here]]. the file types and contents are briefly described below: **marker-based files:** the tutorial data set includes one subject model file, a static standing file, and two walking trials for marker-based motion capture. * **static file (sample_static.c3d):** standing trial from marker-based data set * **model file (sample_model.mdh):** model template for marker-based subject * **motion files (sample_walking*.c3d):** treadmill walking trials from marker-based data set **markerless files:** the markerless and marker-based trials were recorded simultaneously, and the provided markerless file from theia (https://www.theiamarkerless.ca/) is for the first trial of this subject. * **markerless motion file (sample_markerless):** theia markerless motion file. visual3d automatically assigns a model and static trial to this file when loaded in the workspace. * **meta-commands (_.v3m):** copy and paste all included meta commands into the plugins>meta_commands folder in your visual3d program files. restart your visual3d application. see more on meta-commands [[index.php?title=pipeline_commands:_meta_commands&action=edit&redlink=1|**here**]]. ===== marker-based vs markerless landmarks ===== to prepare the subject models for finding the mos in this tutorial, we need to apply foot landmarks for defining the bos during the trial. ==== to apply landmarks to the marker-based data: ==== * **1. open a new visual3d workspace** * **2. run “model_landmarks_for_mos” meta-command** * open the pipelines dialog * open the meta commands subfolder * select **model_landmarks_for_mos** and run pipeline you will first be prompted to provide the name of the targets for the following markers: left heel (l_heel), right heel (r_heel), left mid toes (l_met23), right mid toes (r_met23), left fifth metatarsal (l_met5), right fifth metatarsal (l_met5), left first metatarsal (l_met1), and right first metatarsal (l_met1). **for this tutorial**, the marker-based data targets have the same names as the parameters they are being set to, and in the dialog you can type these same names as the value entry. this is shown in the figure below. **if** you are applying this method to a **different model**, make sure to check what your target signals are named for these markers. next you will be prompted to select your model template file and static file. navigate to the location of the folder downloaded for this tutorial, and select **sample_model.mdh** for the model template and **sample_static.c3d** for the static file (or your own file names if you are not using tutorial data). * **3. load motion files and apply model** * open files: **sample_waling1.c3d** and **sample_waling2.c3d** for the marker-based motion files. * select model > assign model to motion files all motion files in signals and events should now have a model applied, with blue markers on the feet showing the applied landmarks. in the figure, landmarks have been checked to show you which have been added from the pipeline, you do not need to do this yourself. inputs.png landmarks.png ==== to apply landmarks to markeless data: ==== * **1. open a new visual3d workspace** * **2. load markerless motion file** open sample_markerless.c3d for the markerless data. * **3. run “model_landmarks_for_mos_markerless” meta-command** * open the pipelines dialog * open the meta commands subfolder * select **model_landmarks_for_mos_markerless** and run pipeline for markerless data, we do not need to select a static or model template file, as the model is automatically generated by visual3d. once we run the pipeline, the foot landmarks will appear in the workspace (see figure). landmarks_markerless.png ====== margin of stability: original definition ====== at this point in the tutorial, we have loaded our motion files for our sample subject and applied landmarks to the feet that we will use to define bos. to derive the mos as defined in the literature, we will use the following formula: <code> mos = bos -xcom </code> where bos is the area defined by the bounds of the feet during ground contact, and xcom is the extrapolated com: <code> xcom = com + vcom/sqrt(g/l) [3] </code> we will first walk through how we define the bos and xcom before calculating the mos. we will only describe bits of the pipelines and underlying meta-commands throughout this tutorial. if you want more detail about the functionality of these commands, the provided .v3m and .v3s files in the tutorial zip are commented to guide you through. **from this point forward, the process is the same for both markerless and marker-based data!** ===== extrapolated center of gravity ===== we begin this process in the pipeline **find_mos_original.v3s**, where we first use the compute_model_based_data command to find the center of gravity position and velocity of the model

over the trial. <code>
!==========
========== ! 1. define and calculate center of gravity
!==========
========== </code> <code> compute_model_based_data /result_name=cog
/subject_tag=all_subjects /function=model_cog /segment= /reference_segment= ; </code> <code>
compute_model_based_data /result_name=cog_velocity /subject_tag=all_subjects
/function=model_cog_velocity /segment= /reference_segment= ; </code> the extrapolated center of gravity uses an expression in visual3d: <code> evaluate_expression
/expression=extrapolated_cog(current_signal,1) /signal_types=link_model_based+link_model_based
/signal_folder=original+original /signal_names=cog+cog_velocity ! /result_types=derived
/result_folders=center_of_gravity /result_name=xcog ! /apply_as_suffix_to_signal_name=false ;
</code> ===== base of support ===== bossingle.png bosdouble.png the bos uses the ap and ml bounds of the feet from the landmarks we applied to the model. at each instant in the motion trial, the base of support is defined by four boundaries in single or double stance. the bounds are defined using evaluate_expression through the foot landmarks. an example of setting the ml bounds of the right foot is seen here: <code>
!==========
========== ! 2. set medial/lateral bounds of feet
!==========
========== !outer bound right evaluate_expression /expression=landmark::original::rt_met5
/signal_types=landmark /signal_folder=original !/signal_names= /signal_components=x
/result_types=derived /result_folders=bos_ml /result_name=lateral_bound_right !
/apply_as_suffix_to_signal_name=false ; </code> <code> ! inner bound right evaluate_expression
/expression=landmark::original::rt_met1 /signal_types=landmark /signal_folder=original
!/signal_names= /signal_components=x /result_types=derived /result_folders=bos_ml
/result_name=medial_bound_right ! /apply_as_suffix_to_signal_name=false ; </code> after running the **find_mos_original.v3s** pipeline, we can explore the derived signal subfolders to see the bos outcomes. in the **bos_ml** for example, we have plotted here the **lateral_bound_left** and **medial_bound_left** over time (x components since x-axis is in the ml direction for the global coordinate system). we have also plotted the **xcom** signal x-component from the **center_of_gravity** folder. we can see the **xcom** crosses over the medial bound of the left foot several times throughout this trial, so we can expect to find some negative mos values if the xcom is swaying in and out of the stable boundary. however, we haven’t considered single and double support events to know when these bounds should be considered in our mos calculations. to trim our ml and ap bounds to reflect single and double support times, we use two meta-commands called **bos_bilateral_single_support** and **bos_bilateral_double_support**. below is an example of calling these meta-commands from the tutorial pipeline: bounds_untrimmed.png single support: <code>
!==========
========== ! 4. trim all bos values for single stance
!==========
========== ! medial/lateral bos_bilateral_single_support /type=derived /folder=bos_ml
/signal_left=medial_bound_left /signal_right=medial_bound_right ; </code> double support: <code>
!==========
========== ! 5. trim bos values for double stance
!==========
========== ! anterior/posterior bos_bilateral_double_support /type=derived /folder=bos_ap
/signal_left=posterior_bound_left /signal_right=posterior_bound_right /signal_direction="posterior" ;
</code> bostrimmed.png looking now at our **bos_ml** derived folder, we can plot the signals: **lateral_bound_left_ds**, **lateral_bound_left_ss**, **medial_bound_left_ds**,

medial_bound_left_ss, and **xcom**. here we see that the left bounds of the feet are now cut to only have data when the left foot is on the ground. i have also highlighted the events **lto** (blue) and **lhs** (red), which shows that between **lto** and **lhs**, there are no bounds for the left foot. ===== margin of stability ===== to compute the margin of stability, we simply find the distance of the xcom position to each bound defining the bos through the subtract_signals command: example for mos double support to the left anterior bound: <code> !========== ========== ! 7. calculate margin of stability for double stance !========== ========== </code> <code> subtract_signals /signal_types=derived+derived /signal_folder=bos_ap+center_of_gravity /signal_names=anterior_bound_left_ds+xcog /result_name=anterior_left_ds /result_folder=mos_ap /component_sequence=y,y ; </code> if we plot the anterior and posterior mos of the left and right feet during single and double support, we see the following figure: ap_mos.png this is a cluttered plot with 8 different signals, so we have written a meta-command to return the smallest mos (distance to closest boundary) between the ap and ml bounds at each frame. typically, mos is reported as a single value as the distance of the xcom to the closest bound. for example, in this following command we are feeding in the anterior left and posterior right bounds that define the ap bos during the event sequence rdouble, to the meta-command called **closest_bound.v3m**. this will take the smallest mos between the two signals at each frame of the sequence. <code> !========== rdouble condition ========== conditional_statement /iteration_parameter_name=3 /expression="&::seq&" == "rdouble"; ; closest_bound /frames=::frame_ind /sig1=anterior_left_ds /sig2=posterior_right_ds /sequence=lsingle+rsingle+ldouble /direction=ap ; closest_bound /frames=::frame_ind /sig1=lateral_right_ds /sig2=lateral_left_ds /sequence=lsingle+rsingle+ldouble /direction=ml ; conditional_statement_end /iteration_parameter_name=3 ; </code> the following plot in the mos output shows the final ap mos values for the original interpretation. however, by just looking at single values at each point throughout the trial, which is typically shown in the literature, we don’t know the direction of instability. we know that a negative mos is considered “unstable”, but we don’t know if they are in front or behind the bos, or on either side in the ml direction. we propose a new way to define the mos to improve the classification of stability and provide the direction, which we will show in the following sections of this tutorial. ap_mos_2.png ====== margin of stability: new definition ====== newbos.png this section will use the pipeline: **find_mos_new.v3s** this pipeline builds off of the previous section. you must run **find_mos_original.v3s** first! we took a slightly different approach to defining the mos in this second pipeline. here we build a polygon with four vertices based on the foot landmarks, which lies on the ground plane as our bos area. we then projected the xcom in the direction of gravity onto the ground to determine if the xcom lies within the bos (stable) or not (unstable). new mos definition: here we consider a new convention for the mos values that provide more information about where the xcom lies relative to the bos. if the xcom is within the polygon, we set the mos to 0 and say this is stable. if the xcom is beyond the bos anteriorly or to the right, the mos is a positive value. if the xcom is beyond the bos posteriorly or to the left, the mos is a negative value. ===== projected xcom ===== to project the xcom onto the ground plane, we first found the direction of gravity for the current .c3d file: <code> !========== ========== ! 1. project xcom onto ground plane, here we find the direction of gravity first !========== ========== </code> <code> ! find direction of gravity evaluate_expression /expression=gravity() ! /signal_types= ! /signal_folder=original ! /signal_names= ! /signal_components= /result_types=derived /result_folders=center_of_gravity /result_name=gravity_vector ! /apply_as_suffix_to_signal_name=false ; </code> we then found the unit vector and projected the xcom: <code> evaluate_expression

/expression=project_point_on_plane(derived::center_of_gravity::gravity_unit_vector,derived::center_of_gravity::xcog) ! /signal_types= ! /signal_folder=original ! /signal_names= ! /signal_components= /result_types=derived /result_folders=center_of_gravity /result_name=xcom_projected ! /apply_as_suffix_to_signal_name=false ; </code> ===== base of support ===== the bos was defined using vertices with foot landmarks. here we show the left foot vertices as an example: <code> ! ==== make vertices for left foot ========== metric_explicit /result_metric_folder=vertex /result_metric_name=lv1 /metric_value=vector(landmark::original::lt_met1::x,landmark::original::ltoes_distal::y,0) ; metric_explicit /result_metric_folder=vertex /result_metric_name=lv2 /metric_value=vector(landmark::original::lt_met5::x,landmark::original::ltoes_distal::y,0) ; metric_explicit /result_metric_folder=vertex /result_metric_name=lv3 /metric_value=vector(landmark::original::lt_met5::x,landmark::original::lt_heel::y,0) ; metric_explicit /result_metric_folder=vertex /result_metric_name=lv4 /metric_value=vector(landmark::original::lt_met1::x,landmark::original::lt_heel::y,0) ; </code> ===== margin of stability ===== to re-define the mos with the new convention, we used the following procedure. **main pipeline**: * define bos vertices for current event sequence * determine if the projected xcom lands within bos polygon **meta-command (is_in_bounds.v3m)**: * if is inside polygon (returned value of 1), set the mos to 0 for that frame * if not inside polygon (returned value of 0), look at bos bounds and find where the xcom is closest to * set mos to closest bound outside the bos * if anterior or to the right of the bos, make mos a positive value * if posterior or to the left of the bos, make mos a negative value an example of following this process for the rsingle event sequence: <code> !========== rsingle condition========== ! conditional for rsingle - use correct vertices from listed earlier conditional_statement /iteration_parameter_name=1 /expression="&::seq&" == "rsingle"; ; </code> <code> ! is xcom in polygon? evaluate_expression /expression=is_point_inside_polygon(derived::center_of_gravity::xcom_projected,metric::vertex::rv1,metric::vertex::rv2,metric::vertex::rv3,metric::vertex::rv4) /result_types=derived /result_folder=mos_new /result_name=inside_rsingle ; </code> <code> !ap is_in_bounds /xcom=derived::center_of_gravity::xcom_projected /upper_bound=anterior_right_ss /lower_bound=posterior_right_ss /frames=::frame_ind /sig=inside_rsingle /sequence=rdouble+lsingle+ldouble /direction=ap ; !ml is_in_bounds /xcom=derived::center_of_gravity::xcom_projected /upper_bound=lateral_right_ss /lower_bound=medial_right_ss /frames=::frame_ind /sig=inside_rsingle /sequence=rdouble+lsingle+ldouble /direction=ml ; </code> after running the **find_mos_new.v3s** pipeline, we can plot the **derived::mos_original::mos_ap**, and **derived::mos_new::mos_ap** for comparison. here we can see that any time the original signal was positive/"stable", it is now 0 in the new signal. when the subject's xcom is anterior of the ap bos, the mos is positive, and is negative when the xcom is posterior of the ap bos. newmos.png ====== further mos analysis ====== normalizedmossigs.png in the **mos_new** derived folder, you will see several normalized signals for each left or right stride of the selected walking trial. this was done to plot several strides on top of eachother. here we only have a few strides, but analyzing these mos signals normalized between heel strikes may be a more useful way to assess a subject's stability. if we consider that the mos is being classified as 'unstable' for the majority of the stride, but we know the subject walked normally and did not fall, we might be able to detect moments of actual instability through outlier mos strides. normalizedmosplot.png ===== inspect3d ===== time normalization can also be applied to data in inspect3d when groups are defined between event labels. here we show an example of the mos output plotted for the ap and ml directions of one subject with 8 walking trials. to learn how to load workspaces, define groups, and create figures in inspect3d, see [[inspect3d:tutorials:overview|**here**]]. i3dmos.png i3dmos2.png ====== references ======

[1] watson, f. et al. (2021). use of the margin of stability to quantify stability in pathologic gait – a

qualitative systematic review. bmc musculoskeletal disorders, 22 (1). doi:10.1186/s12891-021-04466-4 [2] dingwell, j. b., cusumano, j. p., cavanagh, p. r., & sternad, d. (2001). local dynamic stability versus kinematic variability of continuous overground and treadmill walking. journal of biomechanical engineering, 123(1), 27–32. https://doi.org/10.1115/1.1336798 [3] hof, a. l., gazendam, m. g., & sinke, w. e. (2005). the condition for dynamic stability. journal of biomechanics, 38(1), 1–8. https://doi.org/10.1016/j.jbiomech.2004.03.025
}}}}}}}}}}}}}}}}}}}}}}}}}}