Project Reporting and Evaluation,

(A) Summarize

Within this project we achieved the following: verifying the matrix (k * p) matrix intrinsic and extrinsic of the camera. Then we projected the 3d points onto the camera screens using the p matrix and the 3d points. Part 2.4 we triangulated the 2d points back into the 3d points by drawing the rays from the camera to the image point and extending them. Then minimizing the distance between them we found the 3d point again. We then measured the Euclidean distance and calculated the error. Finally, part 2.6 we computed the epipolar lines between the views and drew them on the images.

This projected required a complete understanding of the relationships between 2D image coordinates and the 3D world coordinated and the chain of transformations that make the pinhole camera model work.

I expected to achieve a higher level of knowledge and understanding about the fundamentals of computer vision.

(B) Outline

The camera parameters needed are k and p. Since the pmat does not include the intrinsic k, however in class the pmat includes the k matrix. This was a big issue when I started out the project. To calculate and verify the p matrix I calculated *myp* from the equation P = KR{I|-C} from the slides and the given information from the *camera.mat* file. This is calculated in the myp function on the MATLAB file. Because the p matric calculated in the function my matches the matrix created from multiplying the k matrix with the p matrix. I can verify that this is the p matrix that we must use in part 2.3 to project the 3d point on the the 2d screen. This is also verified when then I plot the points on the images. I created a Euclideandistance function using the equation from the slides: delta = sqrt((vec3point1(1)-vec3point2(1))^2 + (vec3point1(2)-vec3point2(2))^2 + (vec3point1(3)-vec3point2(3))^2). I first used the F matrix to calculate the line for the epiploon line. However, is it much faster to project the camera position to the screen for every frame. The calculation for the F matrix and the SVD decomposition is commented out within the MATLAB file.

(C) Experimental Observations

Running my program does exactly as expect, the code does need to be run in order, so that the structure are created before the triangulations and the error and epiploon lines can be drawn. Each section of code is divided by the 2.1 – 2.6 as stated in the project description. The output shows that the process was completed successfully and the time it took to complete those sections. Sections that take under a second to finish are not shown. The output is shown below:

```
The camera P matrix was calculated correctly
Projection from 3d points X to 2d points x for each frame on both cameras Complete
Successfully
delta Time = 00:00:05
Triangulation from 2d points x to 3d points X for each frame Complete Successfully
delta Time = 00:01:17
```

```
The error between pairs of 3d points on each frame Complete Successfully
delta Time = 00:00:04
--- Error For each Frame ---
Mean 2.4773e-11
Standard deviation 3.4885e-12
Minimum 1.521e-11
Median 2.4187e-11
Max 4.6613e-11
--- Total Error ---
Mean 2.0644e-12
Standard deviation 7.5638e-13
Minimum 0
Median 1.9952e-12
Max 9.3334e-12
Sum 6.381e-07
```

(D) Quantitative Results

The Code to calculate the error for each frame is in the MATLAB project file and a copy is below.

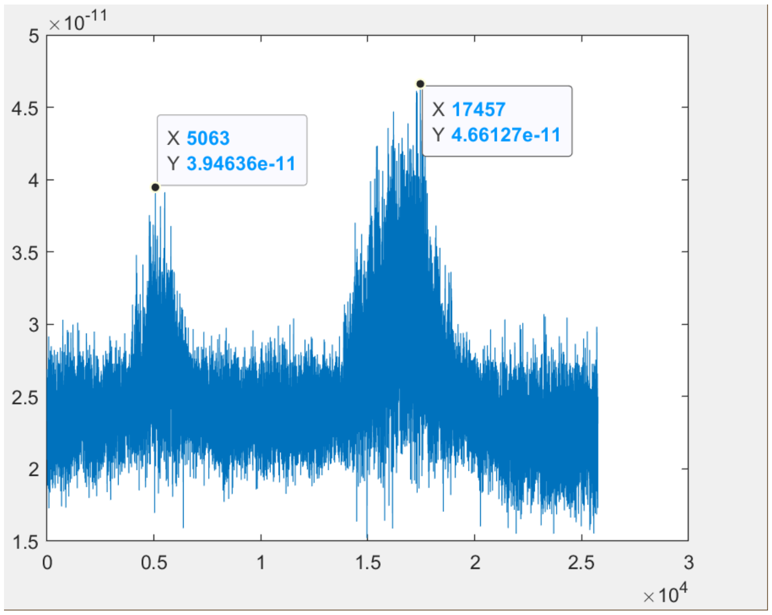
```
ittorator = 1;
    for frame = 1:length(goodFrames)
        mocapFnum = goodFrames(frame).frame; % for each frame
        dxSum = 0;
        for i = 1:1:12 % for each point
            % Compute the L^2 (Euclidean) distance.
            % ((X-X')^2 + (Y-Y')^2 + (Z-Z')^2)
            actualJoint = [goodFrames(frame).mocapJoints(i).X;
                goodFrames(frame).mocapJoints(i).Y;
                goodFrames(frame).mocapJoints(i).Z];
            calculatedJoint = [goodFrames(frame).calculatedjoints(i).x ;
                goodFrames(frame).calculatedjoints(i).y;
                goodFrames(frame).calculatedjoints(i).z];
            dx = Euclideandistance(actualJoint, calculatedJoint);
            dxSum = dxSum + dx;

            % 1 joint all framse error
             jointerror(i).list(frame) = dx;
            % all the frames all the joints
            error(ittorator) = dx;
            ittorator = ittorator + 1;
        end
        % every frame all the joints
        frameerror(frame) = dxSum;
    end
```

This shows the error for each joint independently.

--- Joint Error ---

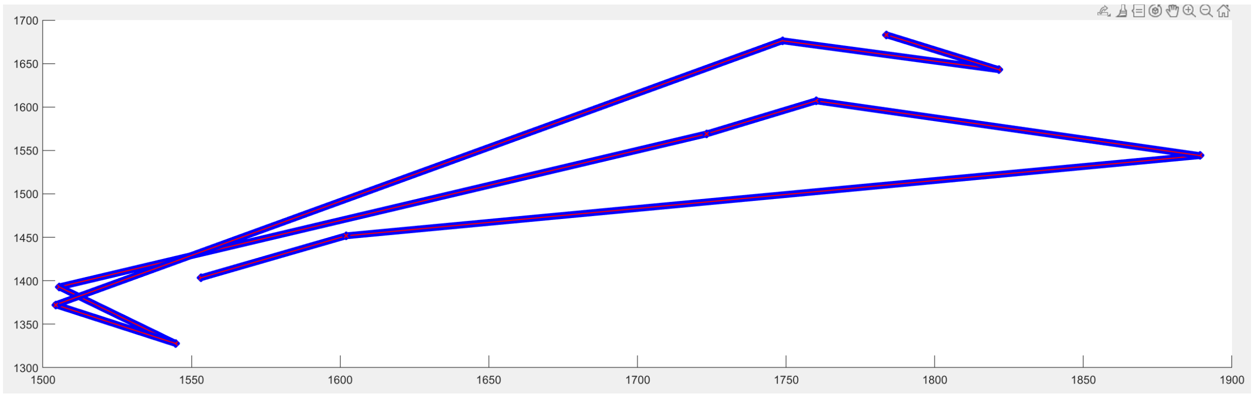
joint 1,	Mean,	STD,	Min,	Midian,	Max
joint 1,	2.0837e-12,	7.8532e-13,	1.1369e-13,	2.0025e-12,	7.9027e-12
joint 2,	2.0514e-12,	7.6786e-13,	0,	1.978e-12,	7.784e-12
joint 3,	2.0556e-12,	7.908e-13,	0,	1.9855e-12,	9.2248e-12
joint 4,	2.1079e-12,	8.3695e-13,	0,	2.0337e-12,	9.3334e-12
joint 5,	2.0951e-12,	8.2333e-13,	0,	2.0145e-12,	8.4923e-12
joint 6,	2.0955e-12,	8.3172e-13,	0,	2.0081e-12,	8.3302e-12
joint 7,	2.0374e-12,	7.2886e-13,	0,	1.9658e-12,	8.0083e-12
joint 8,	2.0181e-12,	6.8446e-13,	1.1369e-13,	1.9625e-12,	7.4627e-12
joint 9,	2.053e-12,	6.7519e-13,	1.1369e-13,	1.9992e-12,	7.1355e-12
joint 10,	2.0567e-12,	7.4915e-13,	0,	1.992e-12,	8.29e-12
joint 11,	2.0476e-12,	6.9208e-13,	5.6843e-14,	1.9887e-12,	7.3783e-12
joint 12,	2.0707e-12,	6.7838e-13,	1.4229e-13,	2.0224e-12,	7.1043e-12

A graph of the error for each frame is below. There are two peaks of increased error around frames 5063 and 17400 as shown below.

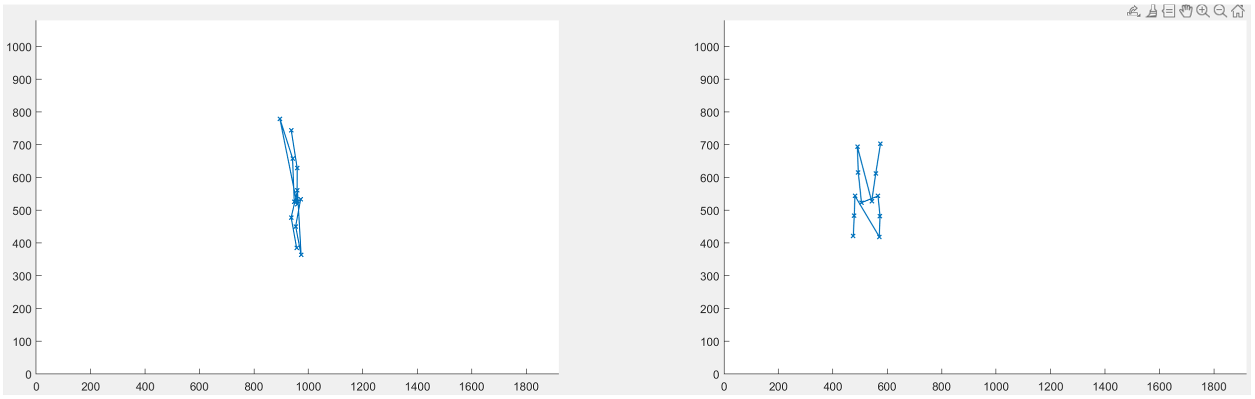


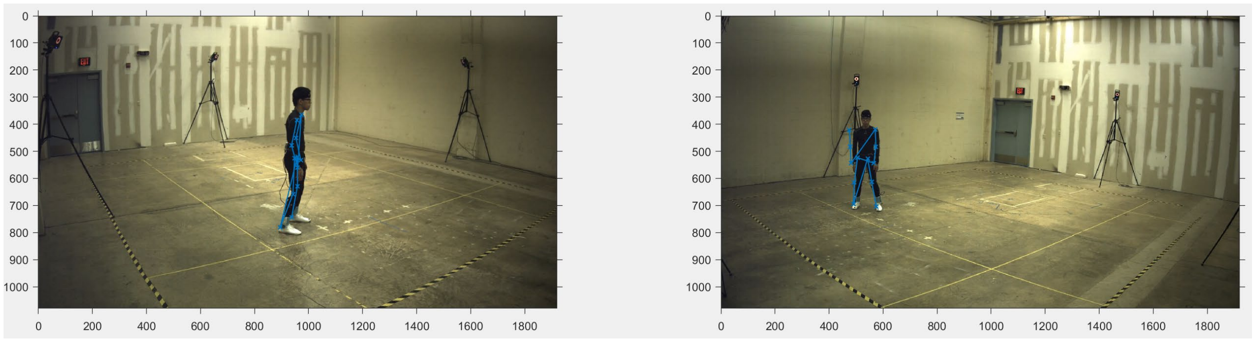
(E) Quantitative Results

Below is the 3d plot of the calculated skeleton(red) and the given joints(blue). Since the points and the lines are extremely close(overlapping) and the error is small it's is consistent with the results above that after triangulation, the 3d points are /within reason, the same.



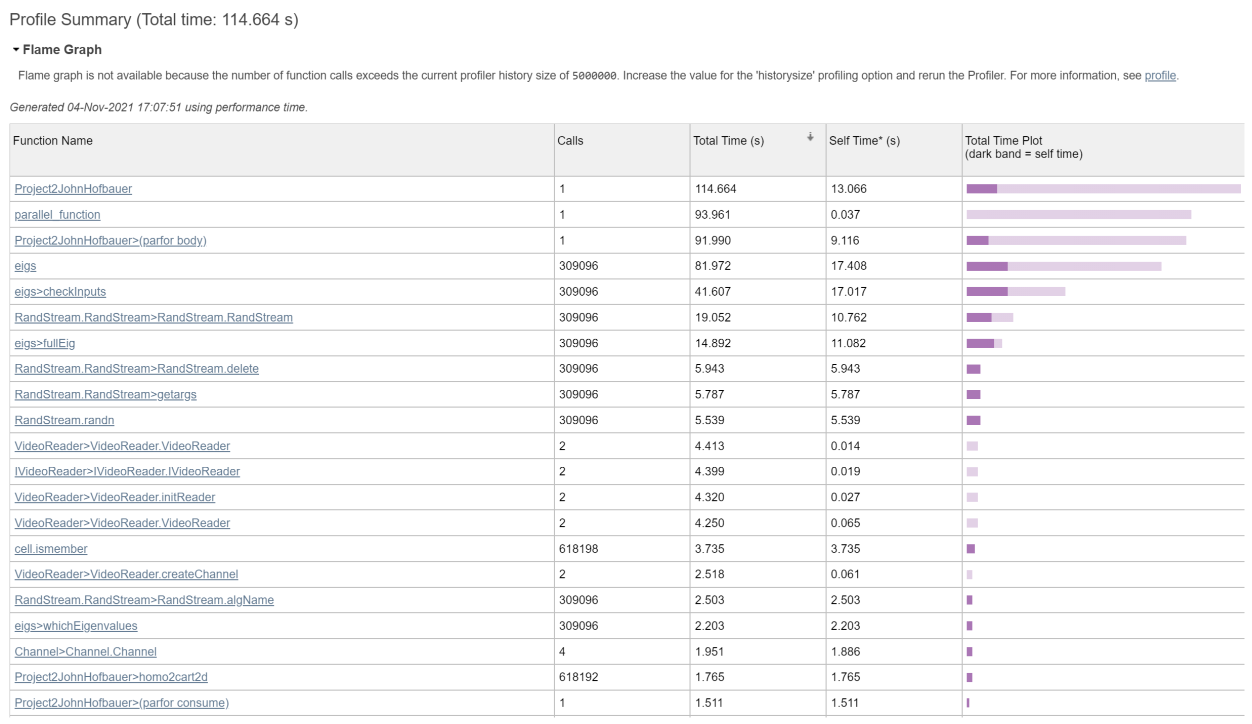
Below is the skeleton from the projected 3D points onto the 2D screen for both cameras where the top is without the image background and the image below includes the image background for the skeleton.





(F) Algorithm Efficiency

Since triangulation has the longest run time; using parfor instead of the regular for loop MATLAB is able to run the triangulations for each frame and every joint in parallel. This along with projecting the camera position instead of using the fundamental matrix decrease the amount of time it takes for the project to run.


Profile Summary (Total time: 114.664 s)

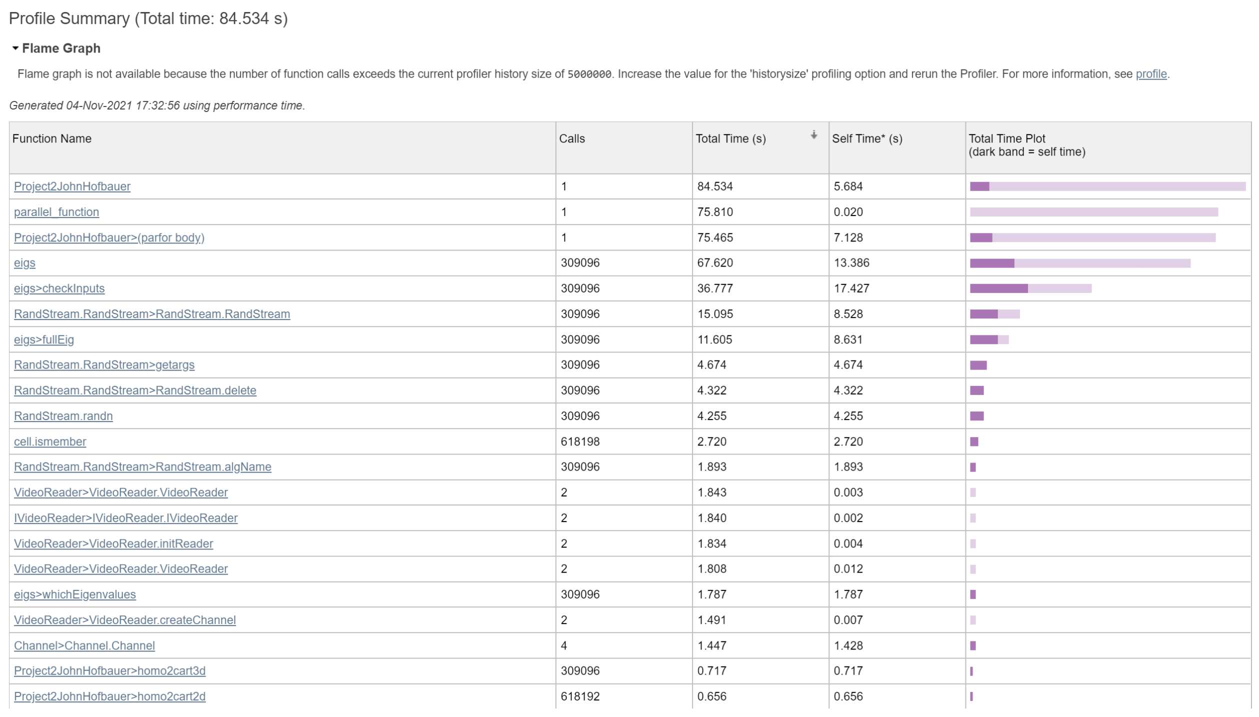
▾ **Flame Graph**

Flame graph is not available because the number of function calls exceeds the current profiler history size of `5000000`. Increase the value for the 'historysize' profiling option and rerun the Profiler. For more information, see profile.

Generated 04-Nov-2021 17:07:51 using performance time.

Function Name	Calls	Total Time (s)	Self Time* (s)	Total Time Plot (dark band = self time)
Project2JohnHofbauer	1	114.664	13.066	
parallel_function	1	93.961	0.037	
Project2JohnHofbauer>(parfor body)	1	91.990	9.116	
eigs	309096	81.972	17.408	
eigs>checkInputs	309096	41.607	17.017	
RandStream.RandStream>RandStream.RandStream	309096	19.052	10.762	
eigs>fullEig	309096	14.892	11.082	
RandStream.RandStream>RandStream.delete	309096	5.943	5.943	
RandStream.RandStream>getargs	309096	5.787	5.787	
RandStream.randn	309096	5.539	5.539	
VideoReader>VideoReader.VideoReader	2	4.413	0.014	
IVideoReader>IVideoReader.IVideoReader	2	4.399	0.019	
VideoReader>VideoReader.initReader	2	4.320	0.027	
VideoReader>VideoReader.VideoReader	2	4.250	0.065	
cell.ismember	618198	3.735	3.735	
VideoReader>VideoReader.createChannel	2	2.518	0.061	
RandStream.RandStream>RandStream.algName	309096	2.503	2.503	
eigs>whichEigenvalues	309096	2.203	2.203	
Channel>Channel.Channel	4	1.951	1.886	
Project2JohnHofbauer>homo2cart2d	618192	1.765	1.765	
Project2JohnHofbauer>(parfor consume)	1	1.511	1.511	

Below is the for loop with 4 work units:


Profile Summary (Total time: 84.534 s)

▾ **Flame Graph**

Flame graph is not available because the number of function calls exceeds the current profiler history size of `5000000`. Increase the value for the 'historysize' profiling option and rerun the Profiler. For more information, see profile.

Generated 04-Nov-2021 17:32:56 using performance time.

Function Name	Calls	Total Time (s)	Self Time* (s)	Total Time Plot (dark band = self time)
Project2JohnHofbauer	1	84.534	5.684	
parallel_function	1	75.810	0.020	
Project2JohnHofbauer>(parfor body)	1	75.465	7.128	
eigs	309096	67.620	13.386	
eigs>checkInputs	309096	36.777	17.427	
RandStream.RandStream>RandStream.RandStream	309096	15.095	8.528	
eigs>fullEig	309096	11.605	8.631	
RandStream.RandStream>getargs	309096	4.674	4.674	
RandStream.RandStream>RandStream.delete	309096	4.322	4.322	
RandStream.randn	309096	4.255	4.255	
cell.ismember	618198	2.720	2.720	
RandStream.RandStream>RandStream.algName	309096	1.893	1.893	
VideoReader>VideoReader.VideoReader	2	1.843	0.003	
IVideoReader>IVideoReader.IVideoReader	2	1.840	0.002	
VideoReader>VideoReader.initReader	2	1.834	0.004	
VideoReader>VideoReader.VideoReader	2	1.808	0.012	
eigs>whichEigenvalues	309096	1.787	1.787	
VideoReader>VideoReader.createChannel	2	1.491	0.007	
Channel>Channel.Channel	4	1.447	1.428	
Project2JohnHofbauer>homo2cart3d	309096	0.717	0.717	
Project2JohnHofbauer>homo2cart2d	618192	0.656	0.656	

(F) Epipolar lines

Below shows the epipolar lines draws on the two images. The epipolar lines are generated by connecting the two points from the projected joints and the projected camera position. The epipolar lines can also be calculated using the fundamental matrix and getting the line equation then solving for

y. This is commented out as it is slower than projecting the camera onto the screen and connecting the two points and has the same result.