BodyMech 3.06.01 for U.L.E.M.A. Tutorial

The processing engine of U.L.E.M.A. is an improved version (for both speed and API) of BodyMech 3.06.01. It can be found inside *Engines / BodyMech3.06.01 / Core*. The new or improved functions can be found inside the folder *BMnewfunctions*.

In this document, you will learn how to create a Body model, pointer (stylus object) and segment reference frames files. The parts of code that do not natively belong to BodyMech 3.06.01 are depicted in blue.

Body model file creation:

Sample Body model files can be found inside the folder *BodyModels*. Each Body model file should have the following lines of code. This code chunk can be used as a template as well. The file doesn't consist of a function, but it is a script. After placing a new file into the folder *BodyModels*, it will be available into the U.L.E.M.A. GUI after restart.

```
% Define important app variables as global
BodyMechFuncHeader();

% Clear the main BODY structure
ClearBody();

% Create the list of markers that will have read from motion
capture files. Normally, this is the list of the technical (or
cluster markers) and anatomical markers. This list must contain
also the names of the pointer markers (4 in this example) used for
anatomical landmarks calibration:
LabelSequence{1} = 'MarkerName1';
LabelSequence{2} = 'MarkerName2';
…
LabelSequence{21} = 'P1';     % This is a pointer marker
LabelSequence{22} = 'P2';     % This is a pointer marker
LabelSequence{23} = 'P3';     % This is a pointer marker
LabelSequence{24} = 'P4';     % This is a pointer marker

% Save the markers names list to the BODY structure
PushLabelsToBody(LabelSequence);

% Create the list of indices (of LabelSequence) referring to
pointer markers. The vector must be named 'iP'
iP = [21, 22, 23, 24];

% Define segments names and the technical markers attached to each
one. Considering the first segment, its name is 'Segment 1', its
index in the BODY.SEGMENT structure is 1, and it is made of the
markers having indices 1,2,3,4 from the list LabelSequence.
Normally, the pointer segment doesn't have to be created.
CreateBodySegmentAdv('Segment 1', 1, [1,2,3,4]);
CreateBodySegmentAdv('Segment 2', 2, [5,6,7]);
…
```

```
% Define the anatomical landmarks for each segment, and if some
landmarks are common between 2 segments. Here the first landmark
of 'Segment 1' is called 'AL 1' and is shared with 'Segment 2';
this means that if the function DeleteUnusedSegments() is called
later for deleting 'Segment 1', this one will be deleted along
with its technical and anatomical landmarks BUT 'AL 1', since it
is necessary (shared with) 'Segment 2'.
BODY.SEGMENT(1).AnatomicalLandmark(1).Name = 'AL 1';
BODY.SEGMENT(1).AnatomicalLandmark(1).SharedWith = {'Segment 2'};
BODY.SEGMENT(1).AnatomicalLandmark(2).Name = 'AL 2';
BODY.SEGMENT(2).AnatomicalLandmark(1).Name = 'AL 3';
…

% Create and define joints. Here we define a joint named 'Joint 1'
, with index 1 in BODY.JOINT, the distal and proximal segments are
the ones respectively with index 2 and 1, the angles decomposition
sequence is Z, X, Y by using 'FirstSolution'. For 'Joint 2', the
proximal segment has index 0. In this case, the proximal segment
can be chose in the U.L.E.M.A. GUI by the user (see full doc). For
details about angles decomposition and solutions, see functions
CalculateJointKinematics.m and RotationMatrixToCardanicAngles.m
CreateBodyJoint('Joint 1', 1, [2 1], [Z X Y], 'FirstSolution');
CreateBodyJoint('Joint 2', 1, [2 0], [Z X Y], 'FirstSolution');
…

% Define angles for each joint. For the first joint (index 1), the
3 joint angles (named 'Angle_1', 'Angle_2', 'Angle_3') will be
multiplied by factors 1,-1,-1, so 'Angle_2' and 'Angle_3' can be
adjusted for sign. For the second joint, one factor is set to 0
and its name is empty. This angle will be ignored. Avoid spaces in
angle names.

SetAnglesMeaning(1, [1,-1,-1], {'Angle_1','Angle_2','Angle_3'});
SetAnglesMeaning(2, [1,0,-1], {'Angle_4','','Angle_5'});

% Define the laboratory reference frame and position
BODY.CONTEXT.MotionCaptureToLab = eye(4);
```

Segment reference frame definition file creation:

Sample segment reference frame definition files can be found inside the folder *AnatCalcs*. After placing a new file into the folder *AnatCalcs*, it will be available into the U.L.E.M.A. GUI after restart. The file is a function, and consists of the following blocks:

```
function MyRefFrameDefFile()

BodyMechFuncHeader;

% The code is relatively “free”.
% Here, inside BODY.SEGMENT(i).AnatomicalLandmark(j).Kinematics,
position for anatomical landmark j for segment i is available in
the form of 3 x Nf matrix, where Nf is the number of frames. You
```

```
can use your own functions for calculating segment reference
frames. After doing that, you have to assign the result to
BODY.SEGMENT(i).AnatomicalFrame.KinematicsPose in the form of a 4
x 4 x Nf array. The first two dimensions define the (affine)
transformation matrix for segment i. A transformation matrix T is
a compact form that embeds rotation matrix R and translation t
(column vector): T = [R, t; 0 0 0 1]; DJC data is inside
BODY.SEGMENT(i).AnatomicalLandmark(j), where segment i is the
proximal segment (see subfields Name and Kinematics). Functional
axis data is inside BODY.SEGMENT(i).FunctionalAxis(j), where
segment i is the proximal segment (see subfield Name). Subfields
Marker1.Kinematics and Marker2.Kinematics contain position of two
virtual markers lying along the functional axis (this one pointing
towards Marker1). These two markers are 2 cm distant between each
other, and the midway position is the optimal position of the
functional axis.

end
```

Pointer (stylus) reference frame definition file creation:

Sample pointer reference frame definition files can be found inside the folder *Pointers*. After placing a new file into the folder *Pointers*, it will be available into the U.L.E.M.A. GUI after restart. The file doesn't consist of a function, but it is a script, and consists of the following blocks:

```
% Define a custom structure containing stylus geometric parameters
to be used later (into the StylusTipReconstructor.m in this
example)
StylusGeom.myParam1 = 10;          # some stylus data...
StylusGeom.myParam2 = [30 12 45];  # ...and some other

% Create the stylus named 'Stylus', composed by 4 markers whose
indices are stored into iP, by using the custom function 'Pointer'
(defined in the folder Pointers/Geometry) that will use geometric
parameters in StylusGeom
CreateStylus('Stylus',4,[iP(1) iP(2) iP(3) iP(4)],'MyTipGetter',
StylusGeom);
```

Inside the folder *Pointers / Geometry* we can write the code performing the pointer tip reconstruction. In the example above, this function is *MyTipGetter.m*:

```
function StylusTip = MyTipGetter(InputData, StylusGeom)

% 'InputData' contains pointer markers global position at a
certain time instant. It is a 3 x N matrix, where N is the number
of pointer markers (in this case, 4). 'StylusGeom' is the same
structure created in the file that is into the folder 'Pointers'.
'StylusTip' must be a 3-elements vector containing the global
coordinates of the pointer tip.

end
```