Assignment - 3 Report

Anurag Sahu (2018121004)
Vineeth Gaddam
Team id - 40

Q1. STEREO DENSE RECONSTRUCTION

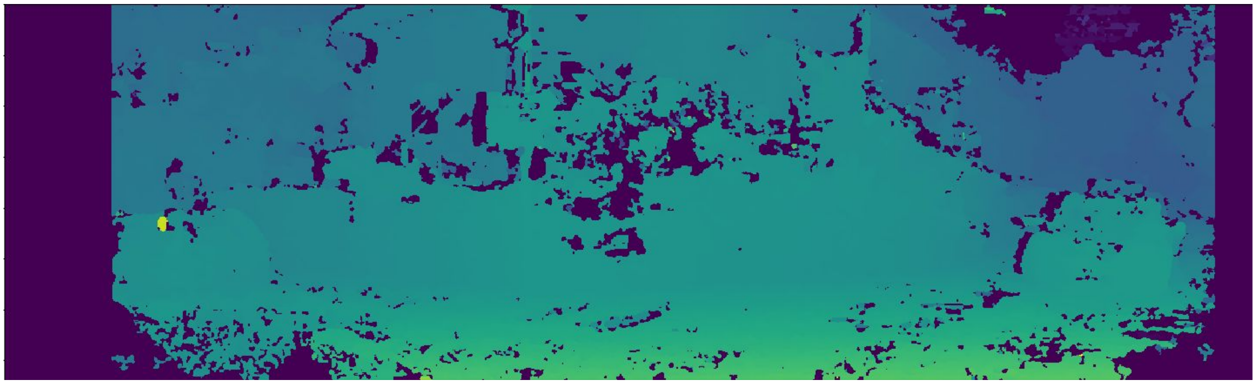
Task : We are given 21 pairs of stereo images with calibration matrix and their Respective ground truth values, and also the baseline values from this data we have to reconstruct a 3d Point cloud representing all the points from the images.

Steps to get the Point clouds:

1. Get the Disparity Map from stereo image pair.
   Math :
   D = x1 - x2
   Where x1 is the location of a point in the left image and x2 is the location of the point in the right Image.

   Code :
   Using the inbuilt function of Open CV, StereoSGBM_create using the tuning parameters of inspired by the blog post :
   http://timosam.com/python_opencv_depthimage
   And then using stereo.compute we calculate the disparity values.

   Output :



2. **Get the point cloud for a pair of images:**

   Math :

   The 3d Point cloud of the images can be obtained by using these disparity values. The formula will be

$$Z = (b * f)/(x1 - x2)$$
$$X = (Z * x)\ /\ f$$
$$Y = (Z * y)\ /\ f$$

   Where:

   b = baseline parameter provided in the question

   f = Focal Length obtained from the K matrix

   x = (x1+x2 /2)

   y = (y1+y2 /2)

   Code :

   We do this operation using the Q matrix way, Were the Q matrix as defined in the Slides [Q matrix](). And Multiplied the Q matrix using Disparity_map with is [x,y,d,1]

   Output :

3. **Register the generated points and into world frame using the given ground truth values (poses.txt)**

   Math :

   We have 3d point [w*x,w*y,w*z,w], and using the Projection matrices in ground truth we get the registered 3d point in the point cloud of a single world frame.

   Code:

   For each of the point in the point cloud multiply the point from the respective projection matrix and get and append these points into a single point cloud. And then visualize them.

OutPut:

For Q2 : Motion Iteration using iterative PnP

Step 1: Select a random R and t and then use them to project these 3d points into an image such that we now have 3d to 2d Correspondences.

Step 2: Now using these 3d to 2d parameters we again estimate this P matrix using an iterative method called Gauss-Newton(GN), A good initialization could to to take the DLT of the J matrix.

- Jacobian Matrix Calculation:

$X = ((p11 * x1) + (p12 * y) + (p13 * z) + (p14))$
$Y = ((p21 * x1) + (p22 * y) + (p23 * z) + (p24))$
$Z = ((p31 * x1) + (p32 * y) + (p33 * z) + (p34))$

$X = X/Z$
$Y = Y/Z$

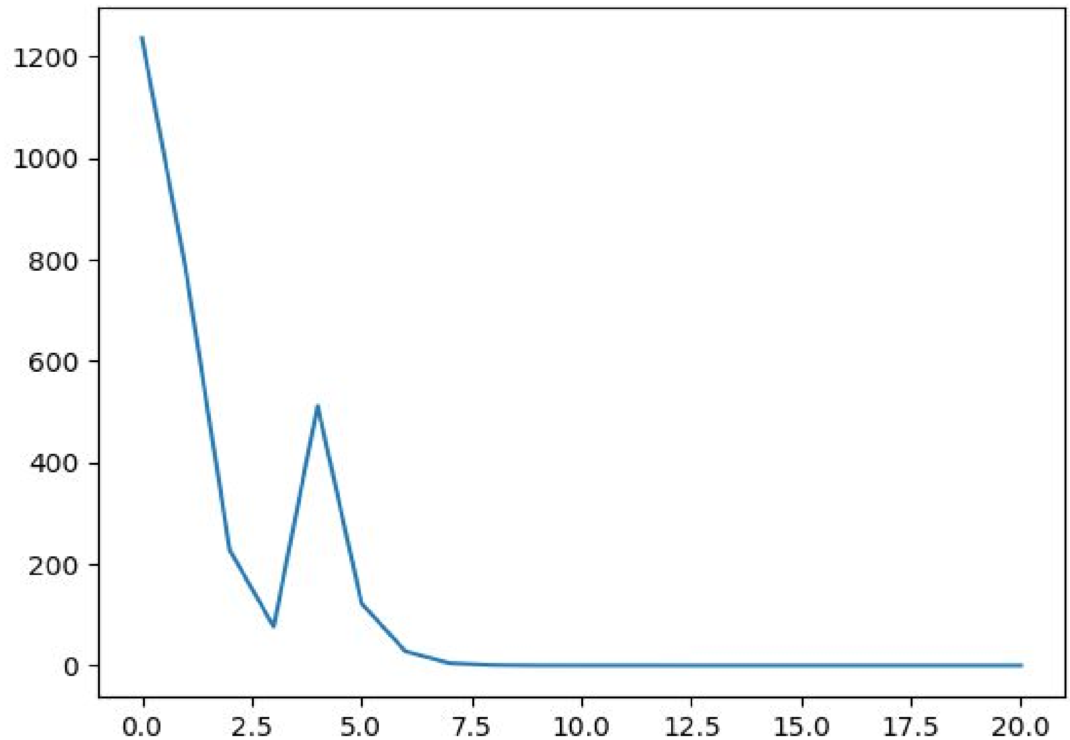
Now we have to estimate all the values $p11, p12, p13, ... p34$ so we differentiate X and Y wrt to $p11, p12, p13, ... p34$ and append it to the jacobian matrix.

- Update the P matrix
  We have the $(JTJ) - 1JT$ matrix and residual matrix $(x - PX)$ $(JTJ) - 1JT$ has dimension $12\ X\ 2n$ and residual matrix has a dimension of $2n\ X\ 1$.

  Multiplying these we get the values of P and then we subtract these P values with the Old Values of P to get the updated P values.

- **Condition for Convergence :**
  We stop the algorithm when we get the Gradient Value (J * residual Matrix) lesser than some error value or cross a certain limit of iterations.

Observation :
The Error Graph is:



The Output of the Code in the terminal is :

iteration : 99 , Error : 1236.4788934830087

iteration : 98 , Error : 778.9765702986545

iteration : 97 , Error : 227.37733167007212

iteration : 96 , Error : 76.69882659377811

iteration : 95 , Error : 511.7007223220701

iteration : 94 , Error : 122.49036678806993

iteration : 93 , Error : 27.95037107547733

iteration : 92 , Error : 4.566025918029325

iteration : 91 , Error : 0.6939735524383976

iteration : 90 , Error : 0.11464610049182598

iteration : 89 , Error : 0.018762598406393063

iteration : 88 , Error : 0.0036619151395460366

iteration : 87 , Error : 0.0005933359790898357
iteration : 86 , Error : 9.177658780915741e-05
iteration : 85 , Error : 1.2848384804699918e-05
iteration : 84 , Error : 1.9906287480040147e-06
iteration : 83 , Error : 3.193006867621129e-07
iteration : 82 , Error : 4.882350309540774e-08
iteration : 81 , Error : 1.7364575101173752e-09
iteration : 80 , Error : 2.884141349539319e-10
iteration : 79 , Error : 4.95139632995627e-11

It can be clearly seen that the error values decrease very rapidly by **approximately 5 times per iteration**.