Extracting and Rendering 3D Point Clouds of Objects From 2D Images using Aruco markers

Ashwin R. Bharadwaj, Hardik Gourisaria, Hrishikesh V, K. Shrinidhi Bhagavath, K. S. Srinivas
Department of Computer Science and Engineering
PES University

***Abstract*—This work explores extraction of spatial orientation and geometric structure of objects and landscapes from 2D images using Deep Neural Networks and Stereo Vision based approaches, to render the structure in 3D space to aid in various application in the field of Virtual and Augmented Reality. Regular 2D images are used to generate a Depth Map using Deep Neural Networks or Stereo Vision, which can then be used to model the 3D structure of the objects in the image.**

***Index Terms*—Image processing, variational auto encoder, convolutional neural networks, Point Clouds, Depth Maps**

I. Introduction

Representation of the world in a three dimensional form virtually has various advantages in the field of medical modelling, terrain mapping and building realistic maps for multi player video games. It could be used to aid doctors in creating simulations of surgical procedures. Three dimensional terrain mapping could potentially help defense personnel understand the terrain in treacherous regions. Furthermore, 3d modelling could aid video game enthusiasts in providing realistic gaming experiance.

This work aims to obtain 3D Point Clouds of real-world objects, using only 2D images of the object, without the aid of any special sensors such as the Kinect sensor. This can be done by generating multiple Depth Maps of the various images taken of the object from various viewpoints and using those Depth Maps to generate a 3D Point Cloud of the object.

Existing techniques to generate/render 3D structure and orientation of objects from 2D images use a combination of interpolation algorithms and Deep Learning Models based on Convolution Neural Networks. Convolution Neural Networks deal with scalars and focus on the features in the images as opposed to orientations. This leads to issues such as inaccurate depth and noise and holes in the output when generating the Depth Maps of the Input Images as can be observed.

Another traditional technique, namely "Structure from Motion" [1], determines the relationships between consecutive frames for obtaining the depth of the objects. However, this method works primarily on static 3D frames. The process of determining correspondence between frames fails when fast-moving objects are represented with low textures.

Capsules Networks [2] introduced by Geoffrey Hinton deal with vectors instead of scalars and can learn the orientation and position of the features in the images. This property of Capsule Networks resolves a major issue prevalent in the Convolution Neural Networks and is one of the models explored in this paper. Stereo Vision based approaches have also been explored without the Parallel Camera setup and RGB images have of the objects have been captured from various arbitrary viewpoints.

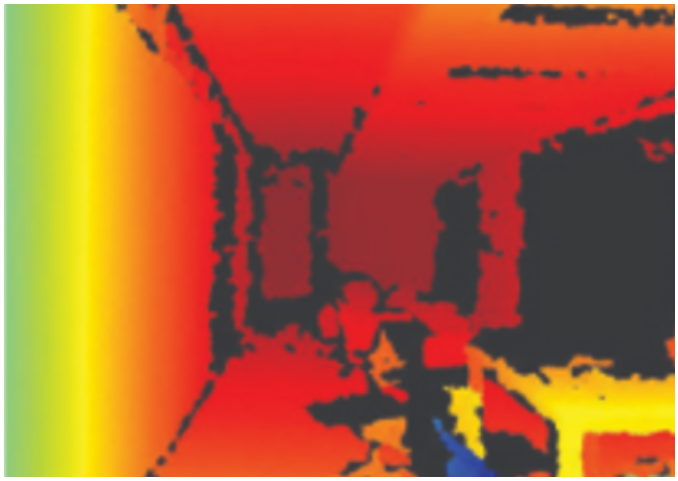

Fig. 1. Depth Map of images when CNN based Deep Learning Networks are used. clearly has a lot of areas where the depth has not been determined (holes).

II. Related Work

Zhao et al. [3] analyse and compare various Monocular Depth Estimation Techniques. Wofk et al. [4] used MobileNet as the encoder for a Deep Learning model to build an efficient light weight Monocular Depth Estimation Model. Lore et al. [5] introduced a GAN based model that used optical flow of images to generate the Depth Map. Facil et al. [6] used the camera parameters to improve the accuracy of the Depth Map generated by the models.

Hinton et al. [2] discuss what a capsule is and why it may be favoured over existing CNN models. The limitation of CNN models is that they deal with scalars and hence look for existence or non-existence of specific features, ignoring the position, orientation, etc. of the features. Due to this, CNNs are likely to fail if any spatial, orientation, etc. variations are introduced to the input. Capsule Networks on the other deal with vectors and hence deal with existence, position, orientation, etc. of features thereby overcoming the mentioned limitation of existing CNN models. The same had been indicated by the improvement in the classification accuracy of overlapping digits in the MNIST dataset, by using Capsule Networks.

Xuan et al. [7] have used a CNN based Deep Learning Model to extract the Depth Maps for all the frames in a video. Any parallax error between the depths maps for two different frames is backpropagated and the model is trained on this to finally generate Consistent depth for the frames in the input video. However, the model is computationally expensive and takes around 1 hour to generate the output video for an input video of 10 seconds duration.

Zhao et al. [8] propose a 3D Capsule Networks model to process 3D data represented as Point Clouds. 3D processing has an advantage over 2D image processing in situations where colour and texture are insufficient to analyse and draw conclusions. 3D Capsule Networks autoencoders are used to reconstruct the 3D images. A Point Cloud is defined as a differentiable 2-manifold embedded in 3D Euclidean space, where a manifold is a space that resembles Euclidean space about the point. A Euclidean space is an affine space, independent of the measure of distance and angles but maintains the properties associated with parallelism and ratio. The input to the 3D Capsule net encoder perceptron is the set of local feature maps that are generated using a pointwise multi-layer perceptron. Each capsule is independent and is a cluster centroid. The model can be used in part replacement and interpolation.

Zhang et al. [9] explore capsule nets with capsule vectors rather than neuron activation (CapProNet). The network learns capsule subspaces which is a projection of input feature vector. The network learns the orthogonal projection matrix. The process of projection is analogous to multi-dimensional weight normalization. The proposed model greatly improves the performance of ResNet and DenseNet. Each subspace corresponds to a class. The length of the capsule is used to estimate the probability that the input sample belongs to one of the classes. However, the projection process is applied only to the output layer of the network.

Saqur et al. [10] have proposed using Capsules based GANs architecture to generate both 2D synthetic images as opposed to using the traditional CNN based GANs. They have replaced the traditional CNN based discriminator of their GAN with a Capsule based discriminator and provided evidence that the Capsule based GAN produces better results than CNN based GAN on generating highly geometrically transformed images on the rotated MNIST dataset. Further, it has been mentioned that there is a potential of using this in generating 3D images of objects and this has yet to be explored.

Xian et al. [11] focuses on using Deep Learning Models to obtain 3D shapes from single or multiple RGB images. The work is organized based on architecture, training method, and output representation. Datasets have been compared according to the year of release, number of images, size, input-output types, image types, number of categories, number of objects per image, and background. The work also presents recent models to construct human faces and bodies along with 2D to 3D reconstruction of objects like chairs, tables, etc. All the existing models are compared by parameters namely training data, testing data, the number of objects per image, background, output type and resolution, used network architecture, IoU (Intersection over Union), time taken and memory. Issues in 2D to 3D reconstruction include lack of training data, generalizing to unseen objects, fine-scale 3d reconstruction, and handling multiple objects in a scene.

Haozhe et al. [12] talk about Pix2Vox, an encoder-decoder model whose output is fed to a context-aware fusion module and then to a refiner. The encoder-decoder is fed images of an object from different points of view and the 3D volumes are obtained as outputs. Context-aware fusion modules are used to adaptively select high-quality reconstructions for each part of the object, from different coarse 3D volumes in parallel to produce a fused reconstruction of the whole object. The context-aware fusion module generates a score map for each coarse volume and then fuses them into one volume by performing the weighted sum of all coarse volumes according to their score maps. The refiner is a small encoder-decoder and is simply used for refining the output to get the final result.

Xi et al. [13] discuss various optimization techniques in an attempt to find the best set of configurations that can be applied to Capsule Nets to yield optimal results on the CIFAR dataset. They have explored dynamic routing - selectively choosing which parent the capsule is sent to. Dynamic routing is shown to be more effective than max pooling. They have used a reconstruction autoencoder to regularize the network instead of using dropouts. This process helps the network learn the general representation of images. They have also increased the number of primary capsules to increase accuracy. To improve accuracy, they have used ensemble training, where a group of networks is trained and the results are averaged. They can conclude that ensemble learning with 4 models, each made up of 2 convolution layers outperformed the other models in terms of training and validation.

Li et al. [14] have used camera intrinsic matrix transformation on images to obtain the approximate image of an object in a segmented image from multiple directions. They do this by assuming the object to be located at the center of a cube and obtaining the image of an object from viewpoints situated on the different faces of the cube. Once these approximate images are obtained, they are used to generate multiple 3D Depth Maps from different viewpoints on the faces of the cube, which are in turn used to generate a dense Point Cloud for the object structure.

[18] article documents all the available functionalities that are built-in OpenCV. The documentation goes into detail on how to obtain the orientation and the pixel coordinates of the Aruco Marker. It also provides details on the algorithm used to estimate the pose of the marker if part of it is obstructed or is out of frame. [19] generates precise Aruco Markers and allows the user to select the dimensions of the marker without hampering its detectability. The markers that are generated using this tool are already present in the OpenCV Aruco dictionaries which makes it easier to integrate with OpenCV's Aruco Detection algorithms.

[20] details the mathematics behind converting the pixel coordinates and the orientation of the Aruco Markers into

real-world coordinates of the marker relative to the camera. The algorithm described will convert any Rotation Matrix into Euler Angles. [21] details the procedure used to calibrate cameras using the chessboard algorithm. It details how to detect and eliminate Radial Distortion and Tangential Distortion. If such distortions are not taken into account, the real-world measurements made based on the data collected from the image will be distorted by a nonlinear function.

[22] details the mathematical model used to represent the pinhole camera model. It also provides a clear understanding of the intrinsic and extrinsic properties of a camera. The intrinsic properties include aspects such as focal length, horizontal and vertical view angles, and the properties of the camera lens, whereas extrinsic properties deal with the orientation of the camera with respect to an arbitrary coordinate system.

[23] details the procedure of rectifying a pair of stereo images. It also detail about the three main types of algorithm used for this task i.e. Planar Rectification, Cylindrical Rectification, and Polar Rectification. Rectification is the process of distorting an image in a manner that would facilitate the calculation of Corresponding Points. To achieve this the Epilines are made to coincide with the Scan Lines.

24 details all the main properties of the Essential and Fundamental matrices. This video also delves deep into the mathematical properties and the algorithms used to calculate the Essential matrices such as the five and eight-point algorithm. The Fundamental Matrix takes into account the properties of the camera and thus works with pixel coordinates, whereas, the Essential Matrix works with calibrated camera and deals with real-world coordinates.

[24] discussed the correspondence problem and explains a few algorithms used to solve the Correspondence problem including Correlation-based Methods, Feature-based Methods. The Correlation-based procedure uses a moving window algorithm to traverse the epiline and finds the best fit. Corners, Edges, and Vertices are used to identify a match for corresponding points using Feature based algorithm.

[25] and [26] discuss the usage of Cosine Similarity and Structural Similarity Index which we have used in our Correspondence problem to find probably corresponding points between a pair of stereo images.

III. Dataset Used

The NYU Depth v2 Dataset was used for training the Depth Map Generation Models. The dataset has 40000 RGB Image and Depth Map pairs. Figure 2 shows a few samples from the dataset. 70% of the dataset was used for training the models, 15% was used for validating the models and the remaining 15% was used for testing the model.

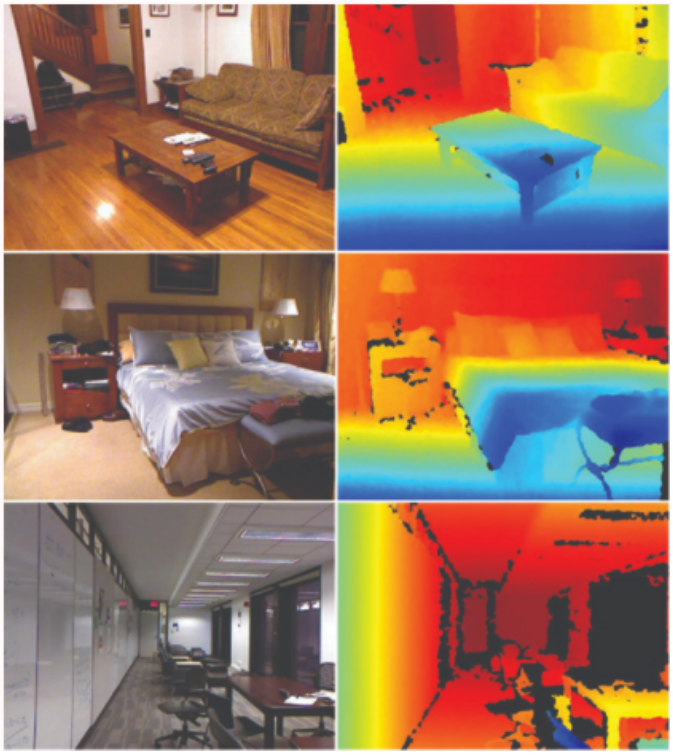

Fig. 2. Samples taken from the NYU Depth v2 dataset

IV. Depth Map Generation

The primary function of the model involves estimation of 2D Depth Maps of 2D RGB images, which can then used to generated a 3D Point Cloud. The model constrains itself to using static frames of the scene rather than videos. This section describes the various approaches that were explored:

- Capsule Network
- Generative Adversarial Network
- Encoder-Decoder Model
- Stereo Vision Based Depth Map Generation

A. Capsule Network based Model

Convolutional Neural Networks assume the input to be scalars thereby emphasizing on the existence of features in images rather than the position and orientation of the features in the images.

To address this issue, Capsule Networks were introduced. Capsule Networks deal with vectors and can retain important information such as the position and orientation of the features in the images. Each capsule layer learns the presence of a feature along with its orientation. We hoped to use this to our advantage and leverage it towards producing more accurate Depth Map from 2D RGB Images.

The output of each capsule is a function of weighted sum of the affine transform of the vectorized input. Affine transform is denoted by the equation below

$$\hat{u}_{j|i} = w_{ij}\bar{u}_i \tag{1}$$

In Eqn 1, $\bar{u}_i$ represents the input vector i. $w_j i$ is the weight. The value obtained is scaled before applying the non linear activation.

$$S_j = \sum c_{ij} \hat{u}_{j|i} \tag{2}$$

$$V_j = \frac{|S_j|^2}{1 + |S_j|^2} * \frac{S_j}{|S_j|} \tag{3}$$

In Eqn 1 and 2, c_{ij} is the value that is learnt by the network. v_j is a non linear function of s_j and is the output of the capsule.

Each capsule passes a high value to the capsule of the following layer if the following layer detects an object that contains the object detected by the previous layer. $c_i j$ is used to determine this relationship. This relationship works similar to k-means clustering. Values that do not agree with each other have a low value in their dot product, while similar objects have a high dot product value. Due to this, a Capsule Network can resist the impact of rotationally distorted images at the cost of consuming high memory and training time.

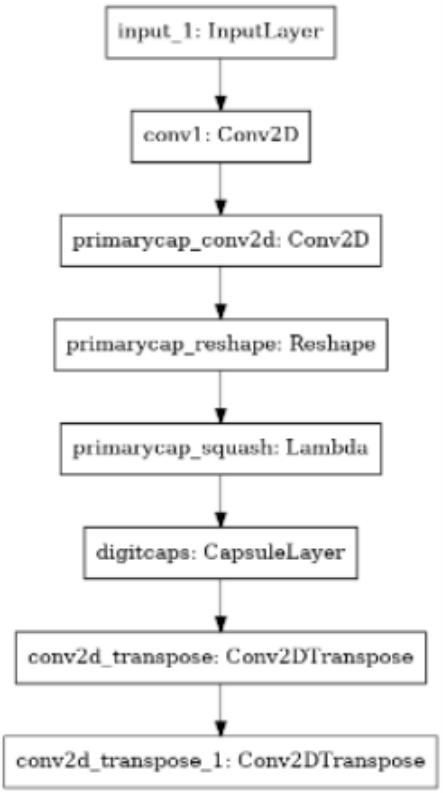

Fig. 3. Architecture of Capsule Network

Capsule networks have a high memory requirement. Hence, the Input Images had to be downsampled to a size of 64 x 64 in order to reduce the number of parameters to be learnt by the model and to reduce the training time and memory requirements of the model. However, even after arduous training of the network, the model did not converge well. We believe that this is due to the following two reasons:

1) Downsampling RGB Images led to huge amount of data loss and hence the model could not learn well
2) Capsule Networks are not suitable for generation of Depth Maps

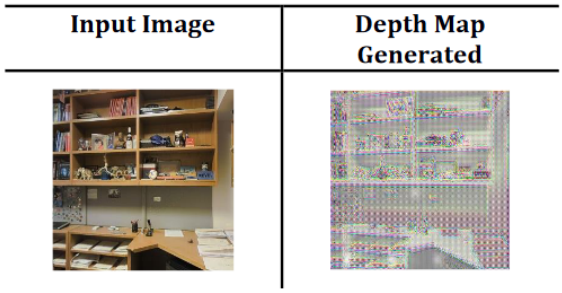

Fig. 4. Performance of Capsule network model

Even after increasing the image size to 128 x 128, no improvements were observed in the performance of Capsule Networks and hence we concluded that that capsule networks are most likely not suitable for generation of Depth Maps. As can be observed, the Depth Map generated looks more like an edge detection output rather than Depth Map generation.

B. Generative Adversarial Network based Model

Generative Adversarial Network is a neural network comprising a Generator and a Discriminator. The Generator is trained to construct a synthetic Depth Map of the input 2D image while the Discriminator is trained to determine whether the synthetic Depth Map is indistinguishable from the actual Depth Map. The discriminator forces the generator to learn the distribution of the input and to simultaneously minimize the loss function in a manner similar to histogram matching. The models learn by being adversaries of each other.

GAN based models were explored to generate Depth Maps where the input to the generator is the RGB Image and the output of the generator was the Depth Map. The generated Depth Map and the expected Depth Map were compared using the discriminator. Once the generator was sufficiently trained, it alone was used to generate the Depth Map from Input Images and the discriminator was discarded. A few modifications were also attempted for the same including

1) Converting input RGB Images to Grayscale Images to reduce the number of features to be learnt and thereby reduce the model size & complexity
2) Representing the Depth Map in the exponential scale to help identify sudden changes in depths across the Depth Map.

GAN based models were able to learn some of the features of the Depth Map but failed to converge to acceptable levels. The Depth Maps generated were extremely pixelated even after excessive training and could not be used to potentially generate the decent Point Cloud. GANs are also prone to

encounter Mode Collapse and that may also be once of the reasons for the failure of the Model.

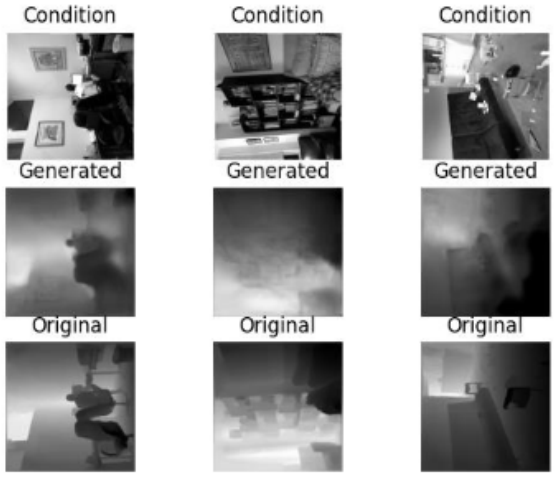

Fig. 5. Performance of GAN based models

C. Encoder-Decoder based Model

A U-Net Model that is based on Encoder-Decoder Model was attempted. A U-Net is a CNN where the outputs of the Encoder Layers are directly concatenated to the corresponding Decoder Layers. This is done to prevent data loss due to down-sampling the encoder layers. Variations of the U-Net Model were attempted where:

- Grayscale Images were provided as input to model to help it converge faster. However, the final Depth Map generated lost a lot of the minute details.
- Model with three different decoders trained in parallel where each decoder identifies the pixels in a certain range distance was attempted where the model did not produce acceptable results.
- Simple Encoder-Decoder Model followed by a series of Simple Convolution Layers where the additional convolutional layers helped the model maintain the correct ratios between the depth of different objects.

Encoder-Decoder based Models with the DenseNet Model as the Encoder provided the best results among the Models attempted so far. Some of the variations tried out also provided comparable results. The variation where Grayscale Images were provided as input to the model, provided results that were nearly as good as the results provided by the model there RGB Images were provided as input. The training time of this variations was also comparatively lesser than the Original Encoder-Decoder Model with RGB Images as input.

Figure 6. indicates the results of the Encoder-Decoder based Model. As can clearly be observed, after just 8 epochs of training the model starts to converge well and the output Depth Map generated make sense. There is no overfitting observed and the training is still underway for a greater number of epochs. However, it must be noted that the Depth Map generated will be accurate only for RGB images captured using the Kinect Sensor as the model is trained only on such images and is highly tuned to the camera parameters of the Kinect and doesn't generalize well to other cameras as stated in [6].

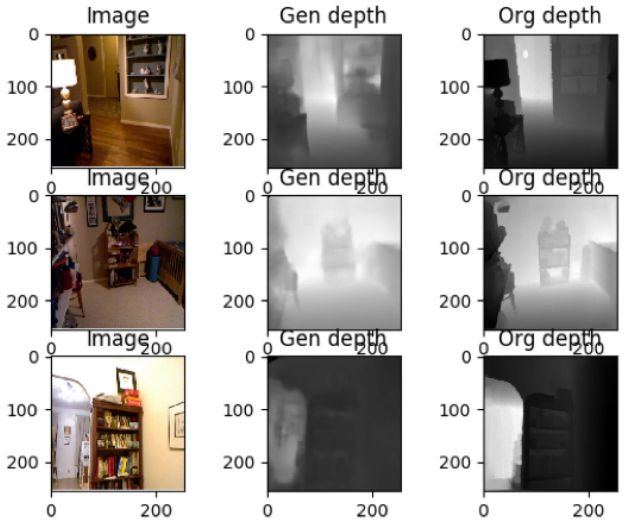

Fig. 6. Performance of Encoder-Decoder U-Net based model

D. Stereo Vision Based Approach

Stereo Depth Map Generation uses Stereo Vision for Depth Map Generation where the orientation of the view-points are obtained from Aurco Markers placed in the scene, thereby eliminating the need for expensive sensors such as the Kinect sensor. This method was attempted because the Deep Learning Based Depth Map Generation approaches that were attempted failed to generate Depth Maps that were satisfactory for our application. Implementing the Stereo Depth Map Generation involved various steps as follows:

1) Determine Camera Position
2) Essential Matrix and Fundamental Matrix Computation
3) Epiline and Epipoint Generation
4) Corresponding Point Detection
5) Depth Map Generation

1) Determine Camera Position and Properties: Having knowledge of the positions of the two stereo cameras is a basic constraint of stereo vision. Usually to deal with this constraint, special camera setups or sensors are used to fix the two camera positions relative to one another. Other methods involve manual determination of the relative positions of the cameras. We have gone a different way and used Aruco Marker based positioning to eliminate any dependency on a special camera or manual labour.

Aruco Marker is placed on the scene and not disturbed. Images of the objects in the scene are captured from multiple directions such that the complete Aruco Marker is present in the images. Aruco Marker detection is performed and the orientation of the Aruco Marker is determined along the 6 degrees of freedom. Using the orientation of the Aruco Marker,

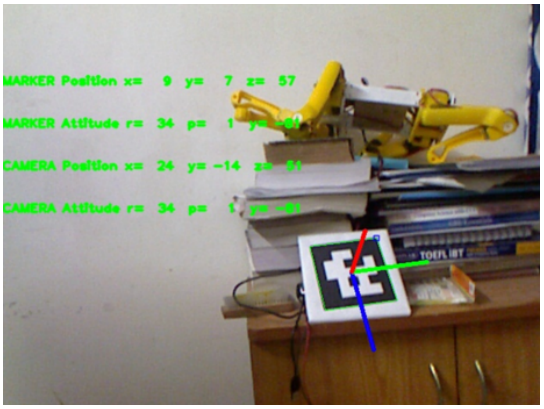

Fig. 7. Detection of Orientation of Aruco Marker

the orientation of the camera while capturing a particular image can be determined.

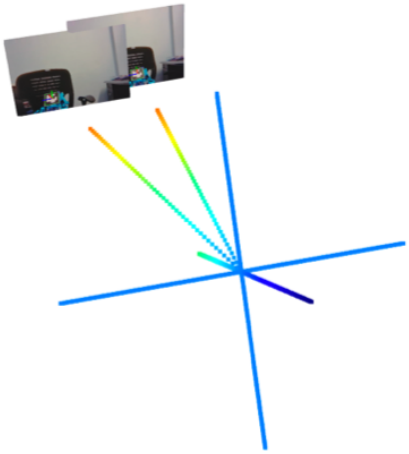

Fig. 8. Determining Position of Camera when the two images were captured

2) Essential Matrix and Fundamental Matrix Computation: Essential Matrix and Fundamental Matrices are computed using the information for the Rotation and Translation of the Camera obtained in the previous step as follows:

$$E = R[T]_x$$

where E is the Essential Matrix and R and T are the Rotation and Translation Matrices. We also know that

$$E = K^T E K$$

where F is the Fundamental Matrix and K are the intrinsic Camera Properties. Using th above, F can be computed as

$$F = (K^{-1})^T E K^{-1}$$

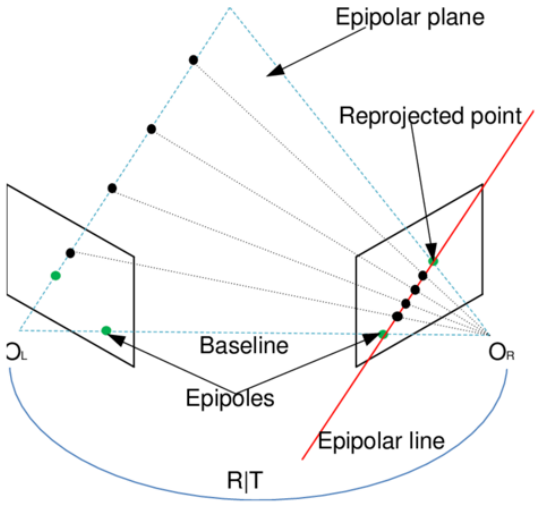

Fig. 9. Summary of Epipolar Geometry

3) Epilines and Epipoints Generation: Epilines and Epipoints are computed using the Essential or Fundamental Matrices. Epilines are lines on the second image in the stereo image pair along which the potential Corresponding Point may lie. Epilines can be obtained as follows: we know that

$$(x')^T E x = 0$$

x is a point in the first image and x' is the corresponding point in the second image and E is the Essential Matrix. The above constraint is a check for the validity of the Essential Matrix as it has been obtained through real world measurements.

Ex is a [3,1] matrix and $(x')^T = [x, y, 1]$. Using this, we can get an equation of a line called the Epiline along which x' may lie. Similar to Essential matrix, the Epiline for x' can be computed using the Fundamental Matrix also. The difference between Essential Matrix and Fundamental Matrix is that Essential Matrix used Real-World Coordinates and Fundamental Matrix uses Pixel Coordinates.

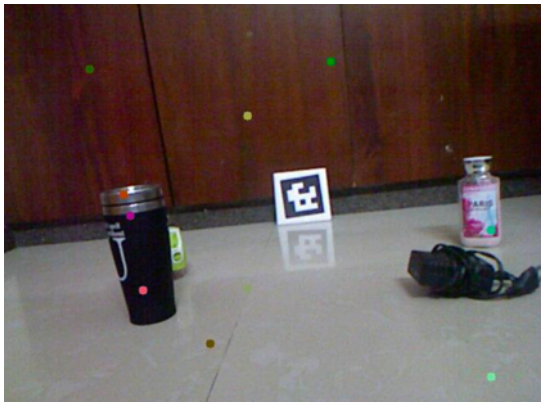

Fig. 10. Randomly Sampled Points on First Image

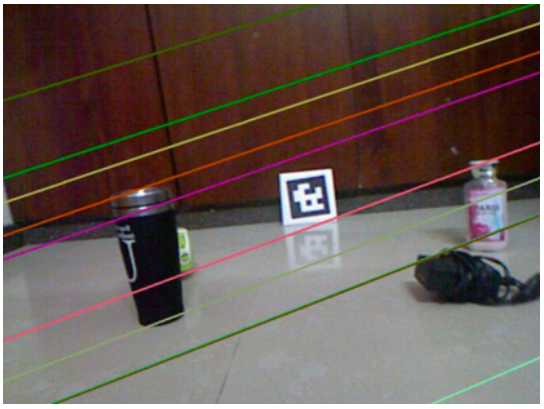

Fig. 11. Epilines Generated for the Second Image

4) Corrsponding Point Detection: Generation of Depth Map required us to obtain the corresponding point pairs between the first and second images of the same scene captured from different viewpoints. Points that are not common between the two images will be ignored.

Using the Epilines obtained from the previous step, we check for possible corresponding points between the pair of images and once a possible corresponding point is obtained, we assume a box around that point and use various methods to check for the similarity between the neighbourhood of these points. These include using SSIM (Structural Similarity Index), Cosine Similarity etc.

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_x\sigma_y + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

$$CosineSimilarity(x,y) = \frac{x.y}{||x||||y||}$$

5) Depth Map Generation: The depth of a point in real world can be computed as

$$z = \frac{fT}{x_2 - x_1}$$

where f is the focal length of the camera, T is the displacement of the camera and x_1 and x_2 are the corresponding points.

V. Point Cloud Generation

The Depth Map generated cannot be directly rendered as a Point Cloud. To render Point Cloud in 3D space, a series of mathematical transformation needs to be performed to obtain the Point Cloud data in the desired format and Deep Learning is not required in this step.

For generation of the Point Cloud from the Depth Map, it is necessary for the Field of View Angles of the the camera to be known along the X and Y axis. The field of view of the Kinect Sensor used is 70^0 and 60^0 approximately.

We can consider the schematic in Figure 13 for generation of Point Cloud data along the X-axis.

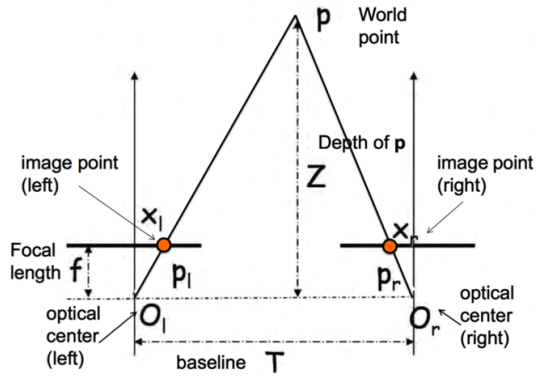

Fig. 12. Recovering Depth from Stereo Images

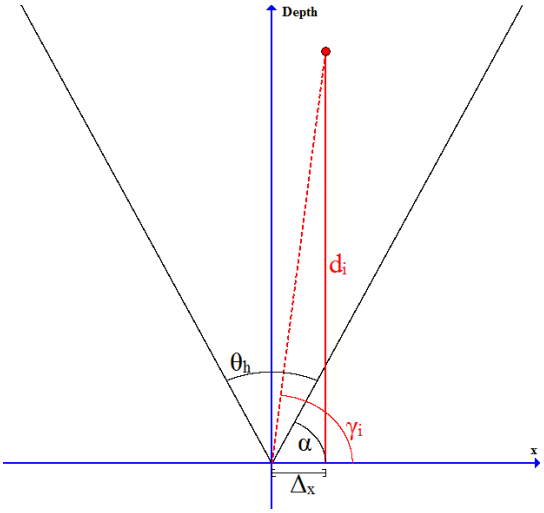

Fig. 13. Schematic for Point Cloud Generation along X-axis

d_i is the orthogonal distance of the object i (red dot) from the plane of the camera in X-axis. The x-coordinate of the object i can then be represented as Δ_x and can be computed as

$$\Delta_x = \frac{d_i}{tan(\gamma_i)} \tag{4}$$

where γ_i is the angle between the object and the sensor. γ_i can be computed as

$$\gamma_i = \alpha + \frac{c_i\theta_h}{n_c} \tag{5}$$

where θ_h is the horizontal view angle of the camera, c_i is the x-pixel position of the object i, n_c is the horizontal resolution of the image, and α is an angle that can be computed as

$$\alpha = \frac{\pi - \theta_h}{2} \tag{6}$$

Similarly, the schematic in Figure 14 can be considered for the generation of Point Cloud data along the y-axis. The method of generation of the Point Cloud along the Y-axis is similar where

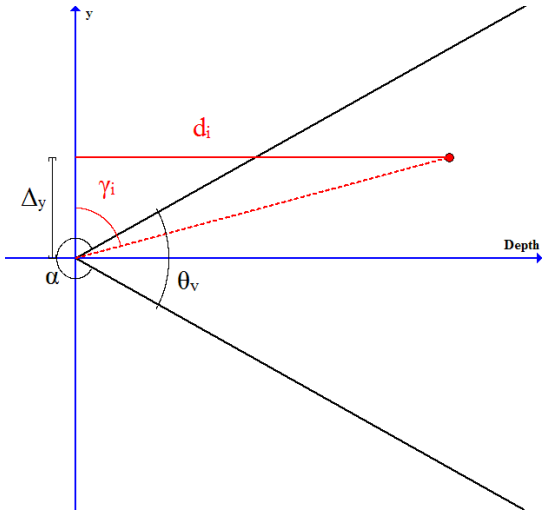

Fig. 14. Schematic for Point Cloud Generation along Y-axis

$$\Delta_y = d_i tan(\gamma_i) \tag{7}$$

$$\gamma_i = \alpha + \frac{r_i \theta_v}{n_r} \tag{8}$$

$$\alpha = 2\pi - \frac{\theta_v}{2} \tag{9}$$

The above computations are performed on each and every pixel in the Depth Map. On performing the above transformations on the Depth Map data, the Point Cloud can be rendered using the Δ_x, Δ_y and depth matrices. The colors for each point in the Point Cloud can be obtained from the original RGB image using the x and y pixel position.

Multiple Images of the same object can be captured from different angles and positions and Depth Maps can be generated. These Depth Maps can be then used to obtain the Point Cloud which can then be rendered to obtain a more detailed 3D representation of the object. For our work, we have used the Open3D Module in Python for rendering the 3D Point Cloud.

VI. Results

A. Deep Learning based Depth Map Generation

Figure 15 indicated the RGB Input Images captured using a Smartphone Camera. These images are provided as input to the Encoder-Decoder U-Net Model to obtain Depth Maps. Figure 16 shows the output Depth Maps for the input RGB images captured using the Smartphone camera. As can be observed, the Depth Map visually appears to be accurate. However, on generating the Point Cloud using these generated Depth Maps, we observe a lot of distortions in the Point Cloud as can be observed from Figure 17. Hence we conclude that the Deep Learning Model has not generalized well for RGB images captured from cameras other than the Kinect Camera.

Fig. 15. Input RGB Images from SmartPhone Camera

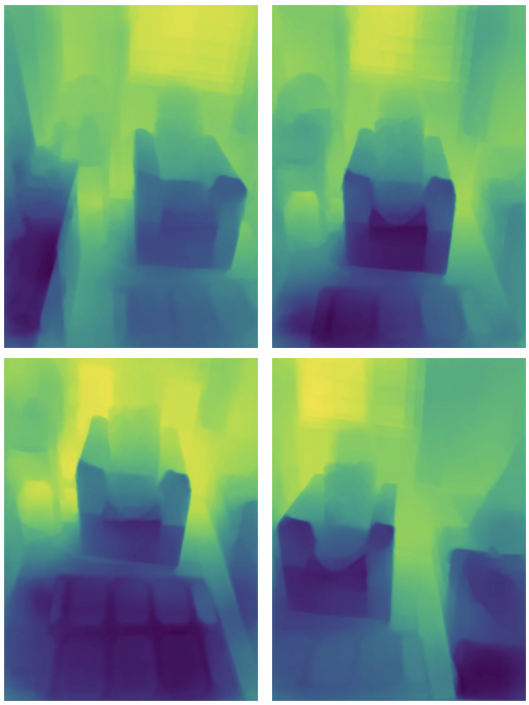

Fig. 16. Generated Depth Maps for Smartphone Pictures by the Encoder-Decoder U-Net Model

Figure 18 are the RGB Input Images captured using the Kinect Camera and Figure 19 is the Point Cloud Generated after generating the Depth Map for these images using the Encoder-Decoder U-Net Model. As can be clearly observed, the distortions in the Point Cloud are considerably less compared to the distortions in the Point Cloud obtained from the Smartphone Camera. This is expected from what is stated in [6]. Hence we conclude that our Deep Learning Model works well but is highly camera specific.

To eliminate any specific camera dependency, Stereo Vision based Depth Map generation was attempted.

Fig. 17. Generated Point Cloud for the Smartphone Camera Pictures

Fig. 18. Input RGB Images from Kinect Camera

Fig. 19. Generated Point Cloud for the Kinect Camera Pictures

B. Stereo Vision Based Depth Map Generation

Figure 20 is the reference RGB Input Image to the Stereo Vision based approach from the Stereo Image Pair. Figure 21 is the generated Depth Map for the Input Image. The generated Depth Map is comparable to the results obtained by other Stereo Vision based approaches while eliminating dependency on any special camera setup through the usage of Aruco Markers. This eliminates any constrain of knowing the camera positions in advance. However, the Depth Map is not yet accurate enough to be used to generate a Point Cloud with minimal distortion.

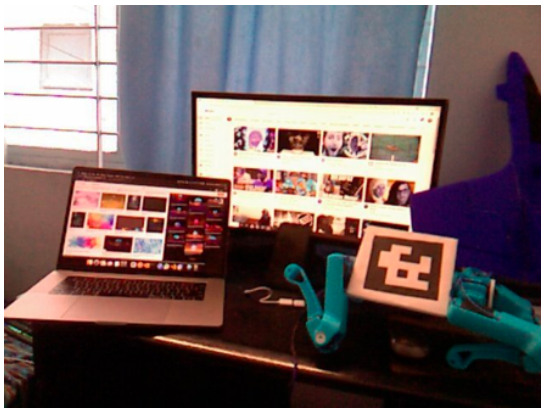

Fig. 20. Reference Input Image to the Stereo Vision based approach from the Stereo Images Pair

Fig. 21. Generated Depth Map for the Reference Input Image

VII. Conclusion

As can be concluded from the previous sections, we have generated Depth Maps successfully for the 2D RGB Input Images using the Deep Learning Models. Although, the Deep Learning Model did not generalize well to generate accurate Depth Maps for images captured from arbitrary cameras, the accuracy of the Depth Maps generated for images captured with the Kinect Camera by the Deep Learning Models generated good 3D Point Clouds.

To generalize Depth Map generation across all cameras, Stereo Vision based Depth Map generation was explored

where we considered images of the scene capture from arbitrary viewpoints and did not restrict ourselves to a parallel camera setup as is used in various expensive sensors. We were able to successfully calibrate the cameras and obtain Aruco Marker orientations and the Camera orientation for the images. We were also able to accurately compute the Essential Matrix and Fundamental Matrix and Epilines. Depth Maps were also generated using the Stereo Vision based approach and the quality of the Depth Maps generated were comparable to other similar approaches while eliminating any dependency on specialised camera setup to determine the camera positions in advance. However, the Depth Map generated is not accurate enough to generate good Point Clouds and hence required more work.

VIII. Limitations and Future Work

Our future work involves:

- Working on the shortcomings of our progress so far which include development of better Deep Learning Models to generate Depth Maps
- Work needs to be out into exploring better algorithms for obtained the Corresponding Points in the image pairs
- Also explore an approach were the two methods of Depth Map generation are combined and Deep Learning is used to generate the Depth Map using Stereo Image Pairs

References

[1] P. Harman, J. Flack, S. Fox, and M. Dowley, "Rapid 2d-to-3d conversion," *Proceedings of SPIE - The International Society for Optical Engineering*, vol. 4660, pp. 78–86, 05 2002.

[2] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic routing between capsules," 2017.

[3] C. Zhao, Q. Sun, C. Zhang, Y. Tang, and F. Qian, "Monocular depth estimation based on deep learning: An overview," *Science China Technological Sciences*, vol. 63, 06 2020.

[4] D. Wofk, F. Ma, T.-J. Yang, S. Karaman, and V. Sze, "Fastdepth: Fast monocular depth estimation on embedded systems," 05 2019.

[5] K. G. Lore, K. Reddy, M. Giering, and E. Bernal, "Generative adversarial networks for depth map estimation from rgb video," pp. 1258–12588, 06 2018.

[6] J. Facil, B. Ummenhofer, H. Zhou, L. Montesano, T. Brox, and J. Civera, "Cam-convs: Camera-aware multi-scale convolutions for single-view depth," pp. 11818–11827, 06 2019.

[7] X. Luo, J.-B. Huang, R. Szeliski, K. Matzen, and J. Kopf, "Consistent video depth estimation," *ACM Transactions on Graphics*, vol. 39, 07 2020.

[8] Y. Zhao, T. Birdal, H. Deng, and F. Tombari, "3d point capsule networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1009–1018, 2019.

[9] L. Zhang, M. Edraki, and G.-J. Qi, "Cappronet: Deep feature learning via orthogonal projections onto capsule subspaces," in *Advances in Neural Information Processing Systems*, pp. 5814–5823, 2018.

[10] R. Saqur and S. Vivona, *CapsGAN: Using Dynamic Routing for Generative Adversarial Networks*, pp. 511–525. 01 2020.

[11] X.-F. Han, H. Laga, and M. Bennamoun, "Image-based 3d object reconstruction: State-of-the-art and trends in the deep learning era," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PP, pp. 1–1, 11 2019.

[12] H. Xie, H. Yao, X. Sun, S. Zhou, and S. Zhang, "Pix2vox: Context-aware 3d reconstruction from single and multi-view images," 07 2019.

[13] E. Xi, S. Bing, and Y. Jin, "Capsule network performance on complex data," 12 2017.

[14] K. Li, T. Pham, H. Zhan, and I. Reid, *Efficient Dense Point Cloud Object Reconstruction Using Deformation Vector Fields: 15th European Conference, Munich, Germany, September 8–14, 2018, Proceedings, Part XII*, pp. 508–524. 09 2018.

[15] Z. Li and N. Snavely, "Megadepth: Learning single-view depth prediction from internet photos," pp. 2041–2050, 06 2018.

[16] Y. Kuznietsov, J. Stückler, and B. Leibe, "Semi-supervised deep learning for monocular depth map prediction," pp. 2215–2223, 07 2017.

[17] C. Godard, O. Aodha, M. Firman, and J. Gabriel, "Digging into self-supervised monocular depth estimation," 11 2019.

[18] "https://docs.opencv.org/3.1.0/d5/dae/tutorial_aruco_detection.html."

[19] "https://chev.me/arucogen/."

[20] "https://learnopencv.com/rotation-matrix-to-euler-angles/."

[21] "https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_calib3d/py_calibration/py_calibration.html."

[22] "https://web.stanford.edu/class/cs231a/course_notes/01-camera-models.pdf."

[23] "http://www.sci.utah.edu/~gerig/cs6320-s2012/materials/cs6320-cv-f2012-rectification.pdf."

[24] "https://www.cse.unr.edu/ bebis/cs791e/notes/ stereocorrespondenceproblem.pdf."

[25] "https://en.wikipedia.org/wiki/structural_similarity."

[26] "https://www.sciencedirect.com/topics/computer-science/cosine-similarity: :text=cosine%20similarity%20measures%20the%20similarity, document%20similarity%20in%20text%20analysis."