# Monocular Depth Estimation and Feature Tracking

Bryan Chiang and Jeannette Bohg

February 9, 2022

## 1 Overview

In the previous section, we discussed the idea of representation learning which leverages unsupervised and self-supervised methods to learn an intermediate, low-dimensional representation of high-dimensional sensory data. These learned features can then be used to solve downstream visual inference tasks. Here, we will examine how representation learning in the context of two common computer vision problems: monocular depth estimation and feature tracking.

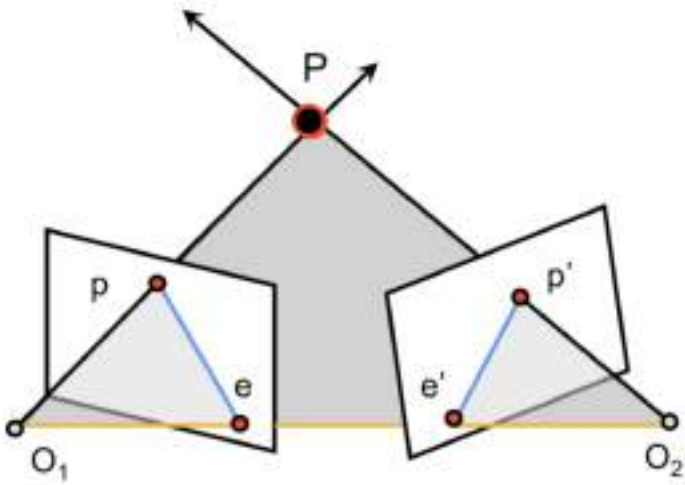
## 2 Monocular Depth Estimation

### 2.1 Background

Depth estimation is common computer vision building block that is crucial to tackling more complex tasks, such as 3D reconstruction and spatial perception for grasping in robotics or navigation for autonomous vehicles. There are numerous active methods for depth estimation such as structured light stereo and LIDAR (3D point clouds), but we focus on depth estimation through passive means here since it does not require specialized, possibly expensive hardware and can work better in outdoor situations.

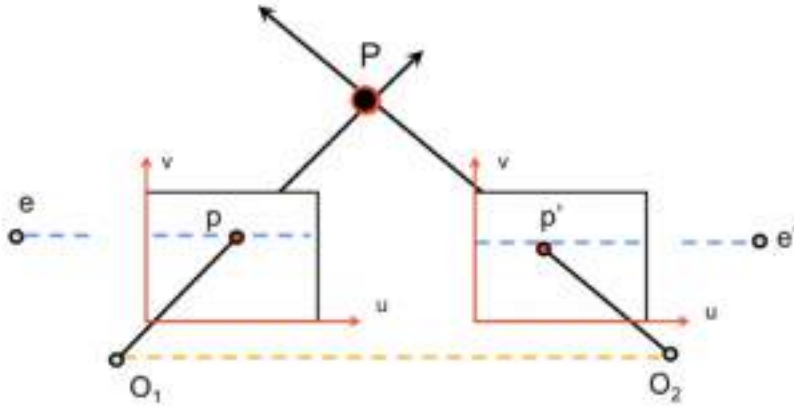
We can view depth estimation as a special case of the **correspondence problem**, which is fundamental in computer vision. It involves finding the 2D locations corresponding to the projections of a physical 3D point onto multiple 2D images taken of a 3D scene. The 2D frames can be captured from multiple viewpoints either using a monocular or a stereo camera.

Figure 1: Epipolar geometry setup with a stereo camera.



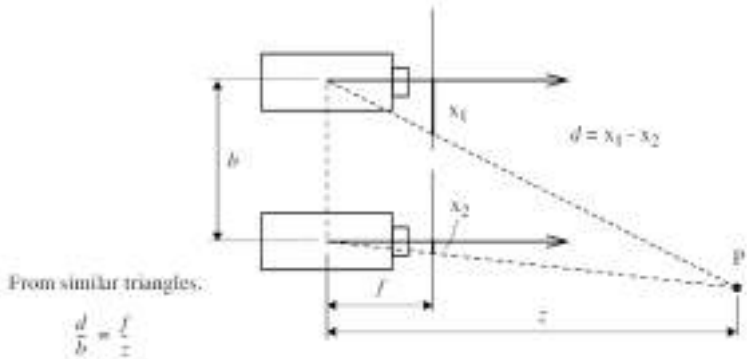

One way to solve for correspondences is through epipolar geometry, illustrated in Figure 1, as we have seen earlier in the course. Recall that given the camera centers $O_1$ and $O_2$ and a 3D point in the scene called $P$, $p$ and $p'$ represent the projection of $P$ into the image planes for the left and right cameras, respectively. Given $p$ in the left image, we know that the corresponding point in the right image $p'$ must lie somewhere on the epipolar line of the right camera, which we defined as the intersection of the image planes with the epipolar plane. This is known as the **epipolar constraint**, which is encapsulated by the fundamental (or essential) matrix between the two cameras since $F$ gives us the known epipolar lines. In the context of depth estimation, we often assume that we are dealing with a stereo setup and rectified images. The epipolar lines are then horizontal and the **disparity** is defined as the (horizontal) distance between the two corresponding points such that $d = p'_u - p_u$ and $p_u + d = p'_u$ (note that $p'_u > p_u$ for all $P$).

Figure 2: Rectified setup with parallel image planes and epipoles at infinity.




We then see that there is simple inverse relationship between disparity and **depth**, which is defined as the z-coordinate of $P$ relative to the camera centers. We can use similar triangles as illustrated in Figure 3 to obtai $z = \frac{fb}{d}$, where $f$ is the focal length of the cameras and $b$ is the length of the baseline between the two cameras (yellow dashed line in Fig 2). Assuming $b$ and the camera intrinsics $K$ are known, we see that if we are able to find correspondences between two rectified images, illustrated in Figure 2, we know their disparity and thus their depth. One approach to identify correspondences $p'$ for $p$ is to run a simple 1D-search along the epipolar line in the other image, using pixel or patch similarities to determine the location of the most likely $p'$. However, such a naive method would run into issues such as occlusions, repetitive patterns, and homogeneous regions (i.e. lack of texture) on real-world images. We turn to modern representation learning methods instead.

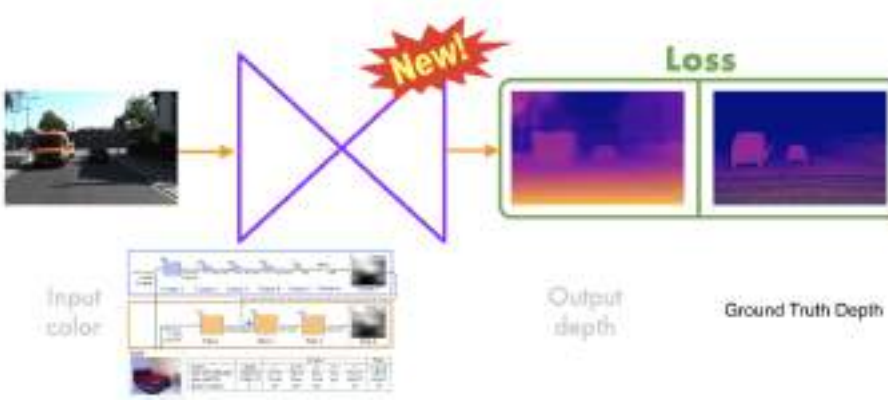
Figure 3: Relationship between depth and disparity.



## 2.2 Supervised Estimation

Here, we focus on the task of **monocular (single-view) depth estimation**: we only have a single image available at test time, and no assumptions about the scene contents are made. In contrast, **stereo (multi-view) depth estimation** methods perform inference with multiple images. Monocular depth estimation is an underconstrained problem, i.e. geometrically it is impossible to determine the depth of each pixel in the image. However, humans can estimate depth well with a single eye by exploiting cues such as perspective, scaling, and appearance via lighting and occlusion. Therefore, when exploiting these cues, computers should be able to infer depth with just a single image. Fully supervised learning methods, illustrated in Figure 4, rely on training models (CNNs) to learn to predict pixel-wise disparity over pairs of ground truth depth and RGB camera frames [8, 11]. The training loss captures the similarity between the predicted and ground-truth depth and the learning method aims to minimize that loss. Since monocular methods can only capture depth up-to-scale, [1] proposes using a scale-invariant error to prior monocular methods.

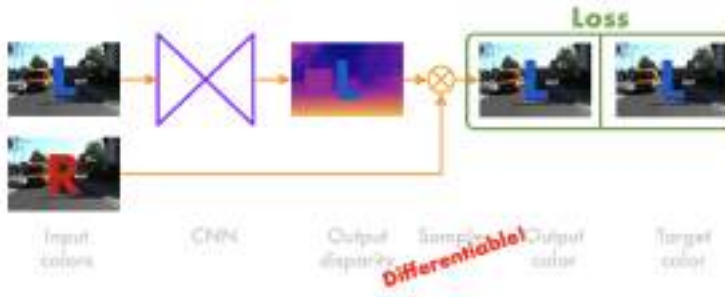
Figure 4: Vanilla supervised learning setup used in [1, 8, 11]. Figure from [3].




## 2.3 Unsupervised Estimation

While supervised learning methods achieved decent results, they are limited to scene types where large quantities of ground depth data are available. This motivates unsupervised learning methods which only require the input RGB frame data and a stereo camera with known intrinsics, and thereby avoid the need for expensive labeling efforts. Here, we examine the approach proposed in [3] as a case study. Instead of using the difference between reconstructed and ground truth depth as the loss, the base unsupervised formulation casts the problem as image reconstruction through the use of an autoencoder to minimize the difference between the input reference image and a reconstructed version $\tilde{I}^l$.

Figure 5: Unsupervised baseline network. The differentiable sampler enables end-to-end optimization.

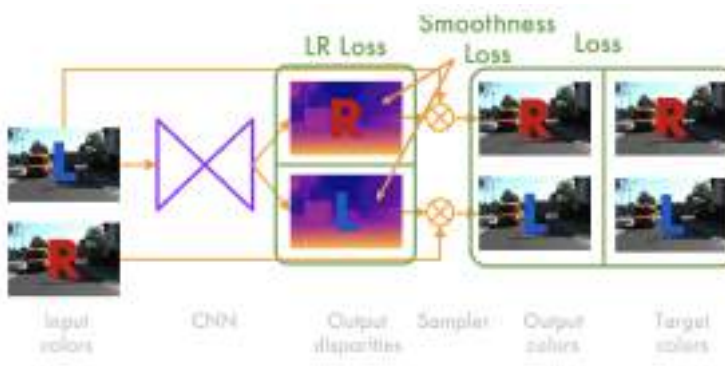



The baseline network, shown in Figure 5, only reconstructs the left image $I^l$. The input to the network is $I^l$, the left frame. A CNN maps the left frame to an output to $d^l$, the disparity (displacement) values required to warp the right image $I^r$ into the left image. The disparity values are then used as an intermediate representation to reconstruct the left image, $\tilde{I}^l$, by sampling from the right image. We could sample from the right image as $\tilde{I}^l(u,v) = I^r(u - d^l(u,v), v)$, but $d^l(u,v)$ is not necessarily an integer so the pixel at the exact new location may not exist. To perform end-to-end optimization to train the network, a fully (sub-) differentiable bilinear sampler [6] is used, illustrated in Figure **??**.

Note that both the left and right images are used to train the network, but only the left image is required to infer the left-aligned depth at test time. The network architecture is fully convolutional, consisting of an encoder followed by a decoder, which outputs multiple disparity maps at doubling spatial scales. For instance, if the first disparity map is of resolution $(D_h, D_w)$, the second output disparity map would be of resolution $(2D_h, 2D_w)$.

Going beyond the baseline, [3] propose a novel architecture to reconstruct both the left and right frames. This is illustrated in Figure 6, and allows for the introduction of a more complex loss term to improve image quality of the disparity.

Figure 6: Proposed novel unsupervised network setup.




The CNN takes in the input left frame $I^l$ and computes the left-aligned disparity $\tilde{d}^l_{i,j}$ and the right-aligned disparity $\tilde{d}^r_{i,j}$. The right-aligned disparity map contains the horizontal displacement values needed to reconstruct the right frame from the left frame, and similarly for the left-aligned disparity. Using the sampler from before, we reconstruct $\tilde{I}^l$ from $d^l$, $\tilde{I}^r$ from $d^r$ and compute the loss between both the left and right input images and reconstructed images (four images in total). The total loss $C$ is the sum of the loss at each scale $s$, $C = \sum_s C_s$.

$$C_s = \alpha_{ap}(C^l_{ap} + C^r_{ap}) + \alpha_{ds}(C^l_{ds} + C^r_{ds}) + \alpha_{lr}(C^l_{lr} + C^r_{lr}) \tag{1}$$

We see that there are three components. Each one has a scaling factor at the front and left and right variants, so while the following equations describe the procedure for the left reconstructed image, know that we also compute the terms in the same manner for the right image, albeit with $\tilde{I}^r$ instead of $\tilde{I}^l$.

The first term, $C_{ap}$, is the reconstruction loss.

$$C^l_{ap} = \frac{1}{N}\sum_{i,j} \alpha \frac{1 - \text{SSIM}(I^l_{ij}, \tilde{I}^l_{ij})}{2} + (1-\alpha)\|I^l_{ij}, \tilde{I}^l_{ij}\| \tag{2}$$

It iterates through every pixel at location $i, j$ and computes a weighted combination of 1) the L1 difference between the reconstructed and ground truth image and 2) the negative Structural Similarity Index (SSIM), which computes the similarity between two corresponding patches centered at the pixel through a combination of the patch's luminance, contrast, and structure. The average is then taken. In the paper, $\alpha = 0.85$, indicating that the SSIM is emphasized more than the L1 difference.

Next, the second term, $C_{ds}$, is the smoothness loss.

$$C^l_{ds} = \frac{1}{N}\sum_{i,j} |\partial_x d^l_{ij}| e^{-\|\partial_x I^l_{ij}\|} + |\partial_y d^l_{ij}| e^{-\|\partial_y I^l_{ij}\|} \tag{3}$$

We see that the loss again iterates through every pixel $x, y$ and penalizes the gradients of the disparity map in both x and y directions ($\partial_x d^l_{ij}, \partial_y d^l_{ij}$). This has the effect of reducing abrupt changes in disparity (high gradients), resulting in a smoother disparity map. However, we would want high disparities at object edges when we transition from one object at a specific depth to another object at a different depth. To account for this, we relax the smoothness penalty if we detect an edge in the original image: if the image gradient is high, then $e^{||\partial_x I^l_{ij}||}, e^{||\partial_y I^l_{ij}||}$ will be small. Note that we take the L2 of the image gradient, but the L1 of the disparity map gradients.

Finally, the third term, $C_l r$, is the left-right consistency loss.

$$C^l_{lr} = \frac{1}{N} \sum_{i,j} |d^l_{ij} - d^r_{ij+d^l_{ij}}| \tag{4}$$

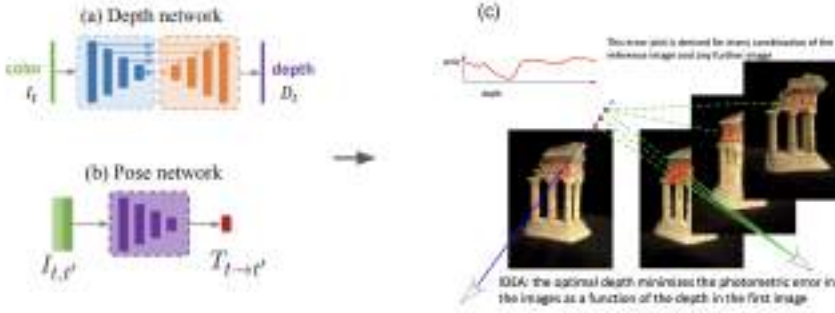
The intuition here is that the absolute distance between corresponding pixels in two image planes , i.e. disparity, should be the same whether computed right to left or left to right. Therefore, we should penalize differences between the predicted disparities for the left and right images that are outputted from the network. To achieve this, we iterate through each pixel $i, j$ and calculate the L1 distance between left-aligned disparity $\tilde{d}^l_{i,j}$ and corresponding right-aligned disparity $d^r_{i,j+d^l_{ij}}$. Note here that we are using the disparities to "sample" from the disparity maps, not the images. In the results, the authors demonstrate that this unsupervised setup outperforms both the baseline and existing state-of-the-art fully supervised methods.

## 2.4 Self-Supervised Estimation

Unsupervised methods have been followed by self-supervised learning for depth estimation; here we review a follow-up paper [4].

Here, the depth estimation problem is framed as novel view synthesis problem. Given a target image of a scene, the learned pipeline aims to predict what the scene would look like from another viewpoint. Depth is used as an intermediate representation to obtain the novel view, so the depth map can be extracted from the pipeline at test time for use in other tasks. In the monocular setup, we rely on monocular video as our supervisory signal: our target image is a color frame $I_t$ at time $t$, and the images from other viewpoints, or the source views, are the temporally adjacent frames $I_{t'} \in \{I_{t-1}, I_{t+1}\}$. Since we are predicting future and past inputs from the current input, as opposed to reconstructing the entire original input, this is an example of self-supervised learning.

Figure 7: Self-supervised depth estimation pipeline. (a) is the depth network architecture for extracting the depth map, (b) predicts the pose transformation from frame to frame, and (c) describes the ambiguity in depth for reprojection in all frames.




The pipeline is illustrated in Figure 7. First, we obtain the intermediate depth representation: given the color input $I_t$, we run it through a convolution encoder-decoder architecture to obtain the depth map $D_t$, as shown in Figure 7a. In parallel, as shown in Figure 7b, we iterate over the source past and future frames $I_{t'}$ and compute the relative pose $T_{t\to t'}$ indicating the transformation from $I^t$ to $I^{t'}$. Assuming the same camera intrinsics $K$ for all target and source views, we can obtain our novel views for $t-1, t+1$ through reprojection, as shown in Figure 7c. Given $K$ and the predicted depth $D_t$ for all pixels, we can backproject specific 2D image coordinates $(u, v)$ to the 3D point location. Since we know the relative pose $T_{t\to t'}$, we can then project our 3D point in the image plane of source view $I_t$ to obtain the 2D coordinates $(u', v')$ in the novel image plane $I_{t'}$. Since we have the monocular video, the ground truth $I_{t'}$ is known. Our "synthesized" view for $I_{t'}$ would then be $\tilde{I}_{t'}(u', v') = I_t(u, v)$. The image patches between the "synthesized" view, $\tilde{I}_{t'}$ and the ground truth view, $I_{t'}$, image patches should be visually similar, which we measure through the **photometric error**, defined as

$$pe(I_a, I_b) = \frac{\alpha}{2}(1 - \text{SSIM}(I_a, I_b)) + (1-\alpha)\|I_a - I_b\|_1 \tag{5}$$

The photometric error finds the optimal depth map $D_t$ such that the reprojection error for every 2D coordinate $pe(\tilde{I}_{t'}(u', v'), I_{t'}(u', v')$ is minimized. We then sum up the photometric error for pairs of $(t, t'), t' \in \{t-1, t+1\}$.

$$L_p = \sum_{t'} pe(I_t, I_{t'\to t}) \tag{6}$$

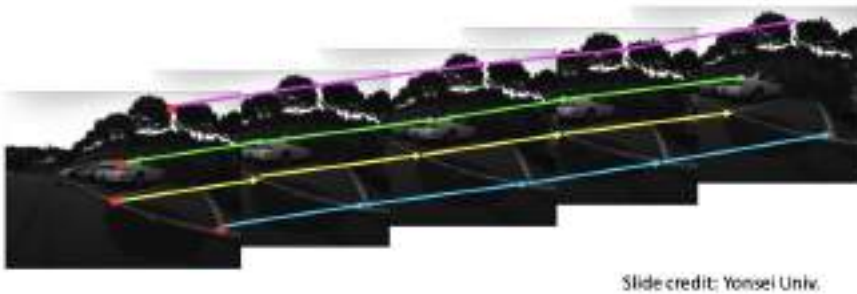
Similar to the last paper, the photometric error, Equation 5, is a weighted combination of the structural similarity index and the L1 difference. In the results, the authors demonstrate how this self-supervised setup outperform existing unsupervised and other self-supervised approaches.

# 3 Feature Tracking

# 4 Motivation

Given a sequence of images, the task of **feature tracking** involves tracking the locations of a set of 2D points across all the images, illustrated in Figure **??**. Just like depth estimation, we can view feature tracking as yet another instance of solving for correspondence problem across an image sequence.

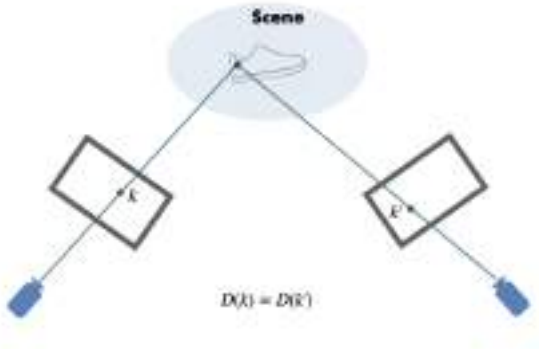
Figure 8: Feature point tracking over time.




Feature tracking can be used to trace the motion of objects in the scene. We make no assumptions about scene contents or camera motion; the camera may be moving or stationary, and the scene may contain multiple moving or static objects.

The challenge in feature tracking lies identifying which feature points we can efficiently track over frames. The appearance of image features can change drastically over frames due to camera movement (feature completely disappears), shadows, or occlusion. Small errors can also accumulate as the appearance model for feature tracking is updated, leading to drift. Our goal is to identify distinct regions (called features or sometimes keypoints) that we can track easily and consistently, and then apply simple tracking methods to continually find these correspondences.

Figure 9: Descriptors for feature tracking.



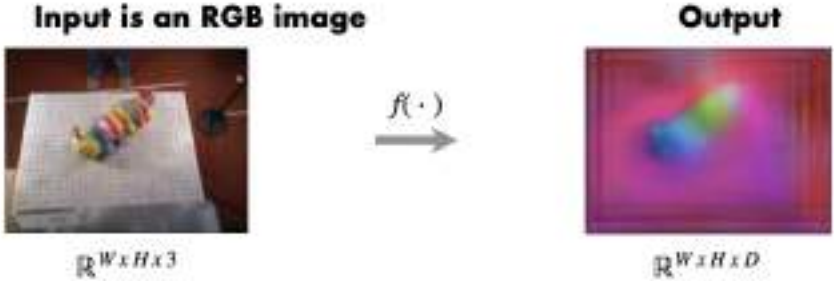

Traditionally, distinct features in images that are easy to track have been detected and tracked using hand-designed method [5, 10, 9, 12, 7]. Specifically, these good features then need to be encoded into a so-called *descriptor* that lends itself well for fast matching with features in other images, i.e. finding correspondences. These methods are also sparse, only yielding descriptors for a subset of pixels in the image. In this section, we will look at how representation learning can also be used to learn descriptors of image features rather than hand-designing them.

In Figure 9, $D(k)$ gives a D-dimensional representation of pixel $k$ (often incorporating neighboring information). Since $k, k'$ correspond to the same 3D point, we would expect that they would be visually similar, so their descriptors should also be the same $D(k) = D(k')$ even though the images are captured from different viewpoints. We can then match pixels in different frames based on the similarity between their descriptors.

# 5 Learned Dense Descriptors

We examine the method for learning dense descriptors proposed in [2].

Figure 10: Representation of dense descriptors.




Given an input color image, we want to learn a mapping $f(\cdot)$ that outputs a $D$-dimensional descriptor for every pixel in the color image. "Dense" here means that we have a descriptor for every point in the input image, not just a sparse set. For visualization purposes in Figure 10, the $D$-dimensional descriptors are mapped to RGB through dimensionality reduction. In practice, $f(\cdot)$ is a learned neural network with a convolutional encoder-decoder architecture. The network is trained on pairs of images of the same object from different views $(I_a, I_b)$ using a **pixel-contrastive loss**, which attempts to mirror the "contrast" in pixels by minimizing the distance for similar descriptors and maximizing the distance for different descriptors.

Figure 11: Loss for matches. The blue arrow indicates a matching correspondence between the two points at the end of the arrow.

We assume that we are given a list of correspondences (here called matches) for an image pair. We run the network to compute the descriptors for all points in the image. For each ground truth match, we calculate the L2 distance between the descriptors at the two corresponding points. We want to minimize the distance in descriptor space $D(I_a, u_a, I_b, u_b)^2$.

$$L_{\text{matches}}(I_a, I_b) = \frac{1}{N_{\text{matches}}} \sum_{N_{\text{matches}}} D(I_a, u_a, I_b, u_b)^2 \tag{7}$$

Figure 12: Loss for non-matches. The blue arrow a pair of non-corresponding points.

For the contrastive part, we also compute the loss term for non-matches (pairs of points that do not correspond to each other). Here, we want to maximize the distance between non-corresponding points (the max operation maximizes this distance up to $M$, the maximum distance), given by

$$L_{\text{non-matches}}(I_a, I_b) = \frac{1}{N_{\text{non-matches}}} \sum_{N_{\text{non-matches}}} \max(0, M - D(I_a, u_a, I_b, u_b)^2) \tag{8}$$

Assuming the true correspondence is known for $u_a$, it is easy to find pairs of non-matches: we can just sample arbitrary points from $I_b$ that are not the corresponding point. Note that in Figure 12, $u_b$ refers to a non-matched point, while in Figure 7, $u_b$ refers to the matched point. The total loss is then the sum of the two

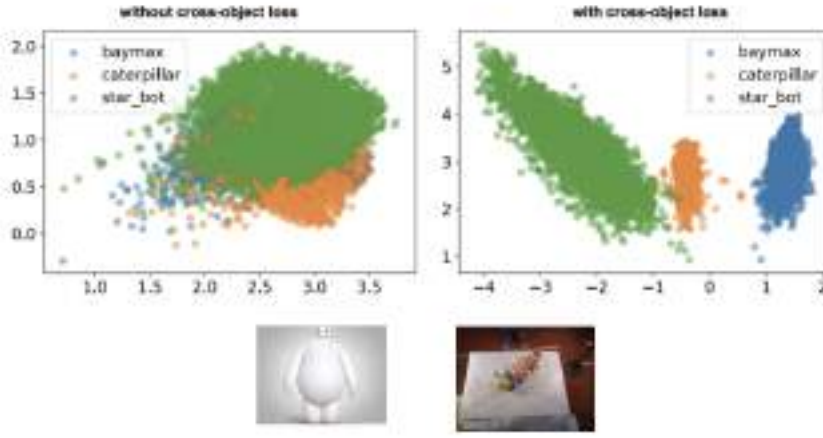
$$L(I_a, I_b) = L_{\text{matches}}(I_a, I_b) + L_{\text{non-matches}}(I_a, I_b) \tag{9}$$

The challenge lies in cheaply obtaining ground truth correspondences at scale with minimal human assistance. To this end, the authors propose using a robotic setup to perform autonomous and self-supervised data collection.

Figure 13: Robotic arm capturing different views and corresponding poses of a stationary object for training.

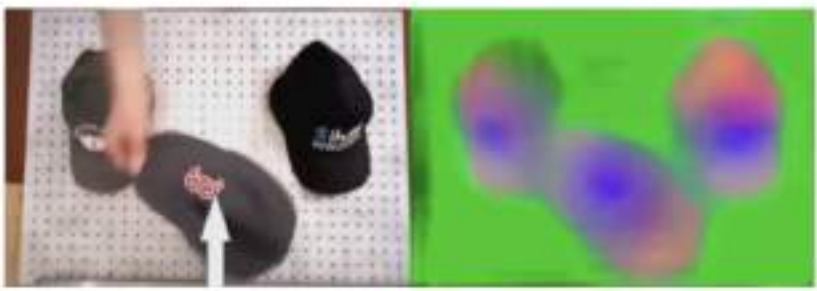

A robotic arm is used to capture images of a stationary object at various poses, illustrated in Figure 13. Since the forward kinematics of this precise robotic arm are known, we have matched pairs of camera pose and the corresponding view. 3D reconstruction is performed using all views to obtain a 3D model of the object. Using the camera poses, 3D points, and images, we can now generate as many ground-truth correspondences as we want. The network is trained using Equation 9 in combination with several other tricks such as background randomization, data augmentation, and hard-negative scaling.

Figure 14: Cross-object loss. The two images at the bottom correspond to the distinct cluster in the plot on the right (blue, orange) when using the cross-object loss.




If we only train on pairs of the same object, the learned descriptors for different objects overlap when they shouldn't since they correspond to completely different entities. If we incorporate a cross-object loss (pixels from images of two different objects are all non-matches), then we see distinct clusters forming in the descriptor space, as show in Figure 14.

Figure 15: Class-consistent descriptors.



In contrast, we would want objects from the same class of objects to exhibit similar descriptors although the visual appearance may not be the same. We see that our network is capable of learning this: while hats have different colors and designs, their descriptors share the same structure and color.

# References


[1] David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. *arXiv preprint arXiv:1406.2283*, 2014.

[2] Peter R Florence, Lucas Manuelli, and Russ Tedrake. Dense object nets: Learning dense visual object descriptors by and for robotic manipulation. *arXiv preprint arXiv:1806.08756*, 2018.

[3] Clément Godard, Oisin Mac Aodha, and Gabriel J Brostow. Unsupervised monocular depth estimation with left-right consistency. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 270–279, 2017.

[4] Clément Godard, Oisin Mac Aodha, Michael Firman, and Gabriel J Brostow. Digging into self-supervised monocular depth estimation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3828–3838, 2019.

[5] C. Harris and M. Stephens. A combined corner and edge detector. In *Proceedings of the 4th Alvey Vision Conference*, pages 147–151, 1988.

[6] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. *Advances in neural information processing systems*, 28:2017–2025, 2015.

[7] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[8] Nikolaus Mayer, Eddy Ilg, Philip Hausser, Philipp Fischer, Daniel Cremers, Alexey Dosovitskiy, and Thomas Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4040–4048, 2016.

[9] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(10):1615–1630, 2005.

[10] Krystian Mikolajczyk and Cordelia Schmid. Scale & affine invariant interest point detectors. *Int. J. Comput. Vision*, 60(1):63–86, oct 2004.

[11] Ashutosh Saxena, Min Sun, and Andrew Y Ng. Make3d: Learning 3d scene structure from a single still image. *IEEE transactions on pattern analysis and machine intelligence*, 31(5):824–840, 2008.

[12] Jianbo Shi and Carlo Tomasi. Good features to track. *IEEE Conference on Computer Vision and Pattern Recognition*, pages 593–600, 1994.