Discriminative Optimization: Theory and Applications to Computer Vision

Submitted in partial fulfillment of the requirements for
the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Jayakorn Vongkulbhisal

B.Eng., Information and Communication Engineering, Chulalongkorn University
M.S., Electrical and Computer Engineering, Carnegie Mellon University

Carnegie Mellon University
Pittsburgh, PA

February, 2018

© Jayakorn Vongkulbhisal, 2018
All Rights Reserved

Acknowledgments

This thesis would not be possible without the help and supervision of my two advisors. I would like to thank Fernando De la Torre for his constant support and advice on my research, and also for his encouragements when it did not go as well as expected. I am also indebted to João Paulo Costeira for his guidance and dedication on issues both inside and outside academia. I will always remember the care he had given me when my health was poor. Both of them provided me a great PhD experience, and I appreciate everything they have done for me.

Besides my advisors, I would like to express my gratitude to my thesis committee, José Moura, Marko Stošić, and Carlo Tomasi, for their interest in my work. Their feedback and suggestions were invaluable to the completion of this work.

I wish to thank the CMU|Portugal Program and Fundação para a Ciência e a Tecnologia [UID/EEA/50009/2013] for providing me this PhD opportunity. I also appreciate CMU|Portugal staffs who made sure every transition across the Atlantic was a smooth one. I give my special thanks to Ana Mateus, who helped me navigate all the complex paperwork during my study and ensure everything was in order.

Throughout the years, I have had the chance to collaborate and share memorable moments with many awesome people. From the Portugal side (incl. CMU|Portugal): Ricardo Cabral, Manuel Marques, Qiwei Han, João Carvalho, Susana Brandão, Zita Marinho, Shaolong Liu, Sabina Zejnilovic, João Xavier, Shanghang Zhang, and Senbo Fu. From the CMU side: Dong Huang, Wen-Sheng Chu, Yi Wang, Zehua Huang, Shitong Yao, Beñat Irastorza Ugalde, Francisco Vincente, Yasuhide Hyodo, Ada Zhang, Stanislav Panev, Dingwen Zhang, Kaili Zhao, Feng Zhou, Xuehan Xiong, Chieko Asakawa, Kris Kitani, and Tatsuya Ishihara. I appreciate all the stimulating discussions and the assistance I received from them.

Outside of my academic circle, many friends–Rathapon Saruthirathanaworakun, Pralom Boonrusssamee, Nat Suriyabantoeng, Pongsthorn Tangnaikuntham, Bencharassamee Rujraweehiran, Phakajira Sricharoen, Tomin Liu, Korapat Pruekchaikul, Nunthanuth Jiaramanytwesin, Rajitha Navarathna–have made my life both in Lisbon and Pittsburgh much more enjoyable. I also would like to thank the staffs at the Royal Thai Embassy in Lisbon for looking after our small community and making us feel more like home.

Lastly, I wish to thank my parents, without whose love and support I would not be able to overcome many difficult times and complete this thesis.

Abstract

Many computer vision problems are formulated as the optimization of a cost function. This approach faces two main challenges: *(i)* designing a cost function with a local optimum at an acceptable solution, and *(ii)* developing an efficient numerical method to search for one (or multiple) of these local optima. While designing such functions is feasible in the noiseless case, the stability and location of local optima are mostly unknown under noise, occlusion, or missing data. In practice, this can result in undesirable local optima or not having a local optimum in the expected solution. On the other hand, numerical optimization algorithms in high-dimensional spaces are typically local and often rely on expensive first or second order information to guide the search.

To overcome these limitations, we propose Discriminative Optimization (DO), a method that learns search directions from data without the need of a cost function. Specifically, DO learns a sequence of updates in the search space that leads to stationary points corresponding to the desired solutions. Using training data, DO can find solutions that are more robust to perturbation of real data, unlike conventional optimization which may fail if there is a mismatch between the cost function and the noise distribution. We provide a formal analysis of DO, proving its convergence in the training phase. We also explore the relation between DO and generalized convexity and monotonicity, and show that the conditions for the convergence of DO are broader than those required by convexity. In terms of applications, we illustrate DO's potential in the problems of 3D point cloud registration, camera pose estimation, and image denoising. We show that DO can generally outperform state-of-the-art algorithms in terms of accuracy, robustness to perturbations, and computational efficiency.

Contents

List of Tables

List of Figures

List of Symbols

The following list describes the symbols in this thesis.

$\mathbb{R}$	The set of real numbers
$\mathbb{R}^n$	The set of n-dimensional vectors
$\mathbb{R}^{n \times m}$	The set of $n \times m$ matrices
x, X	A scalar
$\mathbf{x}$	A vector
$\mathbf{X}$	A matrix
$[\mathbf{x}]_i$	The i^{th} element of $\mathbf{x}$
$[\mathbf{X}]_{i,j}$	The element in row i and column j of $\mathbf{X}$
$[\mathbf{X}]_{i,:}$	Row i of $\mathbf{X}$ as a row vector
$\|\mathbf{x}\|_2$	ℓ_2 norm of $\mathbf{x}$
$\|\mathbf{X}\|_F$	Frobenius norm of $\mathbf{X}$
$\mathbf{1}_n$	An n-dimensional vector of ones
$\mathbf{0}_n$	An n-dimensional vector of zeros
$\mathbf{e}_i$	A standard basis with 1 at the i element and 0 at others
$\mathbf{e}_0$	A vector of zeros (with an abuse of the notation on $\mathbf{e}_i$)
$f \circ g(x)$	Function composition $f(g(x))$
$\delta(\cdot)$	Dirac delta function
$\bigotimes$	Kronecker product
$\bigoplus$	Vector concatenation
∂f	Subdifferential of f

Chapter 1

Learning-Based Optimization

Mathematical optimization plays an important role for solving many computer vision tasks. For instance, optical flow, camera calibration, homography estimation, and structure from motion are tasks which are solved as optimization. Formulating computer vision tasks as optimization problems faces two main challenges: *(i)* Designing a cost function that has a local optimum that corresponds to a suitable solution. *(ii)* Selecting an efficient and accurate algorithm for searching the parameter space. Conventionally, these two steps have been treated independently, leading to different cost functions and search algorithms. However, in the presence of noise, missing data, or inaccuracies of the model, this conventional approach can lead to undesirable local optima or even not having an optimum in the *correct* solution. In this work, we propose a different view on how to model and solve these types of tasks. By exploiting the availability of data and ground truth, we propose Discriminative Optimization (DO), a learning-based approach that searches for a solution of a task instead of designing a cost function and searching for its optimum. The following section provides the motivation and examples for our learning-based approach.

1.1 Motivation

Many computer vision tasks involve finding the parameter $\mathbf{x} \in \mathbb{R}^p$ that solves a set of equations

$$\mathbf{g}_j(\mathbf{x}) = \mathbf{0}_d, j = 1, \dots, J, \tag{1.1}$$

where $\mathbf{g}_j : \mathbb{R}^p \to \mathbb{R}^d$ model the tasks of interest. For example, in the camera resection or homography estimation [53], the vector $\mathbf{x}$ may represent the camera parameters while $\mathbf{g}_j$ may represent the reprojection residual of the j^{th} feature correspondence. In most cases, noise, outliers, and other perturbations prevent the existence of an $\mathbf{x}$ that satisfies (1.1). As a result, we have to resort

to finding an $\mathbf{x}$ that minimizes the deviation of $\mathbf{g}_j$ from $\mathbf{0}_d$, *i.e.*, the *residuals*. This generally leads to an optimization problem of the form

$$\underset{\mathbf{x}}{\text{minimize}} \sum_{j=1}^{J} \varphi(\mathbf{g}_j(\mathbf{x})), \tag{1.2}$$

where $\varphi : \mathbb{R}^d \to \mathbb{R}$ is a penalty function, *e.g.*, ℓ_2 or ℓ_1 norms. Note that, since $\mathbf{g}_j(\mathbf{x})$ represents the residual to be penalized, we will refer to $\mathbf{g}_j$ as the *residual functions*.

Many techniques have been devised to solve (1.2). In the case when (1.2) is differentiable, a conventional approach is to search for a stationary point $\mathbf{x}_*$ using a fixed point iteration scheme, *e.g.*, gradient descent, and return $\mathbf{x}_*$ as the solution [17]. However, in order to use such approach, we need to consider a very important question: How should we select or design a φ that would lead to a desirable solution? Depending on the form of φ, the same iterative scheme could lead to different solutions for the same set of $\mathbf{g}_j$. This is problematic since it may be difficult to determine which solution is more preferable.

Instead of relying on the mentioned approach, our work considers a different setting. Suppose we are given a collection $\mathcal{D} = \{(\{\mathbf{g}_j^{(i)}\}_{j=1}^{J^{(i)}}, \mathbf{x}_*^{(i)})\}_{i=1}^{n}$, where $\mathbf{x}_*^{(i)}$ is *our preferred solution* to the set of equations $\mathbf{g}_j^{(i)}(\mathbf{x}_*^{(i)}) \approx \mathbf{0}_d, j = 1, \ldots, J^{(i)}$. Our question is: Can we leverage the data $\mathcal{D}$ to infer our preferred solution from a new set of equations $\mathbf{g}_j(\mathbf{x}) \approx \mathbf{0}_d, j = 1, \ldots, J$ without having to define what φ is? We provide the following two example problems as a motivation.

Example 1: Can you guess the number?

Let us consider the following *'Can you guess the number'* game: Suppose we are given three sets of input numbers with three outputs, can we guess the output for the fourth set?

Input set	Output
$\{\ 1, 2, 3\ \}$	2
$\{\ 2, 4, 4, 5, 6\ \}$	4
$\{\ -2, 1, 1\ \}$	1
$\{\ 1, 3, 4\ \}$	?

(1.3)

The answer is 3. We can see that the output is the middle number, *i.e.*, the median, of each set.

We can try this again, but with a new set of outputs:

Input set	Output
$\{\ 1, 2, 3\ \}$	2
$\{\ 2, 4, 4, 5, 6\ \}$	4.2
$\{\ -2, 1, 1\ \}$	0
$\{\ 1, 3, 4\ \}$	$?$

(1.4)

The answer this time is $2.\dot{6}$: the mean.

We may start to see the pattern of this game. In each case, the output is the solution of the optimization problem:

$$x_* = \arg\min_x \sum_{j=1}^{J} \varphi(x - x_j), \tag{1.5}$$

where $x_j, j = 1, \ldots, J$, are the elements in the input set, φ is some penalty function, and x_* is the output. In the above games, the explicit forms of φ are the absolute function and the squared function, *resp*.

Now that we find the pattern, let us try again with the following set of outputs:

Input set	Output
$\{\ 1, 2, 3\ \}$	2
$\{\ 2, 4, 4, 5, 6\ \}$	4.0774
$\{\ -2, 1, 1\ \}$	-0.2301
$\{\ 1, 3, 4\ \}$	$?$

(1.6)

Now the answer is not so obvious, even though we know the pattern. This is because the φ is not so simple:

$$\varphi(x) = |x|^{3.21} + |x|^{1.25}. \tag{1.7}$$

As can be seen, the form of φ may be hard to decipher from the given input sets, and we may not be able to figure what φ is in order to compute the output. Motivated by this '*Can you guess the number*' game, we ask the question: Can we solve for the output without knowing the penalty function that generates it? In particular, how can we use DO to obtain such output, and what is the class of functions that DO can handle? (Note: The solution to the above game is 2.5646, and DO got 2.5969.)

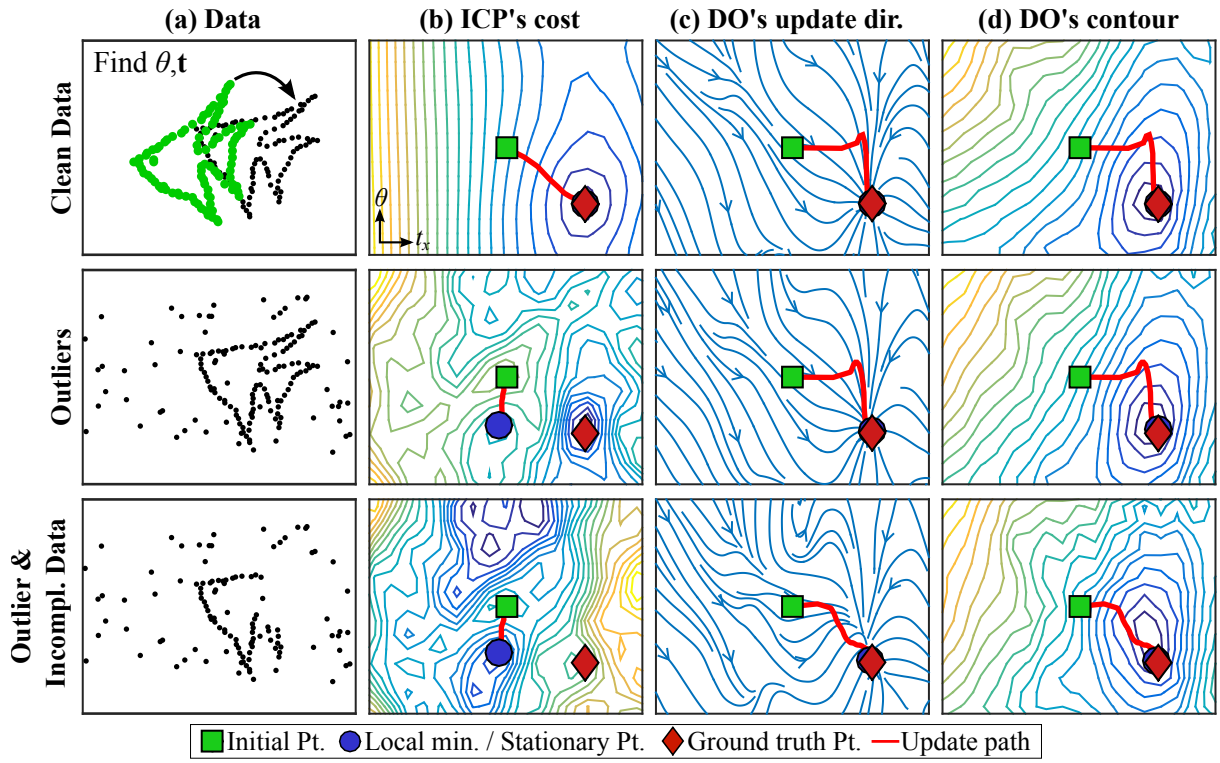


Figure 1.1: 2D point alignment using ICP and DO. (a) Data. (b) Level sets of the cost function for ICP. (c) Update directions of DO. (d) Inferred level sets of DO.

Example 2: Point cloud registration

Consider the task of 2D rigid point cloud registration (See Figure 1.1a). Given a model shape in green and the scene shape in black, we wish to solve for the rotation matrix $\mathbf{R} \in SO(2)$ and translation vector $\mathbf{t} \in \mathbb{R}^2$ that register the shapes together. The residual function for this task is given by

$$\mathbf{g}_{i,j}(\mathbf{R}, \mathbf{t}) = \mathbf{s}_i - \mathbf{R}\mathbf{m}_j - \mathbf{t}, \text{ for appropriate pair } (i, j), \tag{1.8}$$

where $\mathbf{m}_j = (x_j^t, y_j^t) \in \mathbb{R}^2, j = 1, \ldots, n$, is the coordinate of the j^{th} point of model shape, and $\mathbf{s}_i \in \mathbb{R}^2, i = 1, \ldots, m$, is that of the scene shape. With this residual function, we can formulate a combinatorial optimization problem

$$\underset{\substack{\mathbf{R} \in SO(2), \mathbf{t} \in \mathbb{R}^2 \\ \mathbf{P} \in \{0,1\}^{m \times n}, \mathbf{P}\mathbf{1}_n = \mathbf{1}_m}}{\text{minimize}} \quad \sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij} \|\mathbf{s}_i - \mathbf{R}\mathbf{m}_j - \mathbf{t}\|_2^2, \tag{1.9}$$

where $\mathbf{P}$ is the matching matrix. Figure 1.1b-top shows the level sets of the cost function (1.9)

under an ideal condition (*i.e.*, no noise, outliers, missing data)[1]. Observe that there is a well-defined optimum and it coincides with the ground truth (red diamond). Given this cost function, the next step is to find a suitable algorithm that searches for a local optimum. For this particular initialization (green square), the Iterative Closest Point (ICP) [15] algorithm converges to the ground truth (red diamond). However, in realistic scenarios with perturbations, there is no guarantee that there will be a good local optimum in the expected solution, and the number of bad local optima is also likely to increase. Figure 1.1b-middle and Figure 1.1b-bottom show the level set representation for the ICP cost function in the case of corrupted data. We can see that the shape of the cost functions have changed dramatically: there are more local optima, and the ground truth may not corresponds to any of them. In this case, the ICP algorithm with an initialization in the green square will converge to wrong optima. It is important to observe that the cost function is only designed to have an optimum at the correct solution in the ideal case, but little is known about the behavior of this cost function in the surrounding of the solution and how it will change with noise.

1.2 Thesis contributions

In this thesis, we address the following question: **Given a mathematical model (*i.e.*, a residual function) that characterizes a task of interest, how can we leverage the training samples to derive a robust algorithm for solving new instances of the same task?** To address this question, we propose Discriminative Optimization (DO). DO exploits the fact that we often have the training data with the desired solutions. Whereas conventional approaches find their solution by following a descent direction of a cost function, DO directly learns a sequence of update directions leading to a stationary point that is placed *by design* in the training data.

Compared with conventional approaches, DO has two main advantages. First, since DO's directions are learned from training data, they take into account the perturbations in the neighborhood of the ground truth, resulting in more robustness. As can be seen in Figure 1.1c, the update paths of DO change little despite strong perturbations in the data, and more importantly, the stationary points still coincide with the ground truth. Second, because DO does not optimize an explicit function (*e.g.*, ℓ_2 registration error), it is less sensitive to model misfit and more robust to different types of perturbations. Figure 1.1d illustrates the contour level inferred[2] from the update directions learned by DO. It can be seen that the curve levels have a local optimum on the

[1] We used the optimal matching at each parameter value to compute the cost.

[2] The contour level is approximately reconstructed using the surface reconstruction algorithm [52] from the update directions of DO for visualization purpose.

ground truth and fewer bad local optima than the cost surfaces in Figure 1.1b.

We named our approach 'Discriminative Optimization' to reflect the idea of learning to find a stationary point directly rather than optimizing a 'generative' cost function.

We provide theoretical analysis of DO. Specifically, we provide the conditions for the training error of DO to strictly decrease or converge to zero. We also explore the connection between DO and generalized convexity and generalized monotocity, which are important theoretical properties used to guarantee the convergence of many optimization algorithms [17, 87]. Through this connection, we show that the conditions for the convergence of DO are broader that those required by convexity. In terms of applications, we apply DO to the tasks of point cloud registration, camera pose estimation, and image denoising. We demonstrate that DO is computationally efficient, and the results are often superior to those of state-of-the-art algorithms for each problem. Finally, we propose four generalizations that enhance the capabilities of the original DO.

1.3 Organization

This thesis is organized as follows. Chapter 2 reviews related works, including the use of optimization for solving computer vision problems and previous works on learning cost functions and search algorithms. Chapter 3 describes the main DO algorithm. In Chapter 4, we provide a theoretical analysis on the convergence of DO and its relation to generalized convexity and generalized monotonicity. Applications of the algorithm to point cloud registration, camera pose estimation, and image denoising are described in Chapter 5. In Chapter 6, we propose four generalizations of DO: using inverse composition operations for updating the parameter vectors, representing the update as a combination of basis functions, accelerating DO with momentum, and constrained DO. Chapter 7 concludes the thesis and discusses future research directions.

Chapter 2

Related Work

In this chapter, we review previous work which are related to DO. In the first part, we discuss optimization problems in computer vision. In the second part, we review learning-based approaches that learn a suitable cost function for a given task. In the final part, we look at learning-based algorithms that directly learn to search for the solution without relying on an explicit cost function.

2.1 Optimization in computer vision

As mentioned in Section 1.1, many tasks in computer vision involve estimating a set of parameters $\mathbf{x} \in \mathbf{R}^p$ that satisfies a set of equations

$$\mathbf{g}_j(\mathbf{x}) = \mathbf{0}_d, j = 1, \ldots, J, \tag{2.1}$$

where $\mathbf{g}_j : \mathbf{R}^p \to \mathbf{R}^d$ models the task of interest. One common approach to tackle these problems is to formulating an optimization problem of the form

$$\begin{array}{ll} \text{minimize}_{\mathbf{x}} & \sum_{j=1}^{J} \phi(\mathbf{g}_j(\mathbf{x})) \\ \text{subject to} & \mathbf{x} \in \mathcal{S}, \end{array} \tag{2.2}$$

where ϕ is a function that penalizes the values of the residual functions $\mathbf{g}_j$, and $\mathcal{S}$ is the constraint set. The residual functions typically measures the difference between the observed data and the expected value of the task model. Examples of $\mathbf{g}_j$ are the reprojection residual in camera resection tasks; the difference between the pixel intensity of the template and the image in template tracking tasks; the difference between the pixel intensity of the observed image and the reconstructed image in denoising tasks; *etc*. After formulating the optimization in (2.2), it is then solved by searching for the parameters $\mathbf{x}_*$ that optimize its cost function.

Depending on the assumptions on the data nuisance (*e.g.*, noise and outliers), it is important to select an appropriate penalty function for the task. Due to the existence of its derivative, squared ℓ_2 function is typically the first choice for many computer vision applications [1, 5, 110]. However, it is known that ℓ_2 penalty is not robust to outliers. This leads to multiple research works on the properties of different penalty functions and estimation methods [6, 7, 69, 91]. Robust continuous functions, such as ℓ_1 function and Huber loss, are typically use for a moderate amount of noise and outliers [58]. To deal with more severe nuisances, several works [60, 109] proposed to use discontinuous functions, such as ℓ_0 or inlier count. Instead of using a single function, many works propose to use continuation methods to deform the cost function as the optimization progresses in order to increase robustness and reduce the chance of getting caught in bad local minima [16, 72]. In practice, however, it is a challenge to decide which penalty function is suitable for the problem at hand. This challenge is further complicated by the fact that many computer vision tasks, *e.g.*, optical flow estimation and surface reconstruction, are ill-posed and require regularization [14]. As a result, a combination of different penalty functions must be selected for the data and the regularization terms, together with the hyperparameters to weight them. Due to a large variety of choices, it is not trivial to identify a suitable cost function for solving a problem.

2.2 Learning cost functions

Instead of manually selecting a cost function, several works proposed to use machine learning techniques to learn a cost function from available training data. In [3], Avidan proposed to use kernel SVM classification score as the cost function for image-based object tracking. Similarly, Liu [63] and Nguyen and De la Torre [77] proposed to use, respectively, boosting and metric learning for face alignment. For these approaches, since the learned cost function is differentiable w.r.t the parameters, they use Gauss-Newton method, similar to that in Lucas-Kanade [5], to optimize the learned cost for the unseen data. In [80], Paliouras and Argyros used nonlinear regressors and random forests with various features to learn the cost function for hand pose estimation in RGBD images, where the learned cost function was optimized using particle swarm optimization.

A major downside of these approaches is that they need to impose the form of the cost function, *e.g.*, [77] requires the cost function to be quadratic. This restricts the class of problems that they can solve. Another problem with these approaches is that they work with image patches or their features, thus it is not clear how to extend these approaches to other problems, *e.g.*, point cloud registration, where the inputs are not structured.

Besides computer vision, learning cost functions have also been used in planning and control, where the problem is called *inverse optimal control* (IOC) or *inverse reinforcement learning* [75]. IOC involves learning unknown reward functions for a Markov decision process from a set of demonstrations of expert [4]. By assuming the actions of the experts are near-optimal, IOC trains the learning agents so that they imitate the decision making of the expert. However, the problem setting of IOC differs from that of cost function learning in computer vision: the training set in IOC typically comprises sequences of actions where the goals may not be obvious in each step, while in computer vision the training set provides the goals but not the trajectories leading to the goals. We do not focus on IOC in our work.

2.3 Learning search directions

Instead of using search directions from a cost function, recent works proposed to use learning techniques to directly compute such directions. This is done by learning a sequence of regressors that maps a feature vector to an update vector that points to the desired parameters. We refer to these algorithms as supervised sequential update (SSU). Since DO extends the work of SSU, we will first review previous works on SSUs in the literature. Then, we will review compare SSUs with gradient boosting, which relies on a similar mechanism to SSUs.

2.3.1 Supervised sequential update (SSU)

Supervised sequential update (SSU) algorithms are a class of supervised learning algorithms that predict a set of parameters by sequentially refining previously estimated parameters. The refinement or update is performed using the following expression:

$$\mathbf{x}_{t+1} = \mathbf{C}(\mathbf{x}_t, \mathbf{F}_{t+1} \circ \mathbf{h}(\mathbf{x}_t)), \tag{2.3}$$

where $\mathbf{x}_t \in \mathbb{R}^p$ is the estimated parameters at time t, $\mathbf{h} : \mathbb{R}^p \to \mathbb{R}^f$ extracts some feature from the input data, and $\mathbf{F}_{t+1} : \mathbb{R}^p \to \mathbb{R}^d$ is a regressor that maps the feature $\mathbf{h}(\mathbf{x}_t)$ to an update vector, and $\mathbf{C} : \mathbb{R}^p \times \mathbb{R}^d \to \mathbb{R}^p$ composes $\mathbf{x}_t$ with the update vector to get $\mathbf{x}_{t+1}$. Given a training set $\{(\mathbf{x}_0^{(i)}, \mathbf{x}_*^{(i)}, \mathbf{h}^{(i)})\}_{i=1}^n$, where i denotes the i^{th} problem instance, $\mathbf{x}_0^{(i)}$ is the initial parameter estimates, $\mathbf{x}_*^{(i)}$ is the ground truth parameters, and $\mathbf{h}^{(i)}$ extracts some feature, *e.g.*, from image i, the training is performed sequentially from $t = 1, 2, \ldots,$ by solving

$$\mathbf{F}_{t+1} = \underset{\mathbf{F} \in \mathcal{F}}{\operatorname{argmin}} \sum_{i=1}^{N} L(\mathbf{x}_*^{(i)}, \mathbf{C}(\mathbf{x}_t^{(i)}, \mathbf{F} \circ \mathbf{h}^{(i)}(\mathbf{x}_t^{(i)}))) + R(\mathbf{F}), \tag{2.4}$$

where $L : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}$ is a loss function, $\mathcal{F}$ is a class of regressors (*e.g.*, the class of linear functions), and $R : (\mathbb{R}^p \to \mathbb{R}^d) \to \mathbb{R}$ is a regularizer. After a map is learned, all $\mathbf{x}_t^{(i)}$ is updated to $\mathbf{x}_{t+1}^{(i)}$ using (2.3) before learning the next regressor. Once the sequence of regressors $\{\mathbf{F}_t\}$ is obtained, it can be applied to solve a new problem instance. Next, we review the previous work in SSUs. A summary of the review is provided in Table. 2.1.

First, we review the works that use a single map to perform the update. Cootes *et al.* [28] introduced active appearance model (AAM) for facial and nonrigid object alignment in images. AAM uses orthogonal bases to model variations in shapes and appearances. In order to fit a template shape to an image, the difference between intensities of the template shape and the estimated shape from the image is mapped to an update vector. The map used in this paper was a single linear map, which is learned from a set of training data. The update is performed by alternating between computing an update vector and summing it to the current shape parameter vector until convergence[1]. Jurie and Dhome [56] proposed hyperplane approximation as an approximation to the Jacobian matrix. Similar to AAM, they use a single linear map to map from image difference to an update vector. Instead of working with image points, they perform update to rigid, homography, and other linear transformations' parameters for template tracking applications. In [31], Cristinacce and Cootes proposed to use GentleBoost with Haar wavelets to map image intensity to an update vector. Bayro-Corrochano and Ortegon-Aguilar [10, 11] proposed to update the parameters in the Lie algebra of the projective and affine tranformations. Around the same time, Tuzel *et al.* [97] proposed a similar approach which works with Lie algebra, but used Histogram of Oriented Gradients (HOG) feature [33] instead of intensity or intensity difference as in previous works.

More recently, many works proposed to learn the update using a sequence of maps instead of a single map. Saragih and Goecke [88] proposed Iterative Error Bound Minimisation (IEBM), where ν-support vector regression [25] is used to learn a sequence of linear maps to update transformation (warp) parameters for nonrigid image alignment. Dollár *et al.* [39] proposed cascade pose regression (CPR), where they train a sequence of random ferns for image-based object pose estimation. CPR can be used with different parametrization, *e.g.*, 2D image coodinates, scale, and orientation of ellipses. Instead of summing the parameters with the updates, CPR can be used with other inversible composition rules. The paper also shows that the training error decreases exponentially under weak learner assumptions. Cao *et al.* [22] proposed to learn a sequence of boosted regressors that minimizes error in parameter space for facial feature alignment. Sun *et al.* [92] trained a sequence of three convolutional neural networks to both extract the features

[1]Interestingly, active shape model [27] and a later AAM paper [29] does not use a learned linear map to compute the update vector, but rather use gradient methods.

Table 2.1: A summary of SSUs in the literature

Approach	Parameter $\mathbf{x}$	Feature $\mathbf{h}$	Composition $\mathbf{C}$	Map $\mathbf{F}$	#Maps	#Updates	Applications
AAM [28]	Multiple 2D image coor.	Intensity diff.	Summation	Linear map	1	Until conv.	Nonrigid pose estimation (face, knee)
[56]	Transformation matrices	Intensity diff.	Summation	Linear map	1	Until conv.	Template tracking
IEBM[88]	Transformation parameters	Intensity diff.	Summation	Linear map	Multiple	1 per map	Nonrigid pose estimation (face)
[31]	Multiple 2D image coor.	Haar wavelets	Summation	GentleBoost	1	Until conv.	Face alignment
[11]	Lie algebra	Intensity diff.	Summation	Linear map	1	Until conv.	Template tracking
[97]	Lie algebra	HOG	Summation	Linear map	1	Until conv.	Template tracking
CPR [39]	Pose parameters (*e.g.*2D image coor., scale, orientation)	Pose-indexed	Inversible composition	Random ferns	Multiple	1 per map	Nonrigid pose estimation (face, mice, fish)
[22]	Multiple 2D image coor.	Pose-indexed	Summation	Boosting	Multiple	1 per map	Face alignment
[92]	Multiple 2D image coor.	Intensity (deep feature)	Summation	Neural networks	Multiple	1 per map	Face alignment
SDM [104, 105]	Multiple 2D image coor./ Euler angles and translation vector	HOG / Reproj. points	Summation	Linear map	Multiple	1 per map	Face alignment / 3D pose estimation with known shapes
[95]	Multiple 2D image coor.	Intensity (deep feature)	Summation	Recurrent neural network	1	Multiple (fixed)	Face alignment
DO [99, 100]	Problem dependent	Provide a framework to derive $\mathbf{h}$	Summation	Linear map	Multiple	1 per map, then last map until conv.	3D registration, image denoising, camera pose estimation

and map them to an update vector. Xiong and De la Torre proposed Supervised Descent Method (SDM) [104, 105], where they learn a sequence of linear maps as the averaged Jacobian matrices for minimizing nonlinear least-squares functions in the feature space. They also provided conditions for the error to strictly decrease in each iteration. More recent works include learning both Jacobian and Hessian matrices [98]; running Gauss-Newton algorithm after SSU [2]; using different maps in different regions of the parameter space [106]; and using recurrent neural network as the sequence of maps while also learning the feature [95]. While the mentioned works learn the regressors in a sequential manner, Sequence of Learned Linear Predictor [113] first learns a set of linear maps then selects a subset to form a sequence of maps.

We observe that most SSUs focus on image-based tracking and pose estimation. This is because the feature for these problems is rather obvious: they use intensity-based features such as intensity difference, HOG, *etc*. Extending SSUs to other tasks require designing new features. In this work, we propose DO as an extension of previous SSUs by providing a procedure for deriving the feature functions $\mathbf{h}$ for different tasks. We also study the its properties, propose a general framework for designing features, and apply DO to other computer vision problems, such as point cloud registration, camera pose estimation, and image denoising.

2.3.2 Gradient boosting

The concept of SSUs is similar to that of gradient boosting (GB), which was proposed by Friedman in [43] for classification and regression tasks. GB uses weak learners to iteratively update the estimated parameter. Mathematically, given a feature $\mathbf{h} \in \mathbb{R}^d$, GB infers the estimated parameters $\hat{x} \in \mathbb{R}$ as

$$\hat{x} = \sum_{t=1}^{T} w_t F_t(\mathbf{h}), \tag{2.5}$$

where each $F_t : \mathbb{R}^d \to \mathbb{R}, t = 1, \ldots, T$, is a weak regressor, and w_t is a scalar weight. The functions F_t and the weights w_t are learned from training data $\{(\mathbf{h}_i, x_i)\}_{i=1}^n$ in a sequential fashion:

$$F_t = \underset{w \in \mathbb{R}, F \in \mathcal{F}}{\operatorname{argmin}} \sum_{i=1}^{n} L\left(x_i, wF(\mathbf{h}_i) + \sum_{l=1}^{t-1} w_l F_l(\mathbf{h}_i)\right), t = 1, \ldots, T, \tag{2.6}$$

where L denotes a loss function, and $\mathcal{F}$ denotes the space of the weak regressors, *e.g.*, short decision trees. The sequential training of GB has been interpreted as performing gradient descent for minimizing L in the function space $\mathcal{F}$ [68].

Originally, GB has been proposed to solve for scalar outputs, which may not be suitable for computer vision problems where the parameters may have underlying structures. To handle this

issue, several works have extended GB to solve structured prediction problems [37, 38].

The main difference between GB and SSUs (including DO) is that the feature vector $\mathbf{h}$ of GB is fixed, while for SSUs the feature vector $\mathbf{h}$ is a function that is dependent on the estimated parameter $\mathbf{x}_t$. This difference allows the inference of GB (*i.e.*, the summation of the terms in (2.5)) to be performed in any order, while the inference of SSUs in (2.3) can only be done in the same order that their regressors are trained. On the other hand, GB can only use a fixed amount of information from $\mathbf{h}$ while SSUs (and DO) allow the information in $\mathbf{h}$ to be updated as the parameters $\mathbf{x}_t$ are estimated. In this thesis, we will use this update of the feature function $\mathbf{h}$ as an advantage. Specifically, in the next chapter, we will derive a feature function $\mathbf{h}$ that *(i)* allows us to interpret DO as learning to imitate gradient methods for solving optimization without an explicit cost function, and *(ii)* can be applied to multiple tasks.

Chapter 3

Discriminative Optimization

Mathematical optimization involves solving problems of the form

$$\begin{aligned} &\underset{\mathbf{x}}{\text{minimize}} \;\; \Phi(\mathbf{x}) \\ &\text{subject to} \;\; \mathbf{x} \in \mathcal{S}, \end{aligned} \tag{3.1}$$

where $\mathbf{x} \in \mathbb{R}^d$ is a vector of variables, $\Phi : \mathbb{R}^d \to \mathbb{R}$ is an objective function, and $\mathcal{S} \subset \mathbb{R}^d$ is a constraint set. The goal of (3.1) is to find one or multiple globally optimal solutions $\mathbf{x}_* \in \mathcal{S}$ that minimize the cost function, that is $\Phi(\mathbf{x}_*) \leq \Phi(\mathbf{x})$ for all $\mathbf{x} \in \mathcal{S}$. However, solving optimization problems with arbitrary Φ and $\mathcal{S}$ is considered computationally infeasible [74]. Instead of aiming for an efficient universal algorithm, many approaches are developed for specific classes of optimization problems with specific structure. For example, if Φ is differentiable and $\mathcal{S} = \mathbb{R}^d$, then (3.1) becomes an unconstrained optimization problem with a differentiable cost function, which can be solved using gradient descent algorithms[1].

In this chapter, we describe Discriminative Optimization (DO), a learning-based approach for solving tasks which are conventionally formulated as unconstrained differentiable optimization problems. While traditional optimization has been successfully used for many tasks, a major challenge of using it is to design a suitable cost function: If the cost mismatches with the task at hand, its optimum may not correspond to the expected solution. On the other hand, DO uses training data to learn a search algorithm that leads to the desired solution. This allows DO to solve similar tasks without having to rely on any explicit cost function. Using synthetic data, we verify that DO can learn to find an approximate optimum of a cost function without having to infer its explicit form while solving incorrect cost functions results in worse solutions.

[1]Note that, without additional structures on Φ, such as convexity [17], gradient descent may get caught in a local minimum instead of converging to a global minimum.

We begin this chapter with Section 3.1, which provides a general description of our algorithm. Then, in Section 3.2, we describe a framework for deriving task-specific feature functions to use with the algorithm, which provides a concrete relation between our algorithm and gradient-based methods. We also provide a numerical example in each section. We consider the theoretical justification for our approach in Chapter 4, and apply the algorithm to applications in Chapter 5.

3.1 Learning to search for stationary points

3.1.1 Motivation from fixed point iteration

DO aims to learn a sequence of update maps (SUM) to update an initial parameter vector to a stationary point. The idea of DO is based on the fixed point iteration of the form

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \Delta\mathbf{x}_t, \tag{3.2}$$

where $\mathbf{x}_t \in \mathbb{R}^p$ is the parameter at step t, and $\Delta\mathbf{x}_t \in \mathbb{R}^p$ is the update vector. Eq. (3.2) is iterated until $\Delta\mathbf{x}_t$ vanishes, *i.e.*, until a stationary point is reached. An example of fixed point iteration for solving optimization is the gradient descent algorithm [17] (see Figure 3.1b). Let $J : \mathbb{R}^p \to \mathbb{R}$ be a differentiable cost function. The gradient descent algorithm for minimizing J is expressed as

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \mu_t \frac{\partial}{\partial \mathbf{x}} J(\mathbf{x})\bigg|_{\mathbf{x}=\mathbf{x}_t}, \tag{3.3}$$

where μ_t is a step size. One can see that the scaled gradient is used as $\Delta\mathbf{x}_t$ in (3.2), and it is known that the gradient vanishes when a stationary point is reached.

In contrast to gradient descent where the updates are derived from a cost function, DO learns the updates from the training data. The major advantages are that no cost function is explicitly designed, and the neighborhoods around the solutions of the training data are taken into account when the maps are learned, which results in more robustness to different types of perturbations present in the data.

3.1.2 DO search algorithm

DO uses an update rule in the form of (3.2). The update vector $\Delta\mathbf{x}_t$ is computed by mapping the output of a function $\mathbf{h} : \mathbb{R}^p \to \mathbb{R}^f$ with a sequence of matrices[2] $\mathbf{D}_t \in \mathbb{R}^{p\times f}$ (see Figure 3.1c).

[2]Here, we used linear maps due to their simplicity and computational efficiency. However, other non-linear regression functions can be used in a straightforward manner.

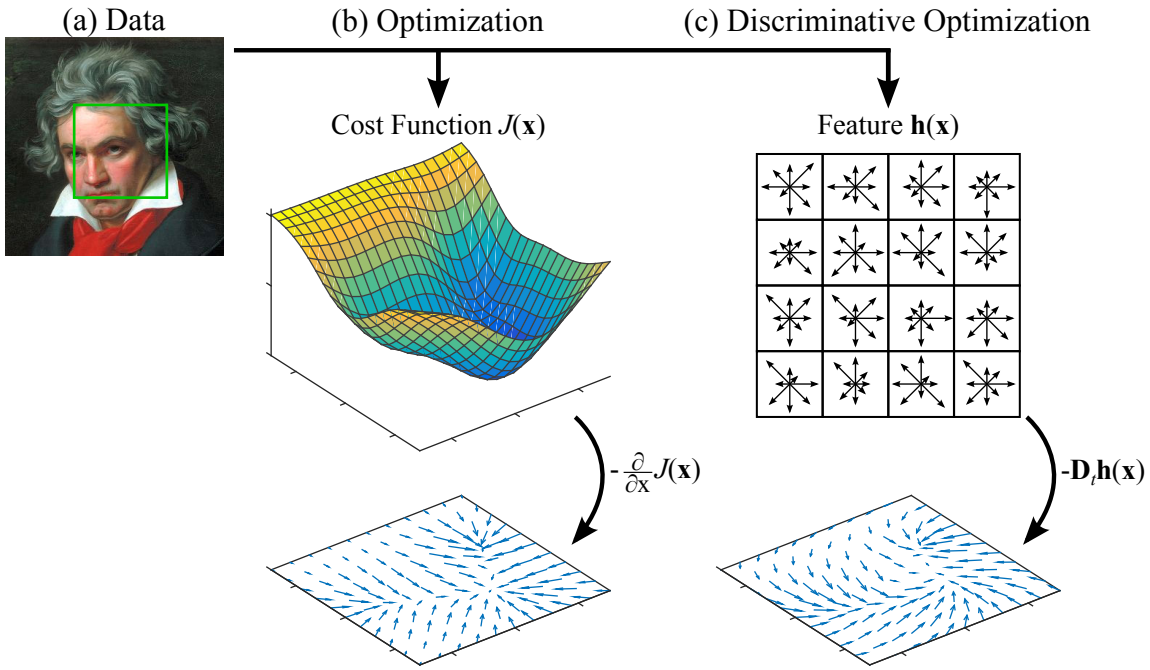


Figure 3.1: Comparison between optimization and discriminative optimization (DO). Given (a) input data, (b) conventional optimization first requires a cost function to be defined, then searches for the minimum by performing updates, *e.g.*, using negative gradients. (c) DO instead extracts a feature $\mathbf{h}(\mathbf{x})$ from the data, then maps the feature to an update vector using a sequence of update maps (SUM) $\{\mathbf{D}_t\}_t$. This SUM is learned from a set of training data.

Here, $\mathbf{h}$ is a function that encodes a representation of the data (*e.g.*, $\mathbf{h}(\mathbf{x})$ extracts features from an image at the position $\mathbf{x}$). Let $\mathbf{x}_0 \in \mathbb{R}^p$ be an initial parameter, DO iteratively updates $\mathbf{x}_t, t = 0, 1, \ldots,$ using:

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \mathbf{D}_{t+1}\mathbf{h}(\mathbf{x}_t), \tag{3.4}$$

until convergence to a stationary point. The sequence of matrices $\mathbf{D}_{t+1}, t = 0, 1, \ldots,$ learned from training data forms a sequence of update maps (SUM).

Learning a SUM

Suppose we are given a training set as a set of triplets $\{(\mathbf{x}_0^{(i)}, \mathbf{x}_*^{(i)}, \mathbf{h}^{(i)})\}_{i=1}^N$, where $\mathbf{x}_0^{(i)} \in \mathbb{R}^p$ is the initial parameter for the i^{th} problem instance (*e.g.*, the i^{th} image), $\mathbf{x}_*^{(i)} \in \mathbb{R}^p$ is the ground truth parameter (*e.g.*, position of the object on the image), and $\mathbf{h}^{(i)} : \mathbb{R}^p \to \mathbb{R}^f$ extracts a feature vector from the i^{th} problem instance. The goal of DO is to learn a SUM $\{\mathbf{D}_t\}_t$ that updates $\mathbf{x}_0^{(i)}$

Algorithm 1 Training a sequence of update maps (SUM)

Require: $\{(\mathbf{x}_0^{(i)}, \mathbf{x}_*^{(i)}, \mathbf{h}^{(i)})\}_{i=1}^N, T, \lambda$
Ensure: $\{\mathbf{D}_t\}_{t=1}^T$
1: **for** $t = 0$ **to** $T-1$ **do**
2: Compute $\mathbf{D}_{t+1}$ with (3.6).
3: **for** $i = 1$ **to** N **do**
4: Update $\mathbf{x}_{t+1}^{(i)} := \mathbf{x}_t^{(i)} - \mathbf{D}_{t+1}\mathbf{h}^{(i)}(\mathbf{x}_t^{(i)})$.
5: **end for**
6: **end for**

to $\mathbf{x}_*^{(i)}$. To learn a map, we use the least squares regression:

$$\mathbf{D}_{t+1} = \arg\min_{\tilde{\mathbf{D}}} \frac{1}{N} \sum_{i=1}^{N} \|\mathbf{x}_*^{(i)} - \mathbf{x}_t^{(i)} + \tilde{\mathbf{D}}\mathbf{h}^{(i)}(\mathbf{x}_t^{(i)})\|_2^2, \tag{3.5}$$

where $\|\cdot\|_2$ is the ℓ_2 norm. After the map $\mathbf{D}_{t+1}$ is learned, we update each $\mathbf{x}_t^{(i)}$ using (3.4), then proceed to learn the next map. This process is repeated until some terminating conditions, such as until the error does not decrease much or a maximum number of iterations is reached. To see why (3.5) learns stationary points, we can see that for i with $\mathbf{x}_t^{(i)} \approx \mathbf{x}_*^{(i)}$, (3.5) will force $\mathbf{D}_{t+1}\mathbf{h}^{(i)}(\mathbf{x}_t^{(i)})$ to be close to zero, thereby inducing a stationary point around $\mathbf{x}_*^{(i)}$. In practice, we use ridge regression to learn the maps to prevent overfitting:

$$\min_{\tilde{\mathbf{D}}} \frac{1}{N} \sum_{i=1}^{N} \|\mathbf{x}_*^{(i)} - \mathbf{x}_t^{(i)} + \tilde{\mathbf{D}}\mathbf{h}^{(i)}(\mathbf{x}_t^{(i)})\|_2^2 + \lambda\|\tilde{\mathbf{D}}\|_F^2, \tag{3.6}$$

where $\|\cdot\|_F$ is the Frobenius norm, and λ is a hyperparameter. The pseudocode for training a SUM is shown in Alg. 1.

Solving a new problem instance

To solve a new problem instance with an unseen function $\mathbf{h}$ and an initialization $\mathbf{x}_0$, we update $\mathbf{x}_t, t = 0, 1, \ldots,$ with the obtained SUM using (3.4) until a stationary point is reached. However, in practice, the number of maps is finite, say T maps. We observed in many cases that the update at the T^{th} iteration is still large, which means the stationary point has not been reached, and that $\mathbf{x}_T$ is far from the true solution. For example, in the point cloud registration task, the rotation between the initial orientation and the solution might be so large that we cannot obtain the solution within a fixed number of update iterations. To overcome this problem, we keep updating $\mathbf{x}$ using the T^{th} map until the update is small or the maximum number of iterations

Algorithm 2 Searching for a stationary point

Require: $\mathbf{x}_0, \mathbf{h}, \{\mathbf{D}_t\}_{t=1}^{T}, maxIter, \epsilon$
Ensure: $\mathbf{x}$
1: Set $\mathbf{x} := \mathbf{x}_0$
2: **for** $t = 1$ **to** T **do**
3: Update $\mathbf{x} := \mathbf{x} - \mathbf{D}_t\mathbf{h}(\mathbf{x})$
4: **end for**
5: Set $iter := T + 1$.
6: **while** $\|\mathbf{D}_T\mathbf{h}(\mathbf{x})\| \geq \epsilon$ **and** $iter \leq maxIter$ **do**
7: Update $\mathbf{x} := \mathbf{x} - \mathbf{D}_T\mathbf{h}(\mathbf{x})$
8: Update $iter := iter + 1$
9: **end while**

is reached. This approach makes DO different from previous work in Section 2.3, where the updates are only performed up to the number of maps. Alg. 2 shows the pseudocode for updating the parameters.

Convergence of DO

We can see that DO operates similarly to many iterative algorithms. Instead of computing the update from a manually defined function, each step of DO is computed based on the training data. This raises the question whether the parameter $\mathbf{x}_t$ would converge to $\mathbf{x}_*$, both in training and test phases. Here, we provide a short summary on this issue, while the complete detail is discussed in Chapter 4. For the convergence in training, we show that the mean squared ℓ_2 distance between $\mathbf{x}_t^{(i)}$ and $\mathbf{x}_*^{(i)}$ would strictly decrease in each iteration or even converge to zero under some weak conditions. These results provide a theoretical foundation for DO as a learning-based algorithm for solving estimation problems. On the other hand, the convergence results for the test phase are not considered in this thesis.

3.1.3 Numerical example: Finding the zero of 1D functions

In this section, we demonstrate properties of DO on the problem of finding the zero of 1D functions. Consider the following function (see Figure 3.2a):

$$h^{(i)}(x) = \frac{1}{1+u_1^{(i)}}\sin(x - x_*^{(i)}) + \frac{1}{1+9u_2^{(i)}}\text{sgn}(x - x_*^{(i)}) + \eta^{(i)}(x), x \in \mathcal{X} = [-1, 1] \quad (3.7)$$

where $\text{sgn}(x) : \mathbb{R} \to \{-1, 0, 1\}$ is the Heaviside step function, $u_1^{(i)}, u_2^{(i)} \in [0, 1]$ scale $\sin$ and sgn, and $\eta^{(i)}(x)$ is a function representing noise. Our goal is to find an $x \in \mathcal{X}$ such that $h^{(i)}(x)$

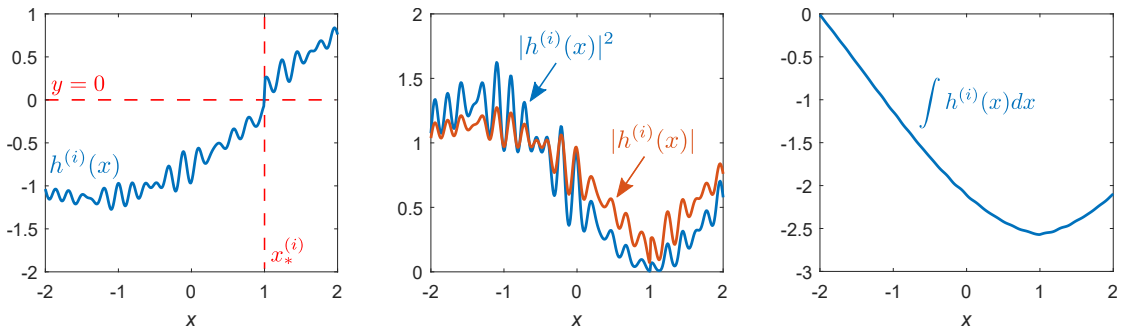


Figure 3.2: Example of 1D synthetic data. (a) A function $h^{(i)}(x)$. The goal of the problem is to find x such that $h^{(i)}(x) \approx 0$. Here, the solution is $x_*^{(i)} = 1$. (b) The cost surface if we formulate the problem as the minimization of $|h^{(i)}(x)|^2$ or $|h^{(i)}(x)|$, which results in numerous local minima. (c) On the other hand, if we integrate $h^{(i)}(x)$, we can see a function where its minimum approximates the correct solution.

is close to zero. In this case, since $\eta^{(i)}(x)$ has small amplitude, we have that $x_*^{(i)}$ is our solution: $h^{(i)}(x_*^{(i)}) \approx 0$.

There are two conventional approaches to this type of problems. First, we can use Newton's method to search for the zero. However, we can see that $h(x)$ is neither monotone nor continuous. This prevents us from directly applying Newton's method. Alternatively, we could formulate an optimization where we would minimize $|h(x)|^2$ or $|h(x)|$. However, as seen in Figure 3.2b, the cost functions of this optimization approach have a large number of local minima, making it challenging to solve for a good solution.

To solve this problem, we observe that $h^{(i)}(\hat{x}) > h^{(i)}(x_*^{(i)})$ for all $\hat{x} > x_*^{(i)}$, and $h^{(i)}(\hat{x}) < h^{(i)}(x_*^{(i)})$ for all $\hat{x} < x_*^{(i)}$. This property allows us to learn the sequence of update maps $D_t \in \mathbb{R}, t = 1, \ldots,$ to solve this problem. We will explore this property in detail in Chapter 4. For the case of 1D functions, this property allows us to think of $h^{(i)}(x)$ as the gradient of some function $\bar{h}^{(i)}(x) = \int h^{(i)}(x)dx$, where the global minimum of $\bar{h}^{(i)}(x)$ corresponds to the correct solution (see Figure 3.2c). With this interpretation, the update $\Delta x = D_t h^{(i)}(x)$ can be thought of as a gradient step of gradient descent, where the learned map D_t represents the step size at time t.

As a numerical example, we generated the training data $\{(x_0^{(i)}, x_*^{(i)}, h^{(i)})\}_{i=1}^N$ where $N = 5 \times 10^3$ and $x_0^{(i)} = 0$ for all i. Similarly, we also generated 5×10^2 test data instances. We learned a total of 100 maps to observe the training error, which is shown in Figure 3.3a. Since the training method in Alg. 1 minimizes average error, we can see that the average error decreases monotonically, while the error of the individual training instances may oscillate. This means that, on average, $x_t^{(i)}$ is moving closer to $x_*^{(i)}$, which is the result we expect. In Figure 3.3b, we can see that the norm of D_t is decreasing in each iteration, which matches the decreasing step size

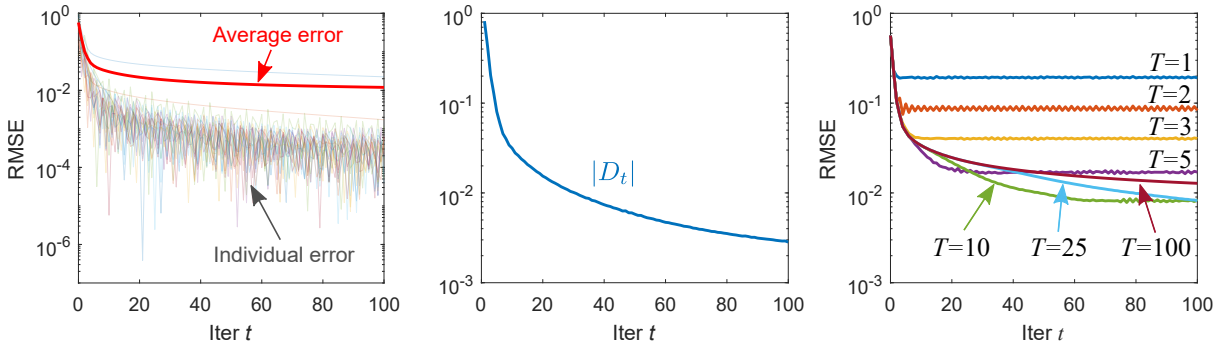


Figure 3.3: (a) The training error of DO. (b) The norm of the learned map in each iteration, which in this case also corresponds to the value of the map, *i.e.*, $D_t = |D_t|$. (c) The test error with different last maps T.

used in gradient methods [74].

For test, we use Alg. 2 while varying the last map T. Recall that after reaching iteration T, we keep iterating using the last map D_T until termination. Here, we terminate after reaching 100 iterations. We generated 10^2 test samples similar to the training data. Figure 3.3c shows the test error. We can make the following three observations. *(i)* We can see the errors keep decreasing when we keep iterating with D_T beyond iteration T. This is because for some problem instance i, $x_T^{(i)}$ may still be far from the solution, and by keeping on iterating, the estimate could move closer to the solution. This validates the additional iterations beyond the last map T. *(ii)* If a small T is used, the estimate may not converge to the true solution, resulting in high errors than using a larger T. This results is similar to the case of the subgradient method with fixed step size, as the estimate oscillates around the minimum but does not converge to it [18]. *(iii)* On the other hand, we can see that using a large T could lead to a slower convergence, *e.g.*, the test error of $T = 10$ decreases much faster than that of $T = 100$. This is because the D_{100} has a smaller norm than D_{10}, which causes the estimate to move towards to solution at a slower pace. From these observations, we can see that the extra steps provide an important improvement over running until only step T. However, selecting which map to use as the last map is also important, as this could control the speed of convergence of the algorithm.

3.1.4 Section summary

We have described Discriminative Optimization (DO), a supervised framework which solves problems by learning a sequence of maps $\mathbf{D}_t, t = 1, \ldots, T$, that maps feature functions $\mathbf{h}$ to update steps $\Delta\mathbf{x}$ in search space. The maps are trained using ridge regression, which can be

computed efficiently. In the test phase, DO only needs to apply the learned maps to the updated feature iteratively, which requires very low computation cost. We provide a numerical example which uses DO to search for the zero of a non-convex and non-monotonic function, and show that, on average, the estimates of DO move closer to the expected solutions in both the training and test phases.

One major question remains: How can we apply DO to practical applications? To accomplish this goal, we need to incorporate the information about the task of interest into the algorithm, which can be done by designing task-specific feature function $\mathbf{h}$. However, it is not trivial to design a new $\mathbf{h}$ for every new task we wish to solve. To handle this challenge, in the following section, we describe a general framework for deriving task-specific $\mathbf{h}$.

3.2 Deriving feature function $\mathbf{h}$

The function $\mathbf{h}$ provides information about the task of interest. For example, if we wish to solve camera pose estimation, $\mathbf{h}$ should convey the information about the matches between the image points and the 3D points, and also about the types of transformation used (*e.g.*, projective, affine, *etc*.). Since different tasks entail different types of information, we need to design new $\mathbf{h}$ for each task, and this is not straightforward. Previous works which focuses on image-based tracking and alignment, *e.g.*, [28, 39, 97, 104, 105], use HOG [33], SIFT [64], or other intensity-based features as $\mathbf{h}$. However, it is not clear how to design $\mathbf{h}$ for tasks which are not image-based, such as the feature-matching-based pose estimation task.

In this section, we describe a framework to derive a task-specific $\mathbf{h}$ based on the gradient descent algorithm without assuming an explicit cost function. This is motivated by the observation that many computer vision tasks aim to find $\mathbf{x}$ such that $\mathbf{g}_j(\mathbf{x}) = \mathbf{0}_d, j = 1, \ldots, J$, where $\mathbf{g}_j : \mathbb{R}^p \rightarrow \mathbb{R}^d$ models the residual of the task of interest. Examples of $\mathbf{g}_j(\mathbf{x})$ include the reprojection error of the j^{th} feature match for homography or camera pose estimation [53], and the intensity residual of pixel j for tracking applications [5]. We will refer to $\mathbf{g}_j(\mathbf{x})$ as the *residual function*. To solve such problem, one typically formulates an optimization problem that minimizes the sum of the residuals:

$$\underset{\mathbf{x}}{\text{minimize}}\, \Phi(\mathbf{x}) = \frac{1}{J} \sum_{j=1}^{J} \varphi(\mathbf{g}_j(\mathbf{x})), \tag{3.8}$$

where $\varphi : \mathbb{R}^d \rightarrow \mathbb{R}$ is a penalty function, *e.g.*, sum of squares, ℓ_1 norm, *etc*. If Φ is differentiable, then we can use gradient descent to find a minimum and return it as the solution. However, this approach has two disadvantages. *(i)* As shown in Section 3.1.3, Φ could have multiple

local minima, and it may be challenging to solve for a good one. *(ii)* The choice of φ has a strong impact on the solution in terms of robustness to different perturbations, and it is not straightforward to select φ that will account for perturbations of the real data. The following framework will derive an $\mathbf{h}$ that, when used with the training method in Alg. 1, can be interpreted as learning to imitate gradient descent on Φ with an unknown φ. Two main advantages are that we can bypass the manual selection of φ and also avoid obtaining undesirable solutions.

To provide a better intuition, we first show how to derive $\mathbf{h}$ for the '*Can you guess the number?*' game from Section 1.1. Then, we derive $\mathbf{h}$ for general residual functions. Finally, we provide a numerical example from *Can you guess the number?* game to verify our approach.

3.2.1 Example: Deriving $\mathbf{h}$ for *Can you guess the number?*

Recall that in the '*Can you guess the number?*' game in Section 1.1, we wish to solve the optimization problem of the following form

$$x_* = \arg\min_x \Phi(x) = \frac{1}{J} \arg\min_x \sum_{j=1}^{J} \varphi(x - x_j), \tag{3.9}$$

where $\varphi : \mathbb{R} \to \mathbb{R}$ is some penalty function. If φ is a known differentiable function, then we could solve (3.9) by gradient descent with the update[3]

$$\Delta\hat{x} = \frac{1}{J} \frac{\partial}{\partial x} \Phi(x) \bigg|_{x=\hat{x}} = \frac{1}{J} \sum_{j=1}^{J} \frac{\partial}{\partial x} \varphi(x - x_j) \bigg|_{x=\hat{x}}, \tag{3.10}$$

where $\hat{x}$ is the current estimate. Here, we will show how to derive the function $\mathbf{h}$ to solve (3.9) when φ is unknown.

Given n pairs of data instance and its minimizers $\{(\mathcal{X}^{(i)}, x_*^{(i)})\}_{i=1}^n$, where $\mathcal{X}^{(i)} = \{x_1^{(i)}, \ldots, x_{J_i}^{(i)}\}$, our approach solves for $\tilde{x}_*$ for an unseen $\tilde{\mathcal{X}}$ using an update rule that imitates the gradients of the unknown function. We denote this imitation of the gradient as a function $\phi : \mathbb{R} \to \mathbb{R}$. By replacing the gradient $\frac{\partial}{\partial x}\varphi$ with ϕ, the update is then modified to

$$\Delta\hat{x} = \frac{1}{J} \sum_{j=1}^{J} \phi(\hat{x} - x_j). \tag{3.11}$$

At this point, we are able to compute $\hat{x} - x_j$ but we do not know what ϕ is. Thus, we approximate (3.11) as the product $\mathbf{D}\mathbf{h}(\hat{x})$, where we will express ϕ as a part of $\mathbf{D}$ and learn it from

[3] We leave out the step size.

the training data with Alg. 1, while $\hat{x} - x_j$ will be expressed in $\mathbf{h}(\hat{x})$ which can be computed independently from ϕ. To do so, we need to decouple ϕ and $\hat{x} - x_j$ into a matrix-vector product. This is done by noting that $\phi(\hat{x} - x_j)$ can be written as the convolution between ϕ and Dirac delta function δ:

$$\Delta\hat{x} = \frac{1}{J}\sum_{j=1}^{J}\phi(\hat{x} - x_j) = \frac{1}{J}\sum_{j=1}^{J}\int \phi(v)\delta(v - \hat{x} + x_j)dv \tag{3.12}$$

$$= \frac{1}{J}\int_V \phi(v)\sum_{j=1}^{J}\delta(v - \hat{x} + x_j)dv, \tag{3.13}$$

where $V = \mathbb{R}$. Note that $\phi(v)$ and $\delta(v - \hat{x} + x_j)$ can be thought of as vectors indexed by v, and the integration as the inner product between them. Recall that the update of DO, $\Delta\mathbf{x} = \mathbf{D}\mathbf{h}(\mathbf{x})$, is also an inner product between each row of $\mathbf{D}$ and $\mathbf{h}(\mathbf{x})$, where $\mathbf{D}$ is learned from training data. This allows us to assign ϕ and δ into $\mathbf{D}$ and $\mathbf{h}$ as[4]

$$\mathbf{D}(v) = \phi(v), \text{and} \tag{3.14}$$

$$\mathbf{h}(v; \hat{x}) = \frac{1}{J}\sum_{j=1}^{J}\delta(v - \hat{x} + x_j). \tag{3.15}$$

Now, we can see that $\mathbf{h}$ depends only on the data x_j and the estimate of the solution $\hat{x}$, while the function ϕ is expressed entirely by $\mathbf{D}$. This allows us to use Alg. 1 to learn the approximation of ϕ as $\mathbf{D}$.

Discretization: To compute Δx in (3.13) in practice, we approximate it as an inner product between two finite-dimensional vectors. The main idea of this approximation is shown in Figure 3.4. Specifically, we discretize the functions into vectors: $\phi(v)$ is discretized into a vector $\boldsymbol{\phi} \in \mathbb{R}^r$ and $\delta(v - \hat{x} + x_j)$ is discretized into a standard basis vector[5] $\mathbf{e}_{\gamma_{q,r}(\hat{x}-x_j)}$. The function $\gamma_{q,r} : \mathbb{R} \to \{0, 1, \ldots, r\}$ is a discretization function that returns the index of the box that its argument discretizes to. It is defined as

$$\gamma_{q,r}(z) = \begin{cases} \left\lceil \frac{r}{2}\left(\frac{z}{q} + 1\right)\right\rceil & z \in [-q, q], \\ 0 & \text{otherwise}, \end{cases} \tag{3.16}$$

where $\lceil\cdot\rceil$ rounds up any number, q denotes the discretization range, and r denotes the number

[4]Here, $\mathbf{D}$ is a scalar function. Notation for matrix (bold capital letter) is used here to remind the readers of the update rule (3.4).

[5]Note that a discrete delta function is similar to a standard basis vector.

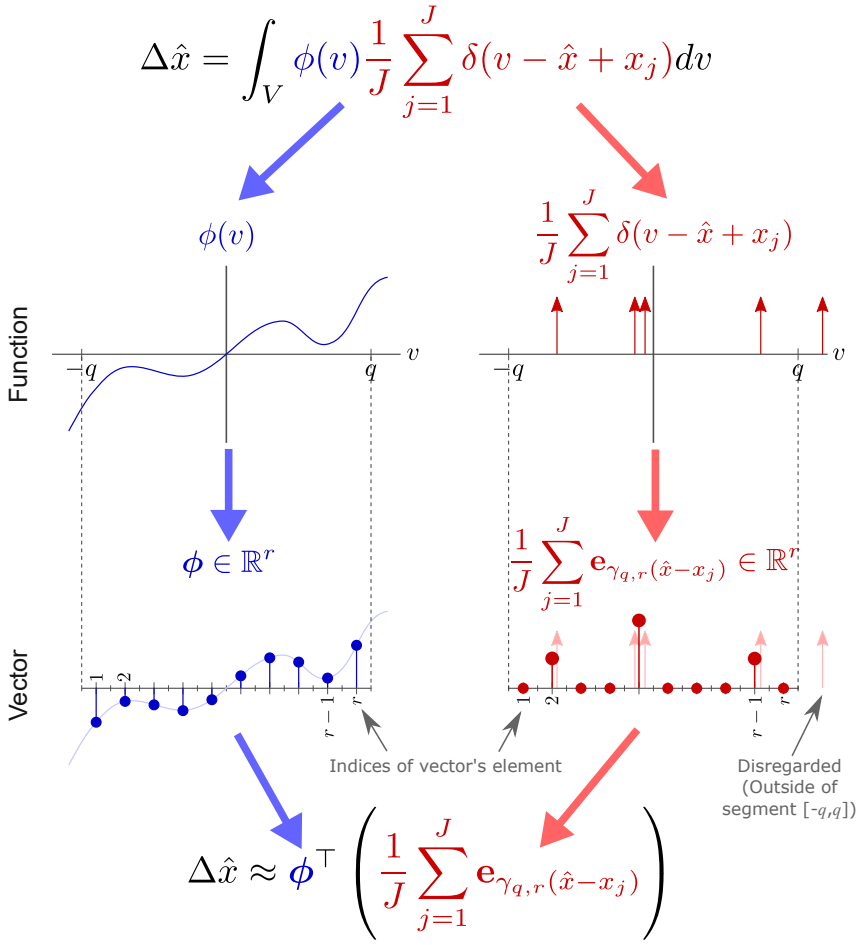


Figure 3.4: Approximating the inner product of functions as the inner product of vectors by discretization. The functions $\phi(v)$ and $\frac{1}{J}\sum_j \delta(v - \hat{x} + x_j)$ are discretized into the vectors $\boldsymbol{\phi}$ and $\frac{1}{J}\sum_j \mathbf{e}_{\gamma_{q,r}(\hat{x}-x_j)}$ in $\mathbb{R}^r$, *resp*. The function $\gamma_{q,r}$ returns the index of the box (*i.e.*, vector element) that each $\delta(v - \hat{x} + x_j)$ is assigned to. Note that only the values between $[-q, q]$ are discretized, and any values outside the segment are disregarded. See text for details.

of boxes. Note that here we define $\mathbf{e}_0 = \mathbf{0}_r$.[6] With the definition of $\gamma_{q,r}$, we can express the approximation of (3.13) as:

$$\Delta\hat{x} = \frac{1}{J} \int_V \phi(v) \sum_{j=1}^{J} \delta(v - \hat{x} + x_j) dv \approx \boldsymbol{\phi}^\top \left(\frac{1}{J} \sum_{j=1}^{J} \mathbf{e}_{\gamma_{q,r}(\hat{x}-x_j)} \right). \tag{3.17}$$

[6]This notation is useful for disregarding large residuals since any $\hat{x}-x_j$ with $|\hat{x}-x_j| > q$ will have $\mathbf{e}_{\gamma_{q,r}(\hat{x}-x_j)} = \mathbf{0}_r$. This means $\hat{x} - x_j$ will not contribute to the feature $\mathbf{h}$ in (3.19).

Similar to (3.14) and (3.15), we can set

$$\mathbf{D} = \boldsymbol{\phi}, \text{and} \tag{3.18}$$

$$\mathbf{h}(\hat{x}) = \frac{1}{J}\sum_{j=1}^{J} \mathbf{e}_{\gamma_{q,r}(\hat{x}-x_j)}. \tag{3.19}$$

In short, the feature function $\mathbf{h}$ is a histogram of indicators of the residuals between the desired output $\hat{x}$ and each of the elements x_j in the input set. On the other hand, $\mathbf{D}$ represents a discretized version of the gradient of the penalty function ϕ.

Given the expression of $\mathbf{h}$ above, we can train a SUM $\{\mathbf{D}_t\}_{t=1}^{T}$ using the procedure in Alg. 1 to solve the '*Can you guess the number?*' game. Before we do that in Section 3.2.3, we derive the $\mathbf{h}$ function for general residual functions.

3.2.2 Deriving $\mathbf{h}$ for general residual functions

In this section, we derive the $\mathbf{h}$ function for the general residual function in (3.8), reproduced here for convenience:

$$\underset{\mathbf{x}}{\text{minimize}}\, \Phi(\mathbf{x}) = \frac{1}{J}\sum_{j=1}^{J} \varphi(\mathbf{g}_j(\mathbf{x})), \tag{3.20}$$

For simplicity, we assume $\Phi(\mathbf{x})$ is differentiable, but the following derivation also applies when it is subdifferentiable. First, let us observe its derivative:

$$\frac{\partial}{\partial \mathbf{x}}\Phi(\mathbf{x}) = \frac{1}{J}\frac{\partial}{\partial \mathbf{x}}\sum_{j=1}^{J} \varphi(\mathbf{g}_j) = \frac{1}{J}\sum_{j=1}^{J} \left[\frac{\partial \mathbf{g}_j}{\partial \mathbf{x}}\right]^{\top} \frac{\partial \varphi(\mathbf{g}_j)}{\partial \mathbf{g}_j}, \tag{3.21}$$

where we express $\mathbf{g}_j(\mathbf{x})$ as $\mathbf{g}_j$ to reduce notation clutter. We can see that the form of φ affects only the last term in the RHS of (3.21), while the Jacobian $\frac{\partial \mathbf{g}_j}{\partial \mathbf{x}}$ does not depend on it. Here, we will derive $\mathbf{h}$ to learn this last term from a set of training data with DO.

Similar to Section 3.2.1, we will express (3.21) as the product $\mathbf{Dh}$. First, we rewrite (3.21) as an update vector $\Delta\mathbf{x}$, where we replace the derivative of φ with a function $\phi : \mathbf{R}^d \to \mathbf{R}^d$

$$\Delta\mathbf{x} = \frac{1}{J}\sum_{j=1}^{J} \left[\frac{\partial \mathbf{g}_j}{\partial \mathbf{x}}\right]^{\top} \phi(\mathbf{g}_j) \tag{3.22}$$

$$= \frac{1}{J}\sum_{j=1}^{J}\sum_{k=1}^{d} \left[\frac{\partial \mathbf{g}_j}{\partial \mathbf{x}}\right]_{k,:}^{\top} [\phi(\mathbf{g}_j)]_k, \tag{3.23}$$

where $[\mathbf{Y}]_{k,:}$ is row k of $\mathbf{Y}$, and $[\mathbf{y}]_k$ is element k of $\mathbf{y}$. Next, we express (3.23) as the convolution between ϕ and Dirac delta function δ:

$$\Delta\mathbf{x} = \frac{1}{J}\sum_{j=1}^{J}\sum_{k=1}^{d}\left[\frac{\partial \mathbf{g}_j}{\partial \mathbf{x}}\right]_{k,:}^{\top}\int_V [\phi(\mathbf{v})]_k \delta(\mathbf{v}-\mathbf{g}_j)d\mathbf{v}, \tag{3.24}$$

where $V = \mathbb{R}^d$. It can be seen that (3.24) is equivalent to (3.21), while being linear in ϕ. This allows us to express (3.24) as the product $\mathbf{Dh}$, where ϕ is included into $\mathbf{D}$ which will be learned using linear least squares regression, while the remaining terms will be incorporated into the function $\mathbf{h}$. For simplicity, we only look at the element l of $\Delta\mathbf{x}$:

$$[\Delta\mathbf{x}]_l = \frac{1}{J}\sum_{j=1}^{J}\sum_{k=1}^{d}\left[\frac{\partial \mathbf{g}_j}{\partial \mathbf{x}}\right]_{k,l}\int_V [\phi(\mathbf{v})]_k \delta(\mathbf{v}-\mathbf{g}_j)d\mathbf{v}, \tag{3.25}$$

$$= \sum_{k=1}^{d}\int_V [\phi(\mathbf{v})]_k \left(\frac{1}{J}\sum_{j=1}^{J}\left[\frac{\partial \mathbf{g}_j}{\partial \mathbf{x}}\right]_{k,l}\delta(\mathbf{v}-\mathbf{g}_j)\right)d\mathbf{v}, \tag{3.26}$$

$$= \sum_{k=1}^{d}\int_V \mathbf{D}(\mathbf{v},k)\mathbf{h}(\mathbf{v},k,l;\mathbf{x})d\mathbf{v}. \tag{3.27}$$

Eq. 3.27 expresses $[\Delta\mathbf{x}]_l$ as an inner product between $\mathbf{D}$ and $\mathbf{h}$ over $\mathbf{v}$ and k, where

$$\mathbf{D}(\mathbf{v},k) = [\phi(\mathbf{v})]_k, \tag{3.28}$$

$$\mathbf{h}(\mathbf{v},k,l;\mathbf{x}) = \frac{1}{J}\sum_{j=1}^{J}\left[\frac{\partial \mathbf{g}_j}{\partial \mathbf{x}}\right]_{k,l}\delta(\mathbf{v}-\mathbf{g}_j). \tag{3.29}$$

We can see from (3.29) that $\mathbf{h}$ is a mixture of weighted Dirac delta functions. Thus, we can think of $\mathbf{h}$ as a function that samples the gradient of φ, which is represented as $\mathbf{D}$ in 3.28. In Section 4.4, we show that under different convexity and Lipschitz continuity assumptions, using $\mathbf{h}$ in (3.29) allows DO's training error to strictly decrease or converge to zero.

Discretization: Eq. (3.29) expresses $\mathbf{h}$ as a function with a continuous domain. To compute $\mathbf{h}$ in practice, we need to express it as a vector. To do so, we follow an approximation similar to that in Section 3.2.1: We will discretize $\mathbf{h}$ in (3.29) into a discrete grid, then vectorize it. Specifically, we first discretize $\delta(\mathbf{v}-\mathbf{g}_j)$ into a d-dimensional grid with r boxes in each dimension, where a box evaluates to 1 if $\mathbf{g}_j$ is discretized to that box, and 0 for all other boxes. We then vectorize this grid into a vector. Using the function $\gamma_{q,r}$ from (3.16), we can approximate the $\delta(\mathbf{v}-\mathbf{g}_j)$ as the following Kronecker product of standard basis vectors:

$$\delta(\mathbf{v}-\mathbf{g}_j) \underset{\text{approx.}}{\rightarrow} \mathbf{e}_{\gamma_{q,r}([\mathbf{g}_j]_1)} \otimes \cdots \otimes \mathbf{e}_{\gamma_{q,r}([\mathbf{g}_j]_d)} = \bigotimes_{\alpha=1}^{d}\mathbf{e}_{\gamma_{q,r}([\mathbf{g}_j]_\alpha)} \in \{0,1\}^{r^d}. \tag{3.30}$$

With this discretization, we can express $\mathbf{h}$ in (3.29) in a discrete form as

$$\mathbf{h}(k,l;\mathbf{x}) = \frac{1}{J}\sum_{j=1}^{J}\left[\frac{\partial \mathbf{g}_j}{\partial \mathbf{x}}\right]_{k,l}\bigotimes_{\alpha=1}^{d}\mathbf{e}_{\gamma_{q,r}([\mathbf{g}_j]_\alpha)}. \tag{3.31}$$

By concatenating $\mathbf{h}(k,l;\mathbf{x})$ over k and l, we obtain the final form of $\mathbf{h}$ as

$$\mathbf{h}(\mathbf{x}) = \frac{1}{J}\sum_{j=1}^{J}\bigoplus_{l=1}^{p}\bigoplus_{k=1}^{d}\left[\frac{\partial \mathbf{g}_j}{\partial \mathbf{x}}\right]_{k,l}\bigotimes_{\alpha=1}^{d}\mathbf{e}_{\gamma_{q,r}([\mathbf{g}_j]_\alpha)}, \tag{3.32}$$

where $\bigoplus$ denotes vector concatenation. The dimension of $\mathbf{h}$ is pdr^d. We show how to apply (3.32) to computer vision tasks in Chapter 5.

The above derivation is one possible approach of designing $\mathbf{h}$ to use with DO. Despite we assume differentiability in the derivation, the steps can also be applied when only subdifferentiability is assumed. In addition, we also explore two other forms of $\mathbf{h}$ in this thesis. *(i)* In Section 5.1, we propose an ad-hoc $\mathbf{h}$ for solving the shape-specific point cloud registration task. *(ii)* In Section 6.2, we provide an alternative derivation of $\mathbf{h}$ from basis functions, disregarding the need to discretize delta functions.

3.2.3 Numerical example: Can you guess the number?

In this section, we use the '*Can you guess the number?*' game from Section 1.1 as a numerical example. Recall that the game is trying to solve an optimization problem without knowing the explicit form of its penalty function. Specifically, given a set of number $X = \{x_1, x_2, \ldots, x_J\}$, we are interested in finding the solution x of the problem

$$g_j(x) = 0 = x - x_j, j = 1, \ldots, J. \tag{3.33}$$

A typical approach to solving this problem is to solve the optimization

$$P : \underset{x:x=x_j+\epsilon_j}{\text{minimize}} \sum_{j=1}^{J}\varphi\left(\epsilon_j\right) \equiv \underset{x}{\text{minimize}} \sum_{j=1}^{J}\varphi\left(x - x_j\right), \tag{3.34}$$

for some function φ. In essence, solving problem P in (3.34) is the equal to finding the M-estimator under some distribution of the noise ϵ_j. Suitable forms of φ depends on the assumption on the distribution of ϵ_j, *e.g.*, the maximum likelihood estimation for i.i.d. Gaussian ϵ_j would use $\varphi(x) = x^2$. If the explicit form of φ is known, then one can compute x_* in closed form (*e.g.*, φ

is squared value or absolute value) or with an iterative algorithm. However, using a φ that does not match with the underlying distribution of ϵ_j may lead to an optimal, but incorrect, solution x_*. Here, we will use DO to solve for x_* from a set of training data.

For this problem, we consider 6 φ_β's as follows:

$$\varphi_1(x) = |x|, \tag{3.35}$$
$$\varphi_2(x) = 0.35|x|^{4.32} + 0.15|x|^{1.23}, \tag{3.36}$$
$$\varphi_3(x) = (3 + \text{sgn}(x))x^2/4, \tag{3.37}$$
$$\varphi_4(x) = |x|^{0.7}, \tag{3.38}$$
$$\varphi_5(x) = 1 - \exp(-2x^2), \tag{3.39}$$
$$\varphi_6(x) = 1 - \exp(-8x^2). \tag{3.40}$$

The first 3 φ_β's are convex, where φ_1 is a nonsmooth function; φ_2 is a combination of different powers; φ_3 is an asymmetric function (*i.e.*, $\varphi_3(x) \neq \varphi_3(-x)$). The latter 3 φ_β's are pseudoconvex[7], where φ_4 has the exponent smaller than 1; while φ_5 and φ_6 are inverted Gaussian functions with different widths. Pseudoconvex functions are typically used as robust penalty functions [79] because they penalize outliers less than convex functions. Note that the sum of pseudoconvex functions may not be pseudoconvex, and can have multiple local minima. The graphs of the functions and their gradients[8] are shown in Figure 3.5a,b,d,e. We refer to the problem in (3.34) that uses $\varphi = \varphi_\beta$ as P_β.

We generate the training data for P_β as $\mathcal{X}_\beta = \{(X_\beta^{(i)}, x_{0,\beta}^{(i)}, x_{*,\beta}^{(i)})\}_{i=1}^N$ where $N = 10^4$; $\mathcal{X}_\beta^{(i)} = \{x_{1,\beta}^{(i)}, \ldots, x_{J_i,\beta}^{(i)}\} \subset [-1, 1]$, where we randomly select J_i as an odd number between 3 and 51 to ensure the sum of φ_1 has strictly one global minimum; $x_{0,\beta}^{(i)} = 0$ is the initial estimate; and $x_{*,\beta}^{(i)}$ is the global minimizer of P_β with the data $X_\beta^{(i)}$. To find the global minimizers, we perform grid search with the step size of 0.0001. We trained the SUMs using the $\mathbf{h}$ from (3.19), reproduced here as follows:

$$\mathbf{h}(x) = \frac{1}{J}\sum_{j=1}^{J} \mathbf{e}_{\gamma_{q,r}(x - x_j)}. \tag{3.41}$$

We use $[-2, 2]$ as the range of $x - x_j$ (*i.e.*, $q = 2$), and discretize it into $r = 40$ boxes. Let us denote the SUM learned from the data X_β as SUM_β. To illustrate the training error, we train up to 15 maps for each β, but for test we set the number of maps T to the last map that reduce the training RMSE more than 0.005. During test, we set $\epsilon = 10^{-3}$ and $maxIter = 100$.

[7] We discuss pseudoconvex functions in Section 4.3.

[8] Here, we abuse the word *gradient* to include subdifferential for nonsmooth convex functions and generalized subdifferential for nonconvex functions [50].

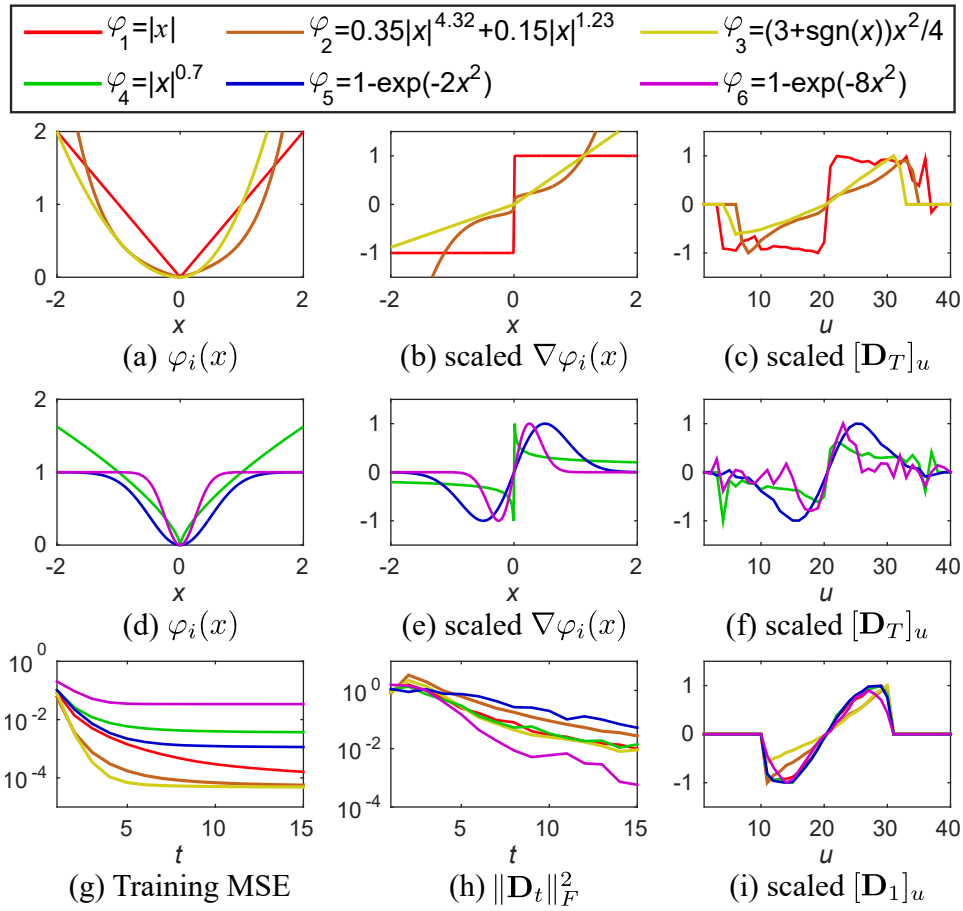


Figure 3.5: Learning to solve unknown cost functions. (a-c) show three convex functions, their gradients, and the learned $\mathbf{D}_T$ for each function. (d-f) show similar figures for pseudoconvex functions. In (c) and (f), their x-axes are the indices of boxes $u = 1, \ldots, 40$ of the corresponding discretized interval $[-2, 2]$. (g) shows training error in each step t. (h) shows the squared norm of the maps $\mathbf{D}_t$. (i) shows the first map of each function.

Figure 3.5c,f show the scaled maps $\mathbf{D}_T$ for each β. The ticks on their x-axes represent the vector indices $u = 1, \ldots, 40$ that correspond to the box indices of the discretized interval $[-2, 2]$. We can see that the maps resemble the gradient of their respective functions, suggesting that DO can learn the approximation of the gradients from training data without an explicit access to the functions. At the far left and right sides of the maps $\mathbf{D}_T$, we can see they take the values of zero. This is because there were no training samples falling into those boxes, and they simply took the values of zero due to the regularization in the learning rule. It should be noted that the first maps for all β in Figure 3.5i are different from their T^{th} maps. This is because the first maps try to move $x_0^{(i)}$ as close to $x_*^{(i)}$ as possible, and thus the shapes of first maps may differ from their respective gradients. The training errors in Figure 3.5g show that convex functions

Table 3.1: MAE of solving unknown cost functions. The first column is P_β used to generate ground truths $x_{*,\beta}$, while the rest show the MAE between the ground truths and the solutions of the algorithms. Best results in underline bold, and second best in bold.

P_β	MAE						
	`fminunc` (Quasi-Newton)						SUM$_\beta$
	P_1	P_2	P_3	P_4	P_5	P_6	
P_1	**<u>.0000</u>**	.0675	.1535	.0419	.0707	.2044	**.0137**
P_2	.0675	**<u>.0000</u>**	.1445	.1080	.1078	.2628	**.0145**
P_3	.1535	.1445	**<u>.0000</u>**	.1743	.1657	.2900	**.0086**
P_4	.0493	.1009	.1682	**.0457**	.0929	.1977	**<u>.0325</u>**
P_5	.0707	.1078	.1657	.0823	**<u>.0000</u>**	.1736	**.0117**
P_6	.2098	.2515	.2791	.1905	.2022	**.1161**	**<u>.0698</u>**

are easier to learn than nonconvex ones. This is because nonconvex functions may have multiple local minima, and $x_t^{(i)}$ may get trapped in a wrong stationary point, causing the error to be high. Figure 3.5h shows that the map have decreasing norms, which represents reducing step sizes as the estimates approach the solutions.

We also perform an experiment on unseen sets of data. Our goal is to compare the SUM learned from the correct cost function against gradient methods where one gradient method minimizes the correct cost function and the rest minimize mismatched ones. Given a set of numbers X, we first find the global minimizer $x_{*,\beta}$ of P_β. Then, we run 7 algorithms and compute the errors between their estimates and $x_{*,\beta}$. In the 7 algorithms, the first is SUM$_\beta$ (learned from $\mathcal{X}_\beta$), and the rest are `fminunc` (quasi-Newton) that minimizes $P_\omega, \omega = 1, \ldots, 6$ (five of which use mismatched cost functions). Table 3.1 shows the mean absolute error (MAE) over 1000 test instances. First, we note that if we use `fminunc` to minimize the correct functions (*i.e.*, estimate $x_{*,\beta}$ by minimizing P_β), then MAEs are generally low. In fact, for convex functions, their MAEs are zero, while MAEs for nonconvex ones may have positive values since gradient methods can get trapped in local minima. On the other hand, we can see that the MAEs are rather large if mismatched functions are used (*e.g.*, estimating $x_{*,1}$ by minimizing P_2). These results illustrate the necessity of selecting the correct cost functions to estimate the desirable solutions. Looking at DO, we see that the solutions of SUM$_\beta$ have errors close to zero and much lower than those of `fminunc` with mismatched cost functions. This shows that DO can learn from the training data to approximate the minimum of an unknown cost function without the need to manually design one as required by conventional optimization. An interesting point to note is that DO seems to be able to solve nonconvex problems better than `fminunc`, suggesting DO can avoid some local minima and more often terminate closer to the global minimum.

We summarize this section in 4 points. *(i)* We show that DO can learn to imitate the gradients

of unknown penalty functions. *(ii)* A very important point to note is that a single training data can have multiple ground truths, and DO will learn to find the solution based on the ground truths provided during the training. Thus, it is unreasonable to use DO that, say, trained with the mean as its ground truth and hope to get the median as the result. *(iii)* A practical implication of this demonstration is that if we optimize a wrong cost function then we may obtain an unexpected optimum as the solution, and it would be more beneficial to obtain training data and learn to solve for the solution directly. *(iv)* We show that for nonconvex problems, DO has the potential to skip local minima and arrive at a better solution than that of the quasi-Newton algorithm (`fminunc`).

3.2.4 Section summary

In this section, we have described a framework for deriving task-specific feature function $\mathbf{h}$ to use with the DO algorithm from Section 3.1. The derivation of the framework is based on the gradient descent for minimizing a cost function which penalized the sum of residual functions $\mathbf{g_j}$ with an unknown penalty function φ. The form of $\mathbf{h}$ is a mixture of weighted Dirac delta functions, which can be thought of as a function that samples the gradient map of φ. We demonstrated using numerical examples that the proposed $\mathbf{h}$ can be used to find the optimum of convex and pseudoconvex functions, while the learned SUMs were shown to approximate the gradient maps of their corresponding functions. This shows the potential of DO as a framework that can robustly solve tasks which are conventionally solved as optimization problems. In the next chapter, we provide theoretical analysis on the convergence in the training phase of DO.

Chapter 4

Theoretical Analysis: Convergence and Relations to Monotonicity and Convexity

In the previous chapter, we describe the DO algorithm and a framework for deriving task-specific feature function $\mathbf{h}$. In this chapter, we perform a theoretical analysis on the algorithm. Specifically, we provide the conditions that guarantee that the estimated parameter $\mathbf{x}_t^{(i)}$ will move closer or converge to the target $\mathbf{x}_*^{(i)}$.

Some convergence aspects of supervised sequential update (SSU) algorithms have been investigated by previous work. In [39], Dollar *et al.* assume a weak learner assumption, and shows that learning a sequence of weak learners allows the estimated parameters to converge to the target parameters in training phase. However, since a weak learner is defined as a regressor that reduces the training error by a constant multiplicative factor, this assumption is rather strong, and only very complex regressors may achieve such error reduction in practice. In addition, [39] focuses on the learners but does not consider their relation with the features, which also has an important role for the convergence result. In [105], Xiong and De la Torre assume *strict monotonicity at a point* and *Lipschitz at a point*, and show that repeatedly applying a single linear map to update the parameters leads to strict reduction in error. These assumptions reveal that the relation between the features and the regressors is an important factor for the error to reduce. However, as we demonstrated with the numerical example in Section 3.1.3, using a single map may not reduce the error beyond a certain value, while a sequence of maps allows the error to decrease much further. The reason for this phenomenon was not mathematically explained in [105]. In addition, the conditions for the training error to converge to zero were not examined.

Our analysis builds upon the work of Xiong and De la Torre [105]. To achieve the convergence results, we rely on two conditions: *Monotonicity at a point* and *relaxed Lipschitz at a point*. These two conditions are similar to, but weaker than, the conditions used in [105]. We also show

that these two conditions can respectively be considered as a generalization of monotonicity and Lipschitz continuity, which are conditions generally used to show the convergence of gradient methods [17]. This relation provides a theoretical link between our algorithm and generalized convexity. Under a different set of the assumptions, we can show that the training error of DO may strictly decrease in each iteration and may even converge to zero. Finally, we show the conditions where using the feature function $\mathbf{h}$ from Section 3.2 allows the training error to strictly decrease or converge to zero. These results form a theoretical foundation for our DO framework.

We begin this chapter by defining *monotonicity at a point* and *relaxed Lipschitz at a point*. Then we provide the main results on the convergence of the training error for the general framework from Section 3.1, followed by the results of $\mathbf{h}$ from Section 3.2. The proofs of all results are provided in Appendix A.

4.1 Definitions

In this section, we define *monotonicity at a point* and *relaxed Lipschitz at a point* .

4.1.1 Monotonicity at a point

Definition 1. (Monotonicity at a point) *A function* $\mathbf{f} : \mathbb{R}^p \to \mathbb{R}^p$ *is*

(i) monotone at $\mathbf{x}_* \in \mathbb{R}^p$*, denoted as* $\mathcal{M}^0(\mathbf{x}_*)$*, if*

$$(\mathbf{x} - \mathbf{x}_*)^\top \mathbf{f}(\mathbf{x}) \geq 0 \tag{4.1}$$

for all $\mathbf{x} \in \mathbb{R}^p$*,*

(ii) strictly monotone at $\mathbf{x}_* \in \mathbb{R}^p$*, denoted as* $\mathcal{M}^+(\mathbf{x}_*)$*, if*

$$(\mathbf{x} - \mathbf{x}_*)^\top \mathbf{f}(\mathbf{x}) \geq 0 \tag{4.2}$$

for all $\mathbf{x} \in \mathbb{R}^p$ *and the equality holds only at* $\mathbf{x} = \mathbf{x}_*$*,*

(iii) strongly monotone at $\mathbf{x}_* \in \mathbb{R}^p$*, denoted as* $\mathcal{M}^{++}(\mathbf{x}_*)$*, if*

$$(\mathbf{x} - \mathbf{x}_*)^\top \mathbf{f}(\mathbf{x}) \geq m\|\mathbf{x} - \mathbf{x}_*\|_2^2 \tag{4.3}$$

for some $m > 0$ *and all* $\mathbf{x} \in \mathbb{R}^p$*.*

We refer to the conditions *monotonicity at a point*, *strict monotonicity at a point*, and *strong monotonicity at a point* without refering to any specific point as $\mathcal{M}^0$, $\mathcal{M}^+$, and $\mathcal{M}^{++}$, *resp.*

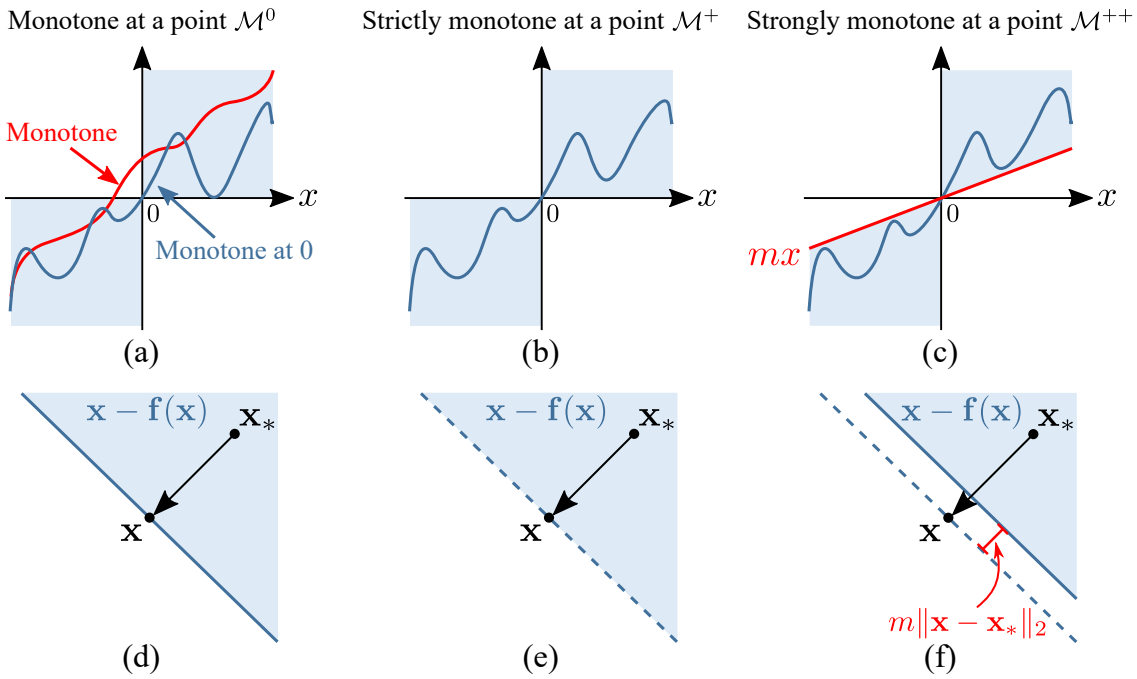


Figure 4.1: Monotonicity at a point. (a-c) Examples of 1D functions which are monotone (*resp.*, strictly, strongly) at 0. (d-f) Examples of the update $\mathbf{x} - \mathbf{f}(\mathbf{x})$ in 2D case. The ranges of the possible update for $\mathbf{f}$'s which are monotone (*resp.*, strictly, strongly) at $\mathbf{x}_*$ are shown in blue.

Figure 4.1 illustrates examples of functions which are monotone at a point. In order for a 1D function $f : \mathbb{R} \to \mathbb{R}$ to be $\mathcal{M}^0(0)$, $\mathcal{M}^+(0)$, and $\mathcal{M}^{++}(0)$, $f(x)$ must lie in the blue regions in Figure 4.1a-c. Figure 4.1a also shows an example of a monotone function in red, which needs to be nondecreasing at every point. On the other hand, a function f which is $\mathcal{M}^0(0)$ can have decreasing values. In fact, f does not even need to be continuous.

We can also obtain the intuition of *monotonicity at a point* by looking at $\mathbf{x} - \mathbf{f}(\mathbf{x})$ as an update operation, *i.e.*, updating $\mathbf{x}$ with $-\mathbf{f}(\mathbf{x})$. In a two dimensional space, Figure 4.1d-f show the range of possible values that $\mathbf{x} - \mathbf{f}(\mathbf{x})$ can take with respect to $\mathbf{x} - \mathbf{x}_*$ for 2D functions $\mathbf{f} : \mathbb{R}^2 \to \mathbb{R}^2$ which are $\mathcal{M}^0(\mathbf{x}_*)$, $\mathcal{M}^+(\mathbf{x}_*)$, and $\mathcal{M}^{++}(\mathbf{x}_*)$, *resp.* It can be seen that the update $\mathbf{x} - \mathbf{f}(\mathbf{x})$ has to lie in the half space on the side of $\mathbf{x}_*$. **This means that if we take a small step in the direction $-\mathbf{f}(\mathbf{x})$, then $\mathbf{x} - \epsilon\mathbf{f}(\mathbf{x})$ should be closer to $\mathbf{x}_*$ than $\mathbf{x}$ for a small $\epsilon > 0$**, which is an important property for our convergence results.

We can also observe the relationship between $\mathcal{M}^0$, $\mathcal{M}^+$, and $\mathcal{M}^{++}$: From Def. 1, we can see that $\mathcal{M}^{++}(\mathbf{x}_*)$ implies $\mathcal{M}^+(\mathbf{x}_*)$, which in turn implies $\mathcal{M}^0(\mathbf{x}_*)$. We will explore the relation between monotonicity and *monotonicity at a point* in more detail in Section 4.3.

Comparison with *monotonicity at a point* of Supervised Descent Method (SDM) [105]

Xiong and De la Torre [105] also introduced similar notions of *monotonicity at a point* and *strict monotonicity at a point*, which are given as follows:

Definition 2. (Monotonicity at a point [105]) *A function* $\tilde{\mathbf{f}} : \mathbb{R}^p \to \mathbb{R}^p$ *is*

(i) monotone at $\mathbf{x}_* \in \mathbb{R}^p$, *if*

$$(\mathbf{x} - \mathbf{x}_*)^\top (\tilde{\mathbf{f}}(\mathbf{x}) - \tilde{\mathbf{f}}(\mathbf{x}_*)) \geq 0 \tag{4.4}$$

for all $\mathbf{x} \in \mathbb{R}^p$,

(ii) strictly monotone at $\mathbf{x}_* \in \mathbb{R}^p$, *if*

$$(\mathbf{x} - \mathbf{x}_*)^\top (\tilde{\mathbf{f}}(\mathbf{x}) - \tilde{\mathbf{f}}(\mathbf{x}_*)) \geq 0 \tag{4.5}$$

for all $\mathbf{x} \in \mathbb{R}^p$ *and the equality holds only at* $\mathbf{x} = \mathbf{x}_*$.

It is important to note that in the SDM paper [105], the SDM update rule proposed is

$$\mathbf{x}_{t+1} = \mathbf{x}_t - (\tilde{\mathbf{f}}(\mathbf{x}_t) - \tilde{\mathbf{f}}(\mathbf{x}_*)), \tag{4.6}$$

where $\tilde{\mathbf{f}}(\mathbf{x}) = \mathbf{D}\tilde{\mathbf{h}}(\mathbf{x})$, and $\tilde{\mathbf{h}}(\mathbf{x}_*)$ is assumed to be a known constant vector (either manually given or learned from a set of training data). Since $\tilde{\mathbf{h}}(\mathbf{x}_*)$ is known, we can modify the SDM update rule into the update rule of DO in (3.4) as:

$$\begin{aligned} \mathbf{x}_{t+1} &= \mathbf{x}_t - \mathbf{D}(\tilde{\mathbf{h}}(\mathbf{x}_t) - \tilde{\mathbf{h}}(\mathbf{x}_*)) && (4.7) \\ &= \mathbf{x}_t - \mathbf{D}\mathbf{h}(\mathbf{x}_t). && (4.8) \end{aligned}$$

This allows us to relate our definitions (Def. 1) to those of SDM [105] (Def. 2) by defining $\mathbf{f}(\mathbf{x})$ in Def. 1 as $\tilde{\mathbf{f}}(\mathbf{x}) - \tilde{\mathbf{f}}(\mathbf{x}_*)$. This shows that Def. 1 subsumes Def. 2. A major difference between the two definitions is that Def. 1 does not require $\mathbf{f}(\mathbf{x}_*)$ to be zero, while for Def. 2 we always have $\tilde{\mathbf{f}}(\mathbf{x}) - \tilde{\mathbf{f}}(\mathbf{x}_*) = \mathbf{0}_p$ when $\mathbf{x} = \mathbf{x}_*$. This shows that Def. 1 is more general than Def. 2. From a practical perspective, it is unlikely that the update $\mathbf{f}(\mathbf{x}) = \mathbf{D}\mathbf{h}(\mathbf{x})$ will be zero even at $\mathbf{x} = \mathbf{x}_*$, since $\mathbf{h}$ might return features such as SIFT, HOG, or some transformation of the residual functions, which are likely to be noisy and have non-zero values when multiplied with $\mathbf{D}$. Thus, Def. 1 defines a class of functions that covers more practical situations than Def. 2.

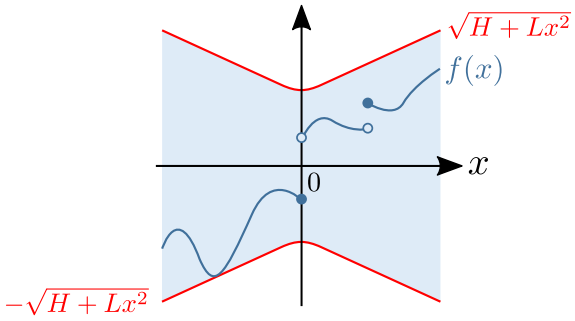


Figure 4.2: An example of 1D (H, L)-relaxed-Lipschitz-at-0 function is shown as the blue curve. In fact, any 1D function that lies in the blue region (bounded by the red curves) is (H, L)-relaxed Lipschitz at 0.

4.1.2 Relaxed Lipschitz at a point ($\mathcal{RL}$)

We define *relaxed Lipschitz at a point* ($\mathcal{RL}$) as follows.

Definition 3. (Relaxed Lipschitz at a point) *A function* $\mathbf{f} : \mathbb{R}^p \to \mathbb{R}^q$ *is* (H, L)*-relaxed Lipschitz at* $\mathbf{x}_*$*, denoted as* (H, L)*-*$\mathcal{RL}(\mathbf{x}_*)$*, if there exists* $H, L \geq 0$ *such that*

$$\|\mathbf{f}(\mathbf{x})\|_2^2 \leq H + L\|\mathbf{x} - \mathbf{x}_*\|_2^2 \tag{4.9}$$

for all $\mathbf{x} \in \mathbb{R}^p$.

Figure 4.2 shows an example of a 1D function which is (H, L)-$\mathcal{RL}(0)$: Any 1D function which lies in the blue region is (H, L)-$\mathcal{RL}(0)$. Similar to a Lipschitz continuous function, a function $\mathbf{f}$ which is $\mathcal{RL}$ has a bound on its growth. However, while Lipschitz continuous functions constrain the function values based on the distance between any pair of points, $\mathcal{RL}(\mathbf{x}_*)$ depends only on the distance from any point $\mathbf{x}$ to a single point $\mathbf{x}_*$. This allows $\mathbf{f}$ to be discontinuous. In particular, if $H > 0$ then (H, L)-$\mathcal{RL}(\mathbf{x}_*)$ allows $\mathbf{f}$ to be discontinuous even at $\mathbf{x}_*$.

Other than Lipschitz continuity, the $\mathcal{RL}$ condition also relates to other similar concepts in the literature. Notably, if $L = 0$ then (H, L)-$\mathcal{RL}$ reduces to a bounded function, while if $H = 0$ then (H, L)-$\mathcal{RL}$ reduces to *Lipschitz at a point* defined in [105]. In [93], Tian and Narasimhan also introduce a similar concept called *relaxed Lipshitz condition*. This condition provides both upper bound and lower bound on the image difference as a ratio of the distortion parameter difference, which in a sense is similar to a combination of Lipschitz continuity and strong monotonicity. However, the *relaxed Lipshitz condition* in [93] also applies to multivalued maps, *i.e.*, its arguments need not be a function. The condition (H, L)-$\mathcal{RL}$ also relates to the concept of inexact oracle in first-order methods for smooth convex minimization (see Cor. 1 in [36], and also [35]).

4.2 Convergence of the training error

Using the above definitions, we can derive the following main convergence result on the training error of DO.

Theorem 1. (Convergence of DO's training error) *Given a training set* $\{(\mathbf{x}_0^{(i)}, \mathbf{x}_*^{(i)}, \mathbf{h}^{(i)})\}_{i=1}^N$, *if there exists a linear map* $\hat{\mathbf{D}} \in \mathbb{R}^{p\times f}$ *where* $\hat{\mathbf{D}}\mathbf{h}^{(i)}$ *is* $\mathcal{M}^+(\mathbf{x}_*^{(i)})$ *for all* i, *and if there exists an* i *where* $\mathbf{x}_t^{(i)} \neq \mathbf{x}_*^{(i)}$, *then using the update rule:*

$$\mathbf{x}_{t+1}^{(i)} = \mathbf{x}_t^{(i)} - \mathbf{D}_{t+1}\mathbf{h}^{(i)}(\mathbf{x}_t^{(i)}), \tag{4.10}$$

with $\mathbf{D}_{t+1} \subset \mathbb{R}^{p\times f}$ *obtained from* (3.5)*, guarantees that the training error strictly decreases in each iteration:*

$$\sum_{i=1}^{N} \|\mathbf{x}_*^{(i)} - \mathbf{x}_{t+1}^{(i)}\|_2^2 < \sum_{i=1}^{N} \|\mathbf{x}_*^{(i)} - \mathbf{x}_t^{(i)}\|_2^2. \tag{4.11}$$

Moreover, if $\hat{\mathbf{D}}\mathbf{h}^{(i)}$ *is* $\mathcal{M}^{++}(\mathbf{x}_*^{(i)})$ *and* (H, L)*-*$\mathcal{RL}(\mathbf{x}_*^{(i)})$*, then the training error converges to zero. If* $H = 0$ *then the training error converges to zero linearly.*

Proof. See Appendix A.1. □

In words, Thm. 1 says that if each instance i is similar in the sense that each $\hat{\mathbf{D}}\mathbf{h}^{(i)}$ is $\mathcal{M}^+(\mathbf{x}_*^{(i)})$, then sequentially learning the optimal maps with (3.5) guarantees that the training error strictly reduces in each iteration. In addition, if $\hat{\mathbf{D}}\mathbf{h}^{(i)}$ is $\mathcal{M}^{++}(\mathbf{x}_*^{(i)})$ and (H, L)-$\mathcal{RL}(\mathbf{x}_*^{(i)})$ then the error converges to zero. Note that $\mathbf{h}^{(i)}$ is not required to be differentiable or continuous. Xiong and De la Torre [105] also present a convergence result for a similar update rule, but it shows the strict reduction of error of a *single* function under a *single ideal* map. It also requires an additional *Lipschitz at a point* condition for the error to strictly decrease. This condition is necessary for bounding the norm of the map, otherwise the update can be too large, preventing the reduction in error. In contrast, Thm. 1 explains the convergence of *multiple* functions under the same sequence of update maps (SUM) learned from the data, where each learned map $\mathbf{D}_t$ can be different from the ideal map $\hat{\mathbf{D}}$. To ensure strict reduction of error, Thm. 1 does not require the *Lipschitz at a point* condition since the norms of the maps are adjusted based on the training data. Meanwhile, to ensure convergence to zero, Thm. 1 requires $\mathcal{RL}$ (recall that $\hat{\mathbf{D}}\mathbf{h}^{(i)}(\mathbf{x}_*^{(i)})$ does not need to be $\mathbf{0}_p$). These weaker assumptions have an important implication as they allow robust discontinuous features, such as HOG in [105], to be used as $\mathbf{h}^{(i)}$. Finally, we wish to point out that Thm. 1 guarantees the reduction in the average error, not the error of each instance i.

4.2.1 Analytical examples

We can see that there are two different convergence results on the training error: strictly decrease (based on $\mathcal{M}^+$) and convergence to zero (based on $\mathcal{M}^{++}$ and $\mathcal{RL}$). This raises some interesting questions. For example, can we show the decrease in error for the weaker $\mathcal{M}^0$? Are the two classes of convergence results strictly different? In this section, we provide analytical examples to show the necessity of each set of assumptions, which implies that each convergence results are strictly different.

Example 1: Monotone-at-a-point functions ($\mathcal{M}^0$)

Here, we show examples of training sets with $\hat{\mathbf{D}}\mathbf{h}$ which is $\mathcal{M}^0$ where their training errors may not strictly decrease in each iteration.

1. A trivial example is $\mathbf{h}(\mathbf{x}) = \mathbf{0}_f$, where $\hat{\mathbf{D}}\mathbf{h}$ is monotone at all $\mathbf{x} \in \mathbb{R}^p$ for any $\hat{\mathbf{D}}$. In this case, for any training set with any $\mathbf{x}_*^{(i)}$, the training error will not decrease.
2. Consider the training set $\{(\mathbf{x}_0^{(i)}, \mathbf{x}_*^{(i)}, \mathbf{h}^{(i)})\}_{i=1}^2 \subset \mathbb{R}^2 \times \mathbb{R}^2 \times (\mathbb{R}^2 \to \mathbb{R}^2)$, where $\mathbf{x}_0^{(1)} = \mathbf{x}_0^{(2)}$, $\mathbf{x}_*^{(1)} = \mathbf{x}_*^{(2)} = \mathbf{0}_2$, and

$$\mathbf{h}^{(1)}(\mathbf{x}) = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \mathbf{x}, \text{ and } \mathbf{h}^{(2)}(\mathbf{x}) = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \mathbf{x}. \tag{4.12}$$

   The function $\mathbf{h}^{(1)}$ generates a clockwise vector field around $\mathbf{0}_2$, while $\mathbf{h}^{(2)}$ generates a counterclockwise one. We can see that with $\hat{\mathbf{D}} = \mathbf{I}_2$, we have $(\mathbf{x} - \mathbf{x}_*^{(i)})^\top \hat{\mathbf{D}}\mathbf{h}^{(i)}(\mathbf{x}) = 0$ for any $\mathbf{x}$ for all i, meaning $\hat{\mathbf{D}}\mathbf{h}^{(i)}$ is $\mathcal{M}^0(\mathbf{x}_*^{(i)})$. Observe that, in the first iteration, we have $\mathbf{h}^{(1)}(\mathbf{x}_0^{(1)}) = -\mathbf{h}^{(2)}(\mathbf{x}_0^{(2)})$ while $\mathbf{x}_0^{(1)} - \mathbf{x}_*^{(1)} = \mathbf{x}_0^{(2)} - \mathbf{x}_*^{(2)}$. This will lead to $\mathbf{D}_1 = \mathbf{0}_{2\times 2}$, which means all updates will be zero and that we will have $\mathbf{x}_0^{(1)} = \mathbf{x}_0^{(2)} = \mathbf{x}_t^{(1)} = \mathbf{x}_t^{(2)}$ for all $t = 1, 2, \ldots$. Thus the training error will not decrease.

   On the other hand, consider a training set $\{(\mathbf{x}_0^{(i)}, \mathbf{x}_*^{(i)}, \mathbf{h}^{(1)})\}_{i=1}^N$ for any $N > 0$, where $\mathbf{x}_*^{(i)} = \mathbf{0}_2$, $\mathbf{x}_0^{(i)}$ is any points in $\mathbb{R}^2$, and $\mathbf{h}^{(1)}$ is defined in (4.12). Then, we can see that for

$$\hat{\mathbf{D}} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}, \tag{4.13}$$

   we have that $\hat{\mathbf{D}}\mathbf{h}^{(i)}$ is $\mathcal{M}^{++}(\mathbf{x}_*^{(i)})$ and $(0, 1)$-$\mathcal{RL}(\mathbf{x}_*^{(i)})$ for all i. From Thm. 1, the training error converges to zero linearly. In fact, for this specific case, the training error converges to zero in a single iteration.

The above examples show that we cannot guarantee that the training error of $\mathcal{M}^0$ functions will strictly decrease in each iteration. Next, we look at $\mathcal{M}^+$ functions.

Example 2: Strictly-monotone-at-a-point function ($\mathcal{M}^+$)

In this example, we show that the training error of a function which is $\mathcal{M}^+(\mathbf{0}_2)$ may not converge to $\mathbf{0}_2$, illustrating the distinction between the convergence results of $\mathcal{M}^+$ and $\mathcal{M}^{++}$ in Thm. 1.

Consider

$$\mathbf{h}(\mathbf{x}) = \begin{cases} (0, \text{sgn}(x_2)) & ; x_2 \neq 0, \\ (\text{sgn}(x_1), 0) & ; x_2 = 0. \end{cases} \tag{4.14}$$

With $\hat{\mathbf{D}} = \mathbf{I}_2$, we can see that $\hat{\mathbf{D}}\mathbf{h}$ is $\mathcal{M}^+(\mathbf{0}_2)$.[1] Suppose we are given the following training set with three samples $\{(\mathbf{x}_0^{(i)}, \mathbf{x}_*^{(i)}, \mathbf{h})\}_{i=1}^3$, where $\mathbf{x}_0^{(1)} = (1, 1)$, $\mathbf{x}_0^{(2)} = (1, -1)$, $\mathbf{x}_0^{(3)} = (0, 2)$, and $\mathbf{x}_*^{(i)} = \mathbf{0}_2$ for all i. With these training samples, we can explicitly derive $\mathbf{D}_t$ and $\mathbf{x}_t^{(i)}, t = 0, 1, \ldots,$ as

$$\mathbf{D}_t = \frac{1}{3^t} \begin{bmatrix} 0 & 0 \\ 0 & 4 \end{bmatrix}, \mathbf{x}_t^{(1)} = \begin{bmatrix} 1 \\ \left(-\frac{1}{3}\right)^t \end{bmatrix}, \mathbf{x}_t^{(2)} = \begin{bmatrix} 1 \\ -\left(-\frac{1}{3}\right)^t \end{bmatrix}, \mathbf{x}_t^{(3)} = \begin{bmatrix} 0 \\ 2\left(\frac{1}{3}\right)^t \end{bmatrix}. \tag{4.15}$$

It can be seen that the samples' second coordinates $[\mathbf{x}_t^{(i)}]_2$ are geometric sequences which approach, but never reach, zero. This prevent $[\mathbf{x}_t^{(i)}]_1, i = 1, 2,$ from decreasing, thus the training error will not converge to 0.

Example 3: Strongly-monotone-at-a-point function ($\mathcal{M}^{++}$)

In this example, we show that the training error of a $\mathcal{M}^{++}$ function without (H, L)-$\mathcal{RL}$ may not converge to 0, implying (H, L)-$\mathcal{RL}$ is an important condition for the convergence of the training error.

Consider the following 1D function:

$$h(x) = \begin{cases} \frac{1}{x} + x & ; x \neq 0, \\ 0 & ; x = 0. \end{cases} \tag{4.16}$$

It can be seen that with $\hat{D} = 1$, $\hat{D}h$ is $\mathcal{M}^{++}(0)$ with $m = 1$. However, the value of $\hat{D}h(x)$ as $x \to 0$ is unbounded. Suppose we are given the following training set with two samples $\{(x_0^{(i)}, x_*^{(i)}, h)\}_{i=1}^2$, where $x_0^{(1)} = 1$, $x_0^{(2)}$ is an arbitrary small positive number, and $x_*^{(i)} = 0$ for all i. We can see that $h(x_0^{(1)}) = 2$, while $h(x_0^{(2)}) \to \infty$ as $x_0^{(2)} \to 0^+$. As $x_0^{(2)} \to 0^+$, we will have $D_1 \to 0$, meaning $x^{(1)}$ will not make any progress towards 0, thus the training error will not converge to 0.

[1] Note that $\hat{\mathbf{D}}\mathbf{h}$ is also $(1, 0)$-$\mathcal{RL}(\mathbf{0}_2)$.

4.3 Relation to generalized monotonicity and generalized convexity

In the previous section, we provide the conditions for the strict reduction and convergence to zero of DO's training error. We can see that *strict and strong monotonicity at a point* are important conditions for the convergence results. In this section, we explore the relation between *monotonicity at a point* and generalized monotonicity and generalized convexity. Specifically, we show that *monotonicity at a point* is a generalization of monotonicity and pseudomonotonocity, which are the properties of the gradient of convex and pseudoconvex functions [57, 83]. This relation allows us to derive the convergence results when we use the feature function $\mathbf{h}$ from Section 3.2 with DO.

We begin this section by providing definitions for monotone and pseudomonotone functions (*i.e.*, *single-valued case*), and their relation to convex and pseudoconvex functions. Then we state our result that relates these concepts to *monotonicity at a point*. Next, we show that these results can be generalized to the case of monotone and pseudomonotone multivalued maps (*i.e.*, *multi-valued case*), which are related to nonsmooth convex and pseudoconvex functions. Finally, we apply the derived relation to show the convergence when using the feature function $\mathbf{h}$ from Section 3.2.

4.3.1 Single-valued case

Convexity is one of the most well-studied properties in optimization. Since all local minima of a convex function are also global minima, convex optimization problems can be solved using computationally efficient algorithms [74]. We will refer to this property as *local-global property*. Convex optimization has been widely applied to a variety of fields from finance and economy, transportation planning, control theory, to machine learning and computer vision [17]. However, the local-global property is not confined to only convex functions. In this work, we focus on a class of functions called *pseudoconvex*, which generalizes convexity while still retaining the local-global property[2]. This property allows gradient-based algorithms to efficiently find a global minimum[3]. Pseudoconvex functions have been used as penalty functions for their stronger

[2]More generally, the class of functions where it is both sufficient and necessary for all stationary points to be global minima is called *Invex functions* [12]. These functions do not require the sublevel sets to be convex sets, while convex and pseudoconvex do. We do not focus on invex functions in this work.

[3]It is known that all pseudoconvex functions are also quasiconvex [57]. However, quasiconvex functions can have stationary points which are not local minima. We do not focus on quasiconvex functions in this work.

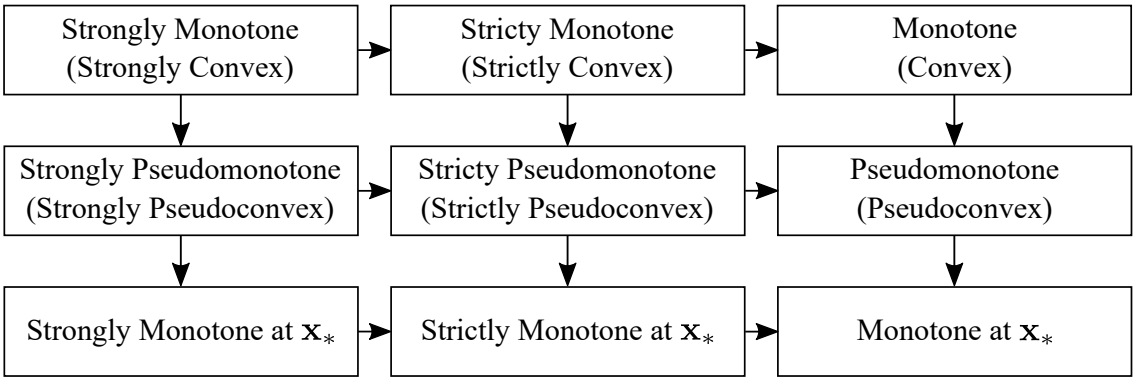


Figure 4.3: Suppose $f : \mathbb{R}^p \to \mathbb{R}$ is a differentiable function with $\mathbf{x}_* = \arg\min_{\mathbf{x}} f(\mathbf{x})$. This diagram provides the relation between the monotonicity conditions of ∇f. The boxes in the top two rows indicate the classes of ∇f and the corresponding classes of f in parentheses.

robustness than convex ones [6, 16, 79]. However, unlike convexity, pseudoconvexity is not preserved under summation, *i.e.*, the sum of pseudoconvex functions may not be pseudoconvex and can have multiple local minima.

Relating to convexity and pseudoconvexity are the concepts of monotonicity and pseudomonotonicity. Specifically, it can be shown that the gradients of convex functions are monotone functions, while the gradients of pseudoconvex functions are pseudomonotone functions [57].

In this section, we will review the relation between these concepts, and show that monotone and pseudomonotone functions are monotone at their zeros. This leads us to the result that says that the gradients of convex and pseudoconvex functions are monotone at their minima. First, we will review the definitions of convexity, pseudoconvexity, monotonicity, and pseudoconvexity. Then, we will provide our result on the relation between these concepts and *monotonicity at a point*. Figure 4.3 provides a summary of the relation in terms of the gradient of a differentiable function.

Definitions: Convex and pseudoconvex functions

Definition 4. (Convexity [83]) *A function $f : \mathbb{R}^p \to \mathbb{R}$ is*

(i) convex *if for any pairs of points $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^p$ and any $\alpha \in [0, 1]$,*

$$\alpha f(\mathbf{x}) + (1 - \alpha) f(\mathbf{x}') \geq f(\alpha \mathbf{x} + (1 - \alpha)\mathbf{x}'), \tag{4.17}$$

(ii) strictly convex *if for any pairs of points $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^p$ and any $\alpha \in [0, 1]$,*

$$\alpha f(\mathbf{x}) + (1 - \alpha) f(\mathbf{x}') \geq f(\alpha \mathbf{x} + (1 - \alpha)\mathbf{x}'), \tag{4.18}$$

where equality holds only if $\mathbf{x} = \mathbf{x}'$*,*

(iii) strongly convex *if there exists* $m > 0$ *such that for any pairs of points* $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^p$ *and any* $\alpha \in [0, 1]$*,*

$$\alpha f(\mathbf{x}) + (1-\alpha) f(\mathbf{x}') \geq f(\alpha \mathbf{x} + (1-\alpha)\mathbf{x}') + \frac{m}{2}\alpha(1-\alpha)\|\mathbf{x} - \mathbf{x}'\|_2^2. \tag{4.19}$$

Definition 5. (Pseudoconvexity [57]) *A differentiable function* $f : \mathbb{R}^p \to \mathbb{R}$ *is*

(i) pseudoconvex *if for any distinct points* $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^p$*,*

$$(\mathbf{x} - \mathbf{x}')^\top \nabla f(\mathbf{x}') \geq 0 \implies f(\mathbf{x}) \geq f(\mathbf{x}'), \tag{4.20}$$

(ii) strictly pseudoconvex *if for any distinct points* $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^p$*,*

$$(\mathbf{x} - \mathbf{x}')^\top \nabla f(\mathbf{x}') \geq 0 \implies f(\mathbf{x}) > f(\mathbf{x}'), \tag{4.21}$$

(iii) strongly pseudoconvex *if there exists* $m > 0$ *such that for any distinct points* $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^p$*,*

$$(\mathbf{x} - \mathbf{x}')^\top \nabla f(\mathbf{x}') \geq 0 \implies f(\mathbf{x}) \geq f(\mathbf{x}') + m\|\mathbf{x} - \mathbf{x}'\|_2^2. \tag{4.22}$$

It can be shown that differentiable convex (*resp.*, strictly convex, strongly convex) functions are pseudoconvex (*resp.*, strictly pseudoconvex, strongly pseudoconvex) [50]. Next, we define monotone and pseudomonotone functions.

Definitions: Monotone and pseudomonotone functions

Definition 6. (Monotonicity [57]) *A function* $\mathbf{f} : \mathbb{R}^p \to \mathbb{R}^p$ *is*

(i) monotone *if for any distinct points* $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^p$*,*

$$(\mathbf{x} - \mathbf{x}')^\top (\mathbf{f}(\mathbf{x}) - \mathbf{f}(\mathbf{x})') \geq 0, \tag{4.23}$$

(ii) strictly monotone *if for any distinct points* $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^p$*,*

$$(\mathbf{x} - \mathbf{x}')^\top (\mathbf{f}(\mathbf{x}) - \mathbf{f}(\mathbf{x})') > 0, \tag{4.24}$$

(iii) strongly monotone *if there exists* $m > 0$ *such that for any distinct points* $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^p$*,*

$$(\mathbf{x} - \mathbf{x}')^\top (\mathbf{f}(\mathbf{x}) - \mathbf{f}(\mathbf{x})') \geq m\|\mathbf{x} - \mathbf{x}'\|_2^2. \tag{4.25}$$

Definition 7. (Pseudomonotonicity [57]) *A function* $\mathbf{f} : \mathbb{R}^p \to \mathbb{R}^p$ *is*

(i) pseudomonotone *if for any distinct points* $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^p$,

$$(\mathbf{x} - \mathbf{x}')^\top \mathbf{f}(\mathbf{x}') \geq 0 \implies (\mathbf{x} - \mathbf{x}')^\top \mathbf{f}(\mathbf{x}) \geq 0, \tag{4.26}$$

(ii) strictly pseudomonotone *if for any distinct points* $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^p$,

$$(\mathbf{x} - \mathbf{x}')^\top \mathbf{f}(\mathbf{x}') \geq 0 \implies (\mathbf{x} - \mathbf{x}')^\top \mathbf{f}(\mathbf{x}) > 0, \tag{4.27}$$

(iii) strongly pseudomonotone *if there exists* $m > 0$ *such that for any distinct points* $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^p$,

$$(\mathbf{x} - \mathbf{x}')^\top \mathbf{f}(\mathbf{x}') \geq 0 \implies (\mathbf{x} - \mathbf{x}')^\top \mathbf{f}(\mathbf{x}) \geq m\|\mathbf{x} - \mathbf{x}'\|_2^2. \tag{4.28}$$

Similar to convexity and pseudoconvexity, it can also be shown that monotone (resp., strictly monotone, strongly monotone) functions are pseudomonotone (resp., strictly pseudomonotone, strongly pseudomonotone) [57]. The following proposition provides a relation between the gradients of convex and pseudoconvex functions and monotone and pseudomonotone functions.

Proposition 1. (Generalized convexity and generalized monotonicity [57]) *A differentiable function* $f : \mathbb{R}^p \to \mathbb{R}$ *is convex (resp., strictly convex, strongly convex, pseudoconvex, strictly pseudoconvex, strongly pseudoconvex) if and only if its gradient is monotone (resp., strictly monotone, strongly monotone, pseudomonotone, strictly pseudomonotone, strongly pseudomonotone).*

Note that Prop. 1 only applies to the gradient of a function; A function $\mathbf{g} : \mathbb{R}^p \to \mathbb{R}^p$ which is monotone may not be a gradient of a convex function. For example, $\mathbf{g}(x_1, x_2) = (-x_2, x_1)$ is not the gradient of any function.

Results: Generalized monotone functions and monotonicity at a point

Next, we derive our result on the relation between *monotonicity at a point* and pseudomonotonicity.

Proposition 2. (Pseudomonotonicity and monotonicity at a point) *If a function* $\mathbf{f} : \mathbb{R}^p \to \mathbb{R}^p$ *is pseudomonotone (resp., strictly pseudomonotone, strongly pseudomonotone) and* $\mathbf{f}(\mathbf{x}_*) = \mathbf{0}_p$, *then* $\mathbf{f}$ *is monotone (resp., strictly monotone, strongly monotone) at* $\mathbf{x}_*$.

Proof. See Appendix A.2. □

Prop. 2 shows that *monotonicity at a point* is a generalization of pseudomonotonicity. The converse of the proposition is not true. For example, $\mathbf{f}(\mathbf{x}) = [x_1 x_2^2 + x_1, x_2 x_1^2 + x_2]^\top$ is strictly

monotone at $\mathbf{0}_2$, but not strictly pseudomonotone (counterexample at $\mathbf{x} = (1, 2)$ and $\mathbf{y} = (2, 1)$).

With Prop. 1 and 2, together with the fact that the gradient of a function at its minimum is zero, we can obtain the following result which implies that *monotonicity at a point* is weaker than the conditions for the gradient map of pseudoconvex and differentiable convex functions.

Corollary 1. (Pseudomonotone functions and monotonicity at a point) *If a function $f : \mathbb{R}^p \to \mathbb{R}$ is pseudoconvex (resp., strictly pseudoconvex, strongly pseudoconvex) with a minimum at $\mathbf{x}_*$, then ∇f is monotone (resp., strictly monotone, strongly monotone) at $\mathbf{x}_*$.*

4.3.2 Multi-valued case

In the previous section, we look at the relation between monotone and pseudomonotone functions and *monotonicity at a point*. In this section, we generalize the results to the case of multivalued maps. Multivalued maps generalize functions: Whereas a function $\mathbf{f} : \mathbb{R}^p \to \mathbb{R}^q$ is a mapping from an input argument to a single value, *i.e.*, $\mathbf{f}(\mathbf{x}) \in \mathbb{R}^q$, a multivalued map $\mathbf{f} : \mathbb{R}^p \to 2^{\mathbb{R}^q}$ is a mapping from an input argument to a set which may contain multiple values $\mathbf{f}(\mathbf{x}) \subset \mathbb{R}^q$. An example of multivalued maps is the subdifferential of convex functions [83], *e.g.*, the subdifferential of $|x|$ at $x = 0$ is the set $[-1, 1]$. Here, we will show that monotone and pseudomonotone multivalued maps are monotone at their zeros. This result allows us to relate *monotonicity at a point* to nonsmooth convex functions.

To obtain the result, we first define monotone and pseudomonotone multivalued maps. Then, using the property that subdifferentials of convex functions are monotone multivalued maps, we can show that the subdifferentials of the convex functions are monotone at their minima.

Definitions: Monotone and pseudomonotone multivalued maps

Definition 8. (Monotone multivalued map [108]) *A multivalued map $\mathbf{f} : \mathbb{R}^p \to 2^{\mathbb{R}^p}$ is*

(1) monotone *if for any distinct points $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^p$ and any $\mathbf{u} \in \mathbf{f}(\mathbf{x}), \mathbf{u}' \in \mathbf{f}(\mathbf{x}')$,*

$$(\mathbf{x} - \mathbf{x}')^\top (\mathbf{u} - \mathbf{u}') \geq 0, \tag{4.29}$$

(2) strictly monotone *if for any distinct points $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^p$ and any $\mathbf{u} \in \mathbf{f}(\mathbf{x}), \mathbf{u}' \in \mathbf{f}(\mathbf{x}')$,*

$$(\mathbf{x} - \mathbf{x}')^\top (\mathbf{u} - \mathbf{u}') > 0, \tag{4.30}$$

(3) strongly monotone *if there exists $m > 0$ such that for any distinct points $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^p$ and any $\mathbf{u} \in \mathbf{f}(\mathbf{x}), \mathbf{u}' \in \mathbf{f}(\mathbf{x}')$,*

$$(\mathbf{x} - \mathbf{x}')^\top (\mathbf{u} - \mathbf{u}') \geq m\|\mathbf{x} - \mathbf{x}'\|_2^2. \tag{4.31}$$

Definition 9. (Pseudomonotone multivalued map [108]) *A multivalued map* $\mathbf{f} : \mathbb{R}^p \to 2^{\mathbb{R}^p}$ *is*

(1) pseudomonotone *if for any distinct points* $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^p$ *and any* $\mathbf{u} \in \mathbf{f}(\mathbf{x}), \mathbf{u}' \in \mathbf{f}(\mathbf{x}')$,

$$(\mathbf{x} - \mathbf{x}')^\top \mathbf{u}' \geq 0 \implies (\mathbf{x} - \mathbf{x}')^\top \mathbf{u} \geq 0, \tag{4.32}$$

(2) strictly pseudomonotone *if for any distinct points* $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^p$ *and any* $\mathbf{u} \in \mathbf{f}(\mathbf{x}), \mathbf{u}' \in \mathbf{f}(\mathbf{x}')$,

$$(\mathbf{x} - \mathbf{x}')^\top \mathbf{u}' \geq 0 \implies (\mathbf{x} - \mathbf{x}')^\top \mathbf{u} > 0, \tag{4.33}$$

(3) strongly pseudomonotone *if there exists* $m > 0$ *such that for any distinct points* $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^p$ *and any* $\mathbf{u} \in \mathbf{f}(\mathbf{x}), \mathbf{u}' \in \mathbf{f}(\mathbf{x}')$,

$$(\mathbf{x} - \mathbf{x}')^\top \mathbf{u}' \geq 0 \implies (\mathbf{x} - \mathbf{x}')^\top \mathbf{u} \geq m\|\mathbf{x} - \mathbf{x}'\|_2^2. \tag{4.34}$$

It is known that monotone (*resp.* strictly monotone, strongly monotone) multivalued maps are pseudomonotone (*resp.* strictly pseudomonotone, strongly pseudomonotone) multivalued maps [50]. The following results connects monotone multivalued maps to convex functions.

Proposition 3. (Convex functions and monotone multivalued maps [65, 83]) *A function* $f : \mathbb{R}^p \to \mathbb{R}$ *is convex (resp., strictly convex, strongly convex) if and only if its subdifferential* ∂f *is monotone (resp., strictly monotone, strongly monotone).*

Results: Generalized monotone multivalued maps and monotonicity at a point

Before we proceed to our result, let us define *induced function*.

Definition 10. (Induced function) *Let* $\mathbf{f} : \mathbb{R}^p \to 2^{\mathbb{R}^p}$ *be a multivalued map. A function* $\hat{\mathbf{f}} : \mathbb{R}^p \to \mathbb{R}^p$ *is an induced function of* $\mathbf{f}$ *if for all* $\mathbf{x}$*, we have* $\hat{\mathbf{f}}(\mathbf{x}) = \mathbf{u}$ *for some* $\mathbf{u} \in \mathbf{f}(\mathbf{x})$.

In words, at each $\mathbf{x}$, the induced function $\hat{\mathbf{f}}$ returns a single value from the set $\mathbf{f}(\mathbf{x})$. We can think of an induced function as a function that picks a single output from the set of outputs from a multivalued map, thereby *flattening* the multivalued map into a function. With the above definitions, we can derive the following result, which generalize Prop. 2 to the case of multivalued maps.

Proposition 4. (Pseudomonotone multivalued map and monotonicity at a point) *Suppose a multivalued map* $\mathbf{f}$ *is pseudomonotone (resp., strictly, strongly) with* $\mathbf{0}_p \in \mathbf{f}(\mathbf{x}_*)$. *Let* $\hat{\mathbf{f}}$ *be an induced function of* $\mathbf{f}$. *Then* $\hat{\mathbf{f}}$ *is monotone (resp., strictly, strongly) at* $\mathbf{x}_*$.

Proof. See Appendix A.3. □

With Prop. 3 and 4, we can obtain the following result which shows that the subgradient of a convex function is monotone at its minimum. This implies that *monotonicity at a point* is a weaker condition than that of the subgradients of convex functions.

Corollary 2. (Nonsmooth convex functions and monotonicity at a point) *Suppose a function $f : \mathbb{R}^p \to \mathbb{R}$ is convex (resp., strictly convex, strongly convex) with a minimum at $\mathbf{x}_*$. Let $\mathbf{g} : \mathbb{R}^p \to \mathbb{R}^p$ be an induced function of the subdifferential ∂f. Then $\mathbf{g}$ is monotone (resp., strictly monotone, strongly monotone) at $\mathbf{x}_*$.*

Extension to nonsmooth pseudoconvex functions

Extending Cor. 1 for the case of nonsmooth pseudoconvex is not simple as Def. 5 defines pseudoconvex functions to be differentiable functions. Unlike convex functions where we can replace gradients with subdifferentials, pseudoconvex functions cannot use the same definition of subdifferentials as those of convex functions as the they may return empty sets when the function is not convex. To handle this issues, different definitions of pseudoconvex functions and subdifferentials have been proposed (see [50]), but their relation to pseudomonotone multivalued maps are not as straightforward when compared with those in the convex cases. Thus, we do not extend our results to nonsmooth pseudoconvex functions in this thesis.

4.4 Convergence of DO's training error with task-specific feature function $\mathbf{h}$

Building upon the results in the previous sections, we present the convergence result for training DO when the task-specific feature function $\mathbf{h}$ from (3.29) is used[4]. First, we begin with the following case for differentiable pseudoconvex cost functions.

Proposition 5. (Convergence of the training error with an unknown differentiable cost function) *Given a training set $\{(\mathbf{x}_0^{(i)}, \mathbf{x}_*^{(i)}, \{\mathbf{g}_j^{(i)}\}_{j=1}^{J_i})\}_{i=1}^N$, where $\mathbf{x}_0^{(i)}, \mathbf{x}_*^{(i)} \in \mathbb{R}^p$ and $\mathbf{g}_j^{(i)} : \mathbb{R}^p \to \mathbb{R}^d$ differentiable, if there exists a function $\varphi : \mathbb{R}^d \to \mathbb{R}$ such that for each i, $\frac{1}{J_i}\sum_{j=1}^{J_i} \varphi(\mathbf{g}_j^{(i)}(\mathbf{x}^{(i)}))$ is differentiable strictly pseudoconvex with the minimum at $\mathbf{x}_*^{(i)}$, then the training error of DO with $\mathbf{h}$ from* (3.29) *strictly decreases in each iteration. Alternatively, if $\frac{1}{J_i}\sum_{j=1}^{J_i} \varphi(\mathbf{g}_j^{(i)}(\mathbf{x}^{(i)}))$ is differentiable strongly pseudoconvex with Lipschitz continuous gradient, then the training error of DO converges to zero linearly.*

[4]Note that this is $\mathbf{h}$ in the function form, not the discretized vector form. It is not possible to derive the convergence result when $\mathbf{h}$ in the discretized form is used (*e.g.*, consider when $q = \infty$ and $r = 1$). However, it may be possible to derive some bounds on the convergence as a function of q and r. We do not study such results on this thesis.

Proof. See Appendix A.4. □

Next, we present a similar result for nonsmooth convex cost functions.

Proposition 6. (Convergence of the training error with an unknown nondifferentiable convex cost function) *Given a training set* $\{(\mathbf{x}_0^{(i)}, \mathbf{x}_*^{(i)}, \{\mathbf{g}_j^{(i)}\}_{j=1}^{J_i})\}_{i=1}^{N}$*, where* $\mathbf{x}_0^{(i)}, \mathbf{x}_*^{(i)} \in \mathbb{R}^p$ *and* $\mathbf{g}_j^{(i)} : \mathbb{R}^p \to \mathbb{R}^d$ *differentiable, if there exists a function* $\varphi : \mathbb{R}^d \to \mathbb{R}$ *such that for each* i*,* $\frac{1}{J_i}\sum_{j=1}^{J_i} \varphi(\mathbf{g}_j^{(i)}(\mathbf{x}^{(i)}))$ *is strictly convex with the minimum at* $\mathbf{x}_*^{(i)}$*, then the training error of DO with* $\mathbf{h}$ *from* (3.29) *strictly decreases in each iteration. Alternatively, if* $\frac{1}{J_i}\sum_{j=1}^{J_i} \varphi(\mathbf{g}_j^{(i)}(\mathbf{x}^{(i)}))$ *is strongly convex with the minimum at* $\mathbf{x}_*^{(i)}$ *and there exist* $L > 0, H \geq 0$ *such that* $\frac{1}{J_i}\sum_{j=1}^{J_i} \bar{\varphi}(\mathbf{g}_j^{(i)}(\mathbf{x}^{(i)}))$ *is* (H, L)*-*$\mathcal{RL}(\mathbf{x}_*^{(i)})$ *for all* $i, \mathbf{x}^{(i)}$*, where* $\bar{\varphi}$ *is any induced function of* $\partial\varphi$*, then the training error of DO converges to zero.*

Proof. See Appendix A.5. □

Roughly speaking, Prop. 5 and 6 say that if there exists a penalty function φ such that for each i the global minimum of (3.8) is at $\mathbf{x}_*^{(i)}$ with no other local minima, then using (3.29) allows the training error of DO to strictly decrease or converges to zero[5]. Note that we do not need to know explicitly what such penalty function is. Also, since *monotonicity at a point* is a weaker condition than that of the subgradient of convex functions, we can say that DO's training error can reduce or converge under broader conditions than those of convexity.

4.5 Chapter summary

In this chapter, we introduce the concepts of *monotonicity at a point* and *relaxed Lipschitz at a point*. We show that, under different varieties of these conditions, the training error of DO may strictly decrease or converge to zero. These results illustrate the benefit of sequentially learning regressors, which was previously treated as heuristics.

We also explore the relation between *monotonicity at a point* and monotonicity, pseudomonotonicity, convexity, and pseudoconvexity. With this relation, we show that using the feature function $\mathbf{h}$ from Section 3.2 can lead to strict reduction or convergence to zero of DO's training error under a broader set of conditions than those of convexity. This analysis provides theoretical support for the interpretation that DO is learning to imitate gradient steps for minimizing an unknown penalty function.

[5]In general, this statement does not apply to *invex functions* [12], *i.e.*, functions where all stationary points are global minima, as they may have nonconvex sublevel sets, and *monotonicity at the minimum* may not hold (*e.g.*, the Rosenbrock function).

Chapter 5

Applications to Computer Vision

In this chapter, we demonstrate the potential of DO on three computer vision applications, namely point cloud registration (Section 5.1), camera pose estimation (Section 5.2), and image denoising (Section 5.3). For each task, we provide details on the feature function, how to parametrize the parameters into the format acceptable by the DO update rule (3.4), and how to train the update maps. We also compared DO against state-of-the-art algorithms of each task, and show that DO can often outperform them in terms of accuracy and computation time. All experiments were performed in MATLAB on a single thread on an Intel i7-4790 3.60GHz computer with 16GB memory.

5.1 Shape-specific point cloud registration

In our first application, we apply DO to the task of rigid point cloud registration (PCReg). PCReg is one of the most studied tasks in computer vision due to its large number of applications, which range from registering partial views into a complete recontruction [78], building 3D maps [82], shape tracking and pose estimation [76], to object recognition [26]. Formally, the PCReg problem can be stated as follows (see Figure 5.1): Let $\mathbf{M} \in \mathbb{R}^{3\times N_M}$ be a matrix containing 3D coordinates of one shape ('model') and $\mathbf{S} \in \mathbb{R}^{3\times N_S}$ of the second shape ('scene'), find the rotation and translation that register $\mathbf{S}$ to $\mathbf{M}$.

Various algorithms have been proposed to solve PCReg in different settings. Although there exist many classes of PCReg algorithms, including branch-and-bound-based global methods [107] and feature-based methods [112], we only focus on local methods which rely on iterative algorithms.

Local algorithms for PCReg can be divided into point-based and density-based. **Point-based:** Arguably, the most well-known algorithms in this class are Iterative Closest Point (ICP) [15] and

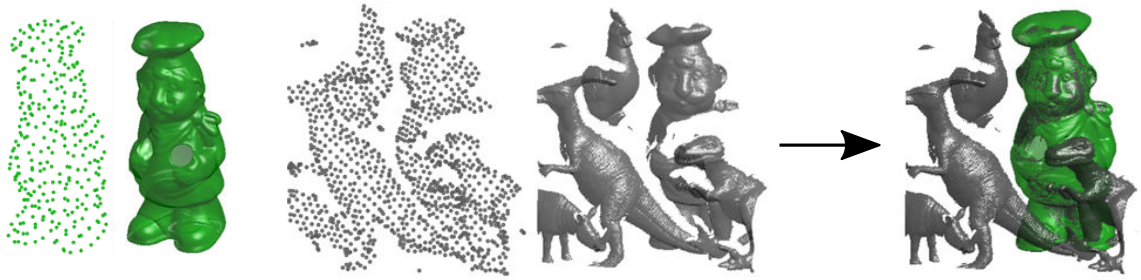


Figure 5.1: Point cloud registration problem. Given a model point cloud $\mathbf{M}$ and a scene point cloud $\mathbf{S}$, find the rotation matrix $\mathbf{R}$ and translation vector $\mathbf{t}$ that register the two point clouds. Here, the surface renderings are shown for visualization purpose.

its variants [42, 86]. These approaches alternate between solving for the correspondence and the geometric transformation until convergence. A typical drawback of ICP is the need for a good initialization to avoid a bad local minimum. To alleviate this problem, Robust Point Matching (RPM) [45] uses soft assignment instead of binary assignment. Recently, Iteratively Reweighted Least Squares (IRLS) [13] proposes using various robust cost functions to provide robustness to outliers and avoid bad local minima. **Density-based:** Density-based approaches model each point as the center of a density function. Kernel Correlation (KC) [96] aligns the densities of the two point clouds by maximizing their correlation. Coherent Point Drift (CPD) [73] assumes the point cloud of one shape is generated by the density of the other shape, and solves for the parameters that maximize their likelihood. Gaussian Mixture Model Registration (GMMReg) [55] minimizes the L_2 error between the Gaussian mixtures of the two point clouds. GMMReg also uses annealing to control the width of the Gaussian, *i.e.*, it reduces the width of the Gaussian in each iteration. More recently, [21] uses Support Vector Regression to learn a new density representation of each point cloud before minimizing L_2 error, while [46] models point clouds as particles with gravity as attractive force and solves differential equations to obtain the registration.

In summary, previous approaches tackle the registration problem by first defining different cost functions, and then solving for the optima using iterative algorithms (*e.g.*, expectation maximization and gradient descent). Our approach takes a different perspective by *not* defining new cost functions, but directly learning a sequence of updates of the rigid transformation parameters such that the stationary points match the ground truths from a training set. In the next section, we describe how to apply DO to PCReg.

5.1.1 DO parametrization and training

In this section, we first describe the parametrization of rotation and translation parameters as $\mathbf{x}$. Then, we describe the feature $\mathbf{h}$ used for the PCReg task. Note that the feature used here is not derived from the framework in Section 3.2. Then, we provide the details on how to precompute the features, which leads to a significant speed up in terms of computation time, and how we generated training data to train the update maps $\mathbf{D}_t$.

Parametrization of the transformations

Rigid transformations are usually represented in matrix form with nonlinear constraints. Since DO does not admit constraints, it is inconvenient to parametrize the transformation parameter $\mathbf{x}$ as matrices. However, it is known that the matrix representation of rigid transformation forms a Lie group, which associates with a Lie algebra [51, 67]. In essence, the Lie algebra is a linear vector space with the same dimensions as the degrees of freedom of the transformation; for instance, $\mathbb{R}^6$ is the Lie algebra of the 3D rigid transformation. Each element in the Lie algebra is associated with an element in the Lie group via exponential and logarithm maps, where closed form computations exists. Being a linear vector space, Lie algebra provides a convenient parametrization for $\mathbf{x}$ since it requires no constraints to be enforced. Note that multiple elements in the Lie algebra can represent the same transformation in the Lie group, *i.e.*, the relation is not one-to-one. However, the relation is one-to-one locally around the origin of the Lie algebra, which is sufficient for our task. Previous works that use Lie algebra include motion estimation and tracking in images [9, 97].

Features for registration

Recall that the function $\mathbf{h}$ encodes information about the problem to be solved, *e.g.*, it extracts features from the input data. For PCReg, we observe that most shapes of interest are composed of points that form a surface, and good registration occurs when the surfaces of the two shapes are aligned. To achieve such alignment, we design $\mathbf{h}$ to be a histogram that indicates the weights of scene points on the 'front' and the 'back' sides of each model point (see Figure 5.2). This allows DO to learn the parameters that update the point cloud in the direction that aligns the surfaces. Let $\mathbf{n}_a \in \mathbb{R}^3$ be a normal vector of the model point $\mathbf{m}_a$ computed from its neighboring points; $\mathcal{T}(\mathbf{y}; \mathbf{x})$ be a function that applies rigid transformation with parameter $\mathbf{x}$ to vector $\mathbf{y}$; $S_a^+ = \{\mathbf{s}_b : \mathbf{n}_a^\top(\mathcal{T}(\mathbf{s}_b; \mathbf{x}) - \mathbf{m}_a) > 0\}$ be the set of scene points on the 'front' of $\mathbf{m}_a$; and S_a^-

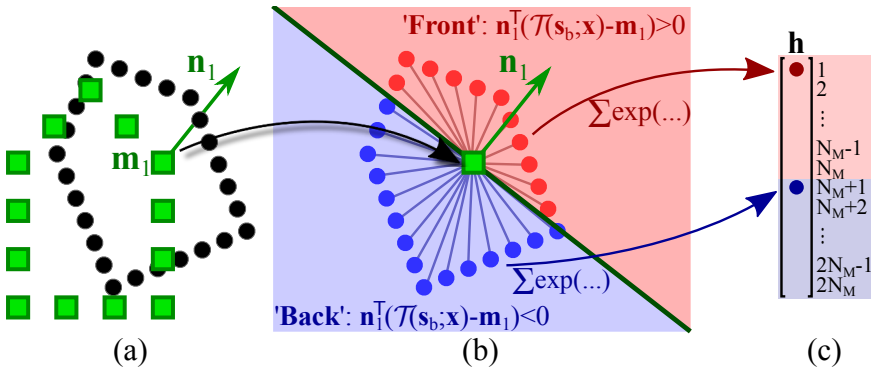


Figure 5.2: Feature $\mathbf{h}$ for point cloud registration. (a) Model points (square) and scene points (circle). (b-c) Weights of $\mathbf{s}_b$ that are on the 'front' or 'back' of model point $\mathbf{m}_1$ are assigned to different indices in $\mathbf{h}$.

contains the remaining scene points. We define $\mathbf{h} : \mathbb{R}^6 \times \mathbb{R}^{3\times N_S} \to \mathbb{R}^{2N_M}$ as:

$$[\mathbf{h}(\mathbf{x};\mathbf{S})]_a = \frac{1}{z}\sum_{\mathbf{s}_b \in S_a^+} \exp\left(-\frac{1}{\sigma^2}\|\mathcal{T}(\mathbf{s}_b;\mathbf{x}) - \mathbf{m}_a\|^2\right), \tag{5.1}$$

$$[\mathbf{h}(\mathbf{x};\mathbf{S})]_{a+N_M} = \frac{1}{z}\sum_{\mathbf{s}_b \in S_a^-} \exp\left(-\frac{1}{\sigma^2}\|\mathcal{T}(\mathbf{s}_b;\mathbf{x}) - \mathbf{m}_a\|^2\right), \tag{5.2}$$

where z normalizes $\mathbf{h}$ to sum to 1, and σ controls the width of the exp function. The exp term calculates the weight depending on the distance between the model and the scene points. The weight due to $\mathbf{s}_b$ is assigned to index a or $a + N_M$ depending on the side of $\mathbf{m}_a$ that $\mathbf{s}_b$ is on. Note that $\mathbf{h}$ is specific to a model $\mathbf{M}$, and it returns a fixed length vector of size $2N_M$. This is necessary since $\mathbf{h}$ is to be multiplied to $\mathbf{D}_t, t = 1, \ldots, T$, which are fixed size matrices with the dimension of $6 \times 2N_M$. Thus, *the SUM learned is also specific to the shape* $\mathbf{M}$. However, $\mathbf{h}$ can take the scene shape $\mathbf{S}$ with an arbitrary number of points to use with the SUM. Although we do not prove that this $\mathbf{h}$ complies with the condition in Thm. 1, we show empirically in Section 5.1.2 that it can be effectively used for our task.

Fast feature computation

Empirically, we found that computing $\mathbf{h}$ directly is slow due to pairwise distance computations and the evaluation of exponentials. To perform fast computation, we quantize the space around the model shape into uniform grids, and store the value of $\mathbf{h}$ evaluated at the center of each grid. When computing features for a scene point $\mathcal{T}(\mathbf{s}_b;\mathbf{x})$, we simply return the precomputed feature of the grid center that is closest to $\mathcal{T}(\mathbf{s}_b;\mathbf{x})$. Note that since the grid is uniform, finding the closest grid center can be done in $\mathcal{O}(1)$. To get a sense of scale in this section, we assume the

model is normalized by scaling and translating to fit in $[-1, 1]^3$. We compute the uniform grid in the range $[-2, 2]^3$ with 81 points in each dimension. We set any elements of the precomputed features that are smaller than 10^{-6} to 0. This causes most of the precomputed feature values to be zero, allowing us to store them in a sparse matrix. We found that this approach significantly reduces the feature computation time by 6 to 20 times while maintaining the same accuracy. In our experiments, the precomputed features require less than 50MB for each shape.

Training

Given a model shape $\mathbf{M}$, we first normalized the data to lie in $[-1, 1]^3$, and generated the scene models as training data by uniformly sampling with replacement 400 to 700 points from $\mathbf{M}$. Then, we applied the following perturbations: *(i) Rotation and translation:* We randomly rotated the model within 85 degrees, and added a random translation in $[-0.3, 0.3]^3$. These transformations were used as the ground truth $\mathbf{x}_*$, with $\mathbf{x}_0 = \mathbf{0}_6$ as the initialization. *(ii) Noise and outliers:* Gaussian noise with standard deviation 0.05 was added to the sample. Then we added two types of outliers: sparse outliers (random 0 to 300 points within $[-1, 1]^3$); and structured outliers (a Gaussian ball of 0 to 200 points with the standard deviation of 0.1 to 0.25). Structured outliers is used to mimic other dense object in the scene. *(iii) Incomplete shape:* We used this perturbation to simulate self occlusion and occlusion by other objects. This was done by uniformly sampling a 3D unit vector $\mathbf{u}$, then projecting all sample points to $\mathbf{u}$, and removing the points with the top 40% to 80% of the projected values. For all experiments, we generated 30000 training samples, trained a total of $T = 30$ maps for SUM with $\lambda = 3 \times 10^{-4}$ in (3.6) and $\sigma^2 = 0.03$ in (5.1) and (5.2), and set the maximum number of iterations to 1000.

5.1.2 Experiments and results

Baselines and evaluation metrics

We compared DO with two point-based approaches (ICP [15] and IRLS [13]) and two density-based approaches (CPD [73] and GMMReg [55]). The codes for all methods were downloaded from the authors' websites, except for ICP where we used MATLAB's implementation. For IRLS, the Huber cost function was used.

We used two performance metrics, which are the registration success rate and the computation time. We considered a registration to be successful when the mean ℓ_2 error between the registered model points and the corresponding model points at the ground truth orientation was less than 0.05 of the model's largest dimension.

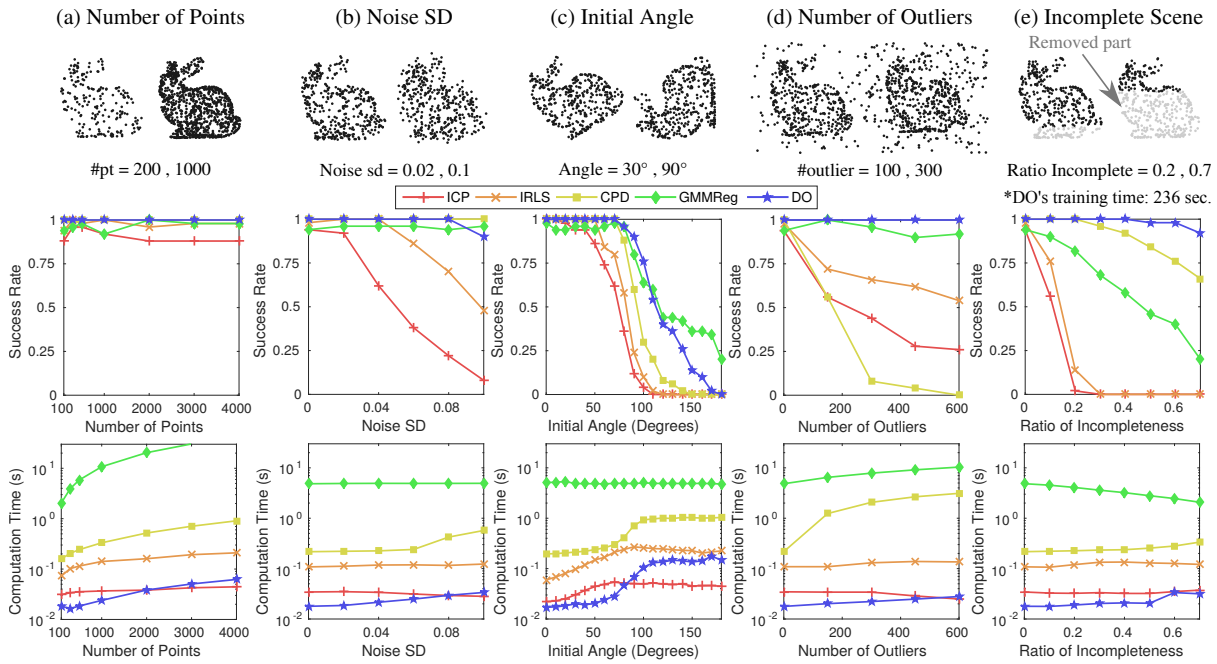


Figure 5.3: Results of 3D registration with synthetic data under different perturbations. (Top) Examples of scene points with different perturbations. (Middle) Success rate. (Bottom) Computation time.

Synthetic data

We performed synthetic experiments using the Stanford Bunny model [90] (see Figure 5.3). We used MATLAB's `pcdownsample` to select 472 points from 36k points as the model $\mathbf{M}$. We evaluated the performance of the algorithms by varying five types of perturbations: *(i)* the number of scene points ranges from 100 to 4000 [default = 200 to 600]; *(ii)* the standard deviation of the noise ranges between 0 to 0.1 [default = 0]; *(iii)* the initial angle from 0 to 180 degrees [default = 0 to 60]; *(iv)* the number of outliers from 0 to 600 [default = 0]; and *(v)* the ratio of incomplete scene shape from 0 to 0.7 [default = 0]. While we perturbed one variable, the values of the other variables were set to the default values. Note that the scene points were sampled from the original 36k points, not from $\mathbf{M}$. All generated scenes included random translation within $[-0.3, 0.3]^3$. A total of 50 rounds were run for each variable setting. Training time for DO was 236 seconds (incl. training data generation and precomputing features).

Examples of test data and the results are shown in Figure 5.3. ICP required low computation time for all cases, but it had low success rates because it tends to get trapped in the local minimum closest to its initialization. CPD generally performed well except when number of outliers was high, and it required a high computation time. IRLS was faster than CPD, but it did not perform well with incomplete targets. GMMReg had the widest basin of convergence but did not perform

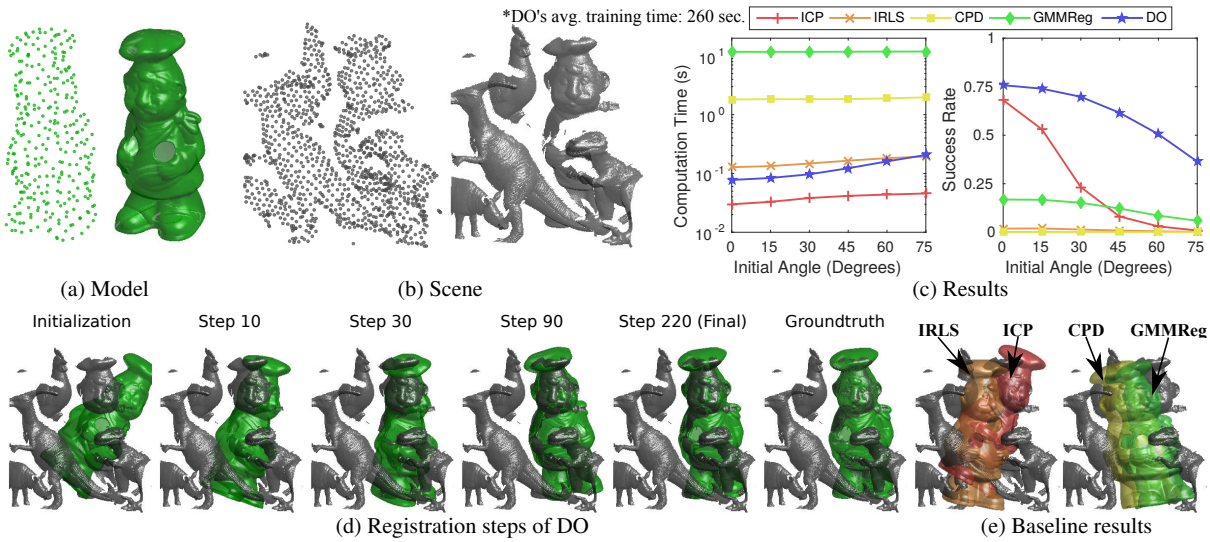
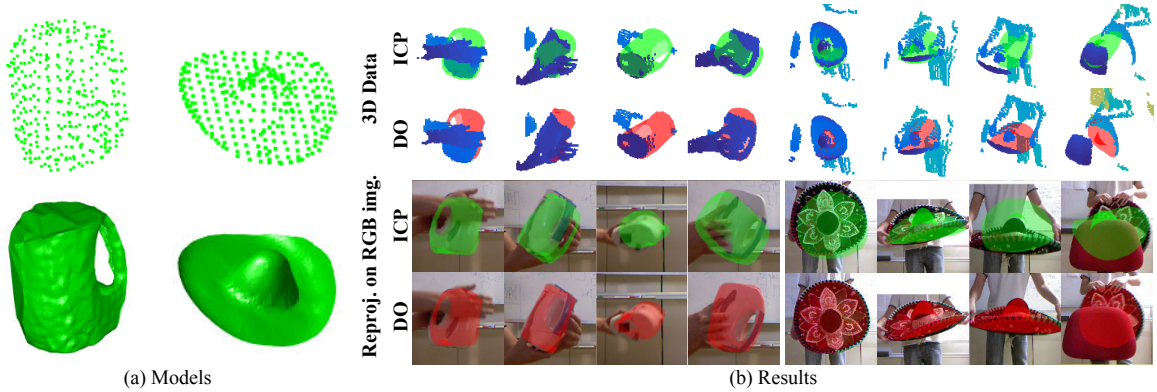


Figure 5.4: Results of 3D registration with range scan data. (a) example 3D model ('chef'). (b) Example of a 3D scene. We include surface rendering for visualization purpose. (c) Results of the experiment. (d) An example of registration steps of DO. The model was initialized 60 degrees from the ground truth orientation with parts of the model intersecting other objects. In addition, the target object is under 70% occlusion, making this a very challenging case. However, as iteration progresses, DO is able to successfully register the model. (e) Registration results of baseline algorithms.

well with incomplete targets, and it required long computation time for the annealing steps. For DO, its computation time was much lower than those of the baselines. Notice that DO required higher computation time for larger initial angles since more iterations were required to reach a stationary point. In terms of the success rate, we can see that DO outperformed the baselines in almost all test scenarios. This result was achievable because DO does not rely on any specific cost functions, which generally are modelled to handle a few types of perturbations. On the other hand, DO *learns* to cope with the perturbations from training data, allowing it to be significantly more robust than other approaches.

Range-scan data

In this section, we performed 3D registration experiment on the UWA dataset [71]. This dataset contains 50 cluttered scenes with 5 objects taken with the Minolta Vivid 910 scanner in various configurations. All objects are heavily occluded (60% to 90%). We used this dataset to test our algorithm under unseen test samples and structured outliers, as opposed to sparse outliers in the previous section. The dataset includes 188 ground truth poses for 4 objects. We performed the test using all the 4 objects on all 50 scenes. From the original model, $\sim$300 points were



(a) Models (b) Results

Figure 5.5: Result for object tracking in 3D point cloud. (a) shows the 3D models of the kettle and the hat. (b) shows tracking results of DO and ICP in (top) 3D point clouds with the scene points in blue, and (bottom) as reprojection on RGB image. Each column shows the same frame.

sampled by `pcdownsample` to use as $\mathbf{M}$ (Figure 5.4a). We also downsampled each scene to $\sim$1000 points (Figure 5.4b). We initialized the model from 0 to 75 degrees from the ground truth orientation with random translation within $[-0.4, 0.4]^3$. We ran 50 initializations for each parameter setting, resulting in a total of 50×188 rounds for each data point. Here, we set the inlier ratio of ICP to 50% as an estimate for self-occlusion. Average training time for DO was 260 seconds for each object model.

The results and examples for the registration with DO are shown in Figure 5.4c and Figure 5.4d, *resp*. IRLS, CPR, and GMMReg has very low success in almost every scene. This is because structured outliers cause many regions to have high density, creating false optima for CPD and GMMReg which are density-based approaches, and also for IRLS which is less sensitive to local minima than ICP. When initialized close to the solution, ICP could register fast and provide some correct results because it typically terminate at the nearest–and correct–local minimum. On the other hand, DO provide a significant improvement over ICP, while maintaining low computation time. We emphasize that DO was trained with synthetic examples of a single object and it had never seen other objects from the scenes. This experiment shows that we can train DO with synthetic data, and apply it to register objects in real challenging scenes.

Application to 3D object tracking

In this section, we explore the use of DO for 3D object tracking in 3D point clouds. We used Microsoft Kinect to capture RGBD videos at 20fps, then reconstruct 3D scenes from the depth images. We used two reconstructed shapes, a kettle and a hat, as the target objects. These two shapes present several challenges besides self occlusion: the kettle has a smooth surface with few features, while the hat is flat, making it hard to capture from some views. We recorded the

objects moving through different orientations, occlusions, *etc*. The depth images were subsampled to reduce computation load. To perform tracking, we manually initialized the first frame, while subsequent frames were initialized using the pose in the previous frames. Here, we only compared DO against ICP because IRLS gave similar results to those of ICP but could not track rotation well, while CPD and GMMReg failed to handle structured outliers in the scene (similar to Section 5.1.2). Figure 5.5b shows examples of the results. It can be seen that DO can robustly track and estimate the pose of the objects accurately even under heavy occlusion and structured outliers, while ICP tend to get stuck with other objects. The average computation time for DO was 40ms per frame. This shows that DO can be used as a robust real-time object tracker in 3D point cloud.

Failure case: We found DO failed to track the target object when the object was occluded at an extremely high rate, and when the object moved too fast. When this happened, DO would either track another nearby object or simply stay at the same position as in the previous frame.

5.2 Camera Pose Estimation

In this section, we apply DO to the camera pose estimation task. The goal of camera pose estimation is to estimate the relative pose between a given 3D and 2D correspondence set. Given $\{(\mathbf{p}_j, \mathbf{s}_j)\}_{j=1}^{J} \subset \mathbb{R}^2 \times \mathbb{R}^3$ where $\mathbf{p}_j$ is a 2D image coordinate and $\mathbf{s}_j$ is the corresponding 3D coordinate of feature j, we are interested in estimating the rotation matrix $\mathbf{R} \in SO(3)$ and translation vector $\mathbf{t} \in \mathbb{R}^3$, such that

$$\tilde{\mathbf{p}}_j \equiv \mathbf{K} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \tilde{\mathbf{s}}_j, j = 1, \ldots, J,$$

where tilde denotes homogeneous coordinate, $\mathbf{K} \in \mathbb{R}^{3\times3}$ is a known intrinsic camera matrix, and $\equiv$ denotes equivalence up to scale. General approaches for camera pose estimation involve solving nonlinear problems [59, 61, 62, 111]. Most of the existing approaches assume that there are no outlier matches in the correspondence set. When outliers are present, they rely on RANSAC [41] to select the inliers. One approach that does not rely on RANSAC is REPPnP [40], which finds the camera pose by solving for the robust nullspace of a matrix that represents algebraic projection error. In this section, we will use DO to find a set of inliers, then postprocess the inliers to obtain the camera pose. We show that our algorithm is more robust than REPPnP while being faster than RANSAC-based approaches when the amount of outliers is large.

5.2.1 DO parametrization and training

Residual function and DO parametrization

Assuming the camera is calibrated (*i.e.*, the intrinsic camera matrix has been factored out), we define the residual function as the following geometric error [53]:

$$\mathbf{g}_j(\mathbf{X}) = \mathbf{p}_j - \begin{bmatrix} \mathbf{x}_1^\top \tilde{\mathbf{s}}_j / \mathbf{x}_3^\top \tilde{\mathbf{s}}_j \\ \mathbf{x}_2^\top \tilde{\mathbf{s}}_j / \mathbf{x}_3^\top \tilde{\mathbf{s}}_j \end{bmatrix} = \mathbf{0}_2, j = 1, \ldots, J, \tag{5.3}$$

where $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3]^\top \in \mathbb{R}^{3\times 4}$ is the parameter to be solved. Note that we do not impose any constraints on $\mathbf{X}$ and simply use all its 12 elements as DO's parameters. This is because we will use $\mathbf{X}$ only to find the inliers, then perform postprocessing to obtain the rotation and translation.

The optimization for solving $\mathbf{X}$ is formulated by summing the error over all correspondences:

$$\underset{\mathbf{X}}{\text{minimize}} \frac{1}{J} \sum_{j=1}^{J} \varphi(\mathbf{g}_j(\mathbf{X})), \tag{5.4}$$

where φ is a penalty function. Following the derivation in Section 3.2, we can derive the following $\mathbf{h}$ function:

$$\mathbf{h}(\mathbf{X}) = \frac{1}{J} \sum_{j=1}^{J} \bigoplus_{l=1}^{12} \bigoplus_{k=1}^{2} \left[\frac{\partial \mathbf{g}_j(\mathbf{X})}{\partial vec(\mathbf{X})} \right]_{k,l} \bigotimes_{\alpha=1}^{2} \mathbf{e}_{\gamma_{q,r}([\mathbf{g}_j(\mathbf{X})]_\alpha)}. \tag{5.5}$$

After computing 5.5, we normalize it into a unit vector and use it as our feature. Note that, although the Jacobian matrix of $\mathbf{g}_j$ is a 2×12 matrix, it has only 12 degrees of freedom. Thus, we need to consider only its 12 values instead of all 24.

Training

We generated DO's training data as follows. Each image was assumed to be 640 by 480 pixels. *Generating 3D shapes*: A 3D shape, composing of 100 to 500 points, was generated as random points in one of the following shapes: *(i)* in a box; *(ii)* on a spherical surface; and *(iii)* on multiple planes. For (iii), we randomly generated normal and shift vectors for 2 to 4 planes, then added points to them. All shapes were randomly rotated, then normalized to fit in $[-1, 1]^3$. *Generating camera matrix*: We randomized the focal length in $[600, 1000]$ with the principal point at the center of the image. We sampled the rotation matrix from $SO(3)$, while the translation was generated such that the projected 3D points lie in the image boundary. *Generating image points*:

We first projected the 3D shape using the generated camera parameters, then randomly selected 0% to 80% of the image points as outliers by changing their coordinates to random locations. All random numbers were uniformly sampled. No noise was added to the training samples. To reduce the effect of varying sizes of images and 3D points, we normalized the image points to lie in $[-0.5, 0.5]^2$.[1] Since the camera matrix is homogeneous, we normalize it to have a unit Frobenius norm. We use $[-1, 1]$ as the range for each dimension of $\mathbf{g}_j$ (*i.e.*, $q = 1$), and discretize it to $r = 10$ boxes. We generated 50000 training samples, and trained 30 maps with $\lambda = 10^{-4}$. The training time was 252 seconds.

We compared 3 DO-based approaches: *DO*, *DO+P3P+RANSAC*, and *DO+RPnP*. For DO, we projected the first three columns of the output $\mathbf{X}$ to $SO(3)$ to obtain the rotation matrix. For DO+P3P+RANSAC and DO+RPnP, we used $\mathbf{X}$ to project all 3D points back to the image then selected the matches with small reprojection errors as inliers. We then used these inliers to compute the camera parameters using P3P+RANSAC [59] and RPnP [62] (without RANSAC).

5.2.2 Experiments and results

Baselines and evaluation metrics

We compared our approach against 5 baselines. EPnP [61] and REPPnP [40] are deterministic approaches. The other three baselines, P3P+RANSAC [59], RPnP+RANSAC [62], and EPnP+RANSAC [61] rely on RANSAC to select inliers and use the respective PnP algorithms to find the camera parameters. The minimum number of matches for each algorithm is 3, 4, and 6, *resp*. We use the code from [40] as the implementation of the PnP algorithms. The RANSAC routine automatically determines the number of iterations to guarantee 99% chance of obtaining the inlier set. The performance are measured in terms of *(i)* mean computation time, *(ii)* mean rotation angle error, and *(iii)* mean inlier reprojection error.

Synthetic data

We first performed experiments using synthetic data. We generated the test data using the same approach as the training samples. We varied 3 parameters: *(i)* the ratio of outliers from 0% to 90% [default = 30%]; *(ii)* the noise standard deviation from 0 to 10 pixels [default = 2]; and *(iii)* the number of points from 200 to 2000 [default = 400]. When one parameter was varied, the other two parameters were set to the default values. We performed a total of 500 trials for each setting.

[1] Camera matrix needs to be transformed accordingly, similar to [54].

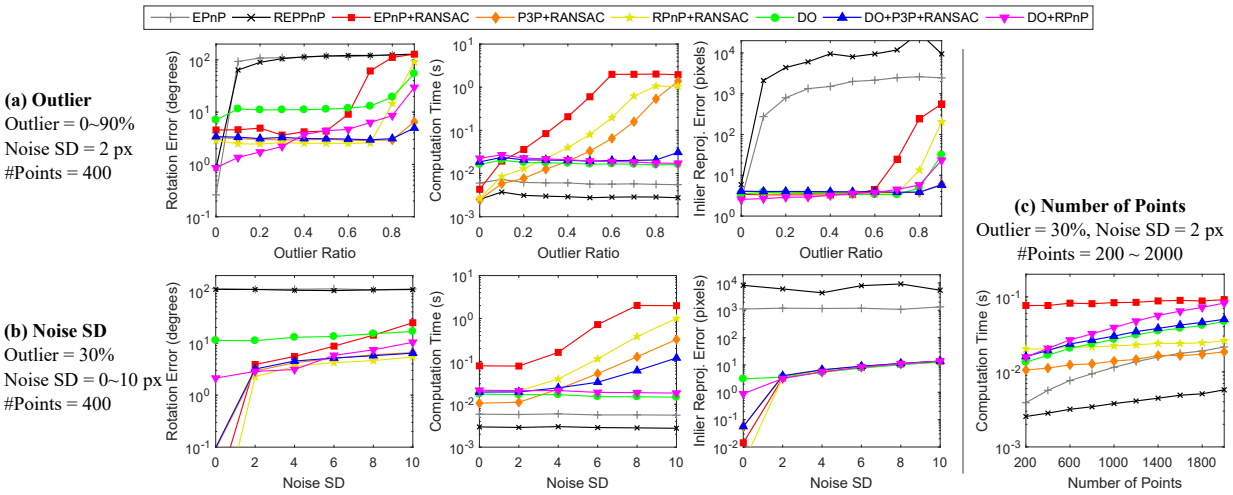
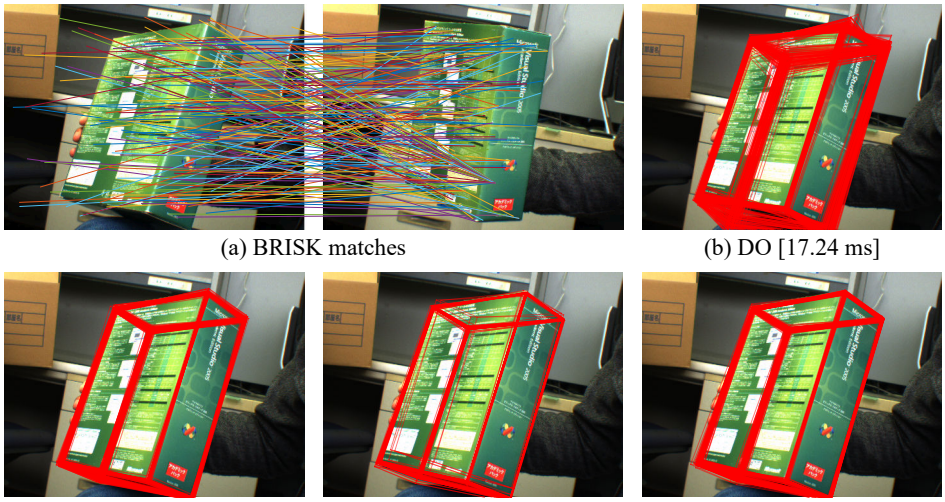


Figure 5.6: Results for PnP with synthetic data. Varying parameters are (a) outlier ratio, (b) noise SD, and (c) number of points.

Figure 5.6 shows the results of the experiments. In Figure 5.6a, we can see that RANSAC-based approaches could obtain accurate results, but their computation time grows exponentially with the outlier ratio. On the other hand, EPnP and REPPnP which are deterministic performed very fast, but they are not robust against outliers even at 10%. For our approaches, it can be seen that DO alone did not obtain good rotations since it did not enforce any geometric constraints. However, DO could accurately align the 3D inlier points to their image points as can be seen by its low inlier reprojection errors. This is a good indication that DO can be used for identifying inliers. By using this strategy, DO+P3P+RANSAC could obtain accurate rotation up to 80% of outliers while maintaining low computation time. In contrast, DO+RPnP could obtain very accurate rotation when there are small outliers, but the error increases as it was easier to mistakenly include outliers in the post-DO step. For the noise case (Figure 5.6b), DO+RPnP has constant time for all noise levels and could comparatively obtain good rotations under all noise levels, while DO+P3P+RANSAC requires exponentially increasing time as points with very high noise may be considered as outliers. Finally, in Figure 5.6c, we can see that computation times of all approaches grow with the number of points, but those of DO approaches grow with faster rate, which is a downside of our approach.

Real data

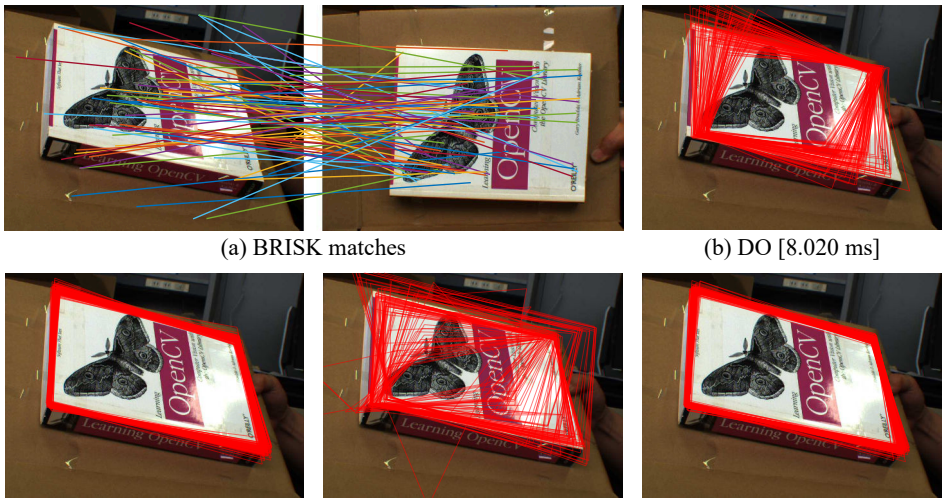
Next, we performed experiments on real images. For DO, we used the same SUM from the previous section. We only compared against P3P+RANSAC, which was the baseline that achieved the best performance in the previous section.



(a) BRISK matches (b) DO [17.24 ms]

(c) DO+P3P+RANSAC [21.64 ms] (d) DO+RPnP [18.99 ms] (e) P3P+RANSAC [159.72 ms]

Figure 5.7: Results for camera pose estimation on a real image. (a) Feature matches. Left and right images contain 2D points and projection of 3D points, *resp*. (b-d) Projected shape with average time over 100 trials.



(a) BRISK matches (b) DO [8.020 ms]

(c) DO+P3P+RANSAC [11.04 ms] (d) DO+RPnP [8.781 ms] (e) P3P+RANSAC [131.5 ms]

Figure 5.8: Results for camera pose estimation on a real image. (a) Feature matches. Left and right images contain 2D points and projection of 3D points, *resp*. (b-d) Projected shape with average time over 100 trials.

Images from [111]: First, we show the results on the images provided with the code in [111]. Figure 5.7a shows the input matches. Notice that the matches are not one-to-one. Although DO is a deterministic algorithm, different configurations of the same 3D shape can affect the result. For example, we might consider either a 3D shape or its 90° rotated shape as the initial configuration with the identity transformation. To measure this effect, we performed 100 trials for DO-based algorithms, where we randomly rotate the 3D shape as the initial configuration. Similarly, we performed 100 trials for P3P+RANSAC. 5.7b-e show the results. It can be seen that DO can gives a rough estimate of the camera pose, then DO+P3P+RANSAC and DO+RPnP can post-process to obtain accurate pose. P3P+RANSAC also obtained the correct pose, but it required 8 times the computation time of DO-based approaches. Figure 5.8 show similar results for a planar object. In Figure 5.8b, we can see that DO alone could not provide a precise estimation. This caused DO+RPnP in Figure 5.8d to be inaccurate. However, DO+P3P+RANSAC in Figure 5.8c could still recover the pose. Note that P3P+RANSAC in Figure 5.8e could also obtain good pose estimates, but, again, it required higher computational time than DO-based methods due to high outlier ratio.

Dataset of University of Oxford's Visual Geometry Group: Next, we used the dataset from University of Oxford's Visual Geometry Group[2] [101, 102] for real image experiments. The dataset contains several images with 2D points and their reconstructed 3D points, camera matrix of each image, and also reconstructed lines. We used 'Corridor' (11 images), 'Merton College I' (3 images), 'Merton College II' (3 images), 'Merton College III' (3 images), 'University Library' (3 images), and 'Wadham College' (5 images). To perform this experiment, we selected one image from each group as a reference image (for extracting feature for the 3D points), and selected another image as the target for estimating camera pose. This results in a total of 182 pairs of images. Similar to the previous real image experiment, we performed 100 trials for each pair, where in each trial we transformed the 3D points with a random rotation for DO-based methods.

Figure 5.9 shows the cumulative plots of rotation error and computation time. P3P+RANSAC obtained the best rotation accuracy since RANSAC can reliably selected the set of inliers, but it could require a long computation time to achieve this result (recall that P3P+RANSAC is the fastest of all RANSAC-based approaches). On the other hand, DO-based approaches require roughly the same amount of time for all cases. Similar to previous experiment, DO and DO+RPnP did not obtain good rotation results, while DO+P3P+RANSAC could still obtain good rotation results. To sum up this experiment, we see P3P+RANSAC obtained best rotation but it could take a long and varying computation to do so, while DO+P3P+RANSAC required roughly

[2]http://www.robots.ox.ac.uk/∼vgg/data/data-mview.html

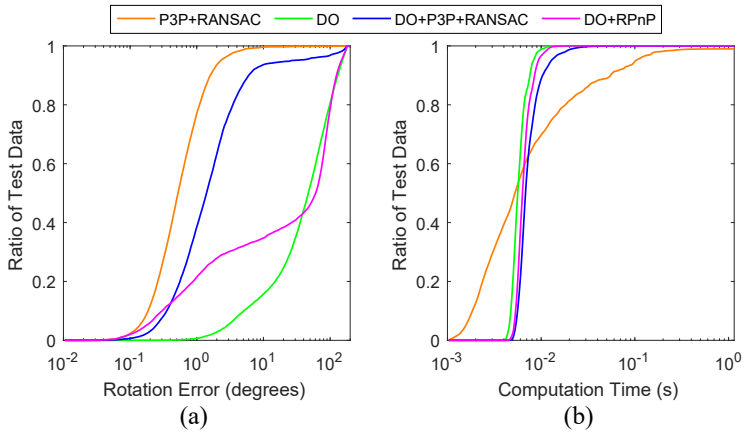


Figure 5.9: Cumulative plots on the Oxford dataset in terms of (a) rotation error and (b) computation time.

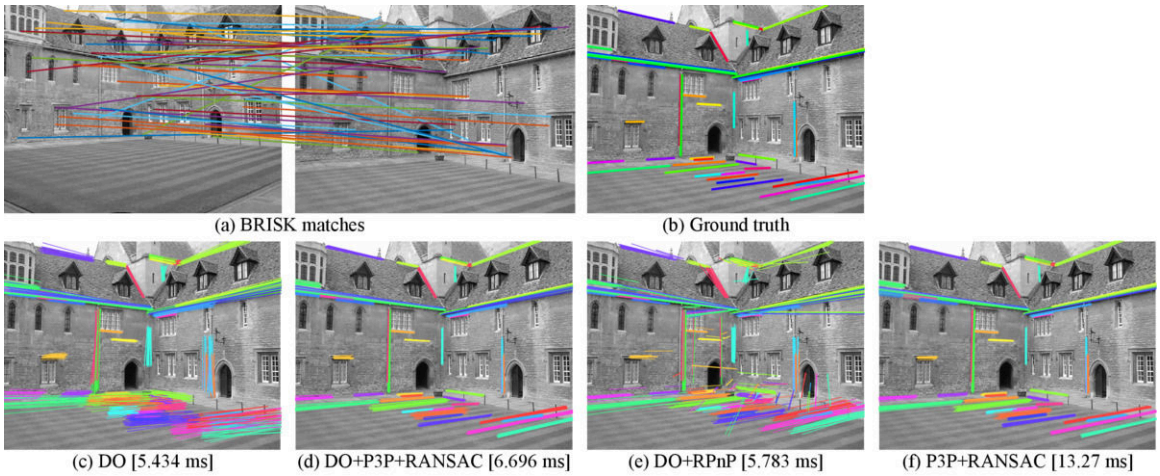


Figure 5.10: Results for camera pose estimation on real image. (a) Feature matches. Left and right images contain 2D points and projection of 3D points, *resp*. (b) Ground truth. (c-e) Projected shape with average time over 20 trials.

the same amount of time but could be less reliable than P3P+RANSAC. This poses a trade-off between the two approaches.

We provide a visualization of the estimated parameters as the projection of the reconstructed lines on the images in Figures 5.10 and 5.11. Subfigures b show the projection using the groundtruth parameters, while subfigures c to f show the projection of 20 trials using each algorithm's estimated camera parameters. In Figure 5.10, we can see that DO can provide a good initialization, which are then further refined by DO+P3P+RANSAC and DO+RPnP. Note that some refinement could worsen the result, as can be seen in Figure 5.10e of DO+RPnP. On the other hand, DO may not always provide good initialization as can be seen in Figure 5.11. Here,

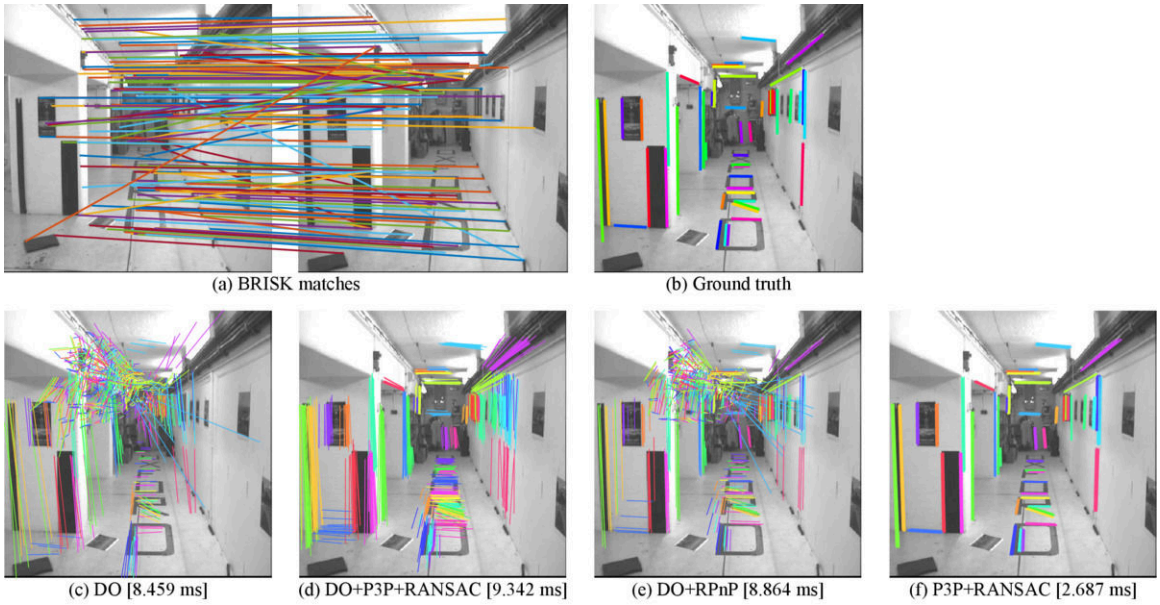


Figure 5.11: Results for camera pose estimation on real image. (a) Feature matches. Left and right images contain 2D points and projection of 3D points, *resp*. (b) Ground truth. (c-e) Projected shape with average time over 20 trials.

DO+P3P+RANSAC can recover from the failure of DO, while most of the time DO+RPnP could not. We believe the reason that DO could not provide a good initialization in this case is because the distribution of the outlier matches is different from DO's training data. This is the main limitation of DO which is a learning-based method. On the other hand, PnP solvers of RANSAC-based approaches can reliably obtain the correct pose since they could repeatedly solve the exact geometric problem. Future work for this application may incorporate real data with synthetic data in the training set so DO can learn to cope with real outlier distribution.

5.3 Image Denoising

The final application in this chapter is the task of image denoising, which can be considered as one of the most fundamental tasks in computer vision and image processing. Traditionally, image denoising is tackled using filtering operations [47], such as Gaussian filter, median filter, Wiener filter, *etc*. More recently, optimization-based method, such as total variation (TV) denoising [24], and learning-based methods, such as deep neural networks [103], have received increasing attention from researchers due to their robustness to different types of noise and outliers.

In this section, we apply DO to the image denoising tsak. This task can illustrate the potential of DO in multiple ways. First, we show that a SUM trained in a simple fashion can compare

favorably against state-of-the-art total variation (TV) denoising algorithms for impulse noises. Second, we show that a SUM can be used to estimate a large and variable number of parameters (number of pixels in this case). This differs from previous experiments that used DO to estimate a small, fixed number of parameters. Third, we show that it is simple for DO to incorporate additional information, such as pixel-wise mask, during both training and testing. Finally, we demonstrate the effect of residual distribution of the training data on the performance of DO.

5.3.1 DO parametrization and training

Residual model and DO parametrization

We based our design of $\mathbf{h}$ on the TV denoising model [24]:

$$\underset{\{x_i\}}{\text{minimize}} \sum_{i\in\Omega} \left(m_i\varphi_1(x_i - u_i) + \sum_{j\in\mathcal{N}(i)} \varphi_2(x_i - x_j) \right), \tag{5.6}$$

where we impose the unknown penalty functions φ_1 and φ_2 on the data fidelity term and the regularization term, *resp.*; Ω is the image support; $u_i \in [0, 1]$ is the intensity at pixel i of the noisy input image; $m_i \in \{0, 1\}$ is a given mask; and $\mathcal{N}(i)$ is the set of neighboring pixels of i. The goal is to estimate the clean image $\{x_i\}$.

In order to allow DO to work with images of different sizes, we will treat each pixel i of an image as if they are different problem instances.[3] In other words, the update $\Delta\mathbf{x} = \mathbf{D}\mathbf{h}(x_i)$ will be used to update only the estimate x_i of pixel i alone, and to update the whole image, we will need to compute $\Delta x_i = \mathbf{D}\mathbf{h}(x_i)$ of each pixel and update them independently. This procedure allows us to apply DO to denoise images of arbitrary sizes.

To derive the $\mathbf{h}$ function for image denoising task, note that (5.6) have two residual terms, which are the data fidelity $x_i - u_i$ and the regularization (neighbor) term $x_i - x_j$. To combine the terms, we follow Section 3.2 to obtain the feature function for each term, then simply concatenate them to form $\mathbf{h}$, resulting in

$$\mathbf{h}(x_i; u_i, \{x_j\}_{j\in\mathcal{N}(i)}) = \left[m_i\mathbf{e}_{\gamma_{q,r}(x_i-u_i)}^\top, \sum_{j\in\mathcal{N}(i)} \mathbf{e}_{\gamma_{q,r}(x_i-x_j)}^\top \right]^\top. \tag{5.7}$$

The first part of $\mathbf{h}_i$ accounts for the data fidelity term, while the second part accounts for the regularization term.

[3]The idea is similar to parameter sharing in deep neural network.

Training

In order to train DO, we randomly sample 1000 patches of size 40×40 to 80×80 pixels from the training images, then randomly replace 0% to 80% of the pixels with impulse noise to create noisy images. We trained 3 SUMs: *(i) DO-SP*, where we used salt-pepper (SP) impulse noise in $\{0, 1\}$; *(ii) DO-RV*, where we used random-value (RV) impulse noise in $[0, 1]$; and *(iii) DO-SPRV*, where 50% of the patches has RV noise, while the rest have SP noise. This is to study the effect of training data on the learned SUMs. Following [109], for images with SP noise we assume the pixels with value 0 and 1 to be corrupted by SP noise, so we set the mask $m_i = 0$ for pixels with intensity 0 and 1, and set $m_i = 1$ for others. For images with RV noise, we set $m_i = 1$ for all pixels as we cannot determine whether a pixel is corrupted by impulse noise or not. The intensity of each pixel in the noisy images is treated as initial estimate x_0, and x_* is its noise-free counterpart. We use $[-2, 2]$ (*i.e.*, $q = 2$) as the ranges for both $x_i - u_i$ and $x_i - x_j$, and discretize them to $r = 100$ boxes. We train a total of 30 maps for DO with $\lambda = 10^{-2}$. The training time took on average 367 seconds. During test, we use maximum number of iterations of 200 as the stopping criteria.

5.3.2 Experiments and results

Baseline and evaluation metrics

We compared our approach with two total variation (TV) denoising algorithms which are suitable for impulse noise. The first baseline is the convex $\ell_1 TV$ [23], which uses ℓ_1 for the data fidelity term and isotropic TV as the regularization term. The optimization is solved by the ADMM algorithm. The second baseline is $\ell_0 TV$ [109], which uses the nonconvex ℓ_0 for the data term and isotropic TV for the regularization term. The optimization is solved by the Proximal ADMM algorithm. The codes of both algorithms are provided in the toolbox of [109]. We used the same mask m_i as in the DO algorithms. We compare the results in terms of Peak Signal-to-Noise Ratio (PSNR).

Results

We downloaded 96 grayscale images of size 512×512 pixels from the Image Database[4] of University of Granada's Computer Vision Group. The first 30 images were used for training DO and selecting the best hyperparameters for the baselines for each noise type, while the remaining

[4]http://decsai.ugr.es/cvg/dbimagenes/g512.php

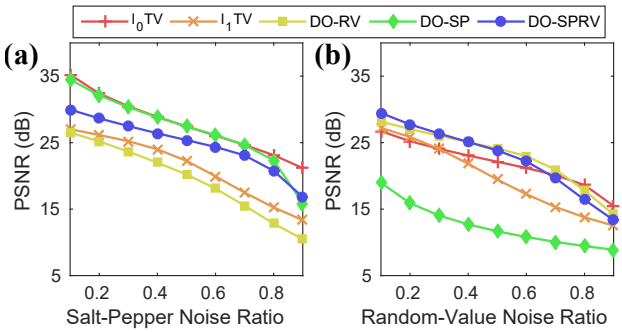
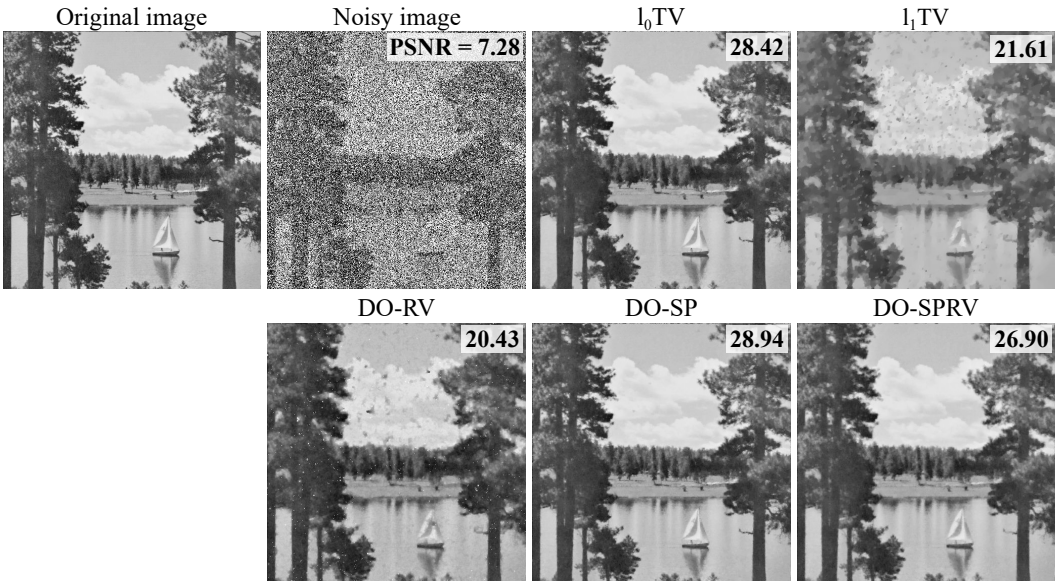


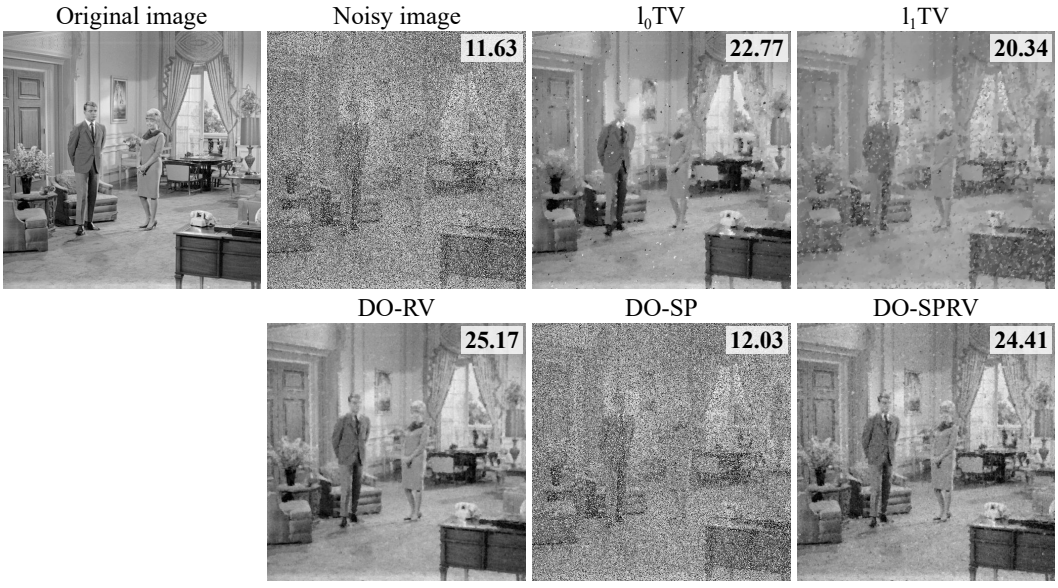
Figure 5.12: Results for image denoising for (a) salt-pepper impulse noise, and (b) random-value impulse noise.

66 images were used for evaluation. For each image, we add impulse noise of 10% to 90% to measure the algorithm robustness.

Figure 5.12 show the result PNSR over different noise ratios. It can be seen that DO trained with the correct noise type can match or outperform the state-of-the-art algorithms, while using a wrong DO give a very bad result. Interestingly, DO-SPRV which was trained with both noise performed well for both cases. Figure 5.13 shows examples of denoising results of each algorithm. For SP noise, $\ell_0 TV$, DO-SP, and DO-SPRV could recover small details, while $\ell_1 TV$ oversmoothed the image and DO-RV returned an image with smudges. For RV noise, DO-RV returned the best result. DO-SPRV also returned an acceptable image but its result still contains some intensity clumps, while DO-SP could not recover the image at all. On the other hand, both baselines oversmoothed the image (notice the persons' heads), and their results still have intensity clumps over the images. This experiment shows that DO can robustly handle different types of impulse noises, and that the training data have a strong effect on the types and the amount of noise that it can handle. The best approach for solving the problem is to select the correctly trained model. Still, training DO with both noise can return a good result, illustrating the potential of DO in solving a hard problem.



(a) 60% SP Noise



(b) 60% RV Noise

Figure 5.13: Examples of image denoising results for (a) 60% salt-pepper impulse noise and (b) 60% random-value impulse noise. The PSNR for each image is shown on the top-right.

Chapter 6

Generalizing DO

In this chapter, we consider four generalizations of the DO framework in Chapter 3. These generalizations are motivated by the following observations.

Observation 1: The summation update can be generalized to composition operations. In Chapter 3, DO updates the parameter vectors by summing the estimated parameter with an update vector. However, for many problems, other composition operations may provide a more natural way to combine two sets of parameters. For example, two rotation matrices should be composed by multiplication, not summation, to form a new rotation matrix. Generalizing DO to composition operations would allow DO to tackle problems, such as registration problems, that deal with such parameter set in a more natural manner.

Observation 2: Feature function h from Section 3.2 cannot handle high-dimensional residuals. In Section 3.2, we describe a framework of deriving task-specific feature function $\mathbf{h}$, and we have successfully applied the framework to computer vision tasks in Chapter 5. However, we notice that a major issue with the framework is that the dimension of $\mathbf{h}$ is exponential in the dimension of the residual function $\mathbf{g}_j$. For example, if RGB images are used in the denoising problem in Section 5.3, we will need to deal with 3-dimensional residuals, and the dimension of the feature will be $\mathcal{O}(r^3)$. This problem prohibits using the framework for problems with high-dimensional residuals. Even with low-dimension cases, training the update maps may require a large number of training data, depending on how fine we discretize the space.

Observation 3: Relation with first-order optimization algorithms. In Chapter 4, we analyze the relation between DO and generalized convexity, which concretely connects DO with the gradients of convex and pseudoconvex functions. In conventional optimization, gradient-based methods play a vital role in solving many classes of optimization problems, and many extensions have been proposed to expand its applicability [74], *e.g.*, acceleration and dealing with constraints. It would be beneficial to import these concepts to DO.

Inspired by these observations, we propose four generalizations of DO: *(i)* using inverse composition operation as the update rule for DO (Section 6.1); *(ii)* deriving $\mathbf{h}$ with basis functions as an alternative to discretizing the residual (Section 6.2); *(iii)* accelerating the convergence of DO with momentum (Section 6.3); and *(iv)* DO with constraints (Section 6.4).

6.1 Inverse Composition Discriminative Optimization (ICDO) for registration tasks

In Chapter 3, we see that DO updates its parameter vector $\mathbf{x}_t$ by computing the sum $\mathbf{x}_t - \Delta\mathbf{x}_t$, where $\Delta\mathbf{x}_t$ is an update vector. However, in many cases, the parameter of interest may form a group structure which can be combined with a composition operation other than summation. Examples of such parameters are the various parametrizations of rotation matrices, such as axis-angle vectors and quaternions. Summing these parameters may not result in a valid parametrization of rotation, while composing them would return an element inside the group. Here, we show a generalization of DO where we use Inverse Composition (IC) [5] operation to update the parameters instead of summing them. We call this approach Inverse Composition Discriminative Optimization (ICDO). In this section, we illustrate that ICDO can be used to solve 3D rigid point cloud registration (PCReg) task, but its formulation can also be extended to other registration tasks, such as image-based template alignment.

We begin this section by describing a general formulation for registration tasks, followed by the Forward Composition (FC) and Inverse Composition (IC) operations. Then, we describe the ICDO algorithm. Finally, we apply ICDO to the PCReg task and show that it can outperform some state-of-the-art PCReg algorithms. Additional analysis specific to the PCReg task is provided in Appendix B.

6.1.1 A general formulation for registration tasks

In this section, we describe a general formulation for registration tasks, which we will use as the foundation for FC and IC operations. Given two objects $\mathbf{I}$ and $\mathbf{J}$ (we can think of them as two images or point clouds), the goal of registration problems is to estimate the parameter $\mathbf{x}$ under the transformation function $\mathcal{T}$ that transforms $\mathbf{J}$ so that the residual between $\mathbf{I}$ and the transformed $\mathbf{J}$ is small. This can be written as an optimization problem

$$\underset{\mathbf{x}}{\text{minimize}} \sum_{a=1}^{N_I} \sum_{b=1}^{N_J} \varphi(\mathbf{I}(\mathbf{y}_{I,a}) - \mathbf{J}(\mathcal{T}(\mathbf{y}_{J,b}; \mathbf{x}))), \tag{6.1}$$

where the meaning of $\mathbf{y}_{I,a}$ and $\mathbf{y}_{J,b}$ depends on specific applications. We provide the following examples for illustration.

Example 1: Template Alignment For the grayscale template alignment problem, we can think of $\mathbf{I}$ as the template image $T : \mathbb{R}^2 \to \mathbb{R}$; $\mathbf{J}$ as the image $I : \mathbb{R}^2 \to \mathbb{R}$ to be aligned with T; and $\mathbf{y}_{I,a}$ and $\mathbf{y}_{J,b}$ are the pixel locations $\mathbf{p}$ of the images. Suppose we are working with affine transformation, $\mathbf{x}$ may denote the transformation parameters in $\mathbb{R}^6$ with the transformation $\mathcal{T}$ defined as

$$\mathcal{T}(\mathbf{p}; \mathbf{x}) = \begin{bmatrix} x_1 & x_3 \\ x_2 & x_4 \end{bmatrix} \mathbf{p} + \begin{bmatrix} x_5 \\ x_6 \end{bmatrix}. \tag{6.2}$$

Finally, if we use the ℓ_2 function as φ, then (6.1) reduces to

$$\underset{\mathbf{x}}{\text{minimize}} \sum_{i=1}^{N} (T(\mathbf{p}_i) - I(\mathcal{T}(\mathbf{p}_i; \mathbf{x})))^2, \tag{6.3}$$

which can be solved using the Lucas-Kanade algorithm [5].

Example 2: 3D Rigid Point Cloud Registration (PCReg) For PCReg, we can consider $\mathbf{y}_{I,a}$ as point a of the model shape $\mathbf{M} \in \mathbb{R}^{3 \times N_M}$; $\mathbf{y}_{J,b}$ as point b of the scene shape $\mathbf{S} \in \mathbb{R}^{3 \times N_S}$; $\mathbf{I}$ and $\mathbf{J}$ as the identity function; and $\mathbf{x}$ as the rotation and translation parameters. Suppose $\mathcal{R}$ and $\mathbf{t}$ return the rotation matrix and the translation vector from $\mathbf{x}$, then the transformation $\mathcal{T}$ can be defined as

$$\mathcal{T}(\mathbf{y}; \mathbf{x}) = \mathcal{R}(\mathbf{x})\mathbf{y} + \mathbf{t}(\mathbf{x}). \tag{6.4}$$

If we use the Gaussian function as φ, then (6.1) reduces to

$$\underset{\mathbf{x}}{\text{minimize}} \sum_{a=1}^{N_M} \sum_{b=1}^{N_S} \exp\left(-\frac{1}{\sigma^2} \|\mathbf{m}_a - \mathcal{T}(\mathbf{s}_b; \mathbf{x})\|^2 \right), \tag{6.5}$$

which is the PCReg formulation used by Kernel Correlation [96] and Gaussian Mixture Registration [55].

With a general formulation for registration tasks, we can consider the FC and IC updates which can be used to solve (6.1).

6.1.2 Forward Composition (FC) and Inverse Composition (IC)

Forward Composition (FC) and Inverse Composition (IC) [5] were proposed to solve image-based template alignment where the transformation parameters form a group structure. While FC and IC are shown to be equivalent [5], IC is the more efficient choice as it requires less com-

putation than FC. Before we describe FC and IC, let us first define the notations for composition. Let $\mathbf{x}_1 \oplus \mathbf{x}_2$ denotes the composition of two parameter vectors $\mathbf{x}_1$ and $\mathbf{x}_2$. This composition affects $\mathbf{y}$ under the transformation operator $\mathcal{T}$ as

$$\mathcal{T}(\mathbf{y}; \mathbf{x}_1 \oplus \mathbf{x}_2) = \mathcal{T}(\mathcal{T}(\mathbf{y}; \mathbf{x}_1); \mathbf{x}_2). \tag{6.6}$$

We also define $(\mathbf{x})^{-1}$ to be the inverse of $\mathbf{x}$, *i.e.*, $\mathbf{x}_1 \oplus \mathbf{x}_2 \oplus (\mathbf{x}_2)^{-1} = \mathbf{x}_1$, and let $\mathbf{0}$ parametrizes the identity transformation. With these notations, we discuss FC and IC used for solving (6.1) as follows.

Forward Composition (FC): Consider (6.1) with a differentiable φ, and let $\mathbf{x}$ and $\mathbf{x}_+$ denote the current and the next estimates, *resp*. FC operates by alternately computing the following gradient step and the update step (we disregard the step size in $\Delta\mathbf{x}$):

$$\Delta\mathbf{x} = -\sum_{a=1}^{N_I}\sum_{b=1}^{N_J}(\nabla_{\tilde{\mathbf{x}}}\mathbf{J}(\mathcal{T}(\mathbf{y}_{J,b}; \mathbf{x} \oplus \tilde{\mathbf{x}})))\nabla_{\mathbf{J}}\varphi(\mathbf{I}(\mathbf{y}_{I,a}) - \mathbf{J}(\mathcal{T}(\mathbf{y}_{J,b}; \mathbf{x} \oplus \tilde{\mathbf{x}})))\Big|_{\tilde{\mathbf{x}}=\mathbf{0}} \tag{6.7}$$

$$\mathbf{x}_+ = \mathbf{x} \oplus \Delta\mathbf{x}. \tag{6.8}$$

Inverse Composition (IC): Unlike FC which computes the gradient of (6.1), IC uses the gradient at $\tilde{\mathbf{x}} = \mathbf{0}$ from

$$\sum_{a=1}^{N_I}\sum_{b=1}^{N_J}\varphi(\mathbf{I}(\mathcal{T}(\mathbf{y}_{I,a}; \tilde{\mathbf{x}})) - \mathbf{J}(\mathcal{T}(\mathbf{y}_{J,b}; \mathbf{x}))). \tag{6.9}$$

In other words, IC uses the gradient that transforms $\mathbf{I}(\mathbf{y}_{I,a})$ to $\mathbf{J}(\mathcal{T}(\mathbf{y}_{J,b}; \mathbf{x}))$ in the update $\Delta\mathbf{x}$. Since $\Delta\mathbf{x}$ is based on the transformation of $\mathbf{I}$ instead of $\mathbf{J}$, $\Delta\mathbf{x}$ must be inversely composed with the previous estimate $\mathbf{x}$ to obtain the next estimate $\mathbf{x}_+$. This results in the following IC update steps:

$$\Delta\mathbf{x} = -\sum_{a=1}^{N_I}\sum_{b=1}^{N_J}\nabla_{\tilde{\mathbf{x}}}\mathbf{I}(\mathcal{T}(\mathbf{y}_{I,a}; \tilde{\mathbf{x}}))\nabla_{\mathbf{I}}\varphi(\mathbf{I}(\mathcal{T}(\mathbf{y}_{I,a}; \tilde{\mathbf{x}})) - \mathbf{J}(\mathcal{T}(\mathbf{y}_{J,b}; \mathbf{x})))\Big|_{\tilde{\mathbf{x}}=\mathbf{0}} \tag{6.10}$$

$$\mathbf{x}_+ = \mathbf{x} \oplus (\Delta\mathbf{x})^{-1}. \tag{6.11}$$

To see why IC requires less computation than FC, notice that IC requires computing $\nabla_{\tilde{\mathbf{x}}}\mathbf{I}(\mathcal{T}(\mathbf{y}_{I,a}; \tilde{\mathbf{x}}))$ only once as it is always evaluated at $\tilde{\mathbf{x}} = \mathbf{0}$. On the other hand, FC requires recomputing $\nabla_{\tilde{\mathbf{x}}}\mathbf{J}(\mathcal{T}(\mathbf{y}_{J,b}; \mathbf{x}))$ every iteration as it depends on the value of $\mathbf{x}$, leading to more computation than IC [5].

In the next section, we will describe how to apply the IC update rule to DO. However, unlike previous works where $\Delta\mathbf{x}$ is computed from the gradient of φ, ICDO learns to compute the IC update $\Delta\mathbf{x}$ from a set of training data.

6.1.3 Learning IC update with DO

In order to learn the update step under the IC update rule, we modify the DO framework from Chapter 3 which uses the summation update rule. Here, we first describe the update rule, followed by the feature function, how to learn the maps, and how to apply them to solve registration problems.

Update rule: Given an initialization $\mathbf{x}_0 = \mathbf{0}$ and a function $\mathbf{h}$ that extracts features from the objects, ICDO updates the estimated parameter at step t using the IC operation

$$\mathbf{x}_t = \mathbf{x}_{t-1} \oplus (\mathbf{D}_t \mathbf{h}(\mathbf{x}_{t-1}; \mathbf{I}, \mathbf{J}))^{-1}, \tag{6.12}$$

where $\mathbf{D}_t, t = 1, 2, \ldots,$ are matrices that map the feature function $\mathbf{h}(\mathbf{x}_{t-1}; \mathbf{I}, \mathbf{J})$ to an update vector $\Delta\mathbf{x}$.

Feature function $\mathbf{h}$: We apply the same framework from Section 3.2 to derive the feature function $\mathbf{h}$. Suppose we have $\mathbf{x} \in \mathbb{R}^p, \mathbf{y}_{I,a}, \mathbf{y}_{J,b} \in \mathbb{R}^z$ and $\mathbf{I}, \mathbf{J} : \mathbb{R}^z \to \mathbb{R}^d$. Taking the gradient of (6.9) as the base for derivation, we can derive $\mathbf{h}$ as

$$\mathbf{h}(\mathbf{x}_{t-1}; \mathbf{I}, \mathbf{J}) = -\sum_{a=1}^{N_I} \sum_{b=1}^{N_J} \bigoplus_{l=1}^{p} \bigoplus_{k=1}^{d} \left[\nabla_{\tilde{\mathbf{x}}} \mathbf{I}(\mathcal{T}(\mathbf{y}_{I,a}; \tilde{\mathbf{x}}))\right]_{k,l}\Big|_{\tilde{\mathbf{x}}=\mathbf{0}_p} \bigotimes_{\alpha=1}^{d} \mathbf{e}_{\gamma_{q,r}([\mathbf{I}(\mathbf{y}_{I,a}) - \mathbf{J}(\mathcal{T}(\mathbf{y}_{J,b}; \mathbf{x}))]_\alpha)}, \tag{6.13}$$

where $\bigotimes$ is the Kronecker product, and $\bigoplus$ denotes vector concatenation operation. Recall from (3.16) that $\gamma_{q,r}$ is the function that discretizes $[-q, q]$ to an integer in $\{0, \ldots, r\}$ and returns zero otherwise. Similar to the $\mathbf{h}$ in Section 3.2, the dimension of $\mathbf{h}$ above is pdr^d.

Learning update maps: Suppose we are given a training set $\{(\mathbf{x}_*^{(i)}, \mathbf{I}^{(i)}, \mathbf{J}^{(i)})\}_{i=1}^N$, where $\mathbf{x}_*^{(i)}$ is the ground truth registration parameter satisfying $\mathcal{T}(\mathbf{J}^{(i)}; \mathbf{x}_*^{(i)}) \sim \mathbf{I}^{(i)}$. Similar to Section 3.1, we use regularized linear least-squares minimization to learn the maps, but use the composition operation to compute the difference between $\mathbf{x}_*^{(i)}$ and $\mathbf{x}_{t-1}^{(i)}$ instead of the minus operation:

$$\mathbf{D}_t = \arg\min_{\tilde{\mathbf{D}}} \frac{1}{N} \sum_{i=1}^{N} \|((\mathbf{x}_*^{(i)})^{-1} \oplus \mathbf{x}_{t-1}^{(i)}) - \tilde{\mathbf{D}} \mathbf{h}(\mathbf{x}_{t-1}^{(i)}; \mathbf{I}^{(i)}, \mathbf{J}^{(i)})\|_2^2 + \lambda \|\tilde{\mathbf{D}}\|_F^2. \tag{6.14}$$

After a map is learned, we update the training instances using the update rule (6.12). We repeat this process until a terminating criteria is reached, *e.g.*, a maximum number of maps. Alg. 3

Algorithm 3 Training ICDO

Require: $\{(\mathbf{x}_*^{(i)}, \mathbf{I}^{(i)}, \mathbf{J}^{(i)})\}_{i=1}^N, T, \lambda, q_0, r, \alpha$
Ensure: $\{\mathbf{D}_t\}_{t=1}^T$
1: Initialize $\mathbf{x}_0^{(i)} := \mathbf{0}, \forall i$; and $q := q_0$
2: **for** $t = 1$ **to** T **do**
3: Compute $\tilde{\mathbf{h}}^{(i)} = \mathbf{h}(\mathbf{x}_{t-1}^{(i)}; \mathbf{I}^{(i)}, \mathbf{J}^{(i)}), \forall i$ from (6.13)
4: Compute $\mathbf{D}_t$ with (6.14)
5: Compute $\mathbf{x}_t^{(i)} := \mathbf{x}_{t-1}^{(i)} \oplus (\mathbf{D}_t \tilde{\mathbf{h}}^{(i)})^{-1}, \forall i$
6: Compute $q := q_0/\alpha^t$
7: **end for**

Algorithm 4 Solving registration with ICDO

Require: $\mathbf{I}, \mathbf{J}, \{\mathbf{D}_t\}_{t=1}^T, q_0, r, \alpha$
Ensure: $\mathbf{x}$
1: Initialize $\mathbf{x} := \mathbf{0}$; $t := 1$; and $q := q_0$
2: **while** not converge **do**
3: Compute $\tilde{\mathbf{h}} := \mathbf{h}(\mathbf{x}; \mathbf{I}, \mathbf{J})$ with (6.13)
4: Compute $\Delta\mathbf{x} := \mathbf{D}_{\min(t,T)}\tilde{\mathbf{h}}$
5: **if** $t > T$ **then**
6: Compute $\Delta\mathbf{x} := (\Delta\mathbf{x} + \Delta\mathbf{x}^-)/2$
7: **end if**
8: Compute $\mathbf{x} := \mathbf{x} \oplus (\Delta\mathbf{x})^{-1}$
9: Compute $\Delta\mathbf{x}^- := \Delta\mathbf{x}$
10: Compute $q := q_0/\alpha^t$
11: Compute $t := t + 1$
12: **end while**

shows a pseudocode for training ICDO.

Solving registration: The pseudocode for solving registration with ICDO is summarized in Alg. 4. Suppose we trained a total of T maps. We first perform the update using (6.12) until step T, then we continue using $\mathbf{D}_T$ to update until a termination criteria is reached, *e.g.*, the update is small. However, in the point cloud registration experiment in the following section, we observe that using $\mathbf{D}_T$ to update beyond iteration T causes the parameter to bounce around the correct solution without converging to it. This behavior resembles subgradient method with constant step size [18], which may not converge to a minimum. To alleviate this issue, we attempted to scale the update with $1/(t-T)$ and $1/\sqrt{t-T}$ for $t > T$ but we found that the updates diminished too fast, leading to a premature termination. The strategy that we found effective is to use $\Delta\mathbf{x}$ from the average of the updates from the current and the previous iterations (line 6 in Alg. 4). This strategy resembles the momentum approach [74] used frequently in first-order optimization.

6.1.4 Application: Shape-independent point cloud registration

We demonstrate the potential of ICDO on the problem of rigid 3D point cloud registration (PCReg). Unlike the PCReg approach in Section 5.1 which is shape-specific, here we develop an algorithm that is shape-independent and show that the learned maps generalize even to shapes which are not included in the training data. Recall that the goal of PCReg is to find the rigid transformation parameters $\mathbf{x}$ that aligns a scene shape $\mathbf{S} \in \mathbb{R}^{3\times N_S}$ to a model shape $\mathbf{M} \in \mathbb{R}^{3\times N_M}$. We first describe our parametrization, the feature function, and our training method. Then, we perform experiments on synthetic and real data to test the performance of ICDO against other state-of-the-art PCReg algorithms. Additional analysis for this task is provided in Appendix B.

DO parametrization and training

Transformation parametrization: We parametrize $\mathbf{x} = [\mathbf{r}^\top, \mathbf{t}^\top]^\top \in \mathbb{R}^6$, where $\mathbf{r} \in \mathbb{R}^3$ is an axis-angle vector parametrizing rotation and $\mathbf{t} \in \mathbb{R}^3$ is the translation vector. The composition between $\mathbf{x}_1 = [\mathbf{r}_1^\top, \mathbf{t}_1^\top]^\top$ and $\mathbf{x}_2 = [\mathbf{r}_2^\top, \mathbf{t}_2^\top]^\top$ is defined as

$$\mathbf{x}_1 \oplus \mathbf{x}_2 = \begin{bmatrix} \mathcal{R}^{-1}(\mathcal{R}(\mathbf{r}_2)\mathcal{R}(\mathbf{r}_1)) \\ \mathcal{R}(\mathbf{r}_2)\mathbf{t}_1 + \mathbf{t}_2 \end{bmatrix}, \tag{6.15}$$

where $\mathcal{R} : \mathbb{R}^3 \to SO(3)$ converts an axis-angle vector to its rotation matrix representation and $\mathcal{R}^{-1} : SO(3) \to \mathbb{R}^3$ reverts a rotation matrix back to its axis-angle vector.

Feature function $\mathbf{h}$: While it is possible to apply (6.13) as the $\mathbf{h}$ function for PCReg, a major problem is that its dimension is exponential in the dimension of the point cloud. In the 3D case, the dimension of $\mathbf{h}$ will be $\mathcal{O}(r^3)$, which can be too large to store and train efficiently. In order to make the learning more tractable, we look at an alternative formulation for PCReg:

$$\underset{\tilde{\mathbf{x}}}{\text{minimize}}\ J(\tilde{\mathbf{x}}) = \sum_{a=1}^{N_M}\sum_{b=1}^{N_S} \varphi(\|\mathcal{T}(\mathbf{m}_a;\tilde{\mathbf{x}}) - \mathcal{T}(\mathbf{s}_b;\mathbf{x})\|). \tag{6.16}$$

The cost function in (6.16) differs from (6.9) in that (6.9) uses the residual–a 3D function–as the argument of φ, while (6.16) uses the norm of the residual, which is a 1D function. This allows us derive a feature function $\mathbf{h}$ with has much smaller dimension than that in (6.13). Define $\mathbf{g}_{a,b}(\tilde{\mathbf{x}};\mathbf{x}) = \mathcal{T}(\mathbf{m}_a;\tilde{\mathbf{x}}) - \mathcal{T}(\mathbf{s}_b;\mathbf{x})$, we take the (negative) derivative of the above cost function as

$$\Delta\tilde{\mathbf{x}} = -\frac{\partial}{\partial\tilde{\mathbf{x}}}J(\tilde{\mathbf{x}}) = -\sum_{a=1}^{N_M}\sum_{b=1}^{N_S} \underbrace{\begin{bmatrix} -[\mathbf{m}]_\times \\ \mathbf{I}_3 \end{bmatrix} \frac{\mathbf{g}_{a,b}(\tilde{\mathbf{x}};\mathbf{x})}{\|\mathbf{g}_{a,b}(\tilde{\mathbf{x}};\mathbf{x})\|}}_{=\mathbf{w}_{a,b}} \frac{\partial\varphi(\|\mathbf{g}_{a,b}(\tilde{\mathbf{x}};\mathbf{x})\|)}{\partial\|\mathbf{g}_{a,b}(\tilde{\mathbf{x}};\mathbf{x})\|}, \tag{6.17}$$

where $\mathbf{I}_3$ is a 3×3 identity matrix. Then, following the feature derivation similar to the approach in Section 3.2 (full detatil provided in Appendix B), we arrive at the matrix $\mathbf{D}$ and the feature $\mathbf{h}$:

$$\mathbf{D} = \mathbf{I}_6 \otimes \boldsymbol{\phi}^\top \tag{6.18}$$

$$\mathbf{h}(\mathbf{x}; \mathbf{M}, \mathbf{S}) = -\sum_{a=1}^{N_M} \sum_{b=1}^{N_S} \bigoplus_{l=1}^{6} [\mathbf{w}_{a,b}]_l \mathbf{e}_{\gamma_{q,r}\left(\|\mathbf{g}_{a,b}(\mathbf{0}_6; \mathbf{x})\|\right)}. \tag{6.19}$$

Here, $\gamma_{q,r}$ only discretizes the nonnegative segment $[0, q]$ into r boxes because $\|\mathbf{g}_{a,b}(\mathbf{0}_6; \mathbf{x})\|$ cannot be negative (this differs from $\gamma_{q,r}$ in (3.16) which discretizes $[-q, q]$). We can see that $\mathbf{h}$ only has the dimension of $6r$, which is much smaller than $\mathcal{O}(r^3)$ of (6.13).

Training ICDO for PCReg: We generated synthetic data to train the maps from seven 3D shapes: Bunny and armadillo from Stanford's 3D scan repository [32] and all 5 shapes from the UWA dataset [71]. Each training model $\mathbf{M}^{(i)}$ was generated by randomly picking a shape; scaling it so that all points are in $[-1, 1]^3$; and randomly rotating in $[0, 180]$ degrees. Next, we copied the model as the scene shape $\mathbf{S}^{(i)}$, then added a random rotation in $[0, 85]$ degrees and translation in $[-0.2, 0.2]^3$ to only $\mathbf{S}^{(i)}$. Then, we applied the following modifications to $\mathbf{M}^{(i)}$ and $\mathbf{S}^{(i)}$ independently: Randomly sampling $200 - 400$ points; adding Gaussian noise with σ in $[0, 0.03]$; and mimicking incomplete shape by randomly sampling a 3D vector $\mathbf{u}$, then removing the points where their dot product with $\mathbf{u}$ are in the top $0 - 30\%$ (this is done only either $\mathbf{M}$ or $\mathbf{S}$ but not both). No outliers were added for the training data as we found this degraded the results. We found $\lambda = 10^{-8}$, $q = 3$, $r = 100$, $\alpha = 1.15$, and $\mathcal{T} = 20$ work well across all experiments. We used a total of 10^5 training samples, and ICDO took 96 minutes to train.

Implementation details: The following section describes implementation details on normalization, how to speed up computation, training procedure, and termination criteria.

Normalization: PCReg algorithms are generally sensitive to variations in the point clouds, *e.g.*, density and scale. These issues are further complicated by the fact that ICDO is learning-based, thus normalization is very important. First, we remove the mean of $\mathbf{M}$ from both $\mathbf{M}$ and $\mathbf{S}$ to maintain their relative configuration. Next, we perform two normalizations for scale and density. *(i) Scale*: Suppose that we have the registration $\mathbf{RS} + \mathbf{t} \sim \mathbf{M}$. If the shapes are scaled by ρ, *e.g.*, $\hat{\mathbf{M}} = \rho\mathbf{M}$, we will have $\mathbf{R}\hat{\mathbf{S}} + \rho\mathbf{t} \sim \hat{\mathbf{M}}$: only the translation vector is scaled but not the rotation, making it harder to learn effectively. To prevent this effect, we scale both $\mathbf{M}$ and $\mathbf{S}$ by $\sqrt{N_M}/\eta$, where η is the mean of $\mathbf{M}$'s singular values. *(ii) Density*: Consider two pairs of point clouds $\theta^{(1)} = (\mathbf{M}, \mathbf{S})$ and $\theta^{(2)} = ([\mathbf{M}, \mathbf{M}], \mathbf{S})$. We can see that $\theta^{(2)}$'s model density is doubled that of $\theta^{(1)}$. This causes an undesirable effect that $\mathbf{h}(\mathbf{x}, \theta^{(2)}) = 2\mathbf{h}(\mathbf{x}, \theta^{(1)})$, meaning the update step of $\theta^{(2)}$ will be double that of $\theta^{(1)}$, while the shapes are the same. To handle this issue, we divide $\mathbf{h}$

in (6.19) by $N_M N_S$.

Speeding up computation: We found that the most time-consuming step is the aggregation of $[\mathbf{w}_{ij}]_l$ into $\mathbf{h}$ in (6.19). To reduce computation, we reduce the number of terms in (6.19) by reducing the value of q in each iteration. Since we keep r constant, an additional advantage of this reduction is that the discretized boxes become finer as iteration increases, allowing more details to be captured. Note that while this reduction speeds up computation, it does not change ICDO's complexity.

Training: We found that the training error in Alg. 3 reduces too fast, which causes the latter maps to have small updates. To handle this issue, we add random rotation in $\mathcal{N}(0, 10)$ degrees and translation vector with the norm in $\mathcal{N}(0, 0.1)$ to the data in each training iteration, and adjust the ground truth $\mathbf{x}_*^k$ accordingly. In addition, notice that $\mathbf{D}$ in (6.18) is block-diagonal with nonzero values only in the elements of $\boldsymbol{\varphi}$. In practice, we also found that the off-block-diagonal elements have very small values. With these observations, we constrain all elements outside the diagonal blocks to be zero when we learn the maps in (6.14).

Termination criteria: We terminate the algorithm when the rotation and the total displacement in the past 5 iterations amount to less than 0.5 degrees and 3×10^{-3}, *resp*. We also terminate if the number of iterations reaches 200.

Experiments and results

Baselines and evaluation metrics: We use the same 4 baselines from Section 5.1.2: *(i)* ICP [15]. *(ii)* IRLS [13], which is similar to ICP but uses the Huber function as penalty. *(iii)* CPD [73], which maximizes the likelihood that one point cloud is generated by the Gaussian mixture of the other. Its optimization is performed using Expectation-Maximization algorithm. *(iv)* GMMReg [55], which minimizes L_2 distance between the two Gaussian mixtures. Note that GMMReg is a gradient-base method. This makes GMMReg most similar to ICDO. We obtained the MATLAB codes from the authors' websites, except ICP which we used MATLAB's implementation. Note that CPD and GMMReg use C implementation of fast Gauss transform (FGT) [49] to accelerate the computation of Gaussian kernels. Here, we do not compare with DO from Section 5.1 as it is for shape-specific task and requires 3-4 minutes to train each shape, thus it is impractical for this experiment.

To measure performance, we use the *registration error*, defined as the pointwise RMSE of the model in the ground truth pose and the model in the estimated pose:

$$(1/\sqrt{N_M})\|T(\mathbf{M}; \mathbf{x}) - T(\mathbf{M}; \mathbf{x}_{gt})\|_F, \tag{6.20}$$

where $\|\cdot\|_F$ is the Frobenius norm, and $\mathbf{x}$ and $\mathbf{x}_{gt}$ are the estimated and the ground truth poses, *resp*. We also report computation time for each algorithm.

Synthetic data: We use 7 shapes (cat, centaur, dog, gorilla, gun, horse, and wolf) from TUM 3D object in clutter dataset [84] for testing. These shapes were selected so that they did not overlap with those in training. The initial shapes were normalized to lie in $[-1, 1]^3$. Similar to Section 5.1.2, we tested 5 modifications: *(i)* Number of points from 100 to 2000 [default = 200 to 400]; *(ii)* Initial angle from 0° to 180° [default = 0° to 60°]; *(iii)* Noise SD from 0 to 0.1 [default = 0 to 0.03]; *(iv)* Outlier ratio against the number of inliers from 0 to 2 [default = 0]; *(v)* Incomplete shape from 0 to 0.9 [default = 0] (generated the same way as in training). All tests included random translation in $[0, 0.3]^3$. Outlier points were randomly generated in $[-1.25, 1.25]^3$. While one parameter was varied, other parameters were set to the default values. For each setting, we tested 500 pairs of point clouds sampled from the 7 shapes. Unlike in training, the model and scene points were independently sampled. Here, we consider a registration successful if the registration error is less than 0.15.

Figure 6.1 shows the results of the synthetic experiment. We can see that ICDO is comparable to the state-of-the-art algorithms: It performed almost perfectly under varying number of points, noisy data, and outlier ratios. ICDO has less success than CPD and GMMReg for large initial angles, while being more successful than ICP and IRLS. GMMReg even has some success with 180° initial angle because its annealing can smooth the shapes enough to avoid bad optima. However, a downside is GMMReg can also oversmooth, leading to some failure even with 0° initial angle. In contrast, ICDO has more success with lower angles and less success with high angles. We also found that ICDO works well with outliers even it was not trained with them. This is because ICDO can be interpreted as learning an annealing schedule (see Section B.3), so outliers have little effect on its performance. This behavior is similar to GMMReg which requires setting the Gaussian widths as an annealing schedule prior to running the algorithm, *i.e.*, the outliers in the data have no effect on the widths. In contrast, CPD estimates the Gaussian width as the algorithm is run, thus outliers can thwart this estimation. Also, outliers create more local minima for ICP and IRLS which use closest matches, leading to bad registration. Under similar reasons, ICDO and GMMReg are the most robust to incomplete shapes. In terms of computation time, ICDO is generally slightly slower than IRLS and CPD while being much faster then GMMReg (Note that CPD and GMMReg use C code for FGT while ICDO is completely written in MATLAB, so their computation times are not directly comparable). This experiment demonstrates that ICDO can be trained and tested on different sets of shapes, while being able to obtain competitive success and time as the state-of-the-art algorithms.

Real data: We perform experiments on two real datasets to evaluate ICDO. *(i)* Stanford's

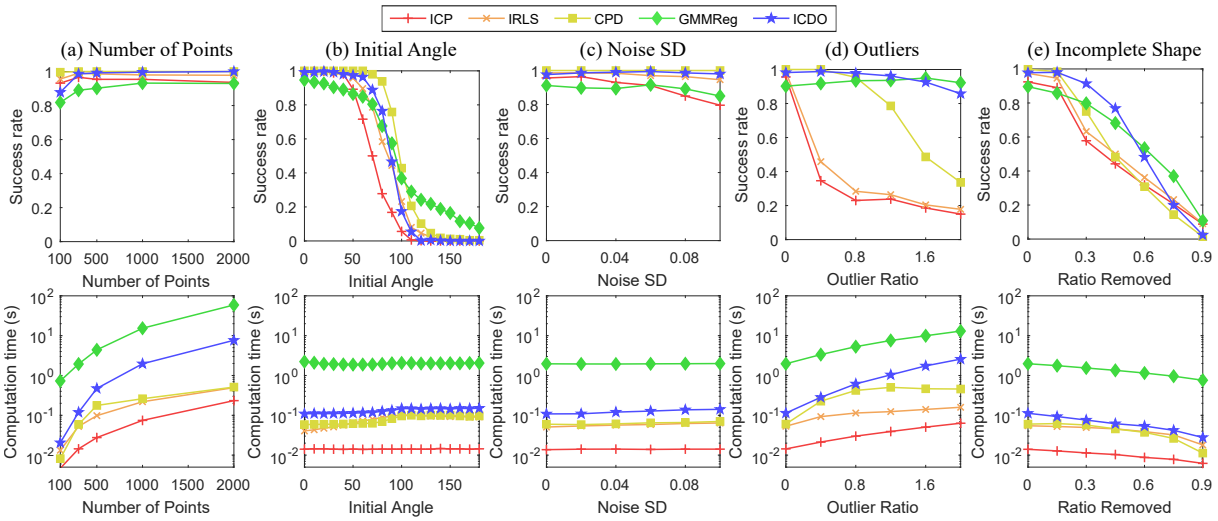


Figure 6.1: Results for synthetic data experiment over different modifications. (Top) Success rate. (Bottom) Computation time.

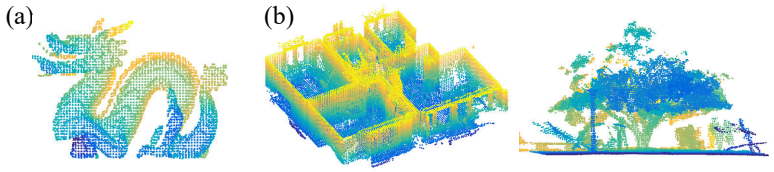


Figure 6.2: Real data examples (modified for visualization). (a) Stanford's dragon. (b) ETH laser registration dataset (Apartment and Gazebo Summer).

dragon [32] and *(ii)* ETH laser registration dataset [81]. Figure 6.2 shows examples from the two datasets. We provide the details and results below.

Stanford's dragon [32]: This dataset contains 15 scans at every 24° of a dragon statue. Following [21, 55], we merge scans at $\pm 24^\circ, \pm 48^\circ, \pm 72^\circ, \pm 96^\circ$, with a total of 30 pairs for each angle. A registration is successful if $\mathbf{q}^\top \mathbf{q}_{gt} > 0.99$ where $\mathbf{q}$ and $\mathbf{q}_{gt}$ are the estimated and the ground truth unit quaternions, *resp*. Each point cloud was downsampled to 2000 points. The result is presented in Table 6.1. The results of the baselines were taken from SVR paper [21], which improves GMMReg by learning the weight of each Gaussian. We can see that ICDO is second to SVR while outperforming ICP, CPD, and GMMReg, illustrating the robustness of our approach against methods which consider all point as having equal weights. Our implementation took 7.7 seconds to register each pair on average.

ETH laser registration dataset [81]: This dataset consists of 3D laser scans from 8 outdoor and indoor environments. Each environment has 31 to 45 scans (total 275), and contains dynamic objects such as people and furniture displacement, which can be considered as outliers. The scans were recorded sequentially as the scanner traversed the environments. In this experiment,

Table 6.1: Successful registration on Stanford's dragon.

Pose	ICP	CPD	GMMReg	SVR	ICDO
$\pm 24°$	28	26	29	**30**	**30**
$\pm 48°$	19	18	20	**29**	26
$\pm 72°$	13	14	13	**16**	15
$\pm 96°$	1	3	2	**4**	0

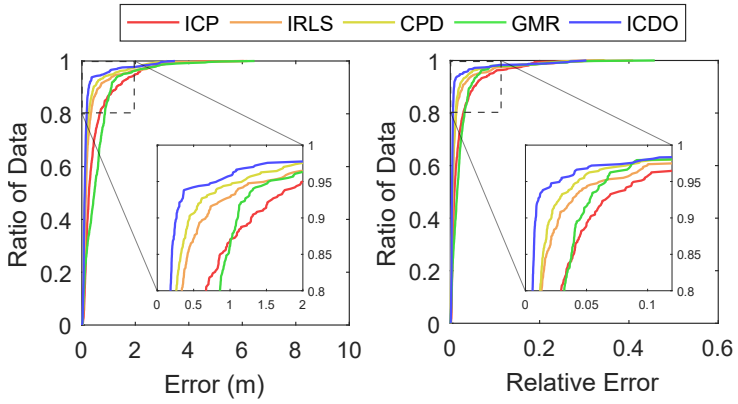


Figure 6.3: Results of ETH laser registration dataset in cumulative plots. (Left) Absolute registration error. (Right) Relative error.

we merge consecutive scans in both forward and backward directions (total 534 pairs). We preprocessed each point cloud by using a box grid filter (MATLAB's `pcdownsample`) at 10cm interval to make the density more uniform, then subsampled to 1000 points.

Figure 6.3 shows the cumulative error plots in terms of the absolute registration error (in meters) and the relative registration error. The latter is defined as the registration error divided by the largest distance between any two model points. We can see that ICDO achieved the best result in both measures. Recall that ICDO was trained with synthetic data synthesized from 7 shapes, which have no similarity to the data in this section. This demonstrates the potential of ICDO as a robust learning-based PCReg algorithm which can generalize to different classes of objects. In terms of the average computation time, we have ICP at 0.06s, IRLS at 0.35s, CPD at 1.62s, GMMReg at 18.66s, and ICDO at 2.14s.

6.1.5 Section summary

In this section, we introduce ICDO, which uses Inverse Composition (IC) operation to perform the update for registration tasks instead of the summation update rule of DO. ICDO allows the parameters that form a group to be updated under their natural composition operation, and require less computation than Forward Composition (FC) approach. We demonstrated the potential of

ICDO in the problem of 3D point cloud registration (PCReg), illustrating that the learned maps generalize to unseen shapes and that its performance is competitive to those of the state-of-the-art PCReg algorithms.

6.2 Representing feature function $\mathbf{h}$ as a combination of basis functions

In Section 3.2, we proposed a method to derive $\mathbf{h}$ function as a mixture of Dirac delta functions, where $\mathbf{Dh}$ can be interpreted as imitating the gradient steps of an unknown penalty function. A major drawback of the approach is that, to compute $\mathbf{h}$, we need to perform grid-based discretization. There are four problems with such discretization. First, it was shown that the dimension of $\mathbf{h}$ is pqr^d, which means that the discretization does not scale for the residual $\mathbf{g}_j : \mathbb{R}^p \to \mathbb{R}^d$ with large d. Second, we can only discretize a bounded portion of the space of $\mathbb{R}^d$. Any values outside of this portion have to be discarded, which could lead to a loss of information. Third, grid-based discretization may lead to overfitting. Since the number of training samples are finite, there could be a large number of boxes where only a few samples fall into. This could cause the learned maps to overfit to these small number of samples. In the many cases, some boxes may not receive any sample at all, and due to regularization the values of the learned maps will be set to zero. Lastly, it may not be obvious how to set the appropriate value of r. A large r would be able to represent a very complex function but would be hard to scale, while a small r may not be able to represent the function well.

To alleviate such issues, in this section we propose to derive $\mathbf{h}$ as a combination of basis functions instead of discretizing the Dirac delta function. Unlike discretization where the residuals outside the interval $[-q, q]$ are discarded, using basis functions allow residuals with arbitrary values to be included, thereby reducing the loss of information. In addition, using basis functions disregards the need to discretize the space, resulting in fewer number of parameters to learn and also less overfitting. We describe the procedure of deriving $\mathbf{h}$ with basis functions as follows.

6.2.1 Learning with basis functions

Recall that in (3.22), we express the update step with an unknown penalty function φ as

$$\Delta \mathbf{x} = \frac{1}{J} \sum_{j=1}^{J} \left[\frac{\partial \mathbf{g}_j}{\partial \mathbf{x}} \right]^\top \phi(\mathbf{g}_j), \tag{6.21}$$

where $\phi : \mathbb{R}^d \to \mathbb{R}^d$ represents the gradient of φ, and the notation $\mathbf{g}_j$ expresses the residual $\mathbf{g}_j(\mathbf{x})$. Given a set of basis functions $\mathcal{B} = \{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_{n_B}\} \subset (\mathbb{R}^d \to \mathbb{R}^p)$, we can approximate $\phi(\mathbf{g}_j)$ as a linear combination of basis functions as

$$\phi(\mathbf{g}_j) \approx \mathbf{B}(\mathbf{g}_j)\mathbf{d} = [\mathbf{b}_1(\mathbf{g}_j), \mathbf{b}_2(\mathbf{g}_j), \ldots, \mathbf{b}_{n_B}(\mathbf{g}_j)]\mathbf{d}, \tag{6.22}$$

where $\mathbf{B} : \mathbb{R}^d \to \mathbb{R}^{p \times n_B}$ computes that function value of the bases, and $\mathbf{d} \in \mathbb{R}^{n_B}$ is the vector containing the weights of the bases. Thus, (6.21) can be approximated as

$$\Delta\mathbf{x} \approx \frac{1}{J} \sum_{j=1}^{J} \left[\frac{\partial \mathbf{g}_j}{\partial \mathbf{x}}\right]^\top \mathbf{B}(\mathbf{g}_j)\mathbf{d} \tag{6.23}$$

$$= \mathbf{h}(\mathbf{x})\mathbf{d}, \tag{6.24}$$

where we express $\mathbf{h}$ as

$$\mathbf{h}(\mathbf{x}) = \frac{1}{J} \sum_{j=1}^{J} \left[\frac{\partial \mathbf{g}_j}{\partial \mathbf{x}}\right]^\top \mathbf{B}(\mathbf{g}_j). \tag{6.25}$$

It can be seen that (6.24) is simply a multiplication between $\mathbf{h}$ and $\mathbf{d}$, where $\mathbf{d}$ does not depend on $\mathbf{g}_j$. This allows us to learn the sequence of maps $\mathbf{d}_t$ with linear regression similar to (3.6). Note that here, the feature $\mathbf{h}$ becomes a matrix of dimension $p \times n_B$, and the update maps $\mathbf{d}_t$ becomes vectors of dimension n_B. This leads to a modified update rule:

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \mathbf{h}(\mathbf{x}_t)\mathbf{d}_{t+1}. \tag{6.26}$$

There are several benefits of using (6.25) as $\mathbf{h}$ compared with the discretization approach. First, the bases can be selected such that each of their domain cover the whole space $\mathbb{R}^p$, thus no information will be disregarded. In addition, using bases also removes the need to discretize the space. With a good set of bases, this could also lead to a reduced number of parameters in the update maps $\mathbf{d}_t$ that we need to learn, thereby reducing the overfitting problem. Finally, if all basis functions are gradients of some functions, then it is possible to approximately reconstruct the penalty function. However, since the learning is performed sequentially, $\mathbf{d}_t$ will be different for each t, and so the penalty function will also change in each iteration. We can think of this as learning to imitate gradient method on a continuation of cost functions.

6.2.2 Discretization as basis functions

We can also interpret the discretization in Section 3.2 as basis functions. This can be seen by considering the indicator of each grid as a basis function:

$$\mathbf{b}_{b_1,\dots,b_d}(\mathbf{g}_j) = \begin{cases} \bigotimes_{\alpha=1}^{d} \mathbf{e}_{b_\alpha} & ;\gamma([\mathbf{g}_j]_\alpha) = b_\alpha, \forall \alpha, \\ \mathbf{0}_{r^d} & ;\text{otherwise}, \end{cases} \tag{6.27}$$

where $\mathbf{b}_{b_1,\dots,b_d}(\mathbf{g}_j)$ is indexed by $b_\alpha \in \{1, 2, \dots, r\}, \alpha = 1, \dots, d$. In this manner, we unify the discretization approaches as a set of basis functions. However, the basis of discretization is discontinuous and it is not a gradient of any function. This prevents us from exactly recovering the unknown penalty function.

6.2.3 Experiments

In this section, we perform synthetic experiment to compare the basis strategy and discretization strategy.

Problem settings

Here, we are interested in the problem of the form

$$\underset{\hat{\mathbf{x}}}{\text{minimize}} \frac{1}{J} \sum_{j=1}^{J} \varphi\left(\mathbf{A}_j(\hat{\mathbf{x}} - \mathbf{y}_j)\right), \tag{6.28}$$

where $\{(\mathbf{A}_j, \mathbf{y}_j)\}_{j=1}^{J} \subset \mathbb{R}^{2\times 2} \times \mathbb{R}^2$ are a given set of data. Examples of problems that can be formulated in this form include template tracking, optical flow, affine point cloud registration, and image deconvolution, and this formulation can be extended to other problems that includes regularization terms, such as image denoising. We inspect three penalty functions:

$$\varphi_{\ell_2}(\mathbf{x}) = \|\mathbf{x}\|_2^2, \tag{6.29}$$

$$\varphi_{\ell_1}(\mathbf{x}) = \|\mathbf{x}\|_1, \tag{6.30}$$

$$\varphi_{IG}(\mathbf{x}) = -\exp\left(-\frac{1}{2 \cdot 0.5^2}\|\mathbf{x}\|_2^2\right) + \frac{1}{10}\|\mathbf{x}\|_2^2, \tag{6.31}$$

where φ_{ℓ_2} is the squared ℓ_2 function, which is smooth and strongly convex; φ_{ℓ_1} is the ℓ_1 function, which is nonsmooth and convex; and φ_{IG} is the sum of inverted Gaussian and squared ℓ_2 function, which is smooth and strongly pseudoconvex.

The training set is given by $\mathcal{X} = \{(\mathbf{x}_0^{(i)}, \mathbf{x}_*^{(i)}, \{(\mathbf{A}_j^{(i)}, \mathbf{y}_j^{(i)})\}_{j=1}^{J^{(i)}})\}_{i=1}^n$, where $\mathbf{A}_j^{(i)}$ was randomly generated from $[-0.75, 0.75]^{2\times 2}$, $\mathbf{y}_j^{(i)}$ from $[-1, 1]^2$, and $J^{(i)}$ from $\{3, \ldots, 50\}$. The initializer $\mathbf{x}_0^{(i)}$ was set to $\mathbf{0}_2$, and the minimum $\mathbf{x}_*^{(i)}$ was generated by solving (6.28) using `fminunc`. We generated 10^4 samples for training and 5×10^3 for test.

Performance measures

We measure the performance of each instance i with the ℓ_2 error from the current estimate to its ground truth: $\|\mathbf{x}_t^{(i)} - \mathbf{x}_*^{(i)}\|$. We report the mean and the median of the training error and the test error against the iteration number t to see how these errors progress through the iterations. Note that the mean error should be used as the main measures for comparison as the DO learning rule minimizes the mean error, while the median error is shown for reference as there are *hard* problem instances that can strongly affect the mean.

Algorithm settings

We compare three main classes of algorithms: DO with discretization-based $\mathbf{h}$ from Section 3.2; DO with the basis functions from Section 6.2.1; and the subgradient method, assuming we know the correct penalty function. We provide the settings of each algorithm as follows.

Discretization: We discretize $\mathbf{g}_j$ in the range $[-2, 2]^2$ (*i.e.*, $q = 2$) with 3 different values of grids r, which are 10, 20, and 40. We denote these discretization strategies as D10, D20, and D40, *resp*. The learned maps of D10, D20, and D40 will have the dimensions of 2×400, 2×1600, and 2×6400, *resp*. Here, we set $\lambda = 10^{-5}$.

Basis functions: We use two basis strategies, denoted by B1 and B2. B1 comprises of 5 sinusoidal functions for each dimension, and B2 contains the basis functions of B1 with 2 additional bases in each dimension. Their basis matrices $mathbf{B}$ (see (6.22)) are given by

$$\mathbf{B}_b(\mathbf{g}_j) = \begin{bmatrix} \hat{\mathbf{b}}_b^\top([\mathbf{g}_j]_1) & \mathbf{0}^\top \\ \mathbf{0}^\top & \hat{\mathbf{b}}_b^\top([\mathbf{g}_j]_2) \end{bmatrix}, b = 1, 2, \tag{6.32}$$

where

$$\hat{\mathbf{b}}_1^\top(\alpha) = [\sin(2\alpha), \sin(4\alpha), \sin(6\alpha), \sin(8\alpha), \sin(10\alpha)], \tag{6.33}$$

$$\hat{\mathbf{b}}_2^\top(\alpha) = [\sin(2\alpha), \sin(4\alpha), \sin(6\alpha), \sin(8\alpha), \sin(10\alpha), \alpha, \mathrm{sgn}(\alpha)], \tag{6.34}$$

where $\mathrm{sgn}(\cdot)$ is the Heaviside step function. The matrix $\mathbf{B}_1$ is the basis matrix of B1, and $\mathbf{B}_2$ is the basis matrix of B2. We use these two sets of bases to compare the situation when we have

the exact gradients of the cost as part of the basis (B2) against the case that we do not (B1): B2 contains functions which are the gradient of φ_{ℓ_2} and φ_{ℓ_1}, while B1 has only sinusoidal functions. The learned maps of B1 and B2 will have the size of 10×1 and 14×1, respectively. Note that all the functions we selected as basis are odd functions, as even functions are not monotone at $\mathbf{0}$ for any subset of its domain except at $\mathbf{0}$. Here, we set $\lambda = 10^{-10}$.

Subgradient: Assuming we know the penalty function φ, we can use the subgradient method to solve for the minimum. Here, we use the subgradient method with decreasing step sizes [89] as a comparison. This is given by

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \frac{1}{t}\mathbf{v}_t, \tag{6.35}$$

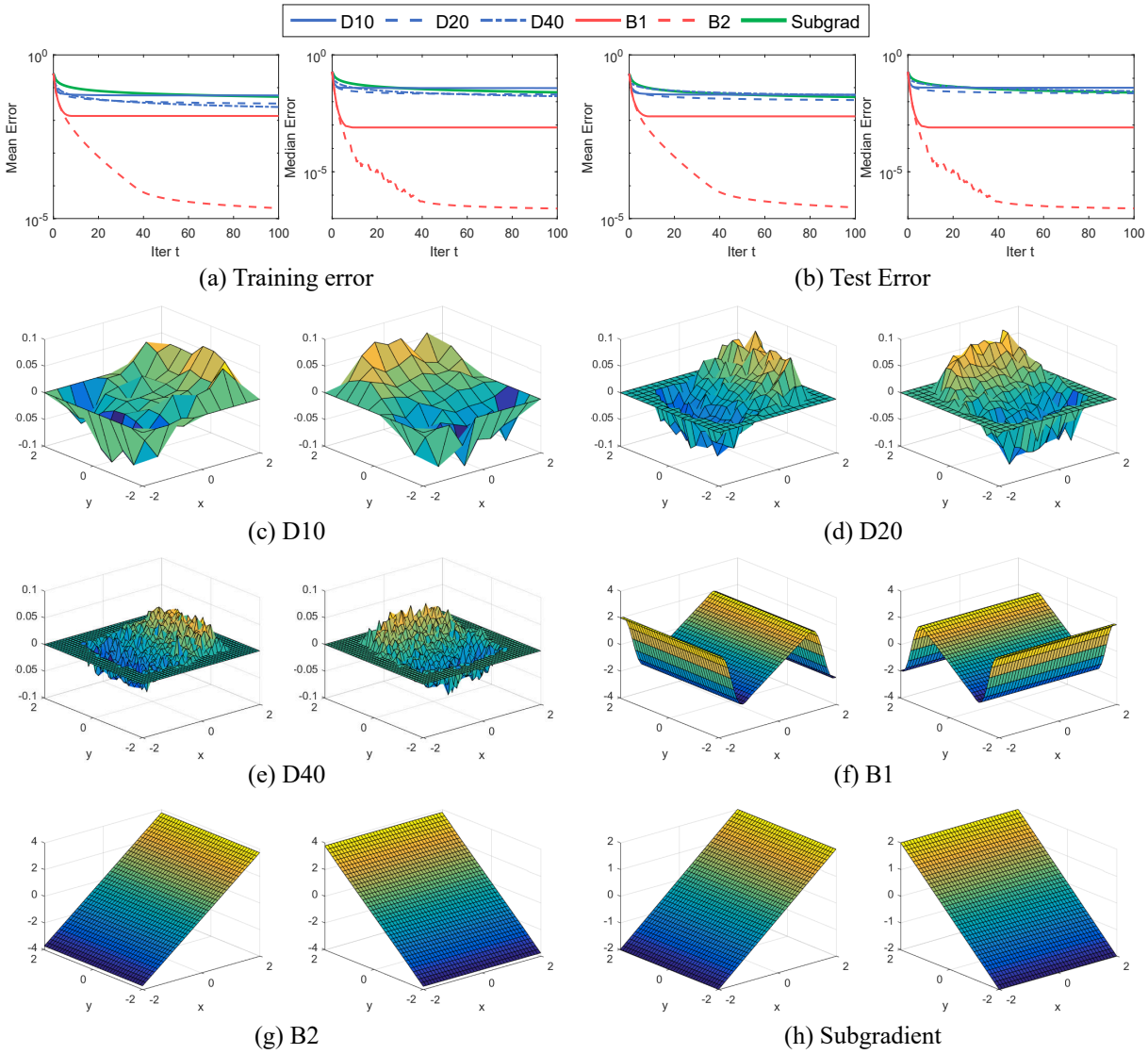
where $\mathbf{v}_t$ is a subgradient of φ at $\mathbf{x}_t$. Note that we include subgradient method here to compare their convergence rate only, but they cannot replace DO in real applications since the subgradient methods assume knowing the cost function while DO learns the update directions from training data.

Results

Figure 6.4 shows the results for φ_{ℓ_2}, which is a differentiable strongly convex function. For the training error in Figure 6.4a, we can see that the training error of the discretization strategies reduces faster as the discretization grid r increases. However, this could also lead to overfitting, as can be seen in Figure 6.4b, where the error of D40 reduce slowest of all strategies. On the other hand, B1 and B2 can reduce the training error faster than all discretization-based methods, while also maintaining good decrease in the test error. We can also see that knowing the correct basis can significantly speed up the convergence from the result that the error of B2 decreases much faster than that of B1. In addition, basis-based methods can decrease error faster than the subgradient method, suggesting that the basis approaches can learn to find a better sequence of step sizes than that of the subgradient method. Another thing to notice is that the mean error can decrease much slower than the median error. This is because there are some hard samples where $\mathbf{A}_j^{(i)}$ have very high condition numbers, *e.g.*, more than 10^6. This cause the cost surface of some problem instances to become almost flat in some directions, and gradient-based methods can take a long time to converge [17]. Since DO imitates gradient-based methods, this condition number also affects DO in a similar fashion.

Next, we take a look at what DO learns. Figures 6.4c,d,e,f,g show reconstructed update maps of each strategy[1]. These maps are comparable to the directional derivative of φ_1, of which $\frac{\partial \varphi_1(\mathbf{x})}{\partial [\mathbf{x}]_1}$

[1]We show that last map of each strategy that reduces training RMSE more than 1×10^{-3}. For discretization strategies, we reshape the learned maps into 2D grids indexed by the location of $\mathbf{g}_j(\mathbf{x})$. For basis strategies, we



(a) Training error (b) Test Error

(c) D10 (d) D20

(e) D40 (f) B1

(g) B2 (h) Subgradient

Figure 6.4: DO with basis with φ_{ℓ_2}. (a) Training error. (b) Test error. (c-g) Learned maps for each method (reconstructed for visualization). (h) Directional subgradient of ϕ_{ℓ_2}.

and $\frac{\partial \varphi_1(\mathbf{x})}{\partial [\mathbf{x}]_2}$ are shown on the left and right of Figure 6.4h, *resp*. We can see that the learned maps resemble the subgradient up to some scale factors. These scale factors replace the use of step size in subgradient method. The maps of D10 and D20 are quite smooth while those of D40 are not. This provides an evidence that D40 is overfitting, which results in the slow decrease in test error shown in Figure 6.4b. More data may be required for training if we wish to discretize with a large number of grids. In addition, we can see that the edges of the maps of D10, D20, and D40 have values of 0. This is due to the fact that the residual $\mathbf{g}_j$ in the training data may

multiply the learned coefficients to the basis functions to obtain the maps.

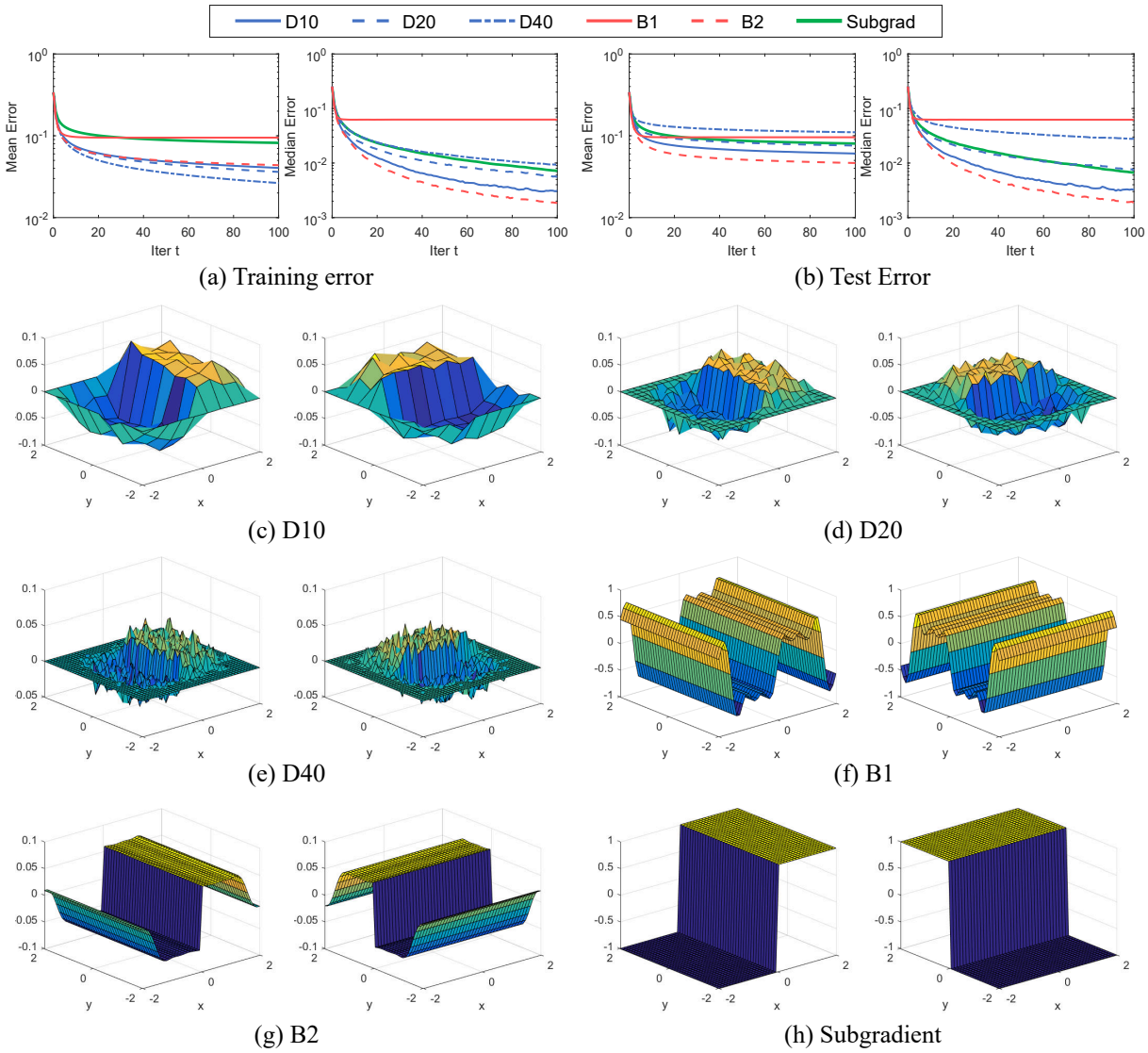


Figure 6.5: DO with basis with ϕ_{ℓ_1}. (a) Training error. (b) Test error. (c-g) Learned maps for each method (reconstructed for visualization). (h) Subgradient of ϕ_{ℓ_1}.

not be able to cover the whole space, and the grids with no residuals will have the value set to 0. On the other hand, for the basis strategies where smooth basis functions are used, their maps are also smooth. The maps of B2 look almost similar to the subgradient in Figure 6.4h. This is because the basis set of B2 also includes the derivative of φ_{ℓ_2}, allowing it to learn a similar set of maps. On the other hand, B1 does not have the derivative of φ_{ℓ_2} as basis functions, thus it learns the update maps that best approximate the subgradient map. In this case, we can see that the sinusoidal basis functions can be use to learn the update maps quite well.

Next, we look at the results of φ_{ℓ_1} in Figure 6.5. Recall that φ_{ℓ_1} is the ℓ_1 norm, which

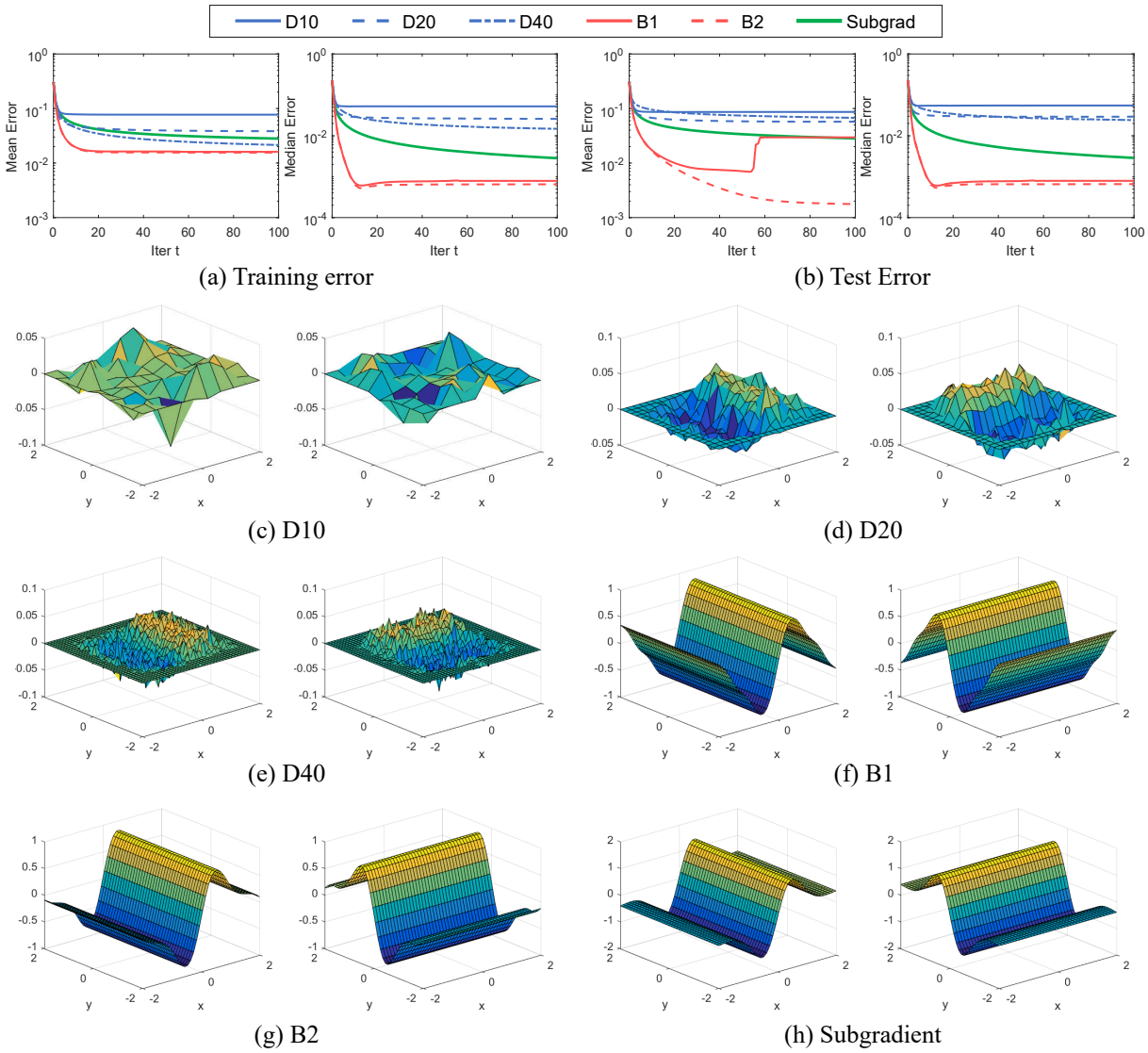


Figure 6.6: DO with basis with ϕ_{IG}. (a) Training error. (b) Test error. (c-g) Learned maps for each method (reconstructed for visualization). (h) Subgradient of ϕ_{IG} (Note that here we use the term *subgradient* loosely as ϕ_{IG} is not convex).

is a nonsmooth convex function. Similar to the results of φ_{ℓ_2}, we can see from the errors in Figure 6.5a,b that the discretization strategies, especially D40, overfit to the training data. The learned maps also reflect this overfitting, as the maps of D20 and D40 are not as flat compared with those of D10 and the subgradient. On the other hand, the results of the two basis strategies differ in this case. The training error and the test error of B1 do not reduce beyond a certain value, while those of B2 still reduce in every iteration. This is because B1 has only continuous basis functions which cannot represent the nonsmooth subgradient of φ_{ℓ_1}, and this also reflects as the wavy surfaces in the reconstructed maps in Figure 6.5f. Meanwhile, B2 has $\operatorname{sgn}(\cdot)$ in its

basis, allowing it to learn a good set of update maps, and its reconstructed maps in Figure 6.5g can well represent the 'jump' in φ_{ℓ_1}'s subgradient.

Finally, we look at the results of φ_{IG}, which is a smooth strongly pseudoconvex function, in Figure 6.6. Here, we can see that the basis strategies can learn good sets of update maps, which can well represent the subgradient maps and reduce both the training error and test error in a fast manner. Note that the mean test error of B1 increases around iteration 60 due to a test instance jumped to an incorrect local minimum. On the other hand, D10, which did not overfit with φ_{ℓ_2} and φ_{ℓ_1}, does not perform well with φ_3: Its errors do not decrease much and the learned maps do not look similar to those of the subgradient. This may be due to the fact that its number of grids is too small, and thus it cannot well represent the smoothly varying true maps. Among the discretization strategies, D20 performed the best, while D40 still show signs of overfitting when looking at the mean test error.

6.2.4 Concluding remarks and discussions

In this section, we show that it is possible to use a set of basis functions to learn the update directions instead of using discretization strategies from Chapter 3.2. Since the domain of basis functions cover the whole space, the learned maps will not have the value of zero beyond the values of the residuals $\mathbf{g}_j$. In addition, since the number of basis functions can be small, they are less likely to overfit than the discretization approaches.

There are also several disadvantages when using basis functions. As can be seen in the case of φ_{ℓ_1}, if the set of basis functions are not well-selected, we may not be able to learn a good sequence of update maps (SUM) that greatly reduces both training and test errors. In addition, here we only show the case where the basis functions are axis-aligned. It is possible to include other non-axis-aligned functions, such as $\sin(\mathbf{W}\mathbf{x})$ for some matrix $\mathbf{W}$. However, in high-dimensional settings, manually selecting $\mathbf{W}$ can be a challenging task, and we may need a large number of such basis to represents the high-dimensional update maps. One approach to tackle this issue is to learn both the update maps and the matrix $\mathbf{W}$.

An interesting observation is that when the differentiable basis functions are used, the whole system becomes differentiable. This may allow the whole system to be learned in an end-to-end manner, similar to deep neural networks (DNNs) [48]. It is also interesting to note that if we use the rectified-linear function (ReLU) or the sigmoid function as the basis function, then DO may look very similar to DNNs. This observation provides a bridge between DO and DNNs.

6.3 Accelerating DO with momentum

Momentum methods [44] are a group of approaches which are used to accelerate the convergence of optimization algorithms. Typically, they involve different weighting schemes that incorporate the estimates from previous steps into computing the next estimates. Suppose we are given a differentiable cost function $f : \mathbb{R}^p \to \mathbb{R}$, a momentum method for minimizing f is given by

$$\mathbf{z}_{t+1} = \beta \mathbf{z}_t + \nabla f(\mathbf{x}_t) \tag{6.36}$$

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \alpha_t \mathbf{z}_{t+1}, \tag{6.37}$$

where $\mathbf{x}_t \in \mathbb{R}^p$ is the estimate at step t, $\mathbf{z}_t \in \mathbb{R}^p$ is an auxiliary vector, $\alpha_t \in \mathbb{R}$ is the step size, and $\beta \in \mathbb{R}$ is a weight constant. By eliminating $\mathbf{z}_t$, we can rewrite the momentum method in a single line as

$$\mathbf{x}_{t+1} = (1 + \beta)\mathbf{x}_t - \beta \mathbf{x}_{t-1} - \alpha_t \nabla f(\mathbf{x}_t). \tag{6.38}$$

From (6.38), we can see that the momentum update is simply a linear combination of $\mathbf{x}_t$, $\mathbf{x}_{t-1}$, and the gradient at $\mathbf{x}_t$.

6.3.1 Incorporating momentum into DO

It is straightforward to incorporate momentum terms into DO. Instead of using fixed weights as in (6.38), we simply concatenate the feature vector $\mathbf{h}$ with the previous parameter estimates $\mathbf{x}_{t-1}, \ldots, \mathbf{x}_{t-\tau}$, where τ is the number of previous steps used. This leads to the following analogy of (3.4):

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \mathbf{D}_{t+1} \mathbf{h}_{mmt}(\mathbf{x}_t, \cdots, \mathbf{x}_{t-\tau}), \tag{6.39}$$

where

$$\mathbf{h}_{mmt}(\mathbf{x}_t, \cdots, \mathbf{x}_{t-\tau}) = [\mathbf{h}^\top(\mathbf{x}_t) \;\; \mathbf{x}_t^\top \;\; \mathbf{x}_{t-1}^\top \;\; \cdots \;\; \mathbf{x}_{t-\tau}^\top]^\top. \tag{6.40}$$

When learning the SUM, the coefficients of the momentum will be directly learned into the matrix $\mathbf{D}_{t+1}$.

We can also discuss the convergence result of the training error when momentum is used. Relying on the convergence result in Thm. 1, it is easy to see that if there exists $\hat{\mathbf{D}}$ such that $\hat{\mathbf{D}}\mathbf{h}^{(i)}$ for all i are strictly monotone at $\mathbf{x}_*^{(i)}$, then there exists $\hat{\mathbf{D}}_{mmt}$ of the form

$$\hat{\mathbf{D}}_{mmt} = [\hat{\mathbf{D}} \;\; \mathbf{0}_{p \times (\tau+1)p}], \tag{6.41}$$

where $\hat{\mathbf{D}}_{mmt}\mathbf{h}_{mmt}^{(i)}$ are strictly monotone at $\mathbf{x}_*^{(i)}$. Thus, the convergence results from Chapter 4

also apply to DO with momentum as well.

6.3.2 Experiments

We perform experiments to test DO with momentum and compare its efficiency against the regular DO. We use the same problem settings and performance measure as in Section 6.2.3.

Algorithm settings

We use D20 and B2 from Section 6.2.3. To denote the algorithms that use previous steps, we append the names with τ and the number of previous steps, *e.g.*, B2τ2 indicates using B2 where its $\mathbf{h}_{mmt}$ includes 0 to 2 previous steps, which are $\mathbf{x}_t$, $\mathbf{x}_{t-1}$, and $\mathbf{x}_{t-2}$. The algorithms using no previous steps are denoted without any following τ, *i.e.*, D20 and B2. Here, we test the cases of no previous steps and $\tau = 0, 1, 2$. For subgradient, we include subgradient with and without momentum, denoted as *Subgrad* and *Subgrad+mmt*, for comparison. For Subgrad+mmt, we use $\beta = 0.5$.

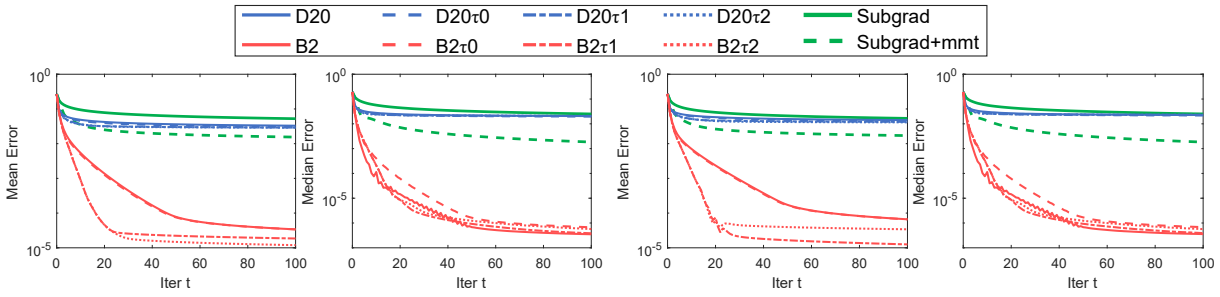
Results

The training errors and test errors on φ_{ℓ_2}, φ_{ℓ_1}, and φ_{IG} are shown in Figures 6.7, 6.8, and 6.9, *resp*. We can see that all approaches, including subgradient, benefit from momentum: All the training errors and test errors decrease faster when momentum is used. For DO approaches, simply adding the current estimate (D20τ0 and B2τ0) does not make the convergence faster. Only when we include the first previous steps (D20τ1 and B2τ1) that we see the effect of momentum. The effect is most apparent for the basis approaches in φ_{ℓ_2} and φ_{IG}, where we can observe a significant improvement on the convergence rate. We believe that this is due to the fact that the momentum can help compensate for the high condition number of the cost function [44]. On the other hand, adding additional previous steps (D20τ2 and B2τ2) does not significantly improve the convergence beyond using a single previous step. This indicates that only a single previous estimate may be adequate for obtaining optimal convergence rate.

6.4 Incorporating constraints into DO

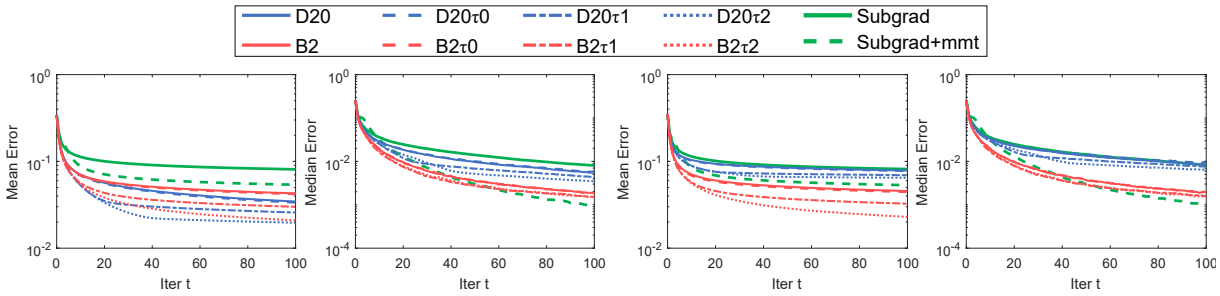
Constrained optimization deals with problems of the form

$$\begin{array}{ll} \text{minimize}_{\mathbf{x}} & f(\mathbf{x}) \\ \text{subject to} & \mathbf{x} \in \mathcal{S}, \end{array} \tag{6.42}$$



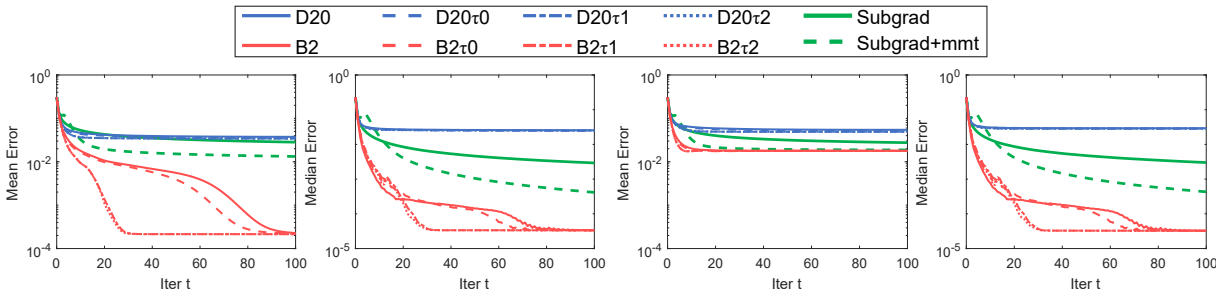
(a) Training error (b) Test Error

Figure 6.7: DO with momentum with φ_{ℓ_2}. (a) Training error. (b) Test error.



(a) Training error (b) Test Error

Figure 6.8: DO with momentum with φ_{ℓ_1}. (a) Training error. (b) Test error.



(a) Training error (b) Test Error

Figure 6.9: DO with momentum with φ_{IG}. (a) Training error. (b) Test error.

where $\mathcal{S}$ is the constraint set, which we assume to be convex. The constraints are used to restrict the set of values that the variables can take. There exists many algorithms for handling constraint in convex optimization [17, 19], such as penalty method, projected subgradient method, Frank-Wolfe algorithm, interior point methods, *etc*.

In this section, we introduce constraints into DO using a combination of the penalty method (PM) and the projected subgradient method (PSM) [66], assuming we have access to the projection operator of $\mathcal{S}$. First, we briefly review the two methods. Then, in Section 6.4.1, we describe our constrained DO, called *Projection-Penalty DO (PPDO)*, and provide our rationale on why

we choose to combine the two methods. In Section 6.4.2, we perform a synthetic experiment to verify PPDO.

Penalty method (PM) [66]: Suppose the set $\mathcal{S}$ in (6.42) is given by a set of equations:

$$\mathcal{S} = \{\mathbf{x} \in \mathbb{R}^p : f_k^{\leq}(\mathbf{x}) \leq 0, k = 1, \ldots, n_1; f_l^{=}(\mathbf{x}) = 0, l = 1, \ldots, n_2\}. \tag{6.43}$$

PM transforms the problem (6.42) into the following unconstrained optimization problem:

$$\underset{\mathbf{x}}{\text{minimize}}\, f(\mathbf{x}) + \alpha \sum_{k=1}^{n_1} (\max(0, f_k^{\leq}(\mathbf{x})))^2 + \alpha \sum_{l=1}^{n_2} (f_l^{=}(\mathbf{x}))^2, \tag{6.44}$$

where α is a large-value hyperparameter. In words, the problem (6.44) severely penalizes the constraint violations, causing the variable $\mathbf{x}$ to remain in the constrain set. It can be shown that as $\alpha \to \infty$, the optimal solution of (6.44) approaches the optimal solution of the original problem 6.42. However, it is not trivial to to solve (6.44) with large α, and this could lead to a very slow convergence.

Projected subgradient method (PSM) [19]: Unlike PM which relaxes constrained problems to unconstrained ones, PSM directly solves the constrained problems by greedily taking a subgradient step then projecting the updated estimates back to constraint set. Mathematically, its update rule is given by

$$\mathbf{x}_{t+1} = \Pi_{\mathcal{S}}(\mathbf{x}_t - \mu_t \mathbf{v}_t), t = 0, 1, \ldots, \tag{6.45}$$

where $\mathbf{v}_t \in \partial f(\mathbf{x}_t)$ is a subgradient of f at $\mathbf{x}_t$, and $\Pi_{\mathcal{S}}$ is the projection operator of the set $\mathcal{S}$. The projection enforces the estimate $\mathbf{x}_t$ to comply with the constraints in every iteration. The downside of PSM is that, besides simple constraint sets where the projection can be computed in closed-form, it may be computationally expensive to solve the projection step as this requires solving an additional optimization problem in each iteration.

6.4.1 Projection-Penalty DO (PPDO)

In Section 3.2, we have shown that we can use DO with unknown penalty function. Here, we adapt the approach to the unconstrained formulation in (6.44). Specifically, we will use DO to learn the functions that penalizes constraint violation. In addition, we will combine this method with the projection operator. This leads to the Projection-Penalty DO (PPDO), which is given by

$$\mathbf{x}_{t+1} = \Pi_{\mathcal{S}}(\mathbf{x}_t - \mathbf{D}_{t+1}\hat{\mathbf{h}}(\mathbf{x}_t)), \tag{6.46}$$

where

$$\hat{\mathbf{h}}(\mathbf{x}_t) = \begin{bmatrix} \mathbf{h}(\mathbf{x}_t) \\ \mathbf{h}^{\leq}(\mathbf{x}_t) \\ \mathbf{h}^{=}(\mathbf{x}_t) \end{bmatrix}, \tag{6.47}$$

where $\mathbf{h} : \mathbb{R}^p \to \mathbb{R}^f$ is the feature function for the task of interest, *e.g.*, the task-specific feature function from Section 3.2, the basis feature function from Section 6.2.1, or the feature used for point cloud registration in Section 5.1. The functions $\mathbf{h}^{\leq}$ and $\mathbf{h}^{=}$ are the feature functions for the inequality constraints and the equality constraints, *resp*. They can be derived based on the feature in Sections 3.2 or 6.2.1. Note that since the constraint functions $f_k^{\leq}$ and $f_l^{=}$ are 1D functions, we can derive the feature for each constraint independently, then simply concatenate them to form $\mathbf{h}^{\leq}$ and $\mathbf{h}^{=}$.

In order to learn the SUM, we simply solve the unconstrained linear least squares minimization, similar to the learning rule in Section 3.1. However, we use the projected update rule in (6.46) to update $\mathbf{x}_t^{(i)}$ instead of the unprojected version in (3.4).

Next, we provide the rationale why we wish to combine the PM with PSM. Based on the interpretation that DO imitates gradient-based methods in unconstrained cases, we should be able to apply the PM directly to DO without the projection. However, a downside of this approach is that we often require a strong penalty on the constraint violation, and this could easily lead to overfitting for learning-based algorithm. In particular, the weights in $\mathbf{D}_t$ of the constraint features $\mathbf{h}^{\leq}$ and $\mathbf{h}^{=}$ would need to be very large compared with those of $\mathbf{h}$ to ensure the constraints are not violated. As a result, since DO can be considered as updating $\mathbf{x}_t$ with a fixed step size, these large weights of the constraint features can cause the update $\Delta_{t+1}\mathbf{x}_t$ to be so large that $\mathbf{x}_t - \Delta_{t+1}\mathbf{x}_t$ will jump far outside $\mathcal{S}$, leading to divergence. On the other hand, if we attempt to apply PSM to DO without PM, we are required us to learn the maps using the following optimization problem:

$$\mathbf{D}_{t+1} = \arg\min_{\mathbf{D}} \sum_{i=1}^{N} \|\mathbf{x}_*^{(i)} - \Pi_{\mathcal{S}}(\mathbf{x}_t^{(i)} - \mathbf{D}\mathbf{h}^{(i)}(\mathbf{x}_t^{(i)}))\|_2^2. \tag{6.48}$$

For a general set $\mathcal{S}$, problem (6.48) is nonconvex and can be very difficult to solve.

We can see that using the update rule and the learning rule proposed earlier can handle the drawbacks of both PM and PSM: we avoid the overfitting of PM using the projection operation since all points will be projected back inside $\mathcal{S}$; and avoid having to solve nonconvex problems for learning the maps of PSM by considering the penalty form of the constraints. This results in the PPDO algorithm. The idea of combining PM and PSM is similar to the algorithm in [66]. However, while PPDO ensures the constraints are satisfied in each iteration using the projector, [66] uses Newton's algorithm to move $\mathbf{x}_t$ closer to the constraint set and does not enforce the

constraints in all iterations. In the next section, we perform an experiment to verify PPDO.

6.4.2 Experiments

Problem settings

We use the problem settings similar to those in 6.28, but we constrain the variables to lie in a convex constraint set:

$$\underset{\hat{\mathbf{x}}}{\text{minimize}} \ \frac{1}{J}\sum_{j=1}^{J}\varphi\left(\mathbf{A}_j(\hat{\mathbf{x}}-\mathbf{y}_j)\right), \tag{6.49}$$

$$\text{subject to } \ \mathbf{x}\in\mathcal{S}, \tag{6.50}$$

where the constraint set $\mathcal{S}$ is given by

$$\mathcal{S}=\{\mathbf{x}\in\mathbb{R}^2 : [x]_1\geq 0, 0.75\geq[x]_2\geq 0, \|\mathbf{x}\|_2\leq 1\}. \tag{6.51}$$

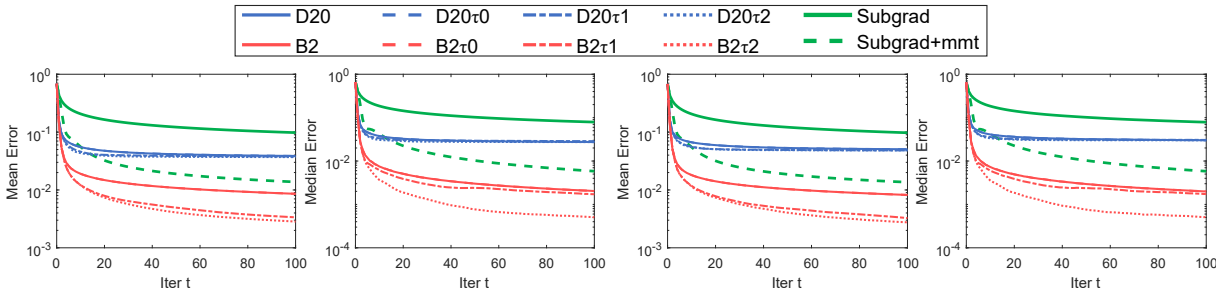
Again, we use the same φ_{ℓ_2},φ_{ℓ_1}, and φ_{IG} from Section 6.2.3 in this experiment.

Algorithm settings

Here, we use the same algorithm settings as in Section 6.3.2, *i.e.*, we include the momentum terms into DO and the subgradient method. However, here we use PPDO algorithm for each DO instead of the unconstrained version. For the subgradient methods, we use PSM, where we simply project the estimate parameters back to the constraint set in each iteration.

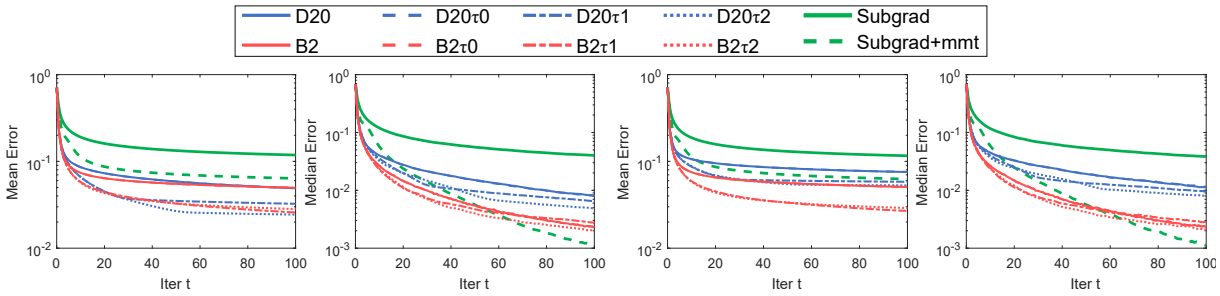
Results

The training errors and test errors of φ_{ℓ_2}, φ_{ℓ_1}, and φ_{IG} are shown in Figures 6.10, 6.11, and 6.12, respectively. We can see similar convergence trends to those in Section 6.3.2 that the momentum terms help speed up the convergence compared with not using momentum. However, due to the constraints, the convergence rates are slower than in those in Section 6.3.2. Still, we can see that the DO methods, especially the basis strategies, can successfully reduce the error faster than PSM. This shows that our combination, inspired by PM and PSM, can be extended to allow constraints into DO.



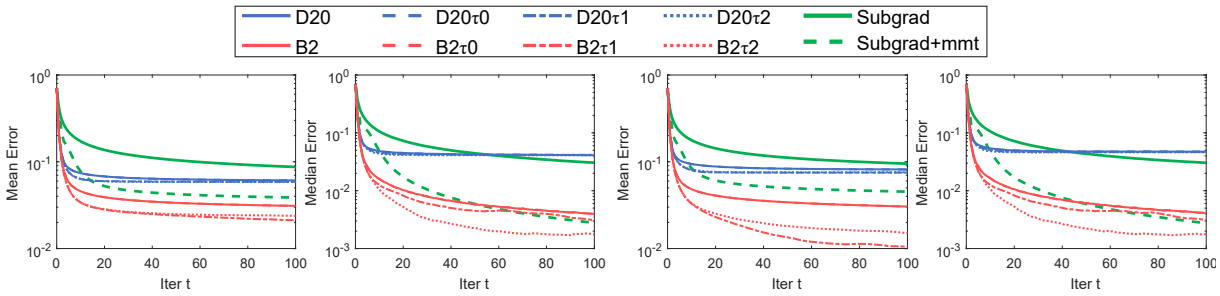
(a) Training error (b) Test Error

Figure 6.10: Constrained DO with φ_{ℓ_2}. (a) Training error. (b) Test error.



(a) Training error (b) Test Error

Figure 6.11: Constrained DO with φ_{ℓ_1}. (a) Training error. (b) Test error.



(a) Training error (b) Test Error

Figure 6.12: Constrained DO with φ_{IG}. (a) Training error. (b) Test error.

6.5 Chapter summary

In this chapter, we explore four generalizations of DO. *(i)* We introduce ICDO that uses inverse composition update rule for registration tasks where the parameters form a group structure. We demonstrate its potential on the task of 3D rigid point cloud registration (PCReg), and show that the learned maps of ICDO generalize to unseen shapes and that its performance matches those of the state-of-the-art PCReg algorithms. *(ii)* We show how to derive the feature function as a combination of basis functions, and empirically show that it can achieve better convergence

results than the discretization method from Section 3.2. We also show that the convergence can be faster than that of the subgradient method even when the cost function is known, which is due to the fact that DO learns the optimal step size that reduce the average error in each iteration. *(iii)* We incorporate momentum terms into the feature $\mathbf{h}$, and show that this helps speed up the convergence. *(iv)* We extend DO to handle constraints by applying ideas from the penalty method and projected subgradient method. These generalizations show that it is possible to import ideas from conventional gradient-based algorithms to DO. They also open the door for applying DO to a wider range of applications and also strengthen the relation between DO and first-order optimization techniques. Future work may explore other types of generalization, *e.g.*, the cases where the solution $\mathbf{x}_*$ is not unique, learning to adapt the step sizes (*e.g.*, similar to line search), *etc*., and also verify these methods with real applications.

Chapter 7

Conclusions and Future Work

7.1 Contributions

In this thesis, we propose Discriminative Optimization (DO), a learning-based approach for solving computer vision tasks. DO searches the parameter space for a solution by following a series of update vectors, computed by mapping feature vectors with a sequence of linear maps. To apply DO to different tasks, we propose a procedure to derive task-specific feature functions, motivated by gradient methods for solving optimization problems. Rather than following descent directions of a cost function, we can interpret DO as learning to imitating gradient methods on an unknown penalty function. While conventional optimization fails to obtain the correct solutions if there is a mismatch between the cost function and the real noise distribution, DO learns from training data to be robust to such nuisances. Using synthetic experiments, we verify that DO can learn to cope with different types of perturbations, and can still approximate the optimum of unknown cost functions. This confers DO an advantage over conventional optimization.

We analyze the convergence properties of DO and show that under a different set of *monotonicity at a point* and *relaxed Lipschitz at a point* conditions, the training error strictly decreases in each iteration or may even converge to zero. By studying these conditions, we unveil the connection between DO and generalized monotonicity and generalized convexity. From these results, we show that DO converges under broader conditions that those of convexity. This allows DO to handle certain *nonconvex* tasks, *e.g.*, camera pose estimation, or learn more robust solutions than those of conventional optimization.

We apply DO to the computer vision tasks of point cloud registration, camera pose estimation, and image denoising. We show that DO can often outperform state-of-the-art algorithms in terms of both computation time and accuracy.

Finally, we explore four generalizations of DO. *(i)* We show how to use the inverse composi-

tion operation as the update rule, enabling DO to handle registration tasks where the parameters may combine under composition operations other than addition. *(ii)* We propose an alternative way of deriving feature function as a linear combination of basis functions. This reduces the number of parameters to learn and allow DO to be used with high-dimensional residual functions. *(iii)* We accelerate the convergence of DO using momentum terms. *(iv)* We propose Projection-Penalty DO (PPDO) that can handle convex constraints on its parameters.

In conclusion, DO provides a new perspective on using training data to solve problems which are conventionally regarded as optimization problems. While techniques similar to DO that use a sequence of regressors to search the parameter space have been applied to many tasks, in this thesis we shed light on why these techniques work and also provide a systematic procedure to apply DO to several tasks. Still, we believe DO is in its infancy, and there are many questions left open. We explore some of these issues in the next section.

7.2 Future work

Of the many questions left open, we list those that are interesting and are promising as future avenue for research.

An end-to-end DO: We show in Section 6.2 that the features of DO can be derived from basis functions. If the basis functions are differentiable, then it is possible to learn DO in an end-to-end fashion instead of sequentially learning each map greedily. In the same spirit as deep neural networks, this may allow different objective functions to be used with DO, *e.g.*, classification error. This could lead to the possibility of DO being used for classification problems, and also for using DO in large-scale problems where we need to estimate a large number of parameters.

Learning with a set of solutions instead of a point: In this thesis, we assume that the provided ground truth for each training instance is a single point, and to show convergence in such case, we require to impose *strict or strong monotonicity at a point*. On the other hand, there are many problems where the solution can be any point in a given set instead of a specific point, *e.g.*, finding the median of a set of numbers, solving geometric problems under homogeneous coordinates, *etc*. Training DO so that it can deal with a set of solutions would allow it to tackle such problems. In terms of theory, this may lead to new convergence results under the regular *monotonicity at a point*, which is weaker than *strict and strong monotonicity at a point*.

Robustness to inaccurate ground truths: In many cases, it may not be possible to provide the exact solution to a task as a part of the training data, while it is possible to provide an approximate one. In such case, our convergence results may not hold. To handle this issue, it is vital to study the robustness of the algorithm when the ground truths are inaccurate.

Adaptive use of the maps: The present DO framework uses each map once, except the last map where we apply it until termination. A problem of this framework is that estimates which are at different distances to the solutions will be updated using the same map, which may not be the optimal approach and can hinder the convergence speed. It would be beneficial to have some criteria to decide how many times we should apply each map, or even adopt a transition criteria between the maps, similar to a finite state machine. This may help reduce the number of steps required for convergence.

DO for factorization, bilinear relationship, and low-rank regularization: Many computer vision problems rely on factorization or bilinear relationship of low-rank matrices, *e.g.*, structure from motion [94], motion segmentation [30], and photometric stereo [8]. There are two main issues if we wish to use DO for these problems. First, due to gauge freedom, the solutions of these factorization problems are not unique, thus it is difficult to provide the ground truth matrices. Second, these problems are typically solved with block-wise coordinate descent [34]. It is not clear how DO would behave in the intermediate steps, where we fix one matrix and solve for the other one. Also related is the problem of low-rank regularization [20], where the solution is governed by singular values of the variable matrix instead of all elements. It is currently not clear how DO can be applied to these problems. Additional modification and analysis are required, and will lead to new approaches for solving large scale problems with numerous applications.

Imitation of higher-order or other classes of algorithms: In this work, we propose a framework that imitates first-order algorithms. Based on the derivation of the feature function in Section 3.2, it should also be possible to derive different frameworks that imitate higher-order algorithms, such as the Newton's algorithm or the Levenberg-Marquardt algorithm [42]. Doing so will allow DO to handle problems with high condition numbers and may lead to faster convergence.

Relation to Single Index Latent Variable Models (SILVar) [70]: SILVar is a nonlinear, nonparametric regression technique for high-dimensional data. It can be interpreted as approximating the gradient of a convex function at a set of sample points. The approach could serve as an alternative, nonparametric method for learning the update directions of DO instead of using the matrices $\mathbf{D}_t$. In addition, since SILVar can perform dimensionality reduction and discover hidden relationships between data samples, it could also help DO cope with high-dimensional residual functions.

Appendix A

Proofs for Theoretical Results

In this appendix, we provide the proofs for the theoretical results in this thesis.

A.1 Proof of Thm. 1

Theorem 1. (Convergence of DO's training error) *Given a training set* $\{(\mathbf{x}_0^{(i)}, \mathbf{x}_*^{(i)}, \mathbf{h}^{(i)})\}_{i=1}^N$, *if there exists a linear map* $\hat{\mathbf{D}} \in \mathbb{R}^{p\times f}$ *where* $\hat{\mathbf{D}}\mathbf{h}^{(i)}$ *is* $\mathcal{M}^+(\mathbf{x}_*^{(i)})$ *for all* i, *and if there exists an* i *where* $\mathbf{x}_t^{(i)} \neq \mathbf{x}_*^{(i)}$, *then using the update rule:*

$$\mathbf{x}_{t+1}^{(i)} = \mathbf{x}_t^{(i)} - \mathbf{D}_{t+1}\mathbf{h}^{(i)}(\mathbf{x}_t^{(i)}), \tag{A.1}$$

with $\mathbf{D}_{t+1} \subset \mathbb{R}^{p\times f}$ *obtained from* (3.5)*, guarantees that the training error strictly decreases in each iteration:*

$$\sum_{i=1}^N \|\mathbf{x}_*^{(i)} - \mathbf{x}_{t+1}^{(i)}\|_2^2 < \sum_{i=1}^N \|\mathbf{x}_*^{(i)} - \mathbf{x}_t^{(i)}\|_2^2. \tag{A.2}$$

Moreover, if $\hat{\mathbf{D}}\mathbf{h}^{(i)}$ *is* $\mathcal{M}^{++}(\mathbf{x}_*^{(i)})$ *and* (H, L)*-*$\mathcal{RL}(\mathbf{x}_*^{(i)})$*, then the training error converges to zero. If* $H = 0$ *then the training error converges to zero linearly.*

Proof. First, we show the case of strictly-monotone-at-a-point ($\mathcal{M}^+$) functions. For simplicity, we denote $\mathbf{x}_{t+1}^{(i)}$ and $\mathbf{x}_t^{(i)}$ as $\mathbf{x}_+^{(i)}$ and $\mathbf{x}^{(i)}$, *resp*. We assume that not all $\mathbf{x}_*^{(i)} = \mathbf{x}^{(i)}$, otherwise all $\mathbf{x}^{(i)}$ are already at their stationary points. Thus, there exists an i such that $(\mathbf{x}^{(i)} - \mathbf{x}_*^{(i)})^\top \hat{\mathbf{D}}\mathbf{h}^{(i)}(\mathbf{x}^{(i)}) > 0$. We need to show that

$$\sum_{i=1}^N \|\mathbf{x}_*^{(i)} - \mathbf{x}_+^{(i)}\|_2^2 < \sum_{i=1}^N \|\mathbf{x}_*^{(i)} - \mathbf{x}^{(i)}\|_2^2. \tag{A.3}$$

This can be shown by letting $\bar{\mathbf{D}} = \alpha\hat{\mathbf{D}}$ where:

$$\alpha = \frac{\beta}{\gamma}, \tag{A.4}$$

$$\beta = \sum_{i=1}^{N} (\mathbf{x}^{(i)} - \mathbf{x}_*^{(i)})^\top \hat{\mathbf{D}}\mathbf{h}^{(i)}(\mathbf{x}^{(i)}), \tag{A.5}$$

$$\gamma = \sum_{i=1}^{N} \|\hat{\mathbf{D}}\mathbf{h}^{(i)}(\mathbf{x}^{(i)})\|_2^2. \tag{A.6}$$

Since there exists an i such that $(\mathbf{x}^{(i)} - \mathbf{x}_*^{(i)})^\top \hat{\mathbf{D}}\mathbf{h}(\mathbf{x}^{(i)}) > 0$, both β and γ are positive, and thus α is also positive. Now, we show that the training error decreases in each iteration as follows:

$$\sum_{i=1}^{N} \|\mathbf{x}_*^{(i)} - \mathbf{x}_+^{(i)}\|_2^2 = \sum_{i=1}^{N} \|\mathbf{x}_*^{(i)} - \mathbf{x}^{(i)} + \mathbf{D}_{t+1}\mathbf{h}^{(i)}(\mathbf{x}^{(i)})\|_2^2 \tag{A.7}$$

$$\leq \sum_{i=1}^{N} \|\mathbf{x}_*^{(i)} - \mathbf{x}^{(i)} + \bar{\mathbf{D}}\mathbf{h}^{(i)}(\mathbf{x}^{(i)})\|_2^2 \tag{A.8}$$

$$= \sum_{i=1}^{N} \|\mathbf{x}_*^{(i)} - \mathbf{x}^{(i)}\|_2^2 + \tag{A.9}$$

$$+ \underbrace{\sum_{i=1}^{N} \|\alpha\hat{\mathbf{D}}\mathbf{h}^{(i)}(\mathbf{x}^{(i)})\|_2^2}_{\alpha^2\gamma} +$$

$$+ 2\alpha \underbrace{\sum_{i=1}^{N} (\mathbf{x}_*^{(i)} - \mathbf{x}^{(i)})^\top \hat{\mathbf{D}}\mathbf{h}^{(i)}(\mathbf{x}^{(i)})}_{=-\beta}$$

$$= \sum_{i=1}^{N} \|\mathbf{x}_*^{(i)} - \mathbf{x}^{(i)}\|_2^2 + \alpha^2\gamma - 2\alpha\beta \tag{A.10}$$

$$= \sum_{i=1}^{N} \|\mathbf{x}_*^{(i)} - \mathbf{x}^{(i)}\|_2^2 + \frac{\beta^2}{\gamma} - 2\frac{\beta^2}{\gamma} \tag{A.11}$$

$$= \sum_{i=1}^{N} \|\mathbf{x}_*^{(i)} - \mathbf{x}^{(i)}\|_2^2 - \underbrace{\frac{\beta^2}{\gamma}}_{>0} \tag{A.12}$$

$$< \sum_{i=1}^{N} \|\mathbf{x}_*^{(i)} - \mathbf{x}^{(i)}\|_2^2. \tag{A.13}$$

Eq. A.8 is due to $\mathbf{D}_{t+1}$ being the optimal matrix that minimizes the squared error. Note that Thm. 1 does not guarantee that the error of each sample i reduces in each iteration, but guarantees the reduction in the average error.

For the case of strongly-monotone-at-a-point ($\mathcal{M}^{+}+$) functions, we make an additional $\mathcal{RL}$ assumption that there exist $H \geq 0, L > 0$ such that $\|\hat{\mathbf{D}}\mathbf{h}^{(i)}(\mathbf{x}^{(i)})\|_2^2 \leq H + L\|\mathbf{x}_*^{(i)} - \mathbf{x}^{(i)}\|_2^2$ for all $\mathbf{x}^{(i)}$ and i. Thus, we have

$$\beta = \sum_{i=1}^{N} (\mathbf{x}^{(i)} - \mathbf{x}_*^{(i)})^\top \hat{\mathbf{D}}\mathbf{h}^{(i)}(\mathbf{x}^{(i)}) \geq m \sum_{i=1}^{N} \|\mathbf{x}^{(i)} - \mathbf{x}_*^{(i)}\|_2^2, \tag{A.14}$$

$$\gamma = \sum_{i=1}^{N} \|\hat{\mathbf{D}}\mathbf{h}^{(i)}(\mathbf{x}^{(i)})\|_2^2 \leq NH + L \sum_{i=1}^{N} \|\mathbf{x}_*^{(i)} - \mathbf{x}^{(i)}\|_2^2. \tag{A.15}$$

Also, let us denote the training error in iteration t as E_t:

$$E_t = \sum_{i=1}^{N} \|\mathbf{x}_*^{(i)} - \mathbf{x}_t^{(i)}\|_2^2. \tag{A.16}$$

From (A.12), we have

$$E_{t+1} \leq E_t - \frac{\beta^2}{\gamma} \tag{A.17}$$

$$\leq E_t - \frac{m^2 E_t^2}{NH + LE_t} \tag{A.18}$$

$$= \left(1 - \frac{m^2 E_t}{NH + LE_t}\right) E_t. \tag{A.19}$$

Recursively applying the above inequality, we have

$$E_{t+1} \leq E_0 \prod_{l=0}^{t} \left(1 - \frac{m^2 E_l}{NH + LE_l}\right). \tag{A.20}$$

Next, we will show the following result:

$$\lim_{t \to \infty} E_{t+1} = 0 \tag{A.21}$$

This can be shown by contradiction. Suppose E_t converges to some positive number $\mu > 0$. Since $\{E_t\}_t$ is a decreasing sequence (A.13), we have that $E_0 > E_t \geq \mu$ for all $t > 0$. This

means

$$0 \leq 1 - \frac{m^2 E_t}{NH + LE_t} < 1 - \frac{m^2 \mu}{NH + LE_0} < 1. \tag{A.22}$$

By recursively multiplying (A.22), we have

$$\lim_{t\to\infty} \prod_{l=0}^{t} \left(1 - \frac{m^2 E_l}{NH + LE_l}\right) \leq \lim_{t\to\infty} \left(1 - \frac{m^2 \mu}{NH + LE_0}\right)^{t+1} \tag{A.23}$$

$$= 0. \tag{A.24}$$

Combining (A.24) and (A.20), we have

$$\lim_{t\to\infty} E_{t+1} \leq E_0 \lim_{t\to\infty} \prod_{l=0}^{t} \left(1 - \frac{m^2 E_l}{NH + LE_l}\right) = 0. \tag{A.25}$$

This contradicts our assumption that $\{E_t\}_t$ converges to $\mu > 0$. Thus, the training error converges to zero.

Next, we consider the case where $H = 0$. In this case, in (A.18), we will have

$$E_{t+1} \leq E_t - \frac{m^2 E_t^2}{LE_t} \tag{A.26}$$

$$= \left(1 - \frac{m^2}{L}\right) E_t. \tag{A.27}$$

Recursively applying the above inequality, we have

$$E_{t+1} \leq \left(1 - \frac{m^2}{L}\right)^{t+1} E_0. \tag{A.28}$$

This proves that the training error converges linearly to zero. □

A.2 Proof of Prop. 2

Proposition 2. (Pseudomonotonicity and monotonicity at a point) *If a function* $\mathbf{f} : \mathbb{R}^p \to \mathbb{R}^p$ *is pseudomonotone (resp., strictly pseudomonotone, strongly pseudomonotone) and* $\mathbf{f}(\mathbf{x}_*) = \mathbf{0}_p$, *then* $\mathbf{f}$ *is monotone (resp., strictly monotone, strongly monotone) at* $\mathbf{x}_*$.

Proof. We will show the case of a pseudomonotone $\mathbf{f}$. From Def. 7, let $\mathbf{x}' = \mathbf{x}_*$, then we have

$$(\mathbf{x} - \mathbf{x}_*)^\top \mathbf{f}(\mathbf{x}_*) = 0, \tag{A.29}$$

which, by the definition of pseudomonotonicity, implies

$$(\mathbf{x} - \mathbf{x}_*)^\top \mathbf{f}(\mathbf{x}) \geq 0, \tag{A.30}$$

for all $\mathbf{x} \in \mathbb{R}^p$. That means $\mathbf{f}$ is monotone at $\mathbf{x}_*$. The proofs for strict and strong cases follow similar steps. □

A.3 Proof of Prop. 4

Proposition 4. (Pseudomonotone multivalued map and monotonicity at a point) *Suppose a multivalued map* $\mathbf{f}$ *is pseudomonotone (resp., strictly, strongly) with* $\mathbf{0}_p \in \mathbf{f}(\mathbf{x}_*)$. *Let* $\hat{\mathbf{f}}$ *be an induced function of* $\mathbf{f}$. *Then* $\hat{\mathbf{f}}$ *is monotone (resp., strictly, strongly) at* $\mathbf{x}_*$.

Proof. We will show the case of a pseudomonotone $\mathbf{f}$. From Def. 9, let $\mathbf{x}' = \mathbf{x}_*$. Since $\mathbf{0}_p \in \mathbf{f}(\mathbf{x}_*)$, we have

$$(\mathbf{x} - \mathbf{x}_*)^\top \mathbf{0}_p = 0, \tag{A.31}$$

which, by the definition of pseudomonotonicity, implies for all $\mathbf{u} \in \mathbf{f}(\mathbf{x})$

$$(\mathbf{x} - \mathbf{x}_*)^\top \mathbf{u} \geq 0, \tag{A.32}$$

for all $\mathbf{x} \in \mathbb{R}^p$. Since $\hat{\mathbf{f}}(\mathbf{x}) \in \mathbf{f}(\mathbf{x})$ for all $\mathbf{x}$, we have that $\hat{\mathbf{f}}$ is monotone at $\mathbf{x}_*$. The proofs for strict and strong cases follow similar steps. □

A.4 Proof of Prop. 5

Proposition 5. (Convergence of the training error with an unknown differentiable cost function) *Given a training set* $\{(\mathbf{x}_0^{(i)}, \mathbf{x}_*^{(i)}, \{\mathbf{g}_j^{(i)}\}_{j=1}^{J_i})\}_{i=1}^N$, *where* $\mathbf{x}_0^{(i)}, \mathbf{x}_*^{(i)} \in \mathbb{R}^p$ *and* $\mathbf{g}_j^{(i)} : \mathbb{R}^p \to \mathbb{R}^d$ *differentiable, if there exists a function* $\varphi : \mathbb{R}^d \to \mathbb{R}$ *such that for each* i, $\frac{1}{J_i}\sum_{j=1}^{J_i} \varphi(\mathbf{g}_j^{(i)}(\mathbf{x}^{(i)}))$ *is differentiable strictly pseudoconvex with the minimum at* $\mathbf{x}_*^{(i)}$, *then the training error of DO with* $\mathbf{h}$ *from* (3.29) *strictly decreases in each iteration. Alternatively, if* $\frac{1}{J_i}\sum_{j=1}^{J_i} \varphi(\mathbf{g}_j^{(i)}(\mathbf{x}^{(i)}))$ *is differentiable strongly pseudoconvex with Lipschitz continuous gradient, then the training error of DO converges to zero linearly.*

Proof. Define the product[1] $\mathbf{Dh}$ between $\mathbf{D} : \mathbb{R}^d \times \{1, \ldots, d\} \to \mathbb{R} \times \{1, \ldots, p\}$ and $\mathbf{h} : \mathbb{R}^d \times$

[1] We can think of $\mathbf{D}$ and $\mathbf{h}$ as those from (3.28) and (3.29), *resp*.

$\{1,\ldots,d\} \times \{1,\ldots,p\} \times \mathbb{R}^p \to \mathbb{R}$ that results in a vector in $\mathbb{R}^p$ as:

$$\mathbf{Dh}(\mathbf{x}) = \begin{bmatrix} \sum_{k=1}^{d} \int_V \mathbf{D}(\mathbf{v},k)\mathbf{h}(\mathbf{v},k,1;\mathbf{x})d\mathbf{v} \\ \sum_{k=1}^{d} \int_V \mathbf{D}(\mathbf{v},k)\mathbf{h}(\mathbf{v},k,2;\mathbf{x})d\mathbf{v} \\ \vdots \\ \sum_{k=1}^{d} \int_V \mathbf{D}(\mathbf{v},k)\mathbf{h}(\mathbf{v},k,p;\mathbf{x})d\mathbf{v} \end{bmatrix} \in \mathbb{R}^p, \tag{A.33}$$

for $V = \mathbb{R}^d$. Let $\Phi^{(i)} = \frac{1}{J_i}\sum_{j=1}^{J_i} \varphi(\mathbf{g}^{(i)}(\mathbf{x}))$. We divide the proof into two cases:

Case 1: Differentiable strictly pseudoconvex $\Phi^{(i)}$.

Since $\Phi^{(i)}$ is differentiable strictly pseudoconvex, by Prop. 1, its gradient $\nabla\Phi^{(i)}$ is strictly pseudomonotone. Also, since $\Phi^{(i)}$ has a minimum at $\mathbf{x}_*^{(i)}$, we have $\nabla\Phi^{(i)}(\mathbf{x}_*^{(i)}) = \mathbf{0}_p$. By Prop. 2, this means that $\nabla\Phi^{(i)}$ is strictly monotone at $\mathbf{x}_*^{(i)}$. If we use $\mathbf{h}$ from (3.29) and set $\hat{\mathbf{D}}(\mathbf{v},k)$ to $\left[\frac{\partial}{\partial \mathbf{v}}\varphi(\mathbf{v})\right]_k$, then we have that $\hat{\mathbf{D}}\mathbf{h}^{(i)} = \nabla\Phi^{(i)}$, meaning $\hat{\mathbf{D}}\mathbf{h}^{(i)}$ is strictly monotone at $\mathbf{x}_*^{(i)}$. Thus, by Thm. 1, we have that the training error of DO strictly decreases in each iteration[2].

Case 2: Differentiable strongly pseudoconvex $\Phi^{(i)}$.

The proof is similar to Case 1, but a differentiable strongly pseudoconvex $\Phi^{(i)}$ will have $\hat{\mathbf{D}}\mathbf{h}^{(i)} = \nabla\Phi^{(i)}$ which is strongly monotone at $\mathbf{x}_*^{(i)}$. Since $\nabla\Phi^{(i)} = \hat{\mathbf{D}}\mathbf{h}^{(i)}$ is Lipschitz continuous and $\nabla\Phi^{(i)}(\mathbf{x}_*^{(i)}) = \mathbf{0}_p$, this means

$$\|\hat{\mathbf{D}}\mathbf{h}^{(i)}(\mathbf{x}^{(i)})\|_2 \leq L\|\mathbf{x}^{(i)} - \mathbf{x}_*^{(i)}\|_2, \tag{A.34}$$

where L is the Lipschitz constant. Thus, by Thm. 1, we have that the training error of DO converges to zero linearly.

□

A.5 Proof of Prop. 6

Proposition 6. (Convergence of the training error with an unknown nondifferentiable convex cost function) *Given a training set* $\{(\mathbf{x}_0^{(i)}, \mathbf{x}_*^{(i)}, \{\mathbf{g}_j^{(i)}\}_{j=1}^{J_i})\}_{i=1}^{N}$*, where* $\mathbf{x}_0^{(i)}, \mathbf{x}_*^{(i)} \in \mathbb{R}^p$ *and* $\mathbf{g}_j^{(i)} : \mathbb{R}^p \to \mathbb{R}^d$ *differentiable, if there exists a function* $\varphi : \mathbb{R}^d \to \mathbb{R}$ *such that for each* i*,* $\frac{1}{J_i}\sum_{j=1}^{J_i} \varphi(\mathbf{g}_j^{(i)}(\mathbf{x}^{(i)}))$ *is strictly convex with the minimum at* $\mathbf{x}_*^{(i)}$*, then the training error of DO with* $\mathbf{h}$ *from* (3.29) *strictly decreases in each iteration. Alternatively, if* $\frac{1}{J_i}\sum_{j=1}^{J_i} \varphi(\mathbf{g}_j^{(i)}(\mathbf{x}^{(i)}))$ *is strongly convex with the minimum at* $\mathbf{x}_*^{(i)}$ *and there exist* $L > 0, H \geq 0$ *such that* $\frac{1}{J_i}\sum_{j=1}^{J_i} \bar{\varphi}(\mathbf{g}_j^{(i)}(\mathbf{x}^{(i)}))$

[2]While $\mathbf{Dh}$ in Thm. 1 is a matrix-vector product, the proof of Thm. 1 does not require $\mathbf{D}$ and $\mathbf{h}$ to be a matrix and a vector in general. Rather, it only requires the product $\mathbf{Dh}$ (for any form of $\mathbf{D}$ and $\mathbf{h}$) to return a vector in $\mathbb{R}^p$. This property is satisfied in this case, thus we can apply Thm. 1 as a part of this proof.

is (H, L)*-*$\mathcal{RL}(\mathbf{x}_*^{(i)})$ *for all* $i, \mathbf{x}^{(i)}$*, where* $\bar{\varphi}$ *is any induced function of* $\partial\varphi$*, then the training error of DO converges to zero.*

Proof. Let $\Phi^{(i)} = \frac{1}{J_i}\sum_{j=1}^{J_i}\varphi(\mathbf{g}^{(i)}(\mathbf{x}))$ and let $\mathbf{Dh}$ be defined as in (A.33). We divide the proof into two cases:

Case 1: Strictly convex $\Phi^{(i)}$.

The subdifferential of $\Phi^{(i)}$ is a multivalued map [85]:

$$\partial\Phi^{(i)} = \frac{1}{J_i}\partial\sum_{j=1}^{J_i}\varphi(\mathbf{g}_j^{(i)}(\mathbf{x})) = \frac{1}{J_i}\sum_{j=1}^{J_i}\frac{\partial\mathbf{g}_j^{(i)}(\mathbf{x})}{\partial\mathbf{x}}\partial\varphi(\mathbf{g}_j^{(i)}(\mathbf{x})), \tag{A.35}$$

where ∂ denotes subdifferential. If we use $\mathbf{h}$ from (3.29) and set $\hat{\mathbf{D}}(\mathbf{v}, k)$ to $[\bar{\varphi}(\mathbf{v})]_k$, we have that $\hat{\mathbf{D}}\mathbf{h}^{(i)}(\mathbf{x}) \in \partial\Phi^{(i)}(\mathbf{x})$.

From Prop. 3, we know that $\partial\Phi^{(i)}$ is a strictly monotone multivalued map, which implies it is also strictly pseudomonotone [50]. Since $\Phi^{(i)}$ is strictly convex and has a global minimum at $\mathbf{x}_*^{(i)}$, we know that $\partial\Phi^{(i)}$ has zero only at $\mathbf{x}_*^{(i)}$. Then, by Prop. 4, $\hat{\mathbf{D}}\mathbf{h}^{(i)}$ is strictly monotone at $\mathbf{x}_*^{(i)}$. Thus, by Thm. 1, we have that the training error of DO strictly decreases in each iteration.

Case 2: Strongly convex $\Phi^{(i)}$.

The proof is similar to Case 1, but strongly convex $\Phi^{(i)}$ will have $\hat{\mathbf{D}}\mathbf{h}^{(i)}(\mathbf{x}) \in \partial\Phi^{(i)}(\mathbf{x})$ which is strongly monotone at $\mathbf{x}_*^{(i)}$. By the *relaxed Lipschitz at a point* assumption ($\mathcal{RL}$), we also have that $\|\hat{\mathbf{D}}\mathbf{h}^{(i)}(\mathbf{x}^{(i)})\|_2^2 \leq H + L\|\mathbf{x}_*^{(i)} - \mathbf{x}_t^{(i)}\|_2^2$ for all $i, \mathbf{x}^{(i)}$. Thus, by Thm. 1, we have that the training error of DO converges to zero. □

Note that many useful convex functions have subgradients that follow *relaxed Lipschitz at a point* assumption ($\mathcal{RL}$). Examples of such functions include differentiable functions with Lipschitz gradient, *e.g.*, squared ℓ_2 norm, and functions which are point-wise maximum of a finite number of affine functions, *e.g.*, ℓ_1 norm. Note that, however, a function which is a point-wise maximum of a finite number of affine functions is not strongly monotone at its minimum since its subgradients are bounded by a constant.

Appendix B

Details and Analysis of ICDO for PCReg

In this appendix, we provide the details of ICDO from Section 6.1.4 which are specific to the task of rigid point cloud registration (PCReg). First, we show how to derive the feature function where its dimension is independent of the point cloud's dimension. Then, we discuss the computational complexity of ICDO for PCReg, followed by an analysis on what the maps learn.

B.1 Deriving the feature function

In this section, we describe how to derive the feature function $\mathbf{h}$ based on the gradient of (6.16), reproduced here for convenience:

$$\underset{\tilde{\mathbf{x}}}{\text{minimize}}\ J(\tilde{\mathbf{x}}) = \sum_{a=1}^{N_M} \sum_{b=1}^{N_S} \varphi(\|\mathcal{T}(\mathbf{m}_a; \tilde{\mathbf{x}}) - \mathcal{T}(\mathbf{s}_b; \mathbf{x})\|) \tag{B.1}$$

We parametrize rotations with axis-angle vectors in $\mathbb{R}^3$, but the derivation can also be used with other parametrizations. The steps to derive $\mathbf{h}$ is similar to those in Section 3.2.2. First, we take the cost's derivative and represent it as an inner product between two functions. Then, we discretize the functions into a feature vector $\mathbf{h}$ and a matrix $\mathbf{D}$, allowing us to learn $\mathbf{D}$ from training data. The details are as follows.

Define $\mathbf{g}$ to be the following residual function

$$\mathbf{g}_{ab}(\tilde{\mathbf{x}}; \mathbf{x}) = T(\mathbf{m}_a; \tilde{\mathbf{x}}) - T(\mathbf{s}_b; \mathbf{x}), \tag{B.2}$$

where $\mathbf{x}$ is the current parameter estimate. Thus, (B.1) is modified into

$$\underset{\tilde{\mathbf{x}}\in\mathbb{R}^6}{\text{minimize}}\ J(\tilde{\mathbf{x}}) = \sum_{a=1}^{N_M}\sum_{b=1}^{N_S}\varphi(\|\mathbf{g}_{ab}(\tilde{\mathbf{x}};\mathbf{x})\|). \tag{B.3}$$

Next, we compute $\Delta\mathbf{x} = -\frac{\partial J(\tilde{\mathbf{x}})}{\partial\tilde{\mathbf{x}}}$ at $\tilde{\mathbf{x}} = \mathbf{0}_6$. For simplicity, we consider a single term (a, b):

$$\begin{aligned}\Delta\mathbf{x}_{ab} &\triangleq -\left.\frac{\partial}{\partial\tilde{\mathbf{x}}}\varphi(\|\mathbf{g}_{ab}(\tilde{\mathbf{x}};\mathbf{x})\|)\right|_{\tilde{\mathbf{x}}=\mathbf{0}_6} \\ &= -\underbrace{\begin{bmatrix}-[\mathbf{m}_a]_\times \\ \mathbf{I}_3\end{bmatrix}\frac{\mathbf{g}_{ab}(\mathbf{0}_6;\mathbf{x})}{\|\mathbf{g}_{ab}(\mathbf{0}_6;\mathbf{x})\|}}_{=\mathbf{w}_{ab}}\left.\frac{\partial\varphi(\|\mathbf{g}_{ab}(\tilde{\mathbf{x}};\mathbf{x})\|)}{\partial\|\mathbf{g}_{ab}(\tilde{\mathbf{x}};\mathbf{x})\|}\right|_{\tilde{\mathbf{x}}=\mathbf{0}_6},\end{aligned} \tag{B.4}$$

where $[\cdot]_\times$ represents cross product operation in a matrix form. We can see that only the rightmost term is dependent on φ. Since we assume we do not have access to φ, we will learn this term from training data using the algorithm in Section 6.1.3. To do so, we need to express $\Delta\mathbf{x}$ as $\mathbf{Dh}$ with φ placed inside $\mathbf{D}$ and other information placed inside $\mathbf{h}$. We do this by replacing $\frac{\partial}{\partial\mathbf{y}}\varphi(\mathbf{y})$ with a function $\phi : \mathbb{R}\to\mathbb{R}$, then factorizing it as the convolution with Dirac delta function δ:

$$\Delta\mathbf{x}_{ij} = -\mathbf{w}_{ab}\phi(\|\mathbf{g}_{ab}(\mathbf{0}_6;\mathbf{x})\|) \tag{B.5}$$

$$= -\mathbf{w}_{ab}\int_V \phi(v)\delta(v - \|\mathbf{g}_{ab}(\mathbf{0}_6;\mathbf{x})\|)dv, \tag{B.6}$$

where $V = \mathbb{R}$. Consider only an element l of $\Delta\mathbf{x}_{ab}$, we see

$$[\Delta\mathbf{x}_{ab}]_l = -\int_V [\mathbf{w}_{ab}]_l\phi(v)\delta(v - \|\mathbf{g}_{ab}(\mathbf{0}_6;\mathbf{x})\|)dv. \tag{B.7}$$

We can see that (B.7) is an inner product between $-\phi(v)$ and $[\mathbf{w}_{ab}]_l\delta(v - \|\mathbf{g}_{ab}(\mathbf{0}_6;\mathbf{x})\|)$. This is similar to $[\mathbf{Dh}]_l$, which can be considered as the inner product between $\mathbf{h}$ and row l of $\mathbf{D}$. Following this connection, we express the product between $\phi(v)$ and $-[\mathbf{w}_{ab}]_l\delta(v-\|\mathbf{g}_{ab}(\mathbf{0}_6;\mathbf{x})\|)$ as a matrix-vector product $[\mathbf{Dh}]_l$. To do so, we discretize the space V into q boxes, leading to (B.7)'s discretized counterpart:

$$[\Delta\mathbf{x}_{ab}]_l \approx -\boldsymbol{\phi}^\top[\mathbf{w}_{ab}]_l\mathbf{e}_{\gamma_{q,r}(\|\mathbf{g}_{ij}(\mathbf{0}_6;\mathbf{x})\|)}, \tag{B.8}$$

where $\gamma_{q,r} : \mathbb{R}\to\{0, 1, \ldots, r\}$ discretizes the segment $[0, q]$ into r boxes[1]; δ is discretized into

[1]This differs from $\gamma_{q,r}$ in (3.16) which discretizes the segment $[-q, q]$.

the standard basis vector $\mathbf{e}_\beta \in \{0,1\}^r$ (recall that we define $\mathbf{e}_0 = \mathbf{0}_r$); and ϕ is discretized into a vector $\boldsymbol{\phi} \in \mathbb{R}^r$. With these discretizations, we can put everything back to the full $\Delta\mathbf{x}$ as

$$\Delta\mathbf{x} = \sum_{a=1}^{N_M}\sum_{b=1}^{N_S} \Delta\mathbf{x}_{ab} \approx \mathbf{D}\mathbf{h}(\mathbf{x};\mathbf{M},\mathbf{S}), \tag{B.9}$$

$$\mathbf{D} = \mathbf{I}_6 \otimes \boldsymbol{\phi}^\top \tag{B.10}$$

$$\mathbf{h}(\mathbf{x};\mathbf{M},\mathbf{S}) = -\sum_{a=1}^{N_M}\sum_{b=1}^{N_S}\bigoplus_{l=1}^{6}[\mathbf{w}_{ab}]_l \mathbf{e}_{\gamma_{q,r}(\|\mathbf{g}_{ab}(\mathbf{0}_6;\mathbf{x})\|)}, \tag{B.11}$$

where $\otimes$ denotes Kronecker product, and $\bigoplus$ denotes vector concatenation. We can see that (B.9) factorizes $\Delta\mathbf{x}$ in (B.4) into a product of two terms: $\mathbf{D} \in \mathbb{R}^{6\times 6r}$ which contains the unknown ϕ, and $\mathbf{h} : \mathbb{R}^6 \times (\mathbb{R}^{N_M} \times \mathbb{R}^{N_S}) \to \mathbb{R}^{6r}$ which contains the known information about the two point clouds. This factorization allows us to use $\mathbf{h}$ as a feature function to learn the update maps with the algorithm in Section 6.1.3.

Our derivation of the feature function differs from that in Section 3.2.2. If we follow Section 3.2.2, we will consider $\hat{\varphi}(\mathbf{g}_{ij}(\tilde{\mathbf{x}};\mathbf{x}))$ with $\hat{\varphi} : \mathbb{R}^3 \to \mathbb{R}$ instead of $\varphi(\|\mathbf{g}_{ij}(\tilde{\mathbf{x}};\mathbf{x})\|)$ with $\varphi : \mathbb{R} \to \mathbb{R}$. Using $\hat{\varphi}$ would allow learning an anisotropic penalty instead of an isotropic one of φ, but the feature $\mathbf{h}$ of $\hat{\varphi}$ will have the dimension of $6r^3$ for 3D cases. This is much larger than $6r$ of (B.11), which is independent of the point cloud's dimension. Moreover, the maps learned from $\hat{\varphi}$ would require a much larger number of training data to prevent overfitting.

B.2 Computational complexity

We can see that the most demanding step of ICDO is the computation of the feature $\mathbf{h}$, which is $\mathcal{O}(N_M N_S)$ due to the pairwise residual $\mathbf{g}_{ab}$. This is equivalent to straightforward implementations of other PCReg algorithms, as they all require computing the pairwise distances. However, ICP can use kd-tree to find the nearest neighbors, which reduces the complexity to $\mathcal{O}(N_M \log N_S)$. Similarly, Gaussian-based approaches, such as CPD [73], KC [96], and GMMReg [55], can use fast Gauss transform (FGT) [49] to compute the Gaussian kernels, which reduces the complexity to $\mathcal{O}(N_M + N_S)$. Unfortunately, the function learned by ICDO can be more general and we do not know of a way to improve its complexity.

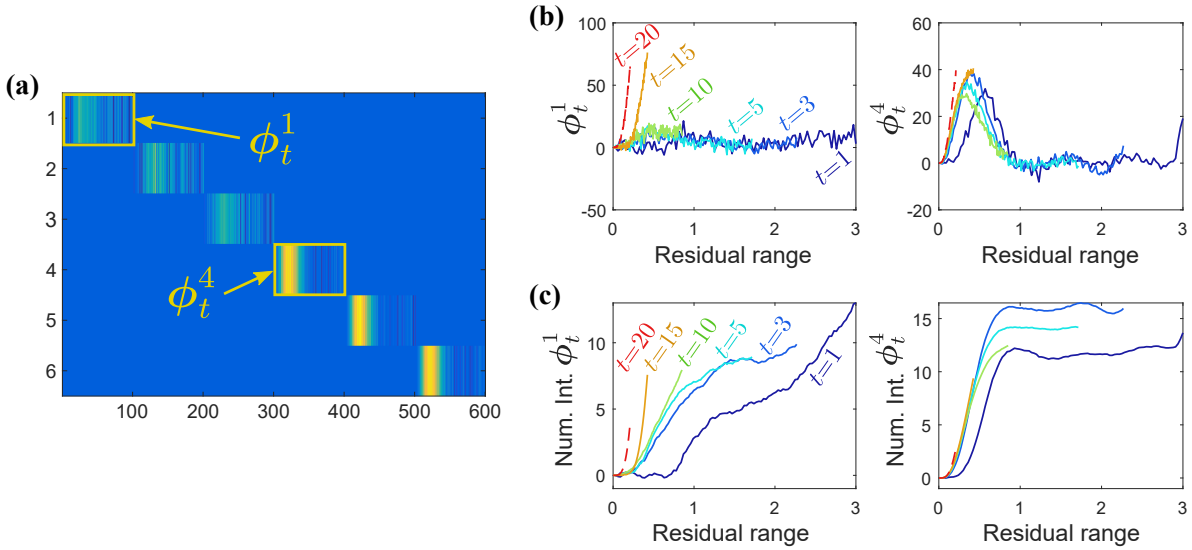


Figure B.1: A visualization of the maps $\mathbf{D}_t$. (a) The learned matrix $\mathbf{D}_5$ (blue - low value, yellow - high value). (b) Plots of the diagonal blocks $\boldsymbol{\phi}_t^1$ and $\boldsymbol{\phi}_t^4$ of $\mathbf{D}_t$ for different t, where we align each element in the vectors to the residual range $[0, q]$ they represent. Note that the length in x-axis of each vector differs since q decreases as t increases. (c) Numerical integration of $\boldsymbol{\phi}_t^1$ and $\boldsymbol{\phi}_t^4$ from (b).

B.3 Analyzing the maps

Figure B.1a shows $\mathbf{D}_5$ as an example of the learned maps. Here, $\boldsymbol{\phi}_t^b$ denotes the vector in the diagonal block b of the map $\mathbf{D}_t$. We observe that $\boldsymbol{\phi}_t^1$, $\boldsymbol{\phi}_t^2$, and $\boldsymbol{\phi}_t^3$ which map to the update in rotation $\mathbf{r}$ are similar, while $\boldsymbol{\phi}_t^4$, $\boldsymbol{\phi}_t^5$, and $\boldsymbol{\phi}_t^6$ for translation $\mathbf{t}$ are also similar. This is because the distribution of the residuals is isotropic. Since the maps of the same type are similar, we visualize $\boldsymbol{\phi}_t^1$ and $\boldsymbol{\phi}_t^4$ of different maps t in Figure B.1b. We can see that the peaks of the curves move toward 0 as t increases. Since we can interpret the maps as imitating a gradient map (Section 3.2.3), we also show the numerical integration of $\boldsymbol{\phi}_t^1$ and $\boldsymbol{\phi}_t^4$ in Figure B.1c, where we can see the functions squeeze closer to 0. These visualizations indicate that ICDO is learning an annealing schedule for PCReg from training data, unlike previous work, *e.g.*, GMMReg, which needs to set the schedule manually. Note that since the maps of rotation and translation are different, the vector fields of the updates cannot be integrated into a single cost function[2].

[2]We tried to learn a shared $\boldsymbol{\phi}$ for all rotation and translation that allows numerical integration to a cost function, but its result was not good.

Bibliography

[1] A. Agarwal, C. V. Jawahar, and P. J. Narayanan. A survey of planar homography estimation techniques. Technical report, International Institute of Information Technology, India, 2005.

[2] E. Antonakos, P. Snape, G. Trigeorgis, and S. Zafeiriou. Adaptive cascaded regression. In *Proc. Int. Conf. Image Process.*, 2016.

[3] S. Avidan. Support vector tracking. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(8):1064–1072, 2004.

[4] J. A. Bagnell. An invitation to imitation. Technical report, Carnegie Mellon University, 2015.

[5] S. Baker and I. Matthews. Lucas-Kanade 20 years on: A unifying framework. *Int. J. Comput. Vision*, 56(3):221–255, 2004.

[6] J. T. Barron. A more general robust loss function. *arXiv preprint arXiv:1701.03077*, 2017.

[7] R. Basri, L. Costa, D. Geiger, and D. Jacobs. Determining the similarity of deformable shapes. *Vision Res.*, 38(15-16):2365–2385, 1998.

[8] R. Basri, D. Jacobs, and I. Kemelmacher. Photometric stereo with general, unknown lighting. *Int. J. Comput. Vision*, 72(3):239–257, 2007.

[9] E. Bayro-Corrochano and J. Ortegn-Aguilar. Lie algebra approach for tracking and 3D motion estimation using monocular vision. *Image and Vision Computing*, 25(6):907921, 2007.

[10] E. Bayro-Corrochano and J. Ortegon-Aguilar. Lie algebra template tracking. In *Proc. Int. Conf. Pattern Recognition*, 2004.

[11] E. Bayro-Corrochano and J. Ortegón-Aguilar. Lie algebra approach for tracking and 3d motion estimation using monocular vision. *Image and Vision Computing*, 25(6):907–921, 2007.

[12] A. Ben-Israel and B. Mond. What is invexity? *The ANZIAM Journal*, 28(1):1–9, 1986.

[13] P. Bergström and O. Edlund. Robust registration of point sets using iteratively reweighted least squares. *Computational Optimization and Applicat.*, 58(3):543–561, 2014.

[14] M. Bertero, T. A. Poggio, and V. Torre. Ill-posed problems in early vision. *Proc. IEEE*, 76(8):869–889, Aug 1988.

[15] P. J. Besl and H. D. McKay. A method for registration of 3-D shapes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 14(2):239–256, 1992.

[16] M. Black and A. Rangarajan. On the unification of line processes, outlier rejection, and robust statistics with applications in early vision. *Int. J. Comput. Vision*, 19(1):57–91, 1996.

[17] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[18] S. Boyd, L. Xiao, and A. Mutapcic. Subgradient methods. Lecture Notes of EE392o, Stanford University, 2003.

[19] S. Bubeck. Convex optimization: Algorithms and complexity. *Foundations and Trends in Mach. Learning*, 8(3-4):231–357, 2015.

[20] R. Cabral, F. De la Torre, J. P. Costeira, and A. Bernardino. Unifying nuclear norm and bilinear factorization approaches for low-rank matrix decomposition. In *Proc. Int. Conf. Comput. Vision*, 2013.

[21] D. Campbell and L. Petersson. An adaptive data representation for robust point-set registration and merging. In *Proc. Int. Conf. Comput. Vision*, 2015.

[22] X. Cao, Y. Wei, F. Wen, and J. Sun. Face alignment by explicit shape regression. In *Proc. Conf. Comput. Vision and Pattern Recognition*, 2012.

[23] A. Chambolle, V. Caselles, M. Novaga, D. Cremers, and T. Pock. An introduction to Total Variation for Image Analysis. working paper or preprint, Nov. 2009. URL `https://hal.archives-ouvertes.fr/hal-00437581`.

[24] T. Chan, S. Esedoglu, F. Park, and A. Yip. Total variation image restoration: Overview and recent developments. In N. Paragios, Y. Chen, and O. Faugeras, editors, *Handbook of Mathematical Models in Computer Vision*, pages 17–31. Springer US, Boston, MA, 2006.

[25] C.-C. Chang and C.-J. Lin. Training v-support vector regression: Theory and algorithms. *Neural Computation*, 14(8):1959–1977, 2002.

[26] A. Collet, D. Berenson, S. S. Srinivasa, and D. Ferguson. Object recognition and full pose registration from a single image for robotic manipulation. In *Int. Conf. Robotics and Automation*, 2009.

[27] T. F. Cootes, C. J. Taylor, D. H. Cooper, and J. Graham. Active shape models-their training and application. *Comput. Vision and Image Understanding*, 61(1):38–59, 1995.

[28] T. F. Cootes, G. J. Edwards, and C. J. Taylor. Active appearance models. In *Proc. Eur. Conf. Comput. Vision*, 1998.

[29] T. F. Cootes, G. J. Edwards, and C. J. Taylor. Active appearance models. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23(6):681–685, 2001.

[30] J. P. Costeira and T. Kanade. A multibody factorization method for independently moving objects. *Int. J. Comput. Vision*, 29(3):159–179, 1998.

[31] D. Cristinacce and T. F. Cootes. Boosted regression active shape models. In *Proc. British Mach. Vision Conf.*, 2007.

[32] B. Curless and M. Levoy. A volumetric method for building complex models from range images. In *Proc. 23rd Annu. Conf. Comput. Graph. and Interactive Techn.*, pages 303–312. ACM, 1996.

[33] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Proc. Int. Conf. Comput. Vision*, 2005.

[34] A. Del Bue, J. Xavier, L. Agapito, and M. Paladini. Bilinear modeling via augmented lagrange multipliers (BALM). *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(8):1496–1508, 2012.

[35] O. Devolder, F. Glineur, and Y. Nesterov. First-order methods with inexact oracle: the strongly convex case. Technical report, Center for Operations Research and Econometrics, 2013.

[36] O. Devolder, F. Glineur, and Y. Nesterov. First-order methods of smooth convex optimization with inexact oracle. *Math. Programming*, 146(1-2):37–75, 2014.

[37] F. Diego and F. A. Hamprecht. Structured regression gradient boosting. In *Proc. Conf. Comput. Vision and Pattern Recognition*, 2016.

[38] T. G. Dietterich, A. Ashenfelter, and Y. Bulatov. Training conditional random fields via gradient tree boosting. In *Proc. Int. Conf. Mach. Learning*, 2004.

[39] P. Dollár, P. Welinder, and P. Perona. Cascaded pose regression. In *Proc. Conf. Comput. Vision and Pattern Recognition*, 2010.

[40] L. Ferraz, X. Binefa, and F. Moreno-Noguer. Very fast solution to the PnP problem with algebraic outlier rejection. In *Proc. Conf. Comput. Vision and Pattern Recognition*, 2014.

[41] M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting

with applications to image analysis and automated cartography. *Commun. ACM*, 24(6): 381–395, June 1981.

[42] A. Fitzgibbon. Robust registration of 2D and 3D point sets. In *Proc. British Mach. Vision Conf.*, 2001.

[43] J. H. Friedman. Greedy function approximation: A gradient boosting machine. *Ann. Statist.*, 29(5):1189–1232, 2001.

[44] G. Goh. Why momentum really works. *Distill*, 2017. doi: 10.23915/distill.00006. URL `http://distill.pub/2017/momentum`.

[45] S. Gold, A. Rangarajan, C.-P. Lu, P. Suguna, and E. Mjolsness. New algorithms for 2D and 3D point matching: pose estimation and correspondence. *Pattern Recognition*, 38(8): 1019–1031, 1998.

[46] V. Golyanik, S. Aziz Ali, and D. Stricker. Gravitational approach for point set registration. In *Proc. Conf. Comput. Vision and Pattern Recognition*, 2016.

[47] R. C. Gonzalez and R. E. Woods. *Digital Image Processing (3rd Edition)*. Prentice-Hall, Inc., 2006.

[48] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[49] L. Greengard and J. Strain. The fast Gauss transform. *SIAM Journal on Scientific and Statistical Computing*, 12(1):79–94, 1991.

[50] N. Hadjisavvas and S. Schaible. Generalized monotone multivalued maps. In *Encyclopedia of Optimization*, pages 1193–1197. Springer, 2001.

[51] B. Hall. *Lie Groups, Lie Algebras, and Representations: An Elementary Introduction*. Springer, 2004.

[52] M. Harker and P. O'Leary. Least squares surface reconstruction from measured gradient fields. In *Proc. Conf. Comput. Vision and Pattern Recognition*, 2008.

[53] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge university press, 2003.

[54] R. I. Hartley. In defense of the eight-point algorithm. *IEEE Trans. Pattern Anal. Mach. Intell.*, 19(6):580–593, 1997.

[55] B. Jian and B. C. Vemuri. Robust point set registration using gaussian mixture models. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(8):1633–1645, 2011.

[56] F. Jurie and M. Dhome. Hyperplane approximation for template matching. *IEEE Trans.*

Pattern Anal. Mach. Intell., 24(7):996–1000, 2002.

[57] S. Karamardian and S. Schaible. Seven kinds of monotone maps. *J. Optimization and Applicat.*, 66(1):37–46, 1990.

[58] Q. Ke and T. Kanade. Robust l_1 norm factorization in the presence of outliers and missing data by alternative convex programming. In *Proc. Conf. Comput. Vision and Pattern Recognition*, 2005.

[59] L. Kneip, D. Scaramuzza, and R. Siegwart. A novel parametrization of the perspective-three-point problem for a direct computation of absolute camera position and orientation. In *Proc. Conf. Comput. Vision and Pattern Recognition*, 2011.

[60] H. Le, T.-J. Chin, and D. Suter. An exact penalty method for locally convergent maximum consensus. In *Proc. Conf. Comput. Vision and Pattern Recognition*, 2017.

[61] V. Lepetit, F. Moreno-Noguer, and P. Fua. EPnP: An accurate o(n) solution to the PnP problem. *Int. J. Comput. Vision*, 81(2):155–166, 2008.

[62] S. Li, C. Xu, and M. Xie. A robust O(n) solution to the perspective-n-point problem. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(7):1444–1450, 2012.

[63] X. Liu. Discriminative face alignment. *IEEE Trans. Pattern Anal. Mach. Intell.*, 31(11): 1941–1954, 2009.

[64] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, 2004.

[65] D. T. Luc. Generalized convexity and some applications to vector optimization. *Vietnam J. Mathematics*, 26(2):95–110, 1998.

[66] D. G. Luenberger. A combined penalty function and gradient projection method for non-linear programming. *J. Optimization and Applicat.*, 14(5):477–495, 1974.

[67] Y. Ma, S. Soatto, J. Kosecka, and S. S. Sastry. *An Invitation to 3-D Vision*. Springer-Verlag New York, 2004.

[68] L. Mason, J. Baxter, P. Bartlett, and M. Frean. Boosting algorithms as gradient descent. In *Proc. Neural Inform. Process. Syst.*, 1999.

[69] P. Meer, D. Mintz, A. Rosenfeld, and D. Y. Kim. Robust regression methods for computer vision: A review. *Int. J. Comput. Vision*, 6(1):59–70, 1991.

[70] J. Mei and J. M. Moura. SILVar: Single Index Latent Variable Models. *arXiv preprint arXiv:1705.03536*, 2017.

[71] A. Mian, M. Bennamoun, and R. Owens. On the repeatability and quality of keypoints for

local feature-based 3D object retrieval from cluttered scenes. *Int. J. Comput. Vision*, 89 (2):348–361, 2010.

[72] H. Mobahi and J. W. Fisher. Coarse-to-fine minimization of some common nonconvexities. In *Proc. Int. Conf. Energy Minimization Methods in Comput. Vision and Pattern Recognition*, 2015.

[73] A. Myronenko and X. Song. Point set registration: Coherent point drift. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(12):2262–2275, 2010.

[74] Y. Nesterov. *Introductory lectures on convex optimization: A basic course*. Springer, 2004.

[75] A. Ng and S. Russell. Algorithms for inverse reinforcement learning. In *Proc. Int. Conf. Mach. Learning*, 2000.

[76] C. V. Nguyen, S. Izadi, and D. Lovell. Modeling kinect sensor noise for improved 3d reconstruction and tracking. In *Int. Conf. 3D Imaging, Modeling, Process., Visualization and Transmission*, 2012.

[77] M. H. Nguyen and F. De la Torre. Metric learning for image alignment. *Int. J. Comput. Vision*, 88(1):69–84, 2010.

[78] K. Nishino and K. Ikeuchi. Robust simultaneous registration of multiple range images. *Digitally Archiving Cultural Objects*, pages 71–88, 2008.

[79] P. Ochs, A. Dosovitskiy, T. Brox, and T. Pock. An iterated ℓ_1 algorithm for non-smooth non-convex optimization in computer vision. In *Proc. Conf. Comput. Vision and Pattern Recognition*, 2013.

[80] K. Paliouras and A. A. Argyros. Towards the automatic definition of the objective function for model-based 3d hand tracking. In *Man–Machine Interactions 4*, pages 353–363. Springer, 2016.

[81] F. Pomerleau, M. Liu, F. Colas, and R. Siegwart. Challenging data sets for point cloud registration algorithms. *Int. J. Robotics Res.*, 31(14):1705–1711, Dec. 2012.

[82] F. Pomerleau, F. Colas, R. Siegwart, et al. A review of point cloud registration algorithms for mobile robotics. *Foundations and Trends in Robotics*, 4(1):1–104, 2015.

[83] R. T. Rockafellar. *Convex analysis*. Princeton University Press, 1970.

[84] E. Rodolà, A. Albarelli, F. Bergamasco, and A. Torsello. A scale independent selection process for 3d object recognition in cluttered scenes. *Int. J. Comput. Vision*, 102(1-3): 129–145, 2013.

[85] G. Romano. New results in subdifferential calculus with applications to convex optimization. *Appl. Mathematics and Optimization*, 32(3):213–234, 1995.

[86] S. Rusinkiewicz and M. Levoy. Efficient variants of the ICP algorithm. In *Proc. Int. Conf. 3-D Digital Imaging and Modeling*, 2001.

[87] E. K. Ryu and S. Boyd. A primer on monotone operator methods. *Appl. and Computational Mathematics*, 15(1):3–43, 2016.

[88] J. Saragih and R. Goecke. Iterative error bound minimisation for aam alignment. In *Proc. Int. Conf. Pattern Recognition*, 2006.

[89] N. Z. Shor. *Minimization Methods for Non-Differentiable Functions*. Springer-Verlag Berlin, 1985.

[90] Stanford Computer Graphics Laboratory. The stanford 3D scanning repository. `https://graphics.stanford.edu/data/3Dscanrep/`, Aug 2014. Accessed: 2016-08-31.

[91] C. V. Stewart. Robust parameter estimation in computer vision. *SIAM Rev.*, 41(3):513–537, 1999.

[92] Y. Sun, X. Wang, and X. Tang. Deep convolutional network cascade for facial point detection. In *Proc. Conf. Comput. Vision and Pattern Recognition*, 2013.

[93] Y. Tian and S. G. Narasimhan. Theory and practice of hierarchical data-driven descent for optimal deformation estimation. *Int. J. Comput. Vision*, 115(1):44–67, 2015.

[94] C. Tomasi and T. Kanade. Shape and motion from image streams under orthography: a factorization method. *Int. J. Comput. Vision*, 9(2):137–154, 1992.

[95] G. Trigeorgis, P. Snape, M. A. Nicolaou, E. Antonakos, and S. Zafeiriou. Mnemonic descent method: A recurrent process applied for end-to-end face alignment. In *Proc. Conf. Comput. Vision and Pattern Recognition*, 2016.

[96] Y. Tsin and T. Kanade. A correlation-based approach to robust point set registration. In *Proc. Eur. Conf. Comput. Vision*, 2004.

[97] O. Tuzel, F. Porikli, and P. Meer. Learning on lie groups for invariant detection and tracking. In *Proc. Conf. Comput. Vision and Pattern Recognition*, 2008.

[98] G. Tzimiropoulos. Project-out cascaded regression with an application to face alignment. In *Proc. Conf. Comput. Vision and Pattern Recognition*, 2015.

[99] J. Vongkulbhisal, F. De la Torre, and J. P. Costeira. Discriminative optimization: Theory and applications to computer vision problems. *arXiv preprint arXiv:1707.04318*, 2017.

[100] J. Vongkulbhisal, F. D. la Torre, and J. P. Costeira. Discriminative optimization: Theory and applications to point cloud registration. In *Proc. Conf. Comput. Vision and Pattern Recognition*, 2017.

[101] T. Werner and A. Zisserman. Model selection for automated architectural reconstruction from multiple views. In *Proc. British Mach. Vision Conf.*, 2002.

[102] T. Werner and A. Zisserman. New techniques for automated architectural reconstruction from photographs. In *Proc. Eur. Conf. Comput. Vision*, 2002.

[103] J. Xie, L. Xu, and E. Chen. Image denoising and inpainting with deep neural networks. In *Proc. Neural Inform. Process. Syst.*, 2012.

[104] X. Xiong and F. De la Torre. Supervised descent method and its application to face alignment. In *Proc. Conf. Comput. Vision and Pattern Recognition*, 2013.

[105] X. Xiong and F. De la Torre. Supervised descent method for solving nonlinear least squares problems in computer vision. *arXiv preprint arXiv:1405.0601*, 2014.

[106] X. Xiong and F. De la Torre. Global supervised descent method. In *Proc. Conf. Comput. Vision and Pattern Recognition*, 2015.

[107] J. Yang, H. Li, D. Campbell, and Y. Jia. Go-ICP: a globally optimal solution to 3D ICP point-set registration. *IEEE Trans. Pattern Anal. Mach. Intell.*, 38(11):2241–2254, 2016.

[108] J.-C. Yao. Multi-valued variational inequalities with k-pseudomonotone operators. *J. Optimization and Applicat.*, 83(2):391–403, 1994.

[109] G. Yuan and B. Ghanem. ℓ_0TV: a new method for image restoration in the presence of impulse noise. In *Proc. Conf. Comput. Vision and Pattern Recognition*, 2015.

[110] Z. Zhang. A flexible new technique for camera calibration. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(11):1330–1334, 2000.

[111] Y. Zheng, Y. Kuang, S. Sugimoto, K. Åström, and M. Okutomi. Revisiting the PnP problem: A fast, general and optimal solution. In *Proc. Int. Conf. Comput. Vision*, 2013.

[112] Q.-Y. Zhou, J. Park, and V. Koltun. Fast global registration. In *Proc. Eur. Conf. Comput. Vision*, 2016.

[113] K. Zimmermann, J. Matas, and T. Svoboda. Tracking by an optimal sequence of linear predictors. *IEEE Trans. Pattern Anal. Mach. Intell.*, 31(4):677–692, 2009.