Internship Report: Self-supervised Depth Denoising

Claudius Kienle[1]
Autonomous Learning Robots Lab
Adenauerring 4
Karlsruhe Institute of Technology
Karlsruhe, Germany
Email: claudius.kienle@student.kit.edu

David Petri[2]
Autonomous Learning Robots Lab
Adenauerring 4
Karlsruhe Institute of Technology
Karlsruhe, Germany
Email: david.petri@student.kit.edu

***Abstract*—[12] Depth cameras are frequently used in robotic manipulation, e.g. for visual servoing. The quality of small and compact depth cameras is though often not sufficient for depth reconstruction, which is required for precise tracking in and perception of the robots working space. In this work, we applied a self-supervised depth denoising approach presented by Shabanov et al. [1] to denoise depth maps of YCB Objects coming from a lower-quality sensor, using depth maps of higher-quality sensors as close-to-ground-truth data. We therefore recorded a dataset consisting of pairs of lower- and higher-quality frames of the same scenes. To train the denoising network, we developed a preprocessing pipeline, which aligns the frame pairs and computes the YCB Objects mask. We validate our trained network against the original approach to investigate the application in the context of robotic manipulation. The implementation of our framework can be found on our GitHub.**

I. Introduction[2]

RGB-D sensors capture not only the RGB color information, but also a depth value for each pixel. In the context of robotics this bears the potential of better perception and manipulation in human environments. But just as the prices vary between RGB-D sensors, so does the quality of their depth images. Smaller, more compact sensors such as the Intel RealSense product line rely mostly on infrared for depth measurements. Those sensors emit infrared light and capture the returning light with two infrared cameras. By comparing the frames of the two cameras, depth information of the scene can be reconstructed. Their price tag, size and high frame rate makes them more applicable for the use on robot systems, but consequently also leads to rather poor and noisy depth images. More expensive sensors, such as the ones offered by Zivid, come with better optics leading to a much higher quality of depth images; but on the other hand, they are also bulkier, heavier, and due to their architecture only capable of a low frame rate. The Zivid cameras, for instance, project visible light patterns on the objects to measure depth information, which reduces the frame rate massively. For these reasons, higher-quality sensors are less feasible for the application on robots. Figure 1 displays the qualitative differences between such lower- and higher-quality RGB-D images.

[1]written by Claudius
[2]written by David

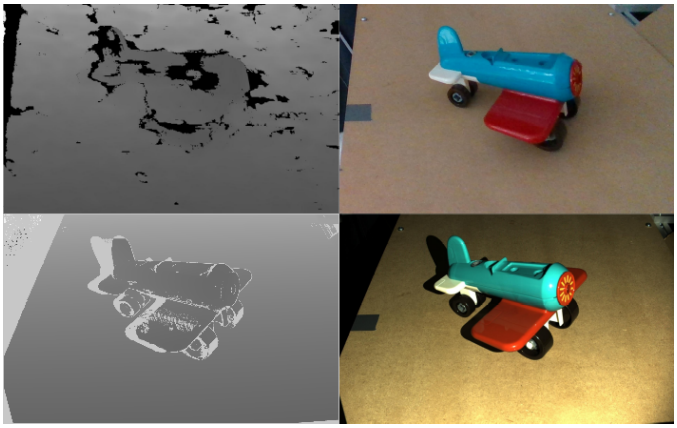


Fig. 1. Depth images (left) and RGB images (right) of RealSense D435 (top) and Zivid One+ (bottom) depict a toy plane. Because of the different fields of view of both sensors top images were cropped, but not transformed, to fit bottom images. The depth image of the RealSense is very noisy, and it is almost impossible to recognize the object contained. On the other hand, the Zivid depth image is very smooth, and the object contained can be identified clearly. Additionally, one can see the influence of the Zivid's depth measuring technique on the RGB image. Surfaces close to the camera appear very bright in the RGB picture and shades can be seen on both images. This is due to the white light pattern the Zivid projects onto the surfaces for depth measurement. RealSense's RGB images on the other hand appear rather dark under the same lightning conditions. The different angles of view on the captured object are also clearly visible. [2]

In a recent paper Shabanov et al. [1] proposed the idea of training a network to learn how to denoise such lower-quality (LQ) RGB-D images by using higher-quality (HQ) images of the same object as ground truth. In their experiment they used the Apple TrueDepth (TD) as lower-quality sensor and Microsoft Kinect V2 (K2) as higher-quality sensor. Both cameras were mounted next to each other and video sequences of RGB-D images of humans were captured. A function was then learned to temporally align the frames of the two sequences. The resulting LQ-HQ frame pairs were then used to learn the extrinsic transformation for every new sequence, to transform HQ onto LQ frames, eliminating different view angles. The transformed frame pairs were lastly used to train a two-level neural network, consisting of a UNet [2] and a ConvLSTM [3] to denoise the LQ images.

In the context of this work the pipeline for self-supervised depth denoising as presented by Shabanov et al. [1] is re-implemented. This work therefore consists into two steps, namely dataset generation and neural network training. The process of dataset generation is explained in Section II. While originally, Shabanov et al. focused on denoising depth frames depicting human bodies, this work tries to apply those techniques on the YCB Dataset [4]. Coming from the field of robotics, we use a different pair of RGB-D sensors, in particular a RealSense D435 (RS) as LQ sensor and a Zivid One+ (Zivid) as HQ sensor, detailed in Section II-A. Due to Zivid's low frame rate, dynamic video capturing is not possible. We additionally reduced the computational overhead introduced by the recalculation of the extrinsic transformation for every new sequence, which is explained in more detail in Section II-B. Section III elaborates the implementation of the network used, where we closely followed the description of Shabanov et al. [1].

II. Dataset Generation[1]

The first part of the pipeline consists of the dataset generation, on which the networks will later be trained on. This first requires the setup of the sensors which we introduce in Section II-A. Section II-B presents the calibration in use, which eliminates the naturally different view angles of the two cameras. After the cameras are calibrated correctly resulting in a pixel-wise correspondence of the RGB-D frames, we recorded the RGB-D dataset on a reduced set of 20 YCB Objects, shown in Figure 2.

Fig. 2. Image of the reduced set of 20 YCB objects used as objects-of-interest in this work. The MaskRCNN first deployed for generating the region masks was trained on this reduced set, which directly compelled us to use the same for our dataset. Additionally, one object of the initially 21 YCB objects was missing, reducing the total amount to 20. [2]

A. Camera Setup[2]

As mentioned in the introductory part, a RealSense D435 (RS) is used as lower-quality (LQ) sensor and a Zivid One+ (Zivid) as higher-quality (HQ) sensor. In the original work of Shabanov et al. [1], both sensors are rather loosely placed on a chessboard as can be seen on the left in Figure 3. On the right side our setup is depicted in comparison. The foundation of our rig is a resin-coated plywood plate, onto which the Zivid is directly fixated by four screws. Next to the Zivid, the RS is mounted onto a small wooden pedestal

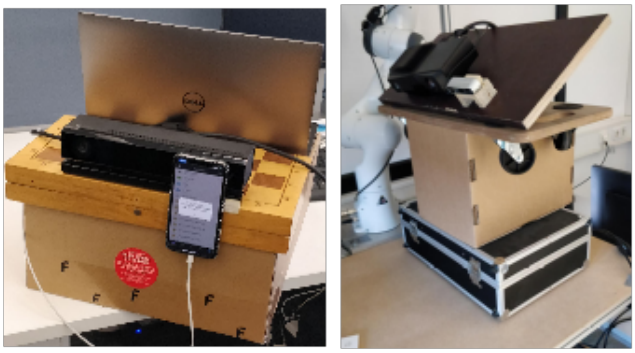


Fig. 3. Camera setup of Shabanov et al. [1] (left) placing a Microsoft Kinect V2 and an Apple iPhone X onto a chessboard. Because the iPhone was manually interacted with over the course of data retrieval, its position varied. Avoiding relative positional changes between the two sensors, a rigid mounting position of the Zivid One+ and the RealSense D435 (right) was enforced by a wooden rig. [2]

using one screw. To ensure the smallest possible deviation when fastening several times, a wooden strip clinging to the RS's back prevents any angular movement. The pedestal itself is fixated onto the plywood plate using four screws. The construction chosen should ensure that both sensors are mounted in the very same relative position across several mounts resulting in consistent data over several acquisitions. In addition, the height and placement of the pedestal were chosen to minimize the difference between the viewing angles of both cameras a priori, placing the sensors of both cameras on a common plane. This includes the angle position of the Zivid One+, as it's viewing angle is shifted by 8.5 degrees counter-clockwise.

For the dataset generation, all 20 YCB objects were individually and compositionally depicted. Every object was randomly moved and rotated to several positions. This ensures a large enough dataset for network training. The raw dataset can be seen as a set of N tuples:

$$\{(\boldsymbol{C}^i_{LQ}, \boldsymbol{D}^i_{LQ}, \boldsymbol{C}^i_{HQ}, \boldsymbol{D}^i_{HQ}) : i = 1, \cdots, N\}$$

where $\boldsymbol{C}^i$ contains the RGB color information and $\boldsymbol{D}^i$ the depth information of the i-th LQ and HQ images respectively. For further processing a set of $N = 1024$ raw image tuples was obtained.

B. Calibration[1]

Due to the fact that both sensors are mounted side by side, the resulting images show the captured objects from a slightly different angle with hugely different fields of view, as shown in Figure 4. To facilitate the learning of the task at hand for the network, an extrinsic transformation must be applied. The extrinsic transformation equals a transform in 3D space that maps an HQ image onto the LQ sensor's plane, resulting in a pixel-wise correspondence of the frames. This was also the approach of Shabanov et al. [1]. But because their two sensors were not mounted in a fixed relative position while dataset generation, they decided to learn a separate transformation for every new sequence.

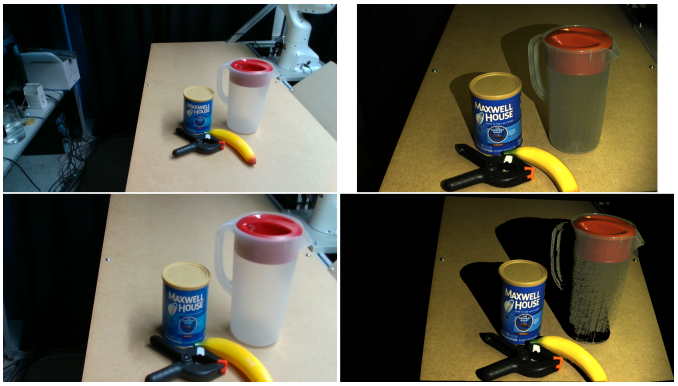


Fig. 4. The upper row displays the raw images of both cameras. The LQ frame is on the left and the HQ frame on the right. It is easy to see that the cameras have different fields of view. The respective calibrated images are visualized below. The rotation and translation that was applied during calibration aligns the objects, but also increases the shadows of the objects in the Zivid frames. [1]

We are leveraging on the fact that our cameras can be mounted in the same relative position repeatedly, which reduces the calibration algorithm massively. Consequently, only a single transformation must be calculated, which avoids the associated computational overhead and also eliminates a possible source of error. Our calibration divides into the three steps: *un-projection*, *transformation*, and *re-projection*.

First, we un-project the color frames $\boldsymbol{C}_{HQ}$, $\boldsymbol{C}_{LQ}$ and depth frames $\boldsymbol{D}_{HQ}$, $\boldsymbol{D}_{LQ}$ of both cameras from image coordinates to camera coordinates with the use of the intrinsic parameters of the respective camera and the formula for the extended pinhole camera model given in Equations (1) to (3).

$$x = \frac{(u - c_x) * z}{f_x} \tag{1}$$

$$y = \frac{(v - c_y) * z}{f_y} \tag{2}$$

$$z = d * d_{scale} \tag{3}$$

Variables u and v are the row and column index of the pixel and d the depth value stored at this pixel in the depth frame $\boldsymbol{D}$. Every point in the point cloud is colored with the RGB value of the color image $\boldsymbol{C}$ at the corresponding pixel. Focal length f_x and f_y, as well as principal point $\begin{bmatrix} c_x & c_y \end{bmatrix}^T$ are fixed, intrinsic, camera specific parameters. This results in point clouds $\boldsymbol{P}_{HQ}$ and $\boldsymbol{P}_{LQ}$, each consisting of points $\boldsymbol{p} = \begin{bmatrix} x & y & z \end{bmatrix}^T \in \mathbb{R}^3$.

$$(\boldsymbol{C}_{LQ}, \boldsymbol{D}_{LQ}), (\boldsymbol{C}_{HQ}, \boldsymbol{D}_{HQ}) \xRightarrow[unproject]{} \boldsymbol{P}_{LQ}, \boldsymbol{P}_{HQ} \tag{4}$$

The next step is to apply an extrinsic transformation on $\boldsymbol{P}_{HQ}$ so that it lines up with $\boldsymbol{P}_{LQ}$. To estimate the correct homogenous matrix that achieves this transformation, K corresponding 3D points $\{(\boldsymbol{p}^i_{HQ}, \boldsymbol{p}^i_{LQ}) \mid i = 1, \ldots, K\}$ in both point clouds must be selected. These points can be selected by hand or with images of a ChArUco board, while the latter results in more accurate transformations. Then, the transformation matrix $\boldsymbol{T}_{ex}$, consisting of a rotation $\boldsymbol{R} \in \mathbb{R}^{3\times3}$ and a translation $\boldsymbol{t} \in \mathbb{R}^3$ can be computed by finding the least-squares solution between the transformed points of the HQ point cloud and the points of the LQ point cloud as depicted in Equation (5) [5].

$$\sum_{k=1}^{K} ||(\boldsymbol{R} * \boldsymbol{p}^i_{HQ} + \boldsymbol{t}) - \boldsymbol{p}^i_{LQ}|| \tag{5}$$

To refine the transformation, we attempted to use ICP [6] to further align the point clouds. Due to the rather large dissimilarity between the point clouds of the HQ and LQ frames, the refined transformation resulted in a worse alignment than with the transformation computed by finding the least-squares. The extrinsic transformation matrix must only be computed once on one point cloud tuple, as the transformation between the two cameras is static and does not change over several recordings.

The extrinsic transformation matrix $\boldsymbol{T}_{ex}$ can then be applied to $\boldsymbol{P}_{HQ}$.

$$\boldsymbol{P}_{HQ} \xRightarrow[Tansf.]{extr.} \widetilde{\boldsymbol{P}}_{HQ} \tag{6}$$

The transformed point cloud $\widetilde{\boldsymbol{P}}_{HQ}$ now matches the position and rotation of the point cloud $\boldsymbol{P}_{LQ}$.

Finally, the transformed point cloud $\widetilde{\boldsymbol{P}}_{HQ}$ can be re-projected back onto the image plane of the LQ camera. The equation to use can be derived by solving Equations (1) to (3) for u, v and d. The color $\widetilde{\boldsymbol{C}}_{HQ}$ and depth information $\widetilde{\boldsymbol{D}}_{HQ}$ of the HQ image can then be retrieved w.r.t. the coordinate system of the LQ image plane.

$$\widetilde{\boldsymbol{P}}_{HQ} \xRightarrow[re-poject]{} \widetilde{\boldsymbol{C}}_{HQ}, \widetilde{\boldsymbol{D}}_{HQ} \tag{7}$$

The re-projection uses only the intrinsic information of the LQ camera, since both point clouds get projected on the LQ's plane.

C. Masking[2]

Shabanov et al. [1] supplied an additional mask as input to the denoising network. This mask is used to inform the network for which areas of the image the denoising should be learned for. In addition, the background and the surroundings of the object of interest are rather noisy and therefore more difficult to learn for the network. Since they trained a network to denoise RGB-D sequences taken with different human actors, we implemented a different, more universal method to generate masks for the training dataset.

The MaskRCNN [7] is at the time of writing the standard for object detection and segmentation. Multiple projects [8], [9] already trained a MaskRCNN on the YCB Objects successfully with the use of the YCB Video Dataset. In it, the reduced amount of 20 YCB objects shown in Figure 2 are deployed [10]. A sample of the YCB Video Dataset can be seen in Figure 5. Because these projects used the by now deprecated Detectron framework [11] for training and also did not publish any snapshot of the trained model, we decided to train the MaskRCNN on the YCB Video Dataset on our own [12]. While the trained model segmented the dataset it was

Fig. 5. Sample of the YCB Video Dataset. A sample image can contain one to multiple YCB objects, while one mask is given for each individual object. The very-left image shows four YCB objects and the four right images show a mask for each of the four objects as an individual ground truth. [2]

trained on really well, the performance on our dataset was significantly worse as shown in Figure 8.

Because of that, we implemented a different, non-deep learning algorithm to mask the calibrated dataset. The whole algorithm computes the mask with the help of both point clouds $\boldsymbol{P}_{HQ}$, $\boldsymbol{P}_{LQ}$, whereby the major part of the algorithm only works on the HQ point cloud. At first, the HQ point cloud gets cropped by a bounding box to a known region where the objects are located in. This removes a lot of the background pixels. To separate the objects from the surface, a normal based region growing is applied that clusters the point cloud into rather smooth surfaces as sown in Figure 6. The region growing therefore has as smoothness constraint the

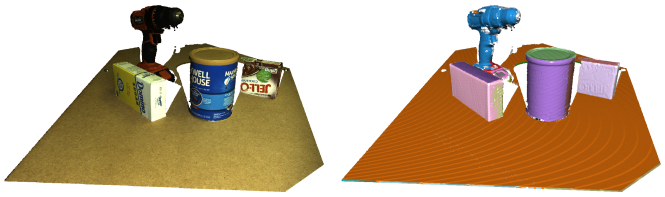


Fig. 6. The image on the right displays the cropped point cloud. On the left, the result of the normal based region growing is visualized. [1]

normal difference to the neighboring point. If the difference surpasses a given threshold, the neighboring point will not be part of the current cluster. The resulting clusters get classified into accepted and rejected clusters based on the distance to certain points that were selected beforehand. The union of the accepted clusters forms the result of this first part of the masking algorithm.

With the result from the normal based region growing, a second density based spatial clustering [13] is started. The goal of this clustering is to remove clusters that are too far away from the largest cluster or smaller than a threshold. Therefore, this clustering mainly removes any outliers that are still present in the point cloud. A clustered point cloud can be seen in Figure 7. This results in a point cloud only containing the objects without any outliers. By re-projecting the HQ point cloud back into color and depth frames, the new NaN values now correspond to the points that got removed. Those points form the mask for the HQ frames. Since the calibrated data does not line up perfectly, we also cluster the masked LQ point cloud using density-based clustering. This results in a second mask respective to the LQ frames. Finally, the resulting mask

Fig. 7. On the left, the result of the previous clustering is depicted. The right shows the clusters computed with the density based clustering. All black points mark outliers, which are points that have a too low density to form a cluster. [1]

equals the intersection of both previous masks. The results of this masking algorithm can be seen in Figure 8.

Fig. 8. The left picture displays the raw, but calibrated HQ point cloud. The middle frame visualizes the mask computed with the MaskRCNN. The mask resulting from the region growing based approach is displayed on the right. The mask generated with MaskRCNN does not detect the objects as precise as the region growing based one, classifying pixels that belong to the background and are relatively near to an object as object. On the other hand, the MaskRCNN mask classifies points of an object, like the plier, as background. The region-based approach classifies the pixels at the mask borders way more accurately. Some object points on the right mask get also classified as background. These pixels have NaN values at in the corresponding depth image. [1]

D. Augmentation[1]

In the paper of Shabanov et al. [1] recorded 51 simultaneous RGB-D sequences of approximately 30 seconds each. The K2 can capture 30 fps at maximum. This consequently leads to a dataset of approximately 46,000 image pairs. In this works context, the upper bound of frame capturing speed is given by the Zivid with a minimum acquisition time of 90 ms at the lowest exposure time of 6.5 ms, resulting in relatively noisy depth information at about 11 fps. This makes our setup rather infeasible for video capturing at the cost of a lower-quality ground truth. We therefore decided to capture frames individually, giving us space to adjust the capturing settings of the Zivid for best possible results. As a result, the dataset acquisition is much more time consuming such that our final dataset consists only of 1024 frame pairs.

To further increase the size of the dataset, we applied data augmentation. Typical data augmentation would falsify our dataset, since the depth values stored in the depth frames would not be augmented. For a rotation, translation or scale, this results in depth values that do not match with the position in the frame. For this reason, we used a different augmentation technique. Instead of augmenting the frames of each data tuple, we applied the augmentation in three-dimensional space. Each tuple $(\boldsymbol{C}_{LQ}, \boldsymbol{D}_{LQ}, \boldsymbol{C}_{HQ}, \boldsymbol{D}_{HQ})$ is un-projected

to point clouds $(\boldsymbol{P}_{LQ}, \boldsymbol{P}_{HQ})$. Then K randomly sampled transformations $\boldsymbol{T}^i_{rand}$ get applied on both of the point clouds. Each transformation consists of a randomly sampled rotation and translation in specified bounds. After re-projection of the newly generated point clouds, this augmentation results in a total of $K+1$ tuples for each data tuple. Figure 9 displays 48

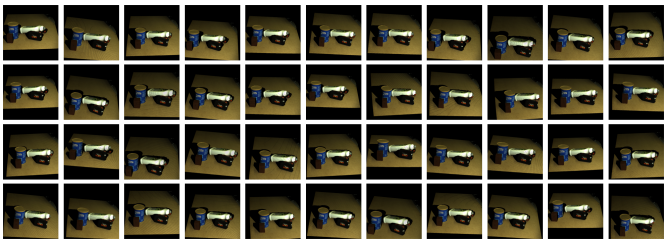


Fig. 9. The picture visualizes 48 augmentations generated on one image. The augmentations consist of a translation of maximal 10 cm and a rotation of maximal 5 degrees. [1]

generated augmentations generated based on one image. The augmentations were generated with a maximal translation of 10 cm and a maximal rotation of 5 degrees.

III. Network & Training

A. Network Architecture[2]

Shabanov et al. [1] used a recurrent model to leverage on the temporal information available by their video based dataset. But they "did not succeed to directly train one due to the limited amount of data" [1]. Therefore, they utilized a two-level training approach based on the out-of-fold (OOF) prediction scheme introduced by Wolpert et al. [14]. The scheme implemented by Shabanov et al. can be seen in Figure 10. For the first level, corresponding with models M^1_1, M^1_2, M^1 of the figure, the researchers used a UNet-like architecture with the goal to depth-denoise on a per-frame basis. They used a UNet-based convolutional LSTM (LSTMUnet) as second-level model with the intent to leverage on temporal correlations of consecutive frames.

With our raw dataset containing 1,024 image pairs, it is significantly smaller than the dataset of Shabanov et al. [1] with approximately 46,000 frame pairs. Additionally, the amount of temporal information held between consecutive frame pairs in our case is significantly smaller, if existent, due to not capturing dynamic image sequences. Splitting our small dataset into three equally sized sets P_1, P_2, P_{test} for the OOF prediction scheme would result in a raw training set size of 340 for each model of the first level. That being a too small number to train any generalizable model, we decided to solely train a single UNet-like network with 90 % of our data corpus, splitting the remaining 10 % equally in validation and test dataset, whereby the test dataset was only used for network evaluation.

The UNet-like architecture used by Shabanov et al. [1] can be seen in Figure 14. It is based on the original architecture proposed by Ronneberger et al. [2], but additionally has skip connections in the downward path. For each downward step,

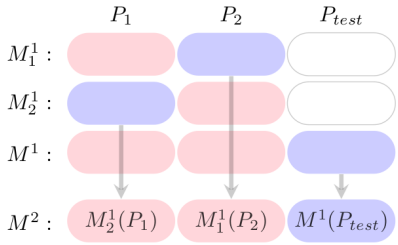


Fig. 10. Out-of-fold (OOF) prediction scheme as used by Shabanov et al. [1]. It splits the dataset into three groups $\{P_1, P_2, P_{test}\}$. Then it learns three models M^1_1, M^1_2, M^1 on the group shown in red a generates their predictions on the blue group. E.g., M^1_1 is trained on P_1 and evaluated on P_2. Then its predictions $M^1_1(P_2)$ and $M^1_2(P_1)$ are used to train a second-level model M_2 using $M^1(P_{test})$ for validation. [2]

the initial network input is repeatedly down-sampled to half of its size with average pooling and then concatenated with the result of the previous double convolution. The amount of output channels in the double convolutional layer pads the channels after concatenation with the skipped input, so that is equals to the according power of two.

For the upward path two options are available. The first one uses simple bilinear upsampling for an upward step, highlighted blue in Figure 14. This upsampling is computed channel-wise and a reduction of the total channel count is not possible. To nevertheless reduce the amount of channels per upward step, the amount of out-channels is split for every double-convolution in the upward path. The second option relies on a 2D transposed convolution with a 2x2 kernel and a stride of 2, highlighted green in Figure 14. The number of out-channels is set to halve the total amount, therefore not requiring the halving in the double convolutional layers as for the first option.

The number of channels for each downward/upward step changes by the power of two. The number of output channels for the first level of the downward path therefore determines the channel dimensions of the following layers. Since our dataset is relatively small, we found an initial channel dimension of 32 to work best. A larger initial channel dimension, like 64, decreased training speed and tends to underfit.

The input of the network consists of the four RGB-D channels of the input image optionally concatenated with the object mask as fifth channel. The network's output on the other hand is a single channel containing the predicted depth at each pixel. Since valid depth values must be greater or equal to zero, we also implemented an optional ReLU output activation that maps all negative depth values to zero.

B. Training[1]

In addition to using the optimizer RMSProp for UNet training, we also deployed a learning rate scheduler to reduce the learning rate when the validation loss reaches a plateau. To speed up training, we enabled automatic mixed precision training (amp) and used network gradient scaling which improves convergence.

We examined multiple options for data preprocessing to improve network convergence and prediction results. For one, the first three input channels encoding the RGB values are scaled linearly to $[0, 1]$. We also tested normalizing the input and target depth channel, but it resulted in a less accurate prediction. To ease network training, the computed object mask was directly applied on the input depth and RGB channels, whereby all pixels outside the object mask get replaced with NaN.

Due to the quality difference of the LQ and HQ depth frames, we also implemented a hyperparameter δ to limit the depth difference for input/target pairs during training. This removes all pixels in the LQ and HQ frames, whose depth difference between the two depth frames surpasses δ-times the mean depth difference. Hence, the depth difference in an input/target pair is limited upwards by the mean depth difference times δ. With a low value for δ, like 1 or 2, the network can focus to learn only small depth differences. A large value for δ, e.g. 3 or more, only removes outliers in the dataset and therefore also eases learning to denoise the important regions. We decided to use the mean as metric, as we did not find any better heuristic for this purpose.

Loss: Shabanov et al. [1] used a standard L1 Loss for network training, whereby pixels outside the mask get replaced with 0, which can also be seen in Equation (8).

$$\mathcal{L}_{orig}(D_{pred}, \tilde{D}_{HQ}) = ||(D_{pred} - \tilde{D}_{HQ}) * M_{nan} * M_{obj}||_1 \quad (8)$$

Although the absolute value of a matrix is not defined, they did not state if they used the mean or the sum as reduction technique to obtain a scalar loss. For this reason, we investigated network training using the most common loss functions, namely the mean L1 Loss, mean L2 Loss, and Huber loss, depicted in Equations (9), (10) and (11).

$$\mathcal{L}_1(D_{pred}, \tilde{D}_{HQ}) = \frac{\sum_{i,j} |d^{ij}|_1}{\sum_{i,j} m^{ij}} \quad (9)$$

$$\mathcal{L}_2(D_{pred}, \tilde{D}_{HQ}) = \frac{\sum_{i,j} |d^{ij}|_2}{\sum_{i,j} m^{ij}} \quad (10)$$

$$\mathcal{L}_h(D_{pred}, \tilde{D}_{HQ}) = \frac{\sum_{i,j} \begin{cases} 0.5 * |d^{ij}|_2 & |d^{ij}|_1 < \delta \\ \delta(|d^{ij}|_1 - 0.5 * \delta) & else \end{cases}}{\sum_{i,j} m^{ij}} \quad (11)$$

$$m^{ij} := m^{ij}_{nan} * m^{ij}_{obj} \quad d^{ij} := (d^{ij}_{pred} - \tilde{d}^{ij}_{HQ}) * m^{ij}$$

Hyperparameter Tuning: The network training requires many hyperparameters to be set. Among them are the initial learning rate, the loss function, how fast the learning rate should decay, the output activation, and many more. To evaluate networks with different hyperparameter settings, we trained 100 models for which the hyperparameters were sampled with random search. For each run, every hyperparameter was sampled uniformly from a set of possible values. For example, the set of possible values for the initial learning rate contains the values 0.1, 0.05, and 0.01. All models were trained for 25 epochs each, whereby the batch size varied depending on the hyperparameter configuration. The lowest batch size used was 8 and the highest 70. All models were trained between 60 minutes to two and a half hours.

IV. Results

A. Hyperparameter Tuning[2]

We evaluated all 100 models on the same test set containing 51 samples that the networks never have seen before. The evaluation consists of multiple metrics which provides information about how well the models denoise the input depth frames. In order to evaluate how much the network denoises the input depth frame with respect to the target depth frame, we computed every metric on the input/target pair as well as on the prediction/target pair.

The first metric that we use for evaluation is the mean L1 Loss also depicted in Equation (9). This metric provides information on how well the model denoises the overall input depth frame, independent on how large the pixel-wise depth differences between input and target were originally. To obtain more insight into which depth differences the model denoises and by how much, we also evaluated the model against three other metrics. Those three metrics are variants of the mean L1 Loss, whereby the loss is not computed over the whole depth pair, but only on the depths, whose depth differences are in the interval $[\delta_{min}, \delta_{max})$. The interval, as well as the depth differences, are given in millimeter. For evaluation, we used three such metrics with intervals $[0mm, 10mm)$, $[10mm, 20mm)$, and $[20mm, inf)$.

The evaluation of the hyperparameter tuning with random search can be seen in Figure 11. The figure depicts the top 5 models of the 100 we trained. Every diagram displays the evaluation on one metric. The color of the box plot that visualizes the metric computed on the input/target pair is displayed lighter than the box plot that displays the metric computed on the prediction/target pair.

It can be seen that all models depicted reduce the overall mean L1 Loss. The median of the left most box plot is reduced from 9.8 mm for the input/target pairs, to 8.4 mm for the prediction/target pairs. Therefore, the predicted depth frame of this model is on average more similar to the target depth frame than the input depth frame. When inspecting the composition of the L1 Loss in more detail, it can interestingly be noticed that all trained models focus to denoise depth values with a depth difference above 10 mm. The best model reduces the mean L1 Loss of input/target for a depth difference between 10 and 20 mm from nearly 14 mm to 9.2 mm in median, which equals a denoising of roughly 35 percent. This implies that the networks mainly denoise relatively large deviations from the target depth.

On the other hand, the models do not succeed to denoise finer deviations, which can be seen on the increasing L1 Loss in the interval of $[0mm, 10mm]$. Even the metric of the best model increases from approximately 4.3 mm to 5.6 mm. After further evaluation of the other models with respect to this metric, none of the 100 trained models during hyperparameter

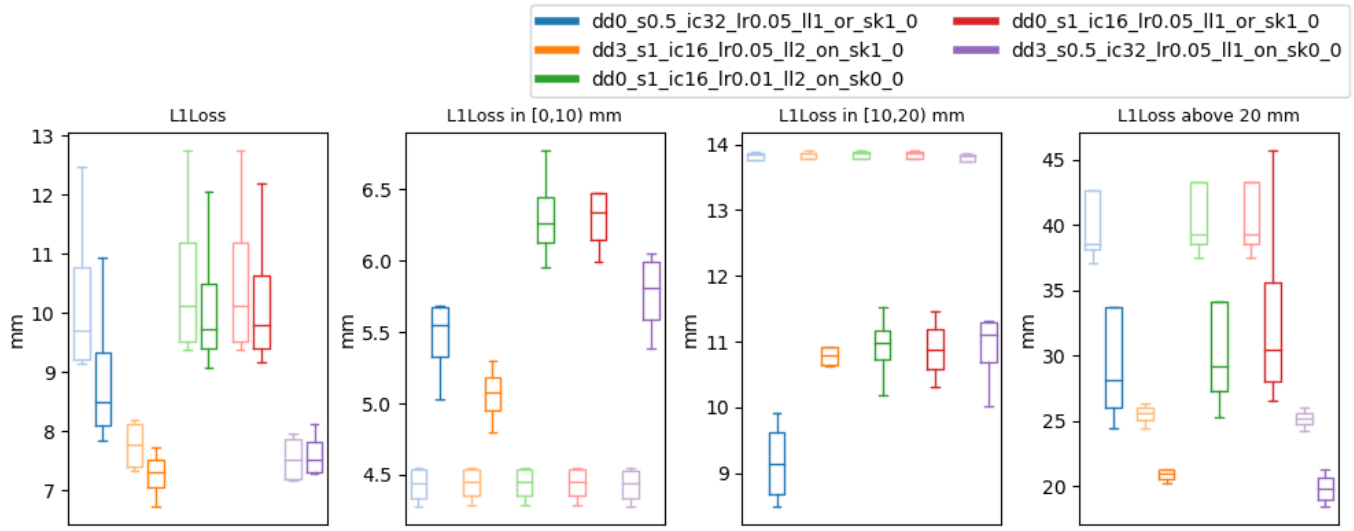


Fig. 11. The figure depicts the evaluation of the hyperparameter tuning with random search. The plot visualizes the prediction/target metrics in the color of the respective model as box plot and on the left of the box plot the respective input/target metric of that evaluation in the same color, but with a lower opacity. The plot displays the top 5 models only, sorted in descending order by how much the median of the prediction/target L1 Loss is lower than the median of the input/target L1 Loss. Therefore, the best model is displayed on the left.

tuning succeeded to denoise those small depth differences between input and target.

The best model resulting from hyperparameter tuning uses skip connections, mean L1 Loss as loss function, ReLU as output activation, 32 initial channels and scales the input images by 0.5.

B. *Refined Evaluation*[1]

After determining the hyperparameters that resulted in the best evaluation, we trained two additional models with those parameters for a longer period, since the models for hyperparameter tuning were only trained for 25 epochs, which equals roughly one and a half hours.

We trained the two models on different datasets, one on the original dataset containing 1,024 samples and the other one on the augmented dataset containing 51,130 samples. The evaluation plots of both models are depicted in Figure 12. The plot also displays the best model from hyperparameter tuning for reference, which is named *hps*. The evaluation indicates that the model *refine hps*, which was refined on the original dataset, denoises the depth difference above 10 mm not significantly better. However, the prediction/target mean L1 Loss for depth differences below 10 mm decreased significantly from 5.6 mm to 4.8 mm. This may be due to the longer training during which the learning rate reduced more than while hyperparameter tuning.

The evaluation of the model that trained on the augmented dataset produces the best results. We trained this model for 19 epochs, which took about 48 hours due to the large dataset. While this model trained on the augmented dataset, we evaluated it also on the non-augmented dataset to ease comparison and ensure consistent evaluation results. The median of all four prediction/target metrics for this model reduced notably

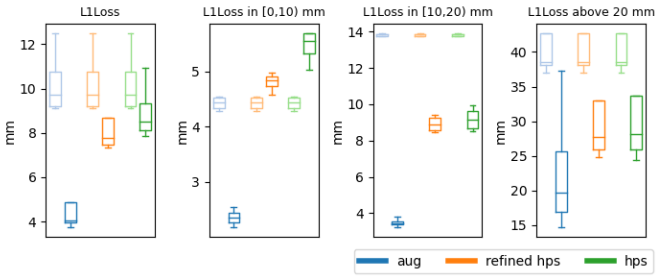


Fig. 12. Model evaluation of three models trained with the same hyperparameters. The *hps* plots display the evaluation of the best model obtained by hyperparameter search. We then refined this model by training it for another 24 epochs, whose evaluation is depicted as *refined hps*. The left most plot *aug* visualizes the evaluation of the model trained from scratch on the augmented dataset.

in comparison to the other two models. Especially the L1 Loss for depth differences below 10 mm reduced from 4.4 mm to 2.4 mm, whereby the other models have not denoised this area at all. An input/target/prediction depth frame tuple is visualized in Figure 13.

It can be seen that the predicted depth frame (bottom left) denoises the input depth frame. While the cereal box's surface in the LQ depth frame (top right) has irregular depth values compared to the HQ depth frame (bottom right), the neural network denoises the frame which results in a smoother surface. This can also be observed on the other surfaces. Besides that, for regions where the observed scene contains jumps in the depth values, for example at the opening of the can, the input depth frame often means over these sharp edges while the target depth frame captures them more precisely. The predicted depth frame also started to learn those sharp jumps,

TABLE I
EVALUATION COMPARISON

	Raw		Results		
	Shabanov	Ours	Shabanov (basic)	Shabanov (LSTM)	Ours
MSE (mm)	57.22	261.17	31.61	21.02	103.74
IT/OT (%)	-	-	55.24	36.76	39.72

Fig. 13. This figure visualizes a sample inferred by the model trained on the augmented dataset. The upper row depicts the LQ RGB and depth frame. In the bottom row, the HQ depth frame is visualized on the right next to the predicted depth frame on the left. All frames are cropped to visualize the objects only.

as can be seen in Figure 15. For this sample, the overall mean L1 Loss reduces from 6.03 mm for input/target pair to 1.44 mm for prediction/target pair.

C. Comparison[2]

The comparison of our approach with the results from Shabanov et al. [1] turns out to be very difficult for multiple reasons.

On the one hand, we used a completely different dataset for network training and evaluation compared to them. While they trained the depth denoising of human bodies, we focused to denoise YCB Objects, which are considerably smaller, have more sharp edges, and are placed closer to the cameras. Our raw dataset therefore has an MSE of 261.17 mm compared to Shabanov et al. [1] with an MSE of 57.22 mm.

On the other hand, it is not clear if they computed the MSE on the whole depth frame or only on the mask. The former would result in a lower MSE, since the denominator also sums over pixels that got projected to 0 in the numerator. Shabanov et al. [1] do not elaborate how they compute the MSE for evaluation.

Therefore, in Table I we included a second metric named "IT/OT" besides the MSE in mm, which relates the MSE of each raw dataset with the respective results. It is computed by dividing the MSE of the results by the raw MSE and can be interpreted as how much noise is still present in the predicted depth frames. A value of 25 % for example indicates that only 25 % of the original noise is remaining.

For our approach, we evaluated the model trained on the augmented dataset since it performed best in comparison to the other models. We computed the MSE metric of our model again on the test set. While the prediction MSE of our approach, with a value of 103.74 mm, is worse than the MSE of Shabanov et al. [1] with 21.02 mm, our raw dataset also has a larger MSE such that a comparison based on this metric is not sufficient. Considering the IT/OT metric, with a value of 39.72 %, our approach is able to denoise the input image so that only 39.72 % of the noise is left in the dataset.

Compared to Shabanov et al. [1], our approach can be placed in the middle of their two approaches. Since we only used the first-level of the two-level approach presented by them, our approach is more similar to the *basic* model. Compared with this approach, our model reduces the noise significantly more with an IT/OT value of 36.76 % compared to the 55.24 % the evaluation of the *basic* model achieved. The better performance of our model may be because the noise reduction between our input depth frames and the target depth frames is larger due to the higher quality camera we used.

V. CONCLUSION[1]

We proposed a framework for data generation and self-supervised training of a network, with the goal of denoising depth frames originating from a lower-quality depth camera, using depth frames of a higher-quality depth camera as close-to-ground-truth data. Our approach is based on the framework proposed by Shabanov et al. [1], but is able to rely on fewer raw data frames, which additionally do not have to be timely correlated. Our pipeline generates input/target RGB-D frame pairs of YCB Objects using simultaneously lower- and higher-quality sensors, which are then spatially aligned and included a mask of the to be denoised YCB Objects. We applied a specially developed augmentation technique to increase the amount of training data, enabling us to rely on shot-by-shot data generation.

Our proposed framework therefore applies the work of Shabanov et al. [1] for the use in robotic manipulation and visual servoing. For this purpose, we adapted multiple steps

of dataset generation and network training to our use case. To name a few, we simplified camera calibration by ensuring a fixed relative position, developed our own masking algorithm to reliably and precisely mask YCB Objects, and adapted the network architecture and training.

A. Discussion[2]

While our method is widely applicable, it also has some drawbacks.

To align the HQ frames to the LQ frames, a rotation in point cloud space is applied, which uncovers areas of the three-dimensional object, for which no pixel values in the two-dimensional image plane exists. These empty regions in the target frames make network training more difficult.

Another weak point of our pipeline is the mask generation. Our ability to generate such clear masks was thanks to the flat underground on which the YCB Objects were placed and the distant background in the image. These characteristics were exploited for our mask generation. The usage of high-quality masks were crucial for the generation of our results. The environment a robot observes in action most probably will not have such characteristics. Therefore, deploying our pipeline in action on a robot may result in poorer results and may require a revised masking tool.

Comparing our denoising task to the task of Shabanov et al. [1], we suggest that the denoising of YCB Objects is more difficult than of human bodies. YCB Objects, especially if multiple are closely grouped together, present many edges and smooth surfaces of different sizes. Human bodies on the other hand, especially as depicted in the paper of Shabanov et al. [1], present rather few edges and large surfaces of overall constant size.

B. Outlook[1]

It could be of interest to integrate our pipeline onto an active robot system for permanent image capturing of the robot's working space. With a large enough data set, the masking of the ROIs can then eventually be omitted, as the network could learn to denoise the whole input depth map. On the other hand, our masking algorithm could be applied to generate a masked YCB dataset without costly human segmentation. A segmentation model, which masks YCB Objects, could then be trained on this dataset. This might result in a more robust and real-time capable masking pipeline.

REFERENCES

[1] A. Shabanov, I. Krotov, N. Chinaev, V. Poletaev, S. Kozlukov, I. Pasechnik, B. Yakupov, A. Sanakoyeu, V. Lebedev, and D. Ulyanov, "Self-supervised depth denoising using lower-and higher-quality rgb-d sensors," in *2020 International Conference on 3D Vision (3DV)*. IEEE, 2020, pp. 743–752.

[2] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241.

[3] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo, "Convolutional lstm network: A machine learning approach for precipitation nowcasting," *Advances in neural information processing systems*, vol. 28, 2015.

[4] B. Calli, A. Walsman, A. Singh, S. Srinivasa, P. Abbeel, and A. M. Dollar, "Benchmarking in manipulation research: The ycb object and model set and benchmarking protocols," *arXiv preprint arXiv:1502.03143*, 2015.

[5] K. S. Arun, T. S. Huang, and S. D. Blostein, "Least-squares fitting of two 3-d point sets," *IEEE Transactions on pattern analysis and machine intelligence*, no. 5, pp. 698–700, 1987.

[6] S. Rusinkiewicz and M. Levoy, "Efficient variants of the icp algorithm," in *Proceedings third international conference on 3-D digital imaging and modeling*. IEEE, 2001, pp. 145–152.

[7] W. Abdulla, "Mask r-cnn for object detection and instance segmentation on keras and tensorflow," https://github.com/matterport/Mask_RCNN, 2017.

[8] Z. Ye, "Use images of the ycb video dataset to train a mask rcnn model." [Online]. Available: https://github.com/iyezhiyu/Mask_RCNN_on_YCB_Video_Dataset

[9] sThalham, "Shelf classification." [Online]. Available: https://github.com/sThalham/shelf_classification

[10] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox, "Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes," *arXiv preprint arXiv:1711.00199*, 2017.

[11] R. Girshick, I. Radosavovic, G. Gkioxari, P. Dollár, and K. He, "Detectron," https://github.com/facebookresearch/detectron, 2018.

[12] P. Kienle, "Mask r-cnn trained on ycb video dataset," https://github.com/alr-internship/Mask_RCNN, 2022.

[13] E. Schubert, J. Sander, M. Ester, H. P. Kriegel, and X. Xu, "Dbscan revisited, revisited: why and how you should (still) use dbscan," *ACM Transactions on Database Systems (TODS)*, vol. 42, no. 3, pp. 1–21, 2017.

[14] D. H. Wolpert, "Stacked generalization," *Neural networks*, vol. 5, no. 2, pp. 241–259, 1992.

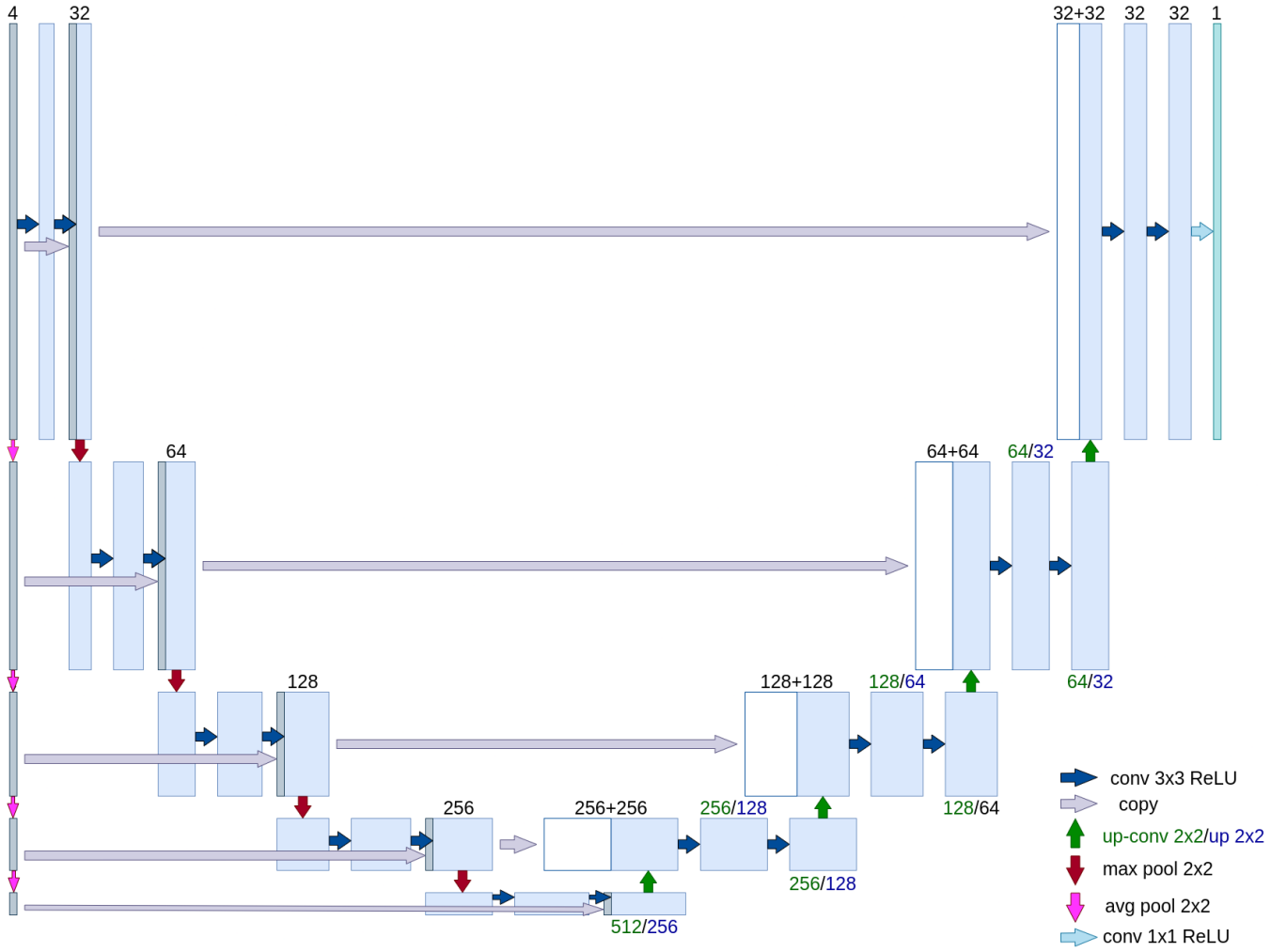
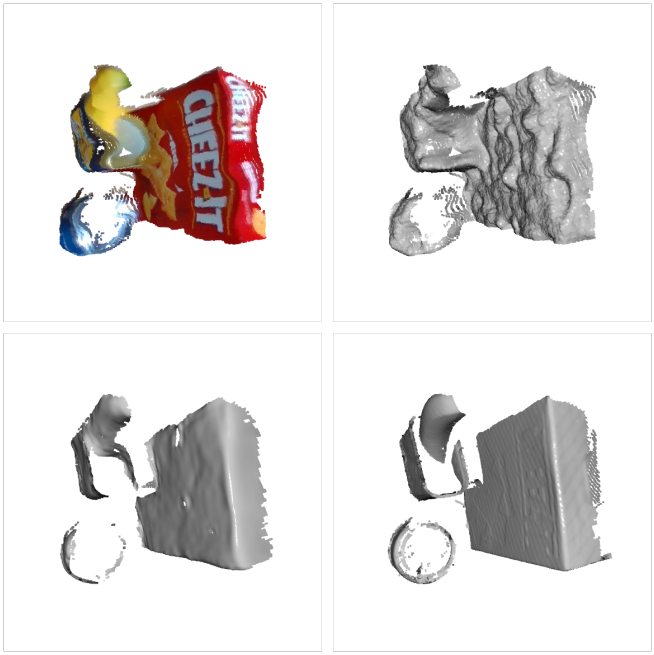


Fig. 14. Visualization of the UNet-like architecture, which was based on the implementation of Shabanov et al. [1]. Additionally to the original UNet architecture of Ronneberger et al. [2], it uses skip connection in the downward path. These skip connections concatenate the results of each double convolution with a downsampled version of the input image. We achieved the best results using 32 initial input channels and bilinear upsampling (denoted in blue). [2]



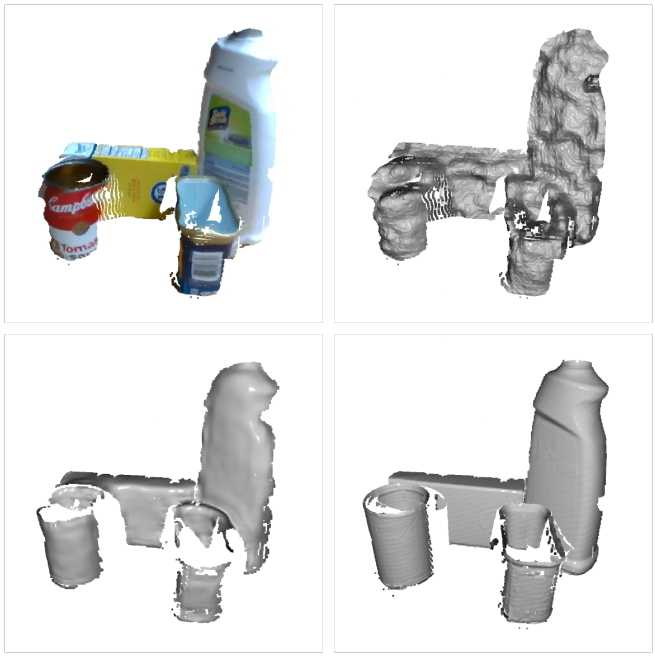


Fig. 15. Top view of an input/prediction/target sample generated with the model trained on the augmented dataset. This visualization displays very clearly, how the network also learned to denoise sharp edges. While the input depth frame (top right) has many pixels at the opening of the can, the predicted depth frame reconstructs the sharp edges which can also be seen in the target depth frame. [1]