Underwater Depth Estimation for Spherical Images

A project of the 2019 Robotics Course of the School of Information Science and Technology (SIST) of ShanghaiTech University
https://robotics.shanghaitech.edu.cn/teaching/robotics2019

Instructor: Prof. Sören Schwertfeger

Jiadi Cui[1] and Lei Jin[2]

***Abstract*— Underwater depth estimation is an open problem in robotics and computer vision. Currently, there still exists many challenges in collecting corresponding ground truth data in the underwater domain. To this end, we propose the leverage of publicly-available in-air RGB-D image pairs for underwater depth estimation in the spherical domain with a unsupervised approach. We are able to recover the depth up-to-scale with no corresponding ground truth data.**

I. Introduction

Underwater depth estimation is an open problem for many marine robotics. Currently, there are no available ground truth depths for underwater spherical images. Still, there exists many challenges to capture RGB-D image pairs in the underwater spherical domain, which makes ground truth depth unavailable. In this report, we propose to leverage publicly-available in-airs spherical images for depth estimation in the underwater domain. Specifically, our approach follows a two-stage pipeline. i) Given in-air RGB-D spherical pairs from Stanford 2D-3D-S dataset [1], we train a style-transfer network [2] to convert in-air images to the underwater domain. ii) Given the generated underwater images and their depth maps, we train a depth estimation network which is specially designed for spherical images. During testing, we can generate depth directly from the input image. Our approach is unsupervised in that only underwater images (*i.e.*, no ground truth underwater depth) are required for the whole training process.

II. Related Works

a) Panoramic Images: One common line of work in panoramic images tackles the lack of dataset. In [3], the author proposed to solve depth estimation and color correction in spherical domains at the same time by solving left-right consistency under a multi-camera setting. Another line of work focuses on the distortion problem. [4] deforms the sampling grid according to the image distortion mode. [5] reprojects the image onto an icosahedral spherical polyhedron.

b) Underwater Color Correction: Color correction can change the images shot by underwater environment to the normal images, and the reversion process can make the normal images into the underwater images. There are many methods to correct the color of underwater images, like deep learning or mathematical methods. In [6], they use Jaffe-McGlamery model[7], [8], a mathematical method, to handle the problems. It considers absorption and scattering effects, which denote $\alpha(\lambda)$ and $\beta(\lambda)$. So the main equations are,

$$\eta(\lambda) = \alpha(\lambda) + \beta(\lambda)$$
$$L(d, \lambda) = Re^{-\eta(\lambda)d}$$

where R is the initial irradiance before propagation through the water column, d is the range from the camera to the scene and L is the final irradiance subject to water column effects. And the coefficients can be optimized by various traditional methods. An ideal method [6] is implemented by the idea for bundle adjustment, which means the errors about color correction could be apportioned into each step.

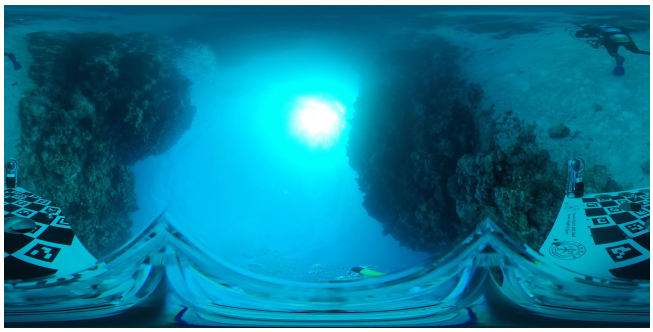

Fig. 1. A typical underwater omni-directional image

c) Underwater Learning: WaterGAN [9] is one the pioneer works in the underwater domain. The author divide the problem of underwater style transfer into three different parts: Attenuation, Back-scattering and Camera Model. The first stage accounts for the attenuation of light. The network is designed to predict the attenuation factor and reconstruct a rough underwater images from an in-air image. In the second stage, to simulate the characteristic haze effect in the underwater images, depth is combined with a random noise vector as input to generate the scattering effects. In the final stage, WaterGAN further models the shading pattern from the camera models into the network. Given the generated underwater images, we want it to look as similar as real underwater images. Another discriminator is appended in the end to tell real or fake underwater images appart. During training, the generator aims at producing photo realistic images so that the discriminator can not tell apart, while the discriminator aims at distinguish them [10]. Our work is

*This work was supported by ShanghaiTech MARS Lab
[1] cuijd@shanghaitech.edu.cn
[2] jinlei@shanghaitech.edu.cn

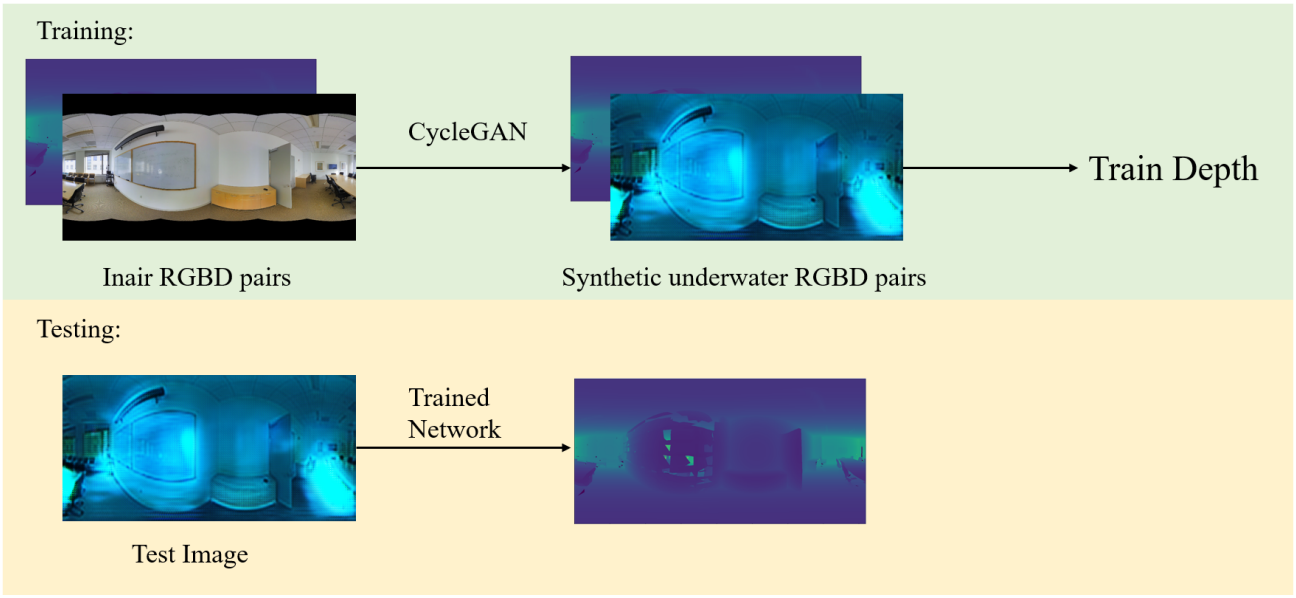

Fig. 2. Full pipeline of our approach. We propose to leverage publicly-available RGB-D datasets for style transfer and depth estimation in an unsupervised approach.

different from WaterGAN [9] in that i) WaterGAN requires depth as input to simulate the attenuation and scattering effect, while we only need underwater and in-air images as input. ii) We aim at depth estimation, while WaterGAN [9] targets at image restoration.

Our work is in spirit most similar to [11]. The author in [11] also proposed a two-stage pipeline to solve underwater omni-directional depth estimation. In the first perspective image pipeline, the author used the WaterGAN [9] to transfer RGBD images to underwater RGBD images. Then, he trained a FCRN [12] depth estimation network with underwater image as input. In the second omni-directional stage, the author synthesised image from in-air equirectangular image to underwater equirectangular image by decreasing the values in red channel (due to its short wavelength nature in underwater environment) and blurring the image based on its distance to the camera origin. Finally, following [4], a distortion-aware convolution module replaced the normal convolution in FCRN based on the spherical longitude-latitude mapping.

III. Methodology

Fig. 2 demonstrates our two-stage pipeline. i) Given in-air RGB-D spherical pairs from Stanford2D-3D-S dataset [1], we train CycleGAN [2] to convert in-air images to the underwater domain. ii) Given the generated underwater images and their depth maps, we train a depth estimation network to learn depth. We will introduce the two parts separately in the following context.

A. Style Transfer

Generative Adversarial Nets (GANs) are designed for data augmentation and now are widely used about the style transfer tasks. GANs are two-player mini-max games between a generative model G and a discriminative model D [13]. The value function about this adversarial process is

$$\begin{aligned}\min_G \max_D V(D,G) =& \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] \\ &+\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1-D(G(\boldsymbol{z})))],\end{aligned}$$

where p_{data} denotes the features in the data, and $p_{\boldsymbol{z}}$ is set by noise variables at first. This value function is also a loss function about deep neural network.

For underwater style transfer, CycleGAN [2] is implemented. Thus, about the mapping function $G : X \rightarrow Y$, the loss function is

$$\begin{aligned}\mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) =& \mathbb{E}_{y\sim p_{\text{data}}(y)}[\log D_Y(y)] \\ &+ \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log(1-D_Y(G(x))].\end{aligned}$$

Moreover, CycleGAN apply a new idea about cycle consistency, which is $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$. And the loss function on this step is

$$\begin{aligned}\mathcal{L}_{\text{cyc}}(G,F) =& \mathbb{E}_{x\sim p_{\text{data}}(x)}[\|F(G(x))-x\|_1] \\ &+ \mathbb{E}_{y\sim p_{\text{data}}(y)}[\|G(F(y))-y\|_1],\end{aligned}$$

so finally, the full objective for CycleGAN is

$$\begin{aligned}\mathcal{L}(G,F,D_X,D_Y) =& \mathcal{L}_{\text{GAN}}(G,D_Y,X,Y) \\ &+ \mathcal{L}_{\text{GAN}}(F,D_X,Y,X) \\ &+ \lambda\mathcal{L}_{\text{cyc}}(G,F)\end{aligned}$$

$$G^*, F^* = \arg\min_{G,F}\max_{D_x,D_Y}\mathcal{L}(G,F,D_X,D_Y)$$

Because the method is pixel-to-pixel, the dataset are pre-processed by cropping and resizing into a reasonable size about the images.

B. Depth Estimation

For depth estimation, we adopt FCRN, one of the state-of-the-art single model on NYUv2 [12]. The network consists a feature extraction model, then several up-convolutions layers to increase the resolution. Finally, we calculate the L1 difference between the output depth and ground truth depth maps.

$$L_{depth} = \sum_{d \in x,y} \|D_{pred} - D_{gt}\|_1$$

For depth estimation in planar images, smoothness regularization has been used frequently in previous research, to encourage estimated depths to be locally similar when no significant image gradient exists. The term is defined as follows:

$$L_{sm} = \sum_{p_t} \sum_{d \in x,y} \|\nabla_d^2 D_t(p_t)\|_1$$

, where L_{sm} is a spatial smoothness term that penalizes the L1 norm of second-order depth gradients along both the x and y directions in 2D space. Here, the number 2 represents the 2nd order.

Further, we also add a surface norm regularize to the network. Given the predicted depth and ground truth depth, we can calculate their surface norm in a local window. Finally, we calculate the cosine distance.

$$L_{normal} = \sum_{d \in x,y} cos(N_{pred} - N_{gt})$$

Our final loss is a weighted combination of the above factors.

C. Implementation Code

We demonstrate some implementation code in this section. [1]

In style transfer, the generative model G and the discriminative model D are designed by these codes.

```
class CycleGANModel(BaseModel):
    def __init__(self, opt):
        self.netG_A = networks.define_G(opt.
    input_nc, opt.output_nc, opt.ngf, opt.netG, opt
    .norm, not opt.no_dropout, opt.init_type, opt.
    init_gain, self.gpu_ids)
        self.netG_B = networks.define_G(opt.
    output_nc, opt.input_nc, opt.ngf, opt.netG, opt
    .norm, not opt.no_dropout, opt.init_type, opt.
    init_gain, self.gpu_ids)

        if self.isTrain:  # define discriminators
            self.netD_A = networks.define_D(opt.
    output_nc, opt.ndf, opt.netD, opt.n_layers_D,
    opt.norm, opt.init_type, opt.init_gain, self.
    gpu_ids)
            self.netD_B = networks.define_D(opt.
    input_nc, opt.ndf, opt.netD, opt.n_layers_D,
    opt.norm, opt.init_type, opt.init_gain, self.
    gpu_ids)
```

[1]All code is put in https://star-center.shanghaitech.edu.cn/gitlab/robotics2019/robotics2019_project_uwdepth/tree/master/code

For the loss function:

```
class CycleGANModel(BaseModel):
    def __init__(self, opt):
        if self.isTrain:
            self.criterionGAN = networks.GANLoss(
    opt.gan_mode).to(self.device)  # define GAN
    loss.
            self.criterionCycle = torch.nn.L1Loss()
            self.criterionIdt = torch.nn.L1Loss()

            self.optimizer_G = torch.optim.Adam(
    itertools.chain(self.netG_A.parameters(), self.
    netG_B.parameters()), lr=opt.lr, betas=(opt.
    beta1, 0.999))
            self.optimizer_D = torch.optim.Adam(
    itertools.chain(self.netD_A.parameters(), self.
    netD_B.parameters()), lr=opt.lr, betas=(opt.
    beta1, 0.999))
            self.optimizers.append(self.optimizer_G
    )
            self.optimizers.append(self.optimizer_D
    )
```

And more specifically,

```
class GANLoss(nn.Module):
    def __init__(self, gan_mode, target_real_label
    =1.0, target_fake_label=0.0):
        super(GANLoss, self).__init__()
        self.register_buffer('real_label', torch.
    tensor(target_real_label))
        self.register_buffer('fake_label', torch.
    tensor(target_fake_label))
        self.gan_mode = gan_mode
        if gan_mode == 'lsgan':
            self.loss = nn.MSELoss()
        elif gan_mode == 'vanilla':
            self.loss = nn.BCEWithLogitsLoss()
        elif gan_mode in ['wgangp']:
            self.loss = None
        else:
            raise NotImplementedError('gan mode %s
    not implemented' % gan_mode)
```

In depth estimation, the L1 loss is implemented as follows. Note that we need to get rid of the invalid values.

```
class MaskedL1Loss(nn.Module):
    def __init__(self):
        super(MaskedL1Loss, self).__init__()

    def forward(self, pred, target):
        valid_mask = (target > 0).detach()
        diff = target - pred
        diff = diff[valid_mask]
        return diff.abs().mean()
```

For the smoothness term:

```
def get_smooth_loss(depth, img):
    grad_disp_x = torch.abs(depth[:, :, :, :-1] -
    depth[:, :, :, 1:])
    grad_disp_y = torch.abs(depth[:, :, :-1, :] -
    depth[:, :, 1:, :])

    grad_disp_x *= torch.exp(-grad_disp_x)
    grad_disp_y *= torch.exp(-grad_disp_X)

    return grad_disp_x.mean() + grad_disp_y.mean()
```

And for the normal term:

```
depth_normal = torch.cat((-depth_grad_dx, -
    depth_grad_dy, ones), 1)
output_normal = torch.cat((-output_grad_dx, -
    output_grad_dy, ones), 1)
```

Methods	RMS[m] ↓	Rel[m] ↓	log10 ↓	$\delta < 1.25$ ↑	$\delta < 1.25^2$ ↑	$\delta < 1.25^3$ ↑
FCRN [12]	0.556	0.152	0.060	0.812	0.938	0.978
+UNet	0.545	0.146	0.057	0.826	0.946	0.980
+L_{grad}	0.530	0.142	0.055	0.830	0.946	0.980
+L_{norm}	0.528	0.140	0.055	0.831	0.947	0.980

TABLE I

PERFORMANCE COMPARISON ON THE STANFORD 2D-3D-S DATASET.

```
loss_normal = torch.abs(1 - cos(output_normal,
    depth_normal)).mean()
```

D. Reproduce the Project

In the $code/style\_transfer$ folder, the dataset should be stored into a folder, contain four sub-folders (named trainA, trainB, testA, and testB). To train CycleGAN with the underwater and in-air images, run

```
python3 train.py  --dataroot [your_data_folder] --
    name [your_result_folder] --model cycle_gan
```

In the $code/style\_transfer/checkpoints$ folder, we can find the model files. Then choose one to be put into the $code/style\_transfer/scripts/checkpoints/[your\_model\_folder]$.

For testing and generating the data after processing, run

```
python3 test.py  --dataroot [your_data_folder] --
    name [your_model_folder] --model test --
    no_dropout
```

In the $code/depth$ folder, to train the depth network with the generated images and depth, run

```
python3 main.py
```

This will train the network and report the performance on the converted underwater Stanford 2D-3D-S dataset.

IV. EVALUATION AND RESULT

a) Dataset: Stanford 2D-3D-S [1] is one of the standard benchmarks for in-air dataset. The dataset provides omni-directional RGB images and corresponding depth information, which are necessary data for underwater depth estimation tasks. Furthermore, it also provides semantics in 2D and 3D, 3D mesh and surface normal.

b) Hyper-parameters: We implement our solutions under the PyTorch framework and train our network with the following hyper-parameters settings during pretraining: mini-batch size (8), learning rate (1e-2), momentum (0.9), weight decay (0.0005), and number of epochs (50). We gradually reduce the learning rate by 0.1 every 10 epochs. Finally, we tune the whole network with learning rate (1e-4) for another 20 epochs.

c) Metrics: Following [12], we use the following metrics for the comparisons on the datasets mentioned above: Root mean square error (RMS) $\sqrt{\frac{1}{T}\sum_p (g_p - z_p)^2}$, Mean relative error (Rel) $\frac{1}{T}\sum_p \frac{\|g_p - z_p\|}{g_p}$, Mean $\log 10$ error ($\log 10$) $\frac{1}{T}\sum_p \|\log_{10} g_p - \log_{10} z_p\|$ and pixel accuracy as the percentage of pixels with $\max(z_i/z_i^{gt}, z_i^{gt}/z_i) < \delta$ for $\delta \in [1.25, 1.25^2, 1.25^3]$. g_p and z_p represent the ground truths and the depth map predictions, respectively.

d) Results: Since no ground truth depth is available for in the underwater domain, we report the results on the converted Stanford 2D-3D-S dataset. Numbers are reported in I. Some quality results of our generated underwater dataset are demonstrated in Fig. 3. In Fig. 4, we show some results of our depth estimation network. Although we are able to generate realistic images in the underwater domain, and achieve a good results on the underwater Stanford 2D-3D-S dataset, the result of the depth from the underwater images still have room for improvement.

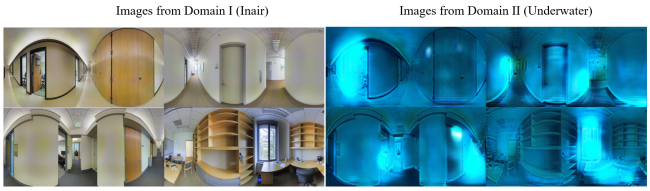

Fig. 3. Generated images with our CycleGAN. On the left are examples from Domain I, inair. On the right are our generated images. We are able to produce the lightening the color effects from the original underwater dataset.

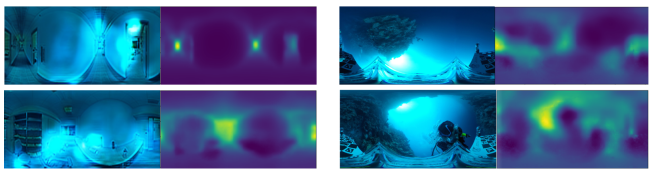

Fig. 4. Generated depth from two datasets. On the left are the input images from the underwater Stanford 2D-3D-S dataset and their predicted depth maps. On the right are images and depth from our underwater datasets.

V. CONCLUSION

In this project, we aim at unsupervised depth learning for the underwater spherical images. However, this is still specially designed for a certain underwater situation. In the future, we are planning working on a unified approach that can work in all kinds of different underwater situations. Collecting a in-air dataset that looks closer to the underwater images might also further improve our performance.

REFERENCES

[1] I. Armeni, S. Sax, A. R. Zamir, and S. Savarese, "Joint 2d-3d-semantic data for indoor scene understanding," *arXiv preprint arXiv:1702.01105*, 2017.

[2] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Computer Vision (ICCV), 2017 IEEE International Conference on*, 2017.

[3] K. A. Skinner, J. Zhang, E. A. Olson, and M. Johnson-Roberson, "Uwstereonet: Unsupervised learning for depth estimation and color correction of underwater stereo imagery," in *2019 International Conference on Robotics and Automation (ICRA)*, May 2019, pp. 7947–7954.

[4] K. Tateno, N. Navab, and F. Tombari, "Distortion-aware convolutional filters for dense prediction in panoramic images," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 707–722.

[5] Y. Lee, J. Jeong, J. Yun, W. Cho, and K.-J. Yoon, "Spherephd: Applying cnns on a spherical polyhedron representation of 360deg images," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 9181–9189.

[6] K. A. Skinner, E. Iscar, and M. Johnson-Roberson, "Automatic color correction for 3d reconstruction of underwater scenes," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 5140–5147.

[7] J. S. Jaffe, "Computer modeling and the design of optimal underwater imaging systems," *IEEE Journal of Oceanic Engineering*, vol. 15, no. 2, pp. 101–111, 1990.

[8] B. McGlamery, "Computer analysis and simulation of underwater camera system performance," *SIO ref*, vol. 75, p. 2, 1975.

[9] J. Li, K. A. Skinner, R. Eustice, and M. Johnson-Roberson, "Watergan: Unsupervised generative network to enable real-time color correction of monocular underwater images," *IEEE Robotics and Automation Letters (RA-L)*, 2017, accepted.

[10] I. Goodfellow, "Nips 2016 tutorial: Generative adversarial networks," *arXiv preprint arXiv:1701.00160*, 2016.

[11] H. Kuang, Q. Xu, and S. Schwertfeger, "Depth estimation on underwater omni-directional images using a deep neural network," 2019.

[12] I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari, and N. Navab, "Deeper depth prediction with fully convolutional residual networks," in *3D Vision (3DV), 2016 Fourth International Conference on*. IEEE, 2016, pp. 239–248.

[13] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.