DEEP-MAPNETS : A RESIDUAL NETWORK FOR 3D ENVIRONMENT REPRESENTATION

Manohar Kuse, Sunil Prasad Jaiswal, Shaojie Shen*

The Hong Kong University of Science and Technology (HKUST)

ABSTRACT

The ability to localize in the co-ordinate system of a 3D model presents an opportunity for safe trajectory planning. While SLAM-based approaches provide estimates of incremental poses with respect to the first camera frame, they do not provide global localization. With the availability of mobile GPUs like the Nvidia TX1 etc., our method provides a novel, elegant and high performance visual method for model based robot localization.

We propose a method to learn an environment representation with deep residual nets for localization in a known 3D model representing a real-world area of 25,000 sq. meters. We use the power of modern GPUs and game engines for rendering training images mimicking a downward looking high flying drone using a photo-realistic 3D model. We use these images to drive the learning loop of a 50-layer deep neural network to learn camera positions. We next propose to do data augmentation to accelerate training and to make our trained model robust for cross domain generalization, which has been verified with experiments. We test our trained model with synthetically generated data as well as real data captured from a downward looking drone. It takes about 25 miliseconds of GPU processing to predict camera pose. Unlike previous methods, the proposed method does not do rendering at test time and does independent prediction from a learned environment representation.

Index Terms— 3D model-based localization, residual network learning, rendered training data, pose regression.

1. INTRODUCTION

Availability of active sensors has propelled research and development in robot motion planning and navigation [1]. At the core, availability of a 3D point cloud of the current environment and the robot's current position at sufficiently high frame rates are required for safe planning and control of mobile robots. Although, active sensors like laser range finders and LiDAR are able to provide real-time 3D point clouds of the environment, the strict pay-load constraints on platforms like quad-rotors, difficulty of frequent camera calibration for stereo camera system and non usability of RGB-D sensors in bright outdoor environments makes a monocular camera with an inertial measurement unit (IMU) an attractive sensor suite.

There have been recent efforts towards online camera-based 3D reconstruction [2, 3, 4, 5]. However, though these methods provide reasonable structure of the environment, their major drawback is that they have a very high computational burden. Another issue is inaccurately mapped points, which make motion planning unsafe. On the other hand, offline approaches to 3D reconstruction are a more mature technology. Numerous commercial and open source tools are available, including Altizure[1], GoogleMaps, insight3d[2] etc. 3D models produced with offline processing (as opposed to online) are more suitable for the proposed render-and-learn approach to learning environment representation.

In recent years advances in deep learning [6] have been an enabler for learning representations for tasks ranging from image categorization to language modeling, speech analysis and so on [7]. These developments coupled with availability of faster and faster GPUs have motivated us to develop a method to learn an environment representation for localization. This paper presents a method for 3D-model based global localization from monocular views. We refer to the proposed method as *MapNets*.

Related Work: Kendall *et al.* [8] introduced a re-localization system based on a convolutional neural network to regress the camera pose. However, their method did not take advantage of the 3D structural information from the reconstruction and had limited data to train the neural network.

Qiu *et al.* [9] proposed a method in which an edge alignment of the captured monocular view is done with a view rendered from a 3D model on the go. The major drawback of this method is that it needs to render a view and its depth at test time, which has a high computational burden. It also needs a good initialization, which was set with GPS in their experiments. Thus their method cannot work on kidnap like scenario. Yet another related work is Ok *et al.* [10], which the authors attempt to align the captured view in the 3D model. Since they rely on an RGB-D sensor, this method cannot be used outdoors. Further as illustrated in Fig 1. of [9], the method by Ok *et al.* [10] may not be able to estimate relative poses with difference between virtual rendered view and real view.

Contributions: In contrast to previous methods, in our proposed simple and efficient approach, we do not need to store the 3D model at test time. Instead our learned model is enough to predict the pose given the input captured image. Our main contributions are summarized as below:

- We propose a deep residual learning framework to model environment representation.
- To generate training data we propose to do monte-carlo sampling of camera-pose (spatial + yaw, 4-DOF) and render views from a 3D model.
- Next to make the model and to help it train faster, we used census transform [11] as augmented data to model local texture and to aid training and cross domain generalization.

2. PROPOSED FRAMEWORK : MAPNETS

In this section we briefly describe the components of our system. In particular, we propose to use rendered views for training, as described in Sect. 2.1. Next we describe the network architecture in

*Email : mpkuse@connect.ust.hk

[1] https://www.altizure.com

[2] http://insight3d.sourceforge.net

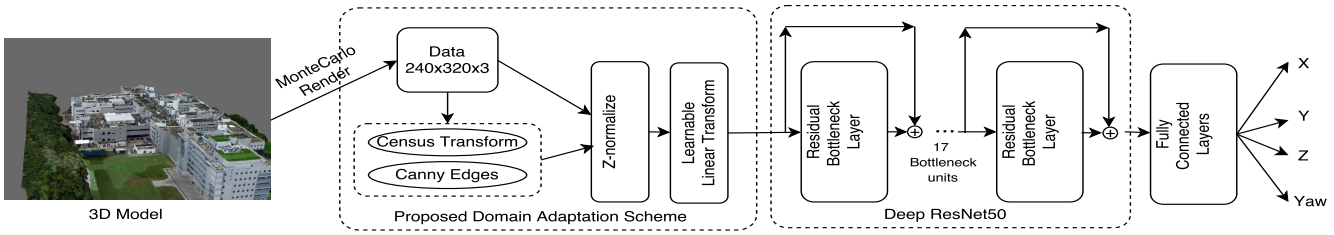


Fig. 1. The structure of our proposed MapNets to learn the environment representation.

Sect. 2.2. Details of the optimization process and our proposed feature augmentation is covered in Sect. 2.2.1 and Sect. 2.3 respectively. For a quick overview of our system, refer to Fig. 1.

2.1. View Rendering

Modern game engines are able to provide upward of 100 fps (320x240 images) rendering. Inside a game engine, the world is represented using polygonal meshes. This provides a good source of training data (as illustrated in Fig. 1) for powerful and data-hungry deep neural networks to learn a model of the environment to predict camera pose. The 3D model used in our experiments was created with the Altizure system using drone video footage of about 20 minutes. We used the panda3d[3] game engine for rendering.

We propose Monte-Carlo sampling of camera-pose (spatial + yaw, 4-DOF) and render views at these locations. These rendered images are fed to a deep neural network along with the camera pose at render time as labels for learning the representation. Essentially, the feed drives a stochastic gradient descent process to optimize a cost function, which is detailed in Sect. 2.2.1.

2.2. Network Architecture

In this work, we propose a framework which outputs a 4-DOF prediction $\boldsymbol{\kappa}$ (spatial + yaw) inferring camera pose from a downward looking drone-captured scene. We refer to our proposed framework as *MapNets*. The logic behind predicting 4-DOF as against 6-DOF is that, 4-DOF provides for a minimal dimensional prediction space, which cannot be predicted drift free from just the IMU measurements.

Furthermore, for a practical system, the pitch and roll angles (usually small for a downward looking drone) are more easily and accurately estimated using IMUs. We also note that since we use only a monocular view at test time, it is possible to predict Z accurately only as long as the difference between the rendering camera's field of view (FOV) and the real camera's FOV is small. In the case that the testing camera's FOV is different from the rendering camera's FOV used for learning, image warping/ cropping may be required to compensate for the difference of FOV.

Let the function f ($f_{\boldsymbol{\theta}} : \Re^{c\times n\times m} \mapsto \Re^4$) be the environment representation. Where n and m are the input image dimensions, and c is the number of input channels. f is parameterized by $\boldsymbol{\theta} \in \Re^p$. The input of our net is the intensity image and the augmented channels. For a discussion of the proposed feature augmentation for domain adaptation, refer to Sect. 2.3. Let the input image be denoted by $\mathbf{x}$, the augmented channels be denoted by $\tilde{\mathbf{x}}$ and $\boldsymbol{\kappa} \in \Re^4$ be the output of the neural function. Thus, $\boldsymbol{\kappa} = f_\theta(\mathbf{x}, \tilde{\mathbf{x}})$

Recently proposed residual nets (ResNets) provide good generalization properties and a high capacity to learn complicated mappings. Residual networks [12, 13] are neural networks that consist of trainable convolutional layers added to the input via the shortcut connection. The non-linear linear unit (ReLU) is used as the activation function. We make use of the bottleneck ResNet units, which have the advantage of having fewer parameters compared to 2-layer 3x3 kernel convolutional units. Each of our ResNet units has 3 convolutional layers, and we have 17 such units (51 layers in all). We perform a max-pooling after every 3 ResNet units while doubling the number of output filters after every max-pooling. After the last layer, the residual net produces a 2048-dimensional vector. This is fed into 2-layers fully connected network to produce a 4-DOF pose.

2.2.1. Optimization

Let $\hat{\boldsymbol{\kappa}}$ be the ground truth pose vector. Note that, this is the same as the camera pose used to render current view. Our objective is to search for a neural function which can predict $\hat{\boldsymbol{\kappa}}$ given the input images. $\mathbf{W}$ is a diagonal weighting matrix to control the relative importance of fitting the spatial predictions, viz. X, Y and Z versus fitting the yaw angle. We empirically set $\mathbf{W}$ as $diag(1, 1, 1, 0.1)$. which means, a fitting loss of about 1 meter spatially is as bad as a fitting loss of 10 degrees for the yaw angle. We thus make use of the regression loss functions as

$$\text{loss} = \sum_{i=0}^{b} \|\mathbf{W}(\mathbf{f}_\theta(\mathbf{x_i}, \tilde{\mathbf{x_i}}) - \hat{\boldsymbol{\kappa}})\|_2 + \lambda \|\boldsymbol{\theta}\|_2 . \quad (1)$$

In order to minimize the neural function, Eq. 1 with respect to parameters $\boldsymbol{\theta}$, we use stochastic gradient descent (SGD) with momentum.[4] We add a parameter norm penalty only for the weight terms as a regularization term in our cost function (Eq. 1), $\lambda = 0.006$. SGD draws a batch, of size b, of random samples of the training space to compute the approximate gradient using back propagation. It then updates the parameters in a negative direction of the gradients scaled by step size [7]. In our experiments we use a batch size of $b = 16$, which fills up about 11 GB of the GPU memory. For SGD, we initialize the training parameters $\boldsymbol{\theta}$ as outlined in He *et al.* [14]. Recently introduced batch normalization [15] accelerates the training by reducing the internal covariate shift by making normalization from batch statistics. We make use of batch normalization at every convolutional layer to accelerate the training.

To further aid the optimization process, make it robust and avoid overfitting, we perform random intensity alterations of the input image and add noise (Gaussian and s&p). Further, we also do training with slight perturbation ($\pm 5^o$) along the pitch and roll axis. Previously in image categorization tasks it has been observed that such perturbation helps the model to generalize better on the test set [16].

[3] https://www.panda3d.org/

[4] We use caffe (http://caffe.berkeleyvision.org/) to perform training.

2.3. Feature Augmentation for Domain Adaptation

Cross domain generalization is a major challenge in learning-based approaches. Motivated by recent success of frustratingly easy Domain Adaptation (DA) [17] approaches by Sun *et al.* [18], (originally proposed by Daume *et al.* [19]), we make an attempt towards domain adaptation by feature augmentation. This approach has been experimentally found to be effective.

As a first step, we propose to z-normalize (subtract mean and divide standard deviation) the incoming data and to use a linear scaling of the input data for effective training. This was realized using the Batch-Normalization frame work, ie. the 1^{st} BN-layer reduces just to the learnable linear transformation scaling as the input data is already z-normalized. Faster convergence was experimentally observed using this approach, which is illustrated in Fig. 1.

Next, given an input 3-channel image, we propose to augment the input space by a representation which captures structural and textural information. For a representation of local texture we propose to use the census transform [11] and we use the canny edges to represent the structural information on the image with adaptive thresholds. The intuition for choosing these transforms is their invariance to changes in illumination, lighting and shading of the observed environment.

We note that although the augmentation of edges might be redundant, convolutional networks cannot learn a transform similar to the census transform, which is a non-linear combination of neighboring pixels. We experimentally demonstrate that augmenting the features aids the learning process and gives robust estimation performance.

Intuitively speaking, this augmented structure learns from a combination of 3 hypotheses and guides the optimization process. This also tends to fit the census transform and the edges, which are relatively invariant towards changes in capturing conditions. It is worth nothing that although the network is able to learn with just the intensity image (1-channel only) it does not seem to generalize well over video captured during different times/exposure. However a network trained with just the census transform image or the edges, does not learn at all (training error does not go down), possibly due to insufficient information in the census transform and edges alone.

Finally, at test time, we load the learned weights and the global estimates of means and variances at every layer along with scaling parameters used for batch normalization. The input image is z-normalized, followed by augmented channels computation from 1-channel representation of the input image.

3. EXPERIMENTS

Implementation Details: For the computations, our workstation consists of Intel i7-6800K CPU @ 3.4 GHz and an Nvidia Titan X (Pascal) with 12 GB of GPU memory. The mesh model used for training consists of 704586 vertices and 1381021 faces (triangles) representing a real world area of about 25,000 square meters.

Without data augmentation, the training is done until there was no change in training loss function (See Fig. 3, ie. about 300k iterations). In case of training with data augmentation, training was performed for 220k iterations. The learning rate was dropped to half each time the average training loss (over 1000 iterations) drops below a threshold. It took about 300 mili-seconds per iteration with batch size of 16. We observe that the loss functions drops faster in case of data augmentation, which shows the effectiveness of the technique.

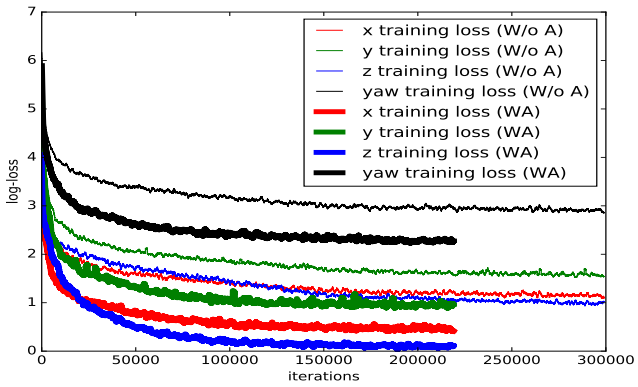


Fig. 2. Showing log-loss as iteration progress. Thick lines denote training with proposed augmentation (WA). Thin lines correspond to training without augmentation (W/o A).

For testing, our flight platform was DJI-Matrice-100 and we use the DJI-OnBoard SDK for logging the camera and flight data (GPS). We also note that at the test time, about 400 MB of GPU memory is required for prediction.

Evaluation from Synthetic Trajectories: We manually set a few way-points and construct a spline passing through these points and render the views along this trajectory, thereby simulating a downward looking drone. Fig. 3 shows the results and compare the prediction with ground truth (render positions). See Table 1 for quantitative results.

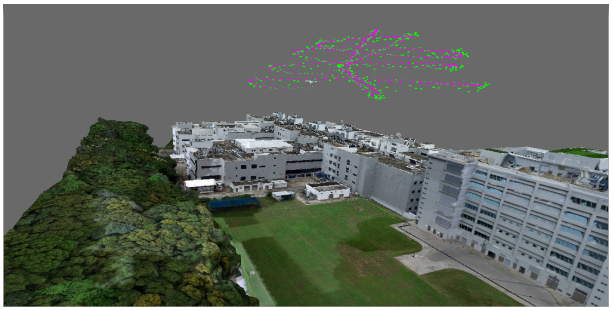
Evaluation from Real Captured Scenes: We also test our trained model using data captured from a drone with a stabilized downward facing camera. The testing is done offline on the desktop computer and takes about 25 ms per frame. However, in a practical system, with availability of embedded GPUs, this can also be done onboard with slight performance penalty. The result is shown in Fig. 4 (green tracks) along with GPS tracks (in pink).

Although the X and Y are observed with high accuracy, jitter of a few meters is observed for the prediction of Z. This is to be expected as the video was captured monocular and at high altitudes (about 65 m).

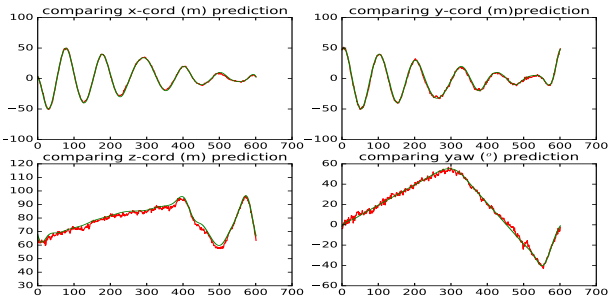
Another issue is with the differences in the FOV of the render camera (used for training) and the real camera. Although care is taken to set the FOV of the virtual render camera as closely as possible to real camera (about 80^o), it is natural to have some uncertainty, which might also be another cause of inaccuracies in the depth (Z) prediction.

Quantitative Result/Effectiveness of Feature Augmentation: Table 1 shows the root mean squared error (RMS) for each trajectory. It is computed using corresponding ground truth (virtual camera positions for synthetic trajectories and GPS tracks for real scenes), ie. the RMS value. We tabulate the performance with the proposed feature augmentation and without feature augmentation. We observe the lower RMS values for model trained with data augmentation compared to the model trained without data augmentation, hence proving its effectiveness.

Further, GPS was used to compare these trajectory which itself suffers from inaccuracies of a few meters. Also measurements of yaw cannot be observed with GPS and hence cannot be compared in this case. Furthermore, the 3D model's co-ordinate system is only approximately aligned to the GPS co-ordinate system and there is variation in GPS axes by a few degrees for each run. All these fac-

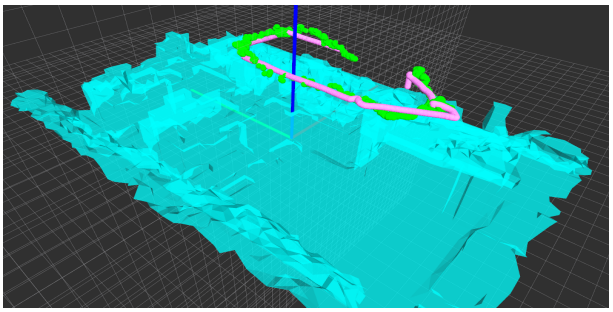


(a) Comparing GT and prediction

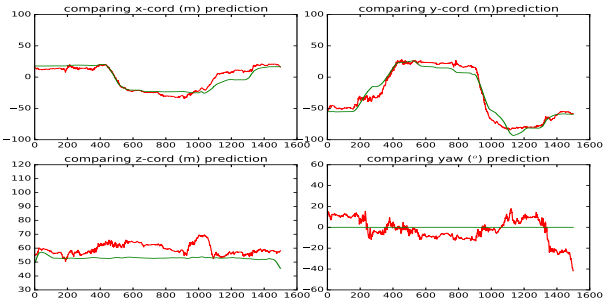


(b) Plot of XYZ and yaw vs time

Fig. 3. (a) Synthetically generated helical trajectory ie. ground truth GT (green) and corresponding predictions using the trained Resnet (pink) overlay on the 3D model. (b) plot X(top left), Y(top right), Z(bottom left) and yaw(bottom right) vs time. Best viewed in color.



(a) Comparing GT and Predicted



(b) Plot of XYZ and Yaw vs time

Fig. 4. (a) GPS track of the drone (pink) and the predictions made using the trained model (green). The 3D model is plotted for reference only. (b) plots X(top left), Y(top right), Z(bottom left) and Yaw(bottom right) vs time. Best viewed in color.

tors make the RMS value comparison for real trajectories with GPS meaningless. We still report the RMS values for reference in Table 1.

	Without Augmentation				With Augmentation			
File	X	Y	Z	Yaw	X	Y	Z	Yaw
Bag11*	6.68	11.46	7.95	N/A	6.85	8.13	6.52	N/A
Bag10*	10.94	13.11	31.30	N/A	8.10	12.56	31.23	N/A
Bag8*	6.517	21.03	6.74	N/A	12.94	25.5	5.28	N/A
Helix	1.16	1.72	1.44	2.55	0.88	1.33	1.11	1.71
BigM	2.13	2.39	2.02	2.70	1.28	1.57	1.44	1.99
yaw_only	1.04	1.31	1.01	2.54	0.56	1.15	0.92	1.23
flat_h	1.08	1.45	1.50	2.65	0.76	1.24	1.11	1.75

Table 1. RMS values between predictions with trained networks and ground truth. '*' marked denote real sequences, whose comparison is made with GPS tracks

4. CONCLUSION AND FUTURE WORK

We have proposed a 50-layer residual convolutional neural network to learn a 4-DOF pose regression. We generate the training images using a 3D model and a game engine. Experimentally, we have found that the proposed data augmentation used to model local texture is effective in the sense that it helps the network to learn faster and gives robust performance during test time. Our method, provides a simple, parameter-free and high-performance method to predict camera pose for a high flying drone.

While SLAM-based approaches provide estimates of incremental poses with respect to the first camera frame, they do not provide global localization. The proposed method is different from previous approaches to global localization [9, 10] as we do not need to render images at test time. Also, since our method does independent predictions for each frame, it does not accumulate errors from previous incorrect predictions.

Using previously predicted poses to infer the current pose while fusing measurements from other modalities, like IMUs (which suffer from long-term drift) and incremental pose estimates from advances in SLAM [20] could be a future direction to make a closed control loop navigation and planning module with neural processing for global localization.

5. REFERENCES

[1] Jean-Claude Latombe, *Robot motion planning*, vol. 124, Springer Science & Business Media, 2012.

[2] Zhenfei Yang, Fei Gao, and Shaojie Shen, "Real-time Monocular Dense Mapping on Aerial Robots Using Visual-Inertial Fusion.," in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE, 2017.

[3] Dryanovski Ivan Srinivasa Siddhartha Klingensmith Matthew and Xiao Jizhong, "Chisel: Real Time Large Scale 3D Reconstruction Onboard a Mobile Device using Spatially Hashed Signed Distance Fields.," in *Robotics: Science and Systems*, 2015.

[4] Vivek Pradeep, Christoph Rhemann, Shahram Izadi, Christopher Zach, Michael Bleyer, and Steven Bathiche, "MonoFusion: Real-time 3D reconstruction of small scenes with a single web camera," in *Mixed and Augmented Reality (ISMAR), 2013 IEEE International Symposium on*. IEEE, 2013, pp. 83–88.

[5] Thomas Schöps, Torsten Sattler, Christian Häne, and Marc Pollefeys, "3d modeling on the go: Interactive 3d reconstruction of large-scale scenes on mobile devices," in *3D Vision (3DV), 2015 International Conference on*. IEEE, 2015, pp. 291–299.

[6] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville, "Deep Learning," Book in preparation for MIT Press, 2016.

[8] Alex Kendall and Roberto Cipolla, "Modelling uncertainty in deep learning for camera relocalization.," in *ICRA*. 2016, pp. 4762–4769, IEEE.

[9] Kejie Qiu, Tianbo Liu, and Shaojie Shen, "Model-based global localization for aerial robots using edge alignment.," in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE, 2017.

[10] Kyel Ok, W Nicholas Greene, and Nicholas Roy, "Simultaneous tracking and rendering: Real-time monocular localization for MAVs," in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE, 2016, pp. 4522–4529.

[11] Ramin Zabih and John Woodfill, "Non-parametric local transforms for computing visual correspondence," in *European conference on computer vision*. Springer, 1994, pp. 151–158.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Identity mappings in deep residual networks," in *European Conference on Computer Vision*. Springer, 2016, pp. 630–645.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[15] Sergey Ioffe and Christian Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.," in *ICML*, Francis R. Bach and David M. Blei, Eds. 2015, vol. 37 of *JMLR Workshop and Conference Proceedings*, pp. 448–456, JMLR.org.

[16] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li, "ImageNet: A large-scale hierarchical image database.," in *CVPR*. 2009, pp. 248–255, IEEE Computer Society.

[17] Vishal M Patel, Raghuraman Gopalan, Ruonan Li, and Rama Chellappa, "Visual domain adaptation: A survey of recent advances," *IEEE signal processing magazine*, vol. 32, no. 3, pp. 53–69, 2015.

[18] Baochen Sun, Jiashi Feng, and Kate Saenko, "Return of frustratingly easy domain adaptation," *arXiv preprint arXiv:1511.05547*, 2015.

[19] Hal Daumé III, Abhishek Kumar, and Avishek Saha, "Frustratingly easy semi-supervised domain adaptation," in *Proceedings of the 2010 Workshop on Domain Adaptation for Natural Language Processing*. Association for Computational Linguistics, 2010, pp. 53–59.

[20] Jorge Fuentes-Pacheco, José Ruiz-Ascencio, and Juan Manuel Rendón-Mancha, "Visual simultaneous localization and mapping: a survey," *Artificial Intelligence Review*, vol. 43, no. 1, pp. 55–81, 2015.