Documentation:

Visual SLAM – by Hritik Singh Kushwah

Project Overview

Title: -Implementing Visual SLAM using only camera and image data.

Our project is divided into 3 tasks:

1. Depth Estimation
2. Perspective Transformation
3. 3-D Visualization

Research and Resources Referred:

- Read all the resources provided and researched further more about Neural Networks and CNN.
- Watched some videos and tutorials about SLAM (not Visual SLAM specifically) but were insightful for getting certain overview. LINKS:
    1. https://www.youtube.com/watch?v=MxuBLW8hmRY&t=339s
    2. https://www.youtube.com/watch?v=jI1Qf7zMeIs (Created by the authors of the GitHub repo. referred)
    3. https://www.youtube.com/watch?v=sIN1Tp3wIbQ
    4. https://www.youtube.com/watch?v=Kja394OvoGY (Another approach for Depth Estimation)
- Analysing and getting familiar with the GitHub repo. Shared: https://github.com/nianticlabs/monodepth2 (MAIN)
- Tested only models 'Mono for 640x192 and 1024x320' and 'Stereo_640x192 and 1024x320' on sample street images through 'depth_prediction_example.ipynb' Jupyter Notebook.
- Referred to the original paper written by the same author of the referred GitHub repo. https://arxiv.org/abs/1806.01260
- Completed the 'Neural Network and Deep learning' by Andrew Ng. I covered the following topics -
    1. Forward Propagation, Back Propagation, Gradient Descent, Different Optimizers like Adam and its implementation in TensorFlow.
    2. Optimization techniques like mini-batch.
    3. How to implement a small neural network using Logistic Regression and create an Image Classifier with 70% accuracy.
- Completed 3 weeks of CNN course by Andrew Ng and following sub-topics has been covered:
    1. Basic Convolution Network step by step implementation.

2.Tensorflow tutorial
3.Basic Concepts like Max pooling, average pooling ,stride and padding.
4. About ResNets and Inception Network and their architecture.
5. Implemented above concepts to create a model to train and test SIGNS dataset.

- For Perspective Transformation part I've referred mostly to the following articles: -
  1. https://towardsdatascience.com/depth-estimation-1-basics-and-intuition-86f2c9538cd1
  2. https://towardsdatascience.com/inverse-projection-transformation-c866ccedef1c
- The GitHub repo referred to for perspective transformation :- https://github.com/darylclimb/cvml_project/tree/master/projections

Environment Setup:

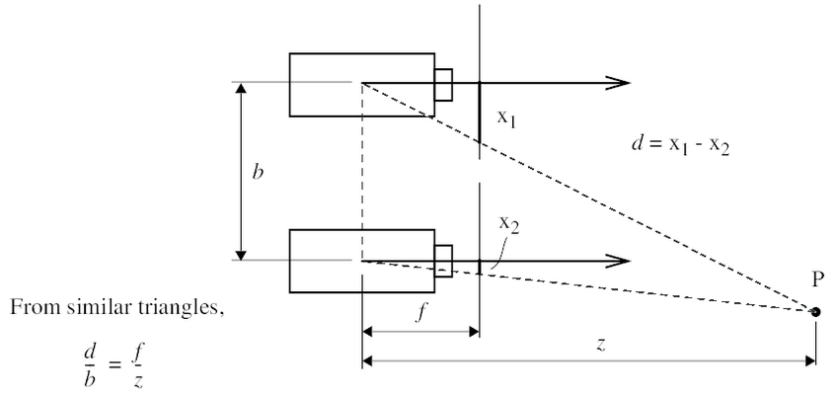
Following installations are required if you are using a fresh Anaconda Distribution: -

1. conda install pytorch=0.4.1 torchvision=0.2.1 -c pytorch
2. pip install tensorboardX==1.4
3. conda install opencv=3.3.1
4. pip install open3d

Basic Algorithm and Understanding the Model for Depth Estimation: -

Stereo Images: -

For estimating depth of a point in an stereo image we use two sample images of the object captured from both the cameras simultaneously and we can estimate the depth of a point through finding distance between the points on the plane of both the cameras to determine the angle that point projects to the observer. This is termed as binocular disparity. The same method is used by eyes to interpret how far an image is (like parallax).

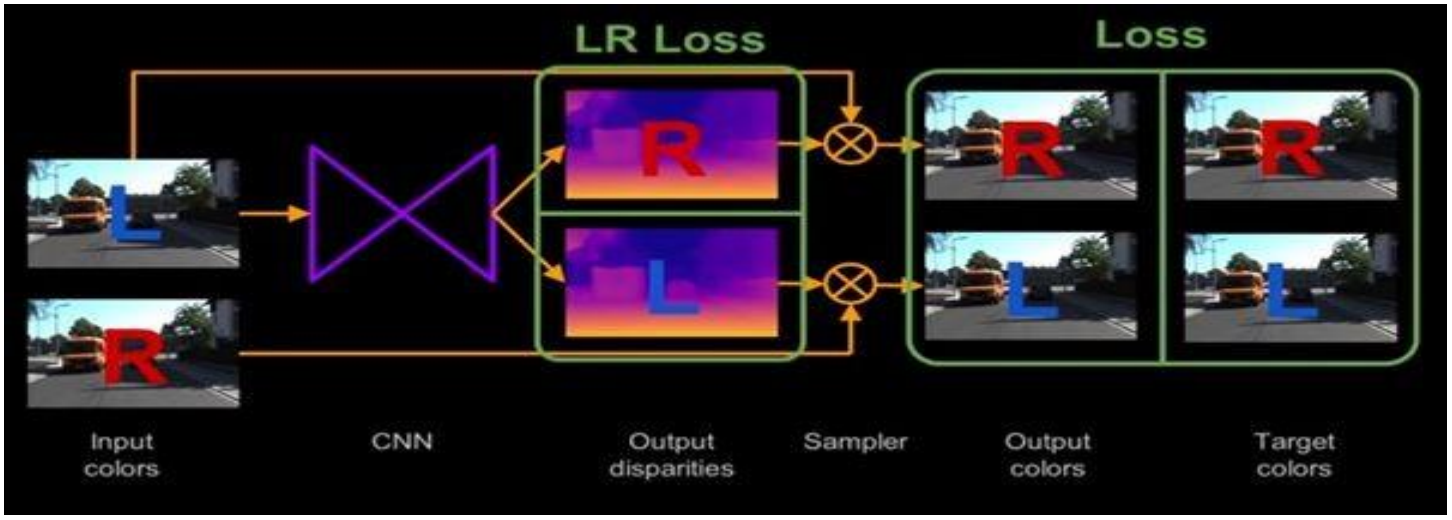


So for training a computer to perform the similar tasks we would try to match the feature from left image pixel in the right image and computing the distance between the two points will end

up giving us the disparity(i.e. d= x2-x1). This disparity can further be used to calculate depth from given the distance between the cameras ‘b’ and focal length ‘f’ using formula [depth = f*b/d].
Monocular Images: -

Due to unavailability of a second reference image in monocular images we are unable to predict their depth. To resolve this, we can train a CNN model which takes a left image as an input and try to predict its corresponding right image and implicitly we estimate depth of the given point.

Understanding the model



Input: - A single left image (A right image is required for training only)

Variable to predict: - Disparity Map.

Procedure:

In this model, we train a CNN model to predict the right image of a given left image. The CNN model generated a disparity map of the right image which when applied to the left image we can reconstruct the right image. Here the loss function is the error between the target image (the actual Right Image) and the predicted right image. The reconstruction is done using a sampler STM (Spatial Transformer Network). To make the model more efficient and lossless the CNN simultaneously tries to predict the left disparity with right disparity to maintain the left -right consistency.
This disparity map can further be used to generate a relative depth map. To generate an absolute depth, we still require information of focal length (f)and the distance between the cameras(b).

Using Formula to estimate depth of a pixel: - [depth = (f*b)/(disparity)]

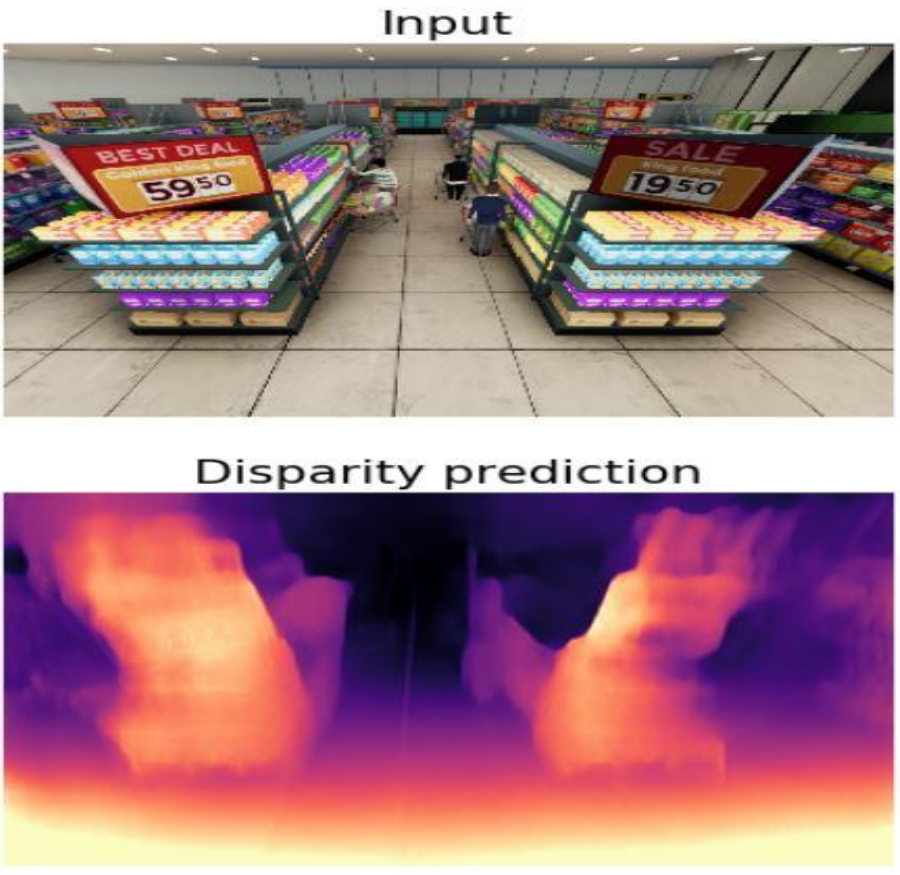
For Code Implementation of running the model on your test image is given in the Jupyter Notebook ‘Explanation in JupyterNotebook.ipy’ in the GitHub repo. with MARKDOWN Texts to guide you through the code.

Observation by using Mono_640x192 model on different sample images

Note: Prediction time is in milliseconds.

1.

```
Model->mono_640x192
Model Predicting time--2843.75

----- Plotting time time = 562.5ms
```

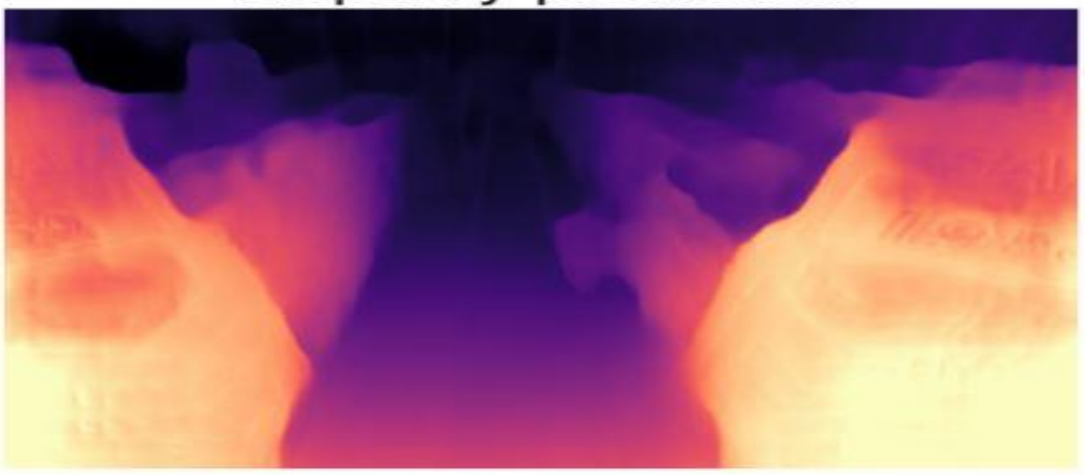


2.

```
Model->mono_640x192
Model Predicting time--3000.0

----- Plotting time time = 515.625ms
```

Input

Disparity prediction



3.

```
Model->mono_640x192
Model Predicting time--2750.0

----- Plotting time time = 562.5ms
```

4.

```
Model->mono_640x192
Model Predicting time--2875.0

----- Plotting time time = 500.0ms
```

I've submitted only 'Mono_640x192' model because it gave the best results for depth estimation and inference time for the given input images as compared to stereo in these specific cases.

Model Wise Analysis:

Analysis of 4 Models:

1. Mono_640x192
2. Stereo_640x192
3. Mono_1024x320

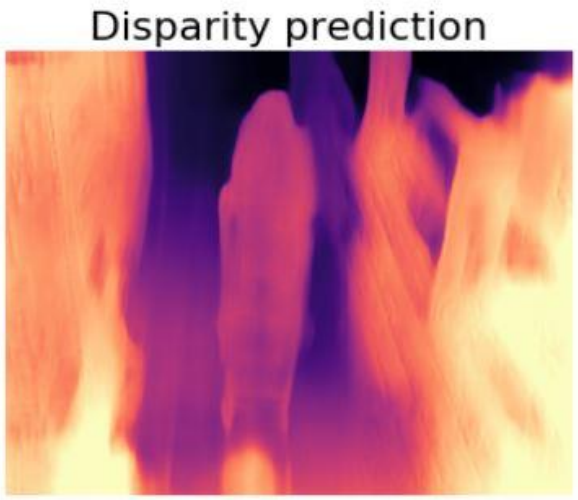
4. Stereo_1024x320

Observations:

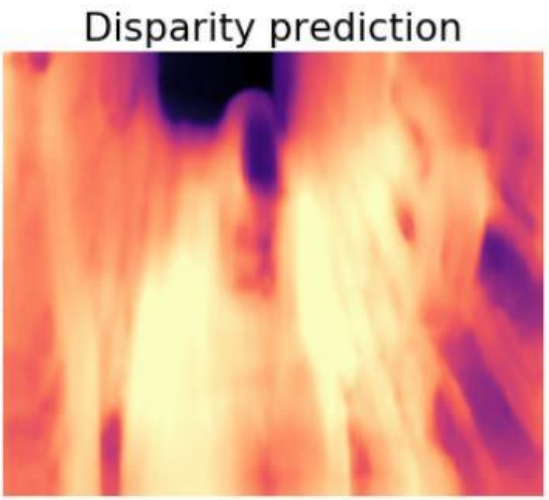
For 640x192 Resolution:

Original Resolution of the Sample Image: - 1778x1420 (Padding is done in both the models).

```
Model->mono_640x192
Model Predicting time--2750.0

----- Plotting time time = 562.5ms
```

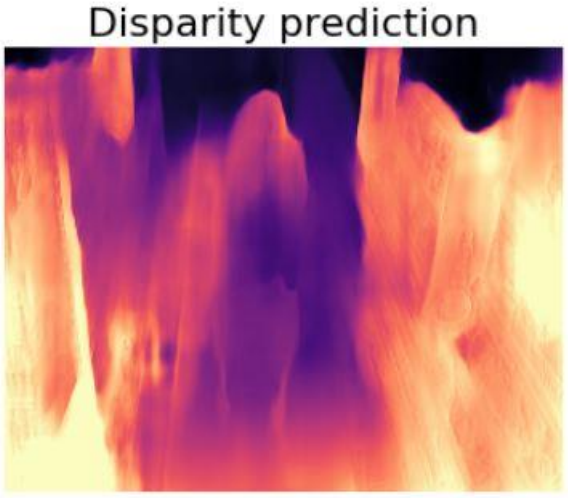



```
Model->stereo_640x192
Model Predicting time--3015.625

----- Plotting time time = 500.0ms
```

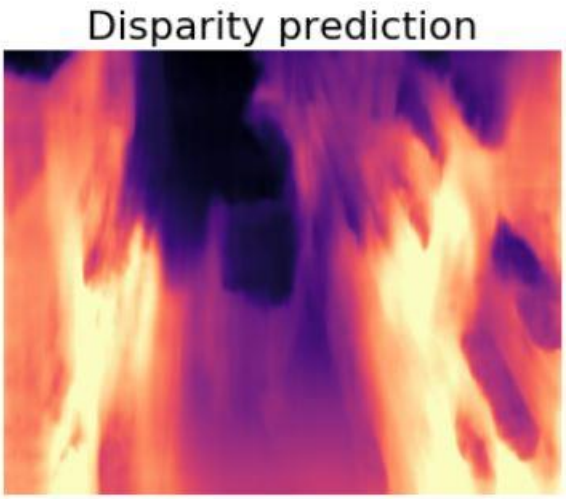


For 1024x320 Resolution:

```
Model->mono_1024x320
Model Predicting time--8375.0ms

----- Plotting time time = 500.0ms
```



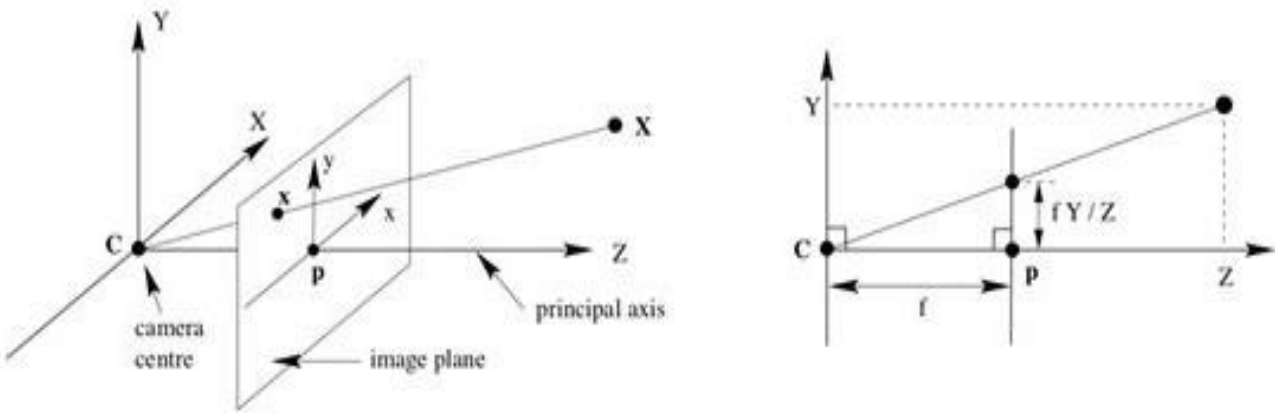

```
Model->stereo_1024x320
Model Predicting time--8328.125ms

----- Plotting time time = 625.0ms
```



It can be concluded from sample outputs that Models mono_640x192 and mono_1024x320 works better in terms of image segmentation and depth estimation than the rest. So, both models can be used according to the resolution of the image.

Algorithm and Code implementation for Perspective Transformation

To generate origin points for perspective transformation using the depth image. We first must understand the geometry involved.



In the above image x is the pixel in the image plane whereas X represents its depth in the Z principal axis.

To perform the transformation of any point or p in image plane to 3-D plane we can use the following transformation: - **p = K[R|t] * P** or matrix representation: -

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Where:

1.[u,v,1]/p ->the image plane coordinates

2.s -> aspect ratio scaling

3. P, [X, Y, Z, 1] ->the 3D point

4. K is the camera intrinsic matrix. In the matrix (fx,fy) represents the focal length coordinates wrt to camera position and (uo,vo) center coordinates.

5.[R/t] is the extrinsic matrix for relative transformation from world frame to camera frame(we will ignore this as P is already assumed in camera frame)

So, we can just find the inverse of the K and multiply with pixel coordinates and flatten the depth for standardizing then we can obtain the 3-D points:

Code for the same:
While running the file you will to input the field of vision (it's recommended to use some value greater than the maximum depth value to get the RGB color scheme)

```
def Perspective_transformation(rgb,depth,args):
    height,width,_ = rgb.shape

    K = intrinsic_from_fov(height, width, 90)  # +- 45 degrees
    K_inv = np.linalg.inv(K)
    pixel_coords = pixel_coord_np(width, height)  # [3, npoints]

    # Apply back-projection: K_inv @ pixels * depth
    cam_coords = K_inv[:3, :3] @ pixel_coords * depth.flatten()

    #Select the range till u want to project points in 3-D
    limit_of_depth = input("Limit of the depth to view\n") # upper limit of depth
    limit_of_depth = float(limit_of_depth)
    cam_coords = cam_coords[:, np.where(cam_coords[2] <= limit_of_depth)[0]]

    return cam_coords
```

Cam_coords contains the 3-D points.

3-D visualization of the points

To visualize the generated 3-D points into a 3-D space we will use an open-sourced library called Open3d.

It has a visualization in-built class which takes 3-D points as Point Cloud and plots it in 3-D space. Here is the sample code to run the 3-D output with depth intensity color scheme:-
Note: - Here the depth range of relative depth has been set from 0.12 units to12 units. We will try to visualize till 8 units to avoid the stretch on horizon points.

```
# Limit points to 8 units in the z-direction for visualisation
cam_coords = cam_coords[:, np.where(cam_coords[2] <= 8)[0]]

# Visualize
pcd_cam = o3d.geometry.PointCloud()
pcd_cam.points = o3d.utility.Vector3dVector(cam_coords.T[:, :3])
pcd_cam.colors = o3d.utility.Vector3dVector(rgb_t.astype(np.float) / 255.0)
# Flip it, otherwise the pointcloud will be upside down
pcd_cam.transform([[1, 0, 0, 0], [0, -1, 0, 0], [0, 0, -1, 0], [0, 0, 0, 1]])
o3d.visualization.draw_geometries([pcd_cam])
```

Generated Depth picture obtained from disparity map by scaling the disparity inverse:-

Final Run Using the ‘FinalDemo.py’ file using mono_640x192 model

The FinalDemo.py file contains all the combination of depth estimation, generating points in 3-D space and it’s visualization. It also contains comments for better understanding. To run the finalDemo.py file use the following command.

```
python finalDemo.py --image_path assets/s3.png --model_name mono_640x192
```

It takes 2 arguments that are image path and model name as shown in the command.

Output of the program: -

1.RGB vs Depth Image

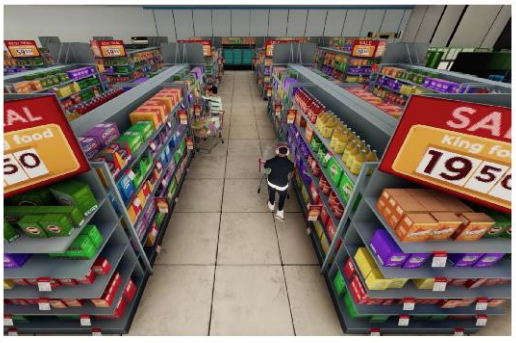
2. Min and Max Value of Depth

3. 3-D plot

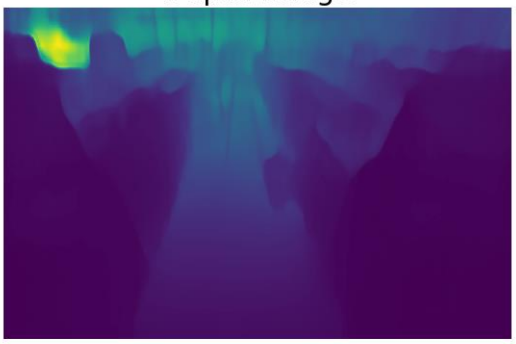
Example: -

Estimating disparity image from RGB image using mono_640x192 model. Then obtaining the depth image from disparity image (See page no.4) by selecting the range of depth as [0.1,1000] units and comparing the RGB and Depth image.

RGM IMAGE



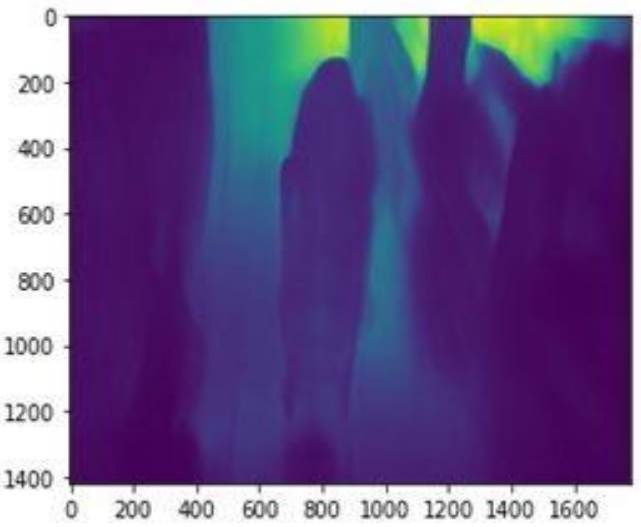
Depth image



Then we'll apply perspective transformation to the depth image by generating origin points for 3-D plotting :-
Here are some screenshots from the 3-D view :-

Using 3-D intensity based color scheme:-

original (rgb image)



depth image

3-D intensity plot :-

As the depth is interpolated due to its high resolution then that of the model, depth of some points was extrapolated and hence looks stretched.

Issues Faced and Recommendations: -

1. Depth estimation was not accurate as the results seems when tested on KIIT dataset that is 87.6% so depth of some points is interpolated and gave ambiguous results.

   To rectify this problem the suggestion is to train your custom datasets on the pre-trained models. You can train on a custom monocular or stereo dataset by writing a new dataloader class which inherits from MonoDataset – see the KITTIDataset class in datasets/kitti_dataset.py for an example. (For a precise demo you can refer to the monodepth2 repository 'Training Section'

2. I faced many issues running and training the model on a Windows Operating System and lower GPU specifications.

So I'll suggest to switch to Linux based Operating System or to use Google Collab as they provide a 12GB GPU with 60GB Drive space and is compatible with Linux commands.

3. While running the finalDemo.py you will need to input the upper limit of the depth you want to view in the 3-D plot. If your limit is less than the absolute upper limit, then the plot will be generated using intensity (magma) colour scheme and not the RGB colour scheme as the model.

4. It is recommended to first go through the documentation of open3D library to understand the 3-D visualizer. http://www.open3d.org/