Lab Tutorial 1: Some basic mocap tools

Copy over the *Tutorial 1* folder into your Documents folder or some place on the local hard drive. Within this folder you will see three folders, one called **PythonCode,** one called **ExampleData** and another called **Mokka**.

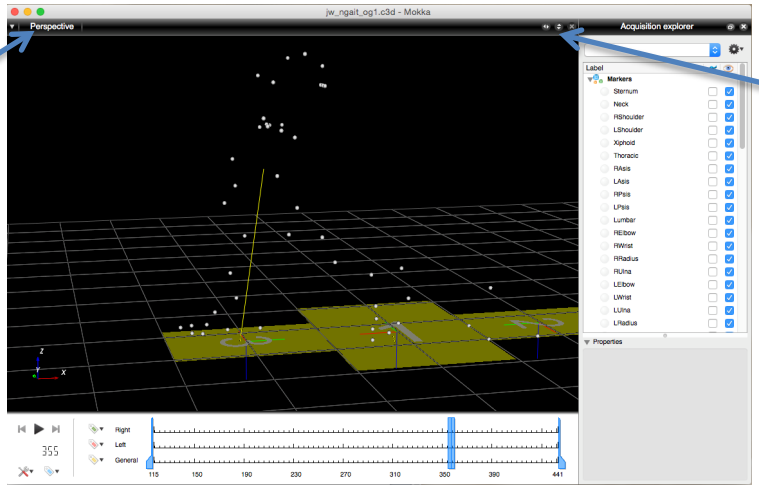
Task 1. Visualising mocap data.

We will use *Mokka* to visualize our mocap data. Mokka is an open source tool developed using the biomechanics tool kit (btk) and is available as a binary executable - for Windows and Mac. You can also access all of the underlying functions via a Matlab interface.

The two data types that we will explore are C3D and TRC files. The C3D file is a standard binary file type (https://www.c3d.org) that motion capture systems use to store marker and analog data (e.g. ground reaction forces from force plates and EMG). The C3D file also contains information about the laboratory setup, including the position and orientation of the force plates. The tracked marker trajectory file (.TRC) is used by *Motion Analysis Corp.* and is also the standard marker format for *OpenSim*. The TRC file is a simple tab-delimited text file containing the x,y,z coordinates of the markers. Let's explore some real data in Mokka…

[Navigate to the Mokka folder Double click on *Mokka.exe* to open Mokka]

Click here to change your view the view to 3D view or Chart (to graph the data simply drag the parameter from the Acquisition explorer [right window] into the empty chart)



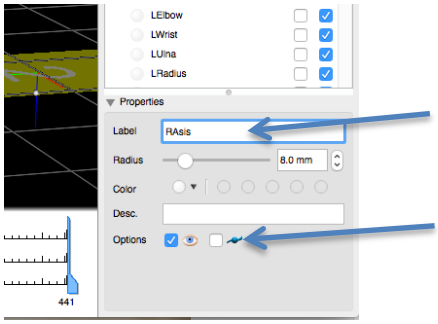
Click here to split the screen into more views

[Navigate to the *ExampleData* folder and click and drag the *jw_ngait_og1.c3d* file into the main Mokka window]

Within Mokka you can see all of the marker trajectories and analog data. The markers here have already been tracked and labeled and you can see their names on the right column of the screen.

[Click on a marker in the main window and it will be highlighted within the list]

This is really useful to determine the names and positions of markers, which can sometimes have non-intuitive names! If you wish, you can also rename markers by clicking on a marker (either in the window or list) and in the properties area (bottom right screen), you can simple type a new name for the marker label:



Type new label name here

Select this box if you want to visualize the marker trajectory

One of the main uses of Mokka is to export marker data or force data into a text file that is compatible with OpenSim.

To export marker data as a TRC file, select *File > Export > Motion Analysis Corp. > TRC file*.

To export force and EMG data as a text file, select *File > Export > Motion Analysis Corp. > ANC file*

Both of these files can easily be opened in Excel, Matlab, or Python for further processing or viewing. Mokka has a bunch of other features and you can explore these on your own. Play around with visualizing the analog data and try looking at the other C3D files in the ExampleData folder.

Task 2. Reading in a TRC file into Python.

Python is our favourite tool for pre-processing mocap data and people often use it to do their own kinematic and kinetic calculations. We will primarily use it for getting our data ready for OpenSim. The TRC file is a convenient file type to read and write (compared to the C3D file, which is a dogs breakfast), so let's explore some example code that enables you to read in a TRC file.

[Open up Jupyter Notebooks (via Anaconda Navigator) and navigate to the *PythonCode* folder that you copied over]

[Navigate to the *ReadTRCfile* folder, which is within the *PythonCode* folder]

Within this folder you will see one python function, one Jupyter notebook and an example TRC file.

[Open up the *ReadTRCfile.ipynb* notebook]

This is an example notebook that calls the read_trc.m function and places all of the marker data (`mkrData["Data"]`) and header information (`mkrData["Information"]`) into a Python dictionary structure.

```
mkr_data = read_trc()
```

This function will prompt you to select a TRC file from a directory and then store the data into a dictionary called `mkr_data`. The key here is to identify the names of the markers that you wish to pull out of the TRC file for analysis. They each have unique names, which have been captured from the header of the TRC file. To view all of the available markers, type

```
for key,value in mkr_data["Data"]["Markers"].items():
    print(key)
```

To pull out the vector of trajectories (x,y,z) of an individual marker, such as the Anterior Superior Iliac Spine of the pelvis, type:

```
LASI = mkr_data["Data"]["Markers"]["LAsis"]["All"];
```

Note that you can also get access to the individual x, y, or z coordinate data, which is also within the structure.

```
LASI_x = mkr_data["Data"]["Markers"]["LAsis"]["X"];
```

The rest of this script creates some plots to help visualize the 3D marker positions. It also calculates some new 'virtual' markers, such as the origin of the Pelvis segment, which is typically taken as the mid-point between the left and right anterior superior iliac spines.

[Run the *ReadTRCfile.ipynb* notebook and the *read_trc* function will prompt you to select a TRC file.]

Task 3. Filter some marker trajectories using a Butterworth low pass filter.

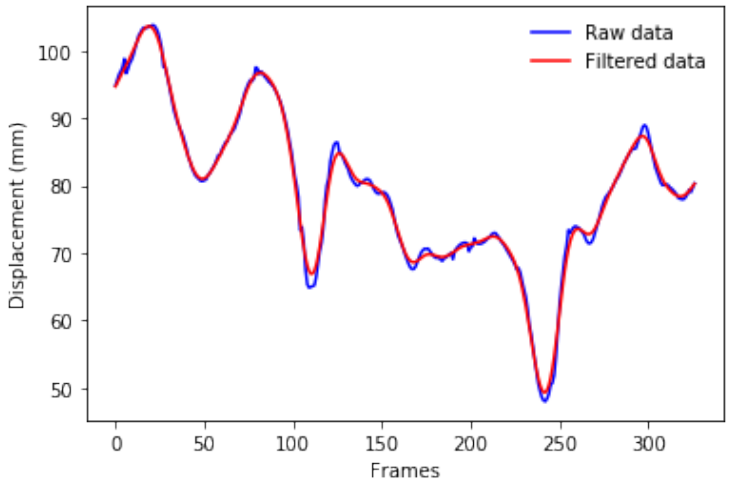
Most human movement occurs at low fundamental frequencies (<15 Hz), with the exception of impacts (e.g. when your foot hits the ground). If you plot raw marker trajectories from a mocap system you will notice some high frequency noise in the signal. This high frequency noise is due to inaccuracies in the marker reconstruction (camera calibration etc) and can also be due to soft tissue artifact (i.e. marker wobble). If we understand the frequency content of interest we can design a filter to reduce this noise and maintain the underlying signal of interest. A 4th order Butterworth filter is a common filter used in biomechanics to reduce noise. The selection of the cut-off frequency is open for debate. There are methods that can optimally select the appropriate cut-off frequency for each component of a marker trajectory, but most will apply the same filter to all of the data.

[open up the *ButterworthFilter.ipynb notebook*]

The key steps in this code are:

1. read in marker data from TRC file
2. pull out a marker of interest
3. define the capture rate (obtained within the header of the TRC file and is obtained using `data_rate = mkr_data["Information"]["DataRate"]`)
4. define the cut-off frequency
5. Use the scipy `butter` function to provide the filter coefficients, `a`, and `b`.
6. Use the scipy `filtfilt` function to pass the filter over the data
7. Plot the results of the raw and filtered data for comparison

[run the *ButterWorthFilter.ipynb* notebook. Play around with selecting different cut-off frequencies to see the effect it has on the marker trajectory]



Task 4. Calculate optimal hip joint centre.

Functional methods to define axes and joint centres can be very useful and are potentially more accurate than standard regression equations. This is particularly the case for the hip joint, where it is impossible to palpate local anatomical landmarks. The method that has been shown to be robust when it comes to finding the centre of rotation of a ball-in-socket joint is one developed by Gamage and Lasenby (2002). This example script will use this approach to define the x,y,z location of the hip joint centre for the right thigh relative to the pelvis. To use this approach all the marker trajectory data must be in the proximal, pelvis coordinate system.

Navigate to the *OptimalHipCentre* folder and you will see two Python notebooks, three Python functions, an example TRC file (*jw_hip_rstar.trc*) as well as the Gamage and Lasenby paper, which I am sure you will be excited to read! You can open up the demo TRC file (*jw_hip_rstar.trc*) in Mokka if you want to see what a 'star' trial looks like.

[Open the *FindHipJointCentre.ipynb* notebook]

The key steps in this code are:

1. Read in marker data from TRC file
2. Pull out a markers of interest. In this case they are the four markers on the pelvis and three markers on the right thigh.
3. Create unit vectors and a segment coordinate system of the pelvis. This uses the `segment_orientation_V1V3.py` function, which requires as input two vectors (the first one being the first defining axis, the second used to create cross products and an orthogonal coordin ate frame)
4. Transform all of the marker data into the pelvis coordinate system
5. Call the `calc_HJC.py` function

[Run the *FindHipJointCentre.ipynb* notebook and select the *jw_hip_rstar.trc* file]

The output of this function is a vector (`HJC_location`) giving the coordinates of the right hip joint centre (x,y,z) within the coordinate frame of the pelvis.

To satisfy your curiosity in making sure this worked, have a go at projecting the HJC_location back into the frame of the pelvis so you can visualize the right hip joint centre in the dynamic file. If you wanted to get fancy you could add the hip joint centre as a new marker in the TRC file and visualize it within Mokka!

Task 5. Coordinate Transformations.

Within the CoordinateTransforms folder are a few Python functions that take you through the calculations of defining a rotation matrix or transformation matrix. This code is nicely commented (thanks to my friend Jonas Rubenson for that!) and takes you through the transforms that are presented in *Craig. (1989). Introduction to Robotics.* For those of you that took a look at the `segment_orientation_V1V3` function from the previous task will see that these functions are doing similar things. The coord_rot.py and coord_tr.py have a bit more flexibility in the ability to define the first axis and which other axes are used to perform a cross-product. The other main difference is that these functions take marker trajectories (n= up to 5) to define vectors, whereas the `segment_orientation_V1V3` function takes two vectors as input.

[Run the CoordinateTransforms.ipynb function, which sets up a dummy set of markers and performs the rotations and transformations. Have a go at sending some real data (from the demo data provided) to these functions]