Final Report

I. Introduction (*3D-Modelling*)

The goal for this project was to build a high-quality 3D model of one of the default objects using the pipeline we outlined in class. The object I chose for this project was a teapot (shown below). I had a total of 39 images per camera per each grab folder as well as 4 color images for each folder. The data sets used in this project were provided by Professor Charless Fowlkes, with frame C0 being the right camera and frame C1 being the left camera in the image file names.

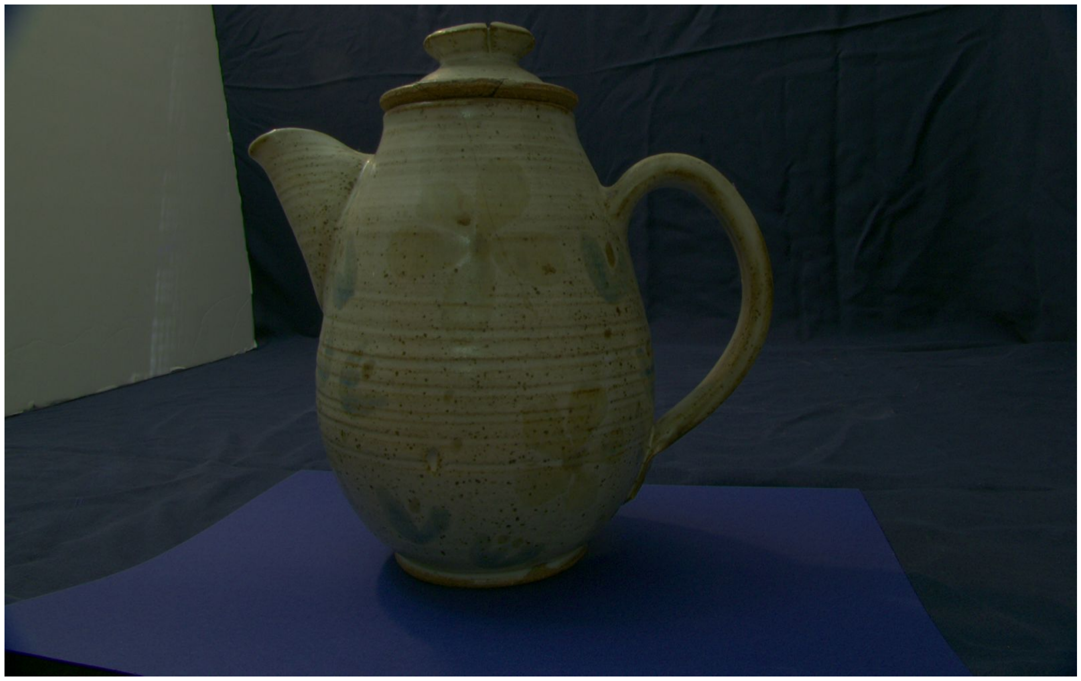


Fig. 1. Teapot object I chose for this project.

The problems I identified to solve when starting this project were, namely, mesh cleanup and smoothing, mesh alignment, combining meshes into a final model and rendering the final model and adding color . Before, I move on I shall first define these many terms.

Firstly, a 3D model of an object is simply a representation of a real world object in the 3-dimensional world of a computer. Having such models proves to be useful in areas such as robotics, fashion and computer graphics. In order to render a 3d model, one must identify a 3d object to model and then create a software pipeline to process the object with a computer. There are many steps involved in a pipeline. The most fundamental part of my pipeline, involves a concept called triangulation in the Computer Science world.

Triangulation takes inspiration from the human eye in order to figure out where points in 3D are from given 2D points. The human eye makes use of a left and right retina to see the convergence of a point from two different perspectives. In a similar manner, triangulation involves making use of two cameras (left and right) to find the convergence of a point.

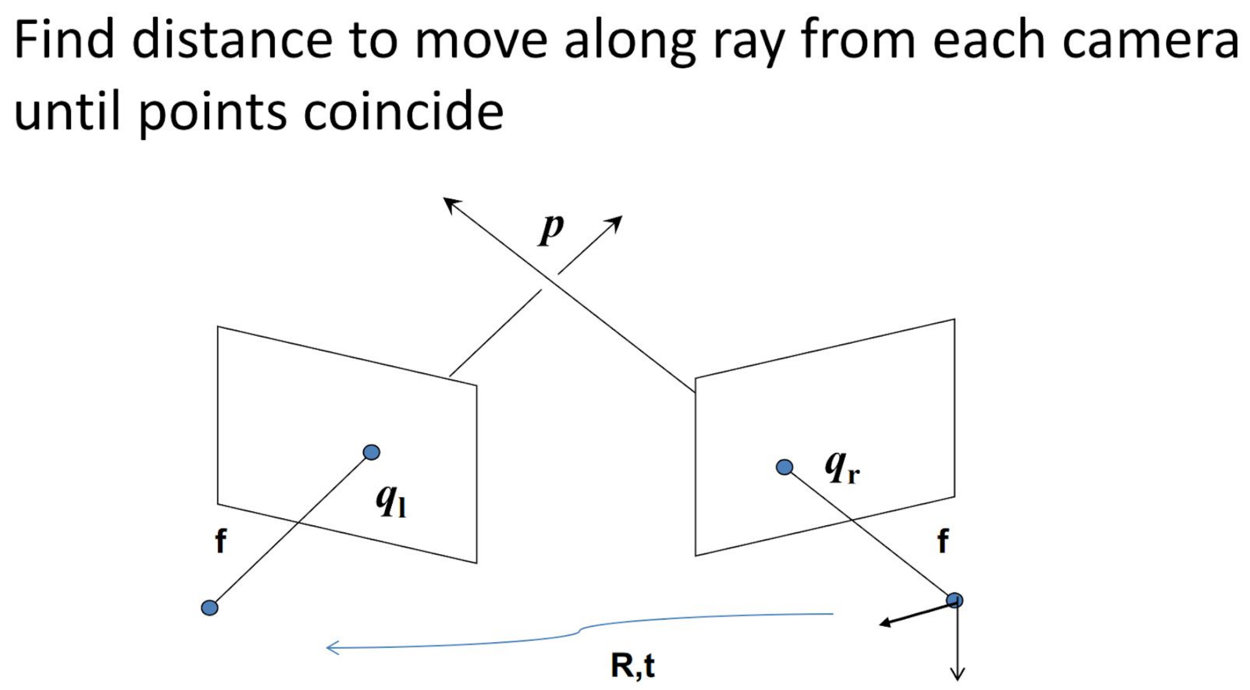

Find distance to move along ray from each camera until points coincide
p
q_l
q_r
f
f
R,t

Fig. 2. Triangulation diagram showing the two cameras and their parameters.

Using this principle, we can take images from left and right and create a function that maps the points of an object located at p. This is most fundamental part of this 3d modelling project. Moving forward, I now had a way to map points to an object in 3D but we need to ensure that our computer has the camera's basic parameters (focal length, center, rotation and translation vector) encoded in it. When creating a 3D model of an object, computer visionists use these parameters to generate what they call a mesh (a skeleton of the object), which they must refine using techniques such as smoothing and cleaning up irrelevant points. Multiple meshes are generated with pictures being taken from different angles of the object. If the implementation of modelling chooses to, the meshes may be cleaned up with a smoothing algorithm or some other clean up algorithm. Smoothing involves removing noise from flat surfaces, by moving points closer together. One such implementation of this algorithm involves taking the weighted average of a point's neighbours and setting its value to that average.

Finally, all the meshes are put together using an algorithm involving user input and then color is added using a reconstruction of the surface and the nearest point's color. The algorithm goes as follows:

1. Have the user select matching points in the two images of the object

2. Find the closest 2D point to each user click (i.e. in xL or xR)
3. Use the corresponding 3D points to estimate the 3D alignment
4. Find closest point in surface reconstruction and set color to that.

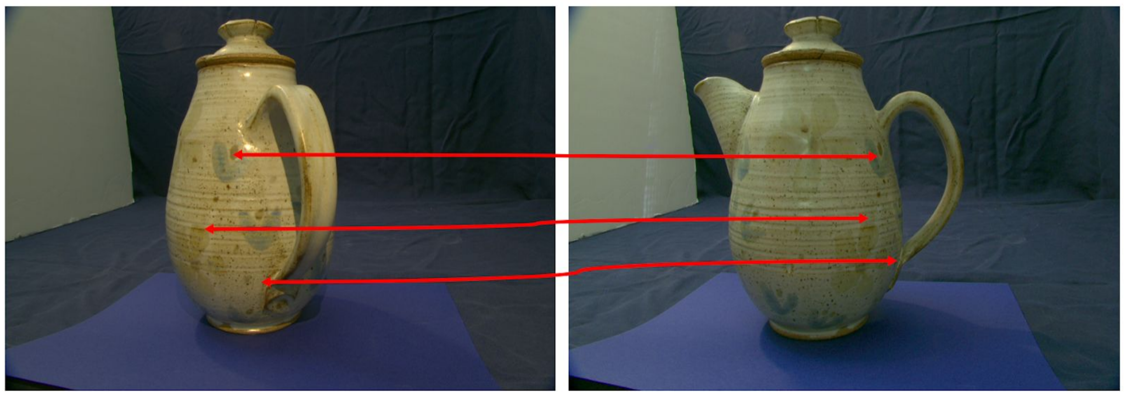


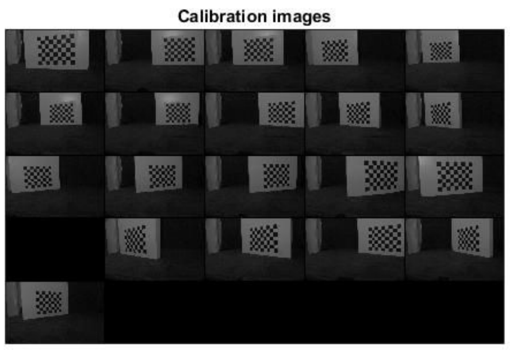
Fig. 3. Mesh alignment with user input clicks.

I initially planned to use the Iterative Closest Point (ICP) algorithm with an open source software MeshLab to construct the final mesh, however, I ran into some issues with my implementation as I will soon discuss.

II. Implementation (*MATLAB Pipeline*)

This section will describe the implementation of the pipeline I mentioned above, where I began to calibrate the cameras used in my teapot images for my computer. To do this, I extracted camera calibration parameters using the MATLAB Camera Calibration tool provided to us. I followed instructions from the website to obtain left (camL) and right (camR) camera parameters. Below are some of the results from my calibration, as well as, the images used to calibrate the cameras.

Fig. 4. Images used to calibrate the cameras in my program.



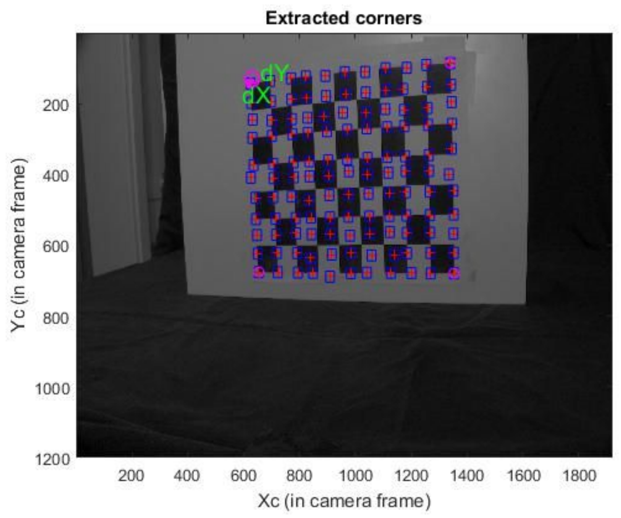
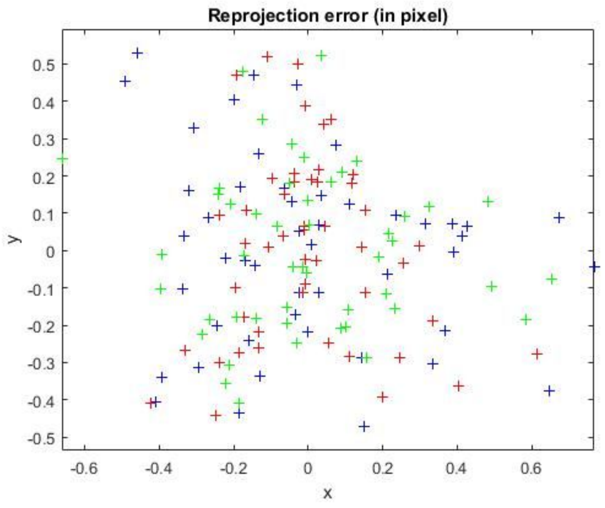


Fig. 5. Process of extracting the corner grids of the coordinate system while calibrating the camera.

For each camera, I manually clicked on the corner points of the grid and then MATLAB's camera calibration tool optimized all of the camera parameters for me. I then saved these parameters to respective camera left and right '.mat' files, parsed them for relevant information and generated a script to load their parameters into objects camL and camR. The below image shows some of the error involved in reprojection of the camera images.

Fig. 6. Error correlation generated while calibrating cameras.



After calibrating the camera's, I worked on my code from Assignment 4 (mesh.m and reconstruct.m) and enabled them to work on multiple directories. In reconstruct.m, I iterated over each grab directory and decoded each one individually in hopes of creating 6 reconstruction MATLAB files. Unfortunately, in my implementation I ran into an indexing error which I was unable to resolve even after debugging for quite some time. However, I did receive some output after the first few reconstructions.

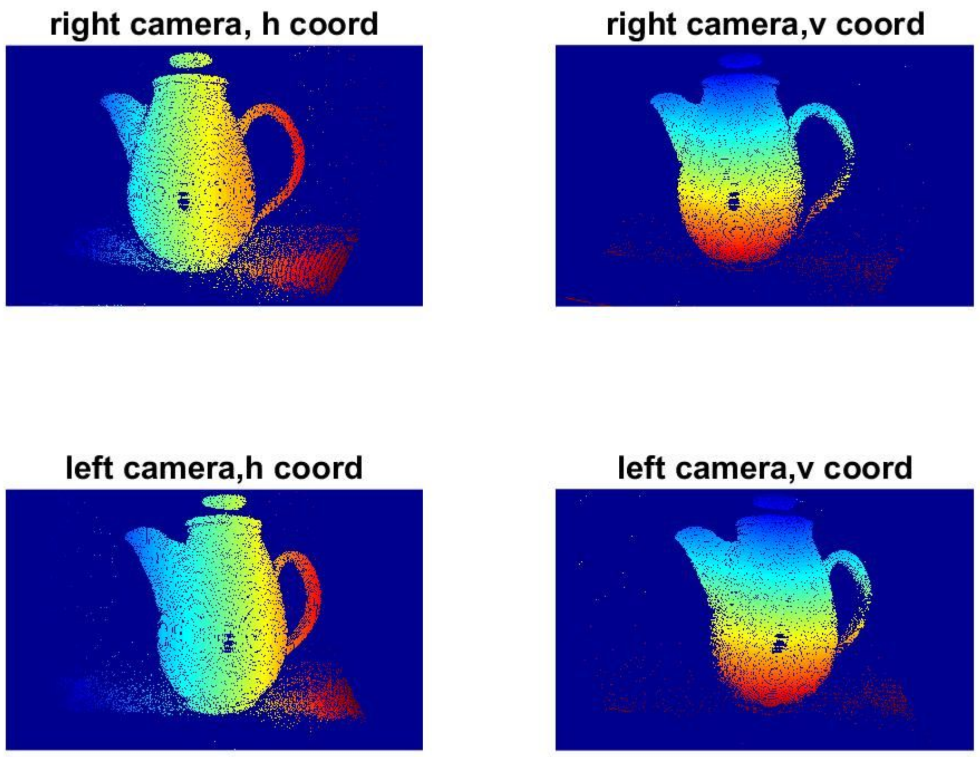


Fig. 7. Reconstruct output from grab_0

The above image shows the output after iterating over grab_0 in reconstruct.m. As you can see, this grab has been decoded correctly which leads me to suspect that the issue is not in my decoding script. This is the error I received while running my project demo--

```
Subscripted assignment dimension mismatch.
Error in reconstruct (line 65)
  xR(1,:) = xx(R_sub_matched); % pull out the x,y coordinates of the matched pixels
```

I did change this script from Assignment 4, in that I allowed it to process multiple directories and I created a new mask for background subtraction. This is the following code I added--

```
 % subtract background from color image for right and left cameras
  R_color_mask = (abs(R_rgb - R_rgb_bg).^2) > thresh;
  L_color_mask = (abs(L_rgb - L_rgb_bg).^2) > thresh;


  %
  % combine horizontal and vertical codes and color masks
  % into a single code and a single mask.
  %
  L_good = L_color_mask & L_h_good & L_v_good; %binary masks for pixels that have both
good horiztonal and vertical codes
  R_good = R_color_mask & R_h_good & R_v_good;
```

I believe the error must be somewhere in this code, but I am not at all sure. I followed the provided algorithm for background subtraction exactly as it was shown in the lecture slides. I tried several things to resolve this issue. My guess was that since the error was a dimension mismatch, I was receiving too many or little matches from R_sub_matched after adding the color mask. I coded in some if statements that checked for the size of this matrix and truncated it to the size of xL(1,:), however, this did not resolve the issue. I have included the reconstruction matrices I generated using this script in my data file, in case the issue is with the matrix itself. Online forums were unable to help me resolve my issue. I tried a suggestion to add the 3rd channel of the RGB mask before AND'ing it with the other masks. I also tried converted the images to gray images and changing their precision to double, but was unable to be successful in creating all 7 meshes without error.

III. Conclusion

Seeing that I was not making any progress, I tried my best to move on and change the code in mesh.m while hoping that the matrices I generated from reconstruct would help me obtain some meshes. Unfortunately, they appeared to be in some sort of incorrect format as I was unable to pass them into the tri function. I received an error stating that tri was exceeding index dimensions.

If I had the time to go back and start again from scratch, I would try to build a system that was easy to debug. Much of my time was spent typing in commands to check mundane things like the size of a matrix in the console, which could have been automated with a debugging script. I do the that the difficulty of my project was not too easy but not too hard. Given that, I think my performance was reasonable because I came very close to generating all the meshes I would need to create the final mesh. I do feel that I was able to experiment with threshold values enough and made some minor improvements its value prior to running into my errors. I believe that I should have scheduled a one on one with the TA in advance, anticipating a bug that would stall my work. I will keep that in mind for the future when building systems of this size.

IV. Appendix

My Functions:

Decode.m

Reconstruct.m

Project_demo.m

Triangulate.m

Create_calibration.m

Test_decode.m

Modified:

Mesh.m (triangle code was taken from the Professor's code, because his implementation was better)

Code Obtained:

MATLAB Calibration Tool from Caltech

MeshLab ICP/Poisson (Not Used)

Mesh2ply.m

Nbr_smooth.m