EE 454 Computer Vision
Project 2
Multiview Projection
November 4th, 2018

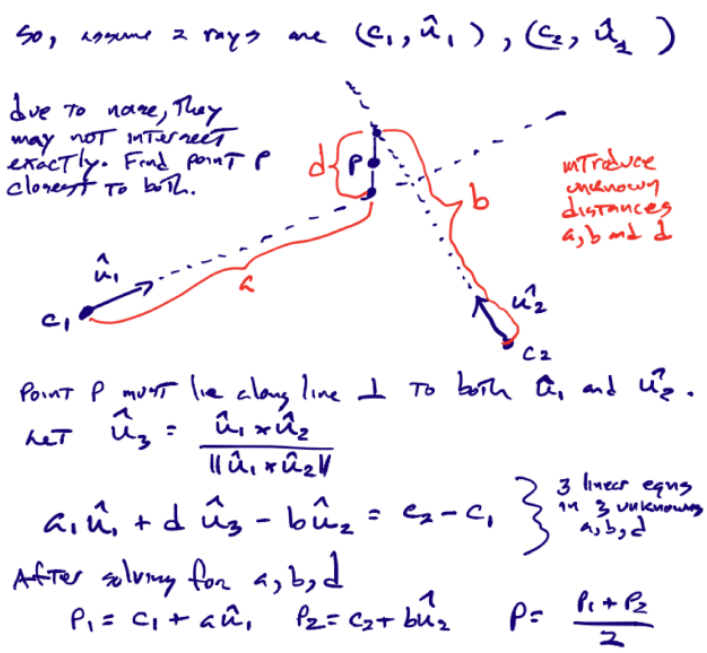
Gursahej Gandhoke
Emmanuel F. Werr

Project Outline and Implementation description:

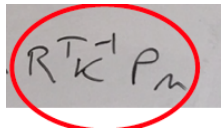
- In this project, we were asked to convert frame by frame a set of 12 coordinates from 3d to 2d and back to 3d.
- From this point forward, up until the next asterisk (*), there is a for loop
- First, we stored the U, V, and W world coordinates in x, y, and z respectively for a specific frame for each joint. Therefore x, y and z are vectors with 12 values each.
- We took those 3 vectors and our initial goal is to transfer them to 2D pixel coordinates. To do that we used the equation 1:

$$\lambda \overset{3x1}{P_u} = \overset{3x3}{K} \left[\underbrace{\overset{3x3}{R} \mid \overset{3x1}{t}} \right] \overset{4x1}{P_W}$$

  where K is *Kmat* from vue2 or vue4, R is *Rmat* or the rotation matrix; the [R|t] term can be represented by *Pmat* (which is our extrinsic parameter) and Pw is a matrix in World coordinate system that we calculated earlier. We have made a function threed2d that takes in the 3 coordinates and the camera vue(2 or 4) and returns an array of pixel locations in x and y coordinates for that vue(2 or 4). We create 2 arrays *Pjoint* and *Pjoint4* for storing 2D pixel values of vue2 and vue4 respectively for each joint.
- Conversion check we do at the end with image plotting.
- Now our job is to convert the 2d pixel points back to 3D points.
- We use the following slide:



- We are using an Ax = b model to solve the matrices to get the distances a and b. In my case we have 2 cameras vue2 and vue4 therefore the b matrix would be [camlocation4 - camlocation2] (which we get from eq. 1)
- A matrix would be a set of the viewing ray u1, -u2, and u3. u1, u2 and u3 is calculated using:



…where Pu is 2d pixel coordinates. In my case u1 is vue2VR, u2 is vue4VR, and u3 is calculated by the formula mentioned in the slide above.

The dimensions and specifications of A are mentioned in the code. Finally the x matrix would be the distances a, b and d which we need to find. Therefore, the solving equation is x = A\b the solution of which is stored in var1. Now we find the p1, and p2 values for both the cameras vue2 and vue4 respectively and use the formula p = (p1+p2)/2 to get the final world coordinated in 3D which is p. P is array of World coordinates for every joint.

- Hence the forward and backward projections are complete. The error values are stored in 'erroravg' which is L^2 avg. Respective error values are 'errormin', 'errormax', 'errorstd' (standard deviation), 'errormean', 'errormedian' for all 26214 frames. We didn't check for confidence since it was taking too much time to run the code to call the function 26214 times that in itself did 12 iterations each and 12+ accesses to memory.
- The errors for the initial and recomputed values are also mentioned in the code for all 3 axes.
- (*)
- The purpose of this for loop is to keep checking every error for each of the 26214 frames by incrementing 'mocapFnum'.
- At the end, we have entered all the plots as well as the plotting the coordinates on a specific frame of vue2 or vue4. we have redone some calculations to calculate them for a specific frame number, mainly the world coordinates, 'Pjoint', 'Pjoint4' and the current time. We can enter what frame number we want to check by altering 'mocapFnum1'. We use imshow and plot the values we got in 2D pixel values onto both images vue2 and vue4 for each joint to check if they work, and they do! We also have a plot to show the error averages (L^2 averages) of each frame.

Questions in section 2.2:

- Which are the internal parameters?
  - Kmat
- Which are the external parameters?
  - Pmat
- Which external parameters combine to form the matrix Pmat?
  - Rmat and (Rmat∗location)
- What is the location of the camera?
  - Vue.position

Experimental Observations:

Initial World Coord. at Frame 1000

p x y z erroravg vue2

1x12 double

	1	2	3	4	5	6	7	8	9	10	11	12
1	1.8312e...	1.8610e...	1.6514e...	1.5724e...	1.6024e...	1.4257e...	1.7544e...	1.7198e...	1.8881e...	1.6515e...	1.5016e...	1.5605e...

p x y z erroravg vue2

1x12 double

	1	2	3	4	5	6	7	8	9	10	11	12
1	1.6299e...	1.6115e...	1.7063e...	1.3033e...	1.2738e...	1.4122e...	1.5302e...	1.6410e...	1.5446e...	1.3967e...	1.4383e...	1.3026e...
2												

p x y z erroravg vue2

1x12 double

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	1.3225e...	1.0279e...	882.5990	1.3384e...	1.0403e...	867.0510	849.6600	435.1120	63.6316	836.6420	465.2640	47.9149	
2													
3													

Final (Forward & Backward projection) World Coord. At frame 1000

p x y z erroravg vue2

3x12 double

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	1.8312e...	1.8610e...	1.6514e...	1.5724e...	1.6024e...	1.4257e...	1.7544e...	1.7198e...	1.8881e...	1.6515e...	1.5016e...	1.5605e...	
2	1.6299e...	1.6115e...	1.7063e...	1.3033e...	1.2738e...	1.4122e...	1.5302e...	1.6410e...	1.5446e...	1.3967e...	1.4383e...	1.3026e...	
3	1.3225e...	1.0279e...	882.5990	1.3384e...	1.0403e...	867.0510	849.6600	435.1120	63.6316	836.6420	465.2640	47.9149	
4													

Initial World Coord. at Frame 26214

p x y z

1x12 double

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	603.4950	654.2270	583.4500	320.2340	386.1450	319.5750	539.4110	544.9610	591.0350	417.7910	437.3080	509.5420	
2													
3													

p x y z

1x12 double

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	1.1581e...	1.0984e...	1.1635e...	859.2240	807.8120	862.8540	1.0590e...	1.0105e...	903.7920	942.1520	939.5210	880.9470	
2													
3													
4													

p x y z

1x12 double

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	1.3901e...	1.0983e...	839.2770	1.4128e...	1.1250e...	852.4150	910.3490	481.6880	61.0172	898.5140	500.6270	52.6460	

Final (Forward & Backward projection) World Coord. At frame 26214

p x y z

3x12 double

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	603.4950	654.2270	583.4500	320.2340	386.1450	319.5750	539.4110	544.9610	591.0350	417.7910	437.3080	509.5420	
2	1.1581e...	1.0984e...	1.1635e...	859.2240	807.8120	862.8540	1.0590e...	1.0105e...	903.7920	942.1520	939.5210	880.9470	
3	1.3901e...	1.0983e...	839.2770	1.4128e...	1.1250e...	852.4150	910.3490	481.6880	61.0172	898.5140	500.6270	52.6460	
4													

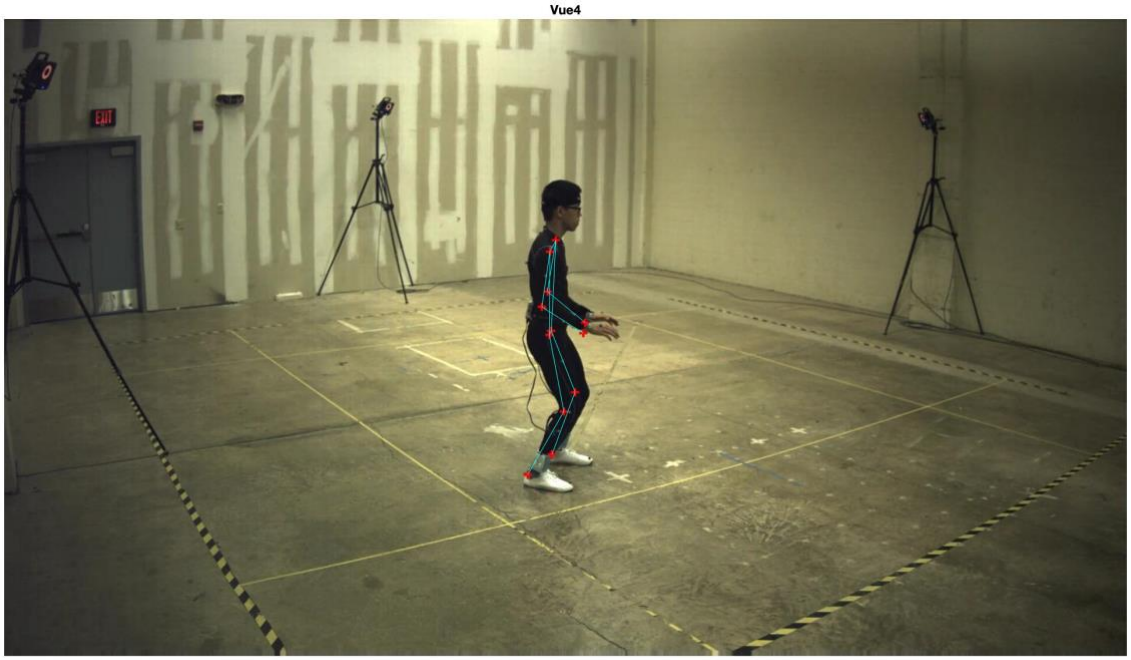
As you can see the x, y, z and p values match, hence backward and forward projection was a success.

Qualitative Results:

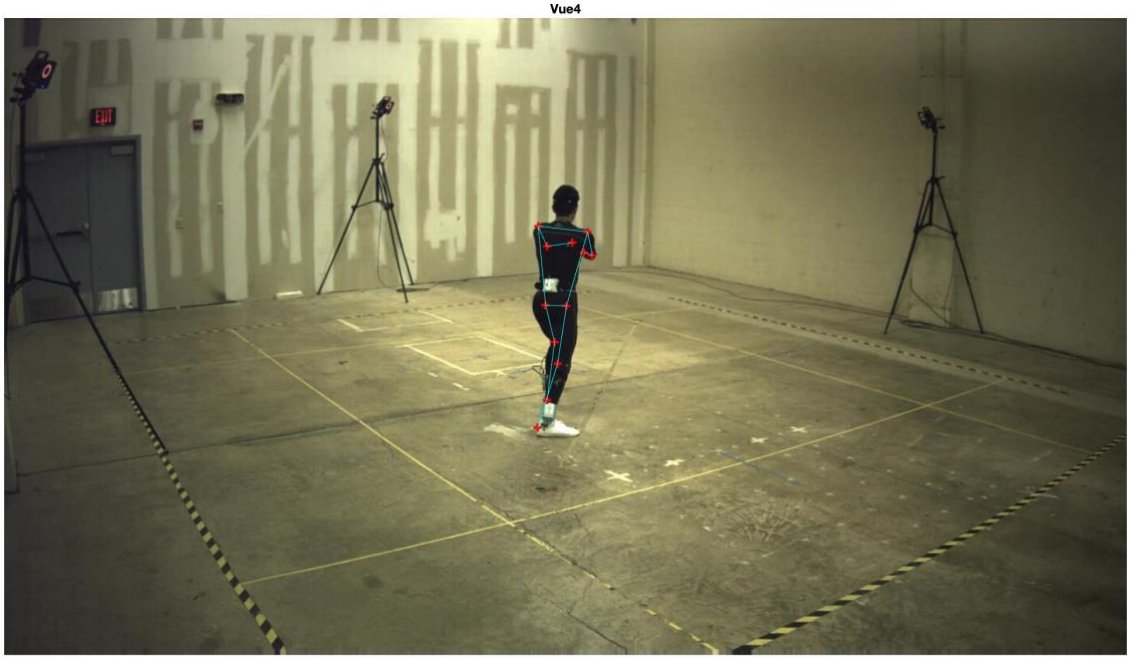
Vue2 (frame 1,000)

Vue2 (frame 10,000)

Vue4 (frame 1,000)

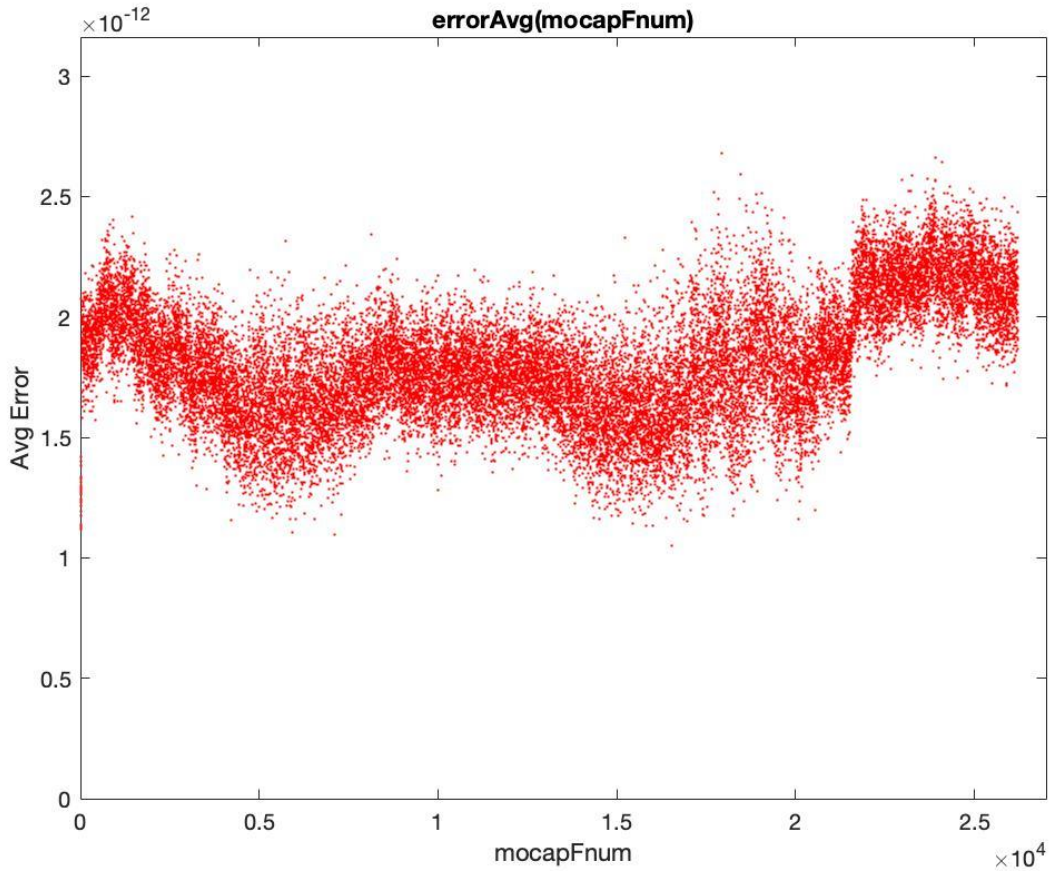


Vue4 (frame 10,000)



Quantitative Results:

Average error for each frame



There are notable peaks and valleys across the curve. It seems to be less accurate during the extremas of our domain (mocapFnum). It also appears somewhat like a sine wave.

Error chart (5 x 13)

Name Box

	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Joint 1	Joint 2	Joint 3	Joint 4	Joint 5	Joint 6	Joint 7	Joint 8	Joint 9	Joint 10	Joint 11	Joint 12	Joint total avg
2 Error()median	2.00E-12	1.94E-12	1.96E-12	1.97E-12	1.86E-12	1.89E-12	1.78E-12	1.68E-12	1.61E-12	1.75E-12	1.65E-12	1.55E-12	1.80E-12
3 Error()max	5.64E-12	5.19E-12	4.99E-12	4.84E-12	5.26E-12	5.23E-12	4.97E-12	5.14E-12	5.44E-12	4.98E-12	4.72E-12	4.74E-12	5.10E-12
4 Error()mean	2.02E-12	1.95E-12	1.97E-12	1.99E-12	1.90E-12	1.91E-12	1.80E-12	1.69E-12	1.60E-12	1.78E-12	1.66E-12	1.57E-12	1.82E-12
5 Error()min	4.86E-13	2.62E-13	2.34E-13	4.10E-13	4.58E-13	1.34E-13	1.71E-13	8.04E-14	0.00E+00	2.80E-13	1.30E-13	1.42E-14	2.22E-13
6 Error()std	5.13E-13	5.48E-13	5.63E-13	5.05E-13	5.47E-13	5.59E-13	5.63E-13	6.04E-13	6.29E-13	5.68E-13	5.93E-13	6.26E-13	5.68E-13
7													

Here is the error chart for each of the 12 joints across all 26214 frames. As well as the joint total avg. error in the final column of the chart.

Algorithm Efficiency:

This is the initial Profile Summary.

Function Name	Calls	**Total Time**	Self Time*	Total Time Plot (dark band = self time)
main	1	13.154 s	8.626 s	
threed2d	629160	1.745 s	1.745 s	
cross	629136	1.124 s	1.124 s	
VideoReader>VideoReader.cacheNextFrame	5	0.933 s	0.007 s	
InputStream>InputStream.read	9	0.926 s	0.035 s	
Stream>Stream.wait	5	0.856 s	0.199 s	
...Reader>VideoReader.set.CurrentTime	1	0.856 s	0.003 s	
...VideoReader>VideoReader.seekToTime	1	0.850 s	0.002 s	
...er>VideoReader.readFrameAtPosition	1	0.848 s	0.002 s	
VideoReader>VideoReader.readFrameAtTime	1	0.846 s	0.002 s	
...ideoReader>VideoReader.VideoReader	2	0.428 s	0.002 s	
VideoReader.VideoReader>VideoReader.init	2	0.426 s	0.006 s	
VideoReader>VideoReader.VideoReader	2	0.402 s	0.012 s	
imshow	1	0.255 s	0.024 s	
...>@(obj)obj.getDataAvailable()>0	7504	0.206 s	0.028 s	
Stream>Stream.isDeviceDone	7503	0.204 s	0.204 s	
VideoReader>VideoReader.createChannel	2	0.186 s	0.012 s	
Stream>Stream.getDataAvailable	7519	0.180 s	0.180 s	
Stream>Stream.isOpen	7497	0.175 s	0.175 s	
initSize	1	0.159 s	0.008 s	
movegui	1	0.137 s	0.004 s	

This is the final, updated, main.m file Profile Summary.

Function Name	Calls	**Total Time**	Self Time*	Total Time Plot (dark band = self time)
main	1	11.345 s	7.340 s	
threed2d	629160	1.564 s	1.564 s	
cross	629136	0.956 s	0.956 s	
VideoReader>VideoReader.cacheNextFrame	5	0.782 s	0.004 s	
InputStream>InputStream.read	10	0.778 s	0.029 s	
...Reader>VideoReader.set.CurrentTime	1	0.723 s	0.002 s	
Stream>Stream.wait	5	0.720 s	0.160 s	
...VideoReader>VideoReader.seekToTime	1	0.717 s	0.001 s	
...er>VideoReader.readFrameAtPosition	1	0.716 s	0.001 s	
VideoReader>VideoReader.readFrameAtTime	1	0.714 s	0.002 s	
...ideoReader>VideoReader.VideoReader	2	0.358 s	0.001 s	
VideoReader.VideoReader>VideoReader.init	2	0.356 s	0.003 s	
VideoReader>VideoReader.VideoReader	2	0.342 s	0.010 s	
imshow	1	0.268 s	0.022 s	
...>@(obj)obj.getDataAvailable()>0	7737	0.178 s	0.023 s	
Stream>Stream.isDeviceDone	7737	0.172 s	0.172 s	
initSize	1	0.171 s	0.008 s	
VideoReader>VideoReader.createChannel	2	0.164 s	0.011 s	
Stream>Stream.getDataAvailable	7753	0.156 s	0.156 s	
Stream>Stream.isOpen	7730	0.151 s	0.151 s	
movegui	1	0.148 s	0.004 s	

The Algorithm efficiency increased when we predefined the matrices as zeros.

Documentation of what each team member did to contribute to the project.

We both did equal work in both the code and the report. Focusing heavily on being very careful to meet expectations and create a project submission that was of high caliber.