Improving the Accuracy and Runtime of Probablistic and Deep Learning-Based Pose Estimation

Mengyun Xu

December 2019

CMU-CS-19-131

School of Computer Science

Carnegie Mellon University

Pittsburgh, PA 15213

Thesis Committee:

David Held, Chair

Sebastian Scherer

Submitted in partial fulfillment of the requirements

for the degree of 5th-year Masters of Computer Science

Keywords: Pose Estimation, Uncertainty Estimation, Deep Learning Acceleration on CPU, Probabilistic Point Cloud Registration, Linear Measurement Model

Abstract

Many applications in robotics require estimating pose, consisting of translation and orientation, between a model frame and a sensor frame. One popular application is robot grasping, where the object pose is used for determining the optimal placement of the robot gripper. This thesis consists of a survey on three methods aimed at improving pose estimation accuracy and runtime. The first method models uncertainty using various distributions and examines their ability to improve the accuracy of the resulting output, when compared to current methods that do not consider uncertainty in their models. The second method searches for the relationship between common system variables and the performance of neural networks with different configurations, in order to accelerate computation by developing a guideline to select the optimal parallel model given certain problem characteristics and hardware resources. The third method improves both accuracy and runtime by mapping the original non-linear and non-convex pose estimation problem into an alternative parameter space where the original problem can become truly linear. The lack of linearization or other approximations avoids high sensitivity to initial estimation error and high computation time.

Acknowledgments

I would like to thank Prof. David Held for being my advisor for my 5th-Year Master's program. He is knowledgeable in providing guidance, generous in offering time and lab resources to support research, and caring for students' well-beings.

I would like to thank Prof. Sebastian Scherer for serving on my thesis committee and providing valuable feedbacks.

I would like to thank Brian Okorn, an outstanding PhD student of Prof. David Held, for being my mentor on chapter 3 in my thesis. He is a great inspiration to me, and he is always supportive and encouraging in this research mentorship. The content of chapter 3 is based on Brian Okorn's undertaken research. My contributions are in algorithm implementation/optimization, experiment design and result analysis of the confusion matrix, isotropic Bingham and tracking sections. The content in this chapter corresponds to the unpublished paper where Brian Okorn is the main author and I'm the secondary author [9].

I would like to thank Prof. Howie Choset for welcoming me as an undergraduate researcher in his Biorobotics Lab and providing me with the resource for my first research experience.

I would like to thank Arun Srivatsan Rangaprasad, a remarkable PhD student of Prof. Howie Choset, for being my mentor throughout my undergraduate research and chapter 5 in my thesis. He was very patient and helpful, and would always answer my questions thoroughly to help me understand the concepts better. The chapter 5 content of this thesis is based on Arun Srivatsan's undertaken research [12]. My contributions are in the registration algorithm implementation and optimization. The content corresponds to a published paper where Arun Srivatsan is the main author and I'm the secondary author [15]. The theoretical background and math part are cited from Arun's work to describe the project. The source code of the project, where my contribution goes, can be better viewed at link.

I would like to thank Prof. Seth Goldstein and his student Angela Li for guiding and collaborating with me on chapter 4 in my thesis. The ULI benchmark described in the chapter uses

Prof. Goldstein's ongoing research active message passing (user-level interrupt) as a parallel model to accelerate neural network computing. My contributions in this project are parallel model designs and implementation, experiment designs, benchmark statistics collection, and result analysis.

I would like to thank all Carnegie Mellon University faculty who helped me achieve who I am today.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Pose estimation is commonly required in robotics applications. It involves finding the rotation and translation going from one reference frame to the other. In the example of robot arm grasping, the inputs are images captured by the camera and information about the target object (such as generated object mask or bounding box). The outputs are the orientation and translation of the target object with respect to the camera frame, which can be extended to the robot frame after applying an offset. In the example of medical image registration, the inputs are two point clouds – one initial, one target, and the outputs are the orientation and translation taken to transform the initial point cloud to the target. In this thesis, the terminology ”pose” is often used to refer orientation alone, since the main focus is on estimating the orientation component accurately.

The first project included in this thesis aims to improve the accuracy of deep learning-based pose estimation. Many existing pose estimators only output single pose as the result. However, real world application often involves uncertainty factors such as lighting, shadow or object occlusion. We believe having pose estimator outputting a pose distribution that describes the uncertainty altogether with the pose parameters can improve estimation accuracy. The additional information about uncertainty can help in many ways. In robot grasping application, without knowing the underlying uncertainty, the robot can grasp an object by a pose with high uncertainty (likely to be wrong), fail the grasp and cause irreversible damage such as breaking the object. With the underlying uncertainty also estimated and incorporated in the estimation output, the robot has the option to take alternative action to lower uncertainty first and then perform the grasp by a pose with low uncertainty (likely to be correct), increasing the success rate. To obtain output as a pose distribution, we pick two existing deep learning-based pose estimators DenseFusion [20] and PoseCNN [21] that output single pose estimate, and two distribution types Bingham and histogram. For each distribution we look into multiple methods to fit the distribution given predicted pose from a base predictor, and analyze how effective the end result can describe the underlying uncertainty.

The deep learning-based pose estimators can be slow when it comes to real time applications. In order to make our pose distribution estimation feasible for real time applications as well, the second project included in this thesis aims to improve runtime by parallelizing neural network (NN) computation with multi-threading. Deep learning is a powerful tool that comes with the cost of network complexity. GPUs have been widely used for its acceleration. GPU contains large quantity of small processing units thus is naturally a good fit for parallel processing. Most of the time computing resources are limited and GPU availability always becomes a bottleneck, while CPUs stay idle at the same time. In fact, with many powerful C parallel libraries, a programmer can also design the program to be processed on CPU in parallel efficiently. The complexity resides in the design of an efficient parallel program. This project doesn't aim to replace GPU by CPU as the mainstream accelerator for NN computation. This project aims to produce a guideline on designing the parallel program in an optimal way so that CPU can be well utilized to provide significant acceleration, in the absence or alongside of GPUs. Using multiple C parallel libraries, we summarize the possible parallel models for program design, and experiment with benchmarks of different configurations. We analyze the benchmark results and extract useful information to form the guideline on designing optimal parallel program for accelerating NN computation on CPU.

Besides deep learning-based pose estimations we also look into existing probabilistic pose estimations. Existing works favor filtering-based methods due to their ability to adapt to noisy sensor measurements. Most of the prior filtering methods use unimodal Gaussian distribution for modeling the uncertainty in the pose parameters. Such a distribution is a good choice to capture the uncertainty in parameters that are defined in Euclidean space. However, the uncertainty in parameters such as unit-quaternions, when modeled using a Gaussian, does not consider the structure of the underlying space. For example, the antipodal symmetry introduced by the fact that rotating around x axis by θ is equivalent to rotating around -x axis by $-\theta$. Also, since pose estimation is inherently a non-linear problem, using linear Kalman filter directly will cause poor result and most of the prior filtering methods linearize the non-linear measurement model. The linearization and other approximations can cause sensitivity to initial pose error and generate high computation overhead. In order to improve accuracy and reduce runtime, we re-formulate the non-linear measurement model into linear space and then use linear Kalman filter, while modeling rotation parameters by Bingham distribution and translation parameters by Gaussian distribution.

Chapter 2

Background

This background section provides a brief introduction to the concepts used in the later chapters.

2.1 Quaternion Used for Rotation Representation

In this thesis, the rotation component of pose is represented by unit quaternion. Unit quaternion lies on a unit 3-sphere and varies continuously as the orientation changes. Thus, using unit quaternion to represent rotation can avoid abrupt jumps.

A quaternion $\tilde{\mathbf{q}}$ is a 4-tuple: $\tilde{\mathbf{q}} = (q_0, \mathbf{q}), \tilde{\mathbf{q}} \in \mathbb{R}^4$, where $\mathbf{q} = (q_1, q_2, q_3)$. Its conjugate is $\tilde{\mathbf{q}}^* = (q_0, \text{-}\,\mathbf{q})$. The norm of a quaternion is $\|\tilde{\mathbf{q}}\| = \sqrt{\text{scalar}(\tilde{\mathbf{q}} \bigodot \tilde{\mathbf{q}}^*)}$, where $\bigodot$ is the quaternion multiplication explained in the next section. A unit quaternion $\|\tilde{\mathbf{q}}\| = 1$ can represent rotation by $\widetilde{\boldsymbol{q}} = \left(\cos\left(\frac{\theta}{2}\right), \boldsymbol{k}\sin\left(\frac{\theta}{2}\right)\right) = (q_0, \mathbf{q})$, where $\mathbf{k}$ is a unit vector describing the rotation axis, θ describes the rotation angle within range $[-\pi, \pi]$.

$\tilde{\mathbf{q}}$ and -$\tilde{\mathbf{q}}$ represent the same rotation because $\tilde{\mathbf{q}} = \cos(\frac{\theta}{2}, \mathbf{k}\sin\frac{\theta}{2})$ represents rotating around $\mathbf{k}$ axis by θ, -$\tilde{\mathbf{q}} = \cos(\frac{-\theta}{2}, -\mathbf{k}\sin\frac{-\theta}{2})$ represents rotating around -$\mathbf{k}$ axis by $-\theta$, and these two rotations are equivalent. Point rotation can be represented as $\tilde{\mathbf{a}} = \tilde{\mathbf{q}} \bigodot \tilde{\mathbf{b}} \bigodot \tilde{\mathbf{q}}^*$, where $\tilde{\mathbf{a}} = (0, \mathbf{a})$ and $\tilde{\mathbf{b}} = (0, \mathbf{b})$ are the quaternion representations of two points **a**, **b**, and the equation states that **a** can be obtained from rotating **b** by quaternion $\tilde{\mathbf{q}}$.

2.1.1 Quaternion Multiplication

An operator $\bigodot$ is defined for multiplying two quaternions $\tilde{\mathbf{p}}$ and $\tilde{\mathbf{q}}$: [17]

$$\tilde{\boldsymbol{p}} \odot \tilde{\boldsymbol{q}} = \begin{bmatrix} p_0 & -\boldsymbol{p}^T \\ \boldsymbol{p} & \boldsymbol{p}^\times + p_0 \boldsymbol{I}_3 \end{bmatrix} \tilde{\boldsymbol{q}} = \begin{bmatrix} q_0 & -\boldsymbol{q}^T \\ \boldsymbol{q} & -\boldsymbol{q}^\times + q_0 \boldsymbol{I}_3 \end{bmatrix} \tilde{\boldsymbol{p}}$$

, where $\tilde{\mathbf{q}} = (q_0, \mathbf{q}), \tilde{\mathbf{p}} = (p_0, \mathbf{p})$, and $\mathbf{p}^\times$ is the skew-symmetric matrix formed from the vector **p**.

This formula of converting quaternion multiplication form to matrix multiplication form is used in the later chapter 5 of converting non-linear problem space into linear space, for the purposing of deriving a linear measurement model.

2.1.2 Bingham Distribution

In this thesis Bingham distribution is used to describe the uncertainty in pose parameter $\tilde{\mathbf{q}}$ (a unit quaternion representing the rotation). This is because Bingham distribution lies on a unit hypersphere, thus addresses the unit norm constraint of the unit quaternion intuitively. Moreover, Bingham distribution has the antipodal symmetry that $f(\tilde{\mathbf{q}}) = f(-\tilde{\mathbf{q}})$, in which f is the PDF. As the result, it has two antipodal modes that capture well the unit quaternion property $\tilde{\mathbf{q}} = -\tilde{\mathbf{q}}$ [3].

Let $\mathbf{S}^{d-1} = \{\boldsymbol{x} \in \mathbb{R}^d : \|\boldsymbol{x}\| = 1\} \subset \mathbb{R}^d$ be the unit hypersphere in $\mathbb{R}^d$. The probability density function $f : S^{d-1} \to \mathbb{R}$ of a Bingham distribution is given by [15]

$$f(\mathbf{x}) = \frac{1}{N} exp(\mathbf{x}^T \mathbf{M} \mathbf{Z} \mathbf{M}^T \mathbf{x})$$

Out of the two parameters $\mathbf{M}$ and $\mathbf{Z}$, $\mathbf{M} \in R^{d\times d}$ is an orthogonal matrix ($\mathbf{M}\mathbf{M}^T = \mathbf{M}^T\mathbf{M} = \mathbf{I}^{d\times d}$), $\mathbf{Z}$ = diag$(0, z_1, ..., z_{d-1}) \in R^{d\times d}$ with $0 \geq z_1 \geq ... \geq z_{d-1}$ is known as the concentration matrix. The descending order of $\mathbf{Z}$'s diagonal elements can be obtained from rearranging columns of $\mathbf{M}$ accordingly. The first element of $\mathbf{Z}$ can be forced to 0 by adding a constant $-z_0$ on every diagonal element of rearranged $\mathbf{Z}$. Under this format, the mode of the distribution is simply the first column of rearranged $\mathbf{M}$. Covariance can be calculated by $Cov(\mathbf{x}) = -0.5(\mathbf{M}(\mathbf{Z} + c\mathbf{I})\mathbf{M}^T)^{-1}$, where c is the smallest diagonal element of $\mathbf{Z}$. In fact when the condition ($\mathbf{Z}$ + c$\mathbf{I}$)) is negative definite, $c \in R$ can be arbitrarily chosen [4]. N is a normalization constant computed by [15]

$$N = \int_{S^{d-1}} exp(\mathbf{x}^T\mathbf{Z}\mathbf{x})d\mathbf{x}$$

In the problem definition within this thesis, the Bingham distribution has $d = 4$ dimensions with the unit quaternion representing rotation as the random variable. Thus $\mathbf{Z}$ will have 4 diagonal entries, and M will be 4×4 matrix.

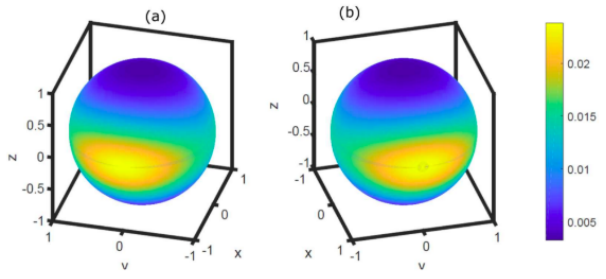


Figure 2.1: An example of a 3D Bingham distribution $f(\mathbf{s}) = \frac{1}{N}exp(\mathbf{s}^T\mathbf{M}\mathbf{Z}\mathbf{M}^T\mathbf{s})$, where $\mathbf{M} = \mathbf{I}_{3\times3}$, $\mathbf{Z}$ = diag(0, -0.5, -2), and $\mathbf{s}$=(x,y,z) [18]. The two antipodal modes exist in $\mathbf{s}$=(1,0,0) and (-1,0,0)

Product of Two Bingham Distributions

The product of two Bingham PDFs is a Bingham distribution, which can be rescaled to form a PDF [6]. This property is used in 5 for state update. Consider two Bingham distributions [15]

$$f_i(\boldsymbol{x}) = \frac{1}{N_i}\exp\left(\boldsymbol{x}^T\boldsymbol{M}_i\boldsymbol{Z}_i\boldsymbol{M}_i^T\boldsymbol{x}\right), i = 1, 2.$$

Then their product is

$$
\begin{aligned}
&f_1(\boldsymbol{x}) \cdot f_2(\boldsymbol{x}) \\
&\qquad = \frac{1}{N_1 N_2} \exp\left(\boldsymbol{x}^T \underbrace{\left(\boldsymbol{M}_1 \boldsymbol{Z}_1 \boldsymbol{M}_1^T + \boldsymbol{M}_2 \boldsymbol{Z}_2 \boldsymbol{M}_2^T \right)}_{A} \boldsymbol{x} \right) \\
&\qquad \propto \frac{1}{N} \exp\left(\boldsymbol{x}^T \boldsymbol{M} \boldsymbol{Z} \boldsymbol{M}^T \boldsymbol{x} \right)
\end{aligned}
$$

where N, **M**, **Z** are the new normalization constant and parameters after renormalization. **M** is composed of the unit eigenvectors of A. $\mathbf{Z} = \mathbf{D}\mathbf{D}_{11}\mathbf{I}_{d \times d}$ where **D** has the eigenvalues of **A** (sorted in descending order) and $\mathbf{D}_{11}$ refers to the largest eigenvalue.

Chapter 3

Modeling Pose Uncertainty

Many existing pose estimation methods output single estimate. In real world there are uncertainty factors such as shadow, lighting and object occlusions that can affect the estimation result. Thus, outputting a pose distribution that incorporates uncertainty with the pose can improve estimation accuracy. With pose distribution as the output, robot now has the option to postpone execution until uncertainty becomes lower, in order to avoid irreversible damages. Pose distributions from multiple views can be combined to generate final estimation for higher accuracy.

There are several approahces to describe pose uncertainty. We focus on using Bingham distribution and histogram distribution. We choose unit quaternion as representation for rotation because of the benefit described in 2.1. With unit quaternion as rotation representation, we choose Bingham distribution because of the benefit described in 2.1.2

3.1 Parametric: Bingham Distribution

There are several ways to obtain a Bingham distribution to describe pose uncertainty.

3.1.1 Fixed isotropic Bingham

Given an image containing the target object, we use a base pose estimator (can be any pose estimator. DenseFusion [20] and PoseCNN [21] in this project specifically) to get a predicted pose. An isotropic Bingham distribution can be fit around the predicted pose with concentration parameter σ, tuned by cross-validation for each object, that constructs the Bingham parameter $\mathbf{Z}$ that describes the spread. Object type affects pose uncertainty, thus we're hoping to see the tuned concentration parameter σ for each object generalize to the same object under other conditions, or even generalize to the objects of same kind. Symmetrical object such as bowl gives identical probability to poses around symmetry axis, causing large uncertainty in the direction representing the symmetry axis. Non-symmmetrical object such as box with distinct prints on different side gives small uncertainty in all directions.

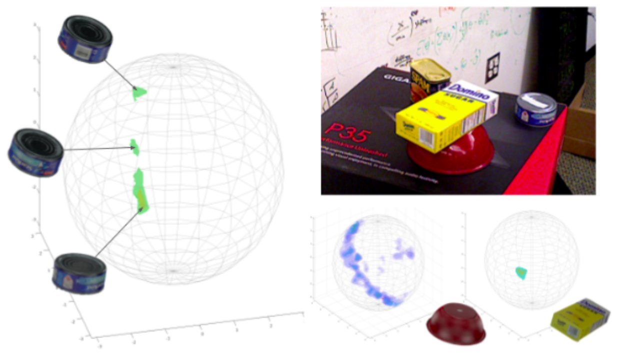


Figure 3.1: Visualization of pose uncertainty. The tuna can object has uncertainty over several poses, the bowl object has uncertainty over rotations around z axis, and the sugar box object has low uncertainty. [9]

The tuned optimal concentration parameter varies from object to object, and varies from base predictor to base predictor. Overall, from the graph below, PoseCNN base predictor (red) generates larger concentration parameter than DenseFusion base predictor (blue), and certain objects (mostly non-symmetric) generate larger concentration parameter than the others (mostly symmetric). Larger concentration parameter refers to narrow or peaky distribution. When a base predictor has high prediction accuracy, the optimal concentration parameter tends to be larger and the distribution tends to be peaky at the base pose. However, if the narrow distribution is

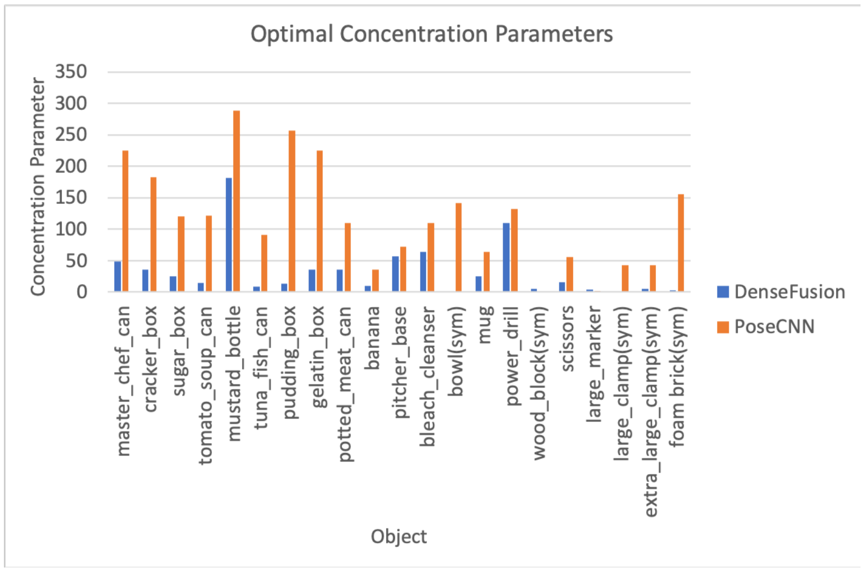


Figure 3.2: Optimal concentration parameter for each object with two different base predictors.

caused by the high bias in base predictor on specific data set, such distribution doesn't generalize to other data set.

Thus, we add a uniform distribution with certain weight to the isotropic Bingham to form a mixture, representing the likelihood of having pose $\tilde{\mathbf{p}}$: $\frac{w_c}{\pi^2} + (1 - w_c) f(\tilde{\mathbf{p}})$, where w_c is the weight of the uniform distribution, f is the pdf of Bingham distribution. Using cross-validation, we tune w_c and concentration parameter σ (part of f) to achieve maximum log likelihood.

Comparing to the optimal concentration parameter tuned with Bingham distribution alone (3.2) , the optimal concentration parameter increases overall when a uniform distribution is added (3.3). The most distinct increases are from DenseFusion base predictor. From 3.4 we observe that the optimal weight for the added uniform distribution is 1 or close to 1 for symmetric objects, especially for DenseFusion base predictor. This means that isotropic Bingham describes the uncertainty in those symmetric objects worse than a uniform distribution, as reflected by the negative log likelihood for those objects in 3.5 (-2.24 is the log likelihood of groundtruth pose with a uniform distribution). For other objects, mostly non-symmetric objects, the addition of a uniform distribution on isotropic Bingham achieves good log likelihood of groundtruth pose.

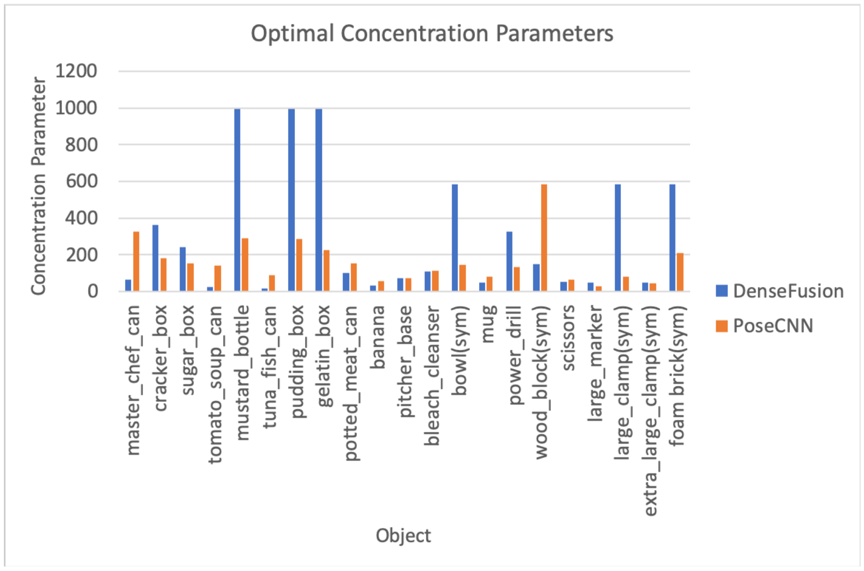


Figure 3.3: Concentration parameter for Bingham distribution in the mixture

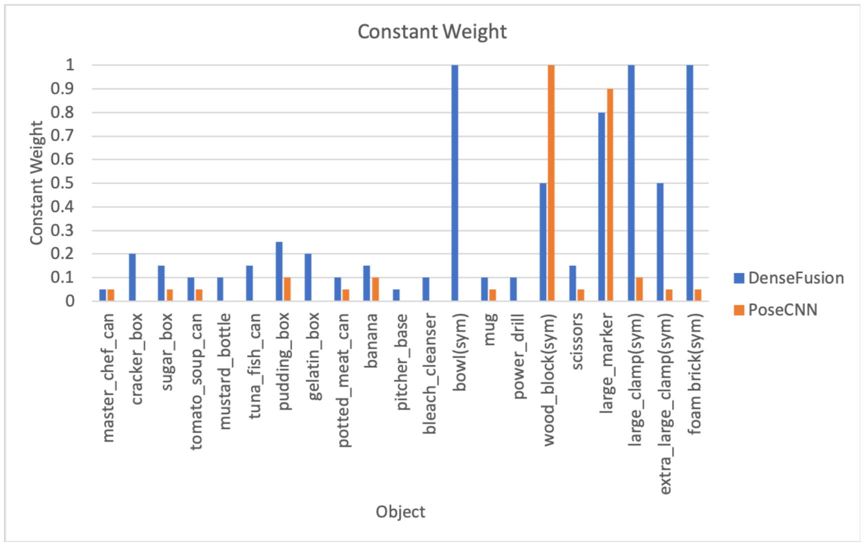


Figure 3.4: The weight for uniform distribution in the mixture

The inconsistency between the concentration parameters learned with DenseFusion and the ones learned with PoseCNN suggests that the output pose distribution varies with the base predictor choice. Certain base predictor has higher bias towards some objects, generating narrower distributions. An interesting future direction is to explore the option to combine multiple base predictors to combine the strengths of each.

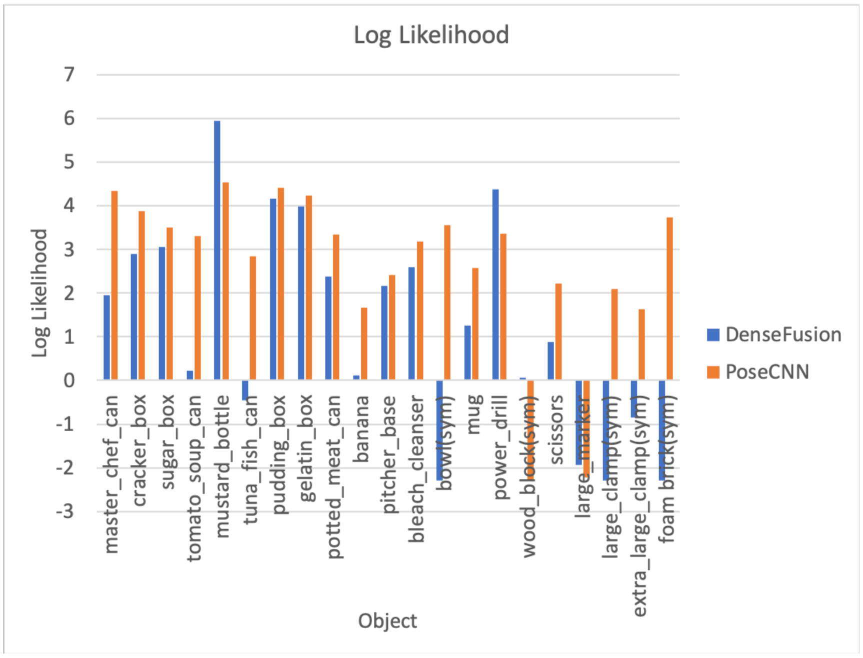


Figure 3.5: Average log likelihood of groundtruth pose using the mixture learned

3.1.2 Mixture of Isotropic Bingham

For base predictors that also generate confidences associated with each pose, we can use the confidences as weights associated with isotropic Binghams centered around the corresponding poses, forming a mixture of isotropic Binghams to describe the pose distribution.

3.1.3 MC-dropout Ensemble

For deep-learning based base predictors, we can insert a dropout layer enabled at evaluation time to produce an ensemble prediction. We can then fit a Bingham distribution over the drawn samples.

3.1.4 Direct Regression

Moreoever, we can directly regress the Bingham distribution parameters by maximizing the log likelihood of the given image.

3.2 Non-parametric: Histogram Distribution

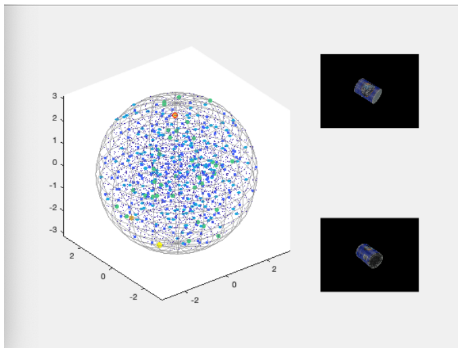
In the non-parametric approach, we use a histogram to describe pose uncertainty. Each bin of the histogram corresponds to a pose in the discretized orientation space suggested by Straub [19].

3.2.1 Confusion Matrix

We generate a confusion matrix using validation and synthetic data. Each row and column index corresponds to a pose in the discretized orientation space. The rows represent predicted poses, and the columns represent ground-truth poses. Using a base pose predictor, such as the previously mentioned PoseCNN and DenseFusion, for each image in data we get a predicted pose from base predictor and ground-truth pose from label. We map the predicted pose to the corresponding row and ground-truth pose to the corresponding column. Then we increase the confusion matrix value at that row and column indices by 1. After the confusion matrix is built from validation and synthetic data, and after normalizing each row, each cell with index (r, c) corresponds to P($\tilde{\mathbf{q}}_c \mid \tilde{\mathbf{q}}_r) = P(\tilde{\mathbf{q}}_{gt} \mid \tilde{\mathbf{q}}_{pred})$. Each row distribution describes given a predicted pose, what the actual poses could be and how likely they are.

The huge size of confusion matrix (3885 $\times$ 3885) causes difficulty in visualization, thus we're instead attaching visualization of rows here.

The left image represents a row in the confusion matrix that generates a wide-spread histogram. For the predicted pose of the upper image, the row mapped to that predicted pose in the confusion


(a) A confusion matrix row for the can object. It has random pattern

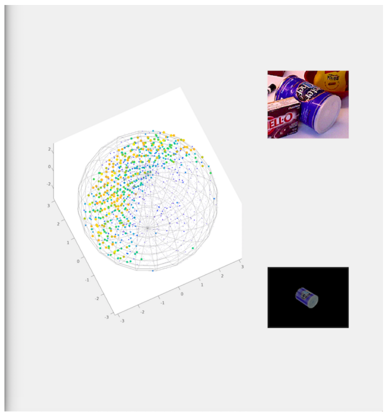

(b) A confusion matrix row for the can object. It has meaningful pattern

Figure 3.6: Visualization of two rows in the confusion matrix (upper image is associated with a predicted pose that determines the row fetched, and lower image is the rendered ground-truth pose corresponding to a selected location on the orientation sphere

matrix describes the possible ground-truth poses and corresponding likelihoods, as visualized by the dots and heat map colors on the sphere. For that specific predicted pose, there are many possible ground-truth poses seen before, each with small number of occurrences. The right image represents a row in the confusion matrix that generates a narrow-spread histogram. For that specific predicted pose, the possible ground-truth poses seen before concentrate on the band on the orientation sphere that represent all poses around one axis. Such pattern is caused by the base predictor recognizing the object as symmetric over one axis (Technically the can is not symmetric due to the prints. But since the confusion matrix is built from predicted poses from the base predictor, it is up to the base predictor's judgement).

3.2.2 Multimodal Distribution Regression

We can train a neural network that has featurization of an image $\Phi(I)$ as input, and outputs probability of each discretized pose given that image featurization $P(\tilde{\mathbf{q}_j} \mid \Phi(I))$, where $\tilde{\mathbf{q}_j}$ is the unit quaternion representing the j-th pose in the discretized orientation space. One method to get the image featurization is extracting an intermediate layer of a CNN having the image as the input.

3.2.3 Comparison based Distribution Estimation

Similar to the multimodal distribution regression, we can have the neural network input as the concatenation of featurization $\Phi(I)$ and featurization $\Phi(I_j)$, where I is the input image, and I_j is the image of the target object rendered at the j-th pose in the discretized orientation space. The neural network will output probablity of the j-th pose given the image featurization concatenation $P(\tilde{\mathbf{q}}_j \mid \Phi(I), \Phi(I_j))$

3.3 Evaluation

We use the test set in YCB-video dataset to evaluate all methods described above. For each test image and ground-truth pose label, we obtain a pose distribution via the methods described above and calculate the log likelihood of ground-truth pose using the pose distribution. We use two base pose predictors for this experiment: DenseFusion and PoseCNN.

Notations used in the table:
Histogram-Comp: Comparison based Distribution Estimation 3.2.3
Histogram-Reg: Multimodal Distribution Regression 3.2.2
Histogram-Conf: Confusion Matrix 3.2.1
Bingham-Fixed: Fixed isotropic Bingham 3.1.1
Bingham-Mix: Mixture of Isotropic Bingham 3.1.2
Bingham-Reg: Direct Regression 3.1.4
Bingham-Drop: MC-dropout 3.1.3

The confusion matrix method gives much higher mean log likelihood of ground-truth pose than

Objects	Histogram					Bingham				
	PoseCNN		DenseFusion			PoseCNN	DenseFusion			
	Comp	Reg	Comp	Reg	Conf	Fixed	Fixed	Mix	Reg	Drop
Non-Sym	0.03	-2.21	-0.24	-0.18	1.52	0.46	0.51	0.75	-1.98	0.59
Sym	-1.30	-2.25	-1.20	-3.58	2.46	-2.38	-2.35	-2.28	-2.10	-8.53
All	-0.19	-2.21	-0.39	-0.72	1.67	0.01	0.05	0.27	-2.00	-0.87

Table 3.1: Mean log likelihood of ground-truth pose

the other methods. The values seem too good to be true, especially the high performance in the usually hard-to-estimate symmetric objects. We perform further analysis on the confusion matrix generated in attempt to explain the reason behind the high values.

We first look at the confusion matrix constructed. It is so sparse that it looks like an empty matrix when visualized (because its size 3885 $\times$ 3885 is so huge and most values concentrate around diagonal). Thus we generate a bin value occurrence plot 3.7 (how many cells in the confusion matrix has value x) to show the sparseness.

This is a truncated graph for readability. The occurrence count near bin value 0 actually goes as high as 15×10^6. Thus we can see this is a very sparse confusion matrix. In average, 27.1% of the confusion matrix rows are empty (have all 0's). This means during the confusion matrix building, there are 27.1% poses in the discretized orientation space that the base predictor never predicts. Since we also use a synthetic dataset to build the confusion matrix, and the synthetic dataset has objects rendered at all poses in the discretized space, we observe that the base predictor has tendency to mis-predict certain poses all the time, probably due to the lack of these poses in the training data. In average, 16.2% of the non-empty rows have values on the diagonal cell in the row. This means in the seen predicted poses, 83.8% of them are never associated with the correct ground-truth pose, thus it is very unlikely for the row distribution to peak around the correct ground-truth pose. By multyping the two ratios: ratio of seen predicted pose $\times$ ratio of correct ground-truth hit for seen predicted poses = $27.1\% \times 16.2\% = 4.4\%$ of the discretized pose space have been seen during confusion matrix building and generated a histrogram distribution with a

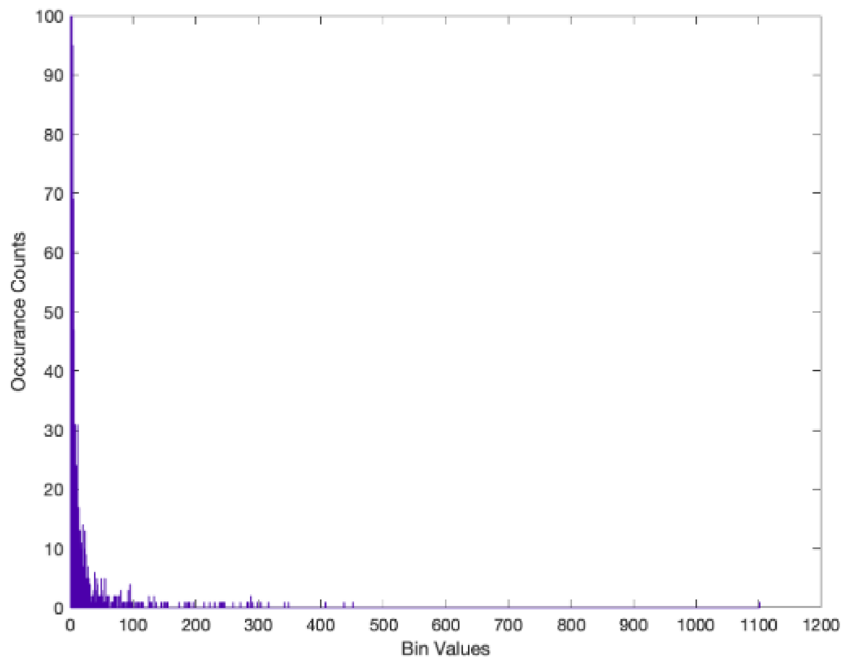


Figure 3.7: Horizontal axis: value that a confusion matrix cell contains. Vertical axis: number of occurrence of bins with certain values

peak around the correct ground-truth pose.

$4.4\% \times 3885$ discretized poses = 171 poses. We suspect the reason that confusion matrix method has high performance despite the sparseness is because YCB-dataset contains mainly these 171 poses, since the camera mainly moves around the objects sitting on top of a table. As the base predictor is trained to have biases towards these particular poses, the confusion matrix built is able to have distinct peak at the correct ground-truth pose for those frequently occurring poses, thus able to perform distinctly well comparing to other pose distribution representations. As the result, confusion matrix method will perform distinctly well on common poses due to its memorization from sufficient dataset. However, when facing an unfamiliar pose, it can only perform as good as a uniform distribution.

3.4 Conclusion

Out of all approaches we use to describe pose uncertainty, we find the confusion matrix approach perform significantly better than others. We perform extra analysis in attempt to explain the high performance. Surprisingly, the confusion matrix built and used for evaluation has many empty rows and low diagonal hit rate. These are two factors that should hurt performance instead. Empty rows mean there are predicted poses that the confusion matrix has no prior knowledge about. If one of those poses is predicted by the base estimator at evaluation time, the confusion matrix will output a uniform distribution, which definitely doesn't provide meaningful information about the true pose. Low diagonal hit rate means the confusion matrix will likely output row distributions that have peaks at the wrong locations. With the wrong peaks, the row distributions don't describe the true pose correctly. The high performance number and the existence of these two performance-harming factors seem contradictory. We see the promising potential for the confusion matrix method to learn meaningful histogram distributions from data, but we suspect that the high performance comes from the data's high bias towards certain common poses. We will need to perform further analysis on the data quality to confirm our doubt.

Chapter 4

Parallelizing Computation with Multiple Threads

The project mentioned in the previous section aims to improve accuracy in pose estimation. This project aims to improve runtime by parallelizing computation with multiple threads.

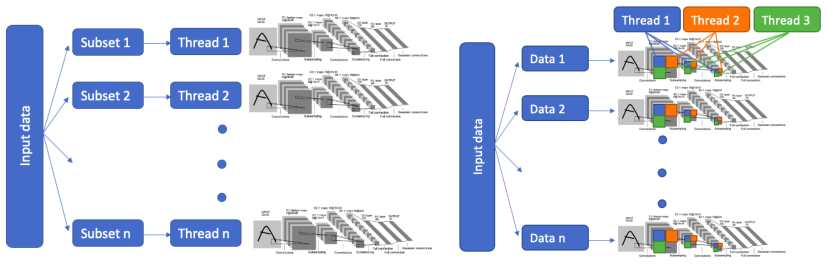
4.1 Introduction

Similar to many other computer vision problems, pose estimation can also utilize the power of deep learning. While GPUs are widely used to accelerate computation-heavy deep learning training and evaluation, we focus more on the potentials of CPUs at achieving the same tasks. The work included in this thesis does not aim to replace GPU by CPU as the mainstream accelerator. This work aims to provide a guideline on selecting the optimal parallel model to use, given the problem characteristics and hardware resources, in order to maximize the acceleration a CPU could provide. Most of the times computing resources are quite limited. Even with GPUs available to use, it is still a good idea to fully utilize the CPUs to engage in the deep learning tasks instead of having tasks pending for the GPUs while the CPUs remain idle.

4.2 Parallel Models

A parallel program consists of a task assignment model and a synchronization model. The task assignment model determines how compute resources distribute the total tasks, and the synchronization model determines how different compute resources merge individual pieces of work. In a multi-threading program, the compute resources will be threads.

4.2.1 Task Assignment Model



(a) Data Parallel (b) Model Paralle

Figure 4.1: Graphic representation of two task assignment models. The network structure in the figure is LeNet-5 [7]

Data Parallel

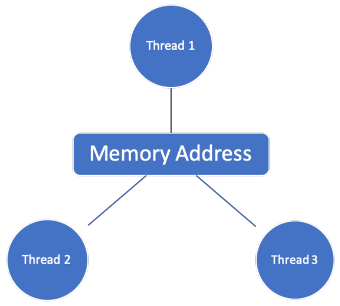
In the case of data parallel task assignment, the input data is split into multiple batches, and network weights update happens after every batch. In a multi-threading program, each batch will be split among threads to compute. Each thread takes one input sample, runs it through the entire network through forward propagation to get values called losses, and through backward propagation to get values called gradients (partial derivative of loss over every network weight). Afterwards, all threads synchronize to get global gradients from all local gradients, and use the global gradients to update the network weights before going into the next batch.

Model Parallel

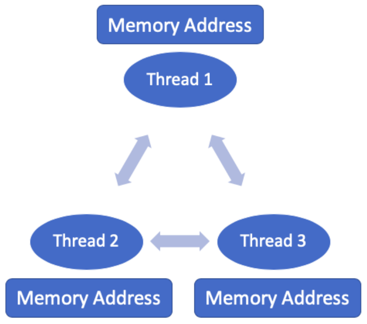
In the case of model parallel assignment, the computation in each network layer is split among threads. As all threads enter one layer of the network, they each compute one region and synchronize to form one completed layer before going onto the next layer.

4.2.2 Synchronization Model

In either assignment model mentioned above, a synchronization model is needed for threads to merge individual work.



(a) Shared Memory



(b) Message Passing

Figure 4.2: Graphic representation of two synchronization models.

Shared Memory

In the case of shared memory synchronization model, all threads merge their local result by accessing and modifying a shared memory region. This might raise contention issue, which requires extra mechanism such as locks to ensure correctness.

Message Passing

In the case of message passing synchronization model, all threads merge their local result by sending messages to each other, which contain data to be shared. Within the message passing synchronization, we additionally investigate active message passing, a novel form of message passing where user processes can send messages to each other while bypassing the operating system (via user-level interrupt).

User-Level Interrupt

Normally the kernel is responsible for acknowledging the interrupt and delivering a notification to the appropriate user process. User Level Interrupts (ULI) save the cost of these expensive context switching procedures by allowing the interrupt handling to happen in user space. An ongoing research project led by Prof. Seth Goldstein has established a ULI library implementation. In this project, in order to include ULI as a synchronization model, we have written a generalized NoC simulator within gem5 to provide an accurate representation of network latency between cores, which will lead to a better representation of active messages (user-level messages) in general.

4.3 Network on Chip

In order for gem5 to simulate active message passing with accurate clock cycle counts, we implemented a network on chip (NoC) model to realize the message traffic. We use a 2D mesh topology for the extensibility to multiple cores. We implement virtual channel allocation (VC) for flow control, and implement modified version on deterministic YX-routing for routing algorithm.

4.4 Benchmark

We train and test LeNet-5 (implemented in C for the use of parallel libraries) with MNIST dataset on the task of handwritten digit classification. The input data contains 60K images of handwritten digits for training and 10K for testing. The LeNet-5 network contains 60K parameters and 700K operations per forward and backward pass. Each benchmark adapts a different parallel design described in the table below. We choose LeNet for benchmark instead of a full-size pose estimation NN due to time constraint. We hope the optimal parallel design discovered for this smaller benchmark can become building blocks for optimizing a larger pose estimation NN, since they share common types of network layers.

Number of Threads	Assignment Model	Synchronization Model	Network Scale
1	None	None	1x, 2x
{2, 4, 8}	Data Parallel	Shared Memory	1x, 2x
{2, 4, 8}	Data Parallel	Message Passing	1x, 2x
{2, 4, 8}	Model Parallel	Shared Memory	1x, 2x
{2, 4, 8}	Model Parallel	Message Passing	1x, 2x

Table 4.1: A Table of Benchmark Configurations

During the experiments we encountered several limitations that we did not foresee, mostly in the benchmarking for ULI. We planned to run the ULI implementation in gem5 with the previously mentioned NoC in order to obtain accurate timing, but the training data for the full NN turned out to be too large for the default gem5 disk image. By the time we found a fix of building a large disk image, we concluded that we do not have enough time to run the full simulation in gem5. So, we resorted to breaking down the computation at each layer for the OpenMPI implementation and the ULI implementation for an analogous comparisons between the two methods of message passing. Thus, the ULI benchmark was removed from the benchmark table and replaced by a separate analysis included in the section below.

4.5 Benchmark Result and Analysis

To interpret the legends in the following graphs:

D_ = data parallel assignment

M_ = model parallel assignment

_S = shared address synchronization

_M = message passing synchronization

1x, 2x = LeNet size

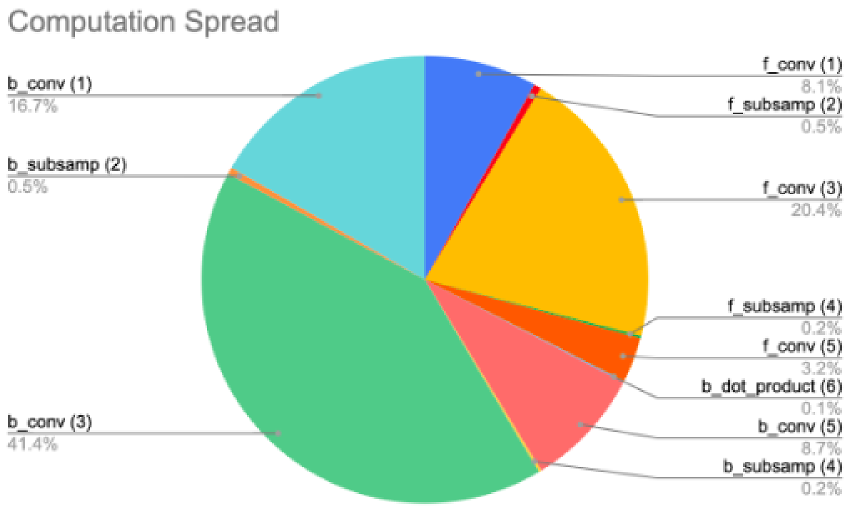


Figure 4.3: The computation split among different NN layers

According to the computation split result, most of the computation time concentrate on convolution layers. Convolution at backward propagation stage has heavier computation than at forward propagation stage. Thus specializing in accelerating convolution layers will achieve large performance improvement.

At both scales of LeNet, DS performs the best and closest to linear speedup (top two lines) 4.4. As LeNet size increases, DM gains speedup improvement, closing the gap between DM and DS, as DS has decreasing speedup as LeNet size grows. (The DS_2x line is below the DS_1x

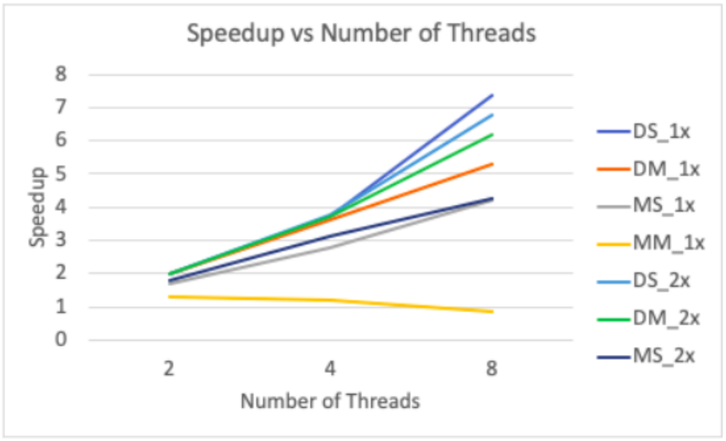


Figure 4.4: Speedup compared to sequential version as the number of thread increases

line, and the DM_2x line is above the DM_1x line). This gap closing effect was an interesting finding, and we performed further analysis described below in order to find the reason behind. MM fails to provide speedup due to frequent synchronization between every two layers at small task granularity.

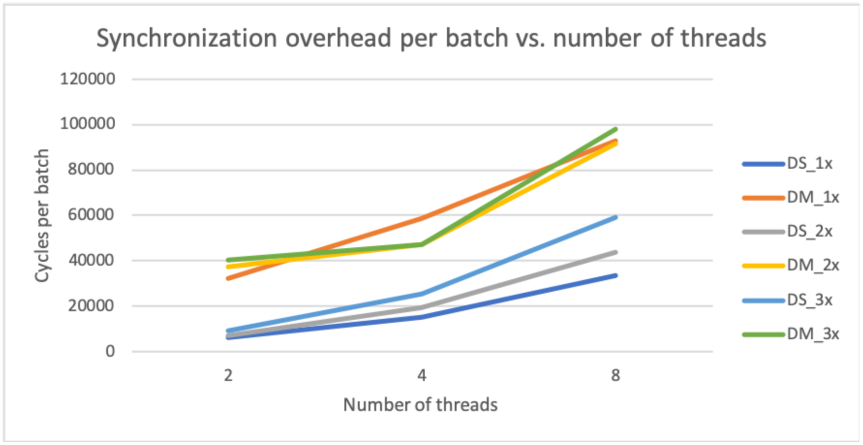


Figure 4.5: Synchronization overhead as the number of thread increases

In order to explain the behavior of the gap between DM and DS closing as network size increases, we recorded the synchronization overhead per training batch under different configurations. 4.5 The synchronization overhead was measured from replacing each thread's synchronization operations with empty code body. Thus, the measurement only includes setting up openmp/openmpi environments and openmp loop scheduling/openmpi message call overheads. The top cluster indicates that as the size of LeNet grows, the synchronization overhead for DM isn't affected much, since all three lines stay relatively close to each other. The bottom cluster indicates that as the size of LeNet grows, the synchronization overhead for DS also grows, since the gap between the three lines increases and the larger network size configuration remains on the top.

Since we weren't able to run a full-size ULI benchmark on the same LeNet due to gem5 limitation, we attempted to run smaller size benchmarks, but unfortunately ran into more gem5 limitation. MPI programs require a special command to run, $mpirun$, which was not supported within gem5. So, we tried to have a one-to-one comparison between a ping-pong program using Linux message queues with ULI. Unfortunately, gem5 does not support any send/recv system calls. As a last resort, we timed the MPI and message queue programs on Andrew machines against the ULI equivalent in gem5. The previous two were ran repeatedly to try to eliminate the scheduling uncertainty from a shared machine. The cycles counts were $80,000$, $6,500$, and 50 respectively. We were unable to provide a thorough trace of the MPI call stack, so we cannot say why the call was so costly. The message queue count was expected, as a system call took around 3,000 cycles each. The cost of a ULI is calibrated conservatively to the cycle count of a L1 to L1 cache transfer, which we measured locally to be 50 cycles.

4.6 Conclusion

From the benchmark outputs, we observe that in image classification context, most (98.5%) of the computation concentrate on the convolutional layers. Thus optimization that specializes on the convolutional structure would be very efficient. Out of all configurations, DS (data parallel assignment and shared memory synchronization) performs overall best and closest to linear speedup. Similarly DM (data parallel assignment and message passing synchronization) performs secondarily well. This indicates that data parallel is a better assignment model comparing to model parallel due to the "wide and short" NN structure. Having threads synchronize at the end of each batch (rightmost end of the network structure) leaves each thread large enough individual work before paying synchronization overhead. We also observe that the gap between DS and DM performances start closing as the network size grows. The explanation we found after further experiment is that the synchronization overhead per batch grows in positive relationship with respect to network size for DS configuration, whereas the overhead remains relatively constant with respect to network size for DM configuration. For shared memory synchronization, all threads need to compete for locks over network weights in order to update the global network weights correctly. As the network size grows, the number of weights increases, the number of locks to compete for also increases, thus the synchronization overhead increases due to higher contention. As for the message passing synchronization, the number of messages sent is dependent on the number of threads, thus synchronization overhead doesn't grow with the size of the network. Therefore, for solving smaller NN such as LeNet whose parameters are in the range of 60K, data parallel assignment and share memory synchronization is a good parallel model to use. For much larger NN such as VGG-16 with 138M parameters, data parallel assignment and message passing synchronization is a good parallel model to use due to non-exploding synchronization overhead. In fact, we believe implementing message passing with active message (with user-level interrupt library) will further improve the performance significantly, because our smaller scale benchmark shows that ULI message pass call takes about 1% cycles as the regular message passing call.

Chapter 5

Projecting the non-linear problem into linear space

The previous two projects focus on improving accuracy and runtime of deep learning-based pose estimations, respectively. This project solves pose estimation problem in probabilistic manner and aims to improve both accuracy and runtime by projecting the non-linear problem into linear space.

5.1 Problem Statement

Pose estimation is inherently non-linear [2]. As the result, using linear Kalman filter directly produces poor result. Linearization and other approximations bring sensitivity to initial estimation error and high computation cost. To address both issues, we can rewrite the measurement model in a truly linear form in order to obtain a linear update model, as inspired by [14].

5.2 Update Model

Assume point correspondences are known (if not, we can build closest point correspondence) and points in the sensor frame arrive one at a time (typically happens in robotics probing application [13]). We form the following formulation after each pair of sensor frame points b_i, b_{i+1} are received with their CAD model correspondences a_i, a_{i+1}:

$$a_i = \mathbf{R}b_i + \mathbf{t}$$

$$a_{i+1} = \mathbf{R}b_{i+1} + \mathbf{t}$$

Since all sensor frame points b_i come from the same point cloud, and all CAD model points a_i come from the same point cloud, there exists one rotation matrix **R** and translation vector **t** which transforms one point cloud to the other (object is rigid), and **R** and **t** are our goals to find.

We re-write the point pair relationship in quaternion form:

$$\tilde{\boldsymbol{a}}_i = \tilde{\boldsymbol{q}} \odot \tilde{\boldsymbol{b}}_i \odot \widetilde{\boldsymbol{q}}^* + \widetilde{\boldsymbol{t}}$$

$$\tilde{\boldsymbol{a}}_{i+1} = \tilde{\boldsymbol{q}} \odot \tilde{\boldsymbol{b}}_{i+1} \odot \widetilde{\boldsymbol{q}}^* + \widetilde{\boldsymbol{t}}$$

Where $\tilde{\boldsymbol{p}} = (0, \mathbf{p})$ is the quaternion form of a point **p**, $\tilde{\boldsymbol{q}}$ is the rotation in unit quaternion form, with conjugate $\widetilde{\boldsymbol{q}}^*$. $\odot$ is quaternion multplication. $\widetilde{\boldsymbol{t}} = (0,$ **t)** is the translation vector **t** in quaternion form.

Then we subtract the second equation from the first, and multiply both sides by $\tilde{\boldsymbol{q}}$ on the right.

$$\widetilde{\boldsymbol{a}}_i - \widetilde{\boldsymbol{a}}_{i+1} = \widetilde{\boldsymbol{q}} \odot \left(\widetilde{\boldsymbol{b}}_i - \widetilde{\boldsymbol{b}}_{i+1}\right) \odot \widetilde{\boldsymbol{q}}^*$$
$$\Rightarrow (\widetilde{\boldsymbol{a}}_i - \widetilde{\boldsymbol{a}}_{i+1}) \odot \widetilde{\boldsymbol{q}} = \widetilde{\boldsymbol{q}} \odot \left(\widetilde{\boldsymbol{b}}_i - \widetilde{\boldsymbol{b}}_{i+1}\right)$$

Move everything to the left hand side, and use matrix form of quaternion multiplication:

$$\Rightarrow (\widetilde{\boldsymbol{a}}_i - \widetilde{\boldsymbol{a}}_{i+1}) \odot \widetilde{\boldsymbol{q}} - \widetilde{\boldsymbol{q}} \odot \left(\widetilde{\boldsymbol{b}}_i - \widetilde{\boldsymbol{b}}_{i+1}\right) = \mathbf{0}$$

$$\Rightarrow \begin{bmatrix} \tilde{\boldsymbol{a}}_v(0) & -\boldsymbol{a}_v^T \\ \boldsymbol{a}_v & \boldsymbol{a}_v^\times + \tilde{\boldsymbol{a}}_v(0)\boldsymbol{I}_3 \end{bmatrix} \widetilde{\boldsymbol{q}} - \begin{bmatrix} \tilde{\boldsymbol{b}}_v(0) & -\boldsymbol{b}_v^T \\ \boldsymbol{b}_v & -\boldsymbol{b}_v^\times + \tilde{\boldsymbol{b}}_v(0)\boldsymbol{I}_3 \end{bmatrix} \widetilde{\boldsymbol{q}} = \mathbf{0}$$

Where $a_v = a_i - a_{i+1}$, $b_v = b_i - b_{i+1}$, and $\mathbf{p}^\times$ is the skew-symmetric matrix formed from the vector $\mathbf{p}$. We can get form $\mathbf{H}\widetilde{\boldsymbol{q}} = \mathbf{0}$, where $\mathbf{H}$ is a 4 by 4 matrix

$$\boldsymbol{H} = \begin{bmatrix} 0 & -(\boldsymbol{a}_v - \boldsymbol{b}_v)^T \\ (\boldsymbol{a}_v - \boldsymbol{b}_v) & (\boldsymbol{a}_v + \boldsymbol{b}_v)^\times \end{bmatrix} \in \mathbb{R}^{4\times 4}$$

Note that $\tilde{\boldsymbol{a}}_v(0) = \tilde{\boldsymbol{b}}_v(0) = 0$ because $\tilde{\boldsymbol{a}}_v = (0, \boldsymbol{a}_v)$, likewise for $\tilde{\boldsymbol{b}}_v(0)$. The conversion between quaternion multiplication and matrix multiplication is described in 2.1.1.

From $\mathbf{H}\widetilde{\boldsymbol{q}} = \mathbf{0}$ we can find $\widetilde{\boldsymbol{q}}$ which lies in the null space of $\mathbf{H}$. After finding the rotation part, we can find the translation part $\widetilde{\boldsymbol{t}}$ by adding the equations of a_i and a_{i+1}.

$$\tilde{\boldsymbol{a}}_i + \tilde{\boldsymbol{a}}_{i+1} = \tilde{\boldsymbol{q}} \odot \tilde{\boldsymbol{b}}_i \odot \widetilde{\boldsymbol{q}}^* + \widetilde{\boldsymbol{t}} + \tilde{\boldsymbol{q}} \odot \tilde{\boldsymbol{b}}_{i+1} \odot \widetilde{\boldsymbol{q}}^* + \widetilde{\boldsymbol{t}}$$

$$\Rightarrow \widetilde{\boldsymbol{t}} = \frac{(\tilde{\mathbf{a}}_i + \tilde{\mathbf{a}_{i+1}}) - \tilde{\mathbf{q}} \odot (\tilde{\mathbf{b}}_i - \tilde{\mathbf{b}_{i+1}}) \odot \tilde{\mathbf{q}}^*}{2}$$

5.2.1 Linear Filter

Now in order to obtain an estimate of $\widetilde{\boldsymbol{q}}$ from $\mathbf{H}\widetilde{\boldsymbol{q}} = \mathbf{0}$, we use a Bingham distribution to model the uncertainty in $\widetilde{\boldsymbol{q}}$. $p(\widetilde{\boldsymbol{q}}) = \frac{1}{N_1}\exp(\widetilde{\boldsymbol{q}}^T \underbrace{\boldsymbol{M}_{k-1}\boldsymbol{Z}_{k-1}\boldsymbol{M}_{k-1}^T\widetilde{\boldsymbol{q}})}_{D_1}$. In the Kalman filter context, the state vector is the pose $\widetilde{\boldsymbol{q}}$ to be estimated. In this work, only static estimation is involved, thus there is no need for a motion model to evolve the pose estimate over time. $\hat{\widetilde{\boldsymbol{q}}}_{t|t-1} = \hat{\widetilde{\boldsymbol{q}}}_{t-1|t-1}$

Uncertainty in sensor measurements a_i, b_i are modeled by Gaussian.

Recall the earlier obtained

$$\boldsymbol{H}\left(\boldsymbol{a}_i, \boldsymbol{a}_{i+1}, \boldsymbol{b}_i, \boldsymbol{b}_{i+1}\right)\widetilde{\boldsymbol{q}} = \mathbf{0}$$

In the Kalman filter context, this will be our pseudo-measurement model $\mathbf{z}_k = \mathbf{H}\widetilde{\boldsymbol{q}}$, where $\mathbf{z}_k$ is forced to be 0. [16].

Incorporating the sensor noise, we re-write the equation as:

$$\boldsymbol{H}\left(\boldsymbol{a}_1^s, \boldsymbol{a}_2^s, \boldsymbol{b}_1^s, \boldsymbol{b}_2^s\right)\widetilde{\boldsymbol{q}} + \boldsymbol{G}(\widetilde{\boldsymbol{q}})\mu = \mathbf{0}$$

$G(\widetilde{\boldsymbol{q}})\mu$ is a zero mean Gaussian noise, $\mathcal{N}(0, \boldsymbol{Q})$

The probability of obtaining a sensor measurement z_k, given the state $\widetilde{\boldsymbol{q}}_k$ is:

$$p(\mathbf{z}_k \mid \widetilde{\boldsymbol{q}}_k) = \frac{1}{N_2} exp(-\frac{1}{2}(\mathbf{z}_k - h(\widetilde{\boldsymbol{q}}_k))^T \mathbf{Q}_k^{-1}(\mathbf{z}_k - h(\widetilde{\boldsymbol{q}}_k)))$$

where $h(\widetilde{\boldsymbol{q}}_k)$ is the expected sensor measurement and $\mathbf{Q}_k$ is the measurement uncertainty.

From the earlier $\mathbf{H}\widetilde{\boldsymbol{q}} = \mathbf{0}$ equation, the measurement is set to z_k = 0 and the measurement model is set to $h(\widetilde{\boldsymbol{q}}_k) = \mathbf{H}\widetilde{\boldsymbol{q}}_k$. Here z_k = 0 is a pseudo-measurement [16].

Thus plugging in $\mathbf{H}\widetilde{\boldsymbol{q}} = \mathbf{0}$ and z_k = 0 in the previous equation we get:

$$p(\mathbf{z}_k \mid \widetilde{\boldsymbol{q}}_k) = \frac{1}{N_2} exp(-\frac{1}{2}(\mathbf{H}\widetilde{\boldsymbol{q}}_k^T \mathbf{Q}_k^{-1}(\mathbf{H}\widetilde{\boldsymbol{q}}_k)) = \frac{1}{N_2} exp(\widetilde{\boldsymbol{q}}_k^T \mathbf{D}_2 \widetilde{\boldsymbol{q}}_k)$$

Where the new Bingham parameter part can be identified as $\mathbf{D}_2 = \frac{1}{2}(-\mathbf{H}^T\mathbf{Q}_k^{-1}\mathbf{H})$. Thus we have derived the measurement to be Bingham distribution.

Assuming the measurements are all independent of each other, we can apply Bayes rule to get the the updated state given the current state estimate and measurement:

$$p(\widetilde{\boldsymbol{q}}_k \mid \mathbf{z}_k) \propto p(\widetilde{\boldsymbol{q}}_k)p(\mathbf{z}_k \mid \widetilde{\boldsymbol{q}}_k)$$

$$\propto \frac{1}{N_1} exp(\widetilde{\boldsymbol{q}}_k^T \mathbf{D}_1 \widetilde{\boldsymbol{q}}_k) \frac{1}{N_2} exp(\widetilde{\boldsymbol{q}}_k^T \mathbf{D}_2 \widetilde{\boldsymbol{q}}_k)$$

$$\propto exp(\widetilde{\boldsymbol{q}}_k^T \mathbf{M}_k \mathbf{Z}_k \mathbf{M}_k^T \widetilde{\boldsymbol{q}}_k)$$

We can see that $p(\widetilde{\boldsymbol{q}}_k \mid \mathbf{z}_k)$ is a Bingham distribution with new parameters $\mathbf{M}_k\mathbf{Z}_k\mathbf{M}_k^T$ obtained from the product of two Binghams.

5.2.2 Simultaneous Multi-measurement Update

In the previous section, we update the state once every pair of measurements is received, until a convergence condition is met or maximum iterations of updates is reached. In order to speedup the process, we can update after a mini-batch of measurement pairs are received.

Similar to the measurement formulated earlier, we have

$$p(\mathbf{z}_k \mid \mathbf{q}_k) = \prod_{j=1}^{m} \frac{1}{N_2^j} exp(-\frac{1}{2}(\mathbf{H}_j\mathbf{q}_k)^T\mathbf{Q}^{-1}(\mathbf{H}_j\mathbf{q}_k))$$
$$= \frac{1}{N_3} exp(\mathbf{q}_k^T\mathbf{D}_3\mathbf{q}_k)$$

, where

$$\mathbf{D}_3 = \frac{1}{2}\sum_j(-\mathbf{H}_j^T\mathbf{Q}^{-1}\mathbf{H}_j)$$

and

$$N_3 = \prod_{j=1}^{m} N_2^j$$

Now the measurement consists of a joint of $\mathbf{H}_j$s, each of them being a measurement pair in the mini-batch $\mathbf{H}_j(a_j, b_j, a_{j+1}, b_{j+1})$

Then this joint measurement is used in Bayes rule to update the state $p(\widetilde{\boldsymbol{q}}_k \mid \mathbf{z}_k)$.

5.3 Evaluation

We run synthesized and real-world point cloud registrations with the Bingham Filtering (BF) method described in this chapter, Dual Quaternion Filter (DQF) method described in this paper [14], Extended Kalman Filter (EKF) method described in this paper [10], and Unscented

Kalman Filter (UKF) method described in this paper [8]. Adapting to the setups used in their original papers, the state vectors in both EKF and UKF are $\mathbf{x} = [t_x, t_y, t_z, \theta_x, \theta_y, \theta_z]^T$ instead of the quaternions used in BF and DQF, and the uncertainties are modeled by Gaussian distribution instead of the Bingham distribution in BF.

	RMS(mm) (Expt. 1)	RMS(mm) (Expt. 2)	RMS(mm) (Expt. 3)
BF	0.00	2.29	12.12
DQF	6.14	7.70	70.99
UKF	2.67	4.91	12.25
EKF	94.65	21.97	56.52

Table 5.1: Experiments on Synthetic Point Clouds. Experiment 2, 3 have noises from range [-2mm, 2mm], [-10mm, 10mm]

Initial Bingham parameters are $M_0 = I_{4\times4}$ and $Z_0 = diag(0,1,1,1) * 10^{-300}$, which doesn't contain any prior information and has high uncertainty. The Gaussian distribution used to model measurement have mean 0 and standard deviation 0.2. The maximum number of updates is set to 100.

Since we use synthetic data, the initial point cloud is randomly draw from the pose space, and the target point cloud is generated by applying a transformation on the initial point cloud. The transformation parameters are also uniformly drawn. In experiment 1, no noise is added to the target point cloud. In experiment 2 and 3, noises from range [-2mm, 2mm] and [-10mm, 10mm] respectively are added the target point cloud.

The experiment result shows that projecting the non-linear problem into linear space and then using linear Kalman filter directly, with state modeled by a Bingham distribution (BF method), can make the registration robust to sensor noise, since the error remains relatively low as the amount of noise increases.

In addition, we estimate the relative pose between the camera and robot frames by tracking

	Time (ms)	RMS(mm)
BF	25.8	8.93
BFM	2.1	5.91
Horn	56.8	4.88

Table 5.2: Robot Tool Tip Tracking Runtime and Accuracy

the robot arm tool-tip using a stereo camera. We use Horn's method [5], which uses ICP, as a comparison.

The experiment result shows when the BF method performs an update with mini-batch 5.2.2, while being able to achieve similar optimal accuracy as Horn's method, it is able to achieve 28x speedup on convergence time. At each iteration of Horn's method, update needs to be performed with all measurements collected so far. At each iteration of BFM method, update only needs to be performed with the most recent mini-batch of measurements collected.

5.4 Conclusion

Comparing to other filtering-based pose estimations, a Bingham filtering-based pose estimation that re-formulates non-linear measurement model into linear space can achieve higher accuracy and speedup. Using unit quaternion as rotation representation and Bingham distribution as rotation distribution avoids discontinuous jump in orientation parameter space and represents rotation symmetry better. Projecting the non-linear problem into linear space by the concept of pseudo-measurement avoids sensitivity to initial estimation error and reduces computation time, comparing to other methods that use linearization and other approximations. The method discussed in this thesis uses the assumption of known point correspondence between two point clouds. Under the situation when point correspondencs are unknown, iterative closest point (ICP) [1] related methods are needed to iteratively find point correspondence while performing least squares optimization.

Chapter 6

Conclusion and Future Works

This thesis summarizes three projects aiming to improve accuracy and runtime for deep learning-based and probabilistic pose estimation. The first project aims to improve accuracy by describing the output as a pose distribution. The distribution gives information about one or more correct poses and uncertainty associated with each. With additional information about uncertainty, robotic applications can have higher success rate due to the ability to take alternative actions when uncertainty is high. The alternative actions, such as moving to a location with clearer view, can help find new pose with lower uncertainty to perform action with. The experiment result shows histogram distributions obtained from confusion matrices built from validation and synthetic data can achieve great performance improvement. However, we need further analysis to prove that the performance increase doesn't simply come from the bias in the dataset used. In future works, we can minimize the similarity between the data used to build confusion matrix and the data used to test, and see whether the confusion matrix method will retain high performance comparing to other methods.

Deep learning-based pose estimation methods are used in the first project as base predictors to fit distribution, and neural network (NN) computation tends to be computationally expensive. The second project aims to improve runtime by exploring a guideline for designing CPU accelerated NN. With parallel version of the program properly designed with parallel libraries in C, CPU utilization can be well utilized to provide significant speedup to deep learning-based pose estimation, alongside or in the absence of GPU resources. GPUs tend to be the resource bottleneck while CPUs stay idle. Having CPUs well utilized to aid NN acceleration will be very helpful. The experiment result shows that for small size NN such as LeNet, the configuration of data parallel assignment model and shared memory synchronization model can achieve the best and closest to linear speedup among all configurations. As the network size grows, the performance gap between shared memory synchronization and message passing synchronization is closing (shared memory performance goes down, message passing synchronization performance goes up). This is because the synchronization traffic overhead of message passing model doesn't grow with the network size as the shared address synchronization does. Although the additional experiment proves this trend, it only has the network scale up to three times of the original LeNet due to system memory constraint. Thus we still don't observe at which point the speedup lines of the two configurations cross (the crossing point will be the exact network size at which the parallel model configuration needs to change). In future works, we should find hardware resource with much larger memory to perform experiments with larger network scales in order to find the crossing point. Also, we can find a way to simulate a larger network with the re-using of a small physical memory.

Besides deep learning-based pose estimation, we also look into improving accuracy and runtime of probabilistic pose estimation. Instead of using Gaussian distribution to describe rotation parameters like many existing methods, we use Bingham distribution with unit quaternion to represent rotation parameters so there is no jumps in orientation space and rotation symmetry is captured. Instead of linearizing the non-linear measurement model before using linear Kalman filter, we re-formulate the non-linear measurement model into linear form with the concept of pseudo-measurement. Then we use linear Kalman filter directly. The experiment result shows that our method is more robust to initial estimation error and sensor noise comparing to other filtering methods using linearization and approximations. Our method achieves significant speedup and equally optimal accuracy comparing to the traditional ICP related Horn's method [5]. Our method assumes known point correspondence. In future works, we will explore the case when point correspondence is unknown. For example, we can use iterative closest point-related methods to establish point correspondence with the aid of additional surface normal information [11].

Bibliography

[1] Paul J Besl and Neil D McKay. A method for registration of 3d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1992. 5.4

[2] Seth D. Billings, Emad M. Boctor, and Russell H. Taylor. Iterative Most-Likely Point Registration (IMLP): A Robust Algorithm for Computing Optimal Shape Alignment. *PLoS ONE*, 10(3):e0117688, Mar 2015. 5.1

[3] Christopher Bingham. An antipodally symmetric distribution on the sphere. *Ann. Statist.*, 2(6):1201–1225, 11 1974. 2.1.2

[4] Igor Gilitschenski, Gerhard Kurz, Simon Julier, and Uwe Hanebeck. Unscented orientation estimation based on the bingham distribution. *IEEE Transactions on Automatic Control*, 61, 11 2013. 2.1.2

[5] Berthold K. P. Horn. Closed-form solution of absolute orientation using unit quaternions. *J. Opt. Soc. Am. A*, 4(4):629–642, Apr 1987. 5.3, 6

[6] Gerhard Kurz, Igor Gilitschenski, Simon Julier, and Uwe D. Hanebeck. Recursive estimation of orientation based on the bingham distribution. *CoRR*, abs/1304.8019, 2013. 2.1.2

[7] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998. (document), 4.1

[8] M. H. Moghari and P. Abolmaesumi. Point-based rigid-body registration using an unscented kalman filter. *IEEE Transactions on Medical Imaging*, 26(12):1708–1728, Dec 2007. 5.3

[9] Brian Okorn, Mengyun Xu, Martial Hebert, and David Held. On orientation distributions for object pose estimation. submitted. (document), 3.1

[10] Xavier Pennec and Jean-Philippe Thirion. A framework for uncertainty and validation of 3-d registration methods based on points and frames. *International Journal of Computer Vision*, 25(3):203–229, Dec 1997. 5.3

[11] K. Pulli. Multiview registration for large data sets. In *Second International Conference on 3-D Digital Imaging and Modeling (Cat. No.PR00062)*, pages 160–168, Oct 1999. 6

[12] Arun Srivatsan Rangaprasad. *Probabilistic Approaches for Pose Estimation*. PhD thesis, Pittsburgh, PA, May 2018. (document)

[13] Arun Srivatsan Rangaprasad, Elif Ayvali, Long Wang, Rajarshi Roy, Nabil Simaan, and Howie Choset. Complementary model update: A method for simultaneous registration and stiffness mapping in flexible environments. In *Proceedings of 2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 924–930, May 2016. 5.2

[14] Arun Srivatsan Rangaprasad, Gillian T. Rosen, Feroze Naina Mohamed Dheen Mohamed Ismail, and Howie Choset. Estimating se(3) elements using a dual quaternion based linear kalman filter. In *Proceedings of Robotics: Science and Systems*, June 2016. 5.1, 5.3

[15] Arun Srivatsan Rangaprasad, Mengyun Xu, Nicolas Zevallos, and Howie Choset. Bingham distribution-based linear filter for online pose estimation. In *Proceedings of Robotics: Science and Systems, 2017*, July 2017. (document), 2.1.2, 2.1.2

[16] P. W. Richards. Constrained kalman filtering using pseudo-measurements. In *IEE Colloquium on Algorithms for Target Tracking*, pages 75–79, May 1995. 5.2.1

[17] Moshe Shoham and Fu-Hua Jen. On rotations and translations with application to robot manipulators. *Advanced Robotics*, 8(2):203–229, 1993. 2.1.1

[18] Rangaprasad Arun Srivatsan, Mengyun Xu, Nicolas Zevallos, and Howie Choset. Probabilistic pose estimation using a bingham distribution-based linear filter. *The International Journal of Robotics Research*, 37(13-14):1610–1631, 2018. (document), 2.1

[19] Julian Straub, Trevor Campbell, Jonathan P How, and John W Fisher III. Efficient global

point cloud alignment using bayesian nonparametric mixtures. In *CVPR*, 2017. 3.2

[20] Chen Wang, Danfei Xu, Yuke Zhu, Roberto Martín-Martín, Cewu Lu, Li Fei-Fei, and Silvio Savarese. Densefusion: 6d object pose estimation by iterative dense fusion. In *CVPR*, 2019. 1, 3.1.1

[21] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. 2018. 1, 3.1.1