Discriminative Optimization: Theory and Applications to Computer Vision

Jayakorn Vongkulbhisal, Fernando De la Torre, and João P. Costeira

Abstract—Many computer vision problems are formulated as the optimization of a cost function. This approach faces two main challenges: designing a cost function with a local optimum at an acceptable solution, and developing an efficient numerical method to search for this optimum. 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 place. 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. DO explicitly learns a sequence of updates in the search space that leads to stationary points that correspond to the desired solutions. We provide a formal analysis of DO and illustrate its benefits in the problem of 3D registration, camera pose estimation, and image denoising. We show that DO outperformed or matched state-of-the-art algorithms in terms of accuracy, robustness, and computational efficiency.

Index Terms—Optimization, gradient methods, iterative methods, image processing and computer vision, machine learning.

1 Introduction

Mathematical optimization plays an important role for solving many computer vision problems. For instance, optical flow, camera calibration, homography estimation, and structure from motion are computer vision problems solved as optimization. Formulating computer vision problems as optimization problems faces two main challenges: *(i)* Designing a cost function that has a local optimum that corresponds to a suitable solution, and *(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.

Consider Fig. 1a-top which illustrates a 2D alignment problem in a case of noiseless data. A good cost function for this problem should have a global optimum when the two shapes overlap. Fig. 1b-top illustrates the level sets of the cost function for the Iterative Closest Point (ICP) algorithm [1] in the case of complete and noiseless data. Observe that there is a well-defined optimum and that it coincides with the ground truth. Given a cost function, the next step is to find a suitable algorithm that, given an initial configuration (green square), finds a local optimum. For this particular initialization, the ICP algorithm will converge to the ground truth (red diamond in Fig. 1b-top), and Fig. 1e-top shows the convergence region for ICP in green. However, in realistic scenarios with the presence of perturbations in the data, there is no guarantee that there will be a good local optimum in the expected solution, while the number of local optima can be large. Fig. 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 cost function has changed dramatically: there are more local optima, and they do not necessarily correspond to the ground truth (red diamond). In this case, the ICP algorithm with an initialization in the green square will converge to a wrong optimum. 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 *surroundings* of the optimum and how it will change with noise.

To address the aforementioned problems, this paper proposes Discriminative Optimization (DO). DO exploits the fact that we often know from the training data where the solutions should be, whereas traditional approaches formulate optimization problems based on an ideal model. Rather than following a descent direction of a cost function, DO directly learns a sequence of update directions leading to a stationary point. These points are placed *by design* in the desired solutions from training data. This approach has three 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 and a larger convergence region, as can be seen in Fig. 1e. Second, because DO does not optimize any explicit function (e.g., ℓ_2 registration error), it is less sensitive to model misfit and more robust to different types of perturbations. Fig. 1c illustrates the contour level inferred from the update directions learned by DO. It can be seen that the curve levels have a local optimum on the ground truth and fewer local optima than ICP in Fig. 1b. Fig. 1e shows that the convergence regions of DO change little despite

- *J. Vongkulbhisal is with the ECE Department, Carnegie Mellon University, USA, and the ISR - IST, Universidade de Lisboa, Portugal. E-mail: jvongkul@andrew.cmu.edu.*
- *F. De la Torre is with Facebook Inc. and the Robotics Institute, Carnegie Mellon University, USA. E-mail: ftorre@cs.cmu.edu.*
- *J. P. Costeira is with the ISR - IST, Universidade de Lisboa, Portugal. E-mail: jpc@isr.ist.utl.pt*

Manuscript received April 19, 2005; revised August 26, 2015.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2018.2826536, IEEE Transactions on Pattern Analysis and Machine Intelligence

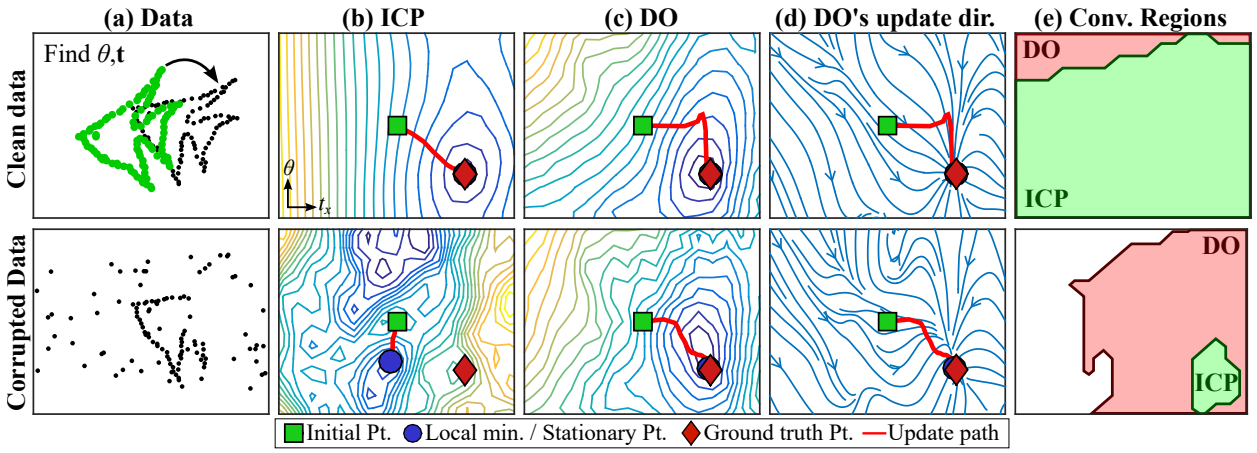


Figure 1. 2D point alignment using ICP and DO. (a) Data. (b) Level sets of the cost function for ICP. We used the optimal matching at each parameter value to compute the ℓ_2 cost. (c) Inferred level sets for the proposed DO, approximately reconstructed with the surface reconstruction algorithm [2] from DO's update directions. (d) The update directions of DO. (e) Regions of convergence for ICP and DO. See text for detailed description.

the perturbations, and always include the regions of ICP. Third, to compute update directions, traditional approaches require the cost function to be differentiable or continuous, whereas DO's directions can always be computed. We also provide a proof of DO's convergence in the training set. We named our approach DO to reflect the idea of learning to find a stationary point directly rather than optimizing a "generative" cost function.

In this work, we study the convergence properties of DO and its relation to generalized convexity. Based on this relation, we propose a framework for designing features where the update directions can be interpreted as the gradient direction of an unknown cost function. We show that our approach can handle both ordered (e.g., image pixels) and unordered (e.g., set of feature matches) data types. We perform a synthetic experiment to confirm our interpretation, and apply DO to the problems of point cloud registration, camera pose estimation, and image denoising, and show that DO can obtain state-of-the-art results.

2 Related Work

2.1 Optimization in computer vision

Many problems in computer vision involve solving inverse problems, that is to estimate the parameters $\mathbf{x} \in \mathbf{R}^p$ that satisfies $\mathbf{g}_j(\mathbf{x}) = \mathbf{0}_d, j = 1, \ldots, J$, where $\mathbf{g}_j : \mathbf{R}^p \rightarrow \mathbf{R}^d$ models the phenomena of interest. For example, in camera pose estimation [3], $\mathbf{x}$ may represent the camera parameters and $\mathbf{g}_j$ the reprojection error of the j^{th} feature match. Optimization-based framework tackles these problems by searching for the parameters $\mathbf{x}_*$ that optimize a certain cost function, which is typically designed to *penalize* the deviation of $\mathbf{g}_j$ from $\mathbf{0}_d$. Since selecting a cost function is a vital step for solving problems, there exists a large amount of research on the robustness properties of different *penalty functions* (e.g., ℓ_2 loss, ℓ_1 loss, etc.) [4], [5], [6]. Instead of using a fixed penalty function, many approaches use continuation methods to deform the function as the optimization is solved [7], [8]. On the other hand, many computer vision problems are ill-posed [9] and require some forms of regularization. This further leads to a combination of different penalty functions and requires setting the value of hyperparameters, which make the issue even more complicated.

Due to a large variety of design choices, it is not trivial to identify a suitable cost function to solve a problem. Instead of manually designing a cost function, several works proposed to use machine learning techniques to learn a cost function from available training data. For example, kernel SVM [10], boosting [11], metric learning [12], and nonlinear regressors [13] have been used to learn a cost function for object tracking, alignment, and pose estimation. Once a cost function is learned, the optimal parameters are solved using search algorithms such as descent methods or particle swarm optimization. However, a downside of these approaches is that they require the form of the cost function to be imposed, e.g., [12] requires to cost to be quadratic, thereby restricting the class of problems that they can solve.

2.2 Learning search direction

Instead of using search directions from a cost function, many works proposed to directly learn to 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). The concept of SSUs resembles gradient boosting (GB) [14], [15], which uses weak learners to iteratively update the parameters. However, they differ in that GB performs update using a fixed feature vector, while SSUs also update the feature vector, which allows new information to be incorporated as the parameter is updated. Here, we provide a brief review of SSUs.

Cascaded pose regression [16] trains a sequence of random ferns for image-based object pose estimation. The paper also shows that the training error decreases exponentially under a weak learner assumption. [17] learns a sequence of boosted regressors that minimizes error in parameter space. Supervised descent method (SDM) [18], [19] learns 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 [20];

running Gauss-Newton algorithm after SSU [21]; using different maps in different regions of the parameter space [22]; and using recurrent neural network as the sequence of maps while also learning the feature [23]. Instead of sequentially learning the regressors, [24] 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 features for these problems are rather obvious: they use intensity-based features such as intensity difference, SIFT, HOG, etc. Extending SSUs to other problems require designing new features. Our recent work [25] proposes a new feature and extends SSU to the problem of point cloud registration. Still, it is not straightforward to design features for other applications. In this work, we propose DO as an extension of previous SSUs. We study its properties, propose a general framework for designing features, and apply DO to computer vision problems not addressed by previous SSUs, such as point cloud registration, camera pose estimation, and image denoising.

Recently, deep neural networks (DNNs) have received tremendous interest for their success in various tasks in computer vision and natural language processing [26]. While DNNs have been used for some applications similar to those in this paper, e.g., camera pose estimation [27] and image denoising [28], they are typically purely data-driven while DO can combine the mathematical model of each problem with the available training data. The model-driven side of DO allows it to be interpreted and analyzed more easily, and significantly reduces the amount of training data and computational power.

3 Discriminative Optimization (DO)

In this section, we provide the motivation and describe the Discriminative Optimization (DO) framework.

3.1 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

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

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. (1) 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 [29]. Let $J\colon \mathbb{R}^p \to \mathbb{R}$ be a differentiable cost function. The gradient descent algorithm for minimizing J is expressed as

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \mu_t \frac{\partial}{\partial \mathbf{x}} J(\mathbf{x})\bigg|_{\mathbf{x}=\mathbf{x}_t}, \tag{2}$$

where μ_t is a step size. One can see that the scaled gradient is used as $\Delta\mathbf{x}_t$ in (1), 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 selected and the neighborhoods around the solutions of the perturbed data are taken into account when the maps are learned, leading to more robustness.

3.2 DO framework

DO uses an update rule in the form of (1). The update vector $\Delta\mathbf{x}_t$ is computed by mapping the output of a function $\mathbf{h}\colon \mathbb{R}^p \to \mathbb{R}^f$ with a sequence of matrices[1] $\mathbf{D}_t \in \mathbb{R}^{p\times f}$. 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 position $\mathbf{x}$). Given an initial parameter $\mathbf{x}_0 \in \mathbb{R}^p$, DO iteratively updates $\mathbf{x}_t, t = 0, 1, \ldots,$ using:

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \mathbf{D}_{t+1}\mathbf{h}(\mathbf{x}_t), \tag{3}$$

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).

3.2.1 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 $\mathbf{h}^{(i)}\colon \mathbb{R}^p \to \mathbb{R}^f$ extracts features from the i^{th} problem instance. The goal of DO is to learn a sequence of update maps $\{\mathbf{D}_t\}_t$ that updates $\mathbf{x}_0^{(i)}$ to $\mathbf{x}_*^{(i)}$. To learn the maps, we solve the least squares problem:

$$\mathbf{D}_{t+1} = \arg\min_{\tilde{\mathbf{D}}} \frac{1}{N}\sum_{i=1}^{N} \|\mathbf{x}_*^{(i)} - \mathbf{x}_t^{(i)} + \tilde{\mathbf{D}}\mathbf{h}^{(i)}(\mathbf{x}_t^{(i)})\|_2^2, \tag{4}$$

where $\|\cdot\|_2$ is the ℓ_2 norm. After we learn a map $\mathbf{D}_{t+1}$, we update each $\mathbf{x}_t^{(i)}$ using (3), 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 (4) learns stationary points, we can see that for i with $\mathbf{x}_t^{(i)} \approx \mathbf{x}_*^{(i)}$, (4) 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:

$$\underset{\tilde{\mathbf{D}}}{\text{minimize}} \frac{1}{N}\sum_{i=1}^{N} \|\mathbf{x}_*^{(i)} - \mathbf{x}_t^{(i)} + \tilde{\mathbf{D}}\mathbf{h}^{(i)}(\mathbf{x}_t^{(i)})\|_2^2 + \lambda\|\tilde{\mathbf{D}}\|_F^2, \tag{5}$$

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

3.2.2 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) 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 is still not reached, and that $\mathbf{x}_T$ is far from the true solution. For example, in the point cloud registration task, the rotation between initial orientation and the solution might be so large that we cannot obtain the solution within T 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 is reached. This approach makes DO different

1. Here, we use linear maps due to their simplicity and computational efficiency, but it can be replaced by other nonlinear regressors.

Algorithm 1 Training a sequence of update maps (SUM)

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

Algorithm 2 Searching for a stationary point

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

from previous works in Sec. 2.2, where the updates are only performed up to the number of maps. Alg. 2 shows the pseudocode for updating the parameters.

3.3 Relation to Supervised Descent Method (SDM)

SDM [19] is a SSU algorithm which uses a sequence of linear maps to update the parameters. While this work uses similar learning and update rules, DO subsumes [19] in the following key issues: *(i)* SDM was inspired as a method for solving nonlinear least-squares problems, but this claim was not verified. In this work, we mathematically relate DO to generalized convexity (Sec. 4.2), allowing us to interpret DO as imitating gradient descent to solve some nonconvex problems with an unknown cost function. *(ii)* We provide a framework for deriving feature $\mathbf{h}$ for different tasks (Sec. 5), unlike SDM and other SSU work where $\mathbf{h}$ needs to be designed *ad hoc*. *(iii)* SDM views the use of multiple maps as heuristic while we provide a theoretical result explaining the necessity and benefits of using multiple maps (Sec. 4.1).

4 Theoretical Analysis of DO

In this section, we analyze the theoretical properties of DO. Specifically, we discuss the conditions for the convergence of the training error, and the relation between DO and generalized convexity.

4.1 Convergence of training error

Here, we show that under weak assumptions on $\mathbf{h}^{(i)}$, we can learn a SUM that updates $\mathbf{x}_0^{(i)}$ to $\mathbf{x}_*^{(i)}$, i.e., the training error converges to zero. First, we define the *monotonicity at a point* condition:

Definition 1. *(Monotonicity at a point) A function* $\mathbf{f}: \mathbb{R}^p \to \mathbb{R}^p$ *is*
(i) monotone at $\mathbf{x}_* \in \mathbb{R}^p$ *if for any* $\mathbf{x} \in \mathbb{R}^p$,

$$(\mathbf{x} - \mathbf{x}_*)^\top \mathbf{f}(\mathbf{x}) \geq 0, \tag{6}$$

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

$$(\mathbf{x} - \mathbf{x}_*)^\top \mathbf{f}(\mathbf{x}) \geq 0, \tag{7}$$

where the equality holds only at $\mathbf{x} = \mathbf{x}_*$,

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

$$(\mathbf{x} - \mathbf{x}_*)^\top \mathbf{f}(\mathbf{x}) \geq m\|\mathbf{x} - \mathbf{x}_*\|_2^2. \tag{8}$$

It can be seen that if $\mathbf{f}$ is strongly monotone at $\mathbf{x}_*$ then $\mathbf{f}$ is strictly monotone at $\mathbf{x}_*$, and if $\mathbf{f}$ is strictly monotone at $\mathbf{x}_*$ then $\mathbf{f}$ is monotone at $\mathbf{x}_*$. With the above definition, we obtain the following result:

Theorem 1. *(Convergence of SUM'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 strictly monotone at* $\mathbf{x}_*^{(i)}$ *for all* i, *and if there exists some* i *where* $\mathbf{x}_t^{(i)} \neq \mathbf{x}_*^{(i)}$, *then the update rule:*

$$\mathbf{x}_{t+1}^{(i)} = \mathbf{x}_t^{(i)} - \mathbf{D}_{t+1}\mathbf{h}^{(i)}(\mathbf{x}_t^{(i)}), \tag{9}$$

with $\mathbf{D}_{t+1} \subset \mathbb{R}^{p \times f}$ *obtained from* (4), *guarantees that the training error strictly decreases in each iteration:*

$$\sum_{i=1}^N \|\mathbf{x}_*^{(i)} - \mathbf{x}_{t+1}^{(i)}\|_2^2 < \sum_{i=1}^N \|\mathbf{x}_*^{(i)} - \mathbf{x}_t^{(i)}\|_2^2. \tag{10}$$

Moreover, if $\hat{\mathbf{D}}\mathbf{h}^{(i)}$ *is strongly monotone at* $\mathbf{x}_*^{(i)}$, *and if there exist* $L > 0, H \geq 0$ *such that*

$$\|\hat{\mathbf{D}}\mathbf{h}^{(i)}(\mathbf{x}^{(i)})\|_2^2 \leq H + L\|\mathbf{x}_*^{(i)} - \mathbf{x}^{(i)}\|_2^2 \tag{11}$$

for all i, *then the training error converges to zero. If* $H = 0$ *then the error converges to zero linearly.*

The proof of Thm. 1 is provided in the appendix. In words, Thm. 1 says that if each instance i is similar in the sense that each $\hat{\mathbf{D}}\mathbf{h}^{(i)}$ is strictly monotone at $\mathbf{x}_*^{(i)}$, then sequentially learning the optimal maps with (4) guarantees that the training error strictly reduces in each iteration. If $\hat{\mathbf{D}}\mathbf{h}^{(i)}$ is strongly monotone at $\mathbf{x}_*^{(i)}$ and upperbounded then the error converges to zero. Note that $\mathbf{h}^{(i)}$ is not required to be differentiable or continuous. Xiong and De la Torre [19] also present a convergence result for a similar update rule, but they show the strict decrease of error of a *single* function $\mathbf{h}$ under a *single ideal* map. It also requires an additional condition called 'Lipschitz at a point.' 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 shows the convergence of *multiple* functions under the same SUM learned from the data, where each learned map $\mathbf{D}_t$ can be different from the ideal map $\hat{\mathbf{D}}$. To ensure reduction of error, Thm. 1 also does not require the 'Lipschitz at a point' as the norms of the maps are adjusted based on the training data. Meanwhile, to ensure the convergence to zero, Thm. 1 requires an upperbound which can be thought of as a relaxed version of 'Lipschitz at a point' (note 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 it allows robust discontinuous features, such as HOG in [19], 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 Relation to generalized convexity

In this section, we explore the relation between DO 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 [30], [31]. Understanding this relation leads to a framework for designing $\mathbf{h}$ in Sec. 5. We begin this section by providing definitions and propositions relating generalized convexity and monotonicity, then we provide our result in the end.

A pseudoconvex function is defined as follows.

Definition 2. *(Pseudoconvexity [31]) A differentiable function* $f\colon \mathbb{R}^p \to \mathbb{R}$ *is*

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

$$(\mathbf{x}-\mathbf{x}')^\top \nabla f(\mathbf{x}') \geq 0 \implies f(\mathbf{x}) \geq f(\mathbf{x}'), \tag{12}$$

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

$$(\mathbf{x}-\mathbf{x}')^\top \nabla f(\mathbf{x}') \geq 0 \implies f(\mathbf{x}) > f(\mathbf{x}'), \tag{13}$$

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

$$(\mathbf{x}-\mathbf{x}')^\top \nabla f(\mathbf{x}') \geq 0 \implies f(\mathbf{x}) \geq f(\mathbf{x}')+m\|\mathbf{x}-\mathbf{x}'\|_2^2. \tag{14}$$

Fig. 2d shows examples of pseudoconvex functions. In essence, pseudoconvex functions are differentiable functions where the sublevel sets are convex and all stationary points are global minima [32]. Pseudoconvex functions generalize convex functions: all differentiable convex functions are pseudoconvex. Pseudoconvex functions are used as penalty functions for their stronger robustness than convex ones [6], [8], [33]. Next, we introduce pseudomonotonicity.

Definition 3. *(Pseudomonotonicity [31]) A function* $\mathbf{f}\colon \mathbb{R}^p \to \mathbb{R}^p$ *is*

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

$$(\mathbf{x}-\mathbf{x}')^\top \mathbf{f}(\mathbf{x}') \geq 0 \implies (\mathbf{x}-\mathbf{x}')^\top \mathbf{f}(\mathbf{x}) \geq 0, \tag{15}$$

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

$$(\mathbf{x}-\mathbf{x}')^\top \mathbf{f}(\mathbf{x}') \geq 0 \implies (\mathbf{x}-\mathbf{x}')^\top \mathbf{f}(\mathbf{x}) > 0, \tag{16}$$

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

$$(\mathbf{x}-\mathbf{x}')^\top \mathbf{f}(\mathbf{x}') \geq 0 \implies (\mathbf{x}-\mathbf{x}')^\top \mathbf{f}(\mathbf{x}) \geq m\|\mathbf{x}-\mathbf{x}'\|_2^2. \tag{17}$$

It can also be shown that monotone (*resp.*, strictly, strongly) functions are pseudomonotone (*resp.*, strictly, strongly) [31]. The following proposition provides a relation between the gradients of pseudoconvex functions and pseudomonotonicity.

Proposition 1. *(Convexity and monotonicity [31]) A differentiable function* $f\colon \mathbb{R}^p \to \mathbb{R}$ *is pseudoconvex (*resp.*, strictly, strongly) if and only if its gradient is pseudomonotone (*resp.*, strictly, strongly).*

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

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

The converse of the proposition is not true. For example, $\mathbf{f}(\mathbf{x}) = [x_1x_2^2+x_1, x_2x_1^2+x_2]^\top$ is strictly monotone at $\mathbf{0}_2$, but not strictly pseudomonotone (counterexample at $\mathbf{x} = (1,2)$ and $\mathbf{y} = (2,1)$). Prop. 2 shows that monotonicity-at-a-point is a generalization of pseudomonotonicity, implying that the conditions in Thm. 1 are weaker than the conditions for the gradient maps of pseudoconvex and convex functions.

5 Designing $\mathbf{h}$

The function $\mathbf{h}$ which provides information about each problem instance is crucial for solving a problem. In this section, we describe a framework to design $\mathbf{h}$ for solving a class of problem based on our analysis in Sec. 4. We are motivated by the observation that many problems in computer vision aim to find $\mathbf{x}$ such that $\mathbf{g}_j(\mathbf{x}) = \mathbf{0}_d, j = 1, \ldots, J$, where $\mathbf{g}_j\colon \mathbb{R}^p \to \mathbb{R}^d$ models the problem of interest (see Sec. 2.1). To solve such problem, one may formulate an optimization problem of the form

$$\underset{\mathbf{x}}{\text{minimize}}\, \Phi(\mathbf{x}) = \frac{1}{J}\sum_{j=1}^{J} \varphi(\mathbf{g}_j(\mathbf{x})), \tag{18}$$

where $\varphi\colon \mathbb{R}^d \to \mathbb{R}$ is a penalty function, e.g., sum of squares, ℓ_1 norm, etc. If $\Phi(x)$ is differentiable, then we can use gradient descent to find a minimum and returns it as the solution. 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 in real data. The following framework is based on the concept of using training data to learn the update directions that mimic gradient descent of an unknown φ, thereby bypassing its manual selection.

5.1 $\mathbf{h}$ from the gradient of an unknown penalty function

For simplicity, we assume $\Phi(\mathbf{x})$ is differentiable, but the following approach also applies when it is not, namely when it is subdifferentiable. Let us observe its derivative:

$$\frac{\partial \Phi(\mathbf{x})}{\partial \mathbf{x}} = \frac{1}{J}\frac{\partial}{\partial \mathbf{x}}\sum_{j=1}^{J}\varphi(\mathbf{g}_j) = \frac{1}{J}\sum_{j=1}^{J}\left[\frac{\partial \mathbf{g}_j}{\partial \mathbf{x}}\right]^\top \frac{\partial \varphi(\mathbf{g}_j)}{\partial \mathbf{g}_j}, \tag{19}$$

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 (19), while the Jacobian $\frac{\partial \mathbf{g}_j}{\partial \mathbf{x}}$ does not depend on it. Since different φ's are robust to different perturbations, this last term determines the robustness of the solution. Here, we will use DO to learn this term from a set of training data.

In order to do so, we need to express (19) as $\mathbf{Dh}$. First, we rewrite (19) as the update vector $\Delta\mathbf{x}$, where we replace the derivative of φ with a generic function $\phi\colon \mathbf{R}^d \to \mathbf{R}^d$:

$$\Delta\mathbf{x} = \frac{1}{J}\sum_{j=1}^{J}\left[\frac{\partial \mathbf{g}_j}{\partial \mathbf{x}}\right]^\top \phi(\mathbf{g}_j) \tag{20}$$

$$= \frac{1}{J}\sum_{j=1}^{J}\sum_{k=1}^{d}\left[\frac{\partial \mathbf{g}_j}{\partial \mathbf{x}}\right]_{k,:}^\top [\phi(\mathbf{g}_j)]_k, \tag{21}$$

where $[\mathbf{Y}]_{k,:}$ is row k of $\mathbf{Y}$, and $[\mathbf{y}]_k$ is element k of $\mathbf{y}$. We then rewrite (21) as the following convolution:

$$\Delta\mathbf{x} = \frac{1}{J}\sum_{j=1}^{J}\sum_{k=1}^{d}\left[\frac{\partial \mathbf{g}_j}{\partial \mathbf{x}}\right]_{k,:}^{\top}\int_{\mathbb{R}^d}[\phi(\mathbf{v})]_k\delta(\mathbf{v}-\mathbf{g}_j)d\mathbf{v}, \tag{22}$$

where $\delta(\mathbf{v})$ is the Dirac delta function. It can be seen that (22) is equivalent to (19), while being linear in ϕ. This allows us to learn ϕ using linear least squares. To do so, we will express (22) in the form of $\mathbf{Dh}$. For simplicity, we will look at the element l of $\Delta\mathbf{x}$:

$$[\Delta\mathbf{x}]_l = \frac{1}{J}\sum_{j=1}^{J}\sum_{k=1}^{d}\left[\frac{\partial \mathbf{g}_j}{\partial \mathbf{x}}\right]_{k,l}\int_{\mathbb{R}^d}[\phi(\mathbf{v})]_k\delta(\mathbf{v}-\mathbf{g}_j)d\mathbf{v}, \tag{23}$$

$$= \sum_{k=1}^{d}\int_{\mathbb{R}^d}[\phi(\mathbf{v})]_k\left(\frac{1}{J}\sum_{j=1}^{J}\left[\frac{\partial \mathbf{g}_j}{\partial \mathbf{x}}\right]_{k,l}\delta(\mathbf{v}-\mathbf{g}_j)\right)d\mathbf{v}, \tag{24}$$

$$= \sum_{k=1}^{d}\int_{\mathbb{R}^d}\mathbf{D}(\mathbf{v},k)\mathbf{h}(\mathbf{v},k,l;\mathbf{x})d\mathbf{v}. \tag{25}$$

Eq. 25 expresses $[\Delta\mathbf{x}]_l$ as the inner product between $\mathbf{D}$ and $\mathbf{h}$ over $\mathbf{v}$ and k, where

$$\mathbf{D}(\mathbf{v},k) = [\phi(\mathbf{v})]_k, \tag{26}$$

$$\mathbf{h}(\mathbf{v},k,l;\mathbf{x}) = \frac{1}{J}\sum_{j=1}^{J}\left[\frac{\partial \mathbf{g}_j}{\partial \mathbf{x}}\right]_{k,l}\delta(\mathbf{v}-\mathbf{g}_j). \tag{27}$$

The following result discusses the convergence of the training error when $\mathbf{h}$ in (27) is used.

Proposition 3. *(Convergence of the training error with an unknown penalty 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* (27) *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 linearly to zero.*

Under similar conditions, we can also show the same convergence results for $\frac{1}{J_i}\sum_{j=1}^{J_i}\varphi(\mathbf{g}_j^{(i)}(\mathbf{x}^{(i)}))$ that is nondifferentiable strictly and strongly convex functions. Roughly speaking, Prop. 3 says that if there exists a penalty function φ such that for each i the global minimum of (18) is at $\mathbf{x}_*^{(i)}$ with no other local minima, then using (27) allows us to learn $\{\mathbf{D}_t\}$ for DO. Note that we do not need to explicitly know what such penalty function is. Thus, we can say that using (27) is equivalent to learning a surrogate of the gradient of an unknown cost function. This illustrates the potential of DO as a tool for solving a broad class of problems where the penalty function φ is unknown.

5.2 Computing $\mathbf{h}$

Eq. (27) expresses $\mathbf{h}$ as a function. To compute $\mathbf{h}$ in practice, we need to express it as a vector. To do so, we will convert $\mathbf{h}$ in (27) into a discrete grid, then vectorize it. Specifically, we first discretize $\delta(\mathbf{v}-\mathbf{g}_j)$ into a d-dimensional grid with r bins in each dimension, where a bin evaluates to 1 if $\mathbf{g}_j$ is discretized to that bin, and 0 for all other bins. Let us denote this grid as $\bar{\delta}_j$, and let $\gamma:\mathbb{R}\to\{1,\dots,r\}$ be a function where $\gamma(y)$ returns the index that y discretizes to. We can express the vectorized $\bar{\delta}_j$ as the following Kronecker product of standard basis vectors:

$$\text{vec}(\bar{\delta}_j) = \mathbf{e}_{\gamma([\mathbf{g}_j]_1)}\otimes\cdots\otimes\mathbf{e}_{\gamma([\mathbf{g}_j]_d)} = \bigotimes_{\alpha=1}^{d}\mathbf{e}_{\gamma([\mathbf{g}_j]_\alpha)} \in \{0,1\}^{r^d}. \tag{28}$$

With this discretization, we can express $\mathbf{h}$ in (27) in a discrete form as

$$\mathbf{h}(k,l;\mathbf{x}) = \frac{1}{J}\sum_{j=1}^{J}\left[\frac{\partial \mathbf{g}_j}{\partial \mathbf{x}}\right]_{k,l}\bigotimes_{\alpha=1}^{d}\mathbf{e}_{\gamma([\mathbf{g}_j]_\alpha)}. \tag{29}$$

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

$$\mathbf{h}(\mathbf{x}) = \frac{1}{J}\sum_{j=1}^{J}\bigoplus_{l=1}^{p}\bigoplus_{k=1}^{d}\left[\frac{\partial \mathbf{g}_j}{\partial \mathbf{x}}\right]_{k,l}\bigotimes_{\alpha=1}^{d}\mathbf{e}_{\gamma([\mathbf{g}_j]_\alpha)}, \tag{30}$$

where $\bigoplus$ denotes vector concatenation. The dimension of $\mathbf{h}$ is pdr^d. We show how to apply (30) to applications in Sec. 6. Note that the above approach is one way of designing $\mathbf{h}$ to use with SUM. It is possible to use different form of $\mathbf{h}$ (e.g., see Sec. 6.2), or replace $\mathbf{D}$ with a nonlinear map.

6 Experiments

In this section, we first provide an intuition into DO with a synthetic problem, then we apply DO to three computer vision tasks: 3D point cloud registration, camera pose estimation, and image denoising. All experiments were performed in MATLAB on a single thread on an Intel i7-4790 3.60GHz computer with 16GB memory.

6.1 Optimization with unknown 1D cost functions

In this experiment, we demonstrate DO's potential in solving 1D problems without an explicit cost function. Specifically, given a set of number $X=\{x_1,x_2,\dots,x_J\}$, we are interested in finding the solution $\hat{x}$ of the problem

$$g_j(\hat{x}) = 0 = \hat{x}-x_j, j=1,\dots,J. \tag{31}$$

A common approach to solve this problem is to solve the optimization

$$P: \underset{\hat{x}:\hat{x}=x_j+\epsilon_j}{\text{minimize}}\ \frac{1}{J}\sum_{j=1}^{J}\varphi(\epsilon_j) \equiv \underset{\hat{x}}{\text{minimize}}\ \frac{1}{J}\sum_{j=1}^{J}\varphi(\hat{x}-x_j), \tag{32}$$

for some function φ. The form of φ depends on the assumption on the distribution of ϵ_i, e.g., the maximum likelihood estimation for i.i.d. Gaussian ϵ_j would use $\varphi(x)=x^2$. If the an explicit form of φ is known, then we can compute $\hat{x}_*$ in closed form (e.g., φ is squared value or absolute value) or with an iterative algorithm. However, using a φ that mismatches with the underlying distribution of ϵ_j could lead to an optimal, but incorrect, solution $\hat{x}_*$. Here, we will use DO to solve for $\hat{x}_*$ from a set of training data.

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

$$\varphi_1(x)=|x| \qquad \varphi_4(x)=|x|^{0.7}$$
$$\varphi_2(x)=0.35|x|^{4.32}+0.15|x|^{1.23} \qquad \varphi_5(x)=1-\exp(-2x^2)$$
$$\varphi_3(x)=(3+\text{sgn}(x))x^2/4 \qquad \varphi_6(x)=1-\exp(-8x^2)$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2018.2826536, IEEE Transactions on Pattern Analysis and Machine Intelligence

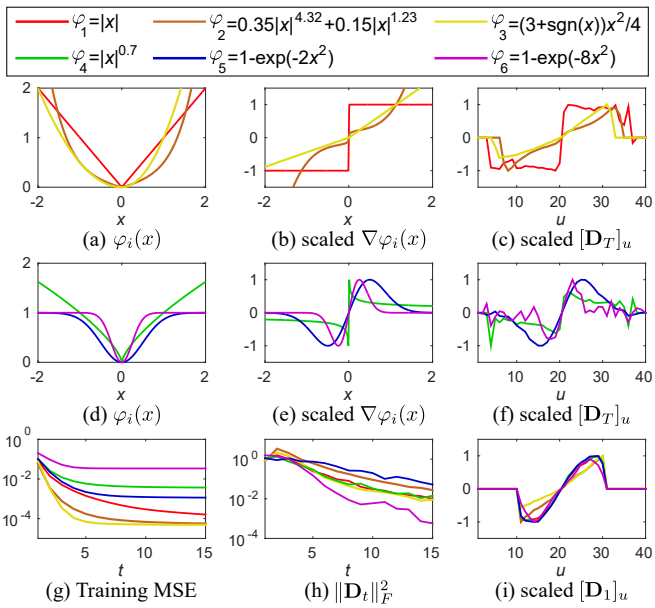


Figure 2. Learning to solve unknown cost functions. (a-c) show three convex functions, their gradients, and the learned $\mathbf{D}_T$ for each function. (d-f) show similar figures for pseudoconvex functions. (g) shows training error in each step t. (h) shows the squared norm of the maps $\mathbf{D}_t$. (i) shows the first map of each function.

The first three φ_i'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 three φ_i's are pseudoconvex, where φ_4 has exponents smaller than 1; while φ_5 and φ_6 are inverted Gaussian functions with different widths. 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[2] are shown in Fig. 2a,d and Fig. 2b,e, *resp*. We call the problem in (32) that uses $\varphi \equiv \varphi_\beta$ as P_β.

We generate the training data for P_β as $X_\beta = \{(X_\beta^{(i)}, \hat{x}_{0,\beta}^{(i)}, \hat{x}_{*,\beta}^{(i)})\}_{i=1}^{10000}$ where $X_\beta^{(i)} = \{x_{1,\beta}^{(i)}, \ldots, x_{J_i,\beta}^{(i)}\} \subset [-1, 1]$; $\hat{x}_{0,\beta}^{(i)} = 0$ is the initial estimate; and $\hat{x}_{*,\beta}^{(i)}$ is the global minimizer of P_β with the data $X_\beta^{(i)}$. To find the minimizers, we use grid search with the step size of 0.0001. We trained the SUMs using the $\mathbf{h}$ in Sec. 5, which in this case is simply

$$\mathbf{h}(\hat{x}) = \frac{1}{J}\sum_{j=1}^{J} \mathbf{e}_{\gamma(\hat{x}-x_j)}. \tag{33}$$

We use $[-2, 2]$ as the range of $\hat{x} - x_j$, and discretize it into $r = 40$ bins. Let us denote the maps that was learned from 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 as the last map that reduce the training RMSE more than 0.005. During test, we set $\epsilon = 10^{-3}$ and $maxIter = 100$.

Fig. 2c,f show the scaled maps $\mathbf{D}_T$ for each β. We can see that the maps resemble the gradients of their respective functions, suggesting that DO can learn the gradients from training data without explicit access to the cost functions. The reason that SUM_β learns the gradient is because the stationary points need to satisfy $\mathbf{D}_T\mathbf{h}(\hat{x}) \approx 0 = \sum_j \nabla\varphi(\hat{x} - x_j)$. It should be noted that the first maps for all β in Fig. 2i are different from their T^{th} maps. This is because the first maps try to move $\hat{x}_0^{(i)}$ as close to $\hat{x}_*^{(i)}$ as possible and thus disregard the placement of the stationary point. The training errors in Fig. 2g show that convex functions are easier to learn than nonconvex ones. This is because nonconvex functions may have multiple local minima, which means there may not exist an ideal map where all training data are monotone at their solutions, thus $\hat{x}_t^{(i)}$ may get stuck at a wrong stationary point. Fig. 2h shows that the map have decreasing norms, which represents reducing step sizes as the estimates approach the solutions.

2. Here, we abuse the word *gradient* to include (generalized) subdifferential for nonsmooth functions [34].

Table 1
MAE of solving unknown cost functions. Best results in underline bold, and second best in bold.

P_β	fminunc						SUM_β
	P_1	P_2	P_3	P_4	P_5	P_6	
P_1	**.0000**	.0675	.1535	.0419	.0707	.2044	**.0137**
P_2	.0675	**.0000**	.1445	.1080	.1078	.2628	**.0145**
P_3	.1535	.1445	**.0000**	.1743	.1657	.2900	**.0086**
P_4	.0493	.1009	.1682	**.0457**	.0929	.1977	**.0325**
P_5	.0707	.1078	.1657	.0823	**.0000**	.1736	**.0117**
P_6	.2098	.2515	.2791	.1905	.2022	**.1161**	**.0698**

We also perform an experiment on unseen sets of data, where we compare the global minimizer $\hat{x}_{*,\beta}$ of P_β of each test data against the solution from `fminunc` of all $P_\omega, \omega = 1, \ldots, 6$, and the solution of SUM_β. Table 1 shows the MAE over 1000 test sets. We can see that DO can approximate the solution better than using incorrect cost functions. An interesting point to note is that DO was able to solve nonconvex problems better than `fminunc`, suggesting DO can avoid some local minima and more often terminates closer to the global minimum.

We summarize this experiment in 4 points. *(i)* We show that DO can learn to mimic gradient 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 data of P_1 and hope to get the minimizer of P_2 as the solution. *(iii)* A practical implication of this demonstration is that if we optimize a wrong cost function then we may obtain a bad optimum as solution, and it can 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 `fminunc`.

6.2 3D point cloud registration

In this section, we perform experiments on the task of 3D point cloud registration. The problem can be stated as follows: Let $\mathbf{M} \in \mathbb{R}^{3\times N_M}$ be a matrix containing 3D coordinates of one shape ('model') and $\mathbf{S} \in \mathbb{R}^{3\times N_S}$ for the second shape ('scene'), find the rotation and translation that registers $\mathbf{S}$ to $\mathbf{M}$. Here, we briefly describe our parametrization and experiments. For more details, please see [25].

6.2.1 *DO parametrization and training*

We use Lie Algebra [35] to parametrize $\mathbf{x}$, which represents rotation and translation, because it provides a linear space

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2018.2826536, IEEE Transactions on Pattern Analysis and Machine Intelligence

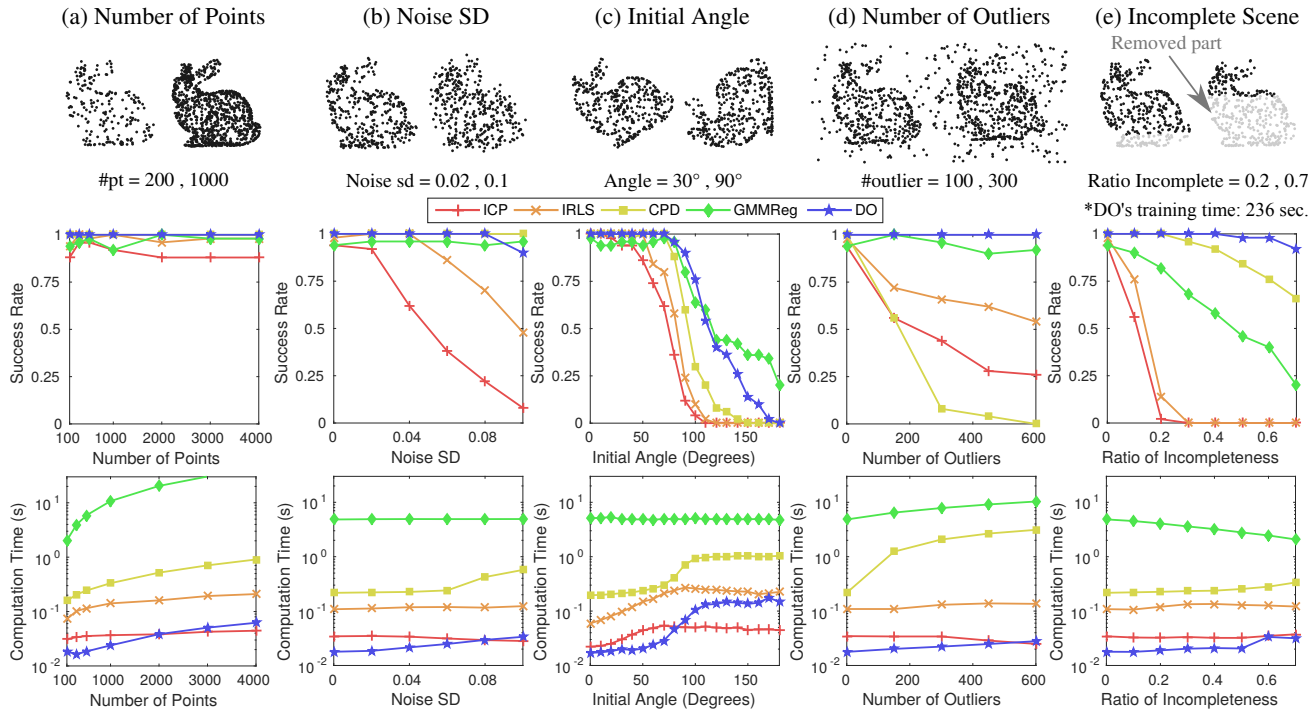


Figure 3. Results of 3D registration with synthetic data under different perturbations. (Top) Examples of scene points with different perturbations. (Middle) Success rate. (Bottom) Computation time.

with the same dimensions as the degrees of freedom of our parameters. For $\mathbf{h}$, we design it as a histogram that indicates the weights of scene points on the 'front' and the 'back' sides of each model point based on its normal vector. Let $\mathbf{n}_a \in \mathbb{R}^3$ be a normal vector of the model point $\mathbf{m}_a$ computed from its neighbors; $\mathcal{T}(\mathbf{y};\mathbf{x})$ be a function that applies rigid transformation with parameter $\mathbf{x}$ to vector $\mathbf{y}$; $S_a^+ = \{\mathbf{s}_b \colon \mathbf{n}_a^\top(\mathcal{T}(\mathbf{s}_b;\mathbf{x}) - \mathbf{m}_a) > 0\}$ be the set of scene points on the 'front' of $\mathbf{m}_a$; and S_a^- contains the remaining scene points. We define $\mathbf{h} \colon \mathbb{R}^6 \times \mathbb{R}^{3\times N_S} \to \mathbb{R}^{2N_M}$ as:

$$[\mathbf{h}(\mathbf{x};\mathbf{S})]_a = \frac{1}{z} \sum_{\mathbf{s}_b \in S_a^+} \exp\left(\frac{1}{\sigma^2}\|\mathcal{T}(\mathbf{s}_b;\mathbf{x}) - \mathbf{m}_a\|^2\right), \quad (34)$$

$$[\mathbf{h}(\mathbf{x};\mathbf{S})]_{a+N_M} = \frac{1}{z} \sum_{\mathbf{s}_b \in S_a^-} \exp\left(\frac{1}{\sigma^2}\|\mathcal{T}(\mathbf{s}_b;\mathbf{x}) - \mathbf{m}_a\|^2\right), \quad (35)$$

where z normalizes $\mathbf{h}$ to sum to 1, and σ controls the width of the exp function. $\mathbf{h}$ can be precomputed to speed up processing time (see [25]).

Given a model shape $\mathbf{M}$, we first normalized the data to lie in $[-1, 1]$, 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, and trained a total of $T = 30$ maps for SUM with $\lambda = 3 \times 10^{-4}$ in (5) and $\sigma^2 = 0.03$ in (34) and (35), and set the maximum number of iterations to 1000.

6.2.2 Baselines and evaluation metrics

We compared DO with two point-based approaches (ICP [1] and IRLS [5]) and two density-based approaches (CPD [36] and GMMReg [37]). The codes for all methods were downloaded from the authors' websites, except for ICP where we used MATLAB's implementation. For IRLS, the Huber penalty function was used.

We used the registration success rate and the computation time as performance metrics. 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.

6.2.3 Synthetic data

We performed synthetic experiments using the Stanford Bunny model [38] (see Fig. 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 100~4000 [default = 200~600]; *(ii)* the standard deviation of the noise ranges between 0~0.1

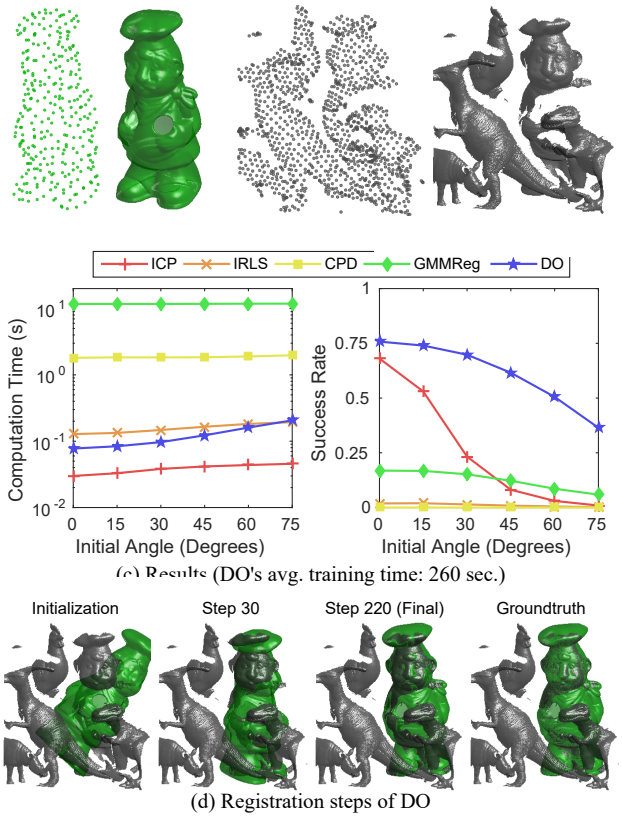


Figure 4. Results of 3D registration with range scan data. (a) example 3D model ('chef'). (b) Example of a 3D scene. We include surface rendering for visualization purpose. (c) Results of the experiment. (d) shows an example of registration steps of DO. The model was initialized 60 degrees from the ground 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.

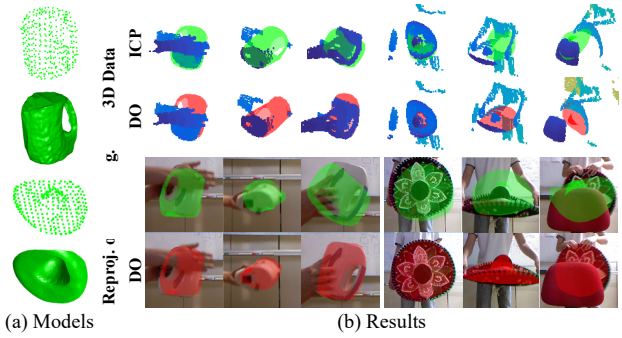


Figure 5. Result for object tracking in 3D point cloud. (a) shows the 3D models of the kettle and the hat. (b) shows tracking results of DO and ICP in (top) 3D point clouds with the scene points in blue, and (bottom) as reprojection on RGB image. Each column shows the same frame.

[default = 0]; *(iii)* the initial angle from 0 to 180 degrees [default = 0~60]; *(iv)* the number of outliers from 0~600 [default = 0]; and *(v)* the ratio of incomplete scene shape from 0~0.7 [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 in $[-0.3, 0.3]^3$. 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 Fig. 3. ICP required low computation time for all cases, but it had low success rates because it tended 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 it did not perform well with incomplete targets and 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 *learned* to cope with the perturbations from training data, allowing it to be significantly more robust than other approaches.

6.2.4 Range-scan data

In this section, we performed 3D registration experiment on the UWA dataset [39]. 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 four objects. We performed the test using all the four objects on all 50 scenes. From the original model, ~300 points were sampled by `pcdownsample` to use as $\mathbf{M}$ (Fig. 4a). We also downsampled each scene to ~1000 points (Fig. 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 all object models.

The results and examples for the registration with DO are shown in Fig. 4c and Fig. 4d, *resp*. IRLS, CPR, and GMMReg has very low success in almost every scene. This was because structured outliers caused 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 provided some correct results because it typically terminated at the nearest, and correct, local minimum. On the other hand, DO provided 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.

6.2.5 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 from 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 were much slower and failed to handle structured outliers in the scene (similar to Sec. 6.2.4). Fig. 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 tended 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 3D object tracker.

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.

6.3 Camera Pose Estimation

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 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

$$\tilde{\mathbf{p}}_j \equiv \mathbf{K} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \tilde{\mathbf{s}}_j, j = 1, \ldots, J, \tag{36}$$

where tilde denotes homogeneous coordinate, $\mathbf{K} \in \mathbb{R}^{3\times3}$ is a known intrinsic matrix, and $\equiv$ denotes equivalence up to a scaling factor. General approaches for camera pose estimation involve solving nonlinear problems [40], [41], [42], [43]. Most of existing approaches assume that there are no outlier matches in the correspondence set. When outliers are present, they rely on RANSAC [44] to select the inliers. One approach that does not rely on RANSAC is REPPnP [45]. It finds the camera pose by solving for the robust nullspace of a matrix that represents algebraic reprojection 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.

6.3.1 DO parametrization and training

To obtain the set of inliers, we solve for a matrix $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3]^\top \in \mathbb{R}^{3\times4}$ such that the geometric reprojection error [3] is zero (assuming $\mathbf{p}_j$ is calibrated):

$$\mathbf{g}_j(\mathbf{X}) = \mathbf{p}_j - \begin{bmatrix} \mathbf{x}_1^\top \tilde{\mathbf{s}}_j / \mathbf{x}_3^\top \tilde{\mathbf{s}}_j \\ \mathbf{x}_2^\top \tilde{\mathbf{s}}_j / \mathbf{x}_3^\top \tilde{\mathbf{s}}_j \end{bmatrix} = \mathbf{0}_2, j = 1, \ldots, J. \tag{37}$$

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

$$\underset{\mathbf{X}}{\text{minimize}} \ \frac{1}{J} \sum_{j=1}^{J} \varphi(\mathbf{g}_j(\mathbf{X})), \tag{38}$$

where φ is a penalty function. Following the derivation in Sec. 5, the $\mathbf{h}$ function can be derived as:

$$\mathbf{h}(\mathbf{X}) = \frac{1}{J} \sum_{j=1}^{J} \bigoplus_{l=1}^{12} \bigoplus_{k=1}^{2} \left[\frac{\partial \mathbf{g}_j(\mathbf{X})}{\partial \text{vec}(\mathbf{X})} \right]_{lk} \bigotimes_{\alpha=1}^{2} \mathbf{e}_{\gamma([\mathbf{g}_j(\mathbf{X})]_\alpha)}. \tag{39}$$

After computing (39), we normalize it to a unit vector and use it our feature. Note that, although the Jacobian matrix of $\mathbf{g}_j$ has 24 elements, it has only 12 degrees of freedom. Thus, we need to consider only its 12 values instead of all 24.

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 for the training samples. To reduce the effect of varying sizes of images and 3D points, we normalized the inputs to lie in $[-0.5, 0.5]$.[3] 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 $\mathbf{g}_j$, and discretize it to 10 bins. 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*. When DO returned $\mathbf{x}$ as result, we transformed it back to a 3×4 matrix $\mathbf{M}$, then projected the first three columns to $SO(3)$ to obtain the rotation matrix. For DO+P3P+RANSAC and DO+RPnP, we used $\mathbf{M}$ from DO to select matches with small reprojection error as inliers, then calculated the camera parameters using P3P+RANSAC [42] and RPnP [43] (without RANSAC).

6.3.2 Baselines and evaluation metrics

We compared our approach against 5 baselines. EPnP [40] and REPPnP [45] are deterministic approaches. The other three baselines, P3P+RANSAC [42], RPnP+RANSAC [43], and EPnP+RANSAC [40] rely on RANSAC to select inliers and use the respective PnP algorithms to find the camera parameters. The number of matches used in each algorithm is 3, 4, and 6, *resp*. We used the code from [45] as the implementation for the PnP algorithms. The RANSAC routine automatically determines the number of iterations

3. Camera matrix needs to be transformed accordingly, similar to [46].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2018.2826536, IEEE Transactions on Pattern Analysis and Machine Intelligence

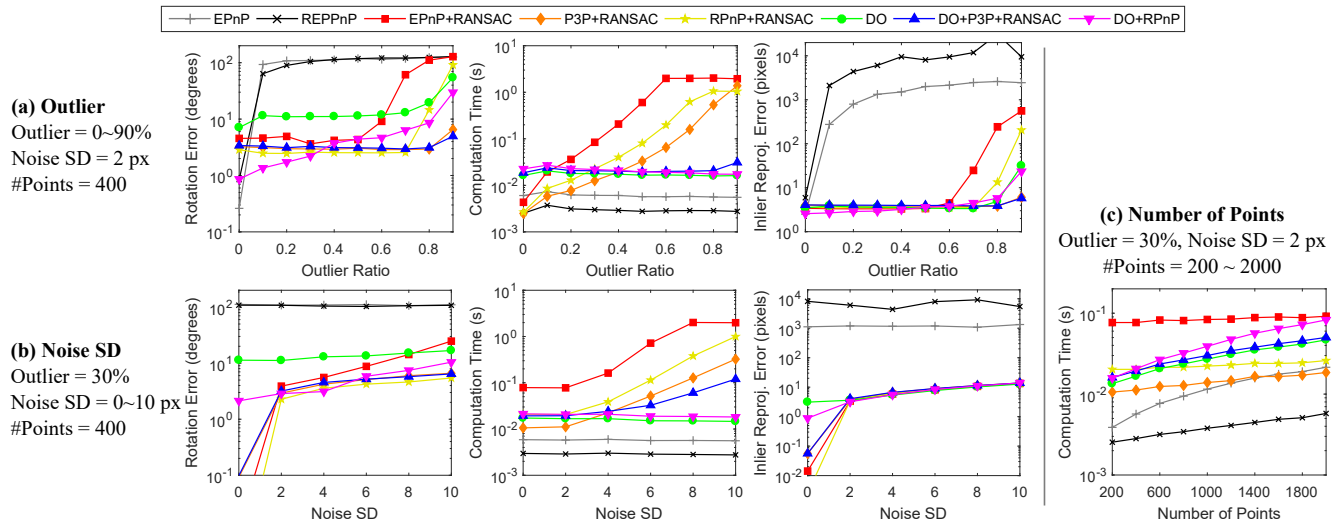


Figure 6. Results for PnP with synthetic data. Varying parameters are (a) outlier ratio, (b) noise SD, and (c) number of points.

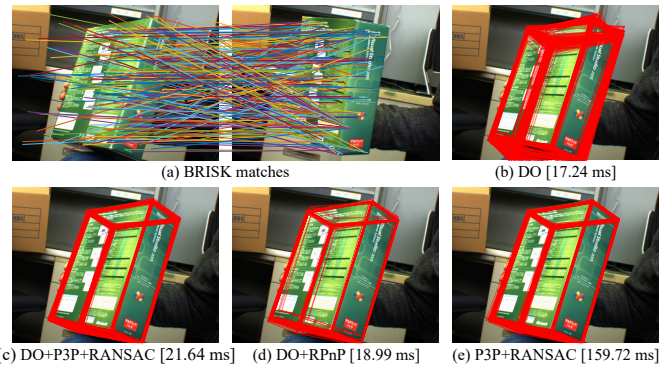


Figure 7. Results for camera pose estimation on real image. (a) Feature matches. Left and right images contain 2D points and projection of 3D points, *resp.* (b-d) Projected shape with average time over 100 trials.

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.

6.3.3 Experiments and results

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

Fig. 6 shows the results of the experiments. In Fig. 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 were small outliers, but the error increases as it was easier to mistakenly include outliers in the post-DO step. For the noise case (Fig. 6b), DO+RPnP has constant time for all noise levels and could comparatively obtain good rotations under all noise levels, while DO+P3P+RANSAC required exponentially increasing time as points with very high noise may be considered as outliers. Finally, in Fig. 6c, we can see that computation times of all approaches grow linearly with the number of points, but those of DO approaches grow with faster rate, which is a downside of our approach.

Next, we performed experiments on real images. We used an image provided with the code in [41]. Fig. 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. Fig. 7b-e show the results. It can be seen that DO can obtain a rough estimate of the camera pose, then DO+P3P+RANSAC and DO+RPnP can postprocess to obtain accurate pose. While P3P+RANSAC also obtained the correct pose, it required 8 times the computation time of DO-based approaches. (More results are provided in the appendix.)

Like all learning-based approaches, DO may fail when tested with data that are not well represented in training. As it is not straightforward to generate training data to cover all possible cases (e.g., all possible depths and perturbation), this could potentially lead to DO's failure to obtain the correct poses. On the other hand, PnP solvers which directly solve the geometric problem can reliably obtain the correct

poses when combined with the robustness of RANSAC (if enough computation time is given).

6.4 Image Denoising

In this final experiment, we demonstrate the potential of DO for image denoising. This experiment illustrates the potential of DO in multiple ways. First, we show that an 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 intensity mask, during both training and testing. Finally, we demonstrate the effect of training data on the results.

6.4.1 DO parametrization and training

We based our design of $\mathbf{h}$ on the TV denoising model [47], where we replace the penalty functions on both the data fidelity term and the regularization term with unknown functions φ_1 and φ_2:

$$\underset{\{x_i\}}{\text{minimize}} \sum_{i\in\Omega} \left(m_i \varphi_1(x_i - u_i) + \sum_{j\in\mathcal{N}(i)} \varphi_2(x_i - x_j) \right), \quad (40)$$

where 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 the learned SUM to work with images of different size, we will treat each pixel i independently: Each pixel will have its own estimate x_i.[4] Since we have two error terms, we follow Sec. 5 and concatenate the indicator of the two errors to form $\mathbf{h}$ as

$$\mathbf{h}(x_i) = \left[m_i \mathbf{e}^\top_{\gamma(x_i - u_i)}, \sum_{j\in\mathcal{N}(i)} \mathbf{e}^\top_{\gamma(x_i - x_j)} \right]^\top. \quad (41)$$

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

To train DO, we randomly sample 1000 patches of size 40×40 to 80×80 from the training image, 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 images have RV noise, while the rest have SP noise. This is to study the effect of training data on the learned SUMs. Following [48], for images with SP noise, we set the mask $m_i = 0$ for pixels with intensity 0 and 1 and $m_i = 1$ for others. For images with RV noise, we set $m_i = 1$ for all pixels as we cannot determine whether a pixel is an impulse noise or not. The intensity of each pixel in the noisy image is treated as initial estimate x_0, and x_* is its noise-free counterpart. We use $[-2,2]$ as the ranges for both $x_i - u_i$ and $x_i - x_j$, and discretize them to 100 bins. 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 $maxIter = 200$ as the stopping criteria.

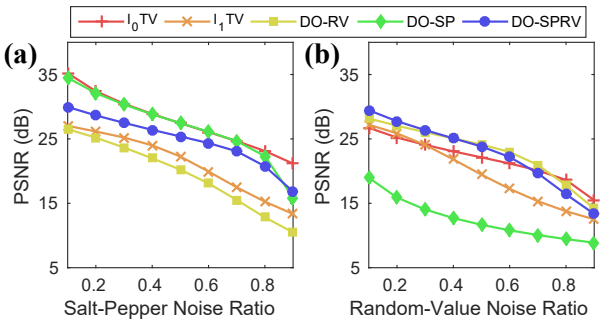


Figure 8. Results for image denoising for (a) salt-pepper impulse noise, and (b) random-value impulse noise.

6.4.2 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 $\ell_1 TV$ [49], 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 $\ell_0 TV$ [48], 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 [48]. 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).

6.4.3 Experiments and results

We downloaded 96 grayscale images of size 512×512 pixels from the Image Database[5] 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 and noise types, while the remaining 66 images were used for evaluation. For each image, we add impulse noise of 10% to 90% to measure the algorithm robustness.

Fig. 8 show the result PNSR over different noise ratios. It can be seen that DO trained with the correct noise type can match or outperform 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. Fig. 9 shows examples of denoising results of each algorithm ($\ell_1 TV$ omitted for clarity of other approaches). For SP noise, $\ell_0 TV$, DO-SP, and DO-SPRV can recover small details, while $\ell_1 TV$ oversmoothed the image and DO-SP returned an image with smudges. For RV noise, DO-RV returned the best result. DO-SPRV also returned an acceptable image but still contain intensity clumps, while DO-SP cannot recover the image at all. On the other hand, both baselines oversmoothed the image (notice the persons' heads) and 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 types and amount of noise that it can handle. The best approach for solving the problem is to select the

4. The idea is similar to parameter sharing in deep neural network.

5. http://decsai.ugr.es/cvg/dbimagenes/g512.php

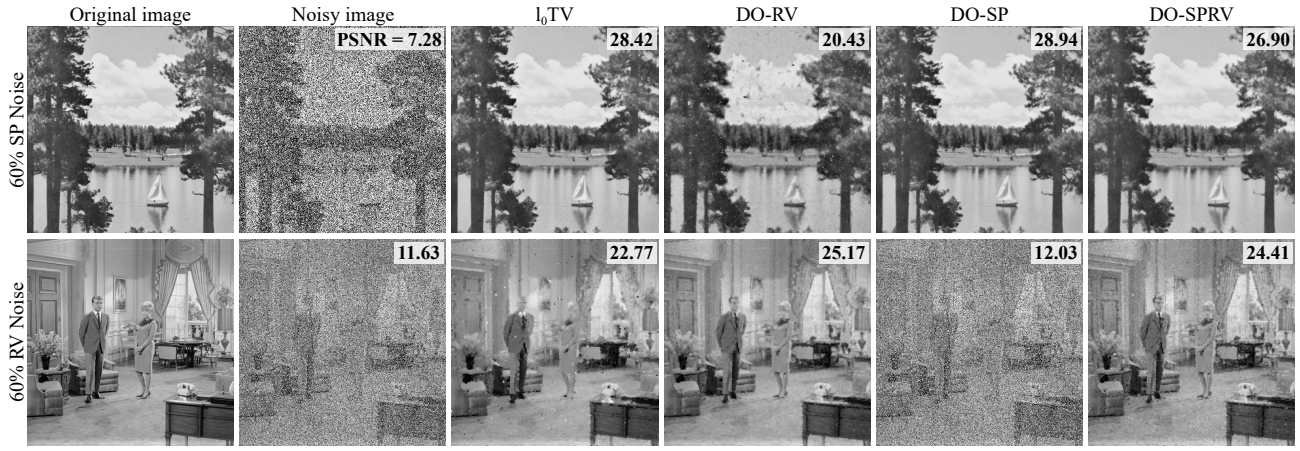


Figure 9. Examples of images denoising results for (top) salt-pepper impulse noise, and (bottom) random-value impulse noise. The PSNR for each image is shown on the top-right. We omitted the results of ℓ_1TV for visibility purpose. (Best viewed electronically).

correctly trained model. Still, training DO with both noise can return a good result, illustrating the potential of DO in solving a hard problem.

7 Conclusion and Discussion

We presented Discriminative Optimization (DO) as a new framework for solving computer vision problems. DO learns a sequence of update maps that update a set of parameters to a stationary point from training data. We provide a theoretical result on the convergence of the training error and study the relation between DO and generalized convexity. We also proposed a framework for designing features for DO based on gradient descent, and provide a theoretical result that DO can learn to imitate gradient descent under unknown cost functions. This result is supported by a synthetic experiment that shows that the maps learn to approximate the gradients. In terms of applications, we show that DO can perform favorably against state-of-the-art algorithms in the problems of 3D point cloud registration and tracking, camera pose estimation, and image denoising. This also shows that DO can deal with both ordered and unordered data.

Although algorithms similar to DO have been proposed previously, this paper opens the connection between DO and optimization. Future work may import ideas and intuition from optimization to DO, such as the incorporation of constraints and momentum methods. We also observe that DO's update rule can be compared with the layer in deep residual network [50], since they both iteratively perform update of the form $\mathbf{x} + T\mathbf{x}$ for some transformation T. This may provide some connections to deep learning algorithms. Future research may also address convergence in the test data or other approaches for designing feature function $\mathbf{h}$. With a strong theoretical foundation and practical potential, we believe DO opens a new exciting research area which would have strong impact to computer vision.

Acknowledgments

The authors would like to thank Beñat Irastorza for his invaluable suggestions. This research was supported in part by Fundação para a Ciência e a Tecnologia (project FCT [SFRH/BD/51903/2012] and a PhD grant from the Carnegie Mellon-Portugal program), the National Science Foundation under the grants RI-1617953, and the EU-Horizon 2020 project #731667 (MULTIDRONE). The content is solely the responsibility of the authors and does not necessarily represent the official views of the supporting agencies.

References

[1] P. J. Besl and H. D. McKay, "A method for registration of 3-D shapes," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 14, no. 2, pp. 239–256, 1992.
[2] M. Harker and P. O'Leary, "Least squares surface reconstruction from measured gradient fields," in *Proc. IEEE Conf. Comput. Vis. Pattern Recog.*, 2008, pp. 1–7.
[3] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision*. Cambridge Univ. press, 2003.
[4] R. Basri, L. Costa, D. Geiger, and D. Jacobs, "Determining the similarity of deformable shapes," *Vision Res.*, vol. 38, no. 15-16, pp. 2365–2385, 1998.
[5] P. Bergström and O. Edlund, "Robust registration of point sets using iteratively reweighted least squares," *Computational Optimization and Applicat.*, vol. 58, no. 3, pp. 543–561, 2014.
[6] J. T. Barron, "A more general robust loss function," *arXiv preprint arXiv:1701.03077*, 2017.
[7] H. Mobahi and J. W. Fisher, "Coarse-to-fine minimization of some common nonconvexities," in *Proc. Int. Conf. Energy Minimization Methods in Comput. Vis. Pattern Recog.*, 2015, pp. 71–84.
[8] M. Black and A. Rangarajan, "On the unification of line processes, outlier rejection, and robust statistics with applications in early vision," *Int. J. Comput. Vis.*, vol. 19, no. 1, pp. 57–91, 1996.
[9] M. Bertero, T. A. Poggio, and V. Torre, "Ill-posed problems in early vision," *Proc. IEEE*, vol. 76, no. 8, pp. 869–889, Aug 1988.
[10] S. Avidan, "Support vector tracking," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 8, pp. 1064–1072, 2004.
[11] X. Liu, "Discriminative face alignment," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 31, no. 11, pp. 1941–1954, 2009.
[12] M. H. Nguyen and F. De la Torre, "Metric learning for image alignment," *Int. J. Comput. Vis.*, vol. 88, no. 1, pp. 69–84, 2010.
[13] K. Paliouras and A. A. Argyros, "Towards the automatic definition of the objective function for model-based 3d hand tracking," in *Man–Mach. Interactions 4*. Springer, 2016, pp. 353–363.
[14] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Ann. Statist.*, vol. 29, no. 5, pp. 1189–1232, 2001.
[15] L. Mason, J. Baxter, P. Bartlett, and M. Frean, "Boosting algorithms as gradient descent," in *Proc. Adv. Neural Inform. Process. Syst.*, 1999, pp. 512–518.
[16] P. Dollár, P. Welinder, and P. Perona, "Cascaded pose regression," in *Proc. IEEE Conf. Comput. Vis. Pattern Recog.*, 2010, pp. 1078 – 1085.

[17] X. Cao, Y. Wei, F. Wen, and J. Sun, "Face alignment by explicit shape regression," in *Proc. IEEE Conf. Comput. Vis. Pattern Recog.*, 2012, pp. 2887–2894.

[18] X. Xiong and F. De la Torre, "Supervised descent method and its application to face alignment," in *Proc. IEEE Conf. Comput. Vis. Pattern Recog.*, 2013, pp. 532 – 539.

[19] ——, "Supervised descent method for solving nonlinear least squares problems in computer vision," *arXiv preprint arXiv:1405.0601*, 2014.

[20] G. Tzimiropoulos, "Project-out cascaded regression with an application to face alignment," in *Proc. IEEE Conf. Comput. Vis. Pattern Recog.*, 2015, pp. 3659–3667.

[21] E. Antonakos, P. Snape, G. Trigeorgis, and S. Zafeiriou, "Adaptive cascaded regression," in *Proc. IEEE Int. Conf. Image Process.*, 2016, pp. 1649–1653.

[22] X. Xiong and F. De la Torre, "Global supervised descent method," in *Proc. IEEE Conf. Comput. Vis. Pattern Recog.*, 2015, pp. 2664–2673.

[23] 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. IEEE Conf. Comput. Vis. Pattern Recog.*, 2016, pp. 4177–4187.

[24] K. Zimmermann, J. Matas, and T. Svoboda, "Tracking by an optimal sequence of linear predictors," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 31, no. 4, pp. 677–692, 2009.

[25] J. Vongkulbhisal, F. De la Torre, and J. P. Costeira, "Discriminative optimization: Theory and applications to point cloud registration," in *Proc. IEEE Conf. Comput. Vis. Pattern Recog.*, 2017, pp. 3975–3983.

[26] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[27] A. Kendall, M. Grimes, and R. Cipolla, "Posenet: A convolutional network for real-time 6-DOF camera relocalization," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2015, pp. 2938–2946.

[28] J. Xie, L. Xu, and E. Chen, "Image denoising and inpainting with deep neural networks," in *Proc. Adv. Neural Inform. Process. Syst.*, 2012, pp. 341–349.

[29] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge Univ. Press, 2004.

[30] R. T. Rockafellar, *Convex analysis*. Princeton Univ. Press, 1970.

[31] S. Karamardian and S. Schaible, "Seven kinds of monotone maps," *J. Optimization and Applicat.*, vol. 66, no. 1, pp. 37–46, 1990.

[32] M. Avriel, W. E. Diewert, S. Schaible, and I. Zang, *Generalized concavity*. SIAM, 2010.

[33] 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. IEEE Conf. Comput. Vis. Pattern Recog.*, 2013, pp. 1759–1766.

[34] N. Hadjisavvas and S. Schaible, "Generalized monotone multivalued maps," in *Encyclopedia of Optimization*. Springer, 2001, pp. 1193–1197.

[35] B. Hall, *Lie Groups, Lie Algebras, and Representations: An Elementary Introduction*. Springer, 2004.

[36] A. Myronenko and X. Song, "Point set registration: Coherent point drift," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 12, pp. 2262–2275, 2010.

[37] B. Jian and B. C. Vemuri, "Robust point set registration using gaussian mixture models," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 8, pp. 1633–1645, 2011.

[38] Stanford Computer Graphics Laboratory, "The Stanford 3D Scanning Repository," https://graphics.stanford.edu/data/3Dscanrep/, Aug 2014, accessed: 2016-08-31.

[39] 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. Vis.*, vol. 89, no. 2, pp. 348–361, 2010.

[40] V. Lepetit, F. Moreno-Noguer, and P. Fua, "EPnP: An accurate o(n) solution to the PnP problem," *Int. J. Comput. Vis.*, vol. 81, no. 2, pp. 155–166, 2008.

[41] Y. Zheng, Y. Kuang, S. Sugimoto, K. Åström, and M. Okutomi, "Revisiting the PnP problem: A fast, general and optimal solution," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2013, pp. 2344–2351.

[42] 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. IEEE Conf. Comput. Vis. Pattern Recog.*, 2011, pp. 2969 – 2976.

[43] S. Li, C. Xu, and M. Xie, "A robust O(n) solution to the perspective-n-point problem," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 7, pp. 1444–1450, 2012.

[44] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Commun. of the ACM*, vol. 24, no. 6, pp. 381–395, Jun. 1981.

[45] L. Ferraz, X. Binefa, and F. Moreno-Noguer, "Very fast solution to the PnP problem with algebraic outlier rejection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recog.*, 2014, pp. 501–508.

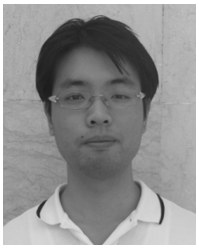
[46] R. I. Hartley, "In defense of the eight-point algorithm," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 19, no. 6, pp. 580–593, 1997.

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

[48] G. Yuan and B. Ghanem, "ℓ_0TV: a new method for image restoration in the presence of impulse noise," in *Proc. IEEE Conf. Comput. Vis. Pattern Recog.*, 2015, pp. 5369–5377.

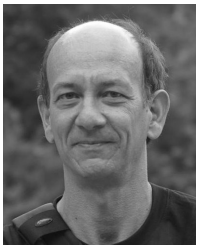
[49] A. Chambolle, V. Caselles, M. Novaga, D. Cremers, and T. Pock, "An introduction to Total Variation for Image Analysis," Nov. 2009, working paper or preprint. [Online]. Available: https://hal.archives-ouvertes.fr/hal-00437581

[50] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recog.*, 2016, pp. 770–778.



Jayakorn Vongkulbhisal received the BEng in information and communication engineering from Chulalongkorn University, Bangkok, Thailand, in 2011, and the MSc in Electrical Computer Engineering from Carnegie Mellon University in 2016. He is currently working toward the PhD degree at Carnegie Mellon and IST-Lisbon. His research interests include optimization and machine learning in computer vision.

Fernando De la Torre received the MSc and PhD degrees in electronic engineering from the La Salle School of Engineering at Ramon Llull University, Barcelona, Spain, in 1994, 1996, and 2002, *resp*. He is an associate research professor with the Robotics Institute at Carnegie Mellon University. His research interests include the fields of computer vision and machine learning. He is currently directing the Component Analysis Laboratory and the Human Sensing Laboratory at Carnegie Mellon University.



João Paulo Costeira received the PhD degree from IST in 1995. He is an associate professor of electrical and computer engineering at Instituto Superior Técnico (IST), Portugal. From 1992 to 1995, he was a member of Carnegie Mellon's Vision and Autonomous Systems Center. He is a researcher at Instituto de Sistemas e Robotica (ISR) since 1994 and scientific coordinator of the thematic area "Signal Processing for Communications Networks and Multimedia" of the ISR-Associated Lab.