Motion Capture Strain Code Guide

Contents

[Introduction 2](#_Toc39574023)

[Code Input 2](#_Toc39574024)

[Data Processing 2](#_Toc39574025)

[Data Organization 2](#_Toc39574026)

[Strain Calculation 3](#_Toc39574027)

[Output Conditioning 4](#_Toc39574028)

[Strain Map Animation 4](#_Toc39574029)

[Time-Series Plot 5](#_Toc39574030)

[Output 5](#_Toc39574031)

[Appendix A 7](#_Toc39574032)

# Introduction

This document provides a brief outline of the code used in the BACPAC analysis of motion capture data from the lower back. The analysis finds the epidermal strain on the lower back between each point in a grid that is discussed in more detail below. Motion capture data was taken using a Qualysis motion capture system on campus of 30 participants who each performed 17 different functional exercises. We hope to use the results from this analysis to optimize the topographical placement of the strain gauges in the wearable sensor array we are building.

# Code Input

The code requires as its input a MATLAB Data (.mat) file generated by Qualysis from a motion capture. These data files are structured as follows:

In the main directory of the data file there are two variables which are accessed by the strain code: the sampling frequency used in the motion capture (called **FrameRate**) and the number of samples taken (called **Frames**).

The other variables accessed from the data file are the **Data** and **Labels** variables in the **Trajectories.Labeled** subdirectory. The **Labels** are used to reorder the **Data** from its default marker order (whatever that may be) to the order that the code recognizes them in; left to right, bottom to top. The code operates under the assumption that there are 36 markers physically placed on the subject’s back in the configuration shown in Figure 1.

These variables are stored locally in the code for use throughout, then the data file is closed.

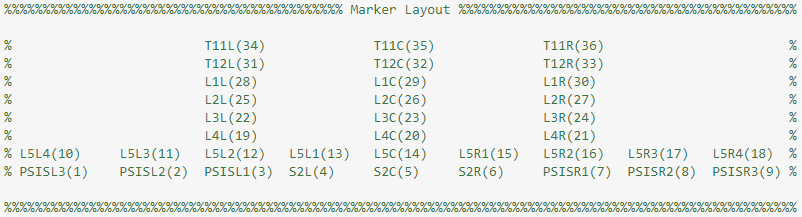


Figure 1. Marker layout. T11C, T12C, L1C, and so forth correspond to the locations of the palpated vertebrae of the subject's back.

# Data Processing

The data processing in the code is essentially divided into 3 parts: Data Organization, Strain Calculation, and Output Conditioning. Each will be discussed in basic detail in this section.

## Data Organization

Once the data has been read in, the trajectory data is reordered/compiled in a variable called **BIG**, which holds each marker’s xyz coordinate at each sample, housing this all in a 2-dimensional matrix ordered left-to-right, bottom-to-top such that marker PSISL3 (the first marker in the matrix) will have its x-coordinates stored in the first column of **BIG**, its y-coordinates stored in the second column, and its z-coordinates in the third, while the second marker, PSISL2, will have its coordinates stored in columns 4-6, and so on. Note that the positional data input has already been filtered to mitigate noise when processed in Qualysis prior to export.

After **BIG** has been compiled, the markers are organized into 20 squares as illustrated in Figure 2, each housing 4 triangles and 6 line segments as the connections between the vertices. Figure 3 illustrates the ordering of the triangles as used in the code. The squares are used for organizational purposes only in the code to keep track of which markers form the vertices of each triangle.

Since we don’t really care about the strains normal to the skin, the 3-d positional data is then flattened into 2-d space for simpler processing and stored in the 3-dimensional matrix **D2**. The first component of **D2** specifies the sample, the second component specifies the axis (x or y), and the third component specifies the marker. For a detailed explanation on how the data is flattened into 2-d space, see appendix A.

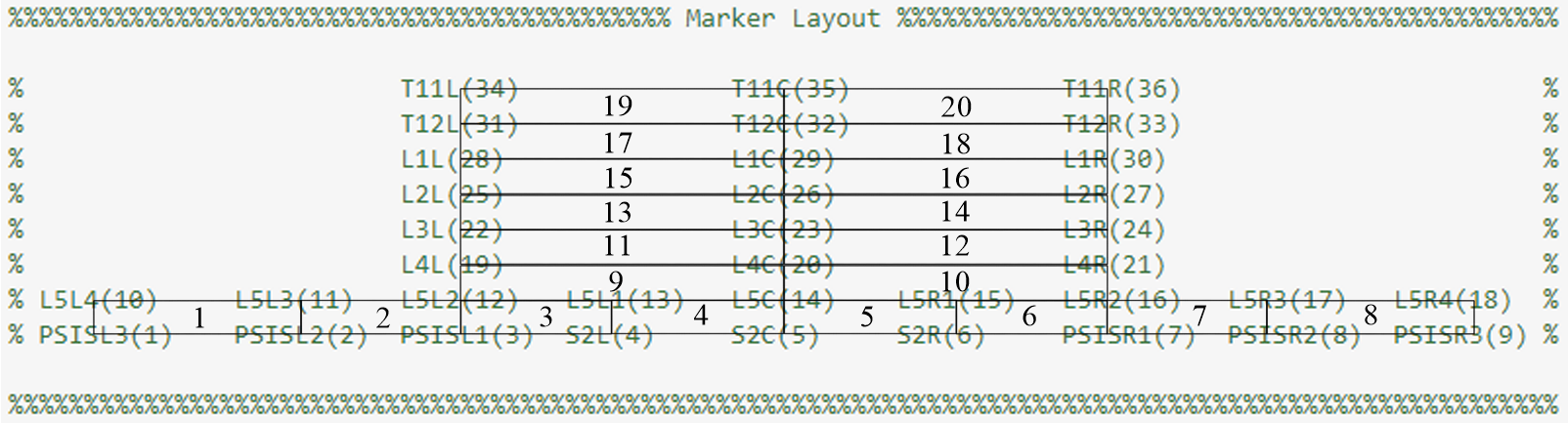


Figure 2. Marker layout with squares shown and ordered as done in the code.

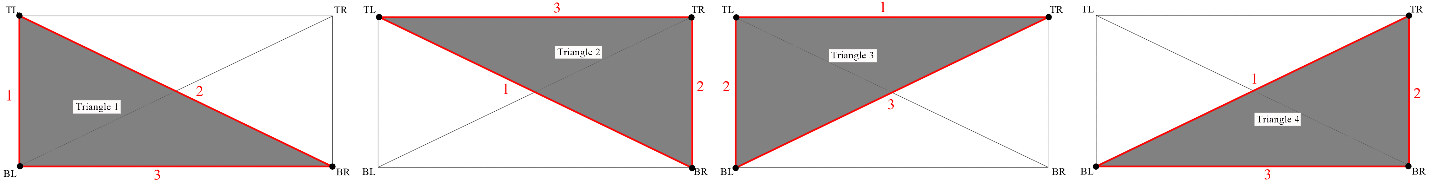


Figure 3. Illustration of the ordering of the triangles in each square as organized in the code. The ordering of segments in each triangle is also illustrated in red.

## Strain Calculation

Once the positional data has been translated into 2-d coordinates, the x-and-y distances between each vertex of each triangle of each square is calculated for the relaxed position and these distances are used to form the matrix **dXM** shown below:

ΔX1 corresponds to the x-component of the distance between the vertices of a segment, while ΔX2 corresponds to the y-component. The red subscripts denote which segment/pair of vertices in the triangle these values correspond to. The segment ordering for each triangle can be seen in Figure 3.

The matrix **ddS** holds the squared change in length for each segment of the triangle passed into it. The value *Δs2* is the squared distance *(x2- x1)2 + (y2- y1)2* between the vertices of the pertinent triangle segment. *Δs2* is calculated newly at each sample, while *Δs02* is calculated only once for the relaxed reference sample. **E** is a matrix housing the strain tensor for each triangle of each square at each sample and is the desired output. **E** is calculated using the mldivide function in MATLAB and the matrices **ddS** and **dXM**. Next, **E** is separated into 3 the matrices **Ex**, **Ey**, and **Exy** (one for each component of the strain tensor) which are then trimmed so that any strain values drastically outside of the realistic range (**Eij**>200%) are removed to help prevent output plots from being blown out by faulty data. The code then reports if there are holes in the data, whether due to this operation or not, by outputting a value greater than 0 to the variable **Holes**.

## Output Conditioning

The output consists of an animation showing a colormap of each component of the strain tensor across the back alongside a rendering of the marker positions in 3-d space over the course of the full trial, a figure showing the strain heatmaps captured at the moment of highest strain alongside a plot of the time series of each component of the strain for the triangle where the max strain occurred, and a MATLAB data file with several key variables. The animation and the figure both require some further processing of the data to provide proper inputs, as described under the next two headings.

### Strain Map Animation

In order to prepare the animation, which uses MATLAB’s patch function, two matrices (**xpatch** and **ypatch**) are generated to house a static, representational set of x and y positional data for each vertex of each triangle shown in the output. Next, three 82-element arrays (**Cx**, **Cy**, and **Cxy**) are generated to house one sample of the **Ex**, **Ey**, and **Exy** data for each of the 80 triangles, with two values at the end used to establish the maximum and minimum reference values for color mapping. This transfer of data from its already established location into a new variable is necessary to meet the formatting requirements of the patch function. Next, the colormap is defined to code the output so that higher tensile strains correspond to red, zero strains correspond to yellow, and compressive strains correspond to blue.

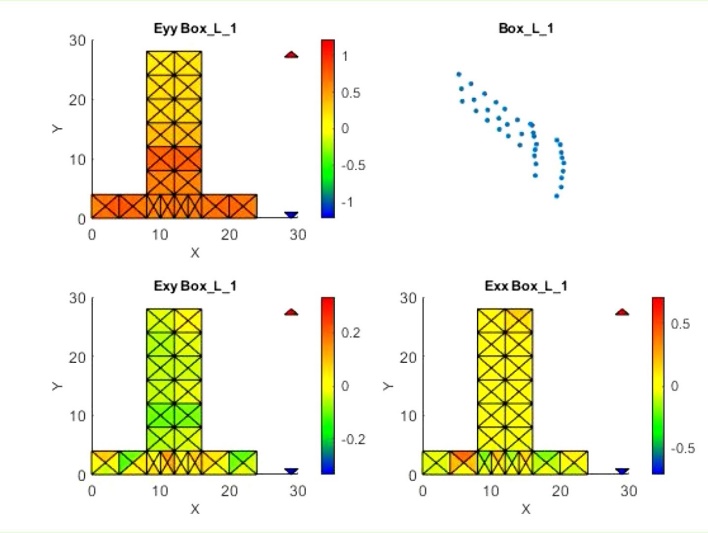


Figure 4. A frame from an output animation. Demonstrates the colormapping.

To generate the animation a 2x2 tiled layout is generated; the top left quadrant is occupied by the patch plot of the Cx for each triangle, the bottom left by Cxy, the bottom right by Cy, and the top right by the 3-d plot of the marker positions. Cx, Cy, Cxy, and the marker positions are updated at each sample and saved as a frame in the animation.

### Time-Series Plot

In order to prepare for the static figure depicting the strain time series for the triangle of maximum strain, the linear index of Ey corresponding to the maximum value in the matrix is extracted, then converted into a multi-dimensional index to identify which triangle of which square showed the maximum. The time-series for the strain tensor components in that triangle is then extracted and plotted in the top right tile of a 2x2 tiled figure, with the other three tiles being occupied by a snapshot of the strain colormaps at the moment of highest strain. See Figure 5 for illustration.

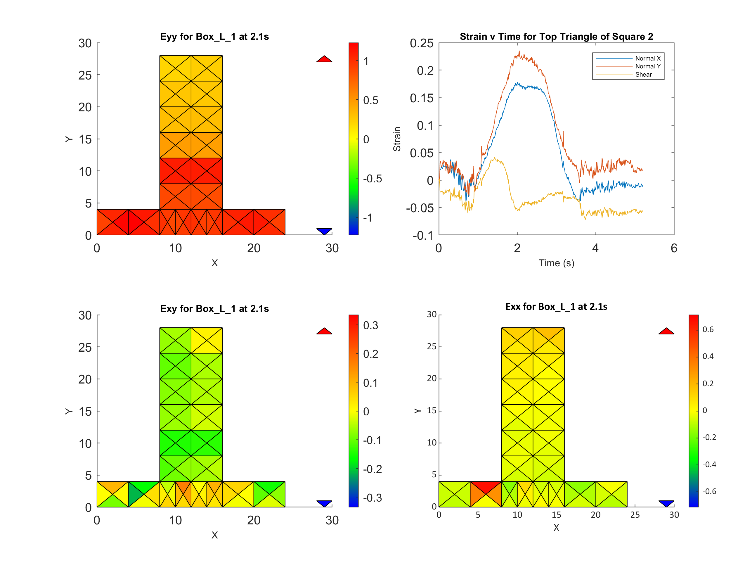


Figure 5. Strain tensor maximum-strain time-series output figure.

# Output

If the code executes without any errors, three files will be generated for each trial, as mentioned in the previous section. First, an animation of the strain tensor colormap; second, a .png file showing the maximum strain tensor colormap for the trial, along with the time-series for the strain tensor components in the triangle where the maximum strain occurred during the exercise; third, a MATLAB data file housing the following variables:

* Sampling frequency – Scalar (**Fs**)
* Number of samples – Scalar (**ns**)
* The time length of the exercise – Scalar (**P**)
* The raw strain tensor matrix – (**E**)
* The components of the strain tensor (**Ex**, **Ey**, and **Exy**)
* The maximum values for each component of the strain tensor (**ExMAX**, **EyMAX**, and **ExyMAX**)
* The time at which the maximum strain was seen in the trial (**TimeMaxStrain**)
* The time series of each component of the strain tensor for only the triangle where the maximum strain was seen (**StrX**, **StrY**, and **StrXY**)
* Whether or not trimming was necessary on the strain tensor (**Holes**)

If there were markers missing from the analysis for whatever reason, the animation and the figure will not be generated, and the generated data file will have as a prefix ‘MError\_’, for ‘Marker Error’. These trials will need to be analyzed separately with the code altered to account for the different triangle structures that result from missing markers.

# Appendix A

In order to flatten the 3-dimensional positional data for each marker into a 2-d space, markers A, B, C, and D from each square are used to define two planes and their axes. Define **AB** as the vector from marker A to marker B and **AC** as the vector from marker A to marker C, and so on, **N** as the normal vector defined by the cross product **AB** x **AC**, and **uX** and **uY1** and **uX** and **uY2** as the coordinate x- and y-axes for the planes. **uX** is the unit vector **AB**/|**AB**|, while **uY1** is defined by the unit vector **Y1**/|**Y1**| of the cross-product **Y1** = (**AB** x **AC**) x **AB** and **uY2** is defined as the unit vector **Y2**/|**Y2**| with **Y2** = (**AB** x **BD**) x **AB**. With these basic terms defined, the **D2** matrix may begin to be filled.

The first marker defined in **D2** is the very bottom left marker – the PSISL3 – and it is given the coordinates (0,0). This is the only time that marker A of a square will need to be specifically initialized, since each successive square will have its marker A defined by either marker B of the previous square or marker C of the square immediately beneath it. Marker B of each square is defined as the position of its marker A plus the vector <dX , dY>, where dX is the dot product **AB ∙ uX** and dY is **AB ∙ uY**. This preserves the distance between the markers but transforms the 3-d vectors into 2-d components for use in the 2-d space of **D2**. Marker C is defined either as having the coordinates of marker D of the previous square or, for the left-most square of each row, as the coordinate of the first marker plus the vector <dX , dY>, where dX is the dot product **AC ∙ uX** and dY is **AC ∙ uY1**. Marker D of each square is defined using **uY2** instead of **uY1**, as the 2-d coordinate of marker A plus the vector <dX , dY>, where dX is the dot product **AD ∙ uX** and dY is **AD ∙ uY2**. What this does is basically rotate the triangle formed by markers ABD about the axis **AB** to be in-plane with the triangle formed by markers ABC and thus make all markers in the square coplanar. Note that this does introduce a small bit of distortion as the distance between markers C and D will be somewhat lessened, an issue that is unavoidable when moving from 3 dimensions to 2. This is typically on the order of a 1.99% decrease.

This process is done for each square and each sample over the course of a trial.