Learning Direct Optimization for Scene Understanding

Lukasz Romaszko [1] Christopher K.I. Williams [1,2] John Winn [3]

[1]School of Informatics, University of Edinburgh, UK
[2]The Alan Turing Institute, UK [3]Microsoft Research
lukasz.romaszko@gmail.com, c.k.i.williams@ed.ac.uk, jwinn@microsoft.com

Abstract

We introduce a Learning Direct Optimization method for the refinement of a latent variable model that describes input image $\mathbf{x}$*. Our goal is to explain a single image* $\mathbf{x}$ *with a 3D computer graphics model having scene graph latent variables* $\mathbf{z}$ *(such as object appearance, camera position). Given a current estimate of* $\mathbf{z}$ *we can render a prediction of the image* $\mathbf{g}(\mathbf{z})$*, which can be compared to the image* $\mathbf{x}$*. The standard way to proceed is then to measure the error* $E(\mathbf{x}, \mathbf{g}(\mathbf{z}))$ *between the two, and use an optimizer to minimize the error. Our novel Learning Direct Optimization (LiDO) approach trains a Prediction Network to predict an update directly to correct* $\mathbf{z}$*, rather than minimizing the error with respect to* $\mathbf{z}$*. Experiments show that our LiDO method converges rapidly as it does not need to perform a search on the error landscape, produces better solutions, and is able to handle the mismatch between the data and the fitted scene model. We apply the LiDO to a realistic synthetic dataset, and show that the method transfers to work well with real images.*

1. Introduction

Figure 1. Real input image, reconstructed 3D scene and a different view of the scene. Due to the interpretable representation, one could easily edit the scene, e.g. refine object positions or their colours, or interact with objects, their properties and relations.

We introduce a *Learning Direct Optimization* (LiDO) method for the optimization of a latent variable model applied to the problem of explaining an image $\mathbf{x}$ in terms of a computer graphics model.

The latent variables (LVs) $\mathbf{z}$ are the scene graph, i.e. the shape, appearance, position and poses of all objects in the scene, plus global variables such as the camera and lighting. Our work is carried out in an analysis-by-synthesis framework, and we develop methods to the problem of scene understanding in 3D from a single image — this is to be contrasted with methods that simply predict 2D image-based bounding boxes or pixel labelling, see Figure 1. Note that optimization of the 3D scene projection in the form of an image is a challenging task even for a few LVs.

Given $\mathbf{z}$ we can render the scene graph to obtain a predicted image $\mathbf{g}(\mathbf{z})$, which can be compared to $\mathbf{x}$. The usual way to improve the match is to measure the error $E(\mathbf{x}, \mathbf{g}(\mathbf{z}))$, and to use an optimizer to update $\mathbf{z}$ via minimization of $E(\mathbf{x}, \mathbf{g}(\mathbf{z}))$. Our Learning Direct Optimization approach is based on the idea that a comparison between $\mathbf{x}$ and $\mathbf{g}(\mathbf{z})$ can provide good clues as to how $\mathbf{z}$ should be updated, and we can *train a network* to predict an update for $\mathbf{z}$ rather than requiring an error measure E to be defined in the image space, and then minimizing it.

In our system we use an *initialization network* to predict the starting configuration $\mathbf{z}_0$ based on $\mathbf{x}$, which also serves as the initialization for all the compared methods. The representation of the LVs consists of global and object LVs and is of variable dimension, i.e. $\mathbf{z} = (\mathbf{z}^{\text{Global}}, \mathbf{z}_1^{\text{Object}}, \ldots, \mathbf{z}_P^{\text{Object}})$, thus initialization network can initialize an arbitrary number of objects.

The Learning Direct Optimization method trains the *Prediction Network* on data where the current state $\mathbf{z}$ does not match the ground truth $\mathbf{z}_{GT}$. This was obtained in two ways: (i) from the initialization network, where based on $\mathbf{x}$, $\mathbf{z}_0$ and $\mathbf{g}(\mathbf{z}_0)$ one can predict $\mathbf{z}_{GT}$, and (ii) by perturbing $\mathbf{z}_{GT}$ to produce $\mathbf{z}'$, and learning to predict $\mathbf{z}_{GT}$ given $\mathbf{x}$, $\mathbf{z}'$ and $\mathbf{g}(\mathbf{z}')$. Note that the training data requirements for Prediction Network are similar to that needed to train the initialization network, and reuse the same data generator, so it has minimal marginal cost.

The main advantages of LiDO compared to standard op-

timization search are:

1. There is no need to choose a specific error metric E to measure of the mismatch between the input and the prediction.

2. Since the updates of LiDO are predicted by a neural network trained on the mistakes in $\mathbf{z}$, the $\mathbf{z}$-update directly targets the optimal $\mathbf{z}$ rather than simply moving downhill.

3. LiDO generally produces better solutions in shorter time and is more stable than standard optimizers, and is better able to handle mismatch between the data generator and the fitted scene model.

The structure of the rest of the paper is as follows: in sec. 2 we explain LiDO and how it contrasts to error-based optimization, and discuss related work. In sec. 3 we describe the vision-as-inverse-graphics framework and the initialization networks. Sec. 4 gives details of the experimental set-up, and results are described in sec. 5.

2. Learning Direct Optimization

The optimization of the latent variables $\mathbf{z}$ starts at configuration $\mathbf{z}_0$, which is obtained from the initialization networks described in sec. 3 below. From $\mathbf{z}_0$, a standard optimization, e.g. basic gradient optimization, would make a step based on the gradient of the error E as

$$\mathbf{z}_{t+1} = \mathbf{z}_t - \mu_t \nabla_{\mathbf{z}} E(\mathbf{x}, \mathbf{g}(\mathbf{z}_t)), \tag{1}$$

where μ_t is a stepsize, and $E(\mathbf{x}, \mathbf{g}(\mathbf{z}_t))$ measures the error between the image $\mathbf{x}$ and the current prediction $\mathbf{g}(\mathbf{z}_t)$. This error could e.g. measure the discrepancy in pixel space, or in some other feature space like the representation obtained in higher layers of a neural network (see e.g. [7, 9]).

Gradient-based optimization is not the only optimization strategy available. Due to the difficulties in obtaining gradients from a renderer, much work for minimizing $E(\mathbf{x}, \mathbf{g}(\mathbf{z}))$ has used gradient-free local search methods such a Simplex search, coordinate descent, genetic algorithms [17], and the COBYLA algorithm (as used in [7]).

In contrast, Learning Direct Optimization procedure takes as input $\mathbf{x}$, $\mathbf{z}_t$, and the current prediction $\mathbf{g}(\mathbf{z}_t)$, as shown in Figure 2. A Prediction Network is then trained to predict $\mathbf{z}_{t+1}^{\text{pred}}$, the true latent variables corresponding to $\mathbf{x}$, as described in sec. 4.2. The update for $\mathbf{z}$ then becomes

$$\mathbf{z}_{t+1} = \mathbf{z}_t + \mu_t(\mathbf{z}_{t+1}^{\text{pred}} - \mathbf{z}_t). \tag{2}$$

Setting the step size $\mu_t = 1$ would move from $\mathbf{z}_t$ to $\mathbf{z}_{t+1}^{\text{pred}}$, but we have found that in the case of multiple updates, using a $\mu_t < 1$ which decreases in time leads to better performance than keeping μ_t fixed.

The key insight is that comparison of $\mathbf{x}$ and the render $\mathbf{g}(\mathbf{z}_t)$ may yield much more information than simply the value of the error measure E. For example, if $\mathbf{x}$ contains a mug, and the prediction of $\mathbf{z}_t$ has the overall size and position of the mug correct, but the pose is incorrect so that the mug handle is predicted in the wrong place, a comparison of the two images (e.g. by subtraction) will show up a characteristic pattern of differences which can lead to a large move in $\mathbf{z}$ space. In contrast, if the handle positions are far apart, there may be no gradient information pushing $\mathbf{z}$ in the correct direction. Another problem is that the optimization may be misled when the observed image contains noisy features in a form of object textures, shadows, etc., while a prediction network can learn to ignore such distractions.

This optimization strategy is not limited to computer graphics and image analysis. Another possible application is e.g. explaining the acoustic signal of a piece of music in terms of notes played by different instruments.

Related Work: To our knowledge, the trained Learning Direct Optimization method which makes *repeated* use of the prediction network is novel. There are a number of works that use a trained *initialization* network, see e.g. [18] who used a neural network to predict the latent variables of deformable hand-written digit models, and more recently work such as AIR [3] network which initializes objects sequentially and NSD (Neural Scene De-rendering, [19]), but these work on synthetic scenes (cartoon objects with fixed colours, e.g. from a Minecraft game). The work of IM2CAD [7] and [13] use such initialization methods in a realistic scenario and apply their models to real scenes.

However, the standard practice for refinement from the obtained initialization is to then minimize some error function, where the reconstruction loss is based on a summary statistics of the image pixels (as described above), to carry out sampling/optimization of the posterior on $\mathbf{z}$. For example, IM2CAD aligns object shapes to the observed image; [6] optimize object poses to fit to the depth channel input; and [20] use a pixel-based error measure when sampling parameters of a 3D face model. In contrast LiDO directly predicts the LV updates.

3. Vision-as-Inverse Graphics and the Initialization Networks

Our work is carried out in a vision-as-inverse-graphics (VIG) or analysis-by-synthesis framework. This is a long-standing idea, see e.g. [4, 5, 21, 11], but it can be reinvigorated using the power of deep learning for the analysis stages.

Our set-up can be viewed as a kind of image autoencoder, with the initialization network being the encoder, the bottleneck consisting of the LVs $\mathbf{z}$, and the graphics ren-

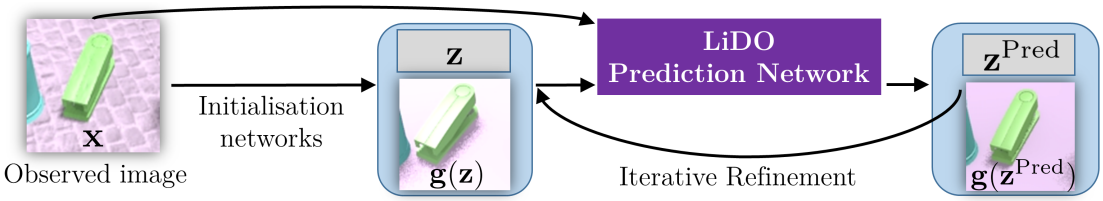


Figure 2. Learning Direct Optimization: given the observed image $\mathbf{x}$, the initialization of the latent variables (LVs) $\mathbf{z}$ is obtained, then the predicted image and LVs plus the observed image serve as the input to the LiDO network. The LVs are then updated according to the prediction and new render is produced. The LVs are then refined iteratively guided by the neural network.

derer being the decoder. The optimization of $\mathbf{z}$ to improve the fit to the image is non-standard in autoencoders.

Our work below considers high resolution scenes with a number of objects (from a set of object classes) on a ground plane (usually a table-top). For each object its associated latent variables are its position, scaling factor, azimuthal rotation, shape (1-of-K encoding) and colour (RGB). The ground plane has a random RGB colour. The camera is taken to be at height $y = h$ above the origin of the (x, z) plane, and to be looking at the ground plane with angle of elevation α, and fixed camera intrinsics. The illumination model is uniform lighting plus a directional source (specified by the strength, azimuth and elevation of the source). Objects and the ground plane have random noisy textures, and the background is a real indoor image (from [15]).

The 1-out-of-K object shape encoding is a simple yet effective baseline. As the predictions are made *per detected object and per object class*, one could extend this to use e.g. shape and texture morphable models like Blanz and Vetter [1] or later work— but note that the contribution of our system is demonstrating strong performance on optimizing *multiple objects* in a complex scene (plus camera, illumination), not just one object.

Figure 3. Examples from the Synthetic dataset, featuring a variablity in the objects present, their poses, appearance, as well as variable illumination and viewpoints.

The initialization networks are trained using synthetic data generated by the stochastic scene generator, rendered at 256×256 pixels. All the convolutional networks are trained on top of all the 13 convolutional layers of VGG-16 network [16], so as to afford transfer to work on real images.

The steps in obtaining an initial scene description $\mathbf{z}_0$ are:

1. Detect objects: class, contact point and size; extract patches $\mathbf{P}_0^{\mathbf{x}}$ from input image $\mathbf{x}$ at the contact points.
2. For each patch $\mathbf{P}_{0;p}^{\mathbf{x}}$ with $p = 1, \ldots P$ predict global LVs and object LVs: $\mathbf{z}_{0;p} = (\mathbf{z}_{0;p}^{\text{Global}}, \mathbf{z}_{0;p}^{\text{Object}})$ (global LVs are predicted by each object).
3. Aggregate votes for global LVs to obtain: $\mathbf{z}_0 = (\mathbf{z}_0^{\text{Global}}, \mathbf{z}_{0;1}^{\text{Object}}, \ldots, \mathbf{z}_{0;P}^{\text{Object}})$.

The above steps make use of the components described below, for each patches are 128×128 pixels (supplementary material provides the details).

Detector: The detector is trained to predict whether a particular object class is present at a given location, together with object size. Trained object detectors are run over the input image to produce a set of detections, which are then sparsified using non-maximum suppression (NMS).

Object and global initialization networks: We extract an image patch centered at each object detection, and use this to predict the ground truth latent variables. The networks are applied individually to each detected object patch. All the objects predict their own LVs as well as all global LVs. All the global LVs are trained/predicted per object, then combined; for robustness, the aggregation is done using a median function.

Back-projection and scene graph: The outputs of the above stages are assembled into a scene graph. During the rendering process the detected objects are back-projected into a 3D scene given the predicted camera to obtain the 3D positions. The object scaling factors are obtained from the predicted object size and the actual distance from the camera to the object after back-projection.

4. Experiments

Stochastic Scene Generator For each image we sample the global LVs and those of up to 7 objects which lie on the ground plane. We consider three object classes, namely mugs, bananas and staplers, for each object we sample its class, shape (one-of-K = 6/15/8 shapes respectively, obtained from ShapeNet[1] and then aligned), colour (albedo), rotation, and scaling factor. See Figure 3 for examples. Since our ultimate goal is understanding of real images, the synthetic images are generated with a rich realistic Blender renderer, where we have added shadows, realistic backgrounds and textures on the objects. The textures

[1] https://www.shapenet.org/

serve as a noise to allow transference of the LiDO to work with richer real images. Since we do not model the textures, the mean effect of texture is absorbed into the ground truth (GT) colour. Supplementary material provides more details of the scene generator.

Training Datasets We train the initialization networks on a dataset of 10k images with 55k+ objects. To train the Prediction Network we use data from two sources. The first (another 10k images and 55k+ objects) is obtained from the $\mathbf{z}_0$ outputs of the initialization network. We paired the detected and GT objects based on the distance of the object contact points to the closest one of the same class within the radius of 10% of the image width (15% for real images since manual annotations are more noisy than the perfect synthetic ones). The second source was to generate a dataset with small noise added to 10k GT images (55k+ objects) to allow LiDO to deal with small errors in further iterations. The noise was uniform for the continuous LVs: $\pm$ the median error made by the initialization network per LV. We also replaced each GT CAD shape by a random one to train LiDO to work well in the case of shape-mismatch. Thus the LiDO training dataset consisted of 110k object examples.

Test Datasets For all the neural networks we used appropriate train-validation-test splits of the synthetic dataset. Furthermore, we used a separate validation set for optimization tasks to choose the hyper-parameters of the LiDO and baseline methods, plus the main final optimization test set to evaluate them (each of 200 images, with over 1k objects).

Figure 4. Two examples from the Real dataset (images and object instance segmentation masks).

As our aim is to apply the LiDO method trained on realistic synthetic images for understanding of real images, we apply the same methods to a dataset consisting of real images with over 750 objects total. The annotated images were manually taken to feature a number of objects of the considered classes at a variety of lighting, viewpoint and object configuration conditions, see Fig. 4. For each object we annotated its class, instance mask, and the contact point using LabelMe software [14]. The renders of predictions consider objects on the infinite ground plane, but since the ground plane is finite for the observed images, for both Synthetic and Real datasets the GT mask is defined as the ground plane up to a horizontal line located at the contact point of the farthest GT object. For real images we annotated the GT ground plane mask, since sometimes the mask might not be the full plane. We will make the datasets publicly available upon publication.

4.1. Error-Based Optimization

We compare LiDO to two optimization methods: gradient-based optimization (GBO) with the best performing optimizer, and the most effective gradient-free method which was Simplex search (denoted Simp). Since these are standard search methods which require a large number of function evaluations, we use a very fast OpenGL renderer. LiDO was also configured to use OpenGL as the internal renderer during optimization.

For the error-based optimization, we compute the match between the actual and rendered image pixels (RGB intensities being between 0 and 1) using a robustified Gaussian likelihood model (with standard deviation 0.1 and inlier probability 0.8), as in [12, eq. 3].

For GBO we use a renderer[2] based on OpenDR: Differentiable Renderer [10], extended to simplify rendering multiple objects. The approximate derivatives of the likelihood computed by OpenDR are fed to an optimizer. To facilitate refinement we use anti-aliasing with 8 samples per pixel to make the gradients more accurate and the likelihood function smoother.

We performed experiments with several optimizers and most of them converged poorly (e.g. L-BFGS-B). We found Truncated Newton Conjugate-Gradient (TNC) to considerably outperform other gradient-based methods, with the Nonlinear Conjugate Gradient optimizer[3] being the only other one that usually converged properly (yet worse than TNC, so we use TNC for GBO).

For gradient-free methods, we found the Simplex (Nelder-Mead) optimizer worked well and significantly better than COBYLA, Simplex also often performed better and faster than GBO.

Setting proper bounds is crucial for proper hill climbing as the stepsize is scaled by the distance between LV boundaries, hence for all the LVs we set the bounds to the respective ranges as used in the scene generator.

Following the work of [12], we fit subsets of the search variables sequentially. The LVs are fit in the following order: ground plane color, object colours (each object separately), object poses (each object separately), illumination and the camera. We experimented with fitting all the object LVs together, and all the LVs together, but it worked a lot less well and overall slower, because the number of variables is larger and likely the optimization landscape is thus more complex.

[2] https://github.com/polmorenoc/inversegraphics

[3] http://learning.eng.cam.ac.uk/carl/code/minimize

4.2. Learning Direct Optimization

We run the initialization network on the synthetic dataset and take object detections and the associated errors as the new dataset for training the LiDO. For a training image $\mathbf{x}$ with ground truth $\mathbf{z}_{GT}$ we obtain a set of image patches $\mathbf{P}_0^{\mathbf{x}}$ extracted at the object detections and the corresponding rendered patches $\mathbf{P}_0^R$ from the render $\mathbf{g}(\mathbf{z}_0)$. From each pair of patches $\mathbf{P}_{0;p}^{\mathbf{x}}$ and $\mathbf{P}_{0;p}^R$, $p = 1, \ldots, P$ and the corresponding $\mathbf{z}_0$ the *Prediction Network* CNN is trained to predict the object-specific GT variables $\mathbf{z}_{GT;p}^{\text{Object}}$ and the global GT variables $\mathbf{z}_{GT}^{\text{Global}}$. Example errors are that an object could be larger, the camera located higher, etc.

The Prediction Network takes as input both the observed and the rendered image patches (128×128 pixels, down-sampled to 64 by 64 resolution), plus their difference. The RGB channels are stacked together thus the input is $64 \times 64 \times 9$. This input is followed by a number of convolutional layers shared across all the LVs. Shared layers are followed by LV-set specific convolutional layers, and finally a few fully-connected layers, each concatenated with the current estimate of $\mathbf{z}_p$ (object LVs, plus global LVs: both as predicted by object and after aggregation as used in the render). The whole network for all the LVs is trained together. We trained the Prediction Network for 20 epochs, this took 15 minutes/epoch on a single GPU. Note that we reuse the same scene generator, and that training time is no more than the training of the initialization networks. Further details on the architecture and training are given in the supplementary material.

Afterwards, we run the iterative refinement for T iterations as summarized below:
`for` $t \in 0..(T-1)$**:**

1. Take patches $\mathbf{P}_t^{\mathbf{x}}, \mathbf{P}_t^R$ from the input image $\mathbf{x}$ and the render $\mathbf{g}(\mathbf{z}_t)$ from the current $\mathbf{z}_t$ detections.
2. Predict $\mathbf{z}_{t+1;p}^{\text{pred}}$ given each $\mathbf{P}_{t;p}^{\mathbf{x}}, \mathbf{P}_{t;p}^R, \mathbf{z}_{t;p}$ using the Prediction Network
   (clip if outside of the range, e.g. colour not in [0,1]).
3. Update $\mathbf{z}_{t+1;p} = \mathbf{z}_{t;p} + \mu_t(\mathbf{z}_{t+1;p}^{\text{pred}} - \mathbf{z}_{t;p})$.
4. Aggregate global LVs to produce $\mathbf{z}_{t+1}$ (in the same manner as in sec. 3).

We set $\mu_t = 1/(t+a)$. The hyper-parameters including $a = 2$ were selected using the validation set split, as described in sec. 4. In the experiments below we run the LiDO iteration for a fixed number of $T = 30$ steps to show the convergence curve, but it would be easy to use a termination condition $|\mathbf{z}_{t+1} - \mathbf{z}_t| < \epsilon$.

When producing the OpenGL renders for the LiDO prediction, one needs to take care because LiDO has been trained to predict $\mathbf{z}$'s that specify a scene for the Blender renderer. While the geometry LVs (objects present, their shape/poses, camera etc.) are common for both the renderers, the optimal colours in OpenGL differ in brightness to Blender, since the OpenGL renderer cannot produce shadows, and has to explain shadowing (e.g. that mugs are dark inside when the light comes from the side) with a lower colour brightness. To render an OpenGL image for the LiDO prediction with Blender colour LVs, we adjust the brightness colour of each object and the ground plane. We do so by scaling the RGB colour by the ratio of the means of brightness calculated at the pixels of the predicted object mask, for LiDO's OpenGL render and the observed image. We can do this as it uses only the *predicted* masks, e.g. this would be equivalent to the final iteration of minimizing the MSE w.r.t. the colour LVs.

4.3. Set-up

Initialization for the Test Datasets: For the synthetic dataset, objects were accurately detected with 94.6% precision, 94.0% recall (94% of objects are detected, 94.6% of all detections are correct), and for these 57% of the object shapes were predicted correctly. For real images, the results were: 97.8% precision, 94.0% recall, thus the initialization networks transferred well to work for real images. All the methods that we compare start from the same initialization with the same set of the detected objects, unpaired objects are treated as false positives/negatives. The set of the instantiated objects and their shapes are kept fixed, as these are discrete variables which are not amendable to optimize with the above methods.

Optimization: We ran the methods long enough to allow convergence: 50 iterations for GBO, 100 iterations for Simplex, and 30 for LiDO, see Fig. 5. All the times are for a 4-core CPU for all the methods, to make the comparison fair LiDO is also executed on a CPU, including the CNNs[4].

We also conducted additional experiment where we trained LiDO to use in each iteration a realistic renderer (Blender) in place of the OpenGL one. The results were very similar and took only 50% more time, demonstrating that our Prediction Network method can make use of a slow yet realistic renderer.

4.4. Evaluation

Evaluation of the LVs: For the synthetic dataset, we evaluate the improvement in the LVs for all the methods. We evaluate seven sets of LVs that we optimize, the first four are object LVs: object position in the image frame, object colour, object size and object azimuthal rotation.

[4] Although the CNNs run on GPUs were a few times faster, this did not affect the overall speed significantly since rendering and other modules take most of the time.

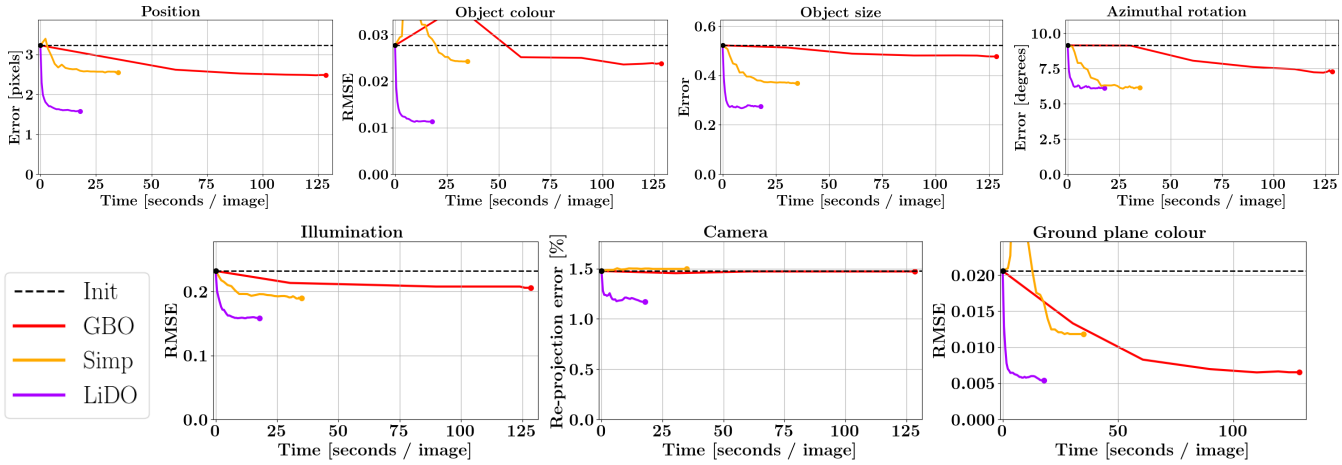


Figure 5. Median errors vs time in seconds (top: object LVs, bottom: global LVs). All three methods (GBO, Simp, LiDO) start from the initialization error (INIT) located at the black dashed line. Note the rapid convergence of the LiDO.

The remaining three are global LVs: illumination, camera, ground plane colour.

Object LVs: Object position error is a distance between object contact points in the image. Object size is the size of the projected object in the image frame, the error is the size difference. For azimuthal rotation we measure the absolute angular difference between the prediction and ground truth, but with wrap-around, so the maximum error is 180°. The error metric of the object colour (albedo) is the RMSE of normalized RGB components[5].

Global LVs: Ground plane colour is evaluated as for object colour above. The lighting is projected onto a sphere and evaluated at 313 points uniformly-distributed on the sphere, then normalized; the error is RMSE. The multiplicative interaction between illumination and albedo introduces a problem when evaluating them separately; by using normalized metrics we overcome this issue, while the joint result of both factors is directly available via pixel intensities (and can be compared via MSE). To assess the camera error, we place a (virtual) checkerboard in the scene, and compute the RMSE of the errors between the GT and predicted positions of the grid points in the image, as used by [13].

Evaluation in the image space (2D projection, pixels): We compare the observed and predicted images using the Intersection-over-Union (IoU) of the predicted and GT masks (of objects and ground plane), and MSE of pixel intensities calculated at the GT mask (of objects and ground plane). We can evaluate these measures for both synthetic and real datasets. Note IoU of the ground plane assesses differences in the present, missing and superfluous objects. The background (the part of the image not belonging to the ground plane or the object masks) is excluded from the explained pixels. Note that each MSE is calculated at the same pixels for all the methods, as these are calculated only at the GT masks.

[5] Normalized RGB is computed as $R/(R+G+B)$ etc.

LV name	GBO	Simp	LiDO
Object position	22.9	21.0	**51.0***
Object colour	14.0	12.4	**59.1***
Object size	8.63	29.4	**47.4***
Azimuthal rotation	20.3	32.6	**33.1**
Illumination	11.3	18.1	**31.6***
Camera	0.3	-1.3	**20.7***
Ground plane colour	68.2	42.4	**73.6**

Table 1. Results for median improvement over the initialization. Each value indicates how much (in %) the errors were lower after the refinement compared to the initialization, * denotes a statistically significantly better method.

5. Results

Evaluation of LVs on the synthetic dataset: Table 1 shows the median improvement of the method (GBO, Simp, LiDO) over the initialization. For all the seven error measures the LiDO method outperforms GBO and Simp. For four out of seven LV sets the LiDO improvements are at least two times better than the competitors (on illumination, camera, object: color, position). We calculate all the different metrics as in Section 3.4, e.g. deviation in pixels for object position or angle in degrees for rotation, and the median values of these errors are given in Fig. 5 as per the y-axis labels. However, since all the LVs are in different units, we compare the percentage improvement over the initialization.

To assess the statistical significance we conduct a paired test on the errors derived from each image (for global LVs) or object (for object LVs), using the Wilcoxon signed-rank

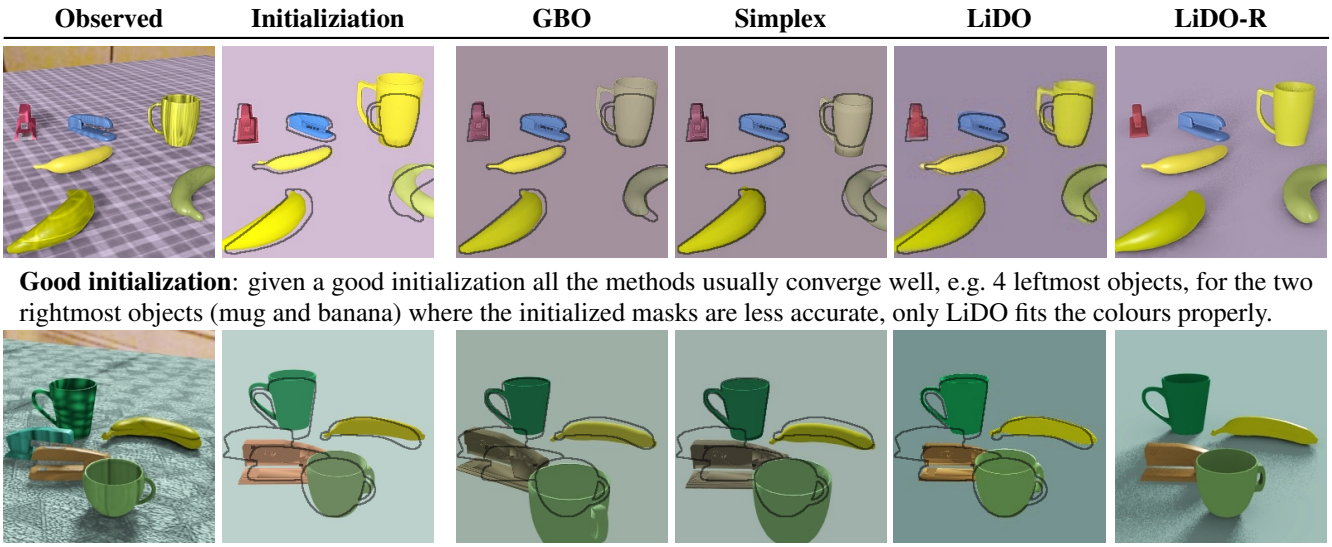


Figure 6. Example runs for Synthetic dataset, showing from left: the observed input image, the initialization (OpenGL), and images after refinement for GBO, Simplex and LiDO using OpenGL renderers, and LiDO using Blender renderer (LiDO-R). We overlay black contours of the ground truth object masks on top of each OpenGL image to ease the comparison of object poses.

test, at the significance level 0.05. For these LiDO outperforms GBO and Simp for 6 out of 7 LVs when comparing LiDO with each of them separately. This is because GBO does well with ground-plane colour since the objective it minimizes are the differences in pixel intensities between the input image $\mathbf{x}$ and the render $\mathbf{g}(\mathbf{x})$, and Simp works similarly to LiDO for the azimuthal rotation, but much worse for all other LVs.

Figure 5 shows the evolution of the median errors over time for the seven error measures; it is notable that the LiDO method obtains a lower error in much shorter time; For error-based methods since we do iterations per LV-set sequentially, we report the time for each iteration as the average time of reaching it.

Figure 8 shows example runs, showing both successes and failure cases, see textual descriptions under each image set. More examples of the fitting are given in the supplementary material, and the video of the fitting at: https://youtu.be/Axc0G8IggVU.

Image-space evaluation for the synthetic and real Datasets: Results for the Synthetic dataset for image-space measures are given in Table 2. To allow an equal comparison of three optimizers, all three methods use the OpenGL renderer. For all the measures LiDO outperforms the other methods, and particularly LiDO works much better for IoU measures. Note that due to unrealisable textures and shadows, the minimal (OpenGL GT) MSE errors are above 0, these are 11.7 for objects, and 8.4 for the ground plane. For the cases when the object shapes were predicted correctly, all the methods obtain object IoUs approximately 10 percentage points higher than for the cases where there is shape mismatch.

Measure	INIT	GBO	Simp	LiDO	
IoU [ob]	66.2	73.1	74.9	**78.9***	↑
IoU [gr]	86.4	88.7	88.8	**91.3***	
MSE [ob]	54.1	29.2	26.4	**21.8***	↓
MSE [gr]	34.1	12.8	13.6	**11.9**	

Table 2. Results of the pixel evaluation for synthetic dataset. Mean IoU (in %) and MSE ($\times 10^3$) for the objects [ob] and ground plane [gr]. The arrows indicate whether higher or lower values are better. * denotes a statistically significantly better method, using the Wilcoxon signed-rank test, at the significance level 0.05.

Measure	INIT	GBO	Simp	LiDO	
IoU [ob]	60.9	63.2	61.5	**71.4***	↑
IoU [gr]	87.6	87.3	86.1	**91.0***	
MSE [ob]	79.1	42.6	46.2	**35.9***	↓
MSE [gr]	69.1	27.5	30.5	**19.2***	

Table 3. Results of the pixel evaluation for real dataset. Mean IoU (in %) and MSE ($\times 10^3$) for the objects [ob] and ground plane [gr]. * denotes a statistically significantly better method, as per Table 2.

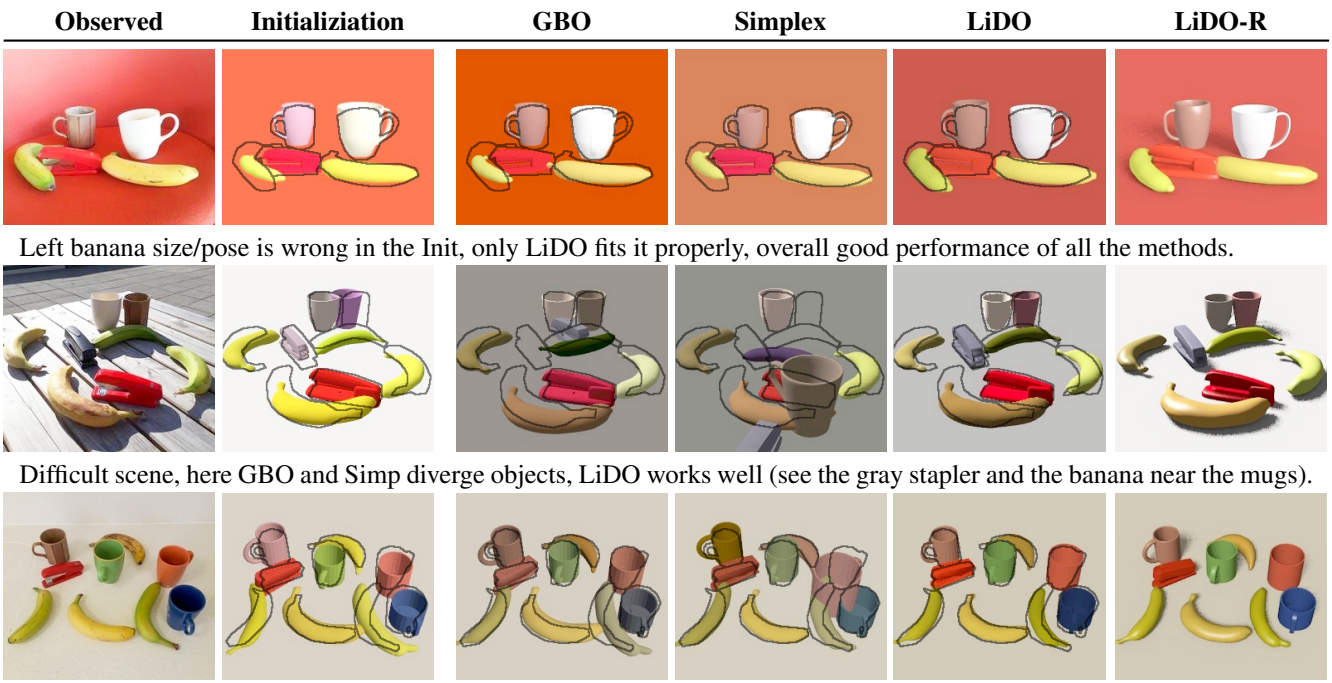


Figure 7. Example runs for Real dataset, the order is the same as in Figure 8. Also note that for each image the camera viewpoint is initialized accurately, and how similar the Observed and LiDO-R images are. Obtaining an exact match to the ground truth outline may be impossible because we only have a fixed set of shapes to choose from, none of which may match the actual object shape.

Results for Real Dataset are given in Table 3. Since real images are more noisy and difficult, GBO and Simp work poorly for IoU (there is a very minor improvement for objects, and no improvement for the ground plane). All the methods improve the pixel colours (MSE), but note this is becasue the pixel match is an explicit error measure for GBO & Simp. LiDO, which has been trained on synthetic data, transfers to work better with real images for all four measures.

Note the initialized CAD shapes for real images are well matched (see similar object shapes in Figure 9), even though these shapes were never observed during training. The objects and global LVs are then refined well, this was facilitated by introducing shape mismatch in the second noisy dataset source of LiDO (see sec. 4, Training Datasets).

The results in Tables 2 and 3 afford a direct comparison of the optimizers, all using the OpenGL renderer. However, we can also run LiDO with its "native" renderer Blender (shown as the LiDO-R column in Figs. 8 and 9), to compare the "system" performance. We calculated the MSE errors for Blender renderer (IoUs are the same for both renderers since only the appearance changes), these were similar to LiDO (for Synthetic dataset: 21.6 [ob] and 11.3 [gr], for Real dataset: 40.9 [ob] and 23.0 [gr]).

Figure 9 shows example runs and explanatory text for real image examples. In general the LiDO method obtain better results in a shorter test-time than the alternatives, and usually converges to a better configuration. LiDO also has the advantage that it can be trained to handle model mismatch, as shown in the real dataset experiments. Common reasons for the poor performance of the error-based methods are that: the updates can point in wrong direction when dealing with cluttered scenes and shadows in observed images; difficulties can arise from an inability to exactly match the target object with one of a different shape; predicted objects overlapping background or other objects may lead optimization to wrong configurations. LiDO is robust to such problems as it is directly learned to optimize the latent space. More examples are given in the supplementary material.

Conclusion: Above we have introduced a novel Learning Direct Optimization method for the refinement of a latent variable model that describes an image, and demonstrated a full framework for inferring a 3D representation of the scene from a single image. The main features of LiDO are: the advantage of not requiring an error metric E to be defined in image space, rapid convergence, and robust refine-

ment in the presence of noise and distractors. Our experiments show that LiDO method generally produces better solutions in shorter time than error-based optimization, and that it is better able to handle mismatch between the data generator and the fitted scene model. In the future work we will investigate whether using a richer input to the LiDO network, such as object masks, may help the network to distinguish object and background pixels for even more accurate inference.

References

[1] V. Blanz and T. Vetter. A morphable model for the synthesis of 3D faces. In *Proc. 26th Annual Conference on Computer Graphics and Interactive Techniques*, pages 187–194, 1999. 3

[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. Ieee, 2009. 13

[3] S. M. A. Eslami, N. Heess, T. Weber, Y. Tassa, K. Kavukcuoglu, and G. E. Hinton. Attend, Infer, Repeat: Fast Scene Understanding with Generative Models. In *Advances in Neural Information Processing Systems 29*, 2016. 2

[4] U. Grenander. *Lectures in Pattern Theory: Vol. 1 Pattern Synthesis*. Springer-Verlag, 1976. 2

[5] U. Grenander. *Lectures in Pattern Theory: Vol. 2 Pattern Analysis*. Springer-Verlag, 1978. 2

[6] R. Guo, C. Zou, and D. Hoiem. Predicting Complete 3D Models of Indoor Scenes. *arXiv preprint arXiv:1504.02437*, 2015. 2

[7] H. Izadinia, Q. Shan, and S. M. Seitz. IM2CAD. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 2

[8] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *CoRR*, 2014. 13

[9] T. D. Kulkarni, W. F. Whitney, P. Kohli, and J. B. Tenenbaum. Picture: A probabilistic programming language for scene perception. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 2

[10] M. M. Loper and M. J. Black. OpenDR: An Approximate Differentiable Renderer. In *Computer Vision–ECCV 2014*, pages 154–169. Springer, 2014. 4

[11] D. Marr and T. Poggio. A computational theory of human stereo vision. *Proc. R. Soc. Lond. B*, 204(1156):301–328, 1979. 2

[12] P. Moreno, C. K. I. Williams, C. Nash, and P. Kohli. Overcoming Occlusion with Inverse Graphics. In *Computer Vision-ECCV 2016 Workshops Proceedings Part III*, pages 170–185. Springer, 2016. LNCS 9915. 4

[13] L. Romaszko, C. K. I. Williams, P. Moreno, and P. Kohli. Vision-as-Inverse-Graphics: Obtaining a Rich 3D Explanation of a Scene from a Single Image. In *ICCV 2017 Geometry Meets Deep Learning Workshop*, 2017. 2, 6

[14] B. C. Russell, A. Torralba, K. P. Murphy, and W. T. Freeman. Labelme: A database and web-based tool for image annotation. *Int. J. Comput. Vision*, 77(1-3):157–173, May 2008. 4

[15] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus. Indoor Segmentation and Support Inference from RGBD Images. In *ECCV*, 2012. 3, 10

[16] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *International Conference on Learning Representations (ICLR)*, 2015. 3

[17] M. R. Stevens and J. R. Beveridge. *Integrating Graphics and Vision for Object Recognition*. Kluwer Academic Publishers, Boston, 2001. 2

[18] C. K. I. Williams, M. Revow, and G. E. Hinton. Instantiating deformable models with a neural net. *Computer Vision and Image Understanding*, 68(1):120–126, 1997. 2

[19] J. Wu, J. B. Tenenbaum, and P. Kohli. Neural Scene De-rendering. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 2

[20] I. Yildirim, T. D. Kulkarni, W. A. Freiwald, and J. B. Tenenbaum. Efficient analysis-by-synthesis in vision: A computational framework, behavioral tests, and comparison with neural representations. In *Thirty-Seventh Annual Conference of the Cognitive Science Society*, 2015. 2

[21] A. Yuille and D. Kersten. Vision as Bayesian inference: analysis by synthesis? *Trends in Cognitive Science*, 10(7):301–308, 2006. 2

Supplementary Material for: Learning Direct Optimization for Scene Understanding

Contents

A. Stochastic Scene Generator details

For each image we sample the global parameters and a number of objects which lie on a table-top plane. These are rendered at 256 $\times$ 256 resolution. We sample the camera height and elevation uniformly in the appropriate ranges: $\alpha \in [0°, 75°]$, $h \in [5, 75]$ cm. Illumination is represented as uniform lighting plus a directional source, with the strength of the uniform light $\in [0, 1]$, the strength of the directional light $\in [0, 2]$, with azimuth $\in [0°, 360°]$ and elevation $\in [0°, 90°]$ of the directional light.

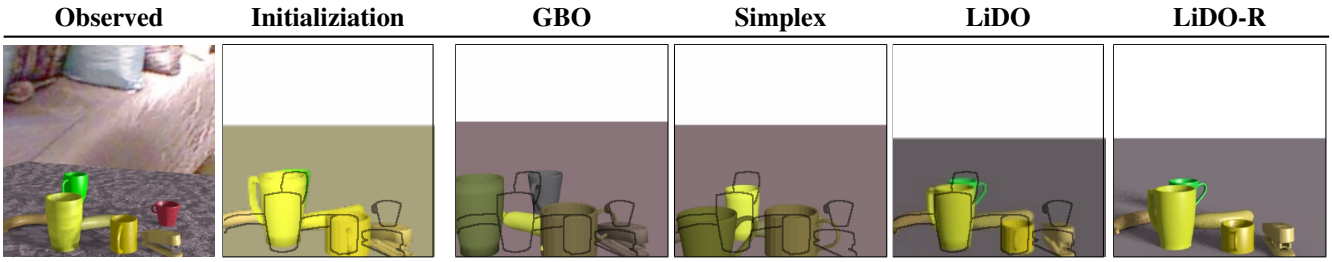
To sample a scene we first select a target number of objects (between 4 and 7). We then sample the camera parameters and the plane colour. Objects are added sequentially to the scene, and a new object is accepted if at least a half of it is present in the image, it does not intersect other objects, and is not occluded by more than 50%. If it is not possible to place the target number of objects in the scene (e.g. when a camera is pointing downwards from a low height) we reject the scene. For each object we sample its class (stapler, mug or banana), shape (one of 6/15/8 shapes from ShapeNet[6]), colour (albedo), rotation, and scaling factor so that stapler length is $\in [12, 16]$cm, mug diameter in $\in [7, 10]$cm, banana length $\in [15, 20]$cm.

Below we describe the process of sampling realistic colours for our scenes. Initially we experimented with sampling from a uniform distribution but it often results in pastel colours, close to gray. Therefore we use a collection of 17 predefined CSS/HTML colours and sample a pair of them with a random mixing proportion. This samples a variety of colours with frequent strong colours (where the RGB value is either close to 0 or 1), as these are common choices for everyday objects. Afterwards, we add a uniform noise of ± 0.2 to the RGB coefficients and clip if necessary. We use this scheme to sample colours of staplers and mugs, for bananas we fix one of the components to be yellow, and the other component to another likely one (yellow, orange, green, brown, olive, or white). For ground plane colours we fix one of the components to be white so as to obtain bright colours more frequently, yet note that still all the colours are possible (e.g. black when a mixing proportion of white is low and uniform noise added is negative).

Background images are taken from the NYU Depth V2 dataset [15]. In addition random textures are applied to objects by converting a set of textures to greyscale and applying them at a random scaling on the surface via multiplication of the initial albedo and the texture intensity. The resulting mean colour is stored as the final albedo.

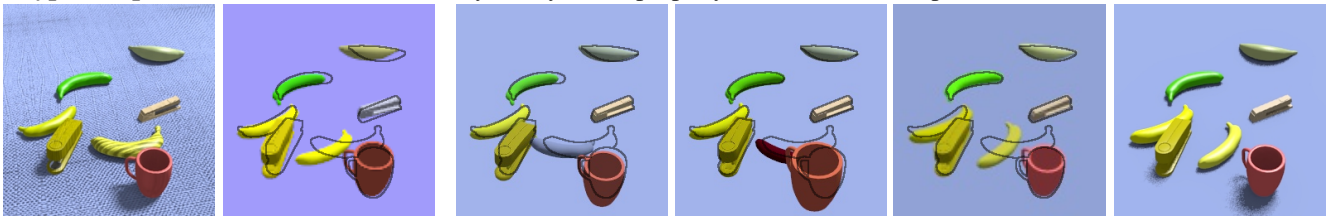
[6] https://www.shapenet.org/

B. More examples of prediction for Synthetic Dataset

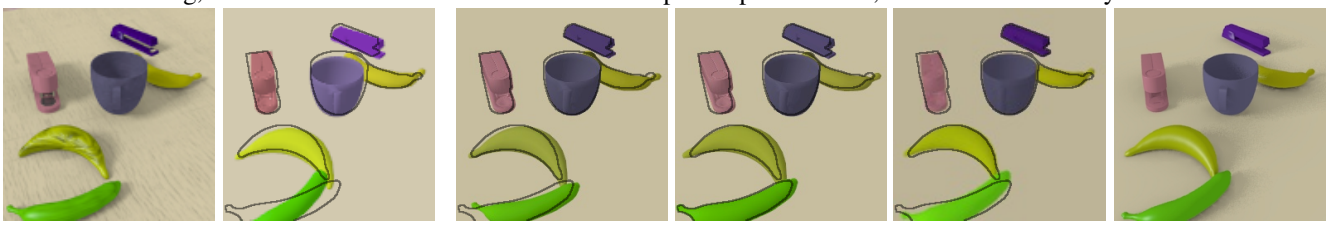


Poor initialization: GBO and Simp converge to wrong configurations of object poses and colours, while LiDO is robust to initialization errors; note here the initialized object sizes are wrong and LiDO improves all the detected objects.

Typical input (1): All methods work fine, yet only LiDO properly refines the blue stapler.



Typical input (2): There are 7 objects, 6 object converge properly for all the methods, the initialized position of the bottom banana in wrong, all the methods fail to fix it: GBO and Simp corrupt the colour, LiDO maintains the yellow colour.



Typical input (3): All objects are initialized well and converge properly, except the green banana for which the azimuthal rotation is wrong, all the methods improve the pose. Note well predicted shadows in LiDO-R.

Strong textures and shadows: For GBO the blue stapler (middle) diverges, for Simplex it becomes brown, LiDO is robust to such distractors: stapler pose/size improves, both mugs become smaller with proper colours (also compare Observed and LiDO-R).

Figure 8. Example runs for Synthetic dataset, showing from left: the observed input image, the initialization (OpenGL), and images after refinement for GBO, Simplex and LiDO using OpenGL renderers, and LiDO using Blender renderer (LiDO-R). We overlay black contours of the ground truth object masks on top of each OpenGL image to ease the comparison of object poses.

C. More examples of prediction for Real Dataset

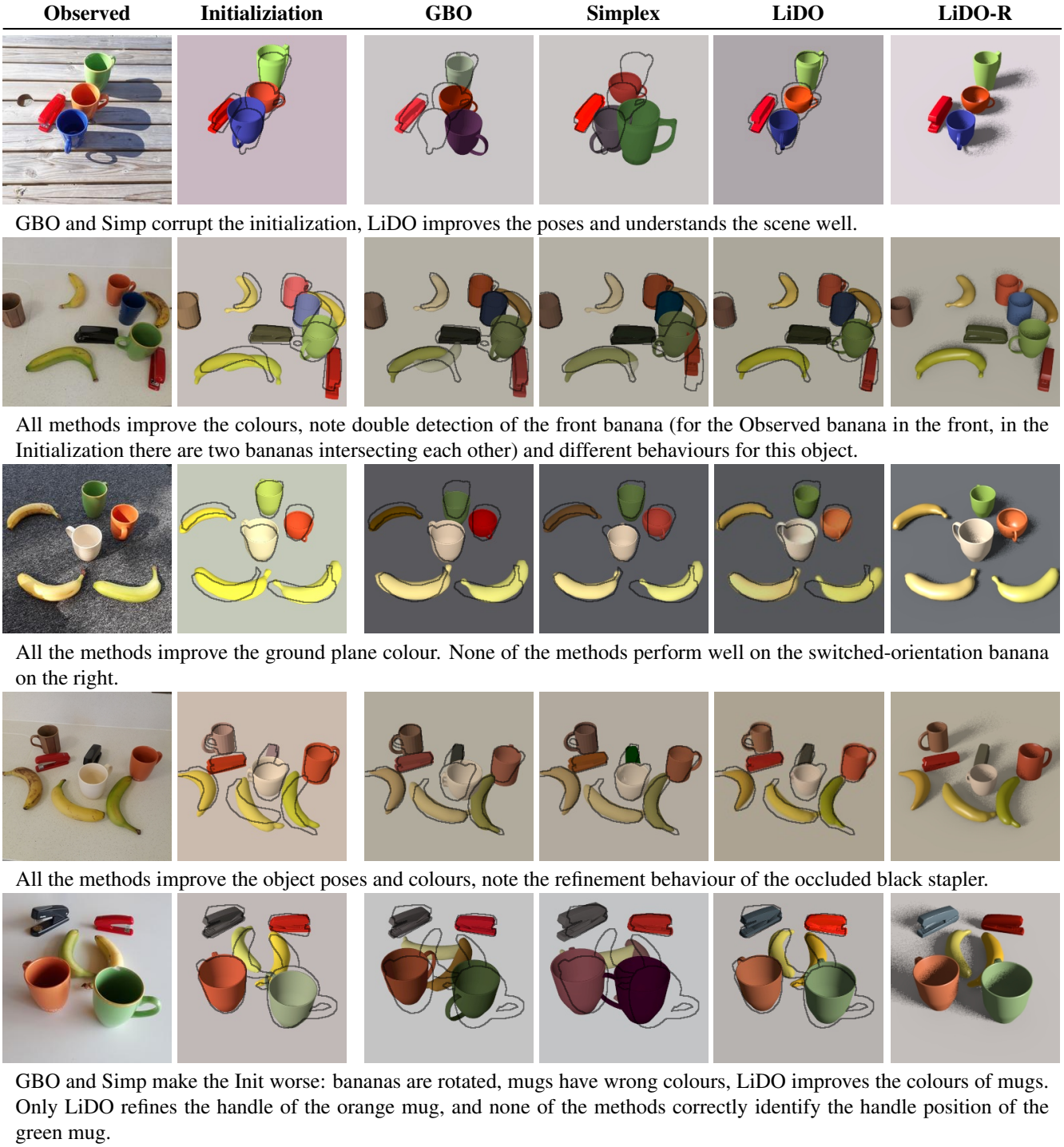


Figure 9. Example runs for Real dataset, the order is the same as in Figure 8. Also note how similar the Observed and LiDO-R images are.

D. CNN Architectures

D.1. Detector and Initialization Networks

Table 4 shows the network configurations and learning rates used for training. We use all 13 convolutional layers of VGG-16 as the core but on 128×128 pixel input. We use only the first three pooling layers, and the VGG weights are kept fixed. Each patch is centred at the predicted object contact point. Area outside the image frame is given as value 0, we use this scheme for all the initialization networks, and then also for the LiDO network.

Layer description, where N denotes the number of units, and K the filter size:

- Convolutional layer: C-N-K,
- Fully connected layer, with its input concatenated with the detector output (position of the detection plus object size): Fd-N,
- Sigmoid output layer: Sigm-N,
- Softmax output layer: Softmax-N.

VGG layer activations are *ReLU*, layers on top of VGG use *tanh* activations. VGG uses padding, but in our convolutional layers we do not use it. The fully connected layers of the detector networks are implemented as filter 1×1 convolutional layers, so they can be efficiently applied in a sliding window manner.

Colour networks (for objects and for ground plane) are simple 3-layer CNNs with leaky rectify activations ([Input, C-27-6 (stride 6, dropout $p = 0.5$), Fd-40, F-40, Sigm-3]), trained with 0.0001 learning rate.

For azimuthal rotation, we predict the rotation discretized into 18 bins of 20 degrees.

The implementation is in Python (Theano) and we use Adam[8] optimizer with L2 or categorical cross-entropy loss to train the networks. This is used for the following separately trained networks: detector (Class) network, detector (Size) network, and the initialization networks: global LVs network (Lighting and Camera), and object Azimuth, object Shape, and object Colour networks (one network per each object class for object LVs). We use dropout in the detector networks with $p = 0.5$ in all the 3 convolutional layers (on top of VGG ones), and after the first one for the initialization network.

The detector is trained on 30k positive patches with objects contact point centred (with small noise of $\pm$ 8 pixels added), and 90k negative patches (30k random patches, 30k patches with the centre nearby the contact point of other objects, and 30k random crops from the ImageNet[2] dataset). The detector is run on the dataset of 10k images to produce the the training dataset for the initialization networks. Afterwards, we apply the initialization networks on another 10k images to produce the dataset for LiDO (the first source).

D.2. LiDO Prediction Network

Table 5 shows the network configurations and learning rates used for training. The implementation is in Python (TensorFlow).

Layer description, where N denotes the number of units, and K the filter size:

- Convolutional layer: C-N-K,
- Fully connected layer: F-N,
- Fully connected layer, with its input concatenated with the current LVs $\mathbf{z}$: Fz-N,
- Max-pooling layer: MaxPool-K,

For each object patch, the inputted current LVs $\mathbf{z}$ (standardized) are: object LVs (discrete class and shape one-hot-encoded), global LVs (as predicted by the object, denoted G_O), global LVs (as used in the render after voting of all the objects, denoted G_V), plus their difference $G_O - G_V$.

To allow convergence the object pose is trained in the object's current coordinates/frame (to predict the delta in object position and rotation from the current value). CNNs are trained together with an L1+L2 error for continuous LVs and categorical cross-entropy for discrete ones. We add the L1 loss to the L2 loss, as when making predictions multiple times small errors aggregate, and L1 appropriately punishes small errors during training. To calculate the overall network loss we sum up each L1+L2 loss and the cross-entropy loss (for which we used a scaling factor 0.2). We use Adam optimizer with learning rate 0.0003.

Detector		**Initialization network**			
Class	Size	Shape	Azimuth (Ob/Lighting)	Lighting	Camera
Input $128 \times 128 \times 3$ (Image)					
VGG-16 (all 13 convolutional layers)					
C-50-6 *(separate per network)* C-50-6 *(separate per network)* C-50-6 *(separate per network)*					
Fd-200 Softmax-4	Fd-200 Sigm-1	Fd-50 Softmax-6/15/8	Fd-50 Softmax-18	Fd-100 Sigm-3	Fd-50 Sigm-2

Learning rate					
Class	Size	Shape	Azimuth	Lighting	Camera
0.001	0.0002	0.001	0.0003	0.0001	0.0001

Table 4. The configurations of the main models and learning rates.

LiDO Prediction Network					
Position	Size	Azimuth	Lighting	Camera	Colour (Ob/Gr)
Input $64 \times 64 \times 9$ (2 images plus their difference, stacked)					
		C-32-3 *(shared)*			C-32-3 (stride 2)
		C*-64-3 *(shared)*			C-64-3 (stride 2)
		MaxPool-2 *(shared)*			
		C*-128-3 *(shared)*			C*-128-3 (stride 2)
		MaxPool-2 *(shared)*			
		C*-64-3 *(separate per network)*			
		MaxPool-2 *(separate per network)*			
		C-32-3 *(separate per network)*			
		Fz-40 *(separate per network)*			Fz-40
		Fz-40 *(separate per network)*			Fz-40
		Fz-40 *(separate per network)*			Fz-40
Fz-2	Fz-1	Softmax(Fz-360)	Fz-5	Fz-2	Fz-3

Table 5. The configurations of the network. Non-colour networks: the first 5 layers are shared across all the 5 sub-networks, all the sub-networks on top of them have the same set-up. Colour networks are simpler (as per the initialization colour networks) and have less layers, and use *ReLU* activations, while non-colour networks use *tanh* activations. For ligthing the output length is 5 becasue we predict: (uniform component strength, directional component: strength, elevation, sin(azimuth), cos(azimuth)). We use dropout with $p = 0.5$ after the convolutional layers denoted with *. The whole LiDO Prediction Network is trained together.