Projct Report to COMP0028

Structured Light Depth Acquisition

1 Introduction

For this project, we are expected to implement a 3D depth reconstruction algorithm based on structured.

This report will be organized as follows, I start with theoretical concept with its implementation performed on synthetic data. On the next section, I will test my implementation on provided real data to validate my program. Finally, my implementation will be used to reconstruct a mannequin's head, the head's and calibration data are captured by my team.

2 Synthetic Image sets

2.1 3D Reconstruction

The overall approach for 3d reconstruction can be seen as following pipelines: Camera calibration, (u,v) decoding, depth estimation and point cloud generation. For this particualr section, let's assume the camera is well calibrated. A detailed calibration process will be given in the next section.

2.1.1 Light Pattern Decoding

The objective of this part is to obtain $[\,u\ v\,]$ codes for each pixel. The idea for $[\,u\ v\,]$ encoding is, for an object, its pixel will be exposed to B&W strips(Grey Code) with its inverse interchangebly 10 times horizontally and vertically. Therefore, there will be 40 images in total, among them, each two forms a pair which is composite to each other. Having these will granted us to do a binary search which will be used for depth estiamtion.

In terms of implementation of decoding, first we should convert the pixel intensity into binaries. In the mean time, seperate out the background by checking whether the consecutive pixels from a pair is both 0.

In practice, binary code can also be converted to decimals directly which gives the measure of "depth". Such meaure

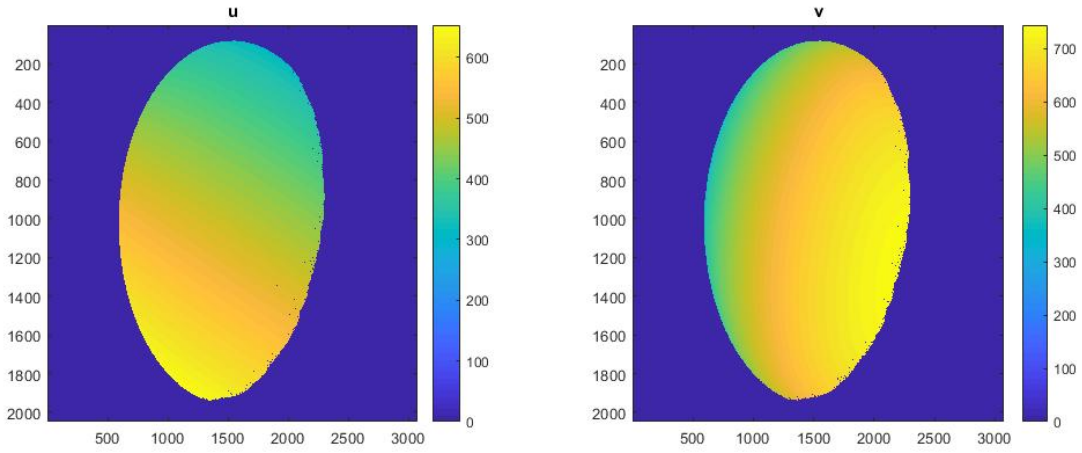


Figure 1: Decoded (u,v) for sphere_T1

Figure 25 gives the visualized results from running aforementioned decoding implementation. We can see the background has been separated out while preserving the details of sphere Although decoding is a pixel-wise operation, in practice, I found the function used for conversion between binary and decimal is fairly time-consuming which would become practically impossible for using larger sized images. For better performance, I create a 1D vetor storing all binary every time instead of doing a direct conversion. After the loop, I can just apply `bi2de` function once to get all decimals I need. By doing this, I received 10-times[1] faster running speed.

The relevant Matlab function for this part is located at `'reconstruction\find_uv.m'`.

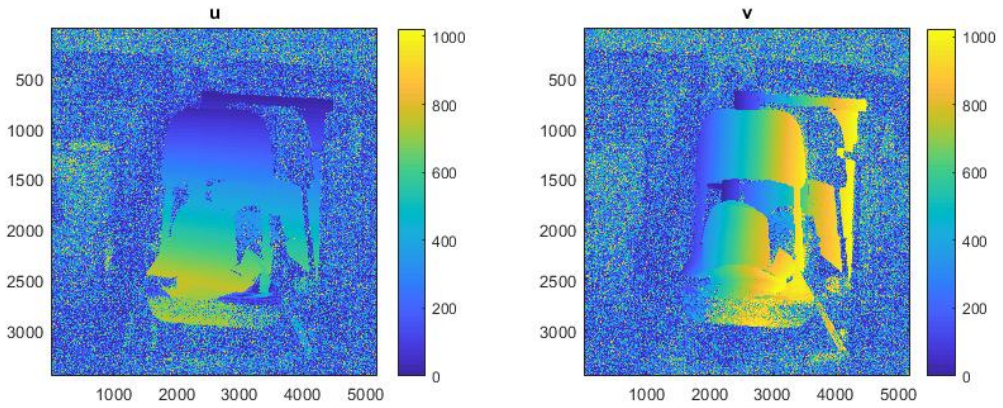
2.1.2 Remove Unreliable Pixels

In practice, scene interreflection as well as the "fog" within projector[2] would make some black and white patterns indistinguishable thus results deteriorate in decoding accuracy.

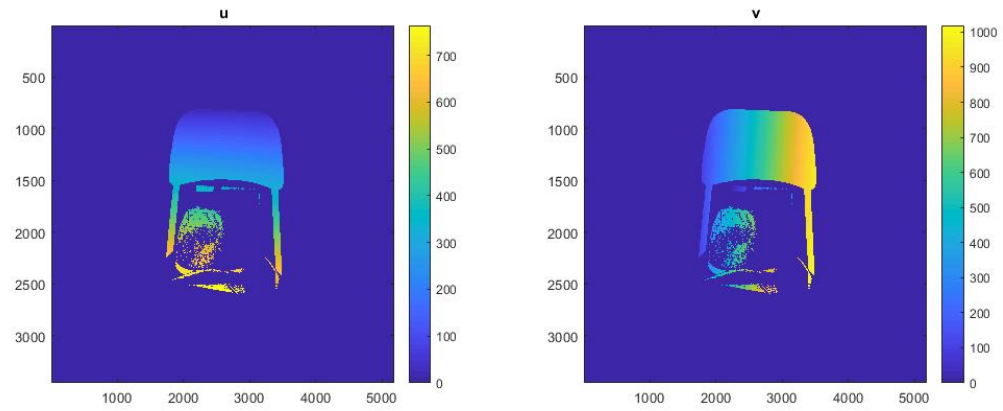
[1]Experiment data is based on running code on given synthetic data 'sphere_T1. Elapsed time from 300 seconds to 30 seconds.Processor: Intel i7-8700

Paper[2] shows one way to ameliorate deterioration which involves thresholding pixel's absolute sum difference between two exposure settings. However, we are not provided with pictures with different exposures, surely, we can increse exposure using post editing tool but that is not the paper's method meant to be.

Alternatively, I can still threshold pixels but in different aspect. During decoding process, I record the absolute difference between each code pairs. These difference is later being thresholded[2] and those pixels are set to be 0.



(a) Without thresholding



(b) Threshold = 2

Figure 2: Remove Unreliable Pixels Result

From Figure 2, we can see without thresholding, there exists huge amount of noise on

[2]Set threshold = 5 is large enough to meet project requirements. In practice, threshod is not unique due to different light condition

background. With proper thresholding, we can achieve pretty clean result as (b) shows.

The relevant Matlab function for this part can be found at `'reconstruction\unreliable_remove.m'`.

2.1.3 Depth Estimation

Recalling the camera equation from Machine Vision Lecture slides 14:

$$\lambda_i \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \begin{bmatrix} \phi_x & \gamma & \delta_x \\ 0 & \phi_y & \delta_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \omega_{11} & \omega_{12} & \omega_{13} & \tau_x \\ \omega_{21} & \omega_{22} & \omega_{23} & \tau_y \\ \omega_{31} & \omega_{32} & \omega_{33} & \tau_z \end{bmatrix} \begin{bmatrix} u_i \\ v_i \\ w_i \\ 1 \end{bmatrix}$$

The objectives of this part is to compute the value of λ_i and $[\, u_i v_i w_i \,]^T$ By multiplying the inverse of intrinsic matrix on both sides:

$$\lambda_i \begin{bmatrix} x_i' \\ y_i' \\ 1 \end{bmatrix} = \begin{bmatrix} \omega_{11} & \omega_{12} & \omega_{13} & \tau_x \\ \omega_{21} & \omega_{22} & \omega_{23} & \tau_y \\ \omega_{31} & \omega_{32} & \omega_{33} & \tau_z \end{bmatrix} \begin{bmatrix} u_i \\ v_i \\ w_i \\ 1 \end{bmatrix} \tag{1}$$

We already have $[\, x_i y_i \,]^T$ for camera and $[\, x_i y_i \,]^T$ for projection is what we computed during the $[\, u \;\; v \,]$ decoding part. Now we can use *intrinsic matrix* from `synthetic.matrices` to get $[\, x_i' \;\; y_i' \,]^T$.

Now we can re-write λ:

$$\lambda_i = \omega_{31} u_i + \omega_{32} v_i + \omega_{33} w_i + \tau_z$$

Substituting λ back:

$$\begin{bmatrix} (\omega_{31} u_i + \omega_{32} v_i + \omega_{33} w_i + \tau_z) x_i' \\ (\omega_{31} u_i + \omega_{32} v_i + \omega_{33} w_i + \tau_z) y_i' \end{bmatrix} = \begin{bmatrix} \omega_{11} & \omega_{12} & \omega_{13} & \tau_x \\ \omega_{21} & \omega_{22} & \omega_{23} & \tau_y \end{bmatrix} \begin{bmatrix} u_i \\ v_i \\ w_i \\ 1 \end{bmatrix}$$

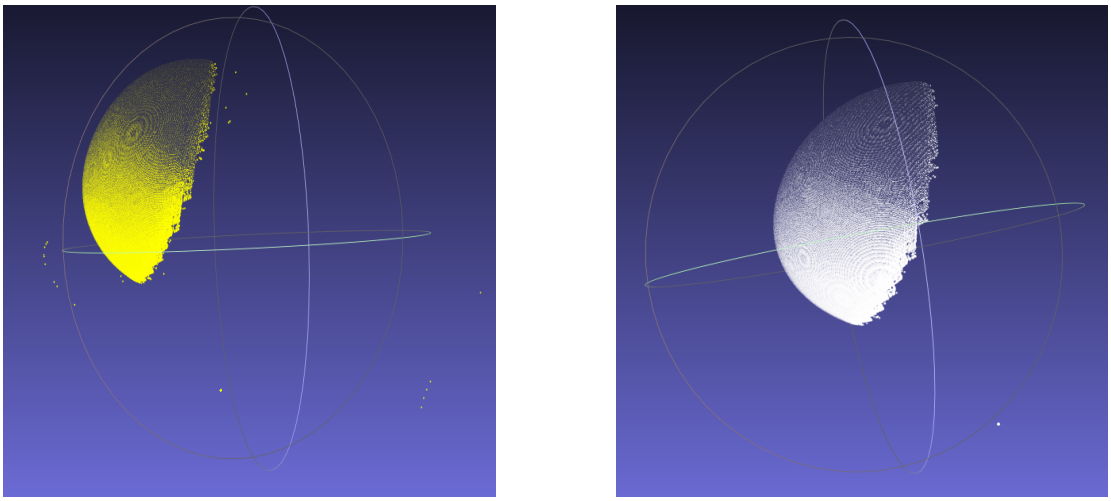
For synthetic data, we are provided with a *calibration matrices*, which gives the value of all ω and τ. At this time, we have enough data do compute $[\, u_i v_i w_i \,]^T$ which is the point cloud data. Using this point cloud data, substitute them back to equation 1 to get the depth data λ.

The above concludes every necessary steps to get depth and point cloud data. The cloud is saved as `.ply` file using function `pcwrite`. The relevant Matlab function for this part can be found at `'reconstruction\determine_depth.m'`.

2.1.4 3D Reconstruction for synthetic data set

So far we have computed $[\, u \;\; v \,]$ and using these data together with the provided calibration matrices, we are capable of getting depth map as well as the point cloud.

This part will show you the point cloud reconstruction and the visualisation of depth map for all synthetic data sets. But before we move on, there is one more thing that needs to tackled which is de-noising.



(a) Noised sphere

(b) Denoised sphere

Figure 3: Denoise result

From Figure 3(a), we see there are some unwanted points lying outside the surface of sphere which is the noise. One way to eliminate them is to use `median_filter`, so all the isolated points in the space will be suppressed by thier neighbours.

As we can see from Figure 3(b), those points are removed.

Now we have everthing ready to run reconstruction over all synthetic data sets and all results will be presented below. For each class of data set, I will give visualised depth-map then provided generated point cloud with two perspective: from depth image camera and a side view. For depth map, there will be a scale telling the relative depth from camera, smaller value stands for shallower in depth which means closer to the camera.

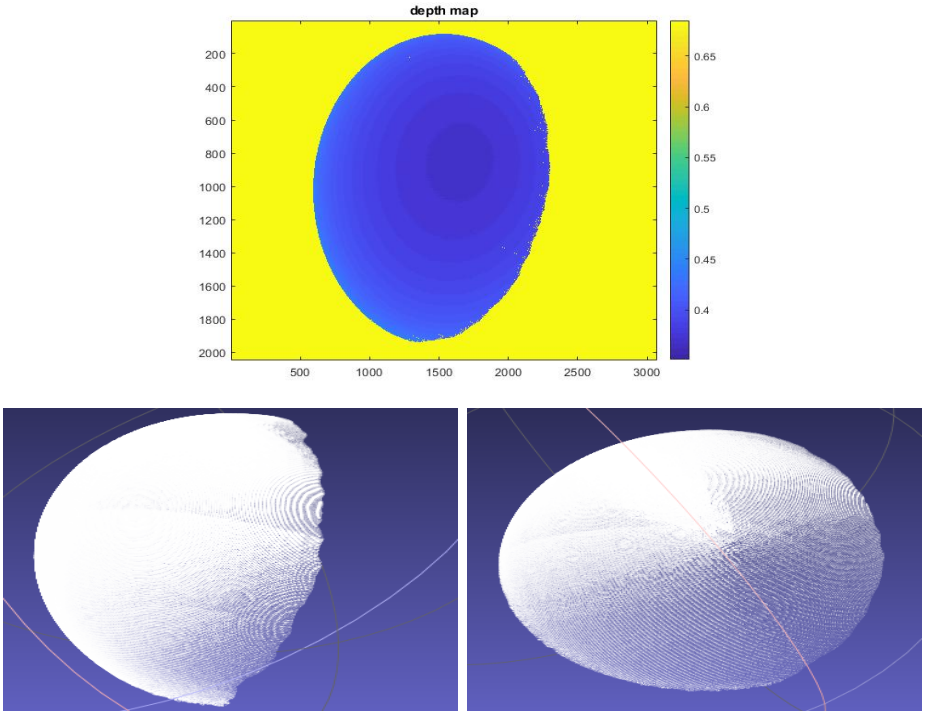


Figure 4: sphere_T1

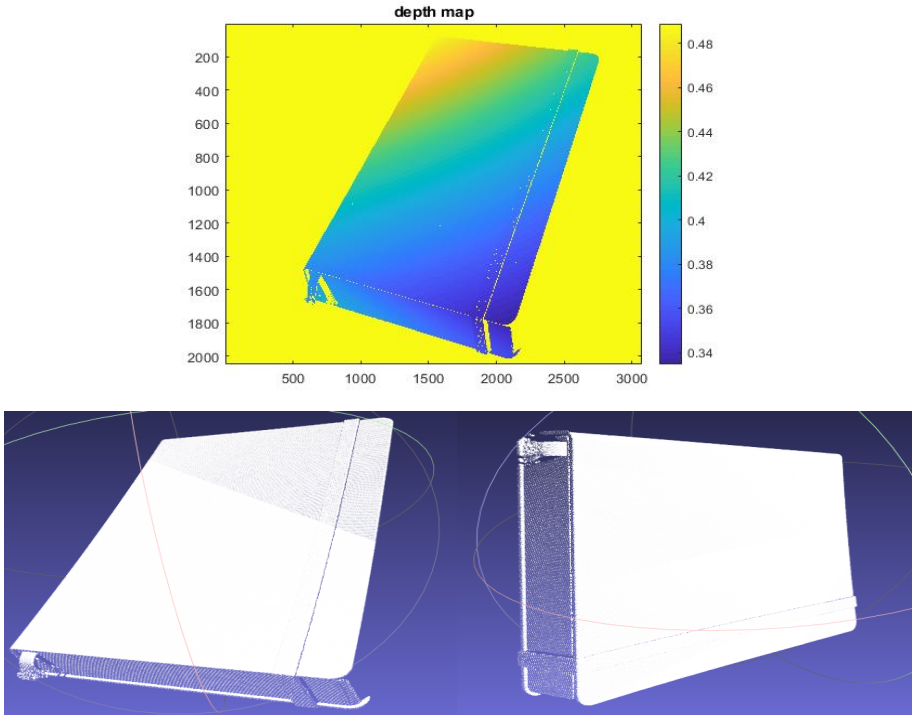


Figure 5: notebook_T1

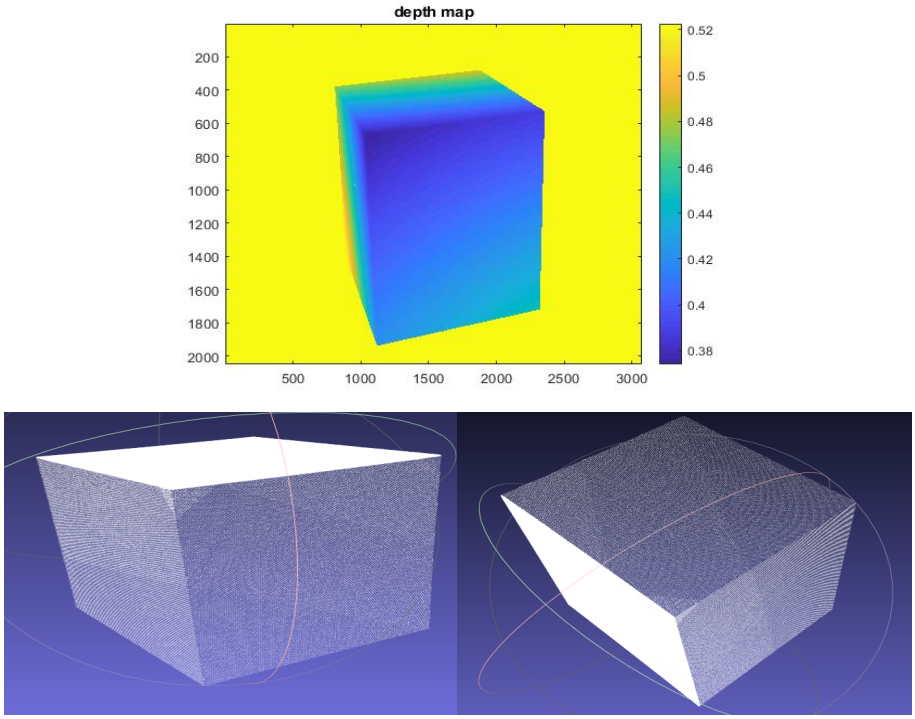


Figure 6: cube_T1

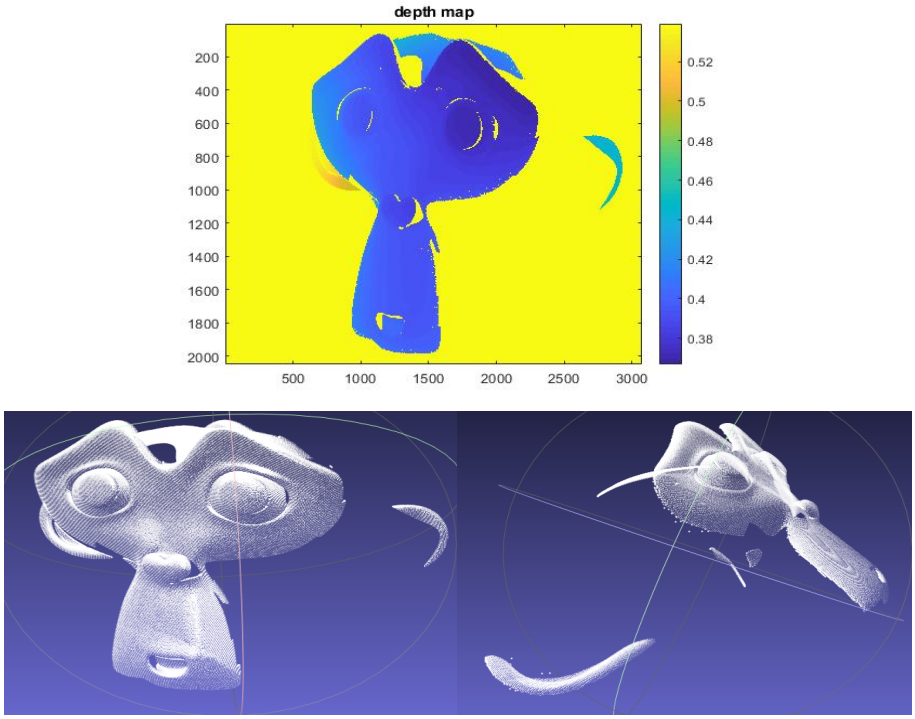


Figure 7: monkey_T1

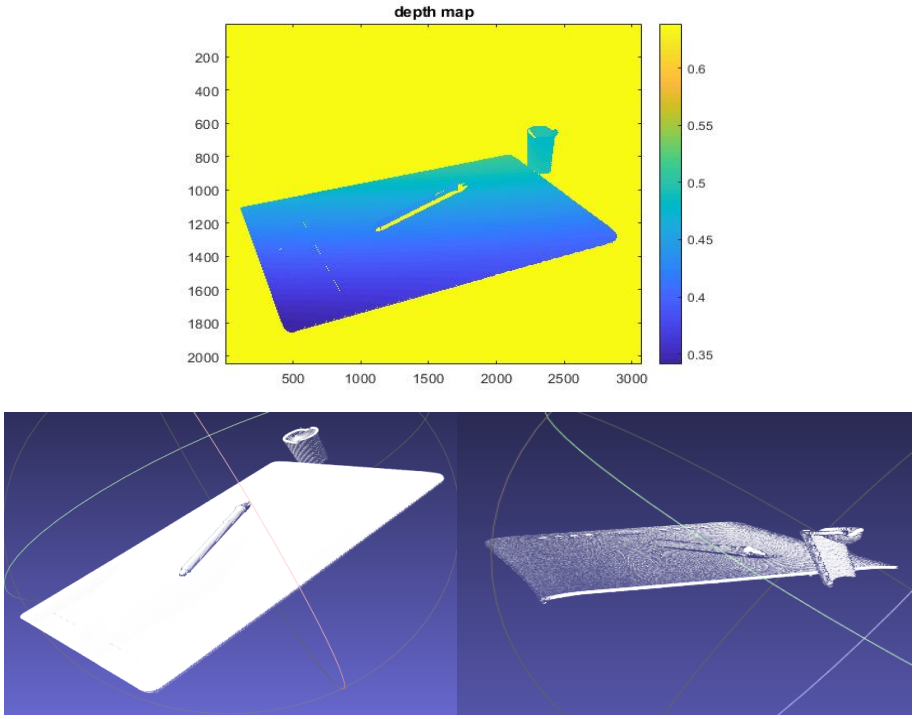


Figure 8: tablet_T1

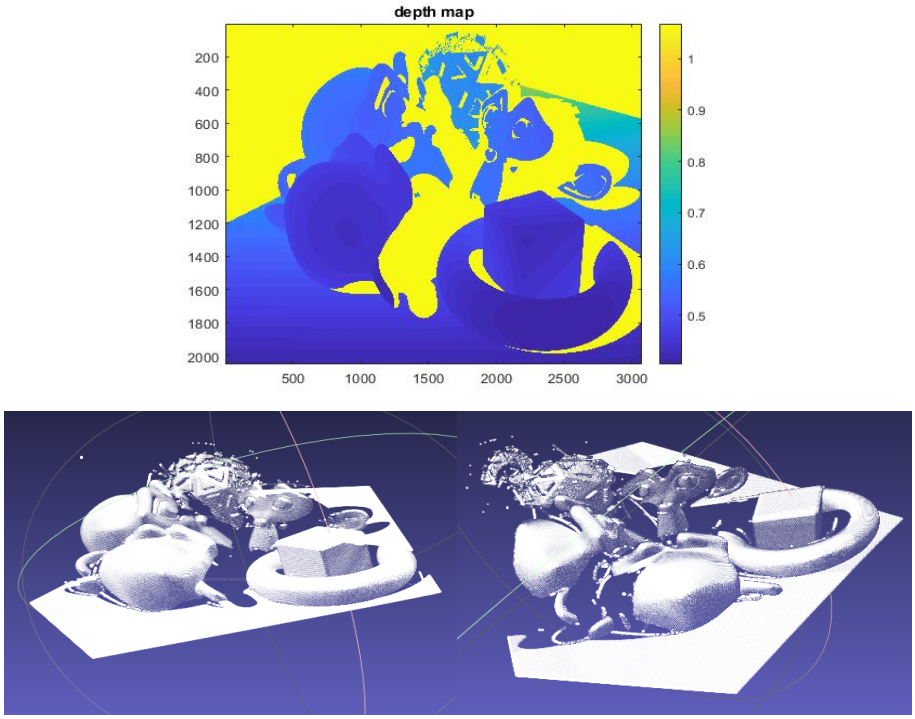


Figure 9: red_T1

2.2 Camera calibration

Camera calibration is the process of finding intrinsic and extrinsic matrices for cameras. The objectives for this section is to find those matrices for camera and projector. The approach used for camera calibration in this section is introduced by[2]. For camera calibration, we can just use the `calib_gui`, a calibration tool introduced by Lab 2 to get intrinsic and extrinsic matrices as long as we have checkerboard pictures ready. However, this tool does not apply directly to projector, because projector cannot "see" what the checkerboard looks like, thus no checkerboard image for it to calibrate. Therefore, we need to find a way to get the image that the project may see if it is a camera.

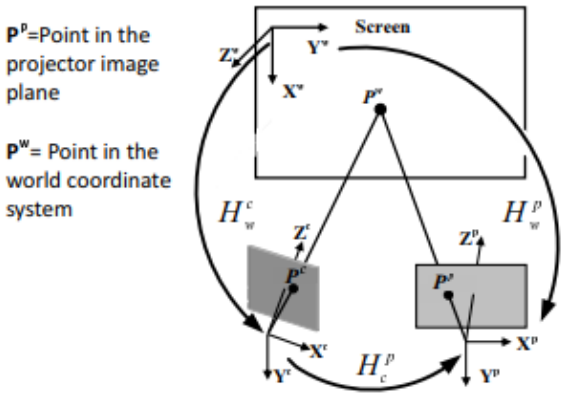


Figure 10: Camera, projector relationship

Figure 10[3] gives the relationship between camera and projector. So we need to find H_c^p first which is *homography*, we then use this H to project image which projected by projector to the projector's image plane coordinate system P^p. This P^p can be used for calibration of our projector.

2.2.1 Reprojection for projector

For synthetic data, we are provided with pattern that will be used for projection to the board $P^p_{projected}$, images from camera which captures projection $P^c_{projected}$ and printed checkerboard image captured by camera $P^c_{printed}$. We can get relationship between these

[3]Picture credit:[2]

terms from [2]:

$$P^p_{projected} = H^p_c P^c_{projected}$$

H^p_c is the *homography* we are aiming to find for this part.

We can then use H^p_c to reproject printed checkerboard into a virtual pattern$P^p_{printed}$ used for projector calibration[2]:

$$P^p_{printed} = H^p_c P^c_{printed}$$

The pre-requisition for computing *homography* is two sets of points. One idea for this approach is to have a set of points from fronto-parallel position and a set of points from other arbitrary perspective. We define those points to be four vertices from checkerboard's edge.

The implementation for learning homography parameters comes from Machine Vision Lecture slides 15:

$$\begin{bmatrix} 0 & 0 & 0 & -u_1 & -v_1 & -1 & y_1u_1 & y_1v_1 & y_1 \\ u_1 & v_1 & 1 & 0 & 0 & 0 & -x_1u_1 & -x_1v_1 & -x_1 \\ 0 & 0 & 0 & -u_2 & -v_2 & -1 & y_2u_2 & y_2v_2 & y_2 \\ u_2 & v_2 & 1 & 0 & 0 & 0 & -x_2u_2 & -x_2v_2 & -x_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & -u_I & -v_I & -1 & y_Iu_I & y_Iv_I & y_I \\ u_I & v_I & 1 & 0 & 0 & 0 & -x_Iu_I & -x_Iv_I & -x_I \end{bmatrix} \begin{bmatrix} \phi_{11} \\ \phi_{12} \\ \phi_{13} \\ \phi_{21} \\ \phi_{22} \\ \phi_{23} \\ \phi_{31} \\ \phi_{32} \\ \phi_{33} \end{bmatrix} = 0$$

Where $[\,X_I \;\; Y_I\,]^T$ and $[\,U_I \;\; V_I\,]^T$ are two sets of points in homogeneous coordinate. For synthetic data, we are given the fronto-parallel points $[\,X_I \;\; Y_I\,]^T$, so I create a GUI for user to manualy selects the rest set of points $[\,U_I \;\; V_I\,]^T$. Above system can be in the form $\mathbf{A}\phi = 0$, we can use SVD $\mathbf{A} = \mathbf{U\Sigma V^T}$ to find Φ which is the last column of $\mathbf{V}$.

In Matlab, once we computed the *homography* we can apply such transformation directly using function `projective2d` and `imwarp`:

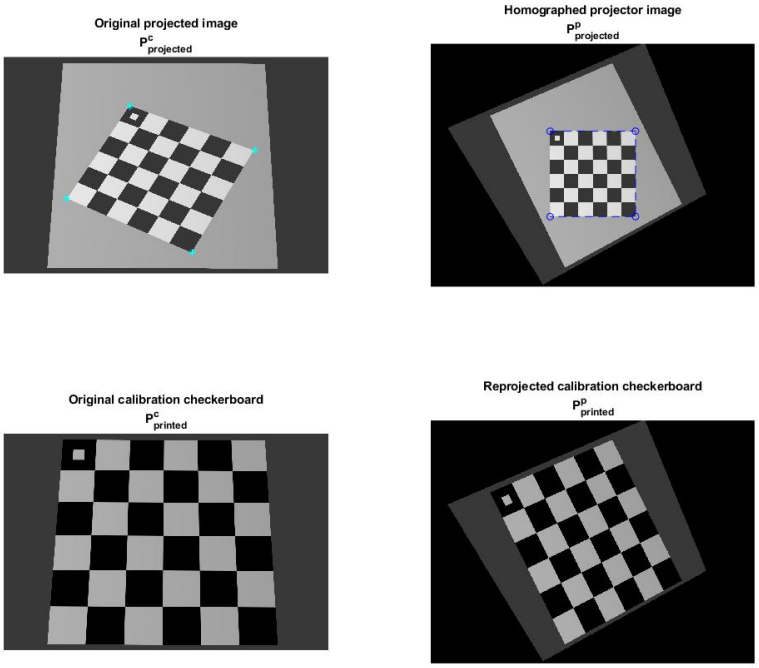


Figure 11: Reprojection results

Figure 11 shows the result of reprojection process. From the figure, cyan star represents the user defined vertices and blue oval represents fronto-parallel vertices which given to us. The relevant implementation for this part is located at `'calibration\get_bestHomography.m'` and `tilt_back.m`.

2.2.2 Intrinsic and Extrinsic Matrices Estimation

Thanks to camera calibration toolbox `calib`[4] By providing toolbox with following image sets:

[4] http://www.vision.caltech.edu/bouguetj/calib_doc/index.html#functions

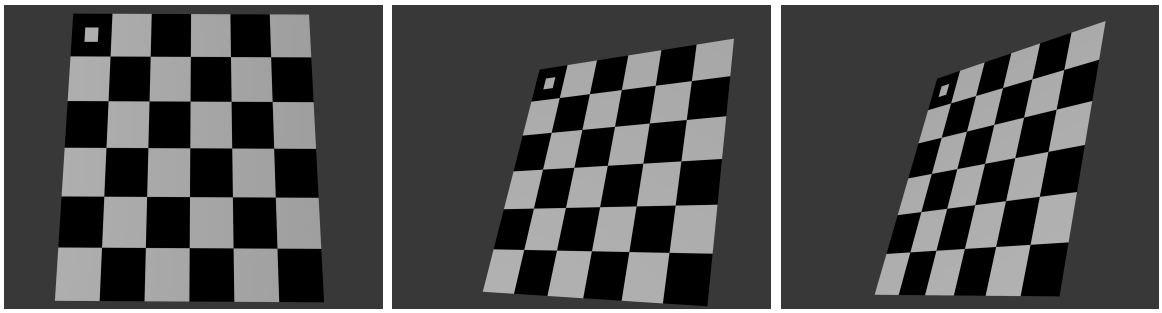


Figure 12: Calibration sets for camera

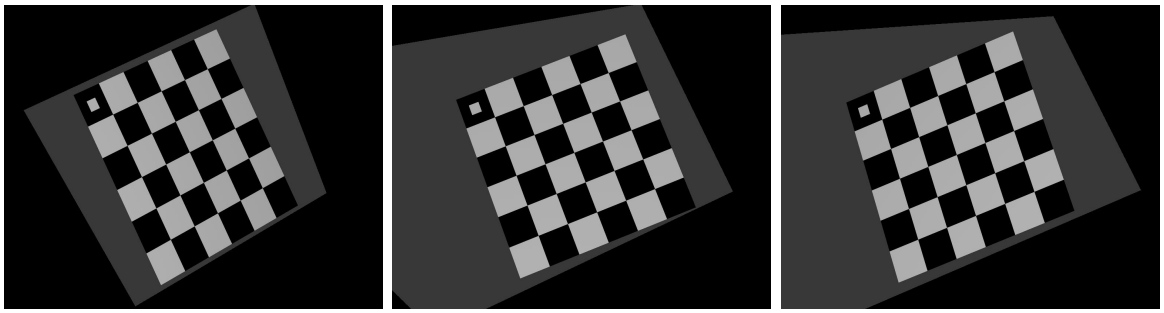
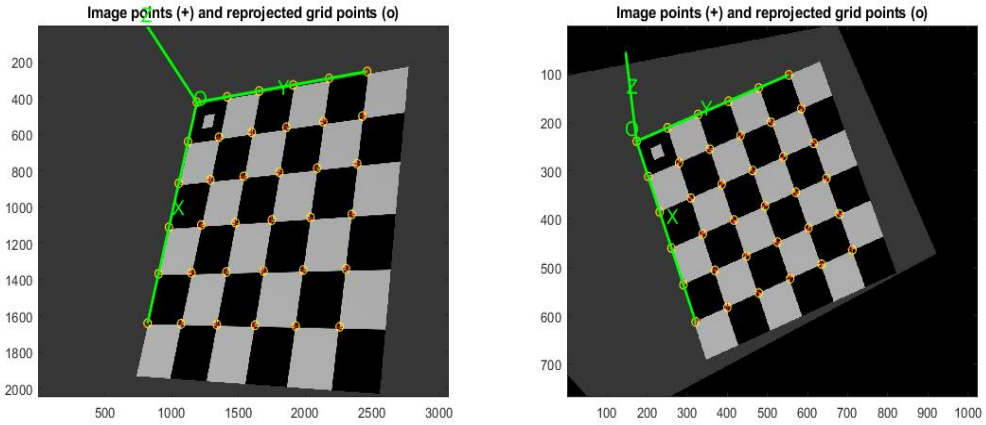


Figure 13: Calibration sets for projector

The process of finding intrinsic matrix starts with loading images then use function `Extract Grid Corners` followed by `Calibration`.



(a) Calibration for Camera

(b) Calibration for Projector

Figure 14: Extrinsic calibration

As we can see from Figure 14, our calibration tool successfully estimate the extrinsic transformation by detecting image plane's coordinate system. During the calibration, it is necessary to input the value correctly as the prompts required. I use $dX, dY = 0.025mm$ to be the size of each equare and I also use 5 by 5 checkerboard for calibration since toolbox works only odd number in checkerboard size.

2.2.3 Calibrated Point Cloud Reconstruction

From section 2.1.4 we use provided camera matrices for point cloud reconstruction. For this section 2.2.3, I will run through the similar reconstruction process but with the calibrated camera matrices we gatherd from 2.2.2.

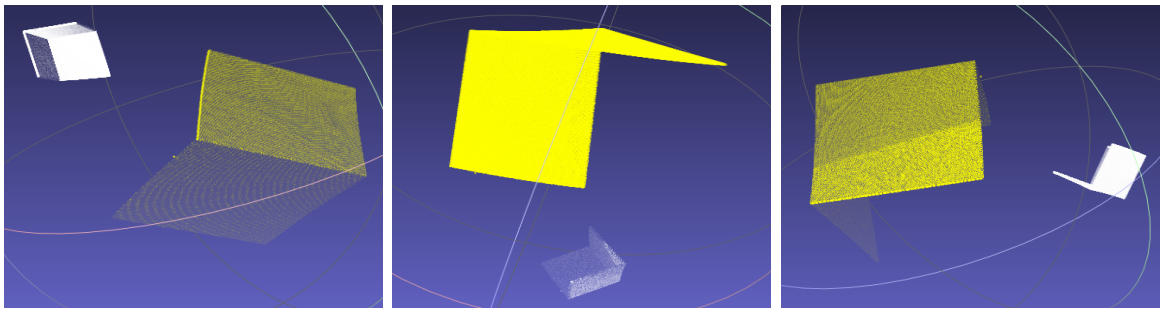


Figure 15: Reconstruction results from GT camera and calibrated camera

From Figure 15 we see two cloud points is not properly aligned. The main cause for this is because the extrinsic and intrinsic matrices is not computed correcly. I reviewed my `homography` results which shows correct. Then I realized it is the improper use for the tool box. Since we have ground truth, I compare my results with GT, the intrinsic is really close to the given one but extrinsic is completely different. When using toolbox, we are expected to input size of checkerboard as well as the size of square. These two terms really confuse me since I don't know the size exactly. When looking at extrinsic calibration results 14 we see a perfect world coordinate estimation. So I belive the main cause for such point cloud deviation is due to those unknown terms.

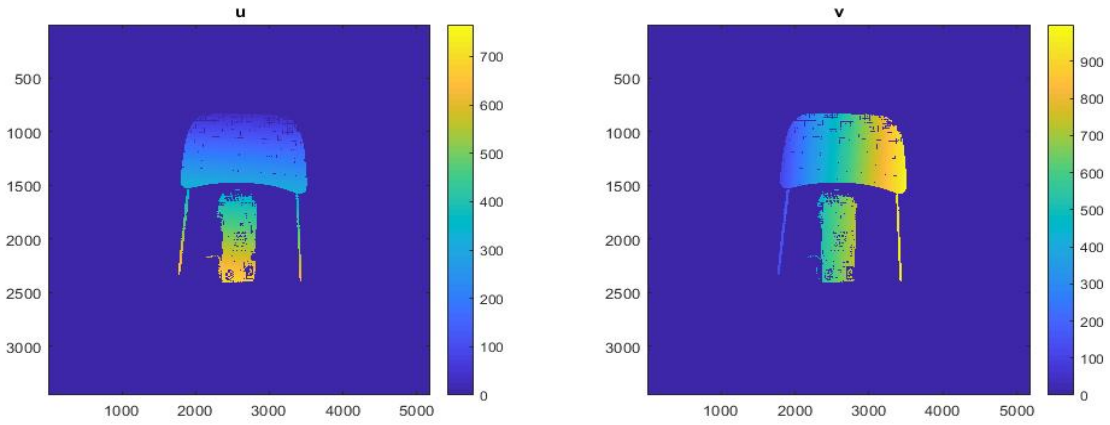
Other factors like improperly select edges or numerical error from inverse matrix in homography part contribute comparatively less error to extrinsic matrices.

3 Real Image Sets Reconstruction

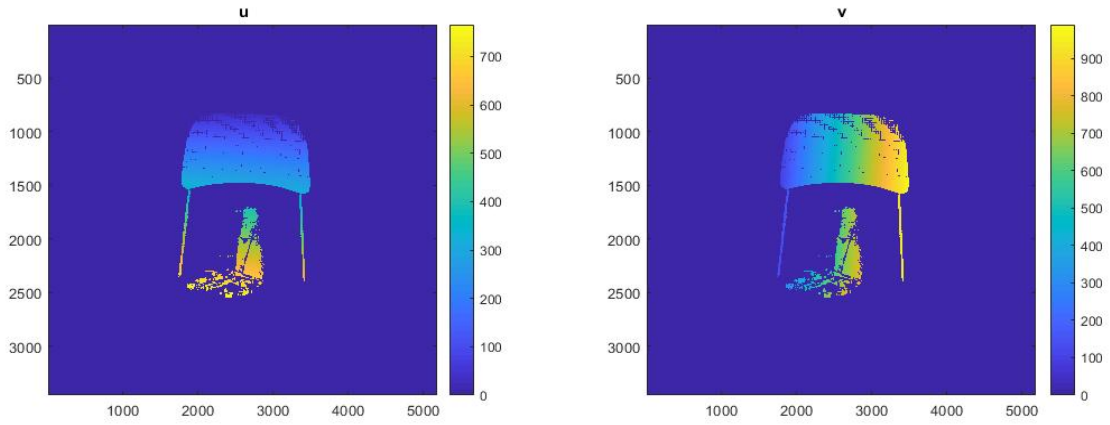
For this section, I will run through calibration and reconstruction again for provided real image data sets: real_tea and real_cryan_dalek.

3.1 Light Pattern Decode

In section 25, I introduced how I approach to the decoding procedure including background separation as well as the unreliable pixel removal. Thus, at this point, I acquired enough gounding for decoding real images.



(a) real_tea



(b) textsfreal_cryan_dalek

Figure 16: Decoded u,v for real_tea and real_cryan_dalek

3.2 Real Camera and Projector Calibration

I have demonstrated the procedure and theory background for camera and projector calibration.

After I applied *homography* to each calibration data, I realized part of printed checkerboard are outside the frame as demonstrated in Figure 17. Therefore, during the calibration process, I use the lower right 3 by 3 corners. The *homography* output can be seen as the below figure.

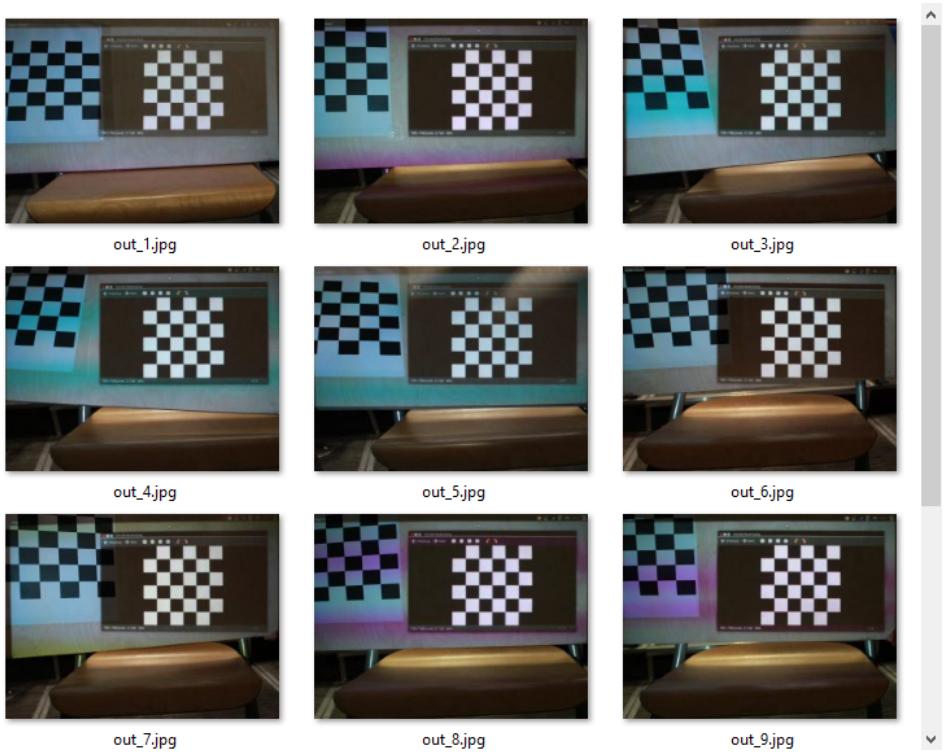


Figure 17: Decoded (u,v) for sphere_T1

Here I provided calibration toolbox results for second pair of images:

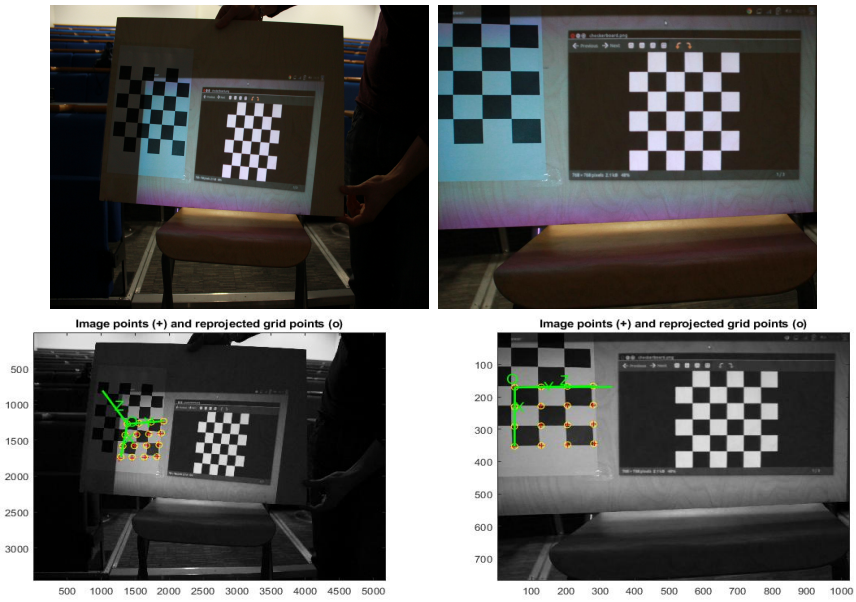


Figure 18: Real camera, projector calibration

The calibration result is placed at `'calibration\output\output_real_calibration'`.

3.3 Depth-map Estimation

For real data, it is more challengeable to estimate depth from its noisy background, in this situation, my solution is to set threshold to be 5, and I get following depth maps:

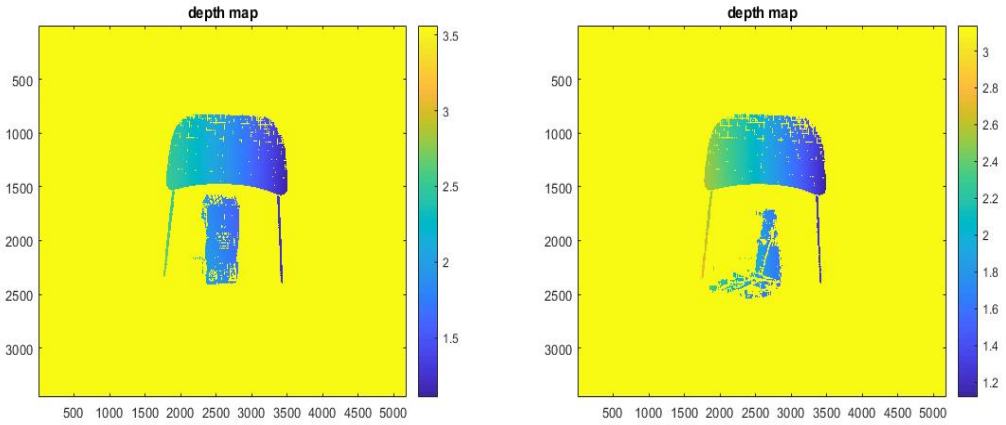


Figure 19: Depth-map for real_tea and real_cryan_dalek

Figure 19 shows my depth estimation for real image works well with the given threshold

value, we can see the objects is not blend into background. At this point, I can bravely predict the generated point cloud will works as expected.

3.4 Point Cloud Reconstruction for Real Image

The presentation for Point Cloud results is similar to what I have done for synthetic data where I will give one look with depth-map perspective followed by two alternative side looks.

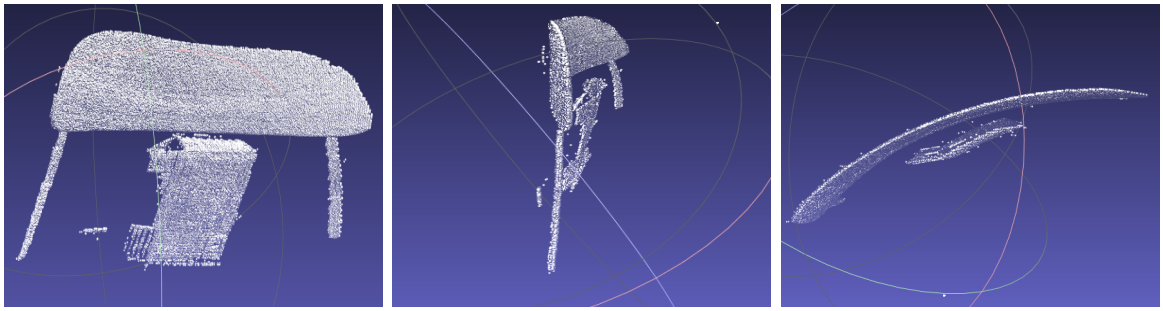


Figure 20: Reconstruction results for real_tea

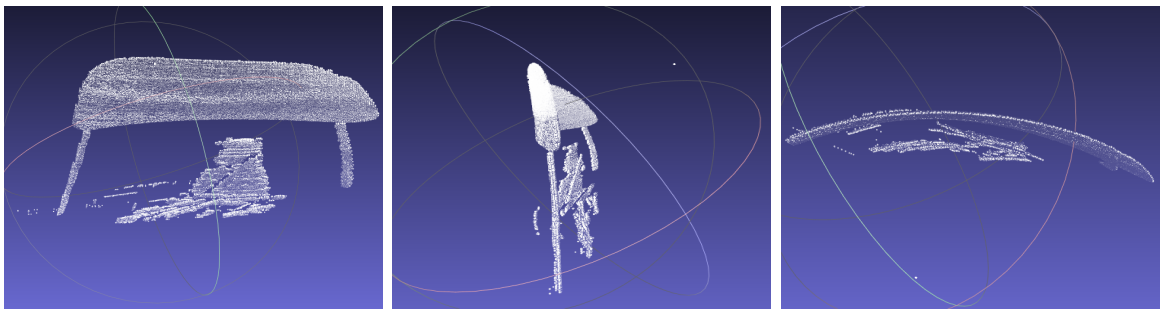


Figure 21: Reconstruction results for real_cryan_dalek

As the results showed, these clouds is not heavily noised and we can clearly see the shape of two objects. One downside of my results is the distortion, distortion correction comes from intrinsic matrix which may suggests my calibration is not done perfectly.

4 Own Image Sets Reconstruction

In this section, me Jialin Yu in collaboration with Lukai Gong and Zakeria Ali captured our own data for reconstruction. The equipment used is my own camera Canon 5D mark

III with EF 24-105 f4 L IS lens.

Our capture process comes with follows, we first takes several images of a wooden board with attached printed checkerboard and projected checkerboard on it. For each shoot, we rotate the board in a certain angle, the gathered data are for calibration purpose. Next, we placed a mannequin head in front of the board and project grey code to it. This process gives image set we can use for depth mapping.

My camera comes with 24MP, due to upload limitation I downsampled images[5].

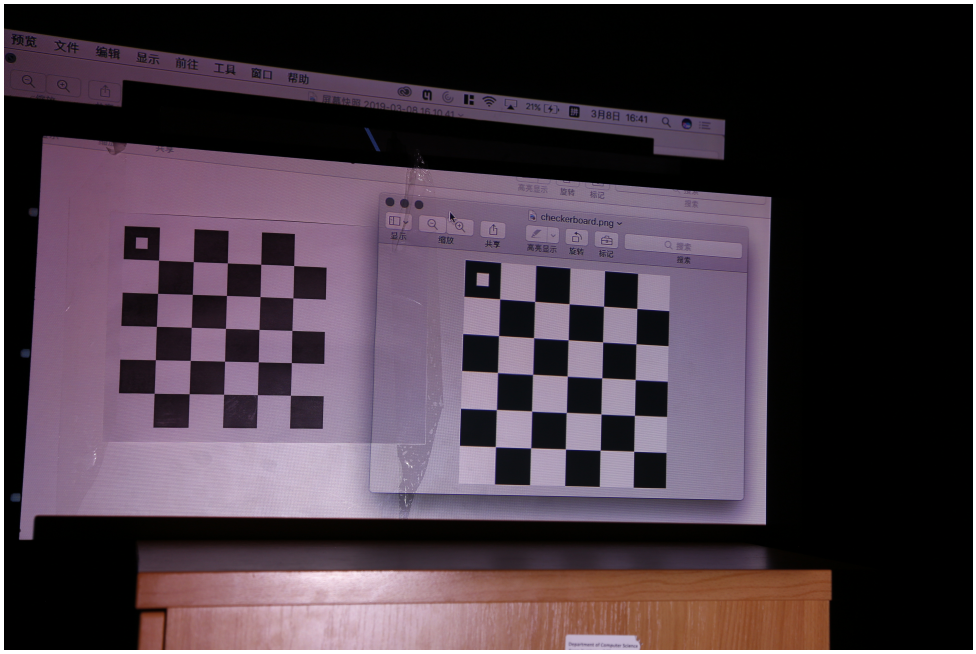


Figure 22: Well exposure set up demonstration

Figure 22 gives the photography light condition while we capturing data sets. To get proper exposure, we did the following things, we turn off the light to get rid off ambient light and make projector the only light source in the room. We use a pure white image found online to light up printed checkerboard which greatly reduce the contrast between printed and projected checkerboard. At this point, my camera's light metering tells me everything is readly for proper exposure. In addition to that, my photography experience tells me to get perfect sharpness of image I have to pick low ISO, otherwise the image may fail to show the pattern for thinnest grey stripes. I also use smaller aperture size

[5]Full resolution data sets can be found here: https://drive.google.com/file/d/1dAVbrSB9SJrZPIsj3Q0jqAV00OSulxlw/view

instead of wide open, this is because lens always performs best if you step down the aperture few stops, this is evidenced by my lens' MTF curve. Finally, I try to composite object into centre of frame so less distortion, color fringing and vignetting effects. Here is my finalized exposure settings : aperture f5, shutter speed 1/60 second, ISO 400 and focal length 45mm. The object data is placed at `'reconstruction\head_own'`. And calibration data can be found at `'calibration\our_calibration'`. Like what I did in previous section, I first decoded our object image:

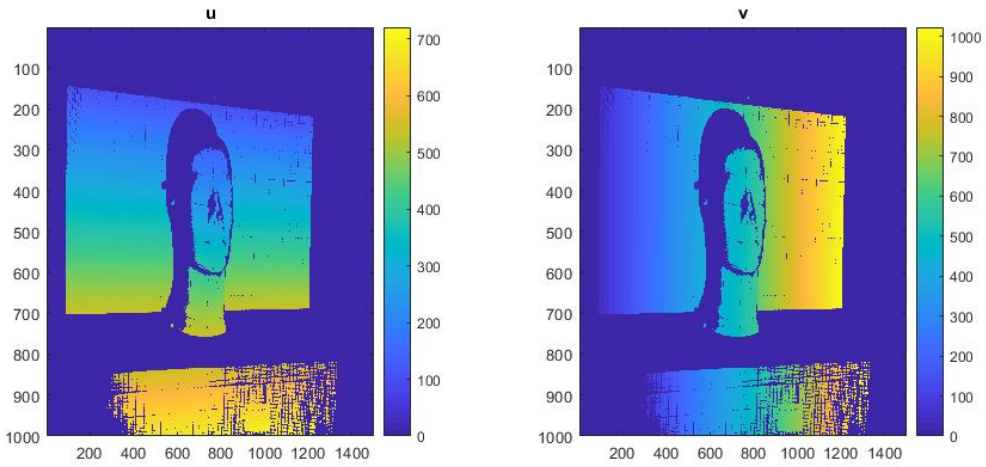


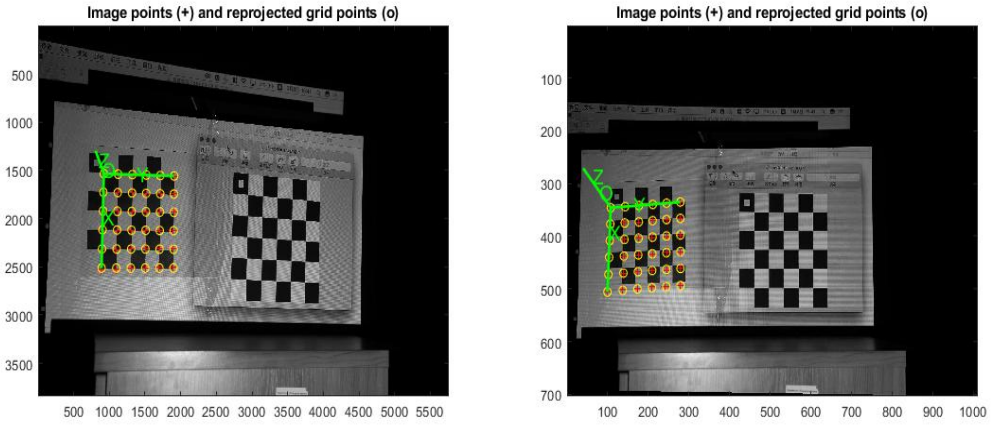
Figure 23: Decoded u,v visualisation

Which looks ok so proceed to the next part.

I do the calibration process to get projection matrices I need for depth estimation.

For calibration image, the board is located at $[2746\ \ 1516]$[6] and the board size is 216 pixels per rectangle.

[6]I use the full resolution for finding homography



(a) Calibration camera

(b) Calibration projector

Figure 24: Calibration result

Above gives the result for calibrating our camera and projector. One noticable thing to is that project calibration is not perfectly aligned, this might due to I did not properly select edge points. The calibration result is placed at `'calibration\output\output_our_calibration'`. With calibration matrices, I can get my depth map:

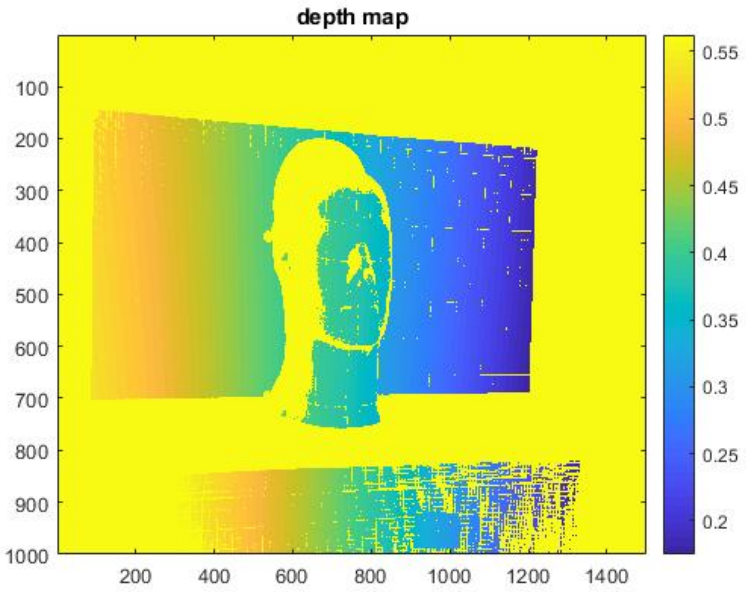


Figure 25: Depth map for our data

Finally, the generated point cloud:

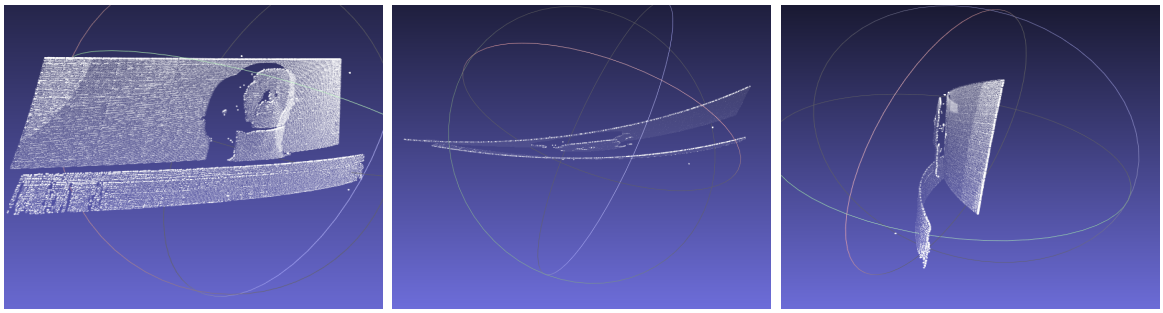


Figure 26: Reconstruction results for our captured data

From the results, we can observe a bar behind the head. In reality it is the desk where we placed our model head.

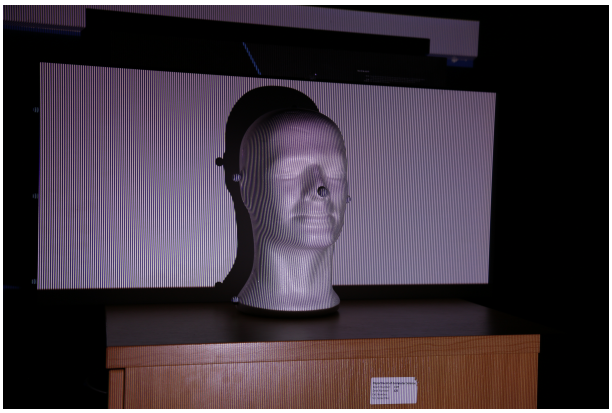


Figure 27: Object capture environment

From above figure, we can see the desk especially left corner has a highlight. This is due the the high specular reflecitve luckily it does not affect our result. From the results we can see, it successfully distinguishes backpanel, head, and desk based on thier depth information which is desired. However, we can still notice the distortion for the point cloud. This is mainly because the calibration error.

In practice, although using a branded tripod and optical image stablisation from lens , we still observeed the image shaking. This is caused by some of my teammate accidently kicked the tripod. Further improvements could be made by turning of camera's mirror lock which will reduce the camera shaking from mirror flipping thus further stablised images.

5 Conclusion

For this project, we are assigned to get depth information based on structured light. I achieved this implementation by completing all aforementioned tasks including (u,v) decoding, camera calibration, homography. I run through all these implementation over provided data as well as our own data. In the end, we successfully reconstructed a mannequin' head based on images taken by ourselves.

References

[1] Daniel Scharstein Richard Szeliski *High-Accuracy Stereo Depth Maps Using Structured Light.*

[2] Hafeez Anwar Irfanud Din Kang Park *Projector calibration for 3D scanning using virtual target images.*