The Quaternion Bingham Distribution, 3D Object Detection, and Dynamic Manipulation

by

Jared Marshall Glover

Submitted to the Department of Electrical Engineering and Computer Science

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2014

© Massachusetts Institute of Technology 2014. All rights reserved.

Signature redacted

Author ........ 

Department of Electrical Engineering and Computer Science

May 21, 2014

Signature redacted

Certified by....... 

Leslie Pack Kaelbling
Professor
Thesis Supervisor

Signature redacted

Certified by..... / 

Tomás Lozano-Pérez
Professor
Thesis Supervisor

Signature redacted

Accepted by ........... 

Professor Leslie A. Kolodziejski
Chair of the Committee on Graduate Students
The Quaternion Bingham Distribution, 3D Object Detection, and Dynamic Manipulation

by

Jared Marshall Glover

Submitted to the Department of Electrical Engineering and Computer Science on May 21, 2014, in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Abstract

Over the past few years, the field of robotic computer vision has undergone a 3-D revolution. One of the biggest challenges in dealing with 3-D geometry lies in appropriately handling 3-D rotational data. To specify “where” an object is in space, one must provide both a position and an orientation for the object. Noise and ambiguity in the robot’s sensory data necessitate a robust model for representing uncertainty on the space of 3-D orientations. This is given by the quaternion Bingham distribution— a maximum entropy probability distribution on the 4-D unit quaternion hypersphere.

In this thesis, we apply the quaternion Bingham to two applications: 3-D object instance detection from RGB-D images, and robot ping pong. The Bingham enables our object detection system to achieve state-of-the-art detection rates in highly cluttered scenes, while also enabling the ping pong robot to track the orientation and spin on flying ping pong balls. To enable the robot to actually play ping pong, we also explored a new method for incorporating human advice into a robot’s motor control exploration policies.

Thesis Supervisor: Leslie Pack Kaelbling
Title: Professor

Thesis Supervisor: Tomás Lozano-Pérez
Title: Professor
Acknowledgments

I began this journey with a change of plans. Upon arriving at Willow Garage for an internship, my hopes for getting their PR2 robot to play ping pong were quickly dashed when it became apparent that the robot would be too slow for the job. Then, in swooped Gary Bradski with his idea of using 3-D local feature orientations for object localization, and a light bulb went off. This thesis would have been written on an entirely different subject if Gary hadn’t planted that idea in my head. I owe Gary, Radu Rusu, and Kurt Konolige my utmost thanks for their guidance and inspiration during my time at Willow; my gratitude also goes to Baskara Marthi for recruiting me to Willow in the first place.

Thanks also to my masters advisers, Nicholas Roy and Daniela Rus, for putting up with me when I had no clue what I was doing—both in research and in time management. Thanks to Mark Schnepf at Two Sigma for teaching me how to be professional and get things done.

My return to MIT would not have been possible without the support of Nick Roy. Nick, Mike Montemerlo, Greg Armstrong, and Sebastian Thrun are responsible for introducing me to the world of robotics way back in the Wean hall days at Carnegie Mellon. Thanks for giving me a sip of the Kool-Aid!

My Ph.D advisers, Leslie Kaelbling and Tomás Lozano-Pérez, are two of the best teachers and nicest human beings you will ever meet. Thank you for giving me the freedom to pursue my own dreams, and for teaching me to step back and see the big picture.

Thanks to Teresa Cataldo for being the best administrator in MIT, and for drinking my kale juice. Thanks to Ron Wiken for helping to install the ping-pong robot’s arm on the wall, and to Greg Esparza (from Barrett) for fixing the arm two days before the AAAI paper deadline! My third reader, Russ Tedrake, provided many helpful comments on the thesis draft. Thanks to Igor Gilitschenski for checking proofs. And thanks again to Leslie Kaelbling for teaching me about the relationship between chicken pump dispensers and ping pong balls.
I have been fortunate to work with a number of great undergraduates over the past four years: Aaron Fryman, Will Grathwohl, Dawit Zewdie, Jessie Zhang, Sanja Popovic, and Charlotte Zhu have all made contributions to the work in this thesis. The object detection system in chapter 6 would not have been possible without Sanja’s help, and Charlotte was essential to developing the control learning methods in chapter 7.

To my LIS labmates—Lawson Wang, Patrick Barragan, Ariel Anders, Alejandro Perez, Sam Davies, Gustavo Goretkin, Huan Liu, George Konidaris, Chris Amato—thanks for the many hours of thought-provoking conversation and laughter. I will miss you all.
# Contents

1 Introduction ........................................ 16
   1.1 Background ...................................... 17
   1.2 Applications .................................... 19
   1.3 Contributions ................................... 19
   1.4 Thesis Organization ............................... 20

2 The Bingham Distribution .......................... 22
   2.1 Definition and Properties ........................ 23
   2.2 The Normalization Constant ..................... 27
       2.2.1 Computing the Normalization Constant .... 30
   2.3 Maximum Likelihood Parameter Estimation .......... 32
       2.3.1 Computing the Maximum Likelihood Estimates .. 35
   2.4 Bayesian Parameter Estimation .................... 36
   2.5 Characterization ................................ 37
   2.6 Sampling ......................................... 37
   2.7 Mode ............................................. 38
   2.8 Moments .......................................... 38
   2.9 Entropy .......................................... 41
   2.10 Divergence ...................................... 42
   2.11 Product of Bingham PDFs ....................... 43
   2.12 Marginal Distributions .......................... 44
   2.13 Mixtures of Binghams ........................... 45
       2.13.1 Parameter Estimation ....................... 45
### 3 The Quaternion Bingham Distribution

3.1 Understanding the Space of 3-D Rotations .......................... 48
3.2 Representing 3-D Rotations as Unit Quaternions ..................... 51
3.3 Using the Bingham as a Distribution on 3-D Rotations ............... 52
   3.3.1 Rotation by a constant quaternion .......................... 53
   3.3.2 Quaternion inversion ....................................... 53
   3.3.3 Composition of quaternion Binghams ........................ 53
   3.3.4 Estimating the error of approximation ....................... 55
3.4 A Preview of Quaternion Bingham Results in Later Chapters ....... 56

### 4 Tracking 3-D Rotations with the Quaternion Bingham Filter

4.1 Discrete-time quaternion Bingham process .......................... 58
4.2 Discrete quaternion Bingham filter .................................. 60
   4.2.1 Extensions .................................................. 62
4.3 Experimental Results ............................................... 63
4.4 Conclusion ................................................................... 71

### 5 A Generative Method for 3-D Object Detection

5.1 Introduction ........................................................... 74
   5.1.1 Outline ......................................................... 75
5.2 Monte Carlo Pose Estimation .......................................... 75
   5.2.1 Problem Statement ............................................ 76
   5.2.2 MC-SOPE Algorithm ......................................... 78
   5.2.3 MC-SCOPE Algorithm ....................................... 78
5.3 Local Feature Distributions ............................................ 78
7.3.1 Policy gradient advice ................................. 127
7.3.2 Exploration Bias with Directional Advice (EBDA) .... 128
7.4 Robot Ping Pong ........................................ 129
7.5 Experimental Results .................................... 135
7.6 Conclusion ............................................... 139

8 Conclusion ................................................. 140
 8.1 Food for Thought ....................................... 141
 8.2 Beyond Ping Pong ....................................... 143
List of Figures

2-1 The Bingham distribution (in 2-D) is formed by taking a zero-mean Gaussian distribution (with equiprobable ellipses shown in (a) and (b)) in $\mathbb{R}^2$ and intersecting it with the unit circle, $S^1$. The resulting Bingham distribution (whose PDF is represented by the intensity of pixels on the circle in (b) and (c)) has non-zero probability density only on the surface of $S^1$. It is also antipodally symmetric (i.e., $x$ and $-x$ have the same probability density), since the original Gaussian was centered on the origin. ........................................... 23

2-2 The same Bingham distribution on $S^1$ resulting from three Gaussians on $\mathbb{R}^2$ of different scales. ........................................... 24

2-3 A Bingham distribution on $S^2$, with concentration parameters $\lambda_1 = -41.7$ and $\lambda_2 = -2.6$. Shown are the mode, $\mu$, and the two orthogonal direction vectors, $\mathbf{v}_1$ and $\mathbf{v}_2$. ........................................... 25

2-4 The progression of $\lambda_2$ towards zero. ........................................... 26

2-5 More examples of 3-D Binghams. ........................................... 27

3-1 ........................................... 50

4-1 Process and graphical models for the discrete quaternion Bingham process. ........................................... 59
Two simulated runs with the quaternion Bingham filter—(b) and (c) are different plots of the same simulation. In all figures, the thick black line is the true process signal, generated with isotropic process noise \( \Lambda_p = (-400, -400, -400) \). The thin blue lines in (a) and (b) are the observation signal, and the thick red line is the filtered output. Rows 1-4 in each figure show the 4 quaternion dimensions \( (q_w, q_x, q_y, q_z) \). Row 5 in (a) and (b) shows the error between filter output and true state (thick red line), together with the QBF sample error 90% confidence bounds (thin red line). Marginal sample 90% confidence bounds are also shown in the thin red lines in (c).

Example image sequences from the spinning ping-pong ball dataset. In addition to lighting variations and low image resolution, high spin rates make this dataset extremely challenging for orientation tracking algorithms. Also, because the cameras were facing top-down towards the table, tracking side-spin relies on correctly estimating the orientation of the elliptical marking in the image, and is therefore much harder than tracking top-spin or under-spin.

Spinning ping-pong ball tracking results. Top row: comparison of QBF-2 (with and without an oracle-given prior) to QBF-1, EKF-1, EKF-2, and random guessing (for spin classification); QBF-1 and EKF-1 do not show up in the orientation error graph because they only tracked spin. Note that QBF-2 quickly converges to the oracle error and classification rates. Bottom row: QBF-2 results broken down into top-spin/under-spin vs. side-spin. As mentioned earlier, the side-spin data is harder to track due to the chosen camera placement and ball markings for this experiment. Note that the time axes are measured in frames (not seconds).

Simulated spinning ping-pong ball QBF-2 tracking results.
4-6 Average QBF spin tracking error as it tracks the spin through the ball’s bounce on the table. Post-bounce errors are significantly lower with the sample-based bounce tracking method (solid black line) outlined section 4.2.1.

4-7 An example ball trajectory (underspin + left-sidespin) and the state of the QBF as it tracks the ball’s orientation and spin through the bounce. In the top row of ball images, the big red (solid-line) axis is the mode of the QBF’s orientation distribution, and the small red axes are random samples from the orientation distribution. The big green (dashed-line) axis is the detected ball orientation in that image. In the bottom row, the big red (solid-line) axis is the mode of the QBF’s spin distribution, and the small red axes are random samples from the spin distribution. The big green (dashed-line) axis is the ground truth spin, and the black axis in the center is the identity (no-spin), for reference.

5-1 Five 3-D point clouds and their corresponding Q-Image transforms, representing the distribution of local 3-D surface orientations on each object (see section 5.3 for details).

5-2 The Single Object Pose Estimation (SOPE) problem: given a model and a (partial) observation (left), we wish to estimate the pose of the object (right).

5-3 The Single Cluttered Object Pose Estimation (SCOPE) problem: given a model, an observation, and a region of interest (a), we wish to estimate the pose of the object (b).
Informative oriented local (surface patch) features. Observing just a single patch of an object can often constrain the set of feasible object poses to a narrow range of possibilities. In the top row, two different patches on a wine glass are shown, along with four possible object poses consistent with each patch. In the bottom row, a bowl feature is shown, along with four object poses consistent with the observed patch. Both of the observed glass features constrain the orientation of the glass to be aligned with the z-axis, while the bowl feature places much weaker constraints on orientation.

Cluster Q-Images.

Given that an oriented local feature with shape type \( \Omega_j \) has the coordinate frame shown in (a), the range of possible feature positions is restricted to lie on the dark band of the cup in (b).

Fitting a Bingham Mixture Model (BMM). (a) A cluster of similar oriented local features. (b) The local coordinate frames of the oriented local features. (c-d) The local coordinate frames (only the axis part of the axis-angle format is visualized), along with the fitted Binghams (represented by the light bands on the two spheres).

Sample error distributions (bar plots), compared to random guessing for orientation (solid red lines). (a-c) MC-SOPE/uncluttered, (d) MC-SCOPE/cluttered.

(a) Search orientation accuracy of MC-SOPE and MC-SCOPE on each dataset. Each point, \((X, Y)\), in the figure shows the percentage \( Y \) of trials for which a “good” pose sample was found in the top \( X \) samples. (b) Comparison to the proposal sample distribution of SAC-IA (orientation error thresholds are the same as in (a)).

Top sample poses for 7 objects found by the MC-SCOPE algorithm for a cluttered scene. When a “good” pose sample is found within the top 20 samples, the caption under the best sample is shown in bold.
6-1 Object detections found with our system, along with the feature correspondences that BPA used to align the model. Surface features are indicated by red points, with lines sticking out of them to indicate orientations (red for normals, orange for principal curvatures). Edge features (which are orientation-less) are shown by magenta points.

6-2 Rigid alignment of two point sets.

6-3 Distance between corresponding points as a function of orientation.

6-4 The 18 objects in our new Clutter data set.

6-5 Our noise models predict range and normal errors (standard deviations) as functions of surface angle and edge distance (both with respect to the viewpoint).

6-6 Single object detection pipeline.

6-7 Examples of objects correctly aligned by BPA with only two correspondences.

6-8 Object detections found with our system, along with the feature correspondences that BPA used to align the model. Surface features are indicated by red points, with lines sticking out of them to indicate orientations (red for normals, orange for principal curvatures). Edge features (which are orientation-less) are shown by magenta points.

6-9 The Clutter testing data set.

6-10 Comparing BPA with ICP. (Left) The average of the minimum position errors in each alignment trial. (Right) The average of the minimum orientation errors in each alignment trial.

7-1 Implicit vs. explicit policies. In the smooth reward function on the left, a good explicit policy mapping $\mathcal{T}$ to $\mathcal{P}$ can be found. In the reward function on the right, any explicit policy mapping will be throwing away information about alternative modes.
The robot interprets advice from a human coach as an exploration bias. The light green shaded areas on top of the reward functions (top-down view of figure 7-1(a)) in the left and right images indicate what the robots believes are “good” regions to explore, and are inferred from a series of policy gradients given by the human coach (e.g., “swing harder” or “angle your paddle up more”). Note that the exploration advice can either be global (left) or local (right). In our system, local advice is the default; advice is only applied globally (for all $\tau \in \mathcal{T}$) at the coach’s discretion (e.g., by saying “always swing harder”). Note that we used a keyboard interface to communicate with the robot---no human language understanding was done for this work.

Our ping pong robot uses a 7-dof Barrett WAM arm with a ping pong paddle rigidly attached at the wrist.

Simulation results (cumulative reward vs. learning trials) with mixture-of-Gaussian reward functions, for task and parameter spaces of different dimensions.

Robot ping pong results (cumulative reward vs. learning trials) for topspin and underspin balls.
List of Tables

4.1 Projected Gaussian process simulations. Average % mean error decrease for QBF over EKF. ........................................ 66

5.1 MC-SOPE .............................................................. 77
5.2 MC-SCOPE .............................................................. 79

6.1 A comparison of precision and recall. ............................... 117
6.2 Recall on the Clutter data set as a function of the number of scene interpretation samples. ................................. 117
Chapter 1

Introduction

Robots have been the workhorses of industrial manufacturing for decades. The speed and precision at which they can accomplish repetitive (and increasingly non-trivial) tasks is astounding. However, if hoping to have your very own personal service robot, you were to steal a robotic arm from a factory in Detroit or Kawasaki and take it home with you, you would quickly find it to be quite useless. Every new task—such as opening a pickle jar—would require days or weeks of programming (assuming you are already an experienced robot programmer), and the slightest change in object shape, material, or even a change in the lighting in the room could mean the difference between finally getting to eat that elusive pickle or spending an afternoon cleaning up broken glass and pickle juice.

This is not to say that robots haven’t made impressive strides towards operation outside the factory. Advances in robotic motion control and sensing capabilities have led to cars that can drive themselves [60], acrobatic helicopters [1], and robots that can walk on rough outdoor terrain [71]. However, generic manipulation capabilities have lagged behind, despite the fact that robot hands are more precise and steady than human hands.

This is partly due to the fact that developing skin to give robots a sense of touch has proven very difficult\(^1\). But one might argue the biggest reason for robots’ “but-
terfingers” is simply that they lack the ability to adequately perceive and understand the physical world.

What is necessary goes beyond simply detecting “what and where” objects are in an image; it involves understanding relationships between objects, like “on-top-of,” “contains,” and “attached.” It involves estimating object attributes, like “slippery” or “sticky.” And most importantly, it involves making predictions about how the objects will move: predictions that depend on those attributes and relationships, and the actions the robot takes. A truly comprehensive model of the physical world (and the ability to perceive and track its state) will significantly improve robots’ ability to manipulate objects because it will allow them to accurately predict the results of their actions (including what might go wrong), and quickly react to disturbances. Furthermore, such a model will form the basis for skill transfer, since actions on similar objects are likely to have similar effects.

1.1 Background

There are two trends that have driven much of the recent progress in robotics: the availability of 3-D sensors, and the development of probabilistic methods.

Over the past few years, the field of robotic computer vision has undergone a 3-D revolution. Cheap 3-D sensors like the Microsoft Kinect have shifted the focus away from image processing and toward reasoning about the 3-D geometry of the world. Even within the “pure” computer vision work that deals with only 2-D images, there has been an increased focus on using 3-D geometric reasoning to explain the relative placement of objects in an image [88, 27].

One of the biggest challenges in dealing with 3-D geometry lies in appropriately handling 3-D rotational data. To specify “where” an object is in space, one must provide both a position and an orientation for the object. There are many ways to specify a 3-D orientation mathematically: as a set of three Euler angles, as a 3x3 rotation matrix, as an axis of rotation and an angle, or as a unit quaternion, which for most tasks.
combines the axis and angle into a single 4-D vector:

\[ q = (\cos(\theta/2), u \sin(\theta/2)). \]  \hspace{1cm} (1.1)

For topological reasons, the unit quaternion representation is often the best. Mathematically, \( q \) can be viewed as a point on the unit hypersphere in four dimensions, \( S^3 \), with a special algebra that enables composition of rotations to be carried out by quaternion multiplication. Because unit quaternions lie on a hypersphere, they lack the singularites associated with Euler angles. Furthermore, they have only one redundant parameter, while rotation matrices have six. The only potential pitfall with unit quaternions is that they aren't unique: \( q \) and \( -q \) represent the same 3-D rotation, since rotating by \( \theta \) about the axis \( u \) is the same as rotating by \( -\theta \) about \( -u \).

Returning to the second trend, much of the recent success in robotics, computer vision, and machine learning has built upon probabilistic methods like Kalman filters, probabilistic roadmaps, and Bayes nets. Yet when work on this thesis began, there were few tools for incorporating 3-D rotations into these probabilistic frameworks. Many methods used local linear approximations to 3-D rotation space, which only worked well when orientation uncertainty was very small.

Fortunately, there is a probability model—the Bingham distribution—that is perfectly suited to modeling uncertainty (large or small) on 3-D rotations when they are represented as unit quaternions. The Bingham distribution is an antipodally-symmetric probability distribution on a unit hypersphere \( S^d \) in any number of dimensions. It is very flexible, and can be used to represent many types of uncertainty, from highly peaked distributions to distributions which are symmetric about some axis, and even the uniform distribution. As we will see in chapter 2, the mathematical form of the Bingham is very similar to a Gaussian distribution, which makes it easy to perform many types of inference. We refer to the Bingham on the space of unit quaternions \( S^3 \) as the quaternion Bingham distribution. Much of the technical part of this thesis is devoted to exploring the quaternion Bingham distribution in depth, although many of the tools we develop are applicable to Bingham in other
1.2 Applications

The goal of this thesis is to explore tools that robots will need to robustly perform complex manipulations. While much of the thesis is devoted to 3-D computer vision applications using directional probability models (based on the Bingham distribution), we also spend a chapter on one specific dynamic manipulation task: robot ping pong. By leveraging the tools we develop in chapter 2 for processing 3-D rotational information, our robot is able to use high-speed cameras to track the spin on ping pong balls. To get the robot to actually play ping pong, we developed a learning system to allow the robot to learn how to hit the ball with experience. One novel feature of the learning system is that it incorporates human advice such as “hit the ball harder” or “angle your paddle up more” which greatly speeds up the rate at which it learns. As a result, the robot was able to learn how to successfully return a variety of topspin and underspin balls at various speeds and locations on the table—a feat that takes many human learners months to master.

The core application of this thesis, however, is the task of object detection in cluttered RGB-D images. While our scope is currently limited to detecting known object instances rather than generalizing over object types, the algorithms and theoretical models developed in this work are foundational, and are likely to be beneficial for more general object detection problems as well. Some of the possible extensions to the methods in this thesis are discussed in chapter 8.

1.3 Contributions

The main contributions of this work are:

- An open-source software library, libbingham, that provides tools in C and Matlab for using the Bingham in practice
• Several formulas of use for inference with the Bingham distribution (of any dimension), such as entropy and KL divergence

• Formulas and theoretical results for the quaternion Bingham distribution; of particular significance is its connection to the least-squares alignment problem

• A method for alignment of oriented point sets called Bingham Procrustean Alignment (BPA)

• An object detection system that leverages BPA to obtain state-of-the-art recognition rates in cluttered RGB-D images

• A recursive filter for 3-D rotations called the Quaternion Bingham Filter (QBF)

• A visual system for tracking the orientation and spin on a ping pong ball

• An algorithm for incorporating exploration-bias advice from human domain experts called Exploration Bias with Directional Advice (EBDA)

• A ping-pong-playing robot that learns to improve its swings with human advice

The majority of these contributions are built upon the theoretical work on the Bingham distribution in the first chapters of this thesis. The most significant practical contribution of this thesis is likely to be the alignment algorithm, BPA—which enables better alignment of 3-D models in cluttered scenes. The work on advice-taking (EBDA) may also lead to important future research directions, but the experiments on ping pong in this work are still preliminary in nature.

1.4 Thesis Organization

We begin in chapter 2 with a tutorial on the Bingham distribution, followed by new theoretical results and formulas for Bingham distributions. In chapter 3 we focus on the quaternion Bingham distribution. Chapter 4 uses these theoretical results to derive recursive filters for 3-D orientation data based on the quaternion Bingham distribution. We present our object detection system in chapters 5 and 6, while chapter 7 covers the
ping pong robot application and advice-taking. We conclude in chapter 8 with a
discussion of how the work in this thesis might be applied to other applications,
along with directions for improvement and future work. Background and related
work are discussed in each chapter.
Chapter 2

The Bingham Distribution

Many geometric quantities have a natural representation as a unit vector on a hypersphere. For example, an angle can be thought of as a point on a circle, a direction in 3-D as a point on a sphere, a 3-D rotation as a unit quaternion (which is a 4-D unit vector on the hypersphere $S^3$), and the “shape” (i.e. all the information that is left when you remove position, scale, and orientation) of a sequence of $N$ points in 2 or 3 dimensions as a point on the hypersphere $S^{2N-1}$ or $S^{3N-1}$, respectively.

Since angles, rotations, and shapes are plentiful in physical systems, there is a need to perform probabilistic inference on hyperspheres when there is uncertainty in the system. However, many existing algorithms that reason about directional statistics ignore the topology of hyperspheres, and instead use a Gaussian noise model in either the embedding space (e.g., $\mathbb{R}^3$ for the sphere $S^2$) or in a local, linear, tangent space to the hypersphere. These methods work well when the errors are small, but as the variance grows, so too does the error of the linear approximation.

Many directional quantities have a further property, called antipodal symmetry, which means that their sign doesn’t matter: $x$ and $-x$ represent the same object. Lines, unit quaternions (representing 3-D orientations), and shapes all share this property. For many directional statistics problems with this property, the Bingham distribution turns out to be a perfect match. The Bingham is a member of the exponential family, and is the maximum entropy distribution on the hypersphere which matches the sample second moment matrix. This means that the distribution...
Figure 2-1: The Bingham distribution (in 2-D) is formed by taking a zero-mean Gaussian distribution (with equiprobable ellipses shown in (a) and (b)) in $\mathbb{R}^2$ and intersecting it with the unit circle, $S^1$. The resulting Bingham distribution (whose PDF is represented by the intensity of pixels on the circle in (b) and (c)) has non-zero probability density only on the surface of $S^1$. It is also antipodally symmetric (i.e., $x$ and $-x$ have the same probability density), since the original Gaussian was centered on the origin.

is flexible enough to match the second moments of any (antipodally-symmetric) set of points on a hypersphere, while making the fewest additional assumptions about the form of the distribution as possible.

2.1 Definition and Properties

The Bingham distribution is an antipodally-symmetric probability distribution on a unit hypersphere. It is derived from a zero-mean Gaussian on $\mathbb{R}^{d+1}$, conditioned to lie on the surface of the unit hypersphere $S^d$. Thus, the Bingham's probability density function (PDF) is proportional to the PDF of a zero-mean Gaussian, but it differs from the Gaussian in its normalization constant—the Bingham PDF must integrate to one over the surface of the hypersphere $S^d$, while the Gaussian’s domain is all of $\mathbb{R}^{d+1}$. Figure 2-1 illustrates the origin of the Bingham distribution in 2-D. It is important to understand that the same Bingham distribution can result from an infinite set of original Gaussians on $\mathbb{R}^{d+1}$ (Figure 2-2). This is because one loses a degree of freedom in going from $\mathbb{R}^{d+1}$ to $S^d$—intuitively, by scaling a Gaussian in the
right way, its intersection with the unit hypersphere $S^d$ can remain the same (up to a constant multiplicative factor).

Figure 2-2: The same Bingham distribution on $S^1$ resulting from three Gaussians on $\mathbb{R}^2$ of different scales.

The PDF of a Bingham distribution on $S^d$ is

$$f(x; \Lambda, V) = \frac{1}{F(\Lambda)} \exp\{\sum_{i=1}^{d} \lambda_i (v_i^T x)^2\}$$

$$= \frac{1}{F(\Lambda)} \exp\{x^T \Lambda^T \Lambda x\}$$

where $x$ is a unit vector on the surface of the sphere $S^d \subset \mathbb{R}^{d+1}$, $\Lambda$ is a $d \times d$ diagonal matrix of concentration parameters $\lambda_1, \ldots, \lambda_d$, the columns of the $(d + 1) \times d$ matrix $V$ are orthogonal unit vectors, and $F$ is a normalization constant that depends only on $\Lambda$. We refer to the density in equation 2.1 as the standard form for the Bingham.

Any reader who is familiar with the multivariate Gaussian distribution will recognize the exponent of equation 2.2 as the exponent of the Gaussian in information form, where the information (inverse covariance) matrix has been decomposed into its eigenvector and eigenvalue matrices, $V$ and $\Lambda$, and where the Bingham exponent is missing a factor of $-\frac{1}{2}$, which is moved into the Bingham’s normalization constant. Also missing are the $(d + 1)$’st eigenvector and eigenvalue, $v_{d+1}$ and $\lambda_{d+1}$. This is because one can add a constant, $\epsilon$, to all of the original $(d + 1)$ $\lambda$’s and obtain the same PDF (up to a proportionality constant, which will be compensated for in the normalization term, $F$):

$$e^{x^T V(\Lambda + \epsilon) V^T x} = e^{x^T V \Lambda V^T x + \epsilon x^T V \epsilon x} = e^{x^T V \Lambda V^T x + \epsilon^T x} = e^{x^T V \Lambda V^T x} \cdot e^\epsilon$$

(2.3)
Figure 2-3: A Bingham distribution on $S^2$, with concentration parameters $\lambda_1 = -41.7$ and $\lambda_2 = -2.6$. Shown are the mode, $\mu$, and the two orthogonal direction vectors, $v_1$ and $v_2$.

since $V$ is orthogonal and $x$ is a unit vector. Therefore, by setting $\epsilon = -\lambda_i$ for any $i$, one can remove one of the $\lambda$'s and $v$'s without changing the distribution.

By convention, one typically subtracts off the highest-valued $\lambda$ and reorders the columns of $A$ and $V$ so that $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_d \leq 0$. That way, all $d$ concentration parameters will have the same sign (negative or zero), making some of the computations we will need to do easier\footnote{Easier still would have been to add a minus-sign to the Bingham’s exponent so that all of the concentration parameters are positive or zero, but alas, history has already spoken.}.

Let us now try to gain some intuition for how the parameters $A$ and $V$ determine the shape of the Bingham distribution. By examining equation 2.1, it is easy to see that the Bingham distribution achieves its highest density when $x \cdot v_i = 0$ for all $i = 1 \ldots d$. Thus, $v_{d+1}$ (the eigenvector of the original Gaussian’s information matrix that was thrown away in order to get the Bingham into standard form) is always a mode of the Bingham distribution. Letting $\mu = v_{d+1}$, the Bingham’s antipodal symmetry means that $-\mu$ is also a mode. (The quadratic form of the Bingham exponent makes it clear that $f(x) = f(-x)$ for any $x$.)

Another thing we can see from equation 2.1 is that the larger (in magnitude) any given $\lambda_i$ becomes, the higher the penalty becomes for deviations of $x$ away from $\mu$ towards $v_i$. This is precisely why the $\lambda$'s are called concentration parameters—they

25
Figure 2-4: The progression of $\lambda_2$ towards zero.

indicate how concentrated the distribution is along the $v_1$ axis, and are analogous to the negative inverse of the variance ($\sigma^2$) parameters of a Gaussian distribution.

In figure 2-3, we see our first example of a Bingham distribution in 3-D (on the sphere $S^2$). In this example, $\lambda_1 < \lambda_2 < 0$, and so the distribution is more spread out in the direction of $v_2$ than it is along $v_1$.

Unlike the Gaussian distribution (which has one mode), we have already seen that the Bingham distribution has at least two modes, $\mu$ and $-\mu$. In fact, the Bingham can also have infinitely many modes! If one or more of the $d$ $\lambda_i$'s in equation 2.1 is zero, then any $x \in S^d$ in the subspace spanned by $\mu$ and the $v_i$'s with $\lambda_i = 0$ will also be a mode of the distribution. As an extreme example, if all of the $\lambda_i$'s are zero, then every $x \in S^d$ is equally likely, which means that the distribution is uniform. If only some of the $\lambda_i$'s are zero, then the distribution will have rings of equal probability around the hypersphere, as shown in figure 2-4. Some more examples of Bingham on $S^2$ are shown in figure 2-5.

This capability of the Bingham distribution to represent distributions that wrap around hyperspheres, all the way to the uniform distribution, is one of the things that makes the Bingham so unique and powerful as a tool for representing information about 3-D orientations. But we'll talk more about this later.

First, we will discuss the mechanics of using the Bingham distribution of any dimension. We will start with the basics—computing the normalization constant, and maximum likelihood fitting of a Bingham to a set of points on $S^d$. Then, we will present some results which are new to this thesis: closed-form solutions for the covari-
ance matrix and higher moments of the Bingham, entropy and KL-divergence, and algorithms for efficient sampling from the Bingham and for computing the product of two or more Bingham PDFs.

2.2 The Normalization Constant

The primary difficulty with using the Bingham distribution in practice lies in computing the normalization constant, $F$. Since the distribution must integrate to one over its domain ($S^d$), we can write the normalization constant as

$$F(\Lambda) = \int_{x \in S^d} \exp\left\{ \sum_{i=1}^d \lambda_i (v_i^T x)^2 \right\}$$

In general, there is no closed form for this integral, which means that $F$ must be solved for numerically.

In his thesis, Bingham [10] derived a series expansion for $F(\Lambda)$ for the special case of $d = 2$, and showed that the expansion is proportional to $\text{}_1F_1(\cdot)$—a hyper-geometric
function of matrix argument$^2$:

$$ F(\Lambda) = 4\pi_1 F_1\left(\frac{1}{2}; \frac{3}{2}; \Lambda\right) $$

Later work [41] on the Bingham distribution makes use of a general formula for $F(\Lambda)$ in any dimension, which we now state as a Theorem. To our knowledge, the proof of this general formula has never been published. Therefore, for completeness, we derive it here.

**Theorem 2.1.** The normalization constant of a Bingham distribution on $\mathbb{S}^d$ with concentration parameters $\Lambda$ is given by

$$ F(\Lambda) = |\mathbb{S}^d| \cdot F_1\left(\frac{1}{2}; \frac{d+1}{2}; \Lambda\right), $$

where $|\mathbb{S}^d|$ is the surface area of the hypersphere $\mathbb{S}^d$.

**Proof.** Our proof follows the same basic pattern as Bingham's derivation of $F(\Lambda)$ for $d = 2$. First, we switch from cartesian to polar coordinates:

$$ v_1^T x = \cos \theta_1 $$
$$ v_2^T x = \sin \theta_1 \cos \theta_2 $$
$$ \vdots $$
$$ v_d^T x = \sin \theta_1 \cdots \sin \theta_{d-1} \cos \theta_d $$

Thus,

$$ \sum_{i=1}^{d} \lambda_i (v_i^T x)^2 = \lambda_1 \cos^2 \theta_1 + \lambda_2 \sin^2 \theta_1 \cos^2 \theta_2 + \cdots + \lambda_d \sin^2 \theta_1 \cdots \sin^2 \theta_{d-1} \cos^2 \theta_d. $$

Next, using the famous Taylor series expansion $e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$ and grouping like terms,

$$ \exp\left\{ \sum_{i=1}^{d} \lambda_i (v_i^T x)^2 \right\} = \sum_{n_1=0}^{\infty} \cdots \sum_{n_d=0}^{\infty} \frac{\lambda_1^{n_1} \cdots \lambda_d^{n_d}}{n_1! \cdots n_d!} \cdot (\cos \theta_1)^{2n_1} \cdots (\cos \theta_d)^{2n_d} \cdot (\sin \theta_1)^{2(n_2+\cdots+n_d)} \cdots (\sin \theta_{d-1})^{2n_d}. $$

$^2$Actually, in Bingham’s original work [11] he used the convention that the probability distribution integrate to $4\pi$ over the unit sphere, so that $F(\Lambda) = \frac{1}{4} F_1\left(\frac{1}{2}; \frac{3}{2}; \Lambda\right)$. However, in this thesis we will follow the standard convention in modern probability theory that PDFs integrate to unity.
Now we can switch the integral in equation 2.6 to polar coordinates and plug in our expansion for $e^{\sum_{i=1}^{d} \lambda_i (v_i \cdot x)^2}$:

$$F(\Lambda) = \int_0^\pi \cdots \int_0^\pi \int_0^{2\pi} e^{\sum_{i=1}^{d} \lambda_i (v_i \cdot x)^2} (\sin \theta_1)^{d-1} (\sin \theta_2)^{d-2} \cdots (\sin \theta_{d-1}) d\theta_d \cdots d\theta_1$$

$$= \sum_{n_1=0}^{\infty} \cdots \sum_{n_d=0}^{\infty} \left\{ \frac{\lambda_1^{n_1} \cdots \lambda_d^{n_d}}{n_1! \cdots n_d!} \cdot \int_0^{2\pi} (\cos \theta_d)^{2n_d} d\theta_d \right\} \cdot \left( \prod_{i=1}^{d-1} \int_0^\pi (\cos \theta_i)^{2n_i} (\sin \theta_i)^{d-i+2(n_{i+1} + \cdots + n_d)} d\theta_i \right).$$

To tackle these polar integrals, we rely on the following two identities for the Beta function:

$$B(x, y) = 2 \int_0^{\pi/2} (\sin \theta)^{2x-1} (\cos \theta)^{2y-1} d\theta$$

and

$$B(x, y) = \frac{\Gamma(x) \Gamma(y)}{\Gamma(x + y)}.$$

By the symmetry of $\cos^2(\cdot)$,

$$\int_0^{2\pi} (\cos \theta_d)^{2n_d} d\theta_d = 4 \int_0^{\pi/2} (\cos \theta_d)^{2n_d} d\theta_d$$

$$= 2 \cdot B\left(\frac{1}{2}, n_d + \frac{1}{2}\right)$$

$$= 2 \cdot \frac{\Gamma\left(\frac{1}{2}\right) \Gamma\left(n_d + \frac{1}{2}\right)}{\Gamma\left(n_d + 1\right)}.$$

Similarly, by the symmetry of $\sin(\cdot)$ (and $\cos^2(\cdot)$),

$$\int_0^\pi (\cos \theta_i)^{2n_i} (\sin \theta_i)^{d-i+2(n_{i+1} + \cdots + n_d)} d\theta_i = 2 \int_0^{\pi/2} (\cos \theta_i)^{2n_i} (\sin \theta_i)^{d-i+2(n_{i+1} + \cdots + n_d)} d\theta_i$$

$$= B(n_{i+1} + \cdots + n_d + \frac{d-i+1}{2}, n_i + \frac{1}{2})$$

$$= \frac{\Gamma(n_{i+1} + \cdots + n_d + \frac{d-i+1}{2}) \Gamma(n_i + \frac{1}{2})}{\Gamma(n_i + \cdots + n_d + 1 + \frac{d-i}{2})}. $$
which yields

\[
\prod_{i=1}^{d-1} \int_0^\pi (\cos \theta_i)^{2n_i} (\sin \theta_i)^{d-i+2(n_i+\cdots+n_d)} d\theta_i
\]

\[
= \prod_{i=1}^{d-1} B(n_{i+1} + \cdots + n_d + \frac{d-i+1}{2}, n_i + \frac{1}{2})
\]

\[
= \frac{\Gamma(n_2 + \cdots + n_d + \frac{d}{2})\Gamma(n_1 + \frac{1}{2})}{\Gamma(n_1 + \cdots + n_d + \frac{d-1}{2}) \Gamma(n_2 + \cdots + n_d + \frac{d}{2})} \frac{\Gamma(n_3 + \cdots + n_d + \frac{d-1}{2})\Gamma(n_2 + \frac{1}{2})}{\Gamma(n_2 + \cdots + n_d + \frac{d}{2})} \cdots \frac{\Gamma(n_{d+1}) \Gamma(n_{d-1} + \frac{1}{2})}{\Gamma(n_{d-1} + \cdots + n_d + \frac{d+1}{2})}
\]

\[
= \frac{\Gamma(n_d + 1)\Gamma(n_1 + \frac{1}{2} + \cdots + \Gamma(n_{d-1} + \frac{1}{2})}{\Gamma(n_1 + \cdots + n_d + \frac{d+1}{2})}
\]

Therefore,

\[
F(\Lambda) = \sum_{n_1=0}^\infty \cdots \sum_{n_d=0}^\infty \frac{\lambda_1^{n_1} \cdots \lambda_d^{n_d}}{n_1! \cdots n_d!} \cdot \frac{1}{2} \cdot \frac{\Gamma\left(\frac{1}{2}\right)\Gamma(n_d + \frac{1}{2})}{\Gamma(n_d + 1)} \cdot \frac{\Gamma\left(\frac{1}{2}\right)\Gamma(n_1 + \frac{1}{2})}{\Gamma(n_1 + n_d + \frac{d+1}{2})} \cdots \frac{\Gamma\left(\frac{1}{2}\right)\Gamma(n_{d-1} + \frac{1}{2})}{\Gamma(n_{d-1} + \cdots + n_d + \frac{d+1}{2})} \cdot \frac{\Gamma\left(\frac{1}{2}\right)\Gamma(n_1 + \frac{1}{2} + \cdots + \Gamma(n_{d-1} + \frac{1}{2})}{\Gamma(n_1 + \cdots + n_d + \frac{d+1}{2})}
\]

(2.7)

Now since \(\Gamma\left(\frac{1}{2}\right) = \sqrt{\pi}\) and \(\Gamma(a + n) = \Gamma(a) \cdot (a)_n\) where \((a)_n\) is the rising factorial \(a(a + 1) \cdots (a + n - 1)\),

\[
F(\Lambda) = \frac{2\pi^{\frac{d+1}{2}}}{\Gamma\left(\frac{d+1}{2}\right)} \cdot \sum_{n_1=0}^\infty \cdots \sum_{n_d=0}^\infty \frac{\lambda_1^{n_1} \cdots \lambda_d^{n_d}}{n_1! \cdots n_d!} \cdot \left(\frac{1}{2}\right)^{n_1} \cdots \left(\frac{1}{2}\right)^{n_d} \cdot \frac{1}{\Gamma\left(\frac{d+1}{2}\right)} \cdot \frac{1}{\Gamma\left(\frac{n_1 + \cdots + n_d + d+1}{2}\right)}
\]

(2.8)

\[
= |S^d| \cdot \frac{}{F_1\left(\frac{1}{2}; \frac{d+1}{2}; \Lambda\right)}
\]

(2.9)

where the infinite series expansion for \(F_1\) above is a special case of \(F_1\left(\frac{1}{2}; b; A\right)\) for symmetric \(A\) with latent roots (eigenvalues) \(\lambda_1, \ldots, \lambda_d\), as proved by Bingham [10].

\[\square\]

### 2.2.1 Computing the Normalization Constant

Many operations on Bingham distributions require computing the normalization constant. Given that we already have an infinite series representation for \(F(\Lambda)\), the most straightforward way to calculate it is via series truncation [11, 43]. However, since each dimension of \(S^d\) adds a new infinite sum to the calculation, this approach
is exponential in $d$, and is therefore limited to low $d$ (e.g., less than 4 or 5). Alternative approaches to calculating the Bingham normalizing constant $F(\Lambda)$ include numerical integration [87], saddle-point approximations [52], and holonomic gradient descent [77].

Since we are primarily interested in Bingham distributions on $S^3$ in this thesis, we will use the direct (series truncation) method. For convenience, we will calculate $\mathbf{1}_F(\frac{1}{2}; \frac{d+1}{2}; \Lambda)$ and then multiply by the surface area of $S^d$ to get $F(\Lambda)$. Starting from equation 2.8,

$$\mathbf{1}_F(\frac{1}{2}; \frac{d+1}{2}; \Lambda) = \sum_{n_1=0}^{\infty} \cdots \sum_{n_d=0}^{\infty} \frac{\lambda_1^{n_1} \cdots \lambda_d^{n_d}}{n_1! \cdots n_d!} \cdot \frac{1}{(\frac{1}{2})^{n_1} \cdots (\frac{1}{2})^{n_d}}$$

(2.10)

However, difficulty arises when we try to calculate $\mathbf{1}_F$ with equation 2.10. Since all the $\lambda$'s are less than or equal to zero, this is an alternating series with slow convergence—a computational nightmare that requires an infinite-precision math library! Fortunately, there is a series transformation$^3$ of $F(\Lambda)$ that makes all the $\lambda_i$'s positive (or zero):

$$F(\lambda_1, \ldots, \lambda_d) = e^{\lambda_1} F(-\lambda_1, \lambda_2 - \lambda_1, \ldots, \lambda_d - \lambda_1).$$

(2.11)

Denoting $z_1 = -\lambda_1$, $z_2 = \lambda_2 - \lambda_1$, $\ldots$, $z_d = \lambda_d - \lambda_1$,

$$F(\Lambda) = e^{\lambda_1} \cdot \mathbf{1}_F(z_1, \ldots, z_d),$$

(2.12)

and so we can compute $\mathbf{1}_F$ with all non-negative arguments. Although this infinite series converges much more quickly than the original due to all of its terms being non-negative, for $d = 3$ it is still a triple infinite sum, which tends to converge very slowly when some of the $z$'s are large in magnitude. Therefore, for the real-time robotics applications in this thesis, we pre-compute a lookup table of $\mathbf{1}_F$-values over a discrete grid of $\Lambda$'s, and use tri-linear interpolation to quickly estimate normalizing constants on the fly. Note that this lookup table approach is only practical for small $d$—in particular, we computed lookup tables in this work for $d = 1, 2, 3$.

$^3$This can be easily proved using the integral form of $F(\Lambda)$ in equation 2.4.
To speed up the normalization constant table calculations, we use the following two recursive formulas. For \( d = 2 \):

\[
_{1}F_{1}(\frac{1}{2}; \frac{3}{2}; z_1, z_2) = \sum_{i=1}^{\infty} \frac{z_1^i}{i! \left( \frac{1}{2} \right)^i} \cdot _{1}F_{1}(\frac{1}{2}; i + \frac{3}{2}; z_2),
\]

and for \( d = 3 \):

\[
_{1}F_{1}(\frac{1}{2}; \frac{4}{2}; z_1, z_2, z_3) = \sum_{i=1}^{\infty} \sum_{j=1}^{\infty} \frac{z_1^{i} z_2^{j}}{i! j! \left( \frac{1}{2} \right)^{i+j}} \cdot _{1}F_{1}(\frac{1}{2}; i+j + \frac{4}{2}; z_3).
\]

That way, by first calculating an intermediate table of \(_{1}F_{1}(\frac{1}{2}; b; z)\)-values which we call the “b-table”, we are able to eliminate one of the infinite sums from both the \( d = 2 \) and \( d = 3 \) table calculations.

### 2.3 Maximum Likelihood Parameter Estimation

Given a \( d + 1 \)-by-\( N \) sample matrix \( X = [\mathbf{x}_1 \cdots \mathbf{x}_N] \) containing \( N \) unit vectors in the columns which we assume to be generated i.i.d. from a Bingham distribution on \( S^d \), we wish to estimate the parameters \((\Lambda, \mathbf{v})\) of the distribution that generated them. There are two main ways of doing this: (i) with maximum likelihood estimation (MLE), and (ii) with a Bayesian approach. We will first cover the maximum likelihood method. Note that this section is adapted from Bingham [11], where we have modified Bingham’s theorems to account for the standard form conventions in this thesis (where \( \lambda_{d+1} \) and \( \mathbf{v}_{d+1} \) are dropped).

Since we assume that \( \mathbf{x}_1, \ldots, \mathbf{x}_N \) are i.i.d., their joint log-likelihood under a Bingham with parameters \((\Lambda, \mathbf{v})\) is

\[
L(\Lambda, \mathbf{v}; X) := \sum_{i=1}^{N} \ln f(\mathbf{x}_i; \Lambda, \mathbf{v}).
\]

The maximum likelihood approach is to pick the \( \Lambda \) and \( \mathbf{v} \) that maximize \( L(\Lambda, \mathbf{v}; X) \). Continuing,

\[
L(\Lambda, \mathbf{v}; X) = \sum_{i=1}^{N} (\mathbf{x}_i^T \Lambda \mathbf{v} \mathbf{x}_i - \ln F(\Lambda))
\]

\[
= tr[X^T \Lambda \mathbf{v} X] - N \ln F(\Lambda)
\]

\[
= tr[\Lambda \mathbf{v} X X^T \Lambda] - N \ln F(\Lambda)
\]
by the cyclic property of trace. Before we can derive the maximum likelihood estimates for \( \Lambda \) and \( V \), we will need the following three lemmas.

**Lemma 2.2.** \(-L(\Lambda, V; X)\) is strictly convex in \( \Lambda \).

**Proof.** We will start by showing that \( \ln F(\Lambda) \) is strictly convex. The Lemma will then follow because the sum of a strictly convex function with a linear function \((-tr[AV^TXX^TV])\) is strictly convex. The Hessian of \( \ln F(\Lambda) \) with respect to \( \Lambda \) is

\[
H = \frac{\nabla^2 F}{F} - \frac{(\nabla F)(\nabla F)^T}{F^2}. \tag{2.19}
\]

where \( \nabla F \) is the gradient of \( F(\Lambda) \) with respect to \( \Lambda \), and \( \nabla^2 F \) is its matrix of second derivatives. From the integral form of \( F(\Lambda) \) in equation 2.4, one can see that

\[
\frac{\partial F}{\partial \lambda_i} = \frac{1}{F} \int_{x \in \mathbb{R}^d} (v_i^T x)^2 \exp\{\sum_{k=1}^{d} \lambda_k (v_k^T x)^2\} \tag{2.20}
\]

\[
= E[(v_i^T x)^2] \tag{2.21}
\]

and

\[
\frac{\partial^2 F}{\partial \lambda_i \partial \lambda_j} = \frac{1}{F} \int_{x \in \mathbb{R}^d} (v_i^T x)(v_j^T x)^2 \exp\{\sum_{k=1}^{d} \lambda_k (v_k^T x)^2\} \tag{2.22}
\]

\[
= E[(v_i^T x)^2(v_j^T x)^2]. \tag{2.23}
\]

Thus, letting \( y = [(v_1^T x)^2 \cdots (v_d^T x)^2]^T \),

\[
H = \frac{\nabla^2 F}{F} - \frac{(\nabla F)(\nabla F)^T}{F^2} \tag{2.24}
\]

\[
= E[yy^T] - E[y]E[y]^T \tag{2.25}
\]

\[
= \text{Cov}[y] \tag{2.26}
\]

which is positive semidefinite by the properties of covariance matrices. Since \( f(x; \Lambda, V) > 0 \) for all \( x \in \mathbb{R}^d \) (for finite \( \Lambda \)), the random variable \( y \) can achieve any point in the \( d \)-dimensional volume \{\( y \mid \forall i, y_i \geq 0, \sum_{i=1}^{d} y_i \leq 1 \}\), which means that \( H = \text{Cov}[y] \) has full rank, and is therefore positive definite. It follows that \( \ln F(\Lambda) \) is strictly convex.
Lemma 2.3 (Von Neumann’s trace inequality). Given square $n$-by-$n$ matrices $A$ and $B$ with singular values $\alpha_1 \leq \cdots \leq \alpha_n$ and $\beta_1 \leq \cdots \leq \beta_n$. Then

$$|\text{tr}AB| \leq \sum_{i=1}^{n} \alpha_i \beta_i$$  \hspace{1cm} (2.27)

Proof. See Mirsky [62].

Lemma 2.4 (Bingham [11], Lemma 6.1). For any $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_{d+1})$ with $\lambda_1 \leq \cdots \leq \lambda_{d+1}$, $\text{tr}[\Lambda V^TXX^TV]$ is maximized (subject to the constraint that $V^TV = VV^T = I$) by $V = W = [w_1 \cdots w_{d+1}]$, where $w_i$ is the eigenvector of $XX^T$ corresponding to eigenvalue $\sigma_i$ with $\sigma_1 \leq \cdots \leq \sigma_{d+1}$.

Proof. When $V = W$, $\text{tr}[\Lambda W^TXX^TW] = \text{tr}[\Lambda W^TW\Sigma] = \text{tr}[\Lambda \Sigma] = \sum_{i=1}^{d+1} \lambda_i \sigma_i$. For arbitrary $V$, $\text{tr}[\Lambda V^TXX^TV] \leq |\text{tr}[\Lambda V^TXX^TV]| \leq \sum_{i=1}^{d+1} \lambda_i \sigma_i$ by von Neumann’s trace inequality, since multiplying $XX^T$ by orthogonal $V$ and $V^T$ doesn’t change its singular values. Thus, $V = W$ achieves a global maximum of $\text{tr}[\Lambda V^TXX^TV]$, subject to the constraint that $V$ be orthogonal.

We can now state the main theorem about the maximum likelihood estimates for $\Lambda$ and $V$.

Theorem 2.5. If the following conditions hold:

(a) $V = [v_1 \cdots v_d]$ contains the first $d$ eigenvectors of $XX^T$, corresponding to eigenvalues $\sigma_1 \leq \cdots \leq \sigma_d \leq \sigma_{d+1}$

(b) $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_d)$ satisfies

$$\frac{\partial F(\Lambda)/\partial \lambda_i}{F(\Lambda)} = \frac{\sigma_i}{N}$$  \hspace{1cm} (2.28)

for all $i = 1, \ldots, d$

then $\Lambda$ and $V$ are maximum likelihood estimates for the parameters of the Bingham distribution that generated the i.i.d. sample matrix $X$. Furthermore, $\Lambda$ is always the unique MLE, while $V$ is only unique if $XX^T$ has no repeated eigenvalues.
Proof. Assume without loss of generality that \( \lambda_1 \leq \cdots \leq \lambda_d \leq 0 \). Then letting \( N' = \text{diag}(\lambda_1, \ldots, \lambda_d, 0) \), Lemma 2.4 tells us that \( \text{tr}[N'V'^TXX^TV'] \) achieves its maximum when \( V' \) is the (full) matrix of \( XX^T \)'s eigenvectors, ordered by eigenvalues \( \sigma_1 \leq \cdots \leq \sigma_{d+1} \). Since \( \lambda_{d+1} = 0 \), we can drop the last column of \( V' \) which means that the \( V \) given in (a) maximizes \( \text{tr}[AV^TXX^TV] \) and therefore also maximizes \( L(A, V; X) = \text{tr}[AV^TXX^TV] - N\ln F(A) \) with respect to \( V \).

Necessary conditions for \( A \) can be found by setting the partial derivatives of \( L(A, V; X) \) with respect to \( A \) to zero. From equation 2.18,

\[
\frac{\partial L(A, V; X)}{\partial \lambda_i} = 0
\]

\[
\Rightarrow \quad \frac{\partial N \ln F(A)}{\partial \lambda_i} = \frac{\partial \text{tr}[AV^TXX^TV]}{\partial \lambda_i}
\]

\[
\Rightarrow \quad \frac{\partial F(A)/\partial \lambda_i}{F(A)} = \frac{1}{N} v_i^TXX^Tv_i = \frac{\sigma_i}{N},
\]

for \( i = 1, \ldots, d \). By the convexity of \( -L(A, V; X) \)—which we proved in Lemma 2.2—\( -A \) is the unique MLE.

\[
\Box
\]

### 2.3.1 Computing the Maximum Likelihood Estimates

The MLE \( V \) can quickly be found by computing the eigenvectors of \( X \). But solving equation 2.28 for \( A \) requires a bit more effort.

Just as we did for \( F(A) \), we can pre-compute values of the gradient of \( F \) with respect to \( A, V, \nabla F \), and store them in a lookup table. Using a kD-tree, we can find the nearest neighbors of a new sample \( \nabla F/F = \frac{1}{N} [\sigma_1 \ldots \sigma_d]^T \) in \( O(d \log M) \) time (where \( M^d \) is the size of the lookup table), and use their indices to find an initial estimate of the MLE \( A \) via interpolation (since the lookup tables for \( F \) and \( \nabla F \) are indexed by \( A \)). This estimate can be further refined using gradient descent on \( -L(A, V; X) \).
2.4 Bayesian Parameter Estimation

The Bayesian approach to parameter estimation for the Bingham is to compute a posterior distribution on parameters $\Lambda, V$ given a data set $X$:

$$p(\Lambda, V | X) \propto p(X | \Lambda, V)p(\Lambda, V).$$

It is desirable for the prior distribution, $p(\Lambda, V)$, to be formed such that the posterior $p(\Lambda, V | X)$ will belong to the same class of probability distribution, $\mathcal{D}$, as the prior. When this occurs, we say that $\mathcal{D}$ is a conjugate prior for the Bingham distribution.

Because the Bingham is in the exponential family, we know that a conjugate prior exists. To find it, let's examine the data likelihood term:

$$p(X | \Lambda, V) = F(\Lambda)^{-N} e^{\text{etr}[\Lambda V^T XX^T V]} = \left( \frac{e^{\text{etr}[\Lambda V^T SV]}}{F(\Lambda)} \right)^N$$

where the first equation is easily derived by exponentiating both sides of equation 2.18, and where $e^{\text{etr}(\cdot)} = e^{\text{tr}(\cdot)}$ is the exponential trace and $S = \frac{1}{N} XX^T$ is the scatter matrix of $X$. This suggests the following distributional form for $p(\Lambda, V)$, which (as it is easy to check) is a conjugate prior for the Bingham:

$$p(\Lambda, V ; S, N) = \frac{1}{\kappa(S, N)} \left( \frac{e^{\text{etr}[\Lambda V^T SV]}}{F(\Lambda)} \right)^N. \quad (2.29)$$

The distribution has two parameters: a real, symmetric, positive definite matrix $S$ with $\text{tr}(S) = 1$, and a positive real number, $N$. $\kappa(S, N)$ is a normalization constant to ensure that the distribution integrates to one.

Given a prior on $\Lambda, V$ with parameters $S_0, N_0$, the posterior is

$$p(\Lambda, V | X) \propto p(X | \Lambda, V)p(\Lambda, V; S_0, N_0) = p(\Lambda, V; \frac{N S + N_0 S_0}{N + N_0}, N + N_0).$$

Thus, the prior parameters $S_0, N_0$ act as a prior on the scatter matrix along with a weight, or prior data count, $N_0$. A common procedure for choosing these prior parameters is to write down the parameters $\Lambda_0, V_0$ of a prior Bingham distribution, then use the formula for the second moments (scatter) matrix of a Bingham from equation 2.33 below to find $S_0$, and then to choose how much weight (in terms of a fictitious count $N_0$) to give the prior scatter matrix.
Oftentimes we are not interested in the full posterior distribution on $\Lambda, V$, but only on their *maximum a posteriori* (MAP) values:

\[
(\Lambda_{MAP}, V_{MAP}) = \arg \max_{\Lambda,V} p(\Lambda, V|X),
\]

which reduces to maximum likelihood estimation on the posterior scatter matrix $S' = \frac{NS + N_0 S_0}{N + N_0}$ when the conjugate prior is used. (Thus, the same lookup-table approach to computing MLE values based on the eigen-decomposition of scatter matrix may be applied to computing MAP parameter estimates.)

### 2.5 Characterization

Notice that the maximum likelihood estimates for $V$ and $\Lambda$ are both computed given only the sample scatter matrix, $\hat{S} = \frac{1}{n}XX^T = E_X[xx']$. Thus, the scatter matrix is a sufficient statistic for the Bingham distribution. In fact, there is a beautiful result from the theory of exponential families which says that the Bingham distribution is the *maximum entropy* distribution on the hypersphere which matches a given second moments (scatter) matrix [58]. This gives us further theoretical justification to use the Bingham distribution in practice, if we assume that all of the relevant information about the data is captured in the matrix of second moments.

In chapter 3, we will see another characterization of the Bingham (for the special case of $d = 3$), resulting from a deep connection between the Bingham and the problem of least-squares alignment of point sets.

### 2.6 Sampling

Because of the complexity of the normalization constant, there is no known way to sample directly from the Bingham distribution. Therefore, we use a Metropolis-Hastings sampler, with target distribution given by the Bingham density, and proposal distribution given by the projected zero-mean Gaussian$^4$ in $\mathbb{R}^{d+1}$ with covariance ma-

---

$^4$To sample from the projected Gaussian, we first sample from a Gaussian with covariance $S$, then project the sample onto the unit sphere.
trix equal to the Bingham's sample scatter matrix, $S$. Empirically, the proposal distribution is often very similar to the target distribution. Thus, the sampling distribution from the Metropolis-Hastings sampler is typically a good approximation to the true Bingham sampling distribution after just a few iterations.

### 2.7 Mode

As was discussed in section 2.1 and illustrated in figure 2-3, the unit vector $\mu$, which is orthogonal to $v_1, \ldots, v_d$, is always a mode of the Bingham. However, it is not unique. When all concentration parameters $\lambda_i$ are non-zero, the Bingham has two modes: $\mu$ and $-\mu$. But when one or more of the $\lambda_i$'s is zero, there are infinitely many modes, corresponding to the subspace of unit vectors spanned by $\mu$ and all the $v_i$'s with $\lambda_i = 0$.

### 2.8 Moments

Due to the Bingham's antipodal symmetry, many of its moments are zero.

**Theorem 2.6.** *All the Bingham's odd moments are zero.* That is,

$$E[\prod_{i=1}^{d+1} x_i^{a_i}] = 0 \quad \text{if any } a_i \text{ is odd.} \quad (2.30)$$

**Proof.** Given a Bingham with parameters $\Lambda, V$. Choose $j$ such that $a_j$ is odd. Then

$$E[\prod_{i=1}^{d+1} x_i^{a_i}] = \int_{x \in S^d} x_j^{a_j} \prod_{i \neq j} x_i^{a_i} f(x; \Lambda, V)$$

$$= \int_{x_j > 0} x_j^{a_j} \prod_{i \neq j} x_i^{a_i} f(x; \Lambda, V) + \int_{x_j < 0} x_j^{a_j} \prod_{i \neq j} x_i^{a_i} f(x; \Lambda, V).$$

Now by symmetry of $S^d$, each $x$ with $x_j > 0$ in the first integral has a counterpart $x'$ in the second integral with $x_j$ replaced by $-x_j$ but all the other $x_i$'s unchanged, so
we can combine the integrals to get

\[
E[\prod_{i=1}^{d+1} x_i^{a_i}] = \int_{x_j > 0} x_j^{a_j} \prod_{i \neq j} x_i^{a_i} f(x; \Lambda, V) + (-x_j)^{a_j} \prod_{i \neq j} x_i^{a_i} f(x; \Lambda, V)
\]

\[
= \int_{x_j > 0} (x_j^{a_j} + (-1)^{a_j} x_j^{a_j}) \prod_{i \neq j} x_i^{a_i} f(x; \Lambda, V)
\]

\[
= \int_{x_j > 0} (x_j^{a_j} - x_j^{a_j}) \prod_{i \neq j} x_i^{a_i} f(x; \Lambda, V) = 0
\]

since \(a_j\) is odd.

As a special case of theorem 2.6, note that the mean of the Bingham distribution, \(E[x]\), is always zero. This makes sense, because the Gaussian distribution from which the Bingham is derived has zero mean and is symmetric.

You might recall that the sufficient statistic for the Bingham distribution is its matrix of second moments, \(E[xx^T]\). As with many other things Bingham, there is a simple formula for \(E[xx^T]\) in terms of \(V, F, \) and \(\nabla F\). But before we can derive this formula, we will first need to understand its projected moments.

**Theorem 2.7.** All odd projected moments are zero for the Bingham distribution. That is, letting \(v_{d+1} = \mu\),

\[
E[\prod_{i=1}^{d+1} (v_i^T x)^{a_i}] = 0 \quad \text{if any } a_i \text{ is odd.} \tag{2.31}
\]

**Proof.** As in the previous proof, we will exploit the symmetry of \(S^d\). To begin, choose \(j\) such that \(a_j\) is odd. Then

\[
E[\prod_{i=1}^{d+1} (v_i^T x)^{a_i}] = \int_{x \in S^d} (v_j^T x)^{a_j} \prod_{i \neq j} (v_i^T x)^{a_i} f(x; \Lambda, V)
\]

\[
= \int_{v_j^T x > 0} (v_j^T x)^{a_j} \prod_{i \neq j} (v_i^T x)^{a_i} f(x; \Lambda, V)
\]

\[
+ \int_{v_j^T x < 0} (v_j^T x)^{a_j} \prod_{i \neq j} (v_i^T x)^{a_i} f(x; \Lambda, V).
\]
Now associate each \( x \) in the first integral (with \( v_j^T x > 0 \)) with its reflection along \( v_j \), \( x' = x - 2(v_j^T x)v_j \), which lies in the second integral because \( v_j^T x' = v_j^T x - 2(v_j^T x)v_j = -v_j^T x \) since \( \|v_j\| = 1 \). Furthermore, for \( i \neq j \), \( v_i^T x' = v_i^T x \) since all the \( v \)'s are orthogonal. In other words, all projections of \( x' \) onto other \( v_i \)'s remain the same, while the sign gets flipped on the projection of \( x' \) onto \( v_j \). Thus,

\[
E[\prod_{i=1}^{d+1}(v_i^T x)^{a_i}] = \int_{v_j^T x > 0} ((v_j^T x)^{a_j} + (-v_j^T x)^{a_j}) \prod_{i \neq j}(v_i^T x)^{a_i} f(x; \Lambda, V) \\
= 0
\]
since \( a_j \) is odd.

\[\square\]

**Theorem 2.8.** The Bingham’s even projected moments\(^5\) are given by

\[
E[\prod_{i=1}^{d}(v_i^T x)^{2a_i}] = \frac{1}{F} \prod_{i=1}^{d} \frac{\partial \lambda_{i}^{a_i}}{\partial \lambda_{i}^{a_i}}. \tag{2.32}
\]

**Proof.** This can be easily seen by writing the Bingham normalization constant in integral form:

\[
F(\Lambda) = \int_{x \in S^d} \exp\{\sum_{i=1}^{d} \lambda_i (v_i^T x)^2\}.
\]

Thus,

\[
\frac{1}{F} \prod_{i=1}^{d} \frac{\partial \lambda_{i}^{a_i}}{\partial \lambda_{i}^{a_i}} = \frac{1}{F} \int_{x \in S^d} \prod_{i=1}^{d}(v_i^T x)^{2a_i} \exp\{\sum_{i=1}^{d} \lambda_i (v_i^T x)^2\} \\
= \int_{x \in S^d} \prod_{i=1}^{d}(v_i^T x)^{2a_i} f(x; \Lambda, V) \\
= E[\prod_{i=1}^{d}(v_i^T x)^{2a_i}].
\]

\[\square\]

**Theorem 2.9.** Let \( \Sigma = \text{diag}(\frac{\Sigma F}{F}) \). The Bingham’s second moments matrix is given by

\[
E[xx^T] = V \Sigma V^T + (1 - tr(\Sigma)) \mu \mu^T. \tag{2.33}
\]

\(^5\)excluding \( v_{d+1} = \mu \)
Proof. Let \( W = [V \mu] \). Since \( WW^T = I \),

\[
\]

by linearity of expectation. Now note that the \( ij \)-th entry of the matrix \( W^T xx^T W \) is \((v_i^T x)(v_j^T x)\) if we let \( v_{d+1} = \mu \). By theorem 2.7, \( E[(v_i^T x)(v_j^T x)] = 0 \) if \( i \neq j \), which means that \( E[W^T xx^T W] \) is a diagonal matrix. By theorem 2.8, \( E[(v_i^T x)^2] = \frac{1}{F} \frac{\partial F}{\partial x_i} \) if \( i \leq d \), which just leaves the bottom-right corner of the matrix, \( E[(\mu^T x)^2] \). Let \( y = W^T x = \begin{bmatrix} v_i^T x \\ \mu^T x \end{bmatrix} \). Since \( W^T \) is orthogonal, it preserves lengths. Thus, \( \|y\| = \|x\| = 1 \), so it follows that

\[
g_{d+1}^2 = 1 - \sum_{i=1}^{d} y_i^2
\]

\[
\implies E[g_{d+1}^2] = E[1 - \sum_{i=1}^{d} y_i^2]
\]

\[
\implies E[(\mu^T x)^2] = 1 - \sum_{i=1}^{d} E[(v_i^T x)^2] = 1 - \text{tr}(\Sigma). \tag{2.34}
\]

Thus, the final form of the second moments matrix is \( E[xx^T] = W \begin{bmatrix} Y & 0 \\ 0 & 1 - \text{tr}(\Sigma) \end{bmatrix} W^T \), which yields equation 2.33 when \( W \) is replaced with \( [V \mu] \). \( \square \)

### 2.9 Entropy

**Theorem 2.10.** The *entropy* of a Bingham distribution with PDF \( f \) is given by:

\[
h(f) = \log F - \Lambda \cdot \frac{\nabla F}{F}.
\tag{2.35}
\]

**Proof.**

\[
h(f) = -\int_{x \in S^d} f(x) \log f(x)
\]

\[
= -\int_{x \in S^d} \frac{1}{F} e^{x^T C x} (x^T C x - \log F)
\]

\[
= \log F - \frac{1}{F} \int_{x \in S^d} x^T C x e^{x^T C x}.
\]

41
Writing \( f \) in standard form, and denoting the hyperspherical integral \( \int_{x \in S^d} x^T C x e^{x^T C x} \) by \( g(\Lambda) \),

\[
g(\Lambda) = \int_{x \in S^d} \prod_{i=1}^d \lambda_i (v_i^T x)^2 e^{\sum_{j=1}^d \lambda_j (v_j^T x)^2}
= \sum_{i=1}^d \lambda_i \frac{\partial F}{\partial \lambda_i} = \Lambda \cdot \nabla F.
\]

Thus, the entropy is \( \log F - \Lambda \cdot \frac{\nabla F}{F} \), as claimed.

Since both the normalization constant, \( F \), and its gradient with respect to \( \Lambda, \nabla F \), are stored in a lookup table, the entropy is trivial to compute, and can be used on the fly without any numerical integration over hyperspheres.

### 2.10 Divergence

The Kullback-Leibler (KL) divergence between two Binghams with PDFs \( f \) and \( g \) (from \( f \) to \( g \)) is

\[
D_{KL}(f \| g) = \int_{x \in S^d} f(x) \log \frac{f(x)}{g(x)}
= h(f, g) - h(f),
\]

where \( h(f) \) is the entropy of \( f \) and \( h(f, g) \) is the cross entropy of \( f \) and \( g \).

**Theorem 2.11.** The cross entropy between two Binghams with PDFs \( f \) and \( g \) (from \( f \) to \( g \)) is

\[
h(f, g) = \log F_g - \sum_{i=1}^d \lambda_{g_i} \left( b_i^2 + \sum_{j=1}^d (a_{ij}^2 - b_i^2) \frac{1}{F_f} \frac{\partial F_f}{\partial \lambda_{f_j}} \right),
\]

where the matrix \( A \) of coefficients \( a_{ij} \) is given by \( A = V_f^T V_g \), and \( b = \mu_f^T V_g \) (where \( \mu_f \) is the mode of \( f \)).
Proof.

\[
    h(f, g) = -\int_{x \in \mathbb{S}^d} f(x) \log g(x) \ 
    = -\int_{x \in \mathbb{S}^d} f(x) \cdot \left( \sum_{i=1}^{d} \lambda_{gi} (v_{gi}^T x) - \log F_g \right) \ 
    = \log F_g - \sum_{i=1}^{d} \lambda_{gi} E_f[(v_{gi}^T x)].
\]

Since \( \begin{bmatrix} A \ b^T \end{bmatrix} = \begin{bmatrix} V_f^T \\
\mu_f^T \end{bmatrix} \) and \( \begin{bmatrix} V_f^T \\
\mu_f^T \end{bmatrix} \) is orthogonal, \( V_g = [V_f \ \mu_f] [ A \ b^T ] \), and so \( v_{gi} = V_f a_i + b_i \mu_f \). Thus,

\[
    E_f[(v_{gi}^T x)] = E_f[((V_f a_i + b_i \mu_f)^T x)^2] \ 
    = b_i^2 E_f[(\mu_f^T x)^2] + \sum_{j=1}^{d} a_{ij}^2 E_f[(v_{gi}^T x)^2]
\]

by linearity of expectation, and since all the odd projected moments are zero (theorem 2.7). Then, since \( E_f[(\mu_f^T x)^2] = 1 - \sum_{j=1}^{d} E_f[(v_{gi}^T x)^2] \) (from equation 2.34) and \( E_f[(v_{gi}^T x)^2] = \frac{1}{F_f} \frac{\partial F_f}{\partial \lambda_{fj}} \) (from theorem 2.8),

\[
    h(f, g) = \log F_g - \sum_{i=1}^{d} \lambda_{gi} \left( b_i^2 + \sum_{j=1}^{d} (a_{ij}^2 - b_i^2) \frac{1}{F_f} \frac{\partial F_f}{\partial \lambda_{fj}} \right)
\]

as claimed. \( \square \)

2.11 Product of Bingham PDFs

The product of two Bingham PDFs is given by adding their exponents:

\[
    f(x; A_1, V_1)f(x; A_2, V_2) = \frac{1}{F_1 F_2} \exp\{x^T (\sum_{i=1}^{d} \lambda_{1i} v_{1i} v_{1i}^T + \lambda_{2i} v_{2i} v_{2i}^T) x \} \ 
    = \frac{1}{F_1 F_2} \exp\{x^T (C_1 + C_2) x \} \ 
\]

After computing the sum \( C = C_1 + C_2 \) in the exponent of equation 2.39, we transform the PDF to standard form by computing the eigenvectors and eigenvalues of \( C \), and then subtracting off the lowest magnitude eigenvalue from each spectral component,
so that only the eigenvectors corresponding to the largest \(d\) eigenvalues (in magnitude) are kept, and \(\lambda_1 \leq \cdots \leq \lambda_d \leq 0\) (as in equation 2.1). We then recompute (or look up) the normalization constant of the new distribution as a function of the concentration parameters.

### 2.12 Marginal Distributions

In some cases, it may be useful to compute the marginal distribution of a single component \(x_i\) of a Bingham random variable \(\mathbf{x} \sim \text{Bingham}(\Lambda, V)\). Closed-form marginals also allow for easy computation of conditional distributions, since \(f(\mathbf{x}|x_i) = \frac{f(\mathbf{x})}{f(x_i)}\). The axis-aligned marginals (i.e., \(f(v_i^T \mathbf{x})\), or \(f(x_i)\) when \(V\)'s columns are taken from the identity matrix) of a Bingham distribution have a particularly simple form.

**Theorem 2.12.** Given an axis-aligned Bingham distribution with PDF\(^6\) \(f(\mathbf{x}; \Lambda) = \frac{1}{F(\Lambda)} \exp \sum_{i=1}^{d+1} \lambda_i x_i^2\). Then the marginal distribution of any component \(x_i\) of \(\mathbf{x}\) is

\[
f(x_i) = \frac{|S^{d-1}|}{|S^{d-1}| F_1(\frac{1}{2}; \frac{d+1}{2}; \Lambda)} e^{\lambda_i x_i^2} (1 - x_i^2)^{d/2} F_1(\frac{1}{2}; \frac{d}{2}; (1 - x_i^2)\Lambda_i^{-1})
\]

where \(\Lambda_i^{-1}\) is \(\Lambda\) with \(\lambda_i\) removed.

**Proof.** We will prove the theorem for \(f(x_{d+1})\). The other \(d\) marginals can be derived with simple changes of indices.

\[
f(x_{d+1}) = \frac{e^{\lambda_{d+1} x_{d+1}^2}}{F(\Lambda)} \int_{x_1^2 + \cdots + x_{d-1}^2 = 1 - x_{d+1}^2} e^{\lambda_1 x_1^2 + \cdots + \lambda_d x_d^2}.
\]

Now let \(y_i = \sqrt{x_i^2 \frac{x_i^2}{1 - x_{d+1}^2}}\) for \(i = 1 \ldots d\). Then \(y_1^2 + \cdots + y_d^2 = 1\), so \(y \in S^{d-1}\). Furthermore, \(x_i^2 = (1 - x_{d+1}^2) y_i^2\) and \(dx_i = \sqrt{1 - x_{d+1}^2} dy_i\). Thus,

\[
f(x_{d+1}) = \frac{e^{\lambda_{d+1} x_{d+1}^2}}{F(\Lambda)} \int_{y \in S^{d-1}} (1 - x_i^2)^{d/2} e^{\lambda_1 (1 - x_{d+1}^2) y_1^2 + \cdots + \lambda_d (1 - x_{d+1}^2) y_d^2}
\]

\[
= \frac{e^{\lambda_{d+1} x_{d+1}^2} (1 - x_{d+1}^2)^{d/2}}{|S^{d-1}| F_1(\frac{1}{2}; \frac{d+1}{2}; \Lambda)} \cdot |S^{d-1}| F_1(\frac{1}{2}; \frac{d}{2}; (1 - x_{d+1}^2)\Lambda_{-(d+1)})
\]

\(^6\)The \((d + 1)\)st term in the exponent is included—even though \(\lambda_{d+1} = 0\)—in order to avoid a separate case for the marginal \(f(x_{d+1})\).
The general case of non-axis-aligned marginals is more difficult, because the integral in equation 2.41 becomes a normalization constant of a Bingham-von Mises-Fisher distribution rather than a Bingham distribution. The *Bingham-von Mises-Fisher* is a hyperspherical distribution derived from a Gaussian distribution with any mean, \( \mu \); its PDF is

\[
f(x; C, \mu) = \frac{1}{z(C, \mu)} e^{(x-\mu)^T C (x-\mu)}. \tag{2.44}
\]

Infinite-series expansions are available for the normalization constant, \( z(C, \mu) \), but they have twice as many infinite sums as the normalization constant expansions for the Bingham distribution.

### 2.13 Mixtures of Binghams

For many applications, such as texture analysis in material science [67], the directional random variables of interest may be best represented by a mixture of Bingham distributions, rather than a single Bingham.

The PDF of a *Bingham mixture* is given by

\[
f(x; \Lambda, V, \alpha) = \sum_{i=1}^{k} \alpha_i f(x; \Lambda_i, V_i) \tag{2.45}
\]

where the \( \Lambda_i \)'s and \( V_i \)'s are component Bingham parameters and the \( \alpha_i \)'s are non-negative weights that sum to one.

#### 2.13.1 Parameter Estimation

There is no closed-form solution to the maximum likelihood parameter estimates for a Bingham mixture distribution, but a modified expectation maximization (EM) algorithm (with a minimum message length criterion to determine the number of components) has been used with success for texture analysis [67]. Fast (non-iterative) heuristic approaches to fitting Bingham mixtures have also been developed [31].
2.13.2 Approximating a Mixture by a Single Bingham

Given a random variable $X$ which is drawn from a mixture of two Binghams, with PDF $f(X) = \alpha f_1(X) + (1 - \alpha) f_2(X)$. Under certain circumstances, one may wish to approximate $f$ by a single Bingham with PDF $g$. To do this, consider what happens when we fit a Bingham distribution to $N$ samples drawn from the mixture model, $f$, as $N \to \infty$. Recall that with maximum likelihood parameter estimation (section 2.3), the sufficient statistic is the sample scatter matrix, $\hat{S} = \frac{1}{N} \sum_{i=1}^{N} x_i x_i^T$, which goes to the true scatter matrix, $S = E[xx^T]$, as $N \to \infty$. Thus, we simply need to fit $g$ to $f$’s scatter matrix,

$$S_f = \alpha E_1[xx^T] + (1 - \alpha) E_2[xx^T].$$

(2.46)

The component scatter matrices $E_1[xx^T]$ and $E_2[xx^T]$ can be computed with equation 2.33. The Bingham distribution which fits this matrix of second moments is the maximum likelihood fit of a single Bingham to a mixture of two Binghams. (For the information-theoretically inclined, this approach is equivalent to the variational approach, where $g$ is found by minimizing the KL divergence from $f$ to $g$, $D_{KL}(f \parallel g)$.) Note that this result can easily be extended to a mixture of $K$ Binghams—we will simply need to sum over $K$ component inertia matrices in equation 2.46 rather than just two.

2.14 Historical Remarks and Contributions

The Bingham distribution was first published by Christopher Bingham in his 1964 thesis [10], and was expanded upon in a 1974 journal article [11]. The next major result was by Mardia in 1975 [58], where the characterization of the Bingham as a maximum entropy distribution was uncovered, and Jupp and Mardia in 1979 [41], where maximum likelihood estimation for the Bingham was extended to hyperspheres of any dimension. (Bingham’s work was primarily focused on the 3-D sphere, $S^2$.)

Later work on the Bingham distribution mostly focused on applications in material science [68, 75, 53], and extensions to complex fields [44, 45] with applications to 2-D
shape analysis.

To our knowledge, most of the material in this chapter after section 2.3 is new to this thesis, with the exception of section 2.5 and section 2.8 up to theorem 2.9. Specifically, the results on Bayesian parameter estimation, sampling, second moments matrix, entropy, KL divergence, product of Bingham PDFs, marginal distributions, mixtures of Binghams, as well as the proof of theorem 2.1 and methods for calculating and storing tables of normalizing constants and parameter estimates, are all believed to be contributions of this thesis. Furthermore, previously-known results have been tailored to be consistent with the standard form of the Bingham distribution in this work (where $\lambda_{d+1}$ and $v_{d+1}$ are dropped).

2.14.1 Software

As a part of this thesis work, an open-source software library called libbingham\textsuperscript{7} has been developed which provides a C and Matlab interface to most of the common operations on Bingham distributions, with a focus on hyperspheres of dimension 4 and lower ($S^1$, $S^2$, and $S^3$).

\textsuperscript{7}https://code.google.com/p/bingham/
Chapter 3

The Quaternion Bingham Distribution

3.1 Understanding the Space of 3-D Rotations

As we discussed in chapter 1, one of the biggest challenges in dealing with 3-D geometry lies in appropriately handling 3-D rotational data. The topological space of 3-D rotations is called $\text{SO}(3)$, the special orthogonal group of dimension 3, and has several unique properties:

- The composition of rotations is non-commutative: in general, $p \circ q \neq q \circ p$.
- Every rotation $p$ has a unique inverse rotation $p^{-1}$.
- Elements of $\text{SO}(3)$ can be thought of either as a 3-D rotation operator (i.e., a function that rotates points in 3-D space), or as a 3-D orientation (i.e., a coordinate frame at the origin in 3-D space).

There are many ways to parameterize $\text{SO}(3)$. Perhaps the most intuitive (and common) way is as a set of three Euler angles $\alpha, \beta, \gamma$, specifying a sequence of rotation angles about three axes (e.g., the $z$-, $x$-, and $z$-axes). However, this representation suffers from a problem known as gimbal lock—when $\beta$ reaches a special value (0 or $\pi$), any fixed sum $\alpha + \gamma$ for $\beta = 0$ (or $\alpha - \gamma$ for $\beta = \pi$) represents the same 3-D
rotation in $SO(3)$. Mathematically, this is called a singularity of the Euler angle chart on $SO(3)$, and presents many problems for engineers—most famously during the Apollo 11 moon landing mission, when an inertial measurement unit (IMU) on the spacecraft shut down because its internal gimbal system (a mechanical realization of Euler angles with interconnected rings) approached the singularity in Euler angle space. The astronauts then had to use the position of the stars to manually re-orient the spacecraft before restarting the IMU.

Gimbal lock also poses a problem when one tries to use a probability distribution on the space of Euler angles to represent uncertainty about 3-D rotations. Any of the common (non-trivial\(^1\)) distributions on angles—such as the wrapped Gaussian—will end up having different probability densities for points in Euler angle space that map to the same 3-D rotation!

Another common way of representing 3-D rotations is with a 3x3 rotation matrix. While this parameterization is useful for computations (rotation of a 3-D vector is accomplished by multiplication with the rotation matrix), it is a severe over-parameterization for $SO(3)$. As the number 3 indicates, the space of 3-D rotations has three degrees of freedom, while 3x3 rotation matrices have nine parameters. This greatly complicates the math involved in using probability distributions on the space of 3x3 rotation matrices. However, such distributions do exist—the matrix Langevin and the matrix Bingham being two examples. For a general discussion of distributions on the Stiefel manifold (the space of matrices with orthonormal columns), see the excellent book by Chikuse [16].

A third parameterization for $SO(3)$ is the axis-angle representation: any 3-D rotation can be described by an axis (unit vector) of rotation $u$ and an angle of rotation $\theta \in [0, \pi]$ (about $u$). The combination $(\theta, u)$ is both an over-parameterization and contains a singularity at $\theta = 0$.

However, when we combine the axis and angle into a single vector $v = \theta u$, it lies in a ball of radius $\pi$ in $\mathbb{R}^3$. This parameterization is sometimes known as the Euler vector. It has both the correct number of parameters (three) and no singularities,

\(^1\)The uniform distribution is an exception, of course.
although it wraps around at antipodal points (since rotating by $\pi$ about $u$ is the same as rotating by $\pi$ about $-u$), which can be problematic for probability distributions.

Yet Euler vectors' biggest flaw is this: space gets "stretched out" the further $\mathbf{v}$ gets from the origin.

To see why this is the case, note that a natural, minimum-effort distance metric on $SO(3)$ is given by the absolute value of the minimum angle of rotation (about any axis) required to rotate one 3-D orientation into another. In other words, if $\mathbf{p}$ and $\mathbf{q}$ are elements of $SO(3)$, $\mathbf{p} = \mathbf{r} \circ \mathbf{q}$ (where "$\circ$" denotes composition of rotations), and the axis-angle representation of $\mathbf{r}$ is $(\theta, \mathbf{u}_r)$, then the natural distance between $\mathbf{p}$ and $\mathbf{q}$ is equal to $|\theta_r|$.

Except for a special case (when $\mathbf{p}$ and $\mathbf{q}$ share the same axis of rotation), the Euclidean distance in Euler vector space overestimates the natural distance on $SO(3)$, and the error between the distances grows (non-linearly) the further the Euler vector gets from the origin. In figure 3-1, the natural and axis-angle (Euclidean in Euler vector space) distances are plotted as a function of $\theta$ for the case of $\theta_p = \theta_q = \theta$ and $\mathbf{u}_p \cdot \mathbf{u}_q = 0.9$. In other words, $\mathbf{p}$ and $\mathbf{q}$ share the same angle of rotation and their axes of rotation are separated by an angle of $\cos^{-1} 0.9 \approx \pi/7$.

Statistical methods that do use Euler vectors must therefore "unstretch" space to respect the natural distance metric of $SO(3)$ using methods from differential geometry like Lie groups and tensors [17].
3.2 Representing 3-D Rotations as Unit Quaternions

Although the Euler vector representation has some issues when we try to use it as a metric space, there is another simple transformation of the axis-angle parameterization of $SO(3)$ which does have much more desirable properties as a metric space, and therefore as a domain for probability distributions. That transformation is

$$q = (\cos(\theta/2), u \sin(\theta/2))$$  \hspace{1cm} (3.1)

which takes $S^1 \times S^2$ to $S^3$, the 4-D unit hypersphere (i.e., the space of all 4-D unit vectors). Furthermore, if we endow $S^3$ with the algebra of quaternions, then quaternion multiplication is equivalent to the composition of 3-D rotations, and quaternion inversion computes the inverse of 3-D rotations.

This unit quaternion parameterization has many desirable properties as a representation for $SO(3)$:

1. It has no singularities.
2. The geodesic distance on $S^3$ is proportional to the natural distance on $SO(3)$.
3. Unit quaternions lie on one of the simplest non-Euclidean spaces—the hypersphere.

The geodesic distance is the shortest-path distance along a manifold (e.g., on the surface of the hypersphere), and for $S^3$ is given by $d(p, q) = \cos^{-1}(p \cdot q)$.

The only potential pitfall with unit quaternions is that they aren’t unique: $q$ and $-q$ represent the same 3-D rotation, since rotating by $\theta$ about the axis $u$ is the same as rotating by $-\theta$ about $-u$. Yet as the reader should already be intimately aware of after reading chapter 2, this antipodal symmetry is a perfect match for the Bingham distribution, which shares the same symmetry!

The Bingham is not the only distribution that has been used on the space of unit quaternions, but it is often the best. Other approaches include using Gaussian
distributions on \( \mathbb{R}^4 \) and pretending that the quaternions need not be unit vectors \([55, 15, 59, 18, 46, 50]\), or using a Gaussian distribution on the tangent space to the unit quaternion hypersphere at the mean of the distribution \([25]\). Such approaches rely on local linear approximations to the true topology of unit quaternion space, and are unable to effectively represent distributions that have large variances.

### 3.3 Using the Bingham as a Distribution on 3-D Rotations

Although the Bingham distribution has been used as a distribution on \( SO(3) \) before this thesis \([53, 5]\), it was more commonly used to represent uncertainty in axial data on \( S^2 \). In geology, it has been used to encode preferred orientations of minerals in rocks \([53]\). Higher dimensional, complex forms of the Bingham distribution have also been used to represent the distribution over 2-D planar shapes \([22]\).

In this thesis, we will refer to the Bingham distribution on 3-D rotations (as represented by unit quaternions in \( S^3 \)) as the \textit{quaternion Bingham distribution (QBD)}. As we will see in the remainder of this thesis, the quaternion Bingham distribution is very flexible, is easy to use (formulas for inference are readily available), and has a deep connection to the problem of least-squares alignment of point sets. All of the results for quaternion Binghams in the remaining chapters are new contributions of this thesis.

In the remainder of this chapter, we will present formulas for fixed transformations of quaternion Bingham random variables, followed by a derivation for the (non-Bingham) distribution resulting from a composition of two quaternion Bingham random variables, as well as a Bingham approximation to the composition. We will conclude this chapter with a preview of the technical contributions for the quaternion Bingham to be found in later chapters.
### 3.3.1 Rotation by a constant quaternion

Given \( q \sim \text{Bingham}(\Lambda, V) \), \( u \in S^3 \), and \( s = u \circ q \), then \( s \sim \text{Bingham}(\Lambda, u \circ V) \), where \( u \circ V \triangleq [u \circ v_1, u \circ v_2, u \circ v_3] \). In other words, \( s \) is distributed according to a Bingham whose orthogonal direction vectors have been rotated (on the left) by \( u \).

Similarly, if \( s = q \circ u \) then \( s \sim \text{Bingham}(\Lambda, V \circ u) \).

**Proof** for \( s = u \circ q \): Since unit quaternion rotation is invertible and volume-preserving, we have

\[
    f_s(s) = f_q(u^{-1} \circ s) = \frac{1}{F} \exp\left\{ \sum_{i=1}^{d} \lambda_i (v_i^T(u^{-1} \circ s))^2 \right\}
\]

\[
    = \frac{1}{F} \exp\left\{ \sum_{i=1}^{d} \lambda_i ((u \circ v_i)^T s)^2 \right\}.
\]

### 3.3.2 Quaternion inversion

Given \( q \sim \text{Bingham}(\Lambda, V) \) and \( s = q^{-1} \), then \( s \sim \text{Bingham}(\Lambda, JV) \), where \( J \) is the quaternion inversion matrix, \( J = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} \). The proof follows the same logic as in the previous section.

### 3.3.3 Composition of quaternion Binghams

Many inference algorithms (such as the recursive filters in the next chapter) call for the composition of random rotations. When the random rotations are independent quaternion Bingham random variables, the methods in this section may be used to compute (or approximate) the distribution of their composition.

Letting \( q \sim \text{Bingham}(\Lambda, V) \) and \( r \sim \text{Bingham}(\Sigma, W) \), we wish to find the PDF of \( s = q \circ r \).

The **true distribution** is the convolution, in \( S^3 \), of the PDFs of the component distributions.
\[ f(s) = \int_{q \in S^3} f(s|q)f(q) = \int f(s; \Sigma, q \circ W)f(q; \Lambda, V) \]
\[ = \frac{1}{F(\Sigma)F(\Lambda)} \int_{q} \exp \sum_{i=1}^{3} \left[ \sigma_i((s \circ w_i^{-1})^T q)^2 + \lambda_i(v_i^T q)^2 \right] \]
\[ = \frac{1}{F(\Sigma)F(\Lambda)} \int_{q} e^{q^T C(s)q} \]

where

\[ C(s) = \sum_{i=1}^{3} \left( \sigma_i(s \circ w_i^{-1})(s \circ w_i^{-1})^T + \lambda_i v_i v_i^T \right). \tag{3.2} \]

Since all the \( \lambda_i \)'s and \( \sigma_i \)'s are negative, the matrix \( C \) is negative semidefinite. Therefore, \( \int_{q} e^{q^T C(s)q} \) is the normalization constant of a Bingham distribution, so that the final form for the composed PDF is a quotient of hypergeometric functions,

\[ f_{\text{true}}(s) = \frac{1F_1(1/2; \frac{\Sigma}{2}; C(s))}{[S^3] \cdot 1F_1(1/2; \frac{\Lambda}{2}, \Lambda)1F_1(1/2; \frac{\Sigma}{2}; \Sigma)}. \tag{3.3} \]

This does not have the form of a quaternion Bingham density.

To approximate the PDF of \( s \) with a Bingham density, \( f_B(s) = f_B(q \circ r) \), it is sufficient to compute the second moments of \( q \circ r \), since the inertia matrix, \( E[ss^T] \) is the sufficient statistic for the Bingham. This moment-matching approach is equivalent to the variational approach, where \( f_B \) is found by minimizing the KL divergence from \( f_B \) to \( f_{\text{true}} \).

Noting that \( q \circ r \) can be written as \( (q^TH_1^T r, q^TH_2^T r, q^TH_3^T r, q^TH_4^T r) \), where \( H_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \), \( H_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \), \( H_3 = \begin{bmatrix} 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \), and \( H_4 = \begin{bmatrix} 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \), we find that

\[ E[s_is_j] = E[q^TH_i^Trr^TH_j] \]

which has 16 terms of the form \( \pm r_a r_b r_c r_d \), where \( a, b, c, d \in \{1, 2, 3, 4\} \). Since \( q \) and \( r \) are independent, \( E[\pm r_ar_br_c r_d] = \pm E[r_ar_b]E[r_c r_d] \). Therefore (by linearity of expectation), every entry in the matrix \( E[ss^T] \) is a quadratic function of elements of the inertia matrices of \( q \) and \( r \), which can be easily computed given the Bingham normalization constants and their partial derivatives.

Let \( A = [a_{ij}] = E[qq^T] \) and \( B = [b_{ij}] = E[rr^T] \) be the inertia matrices of \( q \) and \( r \), respectively. Then \( E[ss^T] \) is given by:
3.3.4 Estimating the error of approximation

To estimate the error in the Bingham approximation to the composition of two quaternion Bingham distributions, \( B_1 \circ B_2 \), we approximate the KL divergence from \( f_B \) to \( f_{\text{true}} \) using a finite element approximation on the quaternion hypersphere

\[
D_{KL}(f_B \| f_{\text{true}}) = \int_{\mathbb{S}^3} f_B(x) \log \frac{f_B(x)}{f_{\text{true}}(x)} \approx \sum_{x \in \mathcal{F}(\mathbb{S}^3)} f_B(x) \log \frac{f_B(x)}{f_{\text{true}}(x)} \cdot \Delta x
\]

where \( \mathcal{F}(\mathbb{S}^3) \) and \( \{\Delta x\} \) are the points and volumes of the finite-element approximation to \( \mathbb{S}^3 \), based on a recursive tetrahedral-octahedral subdivision method [76].

To generate the tetrahedral-octahedral mesh that approximates \( \mathbb{S}^3 \), we begin by placing vertices at \((\pm 1, 0, 0, 0), (0, \pm 1, 0, 0), (0, 0, \pm 1, 0), \) and \((0, 0, 0, \pm 1)\). We then join the vertices with sixteen tetrahedra formed by taking one vertex from each of the four axes. This forms an initial tetrahedral mesh on \( \mathbb{S}^3 \), in the same way that
one can form a triangular mesh on the sphere $S^2$ by placing eight triangles between $(\pm 1, 0, 0), (0, \pm 1, 0),$ and $(0, 0, \pm 1)$. To increase the resolution of the mesh on $S^3$, we subdivide each of our sixteen initial tetrahedra into four tetrahedra (at the corners) and one octahedron (in the center) with vertices at the midpoints of the parent tetrahedron’s edges. Then we project each vertex back onto the unit hypersphere. Octahedra can be subdivided into eight tetrahedra and six octahedra in a similar way. The resulting tetrahedral-octahedral mesh may be subdivided until the necessary resolution on $S^3$ is obtained.

### 3.4 A Preview of Quaternion Bingham Results in Later Chapters

In the next chapter, we will derive a recursive filter for 3-D orientation data that is similar to the Kalman filter, but which uses the quaternion Bingham distribution (rather than the Gaussian distribution) as its noise model. Because the quaternion Bingham isn’t closed under composition, the filter will be approximate. However, in the practical applications we explore (including tracking the orientation and spin on a ping pong ball) the error of approximation due to composition is negligible.

In chapter 5, we use mixtures of quaternion Binghams to form a generative model of the distribution of local feature orientations on 3-D object models. Inverting this distribution allows one to estimate the orientation distribution of an object in a new scene, given a set of observed 3-D feature orientations. This chapter represents the first step towards a Bingham-based object detection system, but stops short of presenting a complete object detection system, as it is missing final alignment and hypothesis evaluation steps in the algorithm.

Chapter 6 presents a complete, discriminative, object detection system. It is based on a deep connection between the quaternion Bingham distribution and the least-squares alignment problem, which will yield a new characterization of the quaternion Bingham distribution (in addition to providing the probabilistic backbone for the
alignment step of the object detection algorithm).
Chapter 4

Tracking 3-D Rotations with the Quaternion Bingham Filter

We now present a new deterministic method—the Quaternion Bingham Filter (QBF)—for approximate recursive inference in quaternion Bingham processes\(^1\). The quaternion Bingham process is a type of dynamic Bayesian network (DBN) on 3-D rotational data, where both process dynamics and measurements are perturbed by random Bingham rotations. The QBF uses the Bingham distribution to represent state uncertainty on the unit quaternion hypersphere. Quaternions avoid the degeneracies of other 3-D orientation representations, while the Bingham enables accurate tracking of high-noise signals. Performing exact inference on quaternion Bingham processes requires the composition of Bingham distributions, which results in a non-Bingham density. Therefore, we approximate the resulting composed distribution as a Bingham using the method of moments, in order to keep the filtered state distribution in the Bingham family.

4.1 Discrete-time quaternion Bingham process

We will derive filters for both first- and second-order rotating systems. Since most physical systems have second-order dynamics, we expect second-order models to be

\(^1\)This work in this chapter was published as an MIT technical report [29].
Second-order quaternion Bingham process
\[ v_n = w_n \circ u_n \circ v_{n-1} \]
\[ x_n = v_{n-1} \circ x_{n-1} \]
\[ y_n = z_n \circ x_n \]

First-order quaternion Bingham process
\[ x_n = w_n \circ u_n \circ x_{n-1} \]
\[ y_n = z_n \circ x_n \]

Figure 4-1: Process and graphical models for the discrete quaternion Bingham process.

a better fit for many real-world rotating systems, such as the spinning ball data set in section 5.4.

The first-order discrete-time quaternion Bingham process has, as its state, \( x_n \), a unit quaternion representing the orientation of interest at time \( n \). The system’s behavior is conditioned on control inputs \( u_n \), which are also unit quaternions. The new orientation is the old orientation rotated by the control input and then by independent noise \( w_n \sim \text{Bingham}(A_p, V_p) \). Note that "\( \circ \)" denotes quaternion multiplication, which corresponds to composition of rotations for unit quaternions. (\( q \circ r \) means “rotate by \( r \) and then by \( q \”).)

The second-order quaternion Bingham process has state \((x_n, v_n)\), where \( x_n \) represents orientation and the quaternion \( v_n \) represents discrete rotational velocity at time \( n \). The control inputs \( u_n \) are analogous to rotational accelerations. Process noise \( w_n \) enters the system in the velocity dynamics.

We use a quaternion to represent rotational velocity so that its uncertainty can be modeled as a quaternion Bingham distribution.
In both the first-order and second-order systems, observations \( y_n \) are given by the orientation \( x_n \) corrupted by independent Bingham noise \( z_n \sim \text{Bingham}(\Lambda_0, V_0) \). One common choice for \( V_p \) and \( V_o \) is \[
\begin{bmatrix}
0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix},
\]
which means that the mode is the quaternion identity, \((1, 0, 0, 0)\). (Any \( V \) matrix whose top row contains all zeros will have this mode.) Figure 4-1 shows the process and graphical models for the discrete quaternion Bingham process.

### 4.2 Discrete quaternion Bingham filter

The state of a discrete-time quaternion Bingham process can be estimated using a discrete-time quaternion Bingham filter, which is a recursive estimator similar in structure to a Kalman filter. Unlike the Kalman filter, however, the QBF is approximate, in the sense that the tracked state distribution is projected to be in the Bingham family after every time step. The second-order QBF will also require an assumption of independence between \( x_n \) and \( v_n \), given all the data up to time \( n \). Both the first-order and second-order QBFs are examples of \textit{assumed density filtering}—a well-supported approximate inference method in the DBN literature [12]. We will start by deriving the first-order QBF, which follows the Kalman filter derivation quite closely. Note that the following derivations rely on several properties of Bingham distributions which were detailed in chapters 2 and 3.

**First-order QBF.** Given a distribution over the initial state \( x_0 \), \( B_{x_0} \sim \text{Bingham}(\Lambda_0, V_0) \), and an action-observation sequence \( u_1, y_1, \ldots, u_n, y_n \), the goal is to compute the posterior distribution \( f(x_n \mid u_1, y_1, \ldots, u_n, y_n) \). We can use Bayes’ rule and the Markov property to decompose this distribution as follows:

\[
B_{x_n} = f(x_n \mid u_1, y_1, \ldots, u_n, y_n)
= f(y_n \mid x_n) \int f(x_n \mid u_1, y_1, \ldots, u_{n-1}, y_{n-1}, u_n)
= f(y_n \mid x_n) \int f(x_n \mid x_{n-1}, u_n) B_{x_{n-1}}(x_{n-1})
= f(y_n \mid x_n)(f_{w_n} \circ u_n \circ B_{x_{n-1}})(x_n),
\]

where \( f_{w_n} \circ u_n \circ B_{x_{n-1}} \) means rotate \( B_{x_{n-1}} \) by \( u_n \) and then convolve with the process
noise distribution, \( f_{w_n} \). For the first term, \( f(y_n|x_n) \), recall that the observation process is \( y_n = z_0 \circ x_n \), so \( y_n|x_n \sim Bingham(y_n; \Lambda_o, V_o \circ x_n) \), where we used the result from section 3.3.1 for rotation of a Bingham by a fixed quaternion. Thus the distribution for \( y_n|x_n \) is

\[
f(y_n|x_n) = \frac{1}{F_o} \exp\left\{ \sum_{i=1}^{3} \lambda_{oi}(y_n^T(V_{oi} \circ x_n)) \right\}.
\]

Now we can rewrite \( y_n^T(V_{oi} \circ x_n) \) as \( (V_{oi}^{-1} \circ y_n)^T x_n \), so that \( f(y_n|x_n) \) is a Bingham density on \( x_n \), \( Bingham(x_n; \Lambda_o, V_o^{-1} \circ y_n) \). Thus, computing \( B_{x_n} \) reduces to multiplying two Bingham PDFs on \( x_n \), which is given in section 2.11.

**Second-order QBF.** Given a factorized distribution over the initial state \( f(x_0, v_0) = B_{x_0} B_{v_0} \) and an action-observation sequence \( u_1, y_1, \ldots, u_n, y_n \), the goal is to compute the joint posterior distribution \( f(x_n, v_n | u_1, y_1, \ldots, u_n, y_n) \). However, the joint distribution on \( x_n \) and \( v_n \) cannot be represented by a Bingham distribution\(^3\). Instead, we compute the marginal posteriors over \( x_n \) and \( v_n \) separately, and approximate the joint posterior as the product of the marginals. The marginal posterior on \( x_n \) is

\[
B_{x_n} = f(x_n | u_1, y_1, \ldots, u_n, y_n) \\
\propto f(y_n|x_n) \int_{x_{n-1}} f(x_{n-1} | x_n) B_{x_{n-1}}(x_{n-1}) \\
= f(y_n|x_n)(B_{v_{n-1}} \circ B_{x_{n-1}})(x_n)
\]

since we assume \( x_{n-1} \) and \( v_{n-1} \) are independent given all the data up to time \( n - 1 \).

Similarly, the marginal posterior on \( v_n \) is

\[
B_{v_n} = f(v_n | u_1, y_1, \ldots, u_n, y_n) \\
\propto \int_{v_{n-1}} f(v_n | v_{n-1}, u_n) B_{v_{n-1}}(v_{n-1}) f(y_n|v_{n-1}, u_1, y_1, \ldots, u_{n-1}, y_{n-1}) \\
= \int_{v_{n-1}} f(v_n | v_{n-1}, u_n) B_{v_{n-1}}(v_{n-1}) \int_{x_n} f(y_n|x_n) f(x_n|v_{n-1}, u_1, y_1, \ldots, u_{n-1}, y_{n-1})
\]

Once again, \( f(y_n|x_n) \) can be written as a Bingham density on \( x_n \), \( Bingham(x_n; \Lambda_o, V_o^{-1} \circ y_n) \). Next, note that \( x_n = v_{n-1} \circ x_{n-1} \) so that \( f(x_n|v_{n-1}, u_1, y_1, \ldots, u_{n-1}, y_{n-1}) = \)

\(^3\)Exploring good distributions for pairs of 3-D rotations—as unit quaternions on \( S^3 \times S^3 \) or for full 3-D poses on \( \mathbb{R}^3 \times \mathbb{R}^3 \) is an area of ongoing research.
\( B_{x_{n-1}}(v_{n-1}^{-1} \circ x_n), \) which can also be re-written as a Bingham on \( x_n. \) Now letting \( x_{n-1} \sim \text{Bingham}(\Sigma, W), \) and since the product of two Bingham PDFs is Bingham, the integral over \( x_n \) becomes proportional to a Bingham normalization constant, \( F(C(v_{n-1})), \) where

\[
C(v_{n-1}) = \sum_{i=1}^{3} \left( \sigma_i(v_{n-1} \circ w_i)(v_{n-1} \circ w_i)^T + \lambda_{oi}(v_{oi}^{-1} \circ y_n)(v_{oi}^{-1} \circ y_n)^T \right).
\]

Comparing \( C(v_{n-1}) \) with the formula for the composition of two independent Bingham random variables (equation 3.2 in section 3.3.3) we find that \( F(C(v_{n-1})) \propto (f_{y_n|x_n} \circ B_{x_{n-1}}^{-1})(v_{n-1}). \) Thus,

\[
B_{v_n} \propto \int_{v_{n-1}} f(v_n \mid v_{n-1}, u_n) B_{v_{n-1}}(v_{n-1}) \cdot (f_{y_n|x_n} \circ B_{x_{n-1}}^{-1})(v_{n-1})
\]

\[
= (f_{w_n} \circ u_n \circ (B_{v_{n-1}} \cdot (f_{y_n|x_n} \circ B_{x_{n-1}}^{-1}))) (v_n)
\]

In other words, to update the belief on \( v_n, \) we first convolve the inverse belief on \( x_{n-1} \) with the measurement distribution, then multiply by the belief on \( v_{n-1}, \) rotate by the control input \( u_n, \) and convolve by the noise distribution, \( f_{w_n}. \) Intuitively, this update rule combines the prior velocity belief, \( B_{v_{n-1}}, \) with the observed velocity information—i.e., the change from \( x_{n-1} \) to \( y_n. \) Then a process update (borrowing terminology from the Kalman filter) is applied by rotating by \( u_n \) and adding noise.

### 4.2.1 Extensions

For some applications (such as tracking a ping pong ball through a bounce), the process and observation models of the QBFs described above may be somewhat restrictive. Several extensions are possible, as we outline here.

**Sampling-based methods / Tracking the ball through a bounce.** The QBF can handle arbitrary process and control models by drawing \( N \) samples at random\(^4\) from the current state distribution, applying the process/control function to each sample, and then fitting new Bingham distributions to the resulting post-process/control samples. This is precisely the method we use in our experiments to

\(^4\)A related method is the unscented quaternion Bingham filter, which draw samples deterministically in order to handle arbitrary process models [28].
track the ping pong ball through a bounce on the table. The process model is taken from Andersson’s ball physics derivation [4], and is a complex function of the ball’s tracked angular and translational velocities,

\[ w_f = w_i + \frac{3\mu}{2r}(\vec{v}_{ry}, -\vec{v}_{rx}, 0)v_{iz}(1 + \epsilon), \]

where \( w_i \) and \( w_f \) are the initial and final (pre- and post-bounce) angular velocity vectors, \( \mu \) and \( \epsilon \) are the coefficients of friction and restitution, \( r \) is the ball’s radius, \( v_{iz} \) is the ball’s initial translational z-velocity, and \( \vec{v}_r = v_r/\|v_r\| \), where \( v_r = (v_{iy} + w_{iz}r, v_{ix} - w_{iy}r, 0) \) is the relative velocity of the surface of the ball with respect to the table.

**Quaternion exponentiation / Continuous-time filters.** Quaternion exponentiation for unit quaternions is akin to scaling in Euclidean space. If \( q \) represents a 3-D rotation of angle \( \theta \) about the axis \( v \), then \( q^a \) is a rotation of \( a\theta \) about \( v \). This operation would be needed to handle a continuous-time update in the second-order Bingham filter, since the orientation needs to be rotated by some fraction of the spin quaternion at each (time-varying) time step.

Although we do not do it in this work, it is possible to incorporate quaternion exponentiation in all parts of the model via a moment-matching method for Bingham exponentiation (akin to the moment-matching method for Bingam composition). However, an additional Taylor approximation is needed to approximate the second moments of the exponentiated Bingham as a function of the second and higher even moments of the original distribution. (Recall from chapter 2 that the odd moments of a Bingham are always zero due to symmetry.)

### 4.3 Experimental Results

We compare the quaternion Bingham filter against an extended Kalman filter (EKF) approach in quaternion space [55], where process and observation noise are generated by Gaussians in \( \mathbb{R}^4 \), the measurement function normalizes the quaternion state (to project it onto the unit hypersphere), and the state estimate is renormalized after
Figure 4-2: Two simulated runs with the quaternion Bingham filter—(b) and (c) are different plots of the same simulation. In all figures, the thick black line is the true process signal, generated with isotropic process noise \( \Lambda_p = (-400, -400, -400) \). The thin blue lines in (a) and (b) are the observation signal, and the thick red line is the filtered output. Rows 1-4 in each figure show the 4 quaternion dimensions \((q_w, q_x, q_y, q_z)\). Row 5 in (a) and (b) shows the error between filter output and true state (thick red line), together with the QBF sample error 90% confidence bounds (thin red line). Marginal sample 90% confidence bounds are also shown in the thin red lines in (c).

every update. We chose the EKF both due to its popularity and because LaViola reports in [55] that it has similar (slightly better) accuracy to the unscented Kalman filter (UKF) in several real tracking experiments. We adapted two versions of the EKF (for first-order and second-order systems) from LaViola’s EKF implementation by changing from a continuous to a discrete time prediction update. We also mapped QBF (Bingham) noise parameters to EKF (Gaussian) noise parameters by empirically matching second moments from the Bingham to the projected Gaussian—i.e., the Gaussian after it has been projected onto the unit hypersphere.

**Synthetic Data.** To test the first-order quaternion Bingham filter, we generated several synthetic signals by simulating a quaternion Bingham process, where the (ve-
locity) controls were generated so that the nominal process state (before noise) would follow a sine wave pattern on each angle in Euler angle space. We chose this control pattern in order to cover a large area of 3-D rotation space with varying rotational velocities. Two examples of synthetic signals along with quaternion Bingham filter output are shown in figure 4-2. Their observation parameters were \( \Lambda_o = (-50, -50, -50) \), which gives moderate, isotropic observation noise, and \( \Lambda_o = (-10, -10, -1) \), which yields moderately high noise in the first two directions, and near-uniform noise in the third direction. We estimated the composition approximation error (KL-divergence) for 9 of these signals, with both isotropic and nonisotropic noise models, from all combinations of \( (\Lambda_p, \Lambda_o) \) in \{(-50, -50, -50), (-200, -200, -200), (-10, -10, -1) \}. The mean composition error was .0012, while the max was .0197, which occurred when \( \Lambda_p \) and \( \Lambda_o \) were both \((-10, -10, -1)\).

For the EKF comparison, we wanted to give the EKF the best chance to succeed, so we generated the data from a projected Gaussian process, with process and observation noise generated according to a projected Gaussian (in order to match the EKF dynamics model) rather than from Bingham distributions. We ran the first-order QBF and EKF on 270 synthetic projected Gaussian process signals (each with 1000 time steps) with different amounts of process and observation noise, and found the QBF to be more accurate than the EKF on 268/270 trials. The mean angular change in 3-D orientation between time steps were 7, 9, and 18 degrees for process noise parameters -400, -200, and -50, respectively (where -400 means \( \Lambda_p = (-400, -400, -400) \), etc.).

The most extreme cases involved anisotropic observation noise, with an average improvement over the EKF mean error rate of 40-50%. The combination of high process noise and low observation noise also causes trouble for the EKF. Table 4.1 summarizes the results. Error rates were computed for the EKF and QBF on each signal by measuring the average absolute difference between the true signal and the modes of the EKF and QBF filtered distributions. Percentage improvement values (in the error rates) for QBF over EKF were then calculated for each signal; the mean and standard deviation of these percentage improvement values across all signals for each process and noise model are shown in table 4.1.
Table 4.1: Projected Gaussian process simulations. Average % mean error decrease for QBF over EKF.

<table>
<thead>
<tr>
<th>Observation noise</th>
<th>-400, -400, -10</th>
<th>-400</th>
<th>-50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process noise</td>
<td>-50, -200, -400</td>
<td>-50</td>
<td>-200</td>
</tr>
<tr>
<td>% Improvement</td>
<td>37.3, 45.6, 54.3</td>
<td>19.9</td>
<td>3.33</td>
</tr>
<tr>
<td>±</td>
<td>5.1, 5.0, 6.4</td>
<td>1.8</td>
<td>0.53</td>
</tr>
<tr>
<td></td>
<td>(5.1) (5.0) (6.4)</td>
<td>(1.8) (0.53)</td>
<td>(0.44)</td>
</tr>
</tbody>
</table>

Figure 4-3: Example image sequences from the spinning ping-pong ball dataset. In addition to lighting variations and low image resolution, high spin rates make this dataset extremely challenging for orientation tracking algorithms. Also, because the cameras were facing top-down towards the table, tracking side-spin relies on correctly estimating the orientation of the elliptical marking in the image, and is therefore much harder than tracking top-spin or under-spin.

Spinning ping-pong ball dataset. To test the second-order QBF, we collected a dataset of high-speed videos of 73 spinning ping-pong balls in flight (Figure 4-3). On each ball we drew a solid black ellipse over the ball’s logo to allow the high-speed (200fps) vision system to estimate the ball’s orientation by finding the position and orientation of the logo.

However, an ellipse was only drawn on one side of each ball, so the ball’s orientation could only be estimated when the logo was visible in the image. Also, since ellipses are symmetric, each logo detection has two possible orientation interpretations. The balls were spinning at 25-50 revolutions per second.

5Detecting the actual logo on the ball, without darkening it with a marker, would require improvements to our camera setup.
6We disambiguated between the two possible ball orientation observations by picking the obser-
(which equates to a 45-90 degree orientation change per frame), making the filtering problem extremely challenging due to aliasing effects. We used a ball gun to shoot the balls with consistent spin and speed, at 4 different spin settings (topspin, underspin, left-sidespin, and right-sidespin) and 3 different speed settings (slow, medium, fast), for a total of 12 different spin types. Although we initially collected videos of 107 ball trajectories, the logo could only be reliably found in 73 of them; the remaining 34 videos were discarded. Although not our current focus, adding more cameras, adding markings to the ball, and improving logo detections would allow the ball’s orientation and spin to be tracked on a larger percentage of such videos. To establish an estimate of ground truth, we then manually labeled each ball image with the position and orientation of the logo (when visible), from which we recovered the ball orientation (up to symmetry). We then used least-squares non-linear regression to smooth out our (noisy) manual labels by finding the constant spin quaternion, \( \hat{s} \), which best fit the labeled orientations for each trajectory\(^7\).

To run the second-order QBF on this data, we initialized the QBF with a uniform orientation distribution \( B_{vo} \) and a low concentration \( (\Lambda = (-3, -3 - 3)) \) spin distribution \( B_{vo} \) centered on the identity rotation, \((1, 0, 0, 0)\). In other words, we provided no information about the ball’s initial orientation, and an extremely weak bias towards slower spins. We also treated the “no-logo-found” (a.k.a. “dark side”) observations as a very noisy observation of the logo in the center of the back side of the ball at an arbitrary orientation, with \( \Lambda_o = (-3.6, -3.6, 0)\)\(^8\). When the logo was detected, we used \( \Lambda_o = (-40, -40, -10) \) for the observation noise. A process noise with \( \Lambda_p = (-400, -400, -400) \) was used throughout, to account for small perturbations to spin.

Results of running the second-order QBF (QBF-2) are shown in figure 4-4. We compared the second-order QBF to the second-order EKF (EKF-2) and also to the variation with highest likelihood under the current QBF belief.

\(^7\)Due to air resistance and random perturbations, the spin was not really constant throughout each trajectory. But for the short duration of our experiments (40 frames), the constant spin approximation was sufficient.

\(^8\)We got this \( \Lambda_o \) by fitting a Bingham to all possible dark side orientations.
first-order QBF and EKF (QBF-1 and EKF-1), which were given the difference between subsequent orientation observations as their observations of spin. The first-order filters were initialized with the same spin distributions as the second-order filters. The solid, thin, blue line in each plot marked "oracle prior" shows results from running QBF-2 with a best-case-scenario prior, centered on the average ground truth spin for that spin type, with $\Lambda = (-10, -10, -10)$. We show mean orientation and spin errors (to regressed ground truth), and also spin classification accuracy using the MAP estimate of spin type (out of 12) given the current spin belief\(^9\). The

\(^9\)Spin was classified into one of the 12 spin types by taking the average ground truth spin for each spin type and choosing the one with the highest likelihood with respect to the current spin belief.

Figure 4-4: Spinning ping-pong ball tracking results. Top row: comparison of QBF-2 (with and without an oracle-given prior) to QBF-1, EKF-1, EKF-2, and random guessing (for spin classification); QBF-1 and EKF-1 do not show up in the orientation error graph because they only tracked spin. Note that QBF-2 quickly converges to the oracle error and classification rates. Bottom row: QBF-2 results broken down into top-spin/under-spin vs. side-spin. As mentioned earlier, the side-spin data is harder to track due to the chosen camera placement and ball markings for this experiment. Note that the time axes are measured in frames (not seconds).
results clearly show that QBF-2 does the best job of identifying and tracking the ball rotations on this extremely challenging dataset, achieving a classification rate of 91% after just 30 video frames, and a mean spin (quaternion) error of 0.17 radians (10 degrees), with an average of 6.1 degrees of logo axis error and 6.8 degrees of logo angle error. In contrast, the EKF-2 does not significantly outperform random guessing, due to the extremely large observation noise and spin rates in this dataset. In the middle of the pack are QBF-1 and EKF-1, which converge much more slowly since they use the raw observations (rather than the smoothed orientation signal used by QBF-2) to estimate ball spin. Finally, to address the aliasing problem, we ran a set of 12 QBFs in parallel, each with a different spin prior mode (one for each spin type), with \( \Lambda = (-10, -10, -10) \). At each time step, the filter was selected with the highest total data likelihood. Results of this “filter bank” approach are shown in the solid, thin, green line in figure 4-4.

**Simulated balls.** Orientation and spin error rates for the QBF-2 were quite high on the real ping-pong ball data set due to poor logo detection rates caused by low-resolution ball images (and by only seeing the logo when it was on the top side of the ball). Therefore, to further validate the accuracy of the second-order QBF, we generate a set of 100 simulated flying ball trajectories with constant spin between 0 and \( \pi/2 \) radians per frame (with an average spin rate of \( 1.16 \approx \pi/2.7 \) radians). Observed orientations were perturbed by random quaternion Bingham noise, with two different noise settings: “low noise” with \( \Lambda_o = (-400, -400, -100) \), and
Figure 4-6: Average QBF spin tracking error as it tracks the spin through the ball’s bounce on the table. Post-bounce errors are significantly lower with the sample-based bounce tracking method (solid black line) outlined section 4.2.1.

“high noise” with $\Lambda_o = (-40, -40, -10)$. We also compared the filter’s performance with the top-side-only ("one side") observations to the performance when orientation measurements were available from any ball orientation ("both sides"). Figure 4-5 summarizes the results. This simulated experiment demonstrates that the QBF-2’s high error rates on the real ball data set are caused by poor quality observations, and not by shortcomings of the quaternion Bingham filter.

**Tracking through the bounce.** We also used the sampling-based method outlined in section 4.2.1 to track the ball through the bounce (on real ball data) for 5 topspin/right-sidespin and 5 underspin/left-sidespin ball trajectories, and found that incorporating the bounce model as a sample-based process update in the quaternion Bingham filter (rather than restarting the filter after the bounce) resulted in a significant reduction in tracking error post-bounce (Figure 4-6).

In Figure 4-7 we show an example of the output of the second-order QBF we used to track the orientation and spin on the ball through one of the underpin/left-sidespin trajectories. In the first image frame, no logo is detected, so the orientation distribution is initialized to the “dark side” of the ball, and the spin distribution is close to a uniform distribution. After a few more frames, the filter has an accurate estimate of both the orientation and spin of the ball, with fairly high concentration parameters (low-uncertainty) in its Bingham distributions. After the bounce, the sample-based process update correctly updates the orientation and spin on the ball,
Figure 4-7: An example ball trajectory (underspin + left-sidespin) and the state of the QBF as it tracks the ball's orientation and spin through the bounce. In the top row of ball images, the big red (solid-line) axis is the mode of the QBF's orientation distribution, and the small red axes are random samples from the orientation distribution. The big green (dashed-line) axis is the detected ball orientation in that image. In the bottom row, the big red (solid-line) axis is the mode of the QBF's spin distribution, and the small red axes are random samples from the spin distribution. The big green (dashed-line) axis is the ground truth spin, and the black axis in the center is the identity (no-spin), for reference.

and the filter maintains correct, high-concentration distributions.

4.4 Conclusion

For many control and vision applications, the state of a dynamic process involving 3-D orientations and spins must be estimated over time, given noisy observations. Previously, such estimation was limited to slow-moving signals with low-noise observations, where linear approximations to 3-D rotation space were adequate. The contribution of our approach is that the quaternion Bingham filter encodes uncertainty directly on the unit quaternion hypersphere, using a distribution—the Bingham—with nice mathematical properties enabling efficient approximate inference, with no restrictions on the magnitude of process dynamics or observation noise. Because of the compact
nature of 3-D rotation space and the flexibility of the Bingham distribution, we can use the QBF not only for tracking but also for identification of signals, by starting the QBF with an extremely unbiased prior, a feat which previously could only be matched by more computationally-intensive algorithms, such as discrete Bayesian filters or particle filters.

The composition operator for quaternion Bingham distributions may also have applications beyond tracking the state of a rotating system. For example, uncertainty propagation on kinematic chains requires the addition (composition) of multiple noisy joint angle measurements, each of which could be represented as a quaternion Bingham random variable. The method-of-moments approximation may be used to approximate the distribution of orientations at the final link in the chain, without the need for sampling.
Chapter 5

A Generative Method for 3-D Object Detection

The success of personal service robotics hinges upon reliable manipulation of everyday household objects, such as dishes, bottles, containers, and furniture. In order to accurately manipulate such objects, robots need to know objects’ full 6-DOF pose, which is made difficult by clutter and occlusions. Many household objects have regular structure that can be used to effectively guess object pose given an observation of just a small patch on the object. In this chapter, we present a method to model the spatial distribution of oriented local features on an object, which we invert to find a distribution over object poses given small sets of observed local features. The orientation distribution for local features is given by a mixture of Binghams on the hypersphere of unit quaternions, while the local feature distribution for position given orientation is given by a locally-weighted (quaternion kernel) likelihood. Experiments on 3D point cloud data of cluttered and uncluttered scenes generated from a structured light stereo image sensor validate our approach.

While this chapter represents the first step towards a Bingham-based object detection system, it stops short of presenting a complete object detection system, as it is missing final alignment and hypothesis evaluation steps. This chapter is included in this thesis primarily as an exploratory prelude to the full object detection system in chapter 6, although the generative models developed here may be useful for related
tasks in the future. The work in this chapter was previously published in the proceedings of Robotics: Science and Systems in 2011 [31], and was the first publication from this thesis.

5.1 Introduction

The goal of this chapter is to determine a set of possible object poses, given a 3D model of the object and a point cloud observation of a scene containing the object. To answer this question, we will consider the observed point cloud to be made up of many tiny overlapping surface patches, and we will construct a model of the information each observed patch gives us about the object’s pose, by considering the range of locations the patch may have come from on the model. When such a surface patch contains orientation information in the form of a normal vector and principal curvature direction (or when the normal and principal curvature can be estimated from the local patch geometry), we call it an oriented local feature (or “oriented feature” for short).

In this chapter, we present a new method to model the spatial distribution of oriented local features on an object, which we use to infer object pose given small sets of observed features. We split up the spatial distribution of oriented local features into two parts, one modeling the distribution over feature orientations, and another modeling the conditional distribution of feature position given orientation. Splitting up the feature distributions in this way allows us to exploit predictable relationships between the orientation of local surface patches on an object and the object’s pose (Figure 5-1).

The distribution for feature orientation is given by a mixture of quaternion Bingham’s, while the distribution for feature position given orientation is given by a locally-weighted (Quaternion kernel) likelihood. Local likelihood is a non-parametric, kernel-based technique for modeling a conditional distribution, \( p(\mathbf{x}|q) \), as a smoothly varying function of the observed variable, \( q \).
Figure 5-1: Five 3-D point clouds and their corresponding Q-Image transforms, representing the distribution of local 3-D surface orientations on each object (see section 5.3 for details).

5.1.1 Outline

The technical portion of this chapter is organized in a top-down fashion. We present our pose estimation algorithms in section 5.2. Our main contribution—a new way to model the distribution of oriented local features on an object—is given in section 5.3. Section 5.4 contains experimental results, followed by related work and conclusions in sections 5.5 and 5.6.

5.2 Monte Carlo Pose Estimation

At its core, this chapter presents a new way to model the spatial relationship between an object and its oriented local features. This model can be used for many perceptual tasks, since there are many uses for such spatial models in a perceptual processing pipeline. As an application, we test the acuity of our model for object pose estimation, in a probabilistic Random-Sampling-Consensus (RANSAC) framework.

RANSAC [26] is a classic Monte-Carlo algorithm for using small random subsets from a large set of features to quickly generate many guesses of whatever the algorithm
is trying to estimate. It then ranks the guesses according to an evaluation criterion, and returns the top $n$ answers. RANSAC is part of a larger trend to solve perceptual problems by applying successive filters, or sieves, to a set of guesses (or samples), keeping good samples and throwing away the bad ones, until (one hopes) only good samples are left. The key to making such a filtering scheme work efficiently is to generate reasonable guesses early on in the pipeline, and throw away as many bad samples with the early, fast filters, so that the later, discriminative (but slow) filters don’t have to sift through as much junk to find the right answers.

Our model fits perfectly at the early stage of such a filtering scheme for pose estimation. By squeezing all the information one can get from a single oriented local feature, and then from small sets of oriented local features, we can quickly generate many good guesses for object pose. Later filters—which for example iteratively align a model at a sample pose with the observed point cloud and then compute a fitness score using all the points in the model—can be used to further refine the set of pose estimates, and are explored in chapter 6. However, that final stage of the estimation process is omitted in this chapter in order to explore the potential benefits of using a generative model of an object’s oriented local features. The hope is that one can get good initial sample poses using just a few local oriented features at a time, which is crucial for pose estimation in cluttered scenes, where only a small portion of an object may be visible.

5.2.1 Problem Statement

We address two primary pose estimation problems in this chapter. The first problem, which we call “Single Object Pose Estimation” (SOPE), is to estimate an object’s pose given a point cloud observation, $F_{obs}$, of part of the object (Figure 5-2). For this task, we have a model (i.e. we know the object’s identity) and a segmentation (i.e. we know which points in the observation correspond to the model), and the goal is to return samples from the distribution over possible object poses $(x, q) \in \mathbb{R}^3 \times S^3$ given the observation, $p(x, q|F_{obs})$. The SOPE problem is well suited to “separable” scenes, for example when one or more objects are separated on a table or a shelf.
Figure 5-2: The Single Object Pose Estimation (SOPE) problem: given a model and a (partial) observation (left), we wish to estimate the pose of the object (right).

- **Given**: a model, $M$, and an observed point cloud, $F_{obs}$.

- For $i = 1 \ldots N$
  - 1) Sample a "proposal" oriented feature, $f_p$, at random from $F_{obs}$.
  - 2) Sample an object pose, $(x_i, q_i)$ from $p_M(x, q | f_p)$.
  - 3) Sample $k$ "validation" oriented features, $\{f_v \ldots k\}$ at random from $F_{obs}$.
  - 4) Set sample weight $w_i \leftarrow p_M(x_i, q_i | \{f_v \ldots k\})$.

- **Return**: the top $n$ samples $(x_i, q_i)$ ranked by weight, $w_i$.

**Table 5.1: MC-SOPE**

The second problem is "Single Cluttered Object Pose Estimation" (SCOPE). In this task, we have a model, an observed point cloud containing several objects, and a rough location (e.g. a bounding box) in which to look for the desired object. The goal once again is to return a set of samples from the posterior pose distribution, $p(x, q | F_{obs})$, of the desired object.
5.2.2 MC-SOPE Algorithm

Our solution to the SOPE problem, MC-SOPE, is shown in table 5.1. It starts by sampling a pose using the information from one "proposal" oriented local feature \( f_p \) which is chosen at random from the observed point cloud, \( F_{obs} \). Then, it uses \( k \) "validation" features, \( f_{v1}, \ldots, f_{vk} \), to compute a weight/score for the proposed object pose. After \( N \) poses have been proposed and weighted, the algorithm returns the top \( n \) samples, ranked by weight. As we will see in section 5.3, sampling from \( p_M(x, q|f_p) \) and computing the validation density \( p_M(x, q|\{f_{v1} \ldots k\}) \) are both \( O(1) \) operations when \( k \) is a small constant (we use \( k = 5 \) in our experiments), so the entire algorithm is \( O(N) \), allowing us to generate large numbers of good samples quickly. This is in contrast to traditional RANSAC-type methods for model fitting and pose estimation, which typically use all the observed features in \( F_{obs} \) to rank each proposal, resulting in a running time of \( O(N \cdot |F_{obs}|) \).

5.2.3 MC-SCOPE Algorithm

Our solution to the SCOPE problem, MC-SCOPE, is shown in table 5.2. It is nearly identical to MC-SOPE, with the main difference being that MC-SCOPE samples validation points using a random walk (since the observed point cloud contains more than one object and the region of interest, \( F_{obs, roi} \) may not be a perfect segmentation of the desired object).

5.3 Local Feature Distributions

Both pose estimation algorithms in the previous section require a model for \( p_M(x, q|f_p) \) and \( p_M(x, q|\{f_{v1} \ldots k\}) \)—the likelihood of an object pose, given one or more observed oriented local features. Although the oriented local features on an object may not be unique, they can still be quite informative. For example, in figure 5-4, observing

\[1\text{It turns out to be quite difficult to generate good samples in clutter, which will motivate our more computationally-expensive detection algorithm in the next chapter.}\]
Given: a model, $M$, and an observed point cloud, $F_{obs}$ with sub-cloud region of interest, $F_{obs,roi}$.

For $i = 1 \ldots N$

1) Sample a “proposal” oriented feature, $f_p$, at random from sub-cloud $F_{obs,roi}$.

2) Sample an object pose, $(x_i, q_i)$ from $p_M(x, q|f_p)$.

3) Sample $k$ “validation” features, $\{f_{v_1 \ldots k}\}$ using a random walk in the full point cloud $F_{obs}$, starting from $f_p$.

4) Set sample weight $w_i \leftarrow p_M(x_i, q_i|\{f_{v_1 \ldots k}\})$.

Return: the top $n$ samples $(x_i, q_i)$ ranked by weight, $w_i$.

Table 5.2: MC-SCOPE

A curved patch of the surface on the cup or a flat patch on the base tells us that the object is aligned with the $z$-axis, at a narrow range of positions. In contrast,
Figure 5-4: Informative oriented local (surface patch) features. Observing just a single patch of an object can often constrain the set of feasible object poses to a narrow range of possibilities. In the top row, two different patches on a wine glass are shown, along with four possible object poses consistent with each patch. In the bottom row, a bowl feature is shown, along with four object poses consistent with the observed patch. Both of the observed glass features constrain the orientation of the glass to be aligned with the z-axis, while the bowl feature places much weaker constraints on orientation.

Oriented local features on the surface of a hemispherical bowl tell us very little about the bowl's orientation, and constrain the bowl's position to a bowl-shaped set of locations, centered on the observed patch.

Our approach to computing $p_M(x, q|f_i)$ will be to use Bayes' rule:

$$ p_M(x, q|f_i) \propto p_M(f_i|x, q)p_M(x, q) $$

where $p_M(x, q)$ is a prior probability and $p_M(f_i|x, q)$ is the likelihood of observing...
given the object pose \((x, q)\). We will gain further purchase on \(p_M(f_i| x, q)\) by examining the components of an oriented local feature, \(f_i\).

### 5.3.1 Oriented Local Features

In this chapter, an oriented local feature is a descriptor for the pose and shape of a local surface patch on a 3D point cloud. We describe an oriented local feature \(f_i\) with three components—

1. a shape descriptor, \(s_i\),

2. a position, \(x_i \in \mathbb{R}^3\), and

3. a quaternion orientation, \(q_i \in S^3\).

The shape descriptor \(s_i\) should be invariant to position and orientation, while describing the local surface geometry of the feature. After testing different 3D shape descriptors, we selected the Fast Point Feature Histogram (FPFH)[73], due to its favorable performance in the presence of noise and missing data. The position \(x_i\) and orientation \(q_i\) are relative to a fixed global coordinate frame; they describe the rigid body transform which maps global coordinates to local feature coordinates. Unit quaternions are used to describe rotations because they avoid the topological degeneracies of other representations, and fit perfectly into the Bingham mixture distributions we develop in this work.

Given a surface patch (i.e. a local set of 3D points \(P\) on a model \(M\)) with estimated surface normals \(U\) at every point, we compute the transform \((x_i, q_i)\) by picking an arbitrary (fixed) model coordinate frame, and estimating the local coordinate frame whose origin is at the center of patch \(P\), with axes given by \(u\), the estimated surface normal at the center of patch \(P\), \(v\), the direction of maximal surface curvature\(^2\), and

\(^2\)The direction of maximal surface curvature, or principal curvature, is estimated using principal components analysis on the neighborhood of local surface patch normals, projected into the tangent space to the central normal vector; the principal curvature is the eigenvector corresponding to the largest eigenvalue, and is only defined up to sign.
\( \mathbf{w} = \mathbf{u} \times \mathbf{v} \), the cross product of \( \mathbf{u} \) and \( \mathbf{v} \). An example of a local coordinate frame is shown on the left side of figure 5-6.

5.3.2 Local Feature Likelihood

The next step in modeling \( p_M(f_i|\mathbf{x}, \mathbf{q}) \) is to discretize shape descriptor space into a “vocabulary” \( \Omega \) of local shape types for each model. The reason for doing this is that when we observe a local feature in a new point cloud, we want to know where on the model that feature may have come from. Discretizing the local feature shapes into a set of distinct classed will give us a simple mapping from shape class to object pose distribution.

We perform the discretization of shape space using K-Means clustering (with \( |\Omega| = 10 \) in this work) on the set of all local shape descriptors, \( \{\mathbf{s}_i\} \) in the model point cloud, yielding a shape vocabulary, \( \{\Omega_j\} \). We then segment the model point
cloud into $|\Omega|$ clusters, where cluster $j$ contains all the oriented local features in the model whose closest vocabulary shape is $\Omega_j$. Examples of three feature clusters on a wine glass, with their local coordinate frames, is shown in the top row of figure 5-5.

For a newly observed oriented local feature, $f_i$, we compute the local feature likelihood, $p_M(f_i|x, q)$ by expanding $f_i$ into components $(s_i, x_i, q_i)$, and classifying $s_i$ into one of the shape vocabulary clusters, $\Omega_j$, yielding

$$p_M(f_i|x, q) = p_M(x_i, q_i|x, q, \Omega_j)p_M(\Omega_j|x, q)$$

(5.1)

where we can drop the dependence on $x$ in the first term (since the orientation of a local feature on an object shouldn’t depend on the object’s position in space), and where $p_M(\Omega_j|x, q)$ is assumed to be uniform. Thus,

$$p_M(f_i|x, q) = p_M(q_i|x, q, \Omega_j)p_M(x_i|q_i, x, q, \Omega_j).$$

(5.2)

To model $p_M(q_i|q, \Omega_j)$, we consider the set of all local feature orientations on the model from feature cluster $\Omega_j$, which we visualize using the “Q-Image” transform\(^3\) for three features types in the bottom row of figure 5-5. We then fit a Bingham Mixture Model (BMM)—defined in section 5.3.4—to the set of all rotations mapping model axes into local axes, so that

$$p_M(q_i|q, \Omega_j) = p(q_iq^{-1}; B_j)$$

(5.3)

where $B_j$ are parameters of the BMM for feature cluster $j$.

To model $p_M(x_i|q_i, x, q, \Omega_j)$, we use a non-parametric technique, called local likelihood, to fit a distribution to the set of all translations mapping model origin to rotated local feature origin; that is,

$$p_M(x_i|q_i, x, q, \Omega_j) = p_M(q^{-1}(x_i - x)|q_iq^{-1}, \Omega_j).$$

(5.4)

\(^3\)The Q-Image of a 3D point cloud is defined as the set of all quaternion local feature orientations; we visualize the Q-Image in 3D using the axis-angle format of 3D rotation.
Local Likelihood is a technique introduced by Tibshirani and Hastie [85] to model the parameters of a conditional probability distribution, \( p(Y|X) \), as a smoothly varying function of \( X \). In our case, we wish to model the conditional local feature pose distribution \( p(x|q) \) within a particular feature class, \( \Omega_j \) (where \( x|q \) is shorthand for \( q^{-1}(x_\Omega - x)|q|q^{-1} \) in equation 5.4). To do so, we choose a particular parametric model for \( p(x|q = q') \) (multivariate normal distribution), with parameters \( (\mu, \Sigma) \). To fit parameters, we use a locally-weighted data log-likelihood function, \( \ell() \), with Gaussian kernel \( K(q, q_i) \) to weight the contribution of each \( (x_i|q_i) \) in the model point cloud according to how close \( q_i \) is to \( q \).

\[
\ell(\mu, \Sigma; q, X, Q) = \sum_{i=1}^{n} K(q, q_i) \ell(\mu, \Sigma; x_i). 
\tag{5.5}
\]

We use a local likelihood model in this work for its flexibility, since finding a good parametric model for \( p(x|q) \) which works for all \( q \)'s is quite difficult. An example of \( p(x|q) \) for a particular \( q \) is shown on the right side of figure 5-6.

### 5.3.3 Multi-Feature Likelihood

Given multiple observed oriented local features, \( \{f_i\} = \{(s_i, x_i, q_i)\}, \ i = 1 \ldots k \), the local feature distribution \( p(\{f_i\}|x, q) \) would be simply a product of individual local feature distributions, \( \prod_i p(f_i|x, q) \), if the local features were independent given the object pose. However, since features may overlap, we use an exponential model which averages the log-likelihood contribution from each feature,

\[
p(\{f_i\}|x, q) = \lambda \exp \left[ \frac{\lambda}{k} \sum_{i=1}^{k} \log p(f_i|x, q) \right] 
\tag{5.6}
\]

where \( \lambda \) is a smoothing parameter (currently set to 0.5 in all of our experiments).

### 5.3.4 Bingham Mixture Models (BMMs)

The Bingham distribution is a natural, maximum entropy model for second order distributions on a hypersphere, \( S^d \). However, we have found that the Q-Image in a local feature distribution model will almost always be explained best by a more
Figure 5-6: Given that an oriented local feature with shape type $\Omega_j$ has the coordinate frame shown in (a), the range of possible feature positions is restricted to lie on the dark band of the cup in (b).

Figure 5-7: Fitting a Bingham Mixture Model (BMM). (a) A cluster of similar oriented local features. (b) The local coordinate frames of the oriented local features. (c-d) The local coordinate frames (only the axis part of the axis-angle format is visualized), along with the fitted Binghams (represented by the light bands on the two spheres).

A complex, multi-modal distribution, as one can see in figures 5-1 and 5-5. For this purpose, we use a Bingham Mixture Model (BMM), with PDF

$$f_{BM}(x; B, \alpha) = \sum_{i=1}^{k} \alpha_i f(x; B_i).$$  \hspace{1cm} (5.7)
Figure 5-8: Sample error distributions (bar plots), compared to random guessing for orientation (solid red lines). (a-c) MC-SOPE/uncluttered, (d) MC-SCOPE/cluttered

where the $B_i$'s are the component Bingham parameters and the $\alpha_i$'s are weights.

### 5.3.5 BMM Parameter Estimation via Sample Consensus

To fit a mixture model to a set of data points we must estimate the cluster parameters, $\theta_i$, the cluster weights, $\alpha_i$, and the number of clusters, $k$. Although iterative, Expectation Maximization (EM) algorithms for fitting BMMs exist [67], we found the following greedy, heuristic algorithm (based on the sample consensus framework) to work well in practice at a much lower computational cost. The BMM sample-consensus (BMM-SAC) algorithm starts by fitting $M$ Bingham to $M$ random sets of 4 points each. The Bingham $B^*$ which fits the data best under a capped loss function (where outliers all contribute a minimum likelihood, $\ell_{\text{min}}$, so as not to give them too much influence on the total fitness) is added to the mixture if it is a good enough fit (i.e. has data likelihood $> \ell_{\text{thresh}}$). Then, the inlier points with respect to $B^*$ (those points with likelihood greater than $\ell_{\text{min}}$) are removed from the training set, and the
Figure 5-9: (a) Search orientation accuracy of MC-SOPE and MC-SCOPE on each dataset. Each point, $(X, Y)$, in the figure shows the percentage $Y$ of trials for which a “good” pose sample was found in the top $X$ samples. (b) Comparison to the proposal sample distribution of SAC-IA (orientation error thresholds are the same as in (a)).

algorithm searches for another Bingham to add to the mixture. When no further Binghams can be found, the remaining points are considered outliers, and a uniform mixture component is added\(^4\). A reasonable choice for the outlier threshold, $\ell_{\text{min}}$, is given by the uniform probability density,

$$
\ell_{\text{min}} = p_{\text{unif}}(x) = 1/A(d),
$$

where $A(d)$ is the surface area of the hypersphere $S^d$.

5.4 Experimental Results

We tested our pose estimation algorithms on three different datasets—one containing 41 rotationally-symmetric IKEA objects (mostly dishes), one containing 29 rotationally-asymmetric household items, such as soap bottles, tools, and food containers, and one containing 8 manipulable objects for a PR2 robot. Each dataset consisted of a set of 3D point clouds which were scanned in from real objects; however, novel point cloud views of objects (for both training and testing) were generated in three different ways.

\(^4\)Recall that a Bingham with all its concentration parameters $\lambda_i$ set to zero is a uniform distribution on $S^d$; thus, the uniform component of a BMM is also a Bingham distribution.
For the IKEA dataset, we used ray-tracing with additive Gaussian noise to generate 41 views of each object. For the Household dataset, we used a 3-D simulator to emulate output from a texture-projection stereo camera pair. Then we ran a stereo vision algorithm on the simulated images to estimate pixel depth, and extracted point clouds from the rectified depth images for 71 views per object. For the PR2 dataset, we captured point clouds from the Willow Garage PR2 robot's projected light stereo cameras. To generate 3D models, we drove the robot around each object, placed in varying orientations on a table, collecting 34-40 views of each object; point clouds from all the views were then aligned using an occupancy-grid approach. The PR2 dataset also contained 11 point clouds of cluttered scenes with varying configurations of the 8 objects on a tabletop in front of the robot; we painstakingly hand-labelled every scene with each object’s true pose for testing purposes.

To fit local feature models for the objects in each dataset, we calculated normals, local shape descriptors (FPFHs), and principal curvatures for the points in each point cloud. After downsampling to reduce the number of oriented local features in each model to around 2000 points, we clustered features across all views by their FPFH using KMeans with \( k = 10 \). We then fit a Bingham Mixture Model (BMM) to the local coordinate frames (Q-Image) in each cluster. For the purposes of testing the MC-SOPE algorithm, we used cross-validation with 5 training and testing set pairs. For each testing set, we used the corresponding (non-overlapping) training set to learn FPFH clusters and BMMs. We then ran MC-SOPE on each point cloud with \( N = 1000 \) proposal samples in the testing sets, returning the top \( n = 100 \) pose samples for each view.

We ran the MC-SCOPE algorithm on the objects in each scene of the cluttered PR2 dataset with \( N = 5000 \) proposal samples, returning the top \( n = 100 \) pose samples for each object. In figure 5-10, we show examples of top pose samples for one of the scenes.

In figure 5-8, average error histograms across all MC-SOPE tests are shown for the three datasets, and for the cluttered PR2 dataset for the MC-SCOPE trials. For the IKEA dataset, we show orientation error in the \( z \)-axis, since the IKEA objects
are all symmetric about the z-axis. For the Household and PR2 datasets, we show orientation error in the z-axis, and also about the z-axis for samples with z-axis error < 20 degrees. Error in translation (XYZ) for samples with z-axis error < 20 degrees is shown for all datasets.

In figure 5-9, we show the search orientation accuracy of MC-SOPE and MC-SCOPE on each dataset. Each point, \((X, Y)\), in figure 5-9 shows the percentage \(Y\) of trials for which a “good” pose sample was found in the top \(X\) samples. A “good” sample is defined as z-axis error < 10 degrees for the IKEA dataset, and both z-axis error < 20 degrees and about-z-axis error < 20 degrees for the Household and PR2 datasets. Position accuracy is not measured in figure 5-9, since we saw in figure 5-8 that most samples with small orientation error have translation error within a few centimeters (and the average model size is about 10 cm). In figure 5-9(b), we also compare performance on the SOPE/uncluttered datasets to the proposal\(^5\) sample accuracy of SAC-IA, the RANSAC-based alignment method in the original FPFH paper [73]. Our method, MC-SOPE, is either comparable to or better than the proposal method of SAC-IA on all three datasets, with the biggest improvement being on the PR2 dataset (which has the most sensor noise). Further gains in MC-SOPE may be achieved by incorporating some of the feature selection/pruning methods in SAC-IA, which we have not yet explored in this work.

5.5 Related Work

Many modern techniques\(^6\) for pose estimation of a rigid 3D object are correspondence-based, relying on the presence of unique visual features on the object’s surface so that a rigid-body transform can be found which minimizes the sum of squared distances between corresponding points [39, 73, 56]. Unfortunately, many common household objects—such as dishes, tools, or furniture—do not have such uniquely-identifiable

\(^5\)The validation part of SAC-IA, which ranks each pose sample by computing an inlier percentage over the entire observed point cloud, is left out of the experiment for a fair comparison of how each algorithm does using only a few local features at a time.

\(^6\)including the object detection method presented in the next chapter.
visual features (or they have too few unique features to uniquely determine a pose).

Current alternatives to the correspondence-based approach tend to be limited in some way. Spherical harmonics can be used in an attempt to bring the object into a "standard" reference frame [14], and tend to work better than pure moment matching techniques. However, such standardization approaches may fail when a unique reference frame can't be found, and are sensitive to noise and occlusions. Brute force techniques using Extended Gaussian Images [37, 42] or registration [9] must exhaustively search the space of 3D object rotations, and are currently too slow for real-time applications. The generalized Hough transform [8] uses local geometric information to vote for object parameters (e.g. position and orientation), and is robust to occlusions. However, it suffers from quantization noise which can make parameter searches in high dimensions difficult [33]. Geometric hashing [54] tries to alleviate this quantization difficulty by voting among a discrete set of basis point tuples, but it has a worst case running time of $O(n^4)$ for $n$ 3D model points. Classification-based techniques, such as [74], which classify object observations by viewpoint, appear to work well when the object is perfectly segmented out of the scene, but are sensitive to clutter and occlusions.

Our method can be most accurately described as a hybrid between the Generalized Hough transform and RANSAC, in that we try to model the relationship between the poses of local features and the pose of the model in a random sampling framework. However, unlike the Hough transform, our model is continuous, which allows us to avoid discretization issues and perform extremely fast, constant time inference.

5.6 Conclusion

Robust recognition and pose estimation of common household objects is an important capability for any personal service robot. However, real-world scenes often contain a great deal of clutter and occlusion, making the estimation task difficult. Therefore, it is necessary to infer as much as possible from small object features, such as the oriented surface patches we considered in this work. Although our results on both
uncluttered and cluttered datasets demonstrated the applicability of our approach to 3D point cloud models, the full power of 3D oriented local feature distributions will only be seen when more feature types (based on color, edges, local image appearance, etc.) are added to the model. Then, the pose of objects with unique visible features, such as the logo on a mug, will be locked down immediately, while observations of non-unique features such as the ones considered in this chapter will still lead to a much smaller set of possible object poses that the algorithm will ultimately need to consider.

In the next chapter, we will present a full object detection system with many of these features. Although it will draw inspiration from the methods of this chapter in that it also uses the Bingham distribution to represent orientation uncertainty, it differs both in its high-level architecture and in the way it uses the Bingham—the Bingham was used in this chapter as a distribution on groups of features' orientations, while in the next chapter it is used as a distribution on individual feature orientations.
Figure 5-10: Top sample poses for 7 objects found by the MC-SCOPE algorithm for a cluttered scene. When a "good" pose sample is found within the top 20 samples, the caption under the best sample is shown in bold.
Chapter 6

A Discriminative Method for 3-D Object Detection

Full, category-level object recognition has been a long-standing goal for the AI community. Much progress has been made over the years, and accurate recognition systems can now be made for constrained environments, such as for self-driving cars, or factory assembly line robots, even though general-purpose, category-level object recognition remains elusive. In household robotics, where manipulating everyday objects is the primary goal, the robotics community has recently shifted to a much simpler goal: recognizing known object instances, and estimating their 3-D positions and orientations in space (so that the robot can manipulate them!). Even before the introduction of the Microsoft Kinect camera in 2010, much progress had already been made towards this goal—starting with the geometric, model-based object localization work in the 1980s [24, 34, 84], continuing in work on spin images in the 1990s [39], which used 3-D geometric features, and more recently, using SIFT features on image data [19]. Since 2010, progress towards object instance recognition has rapidly accelerated, as nearly every robotics research group in the world now has access to good-quality RGB-D data for the first time in history. This has given us, as object recognition researchers, a unique opportunity to finally solve one of the most pressing needs in robotics—the need for reliable object instance recognition and 3-D pose estimation.
But there are many challenges along the way. Real scenes are full of clutter and ambiguity. Many common objects do not have the unique local features which we have come to rely upon in object recognition, and even those that do may not have visible unique features from every viewpoint. Objects can be shiny, or transparent. Data from the Kinect and related sensors isn’t perfect, and their noise process can be complex to model. A complete system must combine both geometric and color information, must reason about the relationships between objects (e.g., to avoid putting two objects in the same space), and must report confidence in its predictions, including any ambiguities that it detects. Sometimes this will mean returning several different possible scene interpretations (sets of object detections), and sometimes this will just mean returning a local model of object pose uncertainty. It must also be fast. No existing system does all of these things well yet.

In this chapter, we will present a system with two main contributions that we hope will bring the field closer to its goal of reliable object instance recognition in clutter. The first contribution is a novel object alignment algorithm, called Bingham Procrustean Alignment (BPA), which uses sparse (but not necessarily unique), oriented 3-D features to very quickly align an object model with an observed scene. BPA uses point correspondences between oriented features to derive a probability distribution over possible model poses. The orientation component of this distribution, conditioned on the position, is shown to be a Bingham distribution. This result also applies to the classic problem of least-squares alignment of point sets, when point features are orientation-less, and gives a principled, probabilistic way to measure pose uncertainty in the rigid alignment problem.

The second contribution is a sequence (i.e., a cascade) of highly discriminative model placement evaluation functions, which combine many different feature types—from positions and surface normals to RGB and depth image edges—with a sophisticated sensor noise model. The combination of a robust object alignment algorithm together with improved evaluation functions leads to state-of-the-art performance on scenes with high clutter, such as in figure 6-1. The work in this chapter was originally published in the proceedings of the 2013 IEEE/RSJ International Conference
Although the object detection system in this chapter is loosely inspired by the generative methods of the previous chapter in that it also uses the Bingham distribution to represent orientation uncertainty, it differs both in its high-level architecture and in the way it uses the Bingham—the Bingham was used in the previous chapter as a distribution on groups of features’ orientations, while in this chapter it is used as a distribution on individual feature orientations. Furthermore, this chapter provides a complete, discriminative object detection algorithm, whereas the previous chapter’s methods were restricted to making initial guesses of object placements only.

6.1 Alignment

Detecting known, rigid objects in RGB-D images relies on being able to align 3-D object models with an observed scene. If alignments are inconsistent or inaccurate, detection rates will suffer. In noisy and cluttered scenes (such as shown in figure 6-1), good alignments must rely on multiple cues, such as 3-D point positions, surface normals, curvature directions, edges, and image features. Yet before the work in this thesis there was no existing alignment method (other than brute force optimization) that could fuse all of this information together in a meaningful way.

We have seen that the Bingham distribution is a good fit mathematically for the space of 3-D orientations (represented by unit quaternions). In this chapter, we derive a surprising result connecting the Bingham distribution to the classical least-squares alignment problem, which allows our new system to easily fuse information from both position and orientation information in a principled, Bayesian alignment system which we call Bingham Procrustean Alignment (BPA).

6.1.1 Background

Rigid alignment of two 3-D point sets $X$ and $Y$ is a well-studied problem—one seeks an optimal (quaternion) rotation $q$ and translation vector $t$ to minimize an alignment cost function, such as sum of squared errors between corresponding points on $X$ and
Figure 6-1: Object detections found with our system, along with the feature correspondences that BPA used to align the model. Surface features are indicated by red points, with lines sticking out of them to indicate orientations (red for normals, orange for principal curvatures). Edge features (which are orientation-less) are shown by magenta points.

Given known correspondences, $t$ and $q$ can be found in closed form with Horn's method [36]. If correspondences are unknown, the alignment cost function can be specified in terms of sum-of-squared distances between nearest-neighbor points on $X$ and $Y$, and iterative algorithms like ICP (Iterative Closest Point) are guaranteed to reach a local minimum of the cost function [9]. During each iteration of ICP, Horn's method is used to solve for an optimal $t$ and $q$ given a current set of correspondences, and then the correspondences are updated using nearest neighbors given the new pose.

ICP can be slow, because it needs to find dense correspondences between the two point sets at each iteration. Sub-sampling the point sets can improve speed, but only at the cost of accuracy when the data is noisy. Another drawback is its sensitivity to outliers—for example when it is applied to a cluttered scene with segmentation error.

Particularly because of the clutter problem, many modern approaches to alignment use sparse point sets, where one only uses points computed at especially unique keypoints in the scene. These keypoint features can be computed from either 2-D (image) or 3-D (geometry) information, and often include not only positions, but also orientations derived from image gradients, surface normals, principal curvatures, etc.
However, these orientations are typically only used in the feature matching and pose clustering stages, and are ignored during the alignment step.

Another limitation is that the resulting alignments are often based on just a few features, with noisy position measurements, and yet there is very little work on estimating confidence intervals on the resulting alignments. This is especially difficult when the features have different noise models— for example, a feature found on a flat surface will have a good estimate of its surface normal, but a high variance principal curvature direction, while a feature on an object edge may have a noisy normal, but precise principal curvature. Ideally, we would like to have a posterior distribution over the space of possible alignments, given the data, and we would like that distribution to include information from feature positions and orientation measurements, given varying noise models.

As we will see in the next section, a full joint distribution on $t$ and $q$ is difficult to obtain. However, in the original least-squares formulation, it is possible to solve for the optimal $t^*$ independently of $q^*$, simply by taking $t^*$ to be the translation which aligns the centroids of $X$ and $Y$. Given a fixed $t^*$, solving for the optimal $q^*$ then becomes tractable. In a Bayesian analysis of the least-squares alignment problem, we seek a full distribution on $q$ given $t$, not just the optimal value, $q^*$. That way we can assess the confidence of our orientation estimates, and fuse $p(q|t)$ with other sources of orientation information, such as from surface normals.

Remarkably, given the common assumption of independent, isotropic Gaussian noise on position measurements (which is implicit in the classical least-squares formulation), we can show that $p(q|t)$ is a quaternion Bingham distribution. This result makes it easy to combine the least-squares distribution on $q|t$ with other quaternion Bingham distributions from feature orientations (or prior distributions), since (as we have seen in previous chapters) the quaternion Bingham is a useful distribution for encoding uncertainty on 3-D rotations represented as unit quaternions.

The mode of the least-squares Bingham distribution on $q|t$ will be exactly the same as the optimal orientation $q^*$ from Horn’s method. When other sources of orientation information are available, they may bias the distribution away from $q^*$. 

97
Thus, it is important that the concentration (inverse variance) parameters of the Bingham distributions are accurately estimated for each source of orientation information, so that this bias is proportional to confidence in the measurements. (See section 6.2.2 for an example.)

We use our new alignment method, BPA, to build an object detection system for known, rigid objects in cluttered RGB-D images. Our system combines information from surface and edge feature correspondences to improve object alignments in cluttered scenes (as shown in figure 6-1), and achieves state-of-the-art recognition performance on both an existing Kinect data set [2], and on a new data set containing far more clutter and pose variability than any existing data set.

6.2 Bingham Procrustean Alignment

Given two 3-D point sets $X$ and $Y$ in one-to-one correspondence, we seek a distribution over the set of rigid transformations of $X$, parameterized by a (quaternion) rotation $q$ and a translation vector $t$. Assuming independent Gaussian noise models on deviations between corresponding points on $Y$ and (transformed) $X$, the condi-

---

1Most existing data sets for 3-D cluttered object detection have very limited object pose variability (most of the objects are upright), and objects are often easily separable and supported by the same flat surface.
tional distribution \( p(q|t, X, Y) \) is proportional to \( p(X, Y|q, t)p(q|t) \), where

\[
    p(X, Y|q, t) = \prod_i p(x_i, y_i|q, t)
    = \prod_i N(Q(x_i + t) - y_i; 0, \Sigma_i)
\]

(6.1)

(6.2)

given that \( Q \) is \( q \)'s rotation matrix, and covariances \( \Sigma_i \).

Given isotropic noise models\(^2\) on point deviations (so that \( \Sigma_i \) is a scalar times the identity matrix), \( p(x_i, y_i|q, t) \) reduces to a 1-D Gaussian PDF on the distance between \( y_i \) and \( Q(x_i + t) \), yielding

\[
    p(x_i, y_i|q, t) = N(\|Q(x_i + t) - y_i\|; 0, \sigma_i)
    = N(d_i; 0, \sigma_i)
\]

where \( d_i \) depends on \( q \) and \( t \).

Now consider the triangle formed by the origin (center of rotation), \( Q(x_i + t) \) and \( y_i \), as shown on the left of figure 6-3. By the law of cosines, the squared-distance between \( Q(x_i + t) \), and \( y_i \) is \( d^2 = a^2 + b^2 - 2ab\cos(\theta) \), which only depends on \( q \) via the angle \( \theta \) between the vectors \( Q(x_i + t) \) and \( y_i \). (We drop the \( i \)-subscripts on \( d, a, b, \) and \( \theta \) for brevity.) We can thus replace \( p(x_i, y_i|q, t) \) with

\[
    p(x_i, y_i|\theta, t) = \frac{1}{Z} \exp \left\{ \frac{ab\cos(\theta)}{\sigma^2} \right\}
\]

(6.3)

which has the form of a Von-Mises distribution on \( \theta \).

Next, we need to demonstrate how \( \theta \) depends on \( q \). Without loss of generality, assume that \( y_i \) points along the axis \((1, 0, 0)\). When this is not the case, the Bingham distribution over \( q \) which we derive below can be post-rotated by any quaternion which takes \((1, 0, 0)\) to \( y_i/\|y_i\| \).

Clearly, there will be a family of \( q \)'s which rotate \( x_i + t \) to form an angle of \( \theta \) with \( y_i \), since we can compose \( q \) with any rotation about \( x_i + t \) and the resulting angle with \( y_i \) will still be \( \theta \). To demonstrate what this family is, we first let \( s \) be a unit quaternion which rotates \( x_i + t \) onto \( y_i \)'s axis, and let \( x'_i = S(x_i + t) \), where \( S \) is \( s \)'s

\(^2\)This is the implicit assumption in the least-squares formulation.
rotation matrix. Then, let \( \mathbf{r} \) (with rotation matrix \( R \)) be a quaternion that rotates \( \mathbf{x}' \) to \( Q(\mathbf{x}_i + t) \), so that \( \mathbf{q} = \mathbf{r} \circ \mathbf{s} \). Because \( \mathbf{y}_i \) and \( \mathbf{x}' \) point along the axis \((1, 0, 0)\), the first column of \( R \), \( \hat{n}_1 \), will point in the direction of \( Q(\mathbf{x}_i + t) \), and form an angle of \( \theta \) with \( \mathbf{y}_i \), as shown on the right side of figure 6-3. Thus, \( \hat{n}_1 \cdot (1, 0, 0) = \hat{n}_{11} = \cos \theta \).

The rotation matrix of quaternion \( \mathbf{r} = (r_1, r_2, r_3, r_4) \) is

\[
R = \begin{bmatrix}
-1 & -2 & -2 & -2 \\
-2 & 1 & -2 & -2 \\
-2 & -2 & 1 & -2 \\
-2 & -2 & -2 & 1
\end{bmatrix}
\]

Therefore, \( \cos \theta = \hat{n}_{11} = r_1^2 + r_2^2 - r_3^2 - r_4^2 = 1 - 2r_3^2 - 2r_4^2 \).

We can now make the following claim about \( p(\mathbf{x}_i, \mathbf{y}_i | \mathbf{q}, t) \):

**Claim 6.1.** Given that \( \mathbf{y}_i \) lies along the \((1, 0, 0)\) axis, then the probability density \( p(\mathbf{x}_i, \mathbf{y}_i | \mathbf{q}, t) \) is proportional to a quaternion Bingham density on \( \mathbf{q} \) with parameters

\[
\Lambda = \left( -\frac{2ab}{\sigma^2}, -\frac{2ab}{\sigma^2}, 0 \right) \quad \text{and} \quad V = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \circ \mathbf{s} = W \circ \mathbf{s}
\]

where \( \circ \) indicates column-wise quaternion multiplication.
Proof. The Bingham density in claim 6.1 is given by

\[
p(q|\Lambda, V) = \frac{1}{F} \exp \sum_{j=1}^{3} \lambda_j ((w_j \circ s)^T q)^2
\]

(6.4)

\[
= \frac{1}{F} \exp \left\{ -\frac{2ab}{\sigma^2} r_3^2 - \frac{2ab}{\sigma^2} r_4^2 \right\}
\]

(6.5)

\[
= \frac{1}{F} \exp \left\{ \frac{ab \cos \theta}{\sigma^2} \right\}
\]

(6.6)

since \((w_j \circ s)^T q = w_j^T (q \circ s^{-1}) = w_j^T r\), and \(\cos \theta = 1 - 2r_3^2 - 2r_4^2\). Since (6.6) is proportional to (6.3), we conclude that \(p(q|\Lambda, V) \propto p(x_i, y_i|q, t)\), as claimed.

Claim 6.2. Let \(s'\) be a quaternion that rotates \((1, 0, 0)\) onto the axis of \(y_i\) (for arbitrary \(y_i\)). Then the probability density \(p(x_i, y_i|q, t)\) is proportional to a Bingham density on \(q\) with parameters

\[
\Lambda = \left( -\frac{2ab}{\sigma^2}, -\frac{2ab}{\sigma^2}, 0 \right) \quad \text{and} \quad V = s' \circ \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \circ s
\]

(6.7)

where “\(\circ\)” indicates column-wise quaternion multiplication.

As explained above, the distribution on \(q\) from claim 6.1 must simply be post-rotated by \(s'\) when \(y_i\) is not aligned with the \((1, 0, 0)\) axis. The proof is left to the reader. Putting it all together, we find that

\[
p(q|t, X, Y) \propto \prod_i \text{Bingham}(q; \Lambda_i, V_i) \cdot p(q|t)
\]

(6.8)

\[
= \text{Bingham}(q; \hat{\Lambda}, \hat{V}) \cdot p(q|t)
\]

(6.9)

where \(\Lambda_i\) and \(V_i\) are taken from claim 6.2, and where \(\hat{\Lambda}\) and \(\hat{V}\) are computed from the formula for multiplication of Bingham PDFs (section 2.11).

Equation 6.9 tells us that, in order to update a prior on \(q\) given \(t\) after data points \(X\) and \(Y\) are observed, one must simply multiply the prior by an appropriate Bingham term. Therefore, assuming a Bingham prior over \(q\) given \(t\) (which includes the uniform distribution), the conditional posterior, \(p(q|t, X, Y)\) is the PDF of a Bingham distribution.
To demonstrate this fact, we relied only upon the assumption of independent isotropic Gaussian noise on position measurements, which is exactly the same assumption made implicitly in the least-squares formulation of the optimal alignment problem. This illustrates a deep and hitherto unknown connection between least-squares alignment and the quaternion Bingham distribution, and paves the way for the fusion of orientation and position measurements in a wide variety of applications.

6.2.1 Incorporating Orientation Measurements

Now that we have shown how the orientation information from the least-squares alignment of two point sets $X$ and $Y$ is encoded as a Bingham distribution, it becomes trivial to incorporate independent orientation measurements at some or all of the points, provided that the orientation noise model is Bingham. Given orientation measurements $(O_X, O_Y)$,

$$p(q|t, X, Y, O_X, O_Y)$$

$$\propto p(X, Y, O_X, O_Y|q, t) \cdot p(q|t)$$

$$= p(X, Y|q, t) \cdot p(O_X, O_Y|q, t) \cdot p(q|t).$$

Similarly as in equation 6.9, $p(O_X, O_Y|q, t)$ is the product of quaternion Bingham distributions from corresponding orientation measurements in $(O_X, O_Y)$, and so the entire posterior $p(q|t, X, Y, O_X, O_Y)$ is Bingham (provided as before that the prior $p(q|t)$ is Bingham).

6.2.2 Estimating the Uncertainty on Feature Orientations

When we extract surface features from depth images, we estimate their 3-D orientations from their normals and principal curvature directions by computing the rotation matrix $R = [n \ p \ p']$, where $n$ is the normal vector, $p$ is the principal curvature vector, and $p'$ is the cross product of $n$ and $p$. We take the quaternion associated with this rotation matrix to be the feature’s estimated orientation.

These orientation estimates may be incredibly noisy, not only due to typical sensing noise, but because on a flat surface patch the principal curvature direction is
undefined and will be chosen completely at random. Therefore it is extremely useful to have an estimate of the uncertainty on each feature orientation that allows for the uncertainty on the normal direction to differ from the uncertainty on the principal curvature direction. Luckily, the Bingham distribution is well suited for this task.

To form such a Bingham distribution, we take the quaternion associated with $R$ to be the mode of the distribution, which is orthogonal to all the $v_i$ vectors. Then, we set $v_3$ to be the quaternion associated with $R' = [n - p - p']$, which has the same normal as the mode, but reversed principal curvature direction. In quaternion form, reversing the principal curvature is equivalent to the mapping:

$$(q_1, q_2, q_3, q_4) \rightarrow (-q_2, q_1, q_4, -q_3).$$

We then take $v_1$ and $v_2$ to be unit vectors orthogonal to the mode and $v_3$ (and each other). Given these $v_i$'s, the concentration parameters $\lambda_1$ and $\lambda_2$ penalize deviations in the normal vector, while $\lambda_3$ penalizes deviations in the principal curvature direction. Therefore, we set $\lambda_1 = \lambda_2 = \kappa$ (we use $\kappa = -100$ in all our experiments), and we use the heuristic $\lambda_3 = \max\{10(1 - c_1/c_2), \kappa\}$, where $c_1/c_2$ is the ratio of the principal curvature eigenvalues. When the surface is completely flat, $c_1 = c_2$ and $\lambda_3 = 0$, so the resulting Bingham distribution will be completely uniform in the principal curvature direction. When the surface is highly curved, $c_1 \gg c_2$, so $\lambda_3$ will equal $\kappa$, and deviations in the principal curvature will be penalized just as much as deviations in the normal.

### 6.2.3 The Alignment Algorithm

To incorporate our Bayesian model into an iterative ICP-like alignment algorithm, one could solve for the maximum a posteriori (MAP) position and orientation by maximizing $p(q, t | X, Y, \ldots)$ with respect to $q$ and $t$. However, it is often more desirable to draw samples from this posterior distribution in order to maximize the coverage of the proposed alignments.

---

3 The principal curvature direction is computed with an eigen-decomposition of the covariance of normal vectors in a neighborhood about the feature.
The joint posterior distribution \( p(q, t|Z) \)—where \( Z \) contains all the measurements \((X, Y, O_X, O_Y, \ldots)\) —can be broken up into \( p(q|t, Z)p(t|Z) \). Unfortunately, writing down a closed-form distribution for \( p(t|Z) \) is difficult. But sampling from the joint distribution is easy with an importance sampler, by first sampling \( t \) from a proposal distribution—for example, a Gaussian centered on the optimal least-squares translation (that aligns the centroids of \( X \) and \( Y \))—then sampling \( q \) from \( p(q|t, Z) \), and then weighting the samples by the ratio of the true posterior (from equation 6.2) and the proposal distribution (e.g., Gaussian times Bingham).

We call this sampling algorithm Bingham Procrustean Alignment (BPA). It takes as input a set of (possibly oriented) features in one-to-one correspondence, and returns samples from the distribution over possible alignments. In section 6.6, we will show how BPA can be incorporated into an iterative alignment algorithm that re-computes feature correspondences at each step and uses BPA to propose a new alignment given the correspondences.

### 6.3 A New Characterization of the Quaternion Bingham Distribution

If any two sets of 3-D points in one-to-one correspondence (endowed with a Gaussian noise model on the error between corresponding points) defines a quaternion Bingham distribution on the space of possible 3-D orientations of one set of points with respect to the other, then what about the reverse statement? Does every quaternion Bingham distribution have a pair of point sets and a Gaussian noise model that generates it? In other words, can one generate all possible quaternion Bingham distributions (covering the space of possible \( \Lambda \)'s and \( V \)'s) by moving around the points? And if the answer is yes, then how many pairs of points does it take to cover quaternion Bingham space?

As it turns out, the answers to those questions are “yes” and “3”.

**Theorem 6.3.** For every quaternion Bingham distribution (with parameters \( \Lambda \) and \( V \)), there exists a set of three pairs of points \( \{ (x_1, y_1), (x_2, y_2), (x_3, y_3) \} \) with unit
error variance \((\sigma^2 = 1)\) that generates the given quaternion Bingham distribution.

**Proof.** Given quaternion Bingham parameters \(\lambda_1 \leq \lambda_2 \leq \lambda_3 \leq 0\) and \(V = [v_1 \ v_2 \ v_3]\) with \(V^T V = I\). Each pair of points \((x_i, y_i)\) will generate a Bingham with PDF of the form \(\frac{1}{k} \exp\{x^T (\alpha_i (w_{11} w_{11}^T + w_{12} w_{12}^T)) x\}\). When the three Binghams are combined, their PDFs are multiplied, yielding an exponent of the form \(x^T C x\), where

\[
C = \alpha_1 (w_{11} w_{11}^T + w_{12} w_{12}^T) + \alpha_2 (w_{21} w_{21}^T + w_{22} w_{22}^T) + \alpha_3 (w_{31} w_{31}^T + w_{32} w_{32}^T).
\]

Now let

\[
\begin{align*}
\alpha_1 &= \frac{1}{2} (\lambda_1 + \lambda_2 - \lambda_3) \\
\alpha_2 &= \frac{1}{2} (\lambda_1 - \lambda_2 + \lambda_3) \\
w_{11} &= v_1 \\
w_{12} &= v_2 \\
w_{21} &= v_1 \\
w_{22} &= v_3.
\end{align*}
\]

We know that \(\alpha_1 \leq \alpha_2 \leq 0\) since \(\lambda_1 \leq \lambda_2 \leq \lambda_3 \leq 0\). For \(\alpha_3, w_{31}, \) and \(w_{32}\), there are two cases to consider (to ensure that \(\alpha_3\) is non-positive).

**Case 1:** If \(\lambda_1 - \lambda_2 - \lambda_3 \leq 0\), then let

\[
\begin{align*}
\alpha_3 &= \frac{1}{2} (\lambda_1 - \lambda_2 - \lambda_3) \\
w_{31} &= v_1 \\
w_{32} &= v_4
\end{align*}
\]

where \(v_4\) completes the basis with \((v_1, v_2, v_3)\). Then the \(C\) matrix of the combined Bingham is

\[
C = \alpha_1 (v_1 v_1^T + v_2 v_2^T) + \alpha_2 (v_1 v_1^T + v_3 v_3^T) + \alpha_3 (v_1 v_1^T + v_4 v_4^T). \tag{6.10}
\]

The eigenvectors of \(C\) are \(v_1, v_2, v_3, \) and \(v_4, \) with eigenvalues \(\lambda_1', \lambda_2', \lambda_3', \lambda_4'\) given

\(^4\)These eigenvectors and eigenvalues can be easily verified by multiplying \(C\) by each \(v_i\) and using the fact that the \(v_i\)'s are orthonormal.
by
\[
\begin{align*}
\lambda'_1 &= \alpha_1 + \alpha_2 + \alpha_3 = \frac{1}{2}(3\lambda_1 - \lambda_2 - \lambda_3), \\
\lambda'_2 &= \alpha_1 = \frac{1}{2}(\lambda_1 + \lambda_2 - \lambda_3), \\
\lambda'_3 &= \alpha_2 = \frac{1}{2}(\lambda_1 - \lambda_2 + \lambda_3), \quad \text{and} \\
\lambda'_4 &= \alpha_3 = \frac{1}{2}(\lambda_1 - \lambda_2 - \lambda_3),
\end{align*}
\]
respectively. Now, to put the resultant Bingham into standard form, we eliminate the fourth spectral component by subtracting \( \lambda'_4 \) from each eigenvalue, yielding
\[
\begin{align*}
\lambda'_1 - \lambda'_4 &= \lambda_1 \\
\lambda'_2 - \lambda'_4 &= \lambda_2 \\
\lambda'_3 - \lambda'_4 &= \lambda_3 \\
\lambda'_4 - \lambda'_4 &= 0,
\end{align*}
\]
as desired.

**Case 2:** If \( \lambda_1 - \lambda_2 - \lambda_3 > 0 \), then let
\[
\begin{align*}
\alpha_3 &= \frac{1}{2}(-\lambda_1 + \lambda_2 + \lambda_3) \\
\mathbf{w}_{31} &= \mathbf{v}_2 \\
\mathbf{w}_{32} &= \mathbf{v}_3.
\end{align*}
\]
The \( C \) matrix of the combined Bingham then becomes
\[
C = \alpha_1(\mathbf{v}_1\mathbf{v}_1^T + \mathbf{v}_2\mathbf{v}_2^T) + \alpha_2(\mathbf{v}_1\mathbf{v}_1^T + \mathbf{v}_3\mathbf{v}_3^T) + \alpha_3(\mathbf{v}_2\mathbf{v}_2^T + \mathbf{v}_3\mathbf{v}_3^T) \quad (6.11)
\]
with eigenvectors \( \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \) and \( \mathbf{v}_4, \) and eigenvalues
\[
\begin{align*}
\lambda'_1 &= \alpha_1 + \alpha_2 = \lambda_1 \\
\lambda'_2 &= \alpha_1 + \alpha_3 = \lambda_2 \\
\lambda'_3 &= \alpha_2 + \alpha_3 = \lambda_3 \\
\lambda'_4 &= 0
\end{align*}
\]
106
as desired.

We have just proven that the above \( \alpha \)'s and \( w \)'s will generate the correct Bingham parameters \((\Lambda, V)\). So all that remains to be shown is that one can find pairs of points \((x_i, y_i)\) to generate the component Bingham with those \( \alpha \)'s and \( w \)'s.

To generate a Bingham with first two axes \( w_1, w_2 \), the quaternions \( s \) (that rotates \( x_i \) onto \([1 \ 0 \ 0]^T\)) and \( s' \) (that rotates \([1 \ 0 \ 0]^T\) onto \( y_i \)) must satisfy:

\[
s' \circ (0, 0, 1, 0) \circ s = w_1
\]
\[
s' \circ (0, 0, 0, 1) \circ s = w_2.
\]

Thus,

\[
s' \circ (-s_3, s_4, s_1, -s_2) = w_1 \tag{6.12}
\]
\[
s' \circ (-s_4, -s_3, s_2, s_1) = w_2. \tag{6.13}
\]

Left-multiplying by \((s')^{-1}\) and substitution for \( s_1, s_2, s_3, \) and \( s_4 \) yields a system of four linear equations in four unknowns for \( s' \):

\[
\begin{bmatrix}
w_{11} - w_{22} & w_{12} + w_{21} & w_{13} + w_{24} & w_{14} - w_{23} \\
w_{12} + w_{21} & w_{22} - w_{11} & w_{23} - w_{14} & w_{13} + w_{24} \\
w_{13} - w_{24} & w_{14} + w_{23} & -w_{11} - w_{22} & w_{21} - w_{12} \\
w_{14} + w_{23} & w_{24} - w_{13} & w_{12} - w_{21} & -w_{11} - w_{22}
\end{bmatrix}
\begin{bmatrix}
s'_1 \\
s'_2 \\
s'_3 \\
s'_4
\end{bmatrix}
= \begin{bmatrix}0 \\
0 \\
0 \\
0\end{bmatrix} \tag{6.14}
\]

The matrix on the left can be shown to have determinant zero (using the fact that \( w_1 \) and \( w_2 \) are orthonormal unit vectors), which means that its nullspace is a subspace of rank at least one. Therefore, a solution \( s' \) can always be found of unit length. Once a \( s' \) is found, we can plug it into equation \( 6.12 \) to solve for \( s \).

Finally, given \( s \) and \( s' \), let \( S \) and \( S' \) be their \( 3 \times 3 \) rotation matrices. Then let \( y_i \) equal the first column of \( S' \), so that \( S' \begin{bmatrix}1 \\ 0 \end{bmatrix} = y_i \), and let \( x_i \) equal \(-\alpha_i/2\) times the first row of \( S \), so that \( S^T \begin{bmatrix}-\alpha_i/2 \\ 0 \end{bmatrix} = x_i \), which means that \( \begin{bmatrix}-\alpha_i/2 \\ 0 \end{bmatrix} = Sx_i \) (since \( S \) is orthogonal). Plugging \( a = 1, b = -\alpha_i/2, \) and \( \sigma = 1 \) into equation \( 6.7 \) yields a component Bingham with shared concentration parameter \( \alpha_i \) and direction vectors \( w_1 \) and \( w_2 \), as desired. \( \square \)
6.4 Building Noise-Aware 3-D Object Models

Our first step in building a system to detect known, rigid objects—such as the ones in figure 6-4—is to build complete 3-D models of each object. However, the end goal of model building is not just to estimate an object’s geometry correctly. Rather, we seek to predict what an RGB-D sensor would see, from every possible viewing angle of the object. To generate such a predictive model, we will estimate both the most likely observations from each viewing angle, and also the degree of noise predicted in those measurements. That way, our detection system will realize that depth measurements near object boundaries, on reflective surfaces, or on surfaces at a high oblique angle with respect to the camera, are less reliable than front-on measurements of non-reflective, non-edge surface points.

In our model-building system, we place each object on a servo-controlled turntable in 2-3 resting positions and collect RGB-D images from a stationary Kinect sensor at $10^\circ$ turntable increments, for a total of 60-90 views. We then find the turntable plane in the depth images (using RANSAC), and separate object point clouds (on top of the turntable) from the background. Next we align each set of 30 scans (taken of the object in a single resting position) by optimizing for the 2-D position of the turntable’s
center of rotation, with respect to an alignment cost function that measures the sum-of-squared nearest-neighbor distances from each object scan to every other scan. We then use another optimization to solve for the 6-dof translation + rotation that aligns the 2-3 sets of scans together into one, global frame of reference\(^5\).

After the object scans are aligned, we compute their surface normals, principal curvatures, and FPFH features \([73]\), and we use the the ratio of principal curvatures to estimate the (Bingham) uncertainty on the quaternion orientation defined by normals and principal curvature directions at each point\(^6\). We then use ray-tracing to build a 3-D occupancy grid model, where in addition to the typical probability of occupancy, we also store each 3-D grid cell’s mean position and normal, and variance on the normals in that cell\(^7\). We then threshold the occupancy grid at an occupancy probability of 0.5, and remove interior cells (which cannot be seen from any viewing angle) to obtain a full model point cloud, with associated normals and normal variance estimates. We also compute a distance transform of this model point cloud, by computing the distance from the center of each cell in the occupancy grid to the nearest model point (or zero if the cell contains a model point).

Next, for a fixed set of 66 viewing angles across the view-sphere, we estimate range edges—points on the model where there is a depth discontinuity in the predicted range image seen from that view angle. We also store the minimum distance from each model point to a range edge for each of the 66 viewing angles. Using these view-dependent edge distances, along with the angles between surface normals and viewpoints, we fit sigmoid models across the whole data set to estimate the expected noise on range measurements and normal estimates as functions of (1) edge distance, and (2) surface angle, as shown in figure 6-5.

---

\(^5\)In ongoing work, we are working to use BPA in this step.

\(^6\)The idea is to capture the orientation uncertainty on the principal curvature direction by measuring the “flatness” of the observed surface patch; see section 6.2.2 for details.

\(^7\)In fact, we store two “view-buckets” per cell, each containing an occupancy probability, a position, a normal, and a normal variance, since on thin objects like cups and bowls, there may be points on two different surfaces which fall in the same grid cell.
Figure 6-5: Our noise models predict range and normal errors (standard deviations) as functions of surface angle and edge distance (both with respect to the viewpoint).

6.5 Learning Discriminative Feature Models for Detection

Similarly to other recent object detection systems, our system computes a set of feature model-placement-score functions, in order to evaluate how well a given model placement hypothesis fits the scene according to different features, such as depth measurements, surface normals, edge locations, etc. In our early experiments with object detection using the generative object models in the previous section, the system was prone to make mis-classification errors, because some objects scored consistently higher on certain feature scores (presumably due to training set bias). Because of this problem, we trained discriminative, logistic regression models on each of the score components using the turntable scans with true model placements as positive training examples and a combination of correct object / wrong pose and wrong object / aligned pose as negative examples. Alignments of wrong objects were found by running the full object detection system (from the next section) with the wrong object on the turntable scans. By adding an (independent) discriminative layer to each of the feature score types, we were able to boost the classification accuracy of our system considerably.
6.6 Detecting Single Objects in Clutter

The first stages of our object detection pipeline are very similar to many other state-of-the-art systems for 3-D object detection, with the exception that we rely more heavily on edge information. We are given as input an RGB-D image, such as from a Kinect. If environmental information is available, the image may be pre-processed by another routine to crop the image to an area of interest, and to label background pixels (e.g., belonging to a supporting surface).

As illustrated in figure 6-6, our algorithm starts by estimating a dense set of surface normals on the 3-D point cloud derived from the RGB-D image. From these surface normals, it estimates principal curvatures and FPFH features. In addition, it finds and labels three types of edges: range edges, image edges, and curvature edges—points in the RGB-D image where there is a depth discontinuity, an image intensity discontinuity, or high negative curvature. This edge information is converted into an edge image, which is formed from a spatially blurred, weighted average of the three edge pixel masks. Intuitively, this edge image is intended to capture the relative likelihood that each point in the image is part of an object boundary. Then, the algorithm uses k-means to over-segment the point cloud based on positions, normals, and spatially-blurred colors (in CIELAB space) into a set of 3-D super-pixels.

Next, the algorithm samples possible oriented feature correspondences from the scene to the model. Then, for each correspondence, a candidate object pose is gen-

---

8 We use the Canny edge detector to find image edges.

9 We currently use only FPFH correspondences in the first stage of detection as we did not find the addition of other feature types, such as SIFT [57] or SHOT [86], to make any difference in our
Figure 6-7: Examples of objects correctly aligned by BPA with only two correspondences.

iterated using BPA. Given a set of sampled model poses from single correspondences, we then reject samples for which more than 20% of a subset of 500 randomly selected model points project into free space—places where the difference between observed range image depth and predicted model depth is above $5\text{cm}$. Next, we run a pose clustering stage, where we group correspondences together whose sampled object poses are within $2.5\text{cm}$ and $\pi/16$ radians of one another. After pose clustering, we reject any sample with less than two correspondences, then re-sample object poses with BPA.

At this stage, we have a set of possible model placement hypotheses, with at least two features correspondences each. Because BPA uses additional orientation information, two correspondences is often all it takes to lock down a very precise estimate of an object’s pose when the correspondences are correct (Figure 6-7).

We proceed with a second round of model placement validation and rejection, this time using a scoring function that includes (1) range and normal differences, which are computed by projecting a new subset of 500 randomly-selected model points into the observed range image, (2) visibility—the ratio of model points in the subset that are unoccluded, (3) edge likelihoods, computed by projecting the model’s edge points from the closest stored viewpoint into the observed edge image, and (4) edge visibility—the ratio of edge points that are unoccluded. Each of the feature score components is computed as a truncated (so as not to over-penalize outliers), average log-likelihood of observed features given model feature distributions. For score components (1) and detection rates.
(3), we weight the average log-likelihood by visibility probabilities, which are equal to 1 if predicted depth < observed depth, and $N(\Delta \text{depth}; 0, \sigma_{\text{vis}})/N(0; 0, 1)$ otherwise.$^{10}$

After rejecting low-scoring samples in round 2, we then refine alignments by repeating the following three steps:

1. Assign observed super-pixel segments to the model.
2. Align model to the segments with BPA.
3. Accept the new alignment if the round 2 model placement score has improved.

In step (1), we sample a set of assigned segments according to the probability that each segment belongs to the model, which we compute as the ratio of segment points (sampled uniformly from the segment) that are within 1cm in position and $\pi/16$ radians in normal orientation from the closest model point. In step (2), we randomly extract a subset of 10 segment points from the set of assigned segments, find nearest neighbor correspondences from the keypoints to the model using the model distance transform, and then use BPA to align the model to the 10 segment points. Segment points are of two types—surface points and edge points. We only assign segment edge points to model edge points (as predicted from the given viewpoint), and surface points to surface points. Figures 6-1 and 6-8 show examples of object alignments found after segment alignment, where red points (with red normal vectors and orange principal curvature vectors sticking out of them) indicate surface point correspondences, and magenta points (with no orientations) are the edge point correspondences.$^{11}$

After round 2 alignments, the system removes redundant samples (with the same or similar poses), and then rejects low-scoring samples using the scores found at the end of the segment alignment process. Then, it performs a final, gradient-based alignment, which optimizes the model poses with a local hill-climbing search to directly maximize model placement scores. Since this alignment step is by far the slowest, it is critical that the system has performed as much alignment with BPA and has rejected as many low-scoring samples as possible, to reduce the computational burden.

$^{10}$We use $\sigma_{\text{vis}} = 1cm$ in all of our experiments.

$^{11}$In future work, we plan to incorporate edge orientations as well.
Finally, the system performs a third round of model placement evaluation, then sorts the pose samples by score and returns them. This third round of scoring includes several additional feature score components:

- Random walk score—starting from an observed point corresponding to the model, take a random walk in the edge image (to stay within predicted object boundaries), then measure the distance from the new observed point to the closest model point.

- Occlusion edge score—evaluate how well model occlusion edges (where the model surface changes from visible to occluded) fits the observed edge image.

- FPFH score—computes how well observed and model FPFH features match.

- Segment score—computes distances from segment points to nearest model points.

- Segment affinity score—measures how consistent the set of assigned segments is with respect to predicted object boundaries (as measured by the observed edge image, and by differences in segment positions and normals).

The random walk, occlusion edge, segment, and segment affinity scores all aim to evaluate how well the model placement fits with object boundary cues in the scene, while the FPFH score evaluates the similarity between local shapes on the model and the scene.

### 6.7 Detecting Multiple Objects in Clutter

To detect multiple objects in a scene, we run the individual object detector from the previous section to obtain the 50 best model placements for each model, along with their individual scores. Then, following Aldoma et al [2], we use simulated annealing to optimize the subset of model placements (out of $50 \times N$ for $N$ models) according to a multi-object-placement score, which we compute as a weighted sum of the following score components: (1) the average of single object scores, weighted by the number of observed points each object explains, (2) the ratio of explained / total observed
Figure 6-8: Object detections found with our system, along with the feature correspondences that BPA used to align the model. Surface features are indicated by red points, with lines sticking out of them to indicate orientations (red for normals, orange for principal curvatures). Edge features (which are orientation-less) are shown by magenta points.

points, and (3) a small penalty for the total number of detected objects. We also keep track of the top 100 multi-object-placement samples found during optimization, so we can return a set of possible scene interpretations to the user (in the spirit of interpretation tree methods [35]). This is particularly useful for robot vision systems because they can use tracking, prior knowledge, or other sensory input (like touch) to provide additional validation of model placements, and we don't want detections from a single RGB-D image to filter out possible model placements prematurely.
6.8 Experimental Results

We tested our object detection system on two Kinect-based data sets—the *Kinect* data set from Aldoma et al. [2] containing 35 models and 50 scenes, and a new, more difficult data set with many more occlusions and object pose variations that we collected for this experiment which we will refer to as *Clutter*, which contains 18 models and 30 scenes (Figure 6-9). We used the same parameters (score component weights, number of samples, etc.) on both data sets. In table 6.1, we compare the precision and recall of the top scene interpretations (multi-object-placement samples) of our method against Aldoma et al. on both data sets\(^{12}\).

Our algorithm (with BPA) achieves state-of-the art recall performance on both data sets. When multiple scene interpretations are considered, we achieve even higher recall rates (Table 6.2). Our precisions are similar to the baseline method (slightly higher on *Clutter*, slightly lower on *Kinect*). We were unable to train discriminative feature models on the *Kinect* data set, because the original training scans were not

\(^{12}\)Since neither ours nor the baseline method uses colors in their object models, we considered a model placement “correct” for the *Clutter* data set if it was within a threshold of the correct pose (2.5cm, π/16 radians) with respect to the model’s symmetry group. For example, we don’t penalize flipping boxes front-to-back or top-to-bottom, since the resulting difference in object appearance is purely non-geometric. For the *Kinect* data set, we used the same correctness measure as the baseline method (RMSE between model in true pose and estimated pose), with a threshold of 1cm.
Table 6.1: A comparison of precision and recall.

<table>
<thead>
<tr>
<th># samples</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>5</th>
<th>10</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>recall</td>
<td>73.3</td>
<td>77.5</td>
<td>80.0</td>
<td>80.8</td>
<td>83.3</td>
<td>84.2</td>
</tr>
</tbody>
</table>

Table 6.2: Recall on the Clutter data set as a function of the number of scene interpretation samples.

provided. Training on scenes that are more similar to the cluttered test scenes is also likely to improve precision on the Clutter data set, since each training scan contained only one, fully-visible object.

For our experiments, we implemented our algorithm in C++ and ran them on a single 64-bit, 2.4 GHz Intel Xeon processor. On a single CPU, each scene took 1-2 minutes to process, compared to 10.4 seconds per scene for Aldoma’s system, which ran in parallel on multiple CPUs [2]. We are now in the process of porting our code to the GPU—our current prototype system can detect single objects in about 1 second on an NVIDIA GeForce GTX 580 with 512 cores.

### 6.8.1 BPA vs. ICP

We evaluated the benefits of our new alignment method, BPA, in two ways. First, we compared it to ICP by replacing the BPA alignment step in round 2 with an ICP alignment step\(^{13}\). This resulted in a drop of 10% in both precision and recall on the Clutter data. The lesson is that using a bad alignment algorithm can actually hurt object detection performance rather than help!

For a second test of BPA, we initialized 50 model placements by adding random

\(^ {13}\)Both ICP and BPA used the same point correspondences; the only difference was that BPA incorporated point feature orientations, while ICP used only their positions.
Gaussian noise to the ground truth poses for each object in each scene of the Cluster data set. Then, we ran BPA and ICP for 20 iterations on each of the model placements. We then computed the average of the minimum pose errors in each alignment trial, where the minimum at time $t$ in a given trial is computed as the minimum pose error from step 1 to step $t$. (The justification for this measure is that this is approximately what the "accept if score improves" step of round 2 is doing.) As shown in figure 6-10, the pose errors decrease much faster in BPA.

Figure 6-10: Comparing BPA with ICP. (Left) The average of the minimum position errors in each alignment trial. (Right) The average of the minimum orientation errors in each alignment trial.

6.9 Related Work

Since the release of the Kinect in 2010, much progress has been made on 3-D object detection in cluttered RGB-D scenes. The two most successful systems to date are Aldoma et. al [2] and Tang et. al [81]. Aldoma et. al's system is purely geometric, and uses SHOT features [86] for model-scene correspondences. It relies heavily on pose clustering of feature correspondences to suggest model placements. \(^{14}\) The main

\(^{14}\)In other words, we repeated the alignment step of round 2 twenty times, regardless of whether the total score improved.

\(^{15}\)This is essentially a sparse version of the Hough transform [7], which is limited by the number of visible features on an object, and is why their recall rates tend to be lower than in our system for objects that are heavily occluded.
contribution of Aldoma et. al's system is that they jointly optimize multiple model placements for consistency, which inspired our own multiple object detection system.

Tang et. al's detection system uses both geometry and image features, and placed first in the ICRA 2011 Solutions in Perception instance recognition challenge. Their system relies heavily on being able to segment objects in the scene from one another, and most of the effort is spent on combining geometry and image features for classification of scene segments. It is unclear how well the system would perform if such segmentations are not easy to obtain, as is the case in our new Clutter data set.

The Bingham distribution was first used for 3-D cluttered object detection in Glover et. al [31] (and was the subject of chapter 5). However, that system was incomplete in that it lacked any alignment step, and differs greatly from this work because it did not use feature correspondences.

6.10 Conclusion and Future Work

We have presented a system for 3-D cluttered object detection which uses a new alignment method called Bingham Procrustean Alignment (BPA) to improve detections in highly cluttered scenes, along with a new RGB-D data set which contains much more clutter and pose variability than existing data sets. Our system relies heavily on geometry, and will clearly benefit from image and color models, such as in Tang et. al [81]. Our Clutter data set, while challenging, contains zero ambiguity, in that a human could easily detect all of the objects in their correct poses, given enough time to study the models. An important direction of future work is to handle ambiguous scenes, where the parts of objects that are visible are insufficient to perform unique alignments, and instead one ought to return distributions over possible model poses. In early experiments we have performed on this problem, the Bingham distribution has been a useful tool for representing orientation ambiguity.
Chapter 7

Dynamic Manipulation: Teaching a Robot to Play Ping Pong

We now turn our attention away from computer vision and towards the final topic in this thesis: dynamic manipulation. Specifically, in this chapter we present work on a ping-pong-playing robot that learns to improve its swings with human advice. Our method learns a reward function over the joint (combined) space of task and policy parameters $\mathcal{T} \times \mathcal{P}$, so the robot can explore policy space more intelligently in a way that trades off exploration vs. exploitation to maximize the total cumulative reward over time. Multimodal stochastic polices (e.g., for when two very different swings could achieve the same high expected reward) can also easily be learned with this approach when the reward function is multimodal in the policy parameters. We extend the recently-developed Gaussian Process Bandit Optimization framework to include exploration-bias advice from human domain experts, using a novel algorithm called Exploration Bias with Directional Advice (EBDA).

As any athlete (or parent) knows, coaching is an essential ingredient to human motor-skill learning for manipulation. Learning to tie one’s shoes, cook a meal, or shoot a basketball is nearly impossible without the active guidance of a teacher. With the right combination of demonstration, verbal advice, goal-setting, and performance feedback, humans routinely teach one another incredibly complex skills in a wide variety of domains. Yet most work on manipulation learning in robotics has focused
on self-improvement. The human teacher may be involved in an initial demonstration via teleoperation or kinesthetic teaching, but after that the robot is on its own; left to explore independently, often using no more than a random walk in policy space, with rewards few and far between.

The work that has been done on coaching robots (beyond the initial demonstration phase) has primarily used goal-setting and feedback (i.e., rewards). But there has been very little work on directly giving robots policy exploration advice. Sometimes a simple “pull harder” or “keep your knees bent” is enough to get human learners to master a skill. Doing the same for robots in arbitrary domains will require AI systems that can translate generic natural language commands into motor skill policy information. While there has recently been some effort on this problem [82], we are a long way from any complete solution.

In this chapter we assume that the mapping between verbal commands and policy parameters can be programmed in ahead of time. Our focus is on integrating an advice signal from a human coach into a robot’s exploration policy. We focus on episodic learning tasks, like hitting a ping pong ball, where the robot executes a policy \( \pi_\theta \) with parameters \( \theta \) for some finite period of time, then receives a reward \( y \) reflecting how well it performed the task. In the case of ping pong, the policy \( \pi_\theta \) is a computer program (parameterized by \( \theta \)) for swinging a robot arm, and the reward depends on whether or not the robot successfully hit the ball back to the other side of the table. The robot’s goal is to maximize the total reward it receives over time. We phrase this as a multi-armed bandit problem, and extend the recently-developed Gaussian Process Bandit Optimization framework [79, 51, 20] to include advice in the regret-minimizing exploration/exploitation strategy.

### 7.1 Policy Search for Dynamic Skill Improvement

There is a long-running debate in robotics on whether model-based or model-free learning of motor control policies is the best way to get robots to perform dynamic tasks like walking, juggling, or playing ping pong. Proponents of model-based ap-
proaches boast that with carefully tuned models of system dynamics, robots can achieve skills equalling or even surpassing those of their human teachers, while the model-free crowd argues that achieving such models isn’t always possible, and that robots’ ability to learn new tasks from demonstration, with little or no prior domain-specific programming, is the true measure of success.

Regardless of whether a robot uses model-based or model-free methods to achieve basic competency at a new skill, skill improvement for model-based methods is limited by the accuracy of the model class. In contrast, model-free methods, which directly explore the space of motor control policies, are only limited by the class of policies. Of course, this space is often too large to search exhaustively, and aggressive search heuristics and priors are needed to increase the data efficiency of model-free learning algorithms in all but the simplest domains.

Policies must also typically be generalized over a task space $\mathcal{T}$, which we here parameterize by a task parameter vector $\tau$. For example, in our application $\tau$ includes the predicted position, velocity and spin of the incoming ball when it bounces on the robot’s side of the table. (Although it may seem that the task space in our application is equivalent to the state space of the ping-pong ball, it actually contains higher-level features which are computed/predicted as a function of a history of the state space variables.) Complicating matters is that $\tau$ is usually a hidden variable: the robot is given a noisy task observation, $\hat{\tau}$, from which it must infer the underlying task parameters.

7.1.1 Representing the policy: explicit vs. implicit

There are two main ways that a policy-search algorithm can represent policies—explicitly, with a direct mapping from task parameters to policy parameters $(\mathcal{T} \rightarrow \mathcal{P})$; or implicitly, with a reward function over the joint space of task and policy parameters $(\mathcal{T} \times \mathcal{P} \rightarrow \mathbb{R})^1$. Although both representations have been tried in classic reinforcement learning domains [23], only direct policy mappings have been used in recent work.

---

1The learning domains in this work are single step (not sequential).
where robots have learned dynamic manipulation skills, like throwing darts, flipping pancakes, or hitting balls [48, 49, 21]. In this (direct) approach, a continuous [48] or piecewise-continuous [21] function Φ is learned from T to P, representing the robot’s current estimate of the reward-maximizing policy parameters for each task parameter vector τ. To explore, the robot perturbs Φ with random noise; for example, by sampling from a Gaussian process [48].

The main problem with this direct-mapping-based exploration policy is that it may be difficult for the learning algorithm to escape from local maxima, since the exploration only makes local perturbations to the policy parameters. In contrast, by learning a reward function over the joint space of task and policy parameters T × P, the robot can explore policy space more intelligently in a way that trades off exploration vs. exploitation to maximize the total cumulative reward over time. Multimodal stochastic polices can also easily be learned with this approach when the reward function is multimodal in the policy parameters. In figure 7-1, we illustrate the difference between implicit and explicit policy representations. For smooth reward functions that are unimodal in P (figure 7-1, left), both explicit and implicit policy models are suitable. But for “bumpy” reward functions with many local maxima, explicit methods will lose information about the multiple good actions that the robot can take given the same task parameters (figure 7-1, right). Note that for many dynamic robotic tasks like ping pong, both T and P are continuous.

7.2 Gaussian Process Bandit Optimization

The multi-armed bandit problem [72] is the decision problem of choosing the sequence of “levers” to pull when each lever ℓ has an associated “payoff” or reward distribution, Dℓ. The agent pulls a lever, receives a reward, and then gets to choose again, incorporating all the information it has obtained from past lever pulls.

Maximizing the reward that a robot learner gets by exploring a reward function f : T × P → ℝ mapping continuous task and policy parameters to the reals is akin to a multi-armed bandit problem with infinitely many levers. At each round t, the
Figure 7-1: Implicit vs. explicit policies. In the smooth reward function on the left, a good explicit policy mapping $\mathcal{T}$ to $\mathcal{P}$ can be found. In the reward function on the right, any explicit policy mapping will be throwing away information about alternative modes.

The world chooses task parameters $\tau_t$ which the robot observes with noise as $\hat{\tau}_t$, then the robot selects policy parameters $\theta_t$, executes policy $\pi_{\theta_t}$, and receives a noisy reward $y_t = f(\tau_t, \theta_t) + \epsilon_t$.

With some mild smoothness assumptions, the reward function $f$ can be modelled as a Gaussian process (GP). There is a large body of literature on exploration heuristics for GPs [65, 13]. We use GP-UCB [79, 51], which has recently been shown to achieve sub-linear regret bounds for cumulative reward in bounded policy domains $\mathcal{P}$. It has not yet been applied to active learning of robot control parameters.

**GP-UCB (Gaussian Process Upper Confidence Bound)**

The GP-UCB algorithm [79, 51] chooses the next $\theta_t$ so as to maximize a combination of the current GP mean and variance estimates of the reward function $f$:

$$\theta_t = \arg \max_{\theta} \mu_{t-1}(\hat{\tau}_t, \theta) + \beta_t^{1/2} \sigma_{t-1}(\hat{\tau}_t, \theta),$$

where $\mu(\cdot)$ and $\sigma(\cdot)$ are the posterior mean and standard deviations functions of the GP after the first $t - 1$ observations, $(\hat{\tau}_1, \theta_1, y_1), \ldots, (\hat{\tau}_{t-1}, \theta_{t-1}, y_{t-1})$. The $\beta_t$'s are exploration constants, which must increase over time as a function of log $t$ in order
to obtain the theoretical regret bounds below.

Letting $\theta_t^*$ be the reward-maximizing policy parameter vector for the true task parameters $\tau_t$, the \textit{true cumulative regret} up to time step $T$ is defined as

$$ R_T := \sum_{t=1}^{T} f(\tau_t, \theta_t^*) - f(\tau_t, \theta_t). $$

(7.2)

Unfortunately, since the $\tau_t$'s are unobserved, the best bound we can achieve for $R_T$ is $O(T)$, since there will always be a small constant expected regret at every time step $t$ due to task parameter observation noise, $\tau_t - \hat{\tau}_t$.

The sub-linear regret bounds that have been found for GP-UCB assume that the agent has access to the true task parameters. We can achieve similar bounds for the noisy task parameter case with a modified definition of regret. Letting $\theta_t^*$ be the reward-maximizing policy parameter vector for task parameters $\hat{\tau}_t$, we define the \textit{observable cumulative regret} up to time step $T$ as

$$ \hat{R}_T := \sum_{t=1}^{T} f(\hat{\tau}_t, \hat{\theta}_t^*) - f(\hat{\tau}_t, \theta_t). $$

(7.3)

$\hat{R}_T$ has a bound of $O(\sqrt{T\beta_T\gamma_T})$ with probability $1 - \delta$, where $\delta \in (0, 1)$ is an exploration parameter that can be controlled by the user. Since the robot doesn’t get to choose $\tau_t$, it is unfair to penalize it for the difference between $\tau_t$ and $\hat{\tau}_t$, and so the observable regret bounds still provide some insight into our problem.

### 7.3 Giving Advice

There are many ways to train a robot. Some of the most common are:

- **Demonstration:** The coach can demonstrate a particular skill to a robot, either by performing the skill while the robot watches, or by teleoperating the robot.

- **Goal setting:** The coach can select a sequence of tasks to give the robot, only moving on when the robot has demonstrated a sufficient level of skill at the current task.
• **Performance feedback:** The coach can give the robot learner a reward signal, which may be in addition to environmental rewards that the robot observes on its own.

• **Policy advice:** The coach can suggest modifications to the robot’s policy parameters, like “hit the ball harder” or “angle your paddle down more”.

Learning from demonstration is a well-explored paradigm in robotics [6, 69]. However, it can be difficult for a human to teleoperate a robot to perform complex and dynamic tasks, and computer vision algorithms are not yet advanced enough to understand most manipulation actions (for example, to determine that a human is using their right hand to hold a jar between their knees while the left hand tries to generate enough force to twist off the cap). Nevertheless, learning from demonstration has been applied successfully to several robotic domains, including ping pong [66].

Setting goals is less explored in the robotics literature, but it is the de facto standard for all of robotics research. Typically, a human researcher explores policy parameter space (by modifying the robot’s program) rather than the robot exploring on its own.

Performance feedback is perhaps the most-explored mechanism for training computer agents (not just robots). Particularly for reinforcement learning domains (MDPs) like computer games and robot navigation, there is a long history of incorporating a human reward signal (called reward shaping) into reinforcement learning algorithms [32, 83, 40, 70].

The approach to training robots that we use in this work is that of giving the robot policy advice. In RL domains, this has been explored with action biasing where the human biases the robot’s action-selection mechanism [47], and control sharing—where the human selects some of the robot’s actions directly [78]. For manipulation, a recent technique is to give the robot policy improvement advice where the human provides the robot with incremental improvements to its manipulation trajectories in order to indicate a step in policy space that may increase reward [38].
Figure 7-2: The robot interprets advice from a human coach as an exploration bias. The light green shaded areas on top of the reward functions (top-down view of figure 7-1(a)) in the left and right images indicate what the robots believes are “good” regions to explore, and are inferred from a series of policy gradients given by the human coach (e.g., “swing harder” or “angle your paddle up more”). Note that the exploration advice can either be global (left) or local (right). In our system, local advice is the default; advice is only applied globally (for all $\tau \in T$) at the coach’s discretion (e.g., by saying “always swing harder”). Note that we used a keyboard interface to communicate with the robot—no human language understanding was done for this work.

### 7.3.1 Policy gradient advice

Giving robots entirely new policies as advice can be a time-consuming process. Instead, we explore policy gradient advice, where the human trainer need only provide a direction in policy space $\mathcal{P}$ in which reward is likely to increase, rather than a new, complete policy. Intuitively, this type of advice can directly capture information like “hit the ball harder” or “angle your paddle down more”, so long as the mapping between the natural language instructions and policy parameters is known.

There are two ways a robot can interpret policy gradient advice: (1) as a pseudo-reward, or (2) as an exploration bias. Treating the advice as a pseudo-reward involves a level of trust: the robot believes that hitting the ball harder is a good
thing to do because the coach said so, and it may take several trials to overcome
the pseudo-reward in the case that the coach is wrong. Using the policy gradient
advice as an exploration bias (which is akin to action biasing in RL domains) is less
susceptible to bad advice, since the robot can decide for itself whether the advice was
useful. In addition, gradient-based pseudo-reward may be difficult to implement in
continuous policy domains, since it isn’t clear how far in the direction of the gradient
one should place the pseudo-rewards, or how many to place. For these reasons, we
interpret policy gradient advice as exploration bias.

7.3.2 Exploration Bias with Directional Advice (EBDA)

We now present our algorithm for turning directional (policy gradient) advice into an
exploration bias. Intuitively, EBDA turns directional information into soft constraints
on future policy parameters; namely, that they be in roughly the same direction as
the advice indicates.

Let \( T_1, \ldots, T_n \) be a sequence of tasks the robot has received, let \( \theta_1, \ldots, \theta_n \) be the
policy parameters it chose for each task, and let \( \psi_1, \ldots, \psi_n \) be the coach’s policy
gradient advice. The direction of each \( \psi_i \) indicates a direction of increasing reward
(according to the coach) in policy parameter space, with respect to the current \( \theta_i \).

The robot interprets \( \psi_1, \ldots, \psi_n \) as a sequence of probabilistic constraints on the
policy parameter vector \( \theta_{n+1} \) for the next task, \( T_{n+1} \). Each constraint is of the form
\( \{ \theta : \psi_i \cdot (\theta - \theta_i) > 0 \} \), and is only applied if the advice is global\(^2\) (indicated by \( |\psi_i| > 1 \))
or if the new task parameter vector \( \tau_{n+1} \) is close to \( \tau_i \). EBDA then flips a biased coin
(based on an advice noise parameter \( \rho \)) to determine whether or not to include each
constraint. Pseudo-code for EBDA is shown in Algorithm 1.

\(^2\)Note that the magnitude of \( \psi_i \) is only used as a binary signal for whether or not the advice is
global; it has no other purpose in the EBDA algorithm.
Algorithm 1 EBDA

**Input:** History of task parameters $\tau_1, \ldots, \tau_n$, policy parameters $\theta_1, \ldots, \theta_n$, policy gradient advice $\psi_1, \ldots, \psi_n$, advice noise $\rho$, and a new task parameter vector $\tau_{n+1}$

**Output:** Exploration region $\mathcal{E} \subset \mathcal{P}$

\[
\mathcal{E} \leftarrow \mathcal{P}
\]

\[
\text{for } i = 1 \text{ to } n \text{ do}
\]
\[
\text{Sample } s \sim \text{Uniform}(0, 1)
\]
\[
\text{if } |\psi_i| > 1 \text{ or } |\tau_{n+1} - \tau_i| < \tau_{\text{thresh}} \text{ then}
\]
\[
\text{if } s > 1 - \rho \text{ then}
\]
\[
\mathcal{E} \leftarrow \mathcal{E} \cap \{\theta : \psi_i \cdot (\theta - \theta_i) > 0\}
\]
\[
\text{end if}
\]
\[
\text{end if}
\]
\[
\text{end for}
\]

### 7.4 Robot Ping Pong

There are several reasons for choosing ping pong as an application to study human coaching for dynamic robots. First, ping pong is a high-speed, underactuated control task, requiring a level of precision and timing not seen in many robot systems outside of the factory setting. At the same time there is substantial uncertainty about the world. The spin on the ball is difficult to observe directly, and the opponent's actions are unpredictable. For these reasons it is very difficult to conquer with model-based optimal control methods alone, leaving room for learning methods to improve significantly upon the state of the art. Ping pong is also similar in spirit to other high-skill manipulation tasks, such as those often found in cooking—chopping vegetables, stirring ingredients in a bowl, flipping burgers—in that they are all dynamic and underactuated and are defined by repetitive motions applied to a limited range of objects. Finally, ping pong has been used as an application for learning dynamic control policies [66], which makes it a good application for comparison.
Background

The first work on robotic ping pong was by Russell Andersson at Bell labs in the late 1980's [4, 3]. A massive feat of engineering at the time, his system combined real-time computer vision to track the ball with high-speed controllers on a standard Puma robot arm and was able to beat novice players on a modified ping pong table (higher net, narrower table). Since computing power was limited at the time, he had to derive all the equations of motion for the system analytically, which he did to varying degrees of success. The effect of spin on the ball’s collision dynamics, for example, was greatly simplified by reducing friction on the robot’s paddle.

The next major research effort in robotic ping pong came in the early 2000’s in Japan by Miyazaki and Matsushima [61, 64, 63]. Their system used a frame attached to the ping pong table with a paddle that translates only in the xy-plane, with 2 additional rotational degrees of freedom. Drawing upon advances in robot juggling, they used locally-weighted regression (LWR) to learn input-output maps for ball collision dynamics, enabling their system to use a modern, inverted rubber ping pong paddle (capable of imparting a great deal of spin on the ball). Due to the limitations of their minimalistic hardware setup, their system was only able to rally with a human opponent for short rallies at moderate speed. But to their credit, the table was unmodified, and in their released videos it appears that the human opponents were skilled ping pong players, accustomed to hitting the ball with spin.

More recently, a Chinese group led by Yuan-Hui Zhang have used humanoid robots to play ping pong, where the robot must balance while swinging the paddle at the same time [89, 80].

But the closest work to ours is [66]. Their robot also learns to improve its swing parameters from experience, but they use an explicit policy mapping from task parameters to policy parameters, and they do not use human advice to help the robot learn. We compare to their learning method, CrKR, in our experimental results.
Figure 7-3: Our ping pong robot uses a 7-dof Barrett WAM arm with a ping pong paddle rigidly attached at the wrist.

Hardware

Our system consists of a 7-dof Barrett WAM arm, mounted sideways on the wall behind the ping pong table, along with a pair of high-speed 200Hz black-and-white Silicon Video SV-640M cameras mounted on a fixed frame above the arm (Figure 7-3). A Kinect is also used to detect human opponents and predict ball spin based on the type of swing they use to hit the ball to the robot.

Sensing

Since the cameras are stationary and the ping pong ball is much lighter in color than the table, we use a simple background-subtraction-based blob tracker to detect and track the ping pong ball. Detections in the left and right cameras are triangulated to estimate depth, and the ball’s 3-D position is then transformed into ping pong table coordinates. The camera-table transform is estimated using a gradient descent algorithm which optimizes the fitness of the table’s edges projected into left and right Canny edge images. The human opponent’s body configuration is tracked in the Kinect depth image with OpenNI.
Tracking and predicting the ball’s trajectory

In order to decide when and where the robot should swing its paddle, it needs to predict the ball’s future trajectory based on its past trajectory. Although it would be possible for the robot to learn the ball’s dynamics autonomously, this is not the focus of our current work. Therefore, for simplicity we use Andersson’s models to predict ball aerodynamics and collisions with the table [4].

The ball’s state is defined by the 9-dimensional vector \( \mathbf{b} = (x, y, z, v_x, v_y, v_z, w_x, w_y, w_z) \), representing the ball’s current position, velocity, and spin. During flight, the ball’s acceleration is given by

\[
\mathbf{a} = -C_d|\mathbf{v}|\mathbf{v} + C_m|\mathbf{v}|\mathbf{w} \times \mathbf{v} - \mathbf{g},
\]  

(7.4)

where \( C_d \) and \( C_m \) are drag and Magnus coefficients, and \( \mathbf{g} \) is the gravity acceleration vector (in the negative \( z \)-direction).

The bounce dynamics model is slightly more complex, since the ball may change from a “slipping” to a “rolling” mode during contact with the table. First define

\[
\mathbf{V}_{\text{rel}} = (v_x - w_y r, v_y + w_x r, 0)
\]

(7.5)

\[
\dot{\mathbf{V}}_r = \frac{\mathbf{V}_{\text{rel}}}{\|\mathbf{V}_{\text{rel}}\|}
\]

(7.6)

where \( r = 20mm \) is the ball’s radius. Then rolling occurs if

\[
\|\mathbf{V}_{\text{rel}}\| < \frac{5}{2} C_f v_z (1 + C_r)
\]

(7.7)

where \( C_f \) is the coefficient of friction and \( C_r \) is the coefficient of restitution. If rolling occurs during the bounce, the ball’s final velocity \( \mathbf{v}_f \) and spin \( \mathbf{w}_f \) are

\[
\mathbf{v}_f = \left( \frac{3v_x + 2rw_y}{5}, \frac{3v_y - 2rw_x}{5}, -C_r v_z \right)
\]

(7.8)

\[
\mathbf{w}_f = \left( \frac{2rw_x - 3v_y}{5r}, \frac{2rw_y + 3v_x}{5r}, w_z \right).
\]

(7.9)

Otherwise, the ball experiences friction during the whole bounce, and the final velocity and spin are

\[
\mathbf{v}_f = \mathbf{v} + (C_f \mathbf{\dot{V}}_r - \mathbf{\hat{k}}) v_z (1 + C_r)
\]

(7.10)

\[
\mathbf{w}_f = \mathbf{w} + \frac{3C_f}{2r} (\mathbf{\dot{v}}_{rx}, -\mathbf{\dot{v}}_{ry}, 0) v_z (1 + C_r)
\]

(7.11)
where \( \hat{k} = (0, 0, 1) \) is the table normal.

To track the ball’s state, we use an Extended Kalman Filter (EKF) derived from these equations of motion. We initialize the ball’s spin vector based on the human opponent’s paddle hand velocity vector (determined by smoothing a finite difference of hand position signals from OpenNI) at the time of contact with the ball. Letting \( (h_x, h_y, h_z) \) be the hand’s velocity vector, we empirically set the ball’s spin to be

\[
 w = (240, 40h_x, 0) \quad \text{if} \quad h_z > 0
\]

for a topspin stroke, and

\[
 w = (-240, -80h_x, 0) \quad \text{otherwise.}
\]

This is clearly a gross approximation to the true effect of the human hand’s velocity vector on the initial ball spin, but with high initial EKF variance, they provide a suitable prior on the ball’s spin state for our experiments. Without this prior, it takes the EKF too long to determine the ball’s spin (which has a great effect on its predicted trajectory), and the robot cannot differentiate between a topspin and underspin ball in time to decide how to hit it.

Note that we are still working on incorporating the quaternion Bingham filter (QBF) approach from chapter 4 into our ball tracking system. Unfortunately, the logo detections are currently too noisy due to poor image resolution in our current system’s high-speed cameras, so tracking the orientation and spin of the ball based on observations of the logo will require a new camera setup.

**Swing planning**

Our approach to swing planning is hierarchical. First, the robot finds a *hit plan* indicating a desired contact between the ball and paddle. The hit plan includes the paddle’s position, velocity, normal, and time when it hits the ball. Then, a kinodynamic paddle trajectory is planned that achieves the desired contact state. Finally, the robot uses inverse kinematics to find a trajectory in joint angle space to satisfy the paddle trajectory constraints.

For our learning experiments, the hit plan is precisely the policy parameter vector, \( \theta \). Task parameters \( \tau \) are the position, velocity and spin of the incoming ball when it bounces on the robot’s side of the table.

The paddle always starts in the same configuration with position \( p_0 = (x_0, y_0, z_0) \)
and normal \( n_0 = (nx_0, ny_0, nz_0) \) at the beginning of every swing. To find a paddle trajectory that achieves the desired hit (paddle position \( p_{hit} \), normal \( n_{hit} \), and velocity \( v_{hit} \) at time \( t_{hit} \)), we compute three separate bang-bang control laws for the paddle’s \( x \)-, \( y \)-, and \( z \)-trajectories, and we interpolate the paddle normals to transition smoothly from \( n_0 \) to \( n_{hit} \).

Let \( T \) be the duration of the swing (starting at \( t_0 = t_{hit} - T \)), discretized into \( N \) time steps (so each time step is \( \Delta t = T/N \)). The bang-bang control law we use is to accelerate at \(-a\) for \( k \) time steps, then at \(+a\) for \( N - k \) time steps. Assuming the system starts at rest at the origin, the velocity trajectory given by this control law is \( \Delta t \cdot a \cdot (1, \ldots, -k, -k + 1, \ldots, N - 2k) \), and so the final position is

\[
x_N = (\Delta t)^2 a \left( \frac{(N - 2k)(N - 2k + 1)}{2} - k^2 \right). \tag{7.12}
\]

Thus the acceleration is uniquely determined by \( x_N \) and \( k \); we set \( a \) to make \( x_N = x_{hit} \), so the paddle always reaches the desired final position. Changing \( k \) yields different final velocities, \( v_N = \Delta t \cdot a \cdot (N - 2k) \); we choose the \( k \) that makes \( v_N \) as close as possible to the desired hit velocity. Note that we compute three different pairs: \((a_x, k_x), (a_y, k_y), (a_z, k_z)\) for controllers in the paddle’s \( x, y, \) and \( z \) coordinates.

**Learning a swing controller**

Given a planned trajectory \( Q = (q_0, \ldots, q_N) \) from times \( t_0 \) to \( t_N \) in joint angle space, the robot must find a way to execute the swing by sending torque commands to the motors at each joint. This is accomplished with a combination of feed-forward and proportional control in joint angle space. Given current joint angles and velocities \( q \) and \( \dot{q} \), and desired joint angles and velocities \( q_t \) and \( \dot{q}_t \), the robot applies torques:

\[
u = K_p(q_t - q) + K_d(\dot{q}_t - \dot{q}) + u_t, \tag{7.13}
\]

where \( K_p \) and \( K_d \) are gain matrices, and \( u_t \) are the feed-forward (constant) torques at time \( t \).

Controllers of this form have been applied to execute dynamic motions with robot arms in the past. Often, \( u_t \) is computed with an inverse dynamics model that estimates the torques needed to achieve varying joint accelerations \( \ddot{q} \) in state \((q, \dot{q})\). Such
models assume $\dot{q}$ is a function of $q$, $\dot{q}$, and $u$; higher-order effects like vibrations, cable stretching, gear thrashing, etc. are ignored.

Our ping pong robot needs to operate at the very limits of its specifications, with paddle speeds up to 3 meters/second and swing durations of less than half a second. Because of this, we found inverse dynamics models to be too noisy to rely on for swing control. Instead, our system learns a library of feed-forward torque trajectories, $U_1, \ldots, U_m$, corresponding to hit plans $H_1, \ldots, H_m$ (which uniquely determine joint trajectories $Q_1, \ldots, Q_m$). This library is learned incrementally—a new $(U, H)$ pair is only added when the error between a desired swing and actual swing (as measured by joint encoders) is larger than a given threshold.

To learn a new swing $(U, H)$ with corresponding $Q$ and $\dot{Q}$, the robot initializes $U$ to either: (i) the $U_i$ corresponding to the closest $H_i$ if $|H - H_i| < \text{thresh}$, or (ii) $G(Q_i)$: the torques needed to cancel out gravity at each joint angle configuration in $Q_i$.

Then, the robot executes the current swing according to equation 7.13 and computes the errors between desired and measured joint angles and velocities, $Q - \dot{Q}$ and $\dot{Q} - \ddot{Q}$. These errors are used to update $U$:

$$dU \leftarrow \lambda K_p(Q - \dot{Q}) + K_d(\dot{Q} - \ddot{Q})$$

$$U \leftarrow U + \min(\max(-dU_{\text{max}}, dU), dU_{\text{max}}),$$

and then the robot repeats the swing and updates $U$ until convergence. This $U$ update is a functional control law for trajectories that tries to drive $Q - \dot{Q}$ and $\dot{Q} - \ddot{Q}$ to zero over time as the robot practices the swing over and over again. Note that each time the robot practices the swing, it decreases the position gains $K_p$ in equation 7.13 so it relies more and more heavily on the feed-forward torques $U$ to drive the swing.

7.5 Experimental Results

We compare five learning methods: GP-UCB, CrKR, GP-UCB + EBDA, CrKR + EBDA, and Direct—where the human coach directly controls the robot’s hit plan.

---

3The anti-gravity torques are also learned: the robot moves to a list of pre-specified configurations with a PID control law and records the torques needed to keep the system at rest.
with incremental changes to policy parameters. CrKR is a direct policy mapping method [48] that has been applied to learning ping pong swings [66]. For the three learning algorithms that use advice, we only allowed the coach to give information about one parameter at a time.

**Simulation**

As a pilot domain, we simulated mixture-of-Gaussian reward functions and ran each learning method (except CrKR + EBDA) for 100 trials. Direct advice was given by adding random noise to the true maximum policy parameter $\theta^*$ in the slice of $\mathcal{P}$ with all other policy parameters fixed to their last values. Directional advice $\psi$ for EBDA was given by the direction of $\theta^*$ with respect to the last $\theta$ along one randomly chosen axis. We repeated each experiment 50 times. Average cumulative rewards for each method are shown in figure 7-4. For GP-UCB, we tried using both the $\beta$ from the original paper [79] and $\beta = \log t$ (beta1 and beta2 in figure 7-4). Since $\beta = \log t$ performed the best in simulation, we used that beta throughout the rest of the experiments. Adding advice (with EBDA) improved GP-UCB further, while direct advice performed the best (because it is an oracle perturbed by noise).

**Robot Ping Pong**

We next compared the learning algorithms on the task of improving the ping pong robot’s hit plans. We ran each algorithm for 100 trials of a human ping pong expert hitting balls to the robot in a predetermined sequence (e.g., “high backhand topspin to (0.2, 0.4) on the table”, “low forehand underspin to (0.7, 0.5) on the table”). If an incoming ball trajectory from the human differed significantly from the desired ball trajectory, the human was prompted to hit another ball to the robot.

As a starting point and a baseline, we implemented a default hit plan as follows. Given an estimate of the ball’s predicted future trajectory, it finds the ball’s highest point in that trajectory within $\pm 15cm$ of the robot’s end of the table. That ball position is the hit plan’s paddle position. The paddle velocity is a constant $(1.3, 0, 0) m/s$, and the paddle normal points in the opposite direction of the ball’s $(x, y)$-velocity.
Figure 7-4: Simulation results (cumulative reward vs. learning trials) with mixture-of-Gaussian reward functions, for task and parameter spaces of different dimensions.
vector (in order to hit the ball back the way it came).

We ran the default hit plan for 50 trials of topspin balls only to generate an initial swing table of 12 swings. Then—resetting to the initial swing table and hit policy each time—we ran 5 different learning methods on 100 trials of topspin balls and 100 trials of underspin balls, for a total of 10 learning experiments. The policy parameter vector for each learning experiment was \( \theta = (x, v_x, v_z, n_y, n_z) \): a subset of the hit plan parameters from section 7.4. \( x \) was defined as a displacement from the default hit plan’s \( x \) position, and controlled whether the ball was hit before or after its highest point. Policy parameters were bounded from \((-0.1, 1, 0, -0.1, -0.5)\) to \((0.1, 2, 0.5, 0.1, 0.5)\).

Due to the real-time nature of the ping pong task, it was infeasible to run the learning methods to choose a hit plan \( \theta \) for every new task parameter vector (i.e., ball parameters) \( \tau \). Instead, a lookup table of hit plans was computed between trials for every \( \tau_1, \ldots, \tau_n \) the robot has seen in that experiment. Then, when a new incoming ball is detected by the robot (with ball parameters \( \tau_{n+1} \)), the robot uses the pre-computed hit plan from the closest \( \tau_i \), or the default hit plan if no nearby \( \tau_i \) is found.

Rewards were given by the experimenter of 0 or 1 depending on whether the ball was successfully hit back to the other side of the table. In addition, a coach’s pseudo-reward of 0.5 was added if the coach liked the way the robot hit the ball, for a total reward in each trial between 0 and 1.5.

The cumulative reward (without the coach’s pseudo-rewards) is shown in figure 7-5 for each learning method. Using EBDA to include human advice increased reward for both GP-UCB and CrKR for both topspin and underspin. GP-UCB and CrKR are comparable on topspin, while GP-UCB is the clear winner for the harder underspin experiment, which requires a great deal more exploration since the default policy is designed to return topspin balls only.

The results support the idea that global exploration of a reward function (with GP-UCB) is better than local exploration of policy space (with CrKR) when the default policy is far from the optimum. However, the best method may be to combine the two: starting globally and then exploring locally for improvement once a good policy has been found.
Figure 7-5: Robot ping pong results (cumulative reward vs. learning trials) for topspin and underspin balls.

Videos of the robot playing ping pong (before and after learning) are available at https://sites.google.com/site/aaairobotpingpong/.

7.6 Conclusion

We have presented EBDA, a general method for incorporating policy gradient advice into a robot's exploration strategy. By incorporating EBDA into two recent exploration methods, CrKR and GP-UCB, we were able to boost the robot's cumulative reward both in simulation and on a ping pong robot. On the robot, we demonstrated an approach to learning both low-level swing controllers (feed-forward torques) and high-level hit parameters. Our robot was able to learn policies to return both topspin and underspin balls—to our knowledge, this is the first time this has been accomplished, as the underspin “push” is a difficult skill to master, even for human ping pong players.
Chapter 8

Conclusion

Over the course of the last seven chapters, we have covered in full the technical contributions of this thesis. The purpose of this final chapter is threefold: to place these contributions in a broader context, to evaluate their strengths and weaknesses, and to illuminate directions for future work.

If we hope to ever get robots cooking and cleaning for us, we must first teach them to perceive and manipulate the extraordinarily wide variety of object configurations that are possible in a home environment—from stacks of dishes and piles of dirty laundry and trash to plastic bags, papers, wires, and furniture. They must be able to push buttons, open lids, bend and fold things, pour and stir liquids, and (once we trust them enough) cut vegetables and cook things on the stove.

Clearly, the contributions towards object detection and motor control learning in this thesis are but small steps in the pursuit of these loftier goals. Our object detection system is currently limited to detecting known instances of rigid objects. And yet, since the core of the detection algorithm is a new method for alignment (BPA), and at the core of the alignment method lies a new way of understanding 3-D orientation information (with the quaternion Bingham distribution), it is hoped that many of the tools developed in this thesis can be easily extended to more general settings. For example, incorporating a “shape fitting” step into BPA would allow it to handle deformable objects or novel objects within a class (e.g., bowls) with similar shapes. In addition, BPA could form the basis of a new object tracking algorithm,
since tracking can be accomplished by iteratively aligning objects with the scene from one frame to the next.

But to truly tackle the complexity of a sink full of dirty dishes or a desk full of papers and wires, robots will need computer vision systems that go beyond detecting “what and where” objects are in the scene. The next logical step is to begin estimating relationships between objects, like “on-top-of,” “contains,” and “attached.” Estimating these relationships in conjunction with detecting the objects will also improve detection rates, since considering physical constraints between objects can lower the size of the object placement hypothesis space considerably. They will also need to understand object attributes, like “slippery” or “sticky.” And most importantly, they must make predictions about how the objects will move—predictions that depend on those attributes and relationships, and the actions the robot takes to manipulate them.

8.1 Food for Thought

Let’s consider the task of having a robot make a sandwich. A reasonable first step is to find a plate. (We’ll ignore high-level planning for the purposes of this discussion, and instead focus on accomplishing each of the sub-tasks, which may either be specified by a human or by a planning algorithm.) Suppose the robot already knows which cupboard the plates are stored in. It opens the cupboard. Now it sees stacks of plates, bowls, and cups. Maybe the bowls are even stacked on top of the plates. So the first challenge is simply to detect the plates, and identify their relationship to objects around them, so that the robot can make a plan to first lift the bowls, then grab the first plate off of the stack, and then place the bowls down again on the remaining plates.

Now the robot must execute the plan without breaking anything. It reaches for the bowls. They are too big to grasp with one hand, so it must use two hands. It can’t see what’s behind the bowls, so it has to feel its way around the back to make sure it has a good grasp. It lifts up the bowls. Now, since both of its hands are
occupied, it has to find a place to put the bowls down before it can retrieve a plate. The robot identifies a clear space on the counter and sets the bowls down. Now it looks inside the cupboard again and finds that there is a small plate on top of the big plates that it didn’t detect before when the bowls were in the way. It uses one hand to lift up the small plate, and then uses the other hand to grab the first big plate underneath. The robot has to use all of its sensors vision, touch and force—to ensure that it has grasped one, and only one of the big plates. It begins to pull the big plate out of the cupboard. Then suddenly, it feels a spike in its force sensors and notices one of the cups tipping over next to the plates. The plate must have bumped into the cup. The robot quickly drops the plate back onto the stack and lunges its free hand over to stabilize the cup before it falls. It pushes the cup out of the way to make room, grabs the plate again, pulls it out of the cupboard, sets down the small plate on top of the remaining big plates, finds a free space for the big plate on the counter, sets it down, and then grabs the bowls with two hands, puts them back in the cupboard, and closes the cupboard door! As you can see, even the first step in making a sandwich (or any meal that requires a plate) is quite complex, involving 18 distinct manipulation actions and 13 perceptual actions, not including the object tracking that goes along with each manipulation action. Even if each one of these manipulation actions has a success rate of 99% (meaning that 1% of the time there is a catastrophic failure, such as the stack of bowls getting dropped on the floor), the robot will still fail at the overall task of retrieving a plate 17% of the time! So robustness is really the key to achieving complex goals.

Looking forward, the rest of the sandwich-making activity will prove to be even harder. The robot must find a loaf of bread (which is tricky because its often inside of a bag), grab the bag and put it on the counter, open the bag, take two pieces of bread out of the bag (assuming its pre-sliced), close the bag and return it to its shelf or drawer. Then the robot needs to open the refrigerator, locate meat, cheese, lettuce, tomatoes, condiments in jars, and any other ingredients, pull them out of the refrigerator, open and close them, grab slices of meat and cheese, use a knife or a spoon to scoop and spread condiments, and so on and so forth.
How can robots achieve robustness at such complex manipulations? I think the four keys to achieving manipulation robustness are:

1. To accurately track the state of the manipulation action.
2. To quickly react to disturbances.
3. To be able to predict what might go wrong.
4. To have good “technique.”

Building better perception algorithms (that incorporate both touch and vision) is the solution to (1). Items (2) and (3) will require models of dynamics for both the robot and the objects. However, they don’t necessarily have to be perfect models of the physics— even a prediction as simple as the rough direction of motion of a tipping or falling object is often enough to move the hand into a position to catch it. The last item on the list is where the advice-based learning methods in this thesis may have the biggest effect.

Consider a task like chopping an onion. In a human culinary class, the teacher will instruct students how to position their elbows and fingers, where and how deep (and in what order) to make the cuts, and the proper angle of the knife and tension in the wrist. To get a robot to learn the same skill, it will be nearly impossible to program in a perfect model of the physics of onion chopping. For one thing, every onion is different! Lacking an accurate model, the space of policy parameters (the angle of the knife, etc.) will be prohibitively large to search through with trial and error alone. So feedback (advice) from human instructors is certain to speed up the rate at which robots can learn to master such skills.

8.2 Beyond Ping Pong

Among the hardest manipulation tasks are those involving dynamics—where the robot does not have complete control authority over the object and must predict its dynamic motion as it tips or rolls or flies through the air. One finds dynamic
manipulation tasks everywhere—chopping vegetables, stirring ingredients in a bowl, flipping hamburgers, moving furniture, playing sports, and so on. Furthermore, many non-dynamic manipulation tasks become dynamic when the robot is pushed to its limits: for example, when we want the robot to move very quickly, or to manipulate very heavy objects. It is critical to these tasks for the robot to have accurate models of object dynamics, so that the robot can predict where objects will end up when an action is taken.

The manipulation problem considered in this thesis—ping pong—is both easier and harder than other manipulation tasks. It is difficult because the system is very fast and dynamic. But because the interaction between the robot’s paddle and the ball is so brief, it is easier to predict how the robot’s actions will affect the ball. There are also relatively few policy parameters to optimize, because the only thing that matters is the state of the paddle when it hits the ball.

For longer-lasting manipulation tasks like chopping an onion or picking up a stack of plates, the robot may want to vary its policy parameters throughout the manipulation in response to sensory feedback. Of course, we can add parameters to the feedback mechanism itself, and learn those in the same way (with exploration and advice) that we learned the instantaneous policy parameters for the ping pong task. But it is unlikely that one can adequately parameterize an onion chopping policy with only five parameters, as we were able to do for ping pong.

The Gaussian-process-based exploration method that we used to explore policy parameter space in chapter 7 can theoretically be extended to higher dimensional parameter spaces. However, with a purely non-parametric model of the reward function, the algorithm has a tendency to explore all of the corners in policy parameter space. And since the number of corners grows exponentially with the dimension of parameter space, this behavior will be quite problematic for tasks with more policy parameters. Using the advice algorithm from chapter 7 (EBDA) alleviates this problem to some extent, but the amount of advice required will also grow with the number of policy parameters. Therefore, it may help to add a graphical structure to the reward function to reduce dependencies between parameters.
Bibliography


