset
%      by 1.  So index 1 really refers to paths that have 0 1s on their path.

% initialize DenomTable and NumTable
maxPossibleOnesOnPath=T;
DenomTable=zeros(2,S,maxPossibleOnesOnPath+1);
%              A B C
% A because we keep tables for last two frames
% B imples that we have 2 x S tables at most
% C because each table has maxPossibleOnesOnPath possible indices
% Now we initialize DenomTable with values for t=1

DenomTable(1,X0  , Y(1,X0)+1)=1;
DenomTable(1,X0-1, Y(1,X0-1)+1)=1;
DenomTable(1,X0+1, Y(1,X0+1)+1)=1;


NumTable=zeros(2,S);
%             A B
% A because we keep tables for last two frames
% B implies that we have 2 x S values at most
% Now we initialize NumTable with values for t=1
NumTable(1,X0  ) = Y(1,X0)+1;
NumTable(1,X0-1) = Y(1,X0-1)+1;
NumTable(1,X0+1) = Y(1,X0+1)+1;
```

```
BestParent=zeros(T,S);
BestParent(1,X0)=X0;
BestParent(1,X0−1)=X0;
BestParent(1,X0+1)=X0;

for t=2:T
 for s=(X0−t):(X0+t)
  % first determine NumTable for this timestep t
  [maxval,maxindex]=max(NumTable(1,(s−1):(s+1)));
  NumTable(2,s)=maxval+Y(t,s);
  if maxindex==1
   maxindex=s−1;
  elseif maxindex==2
   maxindex=s;
  elseif maxindex==3
   maxindex=s+1;
  end
  BestParent(t,s)=maxindex;

  % now determine DenomTable for this timestep t
  % note: we ignore boundary conditions for now. .in the sense
  %       of what if (t,s-1) or (t,s+1) runs us over the boundaries s=1 or s=S
  maxPossOnesBefore=t−1;
  for k=1:maxPossOnesBefore+1
     DenomTable(2,s,k+Y(t,s)) =     DenomTable(1,s−1,k) + ...
                                 DenomTable(1,s,  k) + ...
                                 DenomTable(1,s+1,k);
  end
 end
 % end of s for-loop
```

```
  disp(t);
  disp('entering update loop');
  % update NumTable and DenomTable(ie move info for time 2 to time 1)
  for s=(X0−t):(X0+t)
    NumTable(1,s)=NumTable(2,s);
    maxPossOnesNow=t;
    for k=1:maxPossOnesNow+1
      DenomTable(1,s,k)=DenomTable(2,s,k);
      DenomTable(2,s,k)=0;
    end
  end
  disp('exiting update loop');

end
% end of t loop

% ---------- NOW WE HAVE ALL THE TABLES FOR BEST PATH
% find best path
[maxNumVal,finalPos]=max(NumTable(1,1:S));
BestPath=zeros(1,T);
BestPath(T)=finalPos;
for j=1:(T−1)
  BestPath(T−j)=BestParent(T−j+1, BestPath(T−j+1));
end

% ------------------------------------------ RENDER BEST PATH
% color code:
%    red path is true path
%    blue path is maximum likelihood path
%     gray is noise
figure(3)
red_blue_map=[ 0 0 0; 1 1 1; 1 0 0; 0 0 1];
black=1;
gray=2;
red=3;
blue=4;


RenderThis=zeros(T,S);
for t=1:T
  for s=1:S
    if Y_noise_free(t,s)==1
      RenderThis(t,s)=red;
    elseif Y_noise_free(t,s)==0
      RenderThis(t,s)=black;
    end
    if BestPath(t)==s
      RenderThis(t,s)=blue;
    end
  end
end


imagesc(RenderThis);
colormap(red_blue_map);
axis on;
title('Object Path (w/ Noise, \alpha=.03)');
xlabel('Spatial Location (in pixels)');
ylabel('Time (in frames)');
```

```
% ----------------------- CALCULATE DENOMINATOR
% Calculate Denominator by agregating the denomTables
AggregateDenomTable=zeros(1,maxPossibleOnesOnPath+1);
DenomTerms=zeros(1,maxPossibleOnesOnPath+1);
for j=1:maxPossibleOnesOnPath+1
  AggregateDenomTable(j)=sum(DenomTable(1,1:S,j));
  DenomTerms(j)=AggregateDenomTable(j)*((1-alpha)/alpha)^(2*(j-1));
end
DenomVal=sum(DenomTerms);


% ------------------ CALCULATE MAXPATH PROBABILITY
% Calculate Maximized Probability
maxNumVal
maxVal=((1-alpha)/alpha)^(2*(maxNumVal-1))/DenomVal;

% -------- CALCULATE A DELTA TUBE's PROBABILITY
% Calculate the delta tube that uses BestPath as the axis
%  note: we require that delta be greater than or equal to 1
delta=2;
TubeNumTable=zeros(2,S,maxPossibleOnesOnPath+1);
%                 A B C
% A because we keep tables for last two frames
% B because we have 2 x S tables at most
% C because each table has maxPossibleOnesOnPath+1 possible indices
% Now we initialize NumTable with values for t=1
% note for MATLAB code only: DenomTable and NumTable indices will be offset
%      by 1.  So index 1 really refers to paths that have 0 1s on their path.
```

```
TubeNumTable(1,X0  , Y(1,X0)+1    )=1;
TubeNumTable(1,X0−1, Y(1,X0−1)+1  )=1;
TubeNumTable(1,X0+1, Y(1,X0+1)+1  )=1;


for t=2:T
  for s=(BestPath(t)−delta):(BestPath(t)+delta)
    if abs(s−X0) <= t
      % note: we ignore boundary conditions for now. .in the sense
      %       of what if (t,s-1) or (t,s+1) runs us over the boundary
      maxPossOnesBefore=t−1;
      for k=1:maxPossOnesBefore+1
        TubeNumTable(2,s,k+Y(t,s)) = TubeNumTable(1,s−1,k) + . . .
                                     TubeNumTable(1,s,  k) + . . .
                                     TubeNumTable(1,s+1,k);
      end
    end
  end
  % end of s for-loop

  disp(t);
  disp('entering update loop');
  % now lets update the TubeNumTable (ie move info for time 2 to time 1)
  for s=(BestPath(t)−delta):(BestPath(t)+delta)
    if abs(s−X0) <= t
      maxPossOnesNow=t;
      for k=1:maxPossOnesNow+1
        TubeNumTable(1,s,k)=TubeNumTable(2,s,k);
        TubeNumTable(2,s,k)=0;
      end
    end
  end
  disp('exiting update loop');

end
% end of t loop

% ---------------------------------- CALCULATE NUMERATOR
% Calculate Numerator by agregating the TubeNumTables
AggregateTubeNumTable=zeros(1,maxPossibleOnesOnPath+1);
TubeNumTerms=zeros(1,maxPossibleOnesOnPath+1);
for j=1:maxPossibleOnesOnPath+1
  AggregateTubeNumTable(1,j)=sum(TubeNumTable(1,BestPath(T)-delta:
                              BestPath(T)+delta,j));
  TubeNumTerms(1,j)=AggregateTubeNumTable(j)*((1-alpha)/alpha)^(2*(j-1));

end
TubeNumVal=sum(TubeNumTerms);



% ---------------------------------- CALCULATE TUBE PROBABILITY
TubeVal=TubeNumVal/DenomVal
```

Bibliography

[1] Mohamad A. Akra. *Automated Text Recognition.* PhD dissertation, Massachusetts Institute of Technology, Department of Electrical Engineering, 1995.

[2] D. Bertsekas. *Dynamic programming and optimal control.* Athena Scientific, Belmont, MA., 1995.

[3] Andrew Blake. *Active Contours.* Springer, New York, NY., 1998.

[4] J.M. Coughlan and A.L. Yuille. Bayesian a* tree search with expected o(n) convergence rates for road tracking. In *Proc. of Second International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition*, 1999.

[5] T.M. Cover. *Elements of information theory.* Wiley, New York, NY., 1991.

[6] D. D'Alessandro, M. Dahleh, and I. Mezic. Control of mixing in fluid flow: a maximum entropy approach. *IEEE Transactions on Automatic Control*, 44(10):1852–1863, October 1999.

[7] D. Geman and B. Jedynak. An active testing model for tracking roads in satellite images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(1):1–14, January 1996.

[8] Michael Isard and Andrew Blake. Icondensation: Unifying low-level and high-level tracking in a stochastic framework. In *Proc. Fifth European Conference on Computer Vision*, number 1, pages 893–908, 1998.

[9] S.K. Mitter and N. Newton. The duality between estimation and control. In *Conference in honor of Professor Alain Bensoussan's 60th birthday*, 2000.

[10] A. Papoulis. *Probability, random variables, and stochastic processes.* McGraw-Hill, New York, NY., 1991.

[11] R. Plamondon. The design of an on-line signature verification system: From theory to practice. *Machine Perception and Artificial Intelligence*, 13:795–811, 1994.

[12] R. Plamondon. A kinematic theory of rapid human movements: part i: Movement representation and generation, part ii: Movement time and control. *Biological Cybernetics*, 72(4):295–320, 1995.

[13] R. Plamondon and S. Srihari. On-line and off-line handwriting recognition: A comprehensive survey. *Special Issue of IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):63–84, January 2000.

[14] James M. Rehg. *Visual Analysis of High DOF Articulated Objects with Application to Hand Tracking.* PhD dissertation, Carnegie Mellon University, Department of Electrical and Computer Engineering, 1996.

[15] J.A. Rice. *Mathematical statistics and data analysis.* McGraw-Hill, Belmont, CA., 1995.

[16] Navid Sabbaghi. Conjectured online signature invariants. Unpublished material.

[17] D. Stauffer. *Introduction to percolation theory.* Taylor and Francis, Washington, DC, 1992.

[18] Christopher E. Strangio. Automatic signature verification. Master's thesis, Massachusetts Institute of Technology, Department of Electrical Engineering, 1976.

[19] A.L. Yuille and J.M. Coughlan. High-level and generic models for visual search: when does high level knowledge help? In *Proc. Computer Vision and Pattern Recognition*, 1999.

[20] A.L. Yuille and J.M. Coughlan. Fundamental limits of bayesian inference: Order parameters and phase transitions for road tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(2):160–173, February 2000.