4. METHODOLOGY

4.1. Stereo Camera Setup

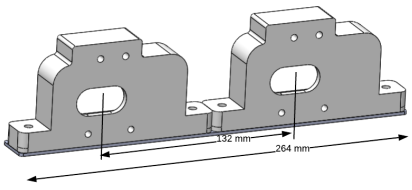
4.1.1. Selection of Camera

For depth estimation using stereo vision, we have used two **Logitech C270** [figure 4.1] cameras with the same specification (i.e focal length(f), the field of view(FoV) etc). As the real parameters doesn't exactly match the specified theoretical values, there is a little manufacture distortion in camera parameters. For our application, we required undistorted images so, We have estimated parameters of the camera using images of a special calibration pattern (9x7 square chessboard). The parameters include camera intrinsic, distortion coefficients, and camera extrinsic values.

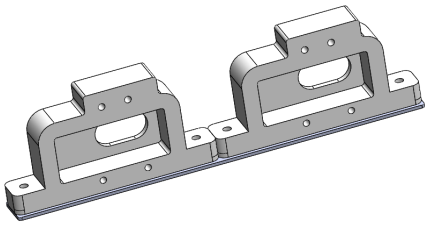
Figure 4.1: Logitech C270

4.1.2. Design of Stereo-Camera holder

For holding the cameras in a stationary position and making it work for stereo-vision, we designed the case using Solid-Works CAD software and 3D printed the model which is as shown in the figure 5.3a and 5.3c. We placed both of our cameras in 3D printed frames which are kept on the same plane. But during camera fitting there is a little error, so the optical axis of cameras are not parallel to each other which results in large disparity error which made it difficult to find matching points between images. It is impractical to maintain perfect coplanarity between cameras.



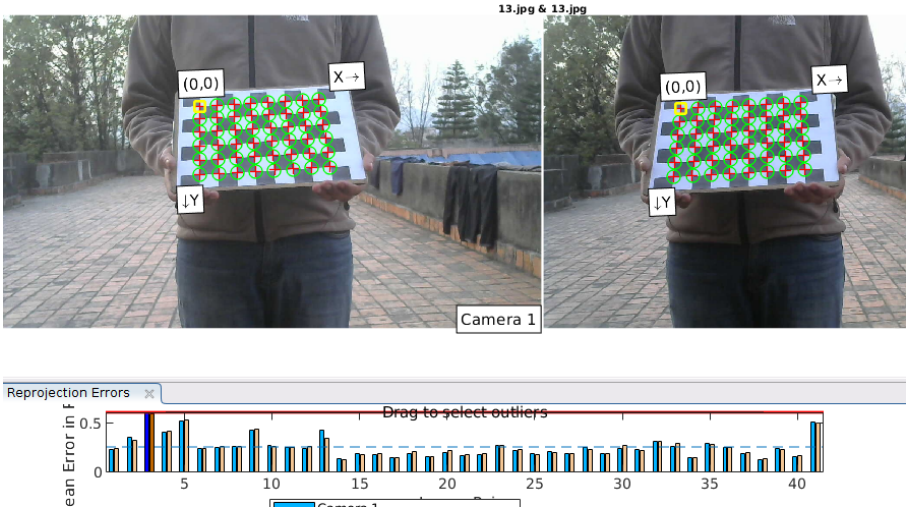
(a) Front View of CAD model



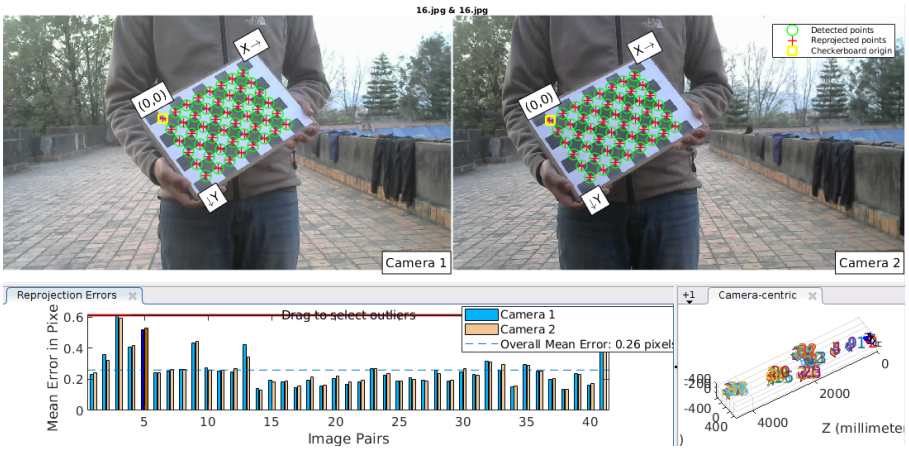
(b) Front View of CAD model

Figure 4.2: Front and back views of custom-made stereo-vision camera case

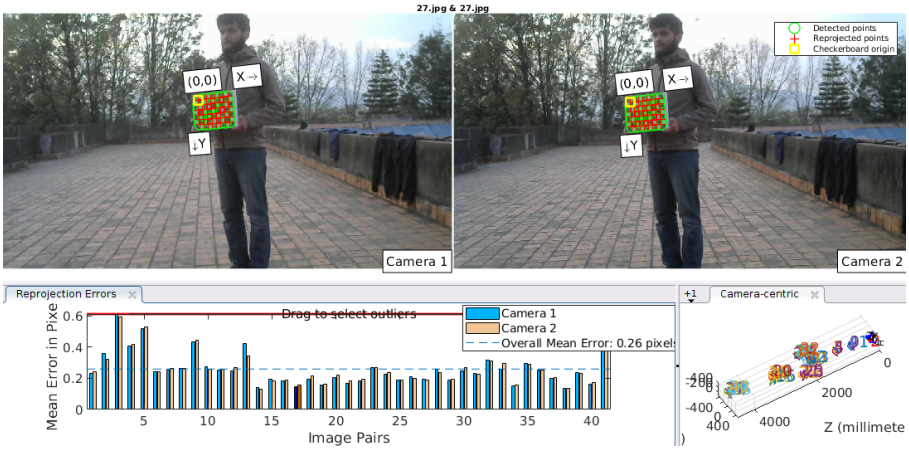
4.1.3. Camera Parameters Calculation Using Chessboard Pattern



(a) Image 1

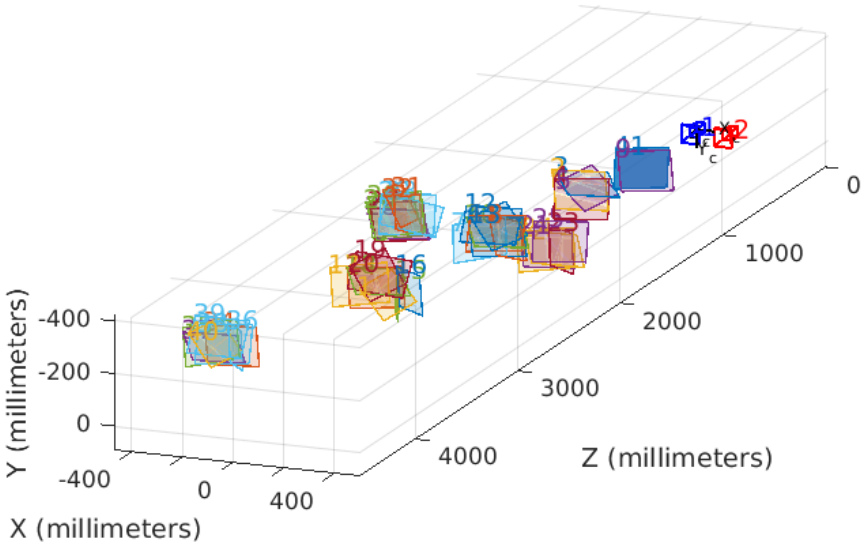


(b) Image 2

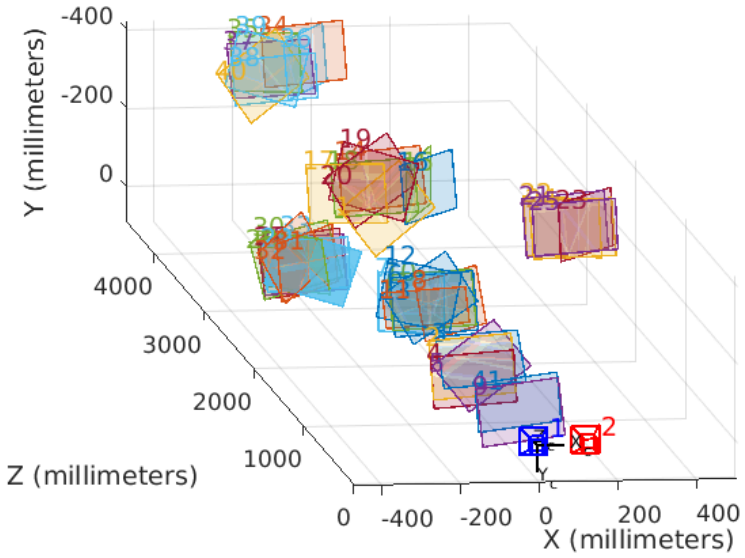


(c) Image 3

Figure 4.3: L and R images for camera parameters calculation



(a) View 1



(b) View 2

Figure 4.4: Views of multiple chessboard placements keeping camera position constant

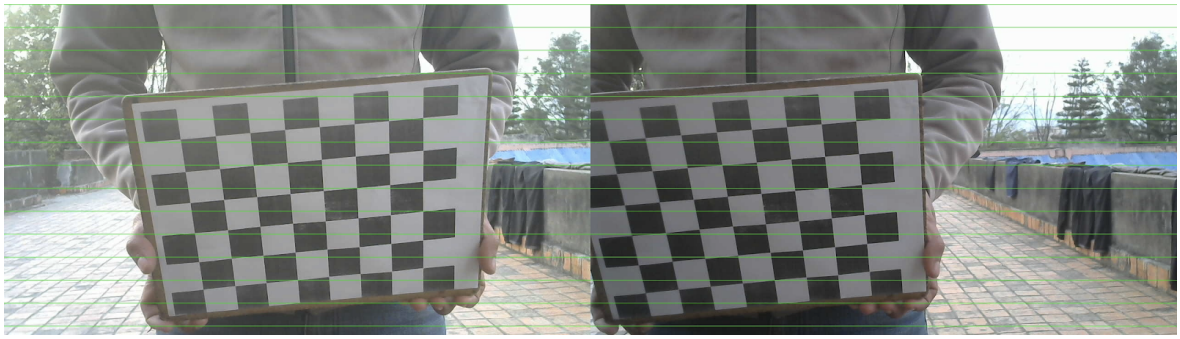
By clicking multiple images that contains 9*7 chessboard as a reference, we were able to find out different camera parameters (intrinsic and extrinsic) and distortion parameters using *Matlab Stereo Camera Calibrator* plugin. Also we can visualize different positions of the photo clicks in 3D space by keeping the camera position constant as depicted in the figure 4.4.

All those parameters that we calculated are shown in the table 4.1.

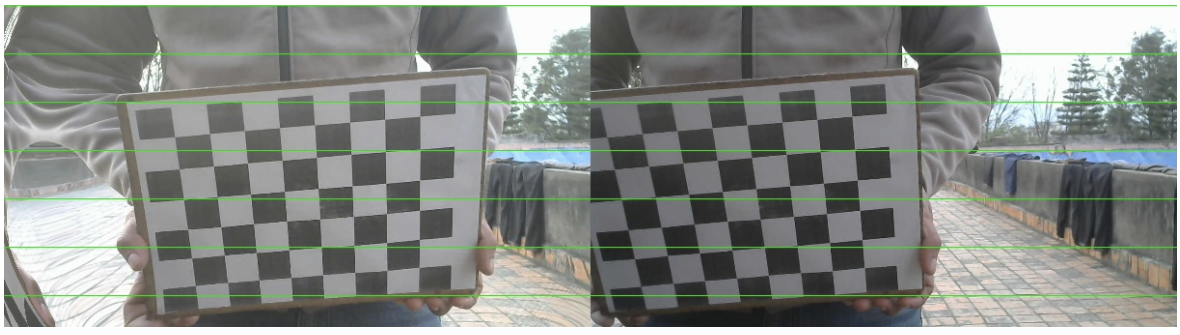
Parameters	Camera 1	Camera 2
Intrinsic Matrix $\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & & 1 \end{bmatrix}$	$\begin{bmatrix} 1439.58 & 0 & 693.067 \\ 0 & 1457.027 & 327.84 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1435.32 & 0 & 696.75 \\ 0 & 1448.44 & 313.23 \\ 0 & 0 & 1 \end{bmatrix}$
Radial Distortion $\begin{bmatrix} k_1 & k_2 & k_3 \end{bmatrix}$	$\begin{bmatrix} -0.069 & 3.17 & -22.44 \end{bmatrix}$	$\begin{bmatrix} -0.069 & 0.874 & -1.77 \end{bmatrix}$
Tangential Distortion $\begin{bmatrix} p_1 & p_2 \end{bmatrix}$	$\begin{bmatrix} -0.0074 & 0.016 \end{bmatrix}$	$\begin{bmatrix} -0.0079 & 0.014 \end{bmatrix}$

Table 4.1: Camera Parameters

4.1.4. Image Rectification



(a) Stereo Images [L & R] before rectification



(b) Stereo Images [L & R] after rectification

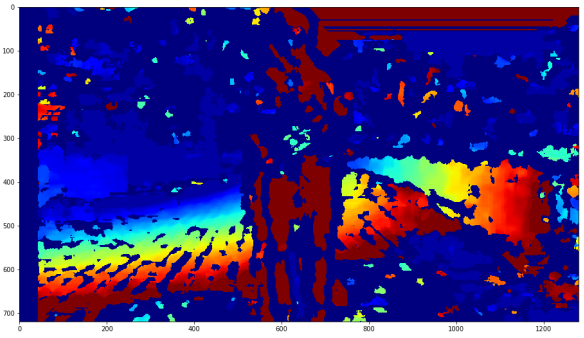
Figure 4.5: Image Rectification

In order to solve the above problem we have used **Image rectification** [as shown in figure 4.5] which warps both images such that they appear as if they have been taken with only a horizontal displacement and as a consequence all epipolar lines are horizontal, which slightly

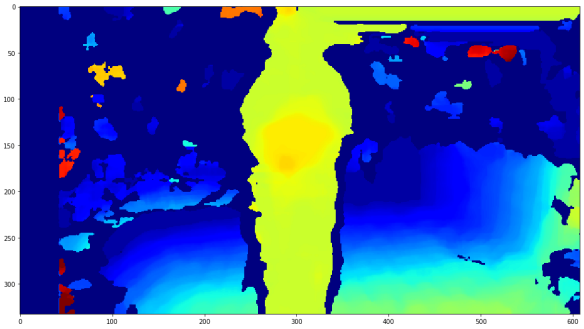
simplifies the stereo matching process. During image rectification one of the cameras is rotated and translated with respect to the other camera coordinate such that, the optical axis are parallel to each other. Therefore, it projects image planes onto a common plane parallel to the line between optical centers. It uses both intrinsic and extrinsic parameters to find rotation, translation and projection matrix for one camera with respect to another.

Before rectification there were no point matches in between left and right images, so it generated larger errors in the correspondence problem. But after rectification, both image plane are co planar and hence the image points are located in same horizontal line which make easier for the correspondence problem.

(a) Test Image



(b) Disparity map before image rectification



(c) Disparity map after image rectification

Figure 4.6: Comparison of disparity maps before and after rectification

4.1.5. Stereo Matching and Calculation of Depth

We have used **Semi Global Block Matching(SGBM)** algorithm for the correspondence problem. It uses block based cost matching. The cost is later smoothed by using the path wise information from multiple path direction. OpenCv has an efficient semi global matching algorithm implementation which is called semi global block matching (*SGBM*). The used function is *StereoSGBM_create()* that takes following parameters as input:

Parameters	Description	Used Value
minDisparity	Minimum possible disparity value	10
numDisparities	Maximum disparity minus minimum disparity	122
blockSize	Minimum possible disparity value	16
P1	The first parameter controlling the disparity smoothness	$8 * 3 * window\_size * *2$
P2	The second parameter controlling the disparity smoothness	$32 * 3 * window\_size * *2$
disp12MaxDiff	Maximum allowed difference (in integer pixel units) in the left-right disparity check	1
uniquenessRatio	Margin in percentage	10
speckleWindowSize	Maximum size of smooth disparity regions to consider their noise speckles and invalidate	100
speckleRange	Maximum disparity variation within each connected component	32

Table 4.2: Parameters for function *StereoSGBM_create()* and their description

Since our focal length is 1390 pixels and baseline distance between two cameras is 132 mm , we can find depth for each pixel using this generalized equation no. 4.1.

$$D = \frac{(1390 * 132)}{d}\, mm \tag{4.1}$$

4.1.6. Implementation of Siamese Network

Detailed architecture of the Siamese network used in this project is shown in figure 3.4. For training our Siamese network, we use kitti dataset 5.4. The image size of the kitti dataset is 1241 × 376, but in our network we have randomly cropped the images in 28x28 patches to increase the number of training data and to reduce the computational cost. Siamese network generalizes for different image resolution, training is done with 28x28 image patch while testing is done with higher resolution. We have trained Siamese network at learning rate of 1e-3, batch size of 16 for 400k iteration. The output of Siamese network is promising but we cannot solely rely upon the above Neural Network for the extraction of depth from the pixels. Various image filters need to be applied in order to smoothing the output and reduce

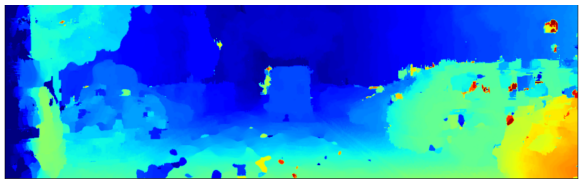
the image grains and noises So, we have tried with different filters. Different filters and its hyper-parameters is described in the next section.

4.1.7. Applying Different Filters on Stereo Output

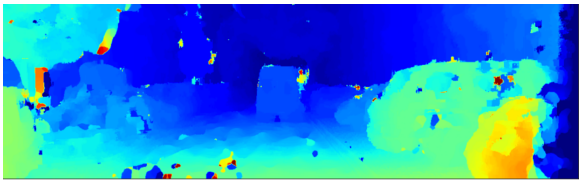
From figure: 4.7 we can clearly see that the raw output has a discontinuous and noisy output with lots of visual artifacts. We apply different methods to mitigate this issue. After applying left right consistency, discontinuity and noise from depth map is reduced to a tolerable limit. We don't prefer mean filter for this purpose since, a single noise in its neighbouring pixel can greatly affect its result. We achieved a promising result after applying median filter as salt and pepper(impulse) noise is slightly reduced. To achieve continuous and smother output we have further applied a bilateral filter which preserves edges and provides a clear boundary smother depth map. Further, we tested with Fast Global Smother Filter(FGSF). After median filter in-place of bilateral filter , we applied FGSF filter. We can see output in Figure: 4.7 that FGSF filter has more smother and continuous depth map in compare with bilateral filter. But bilateral filter results higher accuracy in compare to FGSF.

(a) Left RGB Image

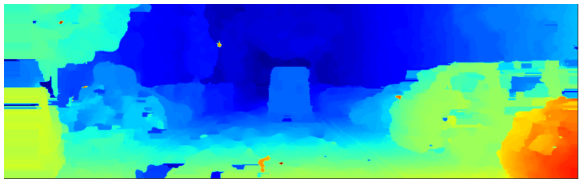
(b) Right RGB Image



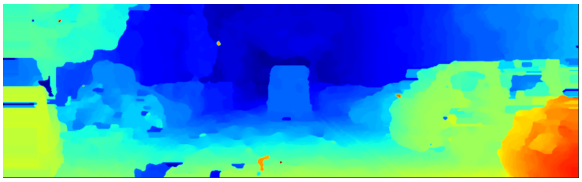
(c) Left disparity after CBCA



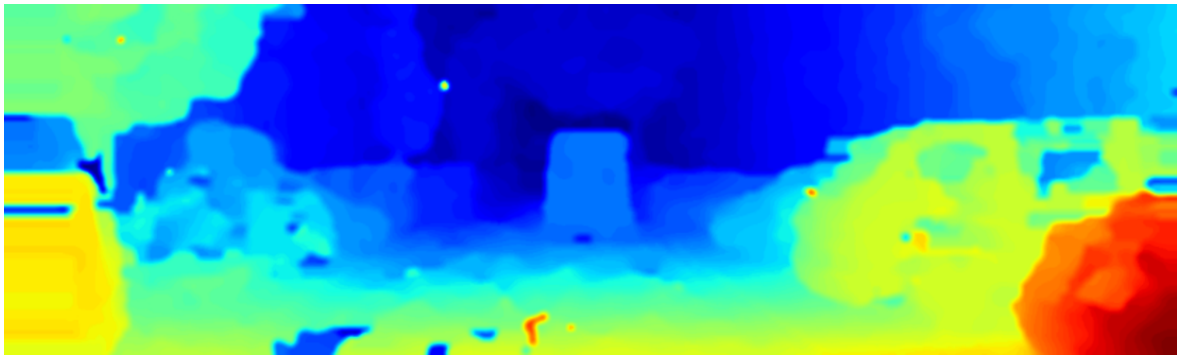
(d) Right disparity after CBCA



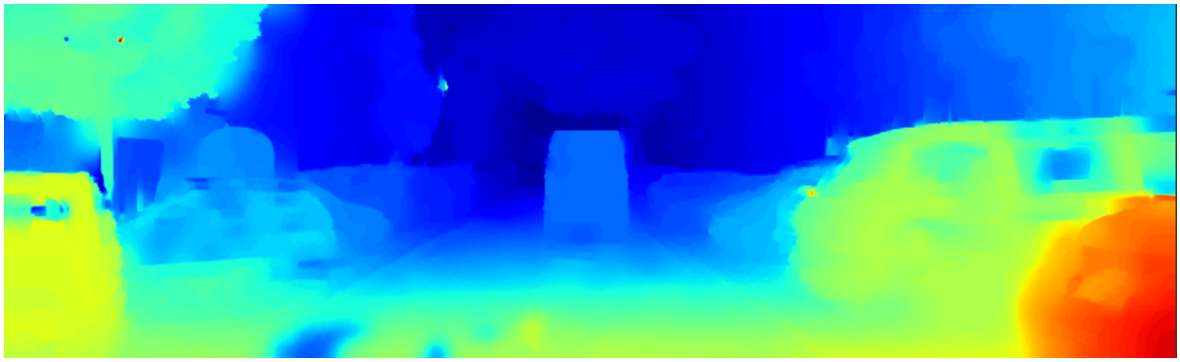
(e) Disparity map after left right consistency



(f) Disparity map after Median Filter



(g) Result after applying bilateral filter



(h) Final result after applying FGSF

Figure 4.7: Applying different filters on stereo(Siamese Neural network) output