Fundamentals of Computer Vision, Spring 2020
Project Assignment 2
(3D point to 2D) Point and Inverse (2D point to 3D ray) Camera Projection
Due Date: Sunday, April 19, 2020 11:59pm EST

1 Motivation

The goal of this project is to implement forward (3D point to 2D point) and inverse (2D point to 3D ray) camera projection, and to perform triangulation from two cameras to do 3D reconstruction from pairs of matching 2D image points. This project will involve understanding relationships between 2D image coordinates and 3D world coordinates and the chain of transformations that make up the pinhole camera model that was discussed in class. Your specific tasks will be to project 3D coordinates (sets of 3D joint locations on a human body, measured by motion capture equipment) into image pixel coordinates that you can overlay on top of an image, to then convert those 2D points back into 3D viewing rays, and then triangulate the viewing rays of two camera views to recover the original 3D coordinates you started with (or values close to those coordinates).

You will be provided:

- 3D point data for each of 12 body joints for a set of motion capture frames recorded of a subject performing a Taiji exercise. The 12 joints represent the shoulders, elbows, wrists, hips, knees, and ankles. Each joint will be provided for a time series that is ~30,000 frames long, representing a 5-minute performance recorded at 100 frames per second in a 3D motion capture lab.
- Camera calibration parameters (Intrinsic and extrinsic) for two video cameras that were also recording the performance. Each set of camera parameters contains all information needed to project 3D joint data into pixel coordinates in one of the two camera views.
- An mp4 movie file containing the video frames recorded by each of the two video cameras. The video was recorded at 50 frames per second.

While this project appears to be a simple task at first, you will discover that practical applications have hurdles to overcome. Specifically, in each frame of data there are 12 joints with ~30,000 frames of data to be projected into 2 separate camera coordinate systems. That is over ~700,000 joint projections into camera views and ~350,000 reconstructions back into world coordinates! Furthermore, you will need to have a very clear understanding of the pinhole camera model that we covered in class, to be able to write functions to correctly project from 3D to 2D and back again.

The specific project outcomes include:

- Experience in Matlab programming

- Understanding intrinsic and extrinsic camera parameters
- Projection of 3D data into 2D images coordinates
- Reconstruction of 3D locations by triangulation from two camera views
- Measurement of 3D reconstruction error
- Practical understanding of epipolar geometry.

2 The Basic Operations

The following steps will be essential to the successful completion of the project:

1. Input and parsing of mocap dataset. Read in and properly interpret the 3D joint data.
2. Input and parsing of camera parameters. Read in each set of camera parameters and interpret with respect to our mathematical camera projection model.
3. Use the camera parameters to project 3D joints into pixel locations in each of the two image coordinate systems.
4. Reconstruct the 3D location of each joint in the world coordinate system from the projected 2D joints you produced in Step3, using two-camera triangulation.
5. Compute Euclidean (L^2) distance between all joint pairs. This is a per joint, per frame L^2 distance between the original 3D joints and the reconstructed 3D joints providing a quantitative analysis of the distance between the joint pairs.

2.1 Reading the 3D joint data

The motion capture data is in file Subject4-Session3-Take4_mocapJoints.mat . Once you load it in, you have a 21614x12x4 array of numbers. The first dimension is frame number, the second is joint number, and the last is joint coordinates + confidence score for each joint. Specifically, the following snippet of code will extract x,y,z locations for the joints in a specific mocap frame.

```
mocapFnum = 1000;  %mocap frame number 1000
x = mocapJoints(mocapFnum,:,1);     %array of 12 X coordinates
y = mocapJoints(mocapFnum,:,2);     %  Y coordinates
z = mocapJoints(mocapFnum,:,3);     %  Z coordinates
conf = mocapJoints(mocapFnum,:,4)   %confidence values
```

Each joint has a binary “confidence” associated with it. Joints that are not defined in a frame have a confidence of 0. Feel free to Ignore any frames don’t have all confidences = 1.

There are 12 joints, in this order:

```
 1 Right shoulder
 2 Right elbow
 3 Right wrist
 4 Left shoulder
 5 Left elbow
 6 Left wrist
 7 Right hip
```

```
 8 Right knee
 9 Right ankle
10 Left hip
11 Left knee
12 Left ankle
```

2.2 Reading camera parameters

There are two cameras, called “vue2” and “vue4”, and two files specifying their calibration parameters: vue2CalibInfo.mat and vue4Calibinfo.mat . Each of these contains a structure with intrinsic, extrinsic, and nonlinear distortion parameters for each camera. Here are the values of the fields after reading in one of the structures

```
vue2 =
struct with fields:
         foclen: 1557.8
    orientation: [-0.27777 0.7085 -0.61454 -0.20789]
       position: [-4450.1 5557.9 1949.1]
      prinpoint: [976.04 562.82]
         radial: [1.4936e-07 4.3841e-14]
    aspectratio: 1
           skew: 0
           Pmat: [3×4 double]
           Rmat: [3×3 double]
           Kmat: [3×3 double]
```

Part of your job will be figuring out what those fields mean in regards to the pinhole camera model parameters we discussed in class lectures. Which are the internal parameters? Which are the external parameters? Which internal parameters combine to form the matrix Kmat? Which external parameters combine to form the matrix Pmat? Hint: the field “orientation” is a unit quaternion vector describing the camera orientation, which is also represented by the 3x3 matrix Rmat. What is the location of the camera? Verify that location of the camera and the rotation Rmat of the camera combine in the expected way (expected as per one of the slides in our class lectures on camera parameters) to yield the appropriate entries in Pmat.

2.3 Projecting 3D points into 2D pixel locations

Ignoring the nonlinear distortion parameters in the “radial” field for now, write a function from scratch that takes either a single 3D point or an array of 3D points and projects it (or them) into 2D pixel coordinates. You will want to refer to our lecture notes for the transformation chain that maps 3D world coordinates into 2D pixel coordinates.

For verification, it will be helpful to visualize your projected 2D joints by overlaying them as points on the 2D video frame corresponding to the motion capture frame. Two video files are given to you: Subject4-Session3-24form-Full-Take4-Vue2.mp4 is the video from camera vue2, and Subject4-Session3-24form-Full-Take4-Vue4.mp4 is the video from camera vue4. To get a video frame out of the mp4 file we can use VideoReader in matlab. It is nonintuitive to use, so

to help out, here is a snippet of code that can read the video frame from vue2 corresponding to the motion capture frame number mocapFnum.

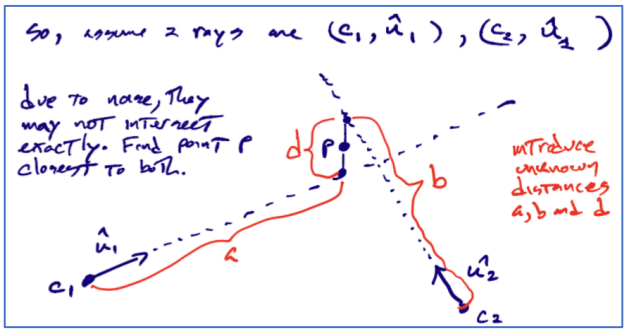
```
%initialization of VideoReader for the vue video.
%YOU ONLY NEED TO DO THIS ONCE AT THE BEGINNING

filenamevue2mp4 = 'Subject4-Session3-24form-Full-Take4-Vue2.mp4';
vue2video = VideoReader(filenamevue2mp4);

%now we can read in the video for any mocap frame mocapFnum.
%the (50/100) factor is here to account for the difference in frame
%rates between video (50 fps) and mocap (100 fps).

vue2video.CurrentTime = (mocapFnum-1)*(50/100)/vue2video.FrameRate;
vid2Frame = readFrame(vue2video);
```

The result is a 1088x1920x3 unsigned 8-bit integer color image that can be displayed by image(vid2Frame).

If all went well with your projection of 3D to 2D, you should be able to plot the x and y coordinates of your 2D points onto the image, and they should appear to be in roughly the correct places. IMPORTANT NOTE: since we ignore nonlinear distortion for now, it might be the case that your projected points look shifted off from the correct image locations. That is OK. However, if the body points are grossly incorrect (body is much larger or smaller or forming a really weird shape that doesn't look like the arms and legs of the person in the image), then something is likely wrong in your projection code.

2.4 Triangulation back into a set of 3D scene points

As a result of the above step, for a given mocap frame you now have two sets of corresponding 2D pixel locations, in the two camera views. Perform triangulation on each of the 12 pairs of 2D points to estimate a recovered 3D point position. As per our class lecture on triangulation, this will be done, for a corresponding pair of 2D points, by converting each into a viewing ray represented by camera center and unit vector pointing along the ray passing through the 2D point in the image and out into the 3D scene. You will then compute the 3D point location that is closest to both sets of rays (because they might not exactly intersect). Go back and refer to our lecture on Triangulation to see how to do the computation.

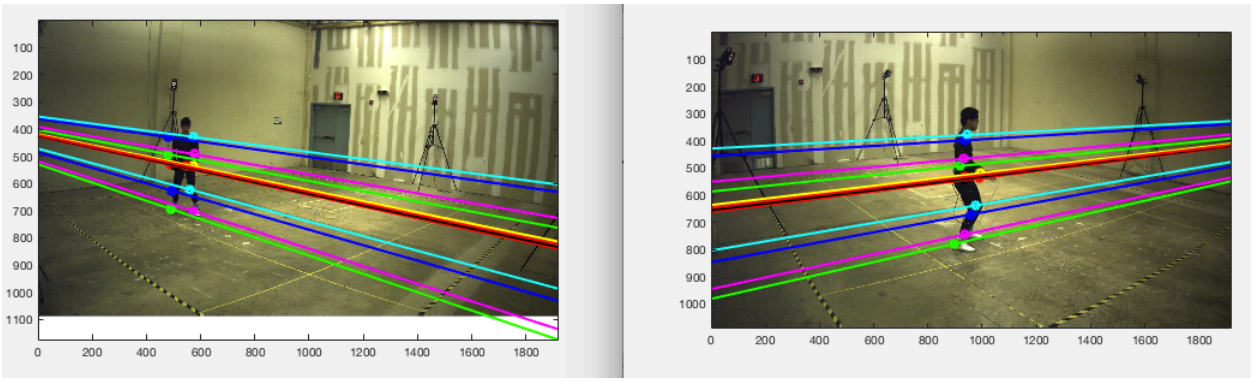


2.5 Measure error between triangulated and original 3D points

Because you are projecting 3D points into 2D (forward projection) and then backprojecting those into viewing rays (backward projection) to do triangulation back into 3D, you should get reconstructed point locations that are very similar to the original locations. However, the computed locations may not be perfectly the same as the original locations due to inaccuracies in the projection equations or even just small but nonzero numerical inaccuracies of floating point calculations. To quantitatively measure how close your reconstructed 3D points come to the original set of 3D points, you can compute L^2 (Euclidean) distance between them. If an original point location is (X,Y,Z) and the recovered location is (X',Y',Z'), then the L^2 distance between them is sqrt($(X-X')^2 + (Y-Y')^2 + (Z-Z')^2$). Consider evaluating the average L^2 distance or sum of L^2 distances to summarize the reconstruction accuracy for any given frame. More rigorous error analysis is described in the Evaluation section.

2.6 Computing epipolar lines between the two views

Because we are using two calibrated cameras, we also have an opportunity to explore the epipolar geometry of the two views. The picture below shows image projections of mocap points in the two camera views, along with the corresponding epipolar lines in both views. Conjugate epipolar lines have the same color, which is also the color of the projected image points that should lie on them. For example, right ankle is drawn as green points in both views, and the green epipolar lines should pass through them (or at least close to them – again we note that we ignoring nonlinear lens distortion that would make our projection model even more accurate).



Write a function that displays a similar-looking image pair for some other mocap frame/pose in the dataset. There are several approaches you could take, based on our understanding of what epipolar geometry is and how it relates points and lines in two camera views. Here are three approaches I can think of off the top of my head: 1) determine the epipoles in each view from the given the camera calibration data (you will need to remember what an epipole really is, in terms of camera locations). For each projected mocap point in the image, you could then draw the line containing that point and the epipole for that image; 2) for each projected point in the image, determine a viewing ray that backprojects into the scene (which you needed to do already for doing triangulation). Project two points on that viewing ray into the other camera

view, and draw the line that contains them; 3) this one is hardest mathematically, but the most theoretically rigorous – compute the 3x3 fundamental matrix F between the two views using the camera calibration information given, and then use your knowledge of how to compute epipoles and epipolar lines given matrix F to draw the epipolar lines. To do this, you will need to determine from the camera calibration information what the location of the vue4 camera is with respect to the coordinate system of vue2 (or vice versa) and the relative rotation between them, then combine those to compute the essential matrix E = R S, and finally pre and post multiply E by the appropriate film-to-pixel affine matrices to turn E into fundamental matrix F. You have all the camera information you need, but it is a little tricky to get E because although we know how the cameras are related to the same world coordinate system, we aren't directly told how the two camera coordinate systems are related to each other – some mathematical derivation is necessary to figure this out.

3 Evaluation

As part of your report, and to convince us that your code is working correctly, we would like you to show the following quantitative and qualitative evaluations of your work.

Quantitative

For this project, quantifying the accuracy of the algorithm's implementation is difficult for a human because of the large amount of data, but easy for a computer to calculate and analyze. There is one fundamental method you are to implement and document for quantitative results, the L^2 distance, described above in section 2.5. Once the L^2 distance is calculated for each joint pair in each mocap frame (you only have to use the mocap frames where all 12 joints are valid, i.e. all 12 have confidence values of 1), the following statistics should be calculated from the L^2 distance data:

- For each of the twelve joint pairs (original, recovered), provide the mean, standard deviation, minimum, median, and maximum of the L^2 distance computed over the entire time sequence.
- For all joint pairs (independent of their locations), provide the mean, standard deviation, minimum, median, and maximum of the L^2 distance computed over the entire time sequence.
- For each mocap frame, compute the sum of L^2 distances across the 12 joint pairs in that frame, and plot that sequence of "total error" values to see whether there are any interesting correlations between frame numbers and amount of reconstruction error.

Qualitative

The report should also provide visualizations of each step of the data processing starting with the input 3D data, ending with reconstructed 3D, and including images of each of the 2D projections in the two camera views. These example images should be of selected frames and those specific frame indexes should be used throughout the report when showing examples. The example frames should represent an accurate cross section of your results but must include the frame indexes with the minimum and maximum total error for all joints.

For ease of viewing in both 2D and 3D, rather than visualizing joints as independent points (using plot3 in 3D, and plot in 2D), we'd like you to also draw line segments between suitable body joint pairs to form a skeleton. For example, draw a line between the right shoulder and right elbow, between the right elbow and right wrist, etc. It should be obvious which joints to connect together to draw each of the four limbs. However, please also draw lines connecting the right shoulder to left shoulder, right hip to left hip, and a "spine" that connects the midpoint of the two shoulder joints to the midpoint of the two hip joints.

Efficiency

Use the Matlab profile() function to analyze and quantify the efficiency of your implementation. This tool allows for a full breakdown of the processor time used by each function call in a Matlab script. It isn't required that your code be super-efficient, and we won't penalize slow functions, but it's good practice to check where the bulk of the running time is coming from, as that may indicate places that you could speed up in the future. More information on the Matlab profile function is here: https://www.mathworks.com/help/matlab/ref/profile.html

4 What Libraries Can I Use?

The intent is that you will implement the various computational modules described using general processing functions (https://www.mathworks.com/help/matlab/functionlist.html) in Matlab. The single notable exception is that you MAY use the image processing toolbox (https://www.mathworks.com/help/images/index.html). Specifically, you MAY NOT use anything from the computer vision toolbox, or any third-party libraries/packages. Don't plagiarize any code from anywhere or anyone else.

5 Project Reporting and Evaluation

Half of your grade will be based on submitting a fully operation program, and the other half will be based on a written report discussing your program, design decisions, and experimental observations. The following components will be submitted:

1) Submit a written report in which you discuss at least the following:
   a) Summarize *in your own words* what you think the project was about. What were the tasks you performed; what did you expect to achieve?
   b) Present an outline of the procedural approach along with a flowchart showing the flow of control and subroutine structure of your Matlab code. Explain any design decisions you had to make. For example, how did you choose the camera parameters that were needed and how did you use those parameters? Include answers to the questions asked in Section 2.2. How did you choose the specific visualizations to convince us that your code is working? What mathematical equations were used to calculate the Euclidean distance statistics? Basically, explain at each step why you chose to do whatever you did. Be sure to document any deviations you made from the above

project descriptions (and why), or any additional functionality you added to increase robustness or generality or efficiency of the approach.

c) Experimental observations. What do you observe about the behavior of your program when you run it? Does it seem to work the way you think it should? Run the different portions of your code and show pictures of intermediate and final results that convince us that the program does what you think it does.

d) Quantitative Results: The report should contain a table with 5 columns, one for each of the statistical metrics, and 13 rows, one for each of the 12 joints plus one for the overall joint results. The closer the entries in this table are to zero, the more accurate your camera projection and 3D reconstruction algorithms. The Matlab code used for these calculations should also be included in the report. Also include the plot of total error across all frames of the sequence. Is the curve constant, or are there notable peaks and valleys?

e) Qualitative results: At least one image showing the input 3D skeleton and output 3D reconstructed skeleton on the same plot, with each skeleton being a different color. Also include images of the two camera projections, preferable overlaid over the color images extracted from the video files for cameras vue2 and vue4.

f) Algorithm Efficiency: Provide an image showing the Matlab profiler output. If you decide to make performance improvements, show the profiler results both before and after to convince us that the efficiency has been improved, and in what functions.

g) Show the two-view epipolar lines visualization discussed in 2.6 for at least one mocap pose, and tell how you generated the epipolar lines.

2) Turn in a running version of your main program and all subroutines in a single zip/tar archive file (e.g. all code/data in a single directory, then make a zip file of that directory).

a) Include a demo routine that can be invoked with no arguments and that loads the input data from the local directory and displays intermediate **and** final results showing the different portions of your program are working as intended.

b) We might be running the code ourselves on other input datasets, so include a routine that takes as input a joint dataset and that outputs the Euclidean distance between input and output with the statistics that are calculated for the report.

c) Include enough comments in your functions so that we have a clear understanding of what each section of code does. The more thought and effort you put in to demonstrating / illustrating in your written report that your code works correctly, the less likely we feel the need to poke around in your code, but in case we do, make your code and comments readable.