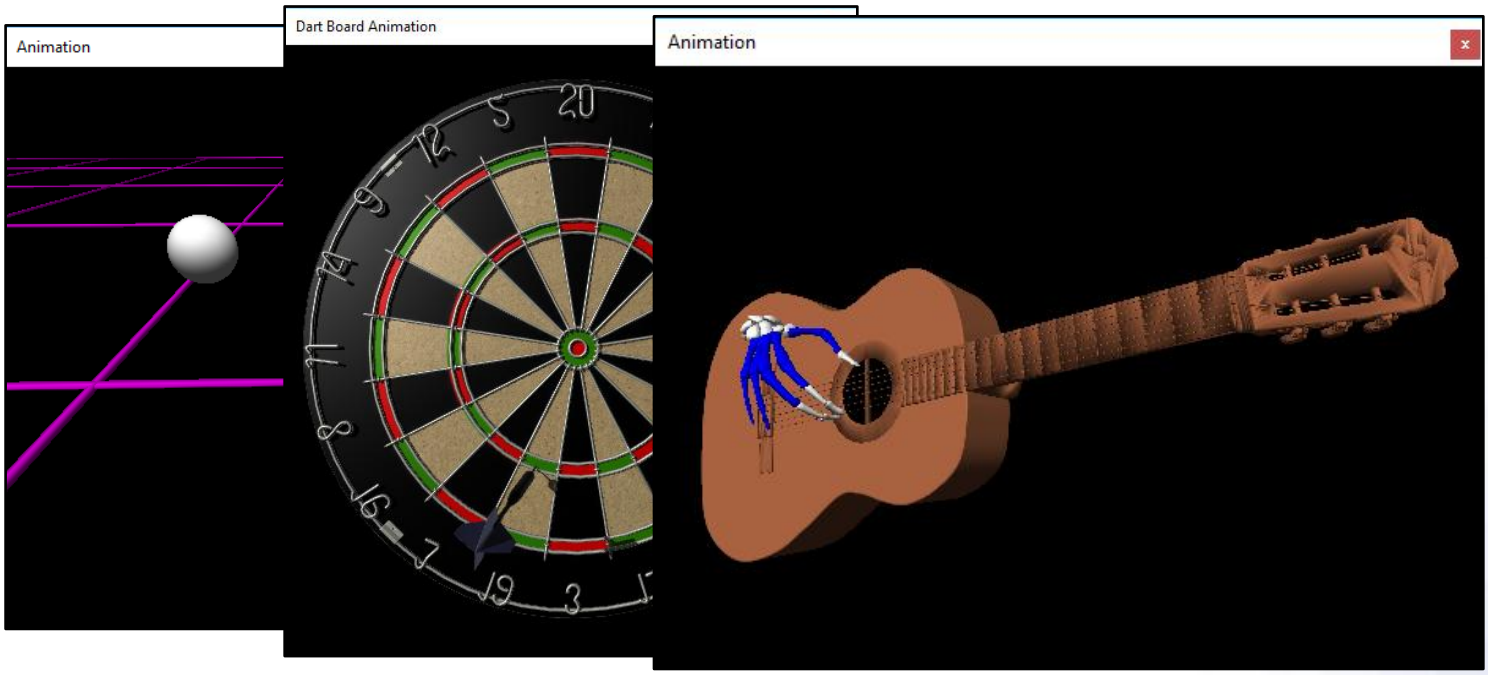
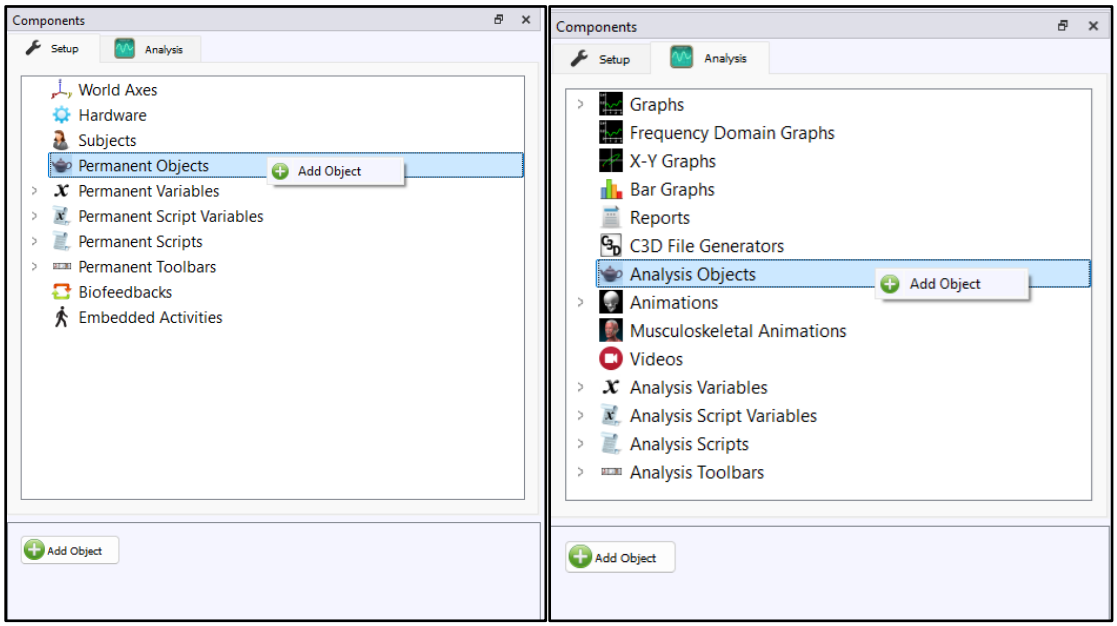
The MotionMonitor xGen Software Guide: Adding Objects to the Animation Window

This document reviews the process of adding objects to the Animation window in The MotionMonitor xGen. An object can represent an item from the real world, such as a golf club, computer screen, or table surface. They can be purely used as a visualization or as a means for engaging with a participant or to represent data in a biofeedback paradigm. Objects can remain static or receive their positioning and orientation from hardware devices. In instances where they receive data from the hardware, an object's size, shape, position, orientation, and color can all be controlled for a more customizable visualization. Data from an object, including its position, orientation and in some instances, intersection with other items can also be used in analyses.

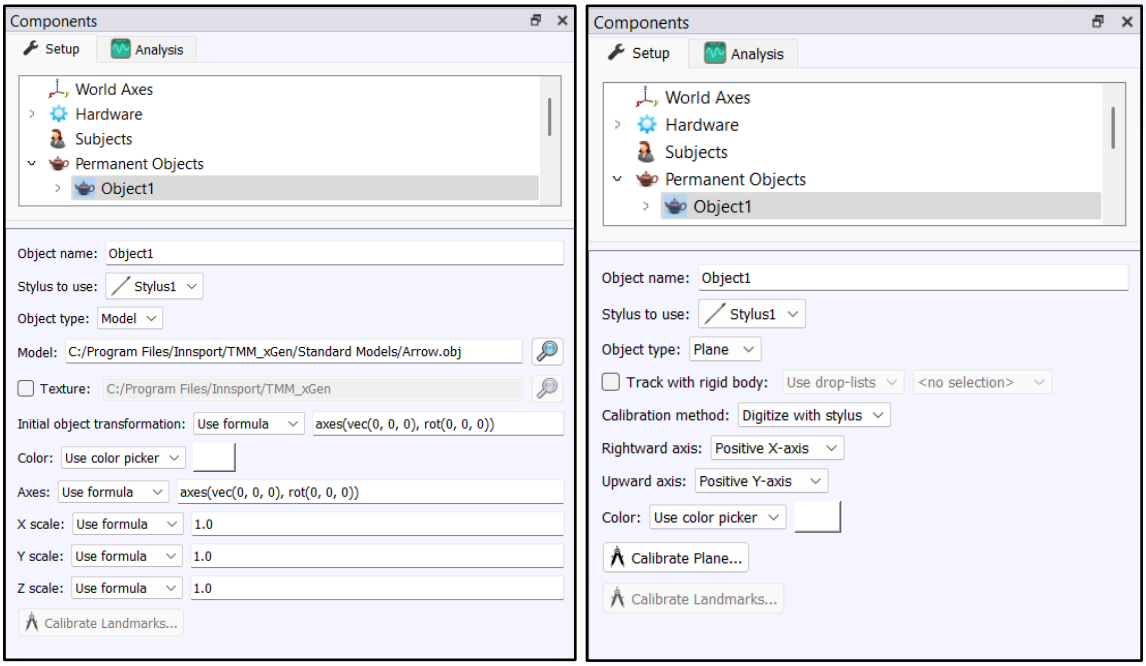
An object file (.obj) is an image file that contains data for rendering a three-dimensional object. The file itself does not contain any units of measure, but rather includes vertices relative to some arbitrary reference frame and polygon faces that are used for rendering the object. Objects can take any form, from basic Geometric shapes such as a sphere or cube, to more complex shapes such as a dart board or guitar. Optional texture files, which are images that can be overlaid on object files, can also be applied to enhance the visualization in the Musculoskeletal Animation within The MotionMonitor xGen. A few examples of objects are displayed below.



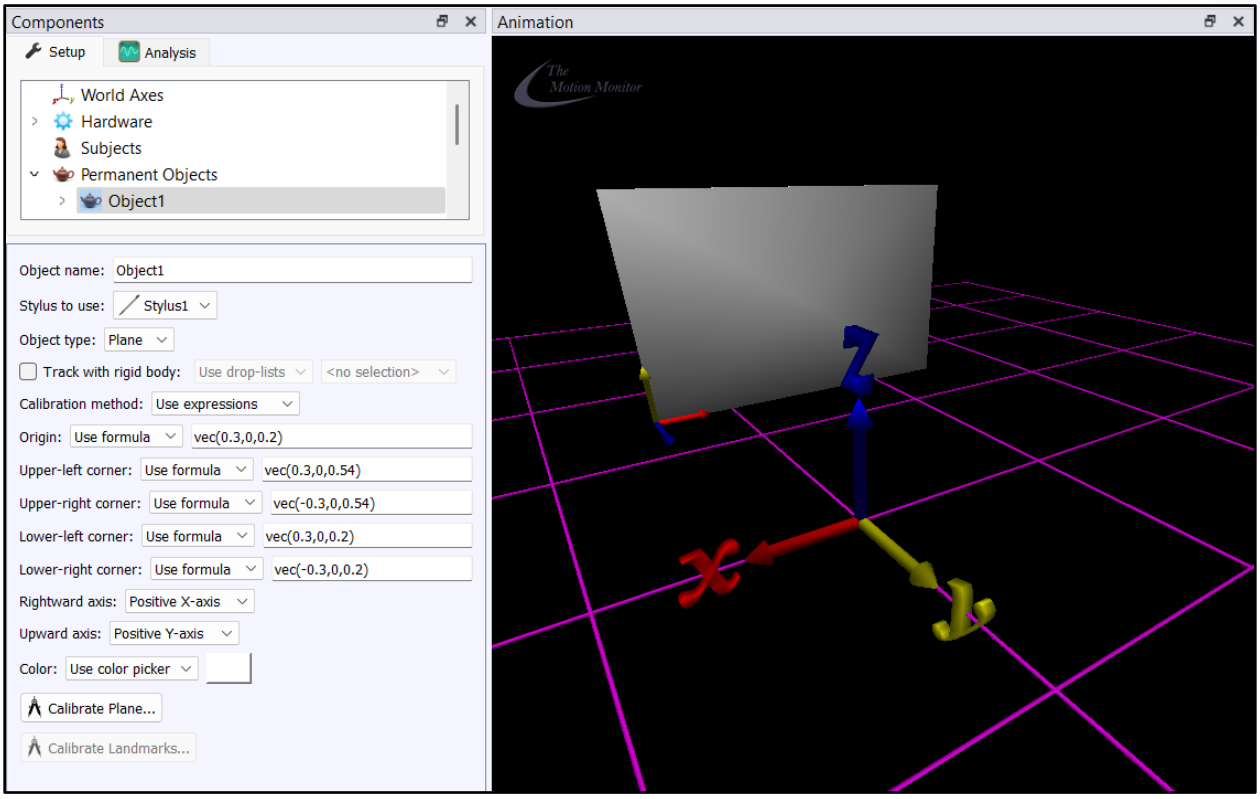
Although objects can be added to an Activity, they are more commonly added from the Live Workspace before data is captured. When added to an Activity, objects are used more as just a visualization since the object and their landmarks cannot be calibrated. Objects can be added as Permanent Objects through the Setup Components tab or as Analysis Objects through the Analysis Components tab. While they share the same capabilities, only Analysis Objects can be applied through Analyses. This also means that any Analysis Objects will be overwritten when an Analysis file is loaded. For this reason, when digitizing and tracking landmarks for an object or when using an object to generate data, it's recommended to configure the object as a Permanent Object to ensure that loading an Analysis doesn't overwrite the data associated with the object. Whether adding an object in the Setup or Analysis Components tab, the process will be the same. Go to the Objects node in the Setup or Analysis tab of the Components window and Add an Object from the Add button in the parameters panel at the bottom of the Components window or by right clicking the Objects node.



When added, the “Object name” can be specified for the object. This will be the name used throughout the software when referencing the object. When defining an object, the “Object type” can be selected as either a “Model” (.obj file) or a “Plane”.

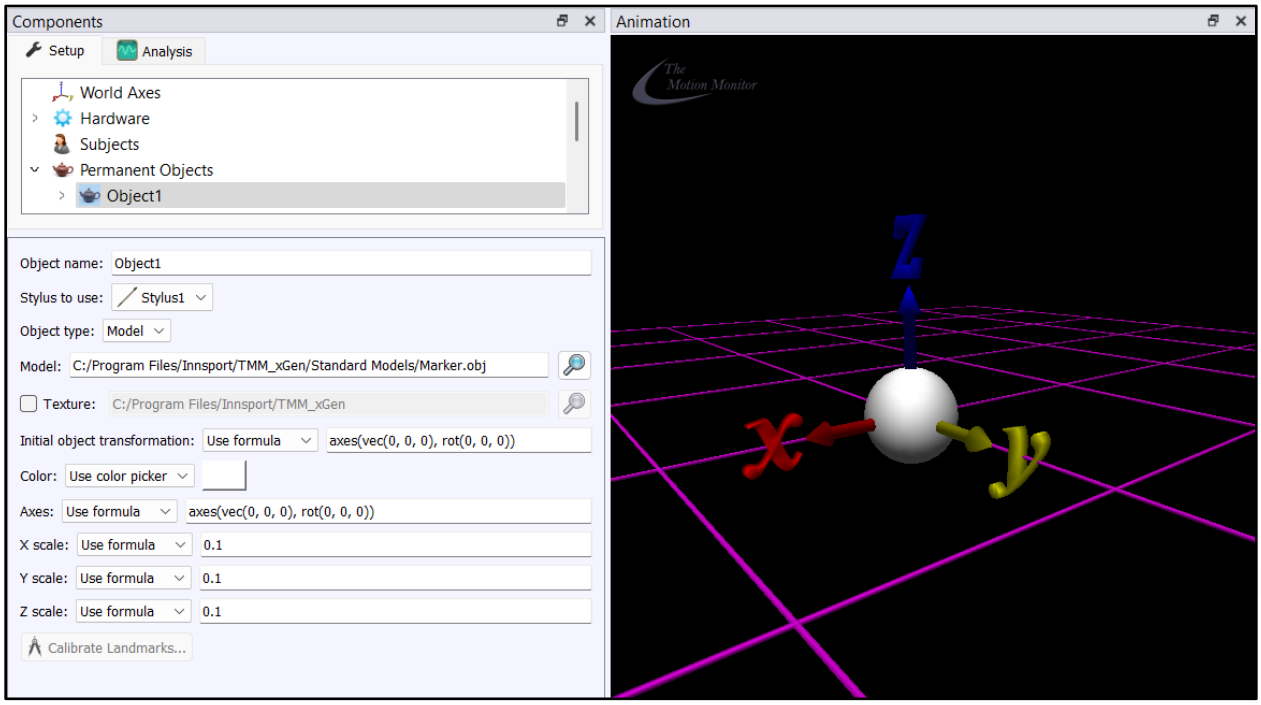


A plane would typically be selected when defining a floor, wall, table, monitor screen or other planar surface for an item within the measurement volume. A plane must be configured before data are recorded. Planes can have a fixed location or be dynamically tracked with a rigid body, and can be calibrated using a digitization method, with a stylus, or using variable expressions to define their origin and upper-left, upper-right, lower-left and lower-right corner locations. The local coordinate system axis directions and color for the plane can be selected and these can be modified after data have been recorded. The "Calibrate Plane" button will calibrate the plane based on the selected parameters.



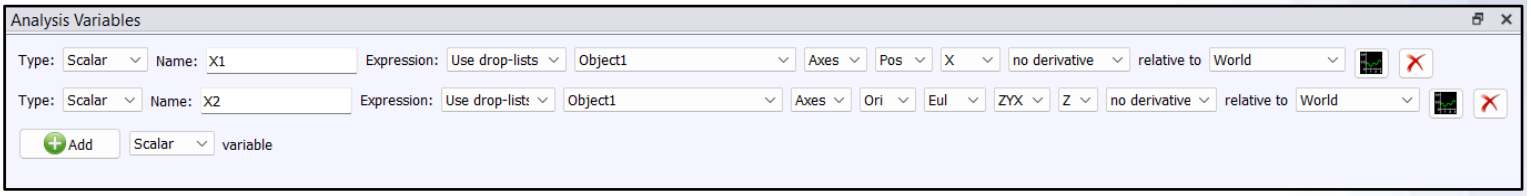
In the example shown above, a plane was defined to represent a 60cm x 34cm monitor screen whose origin is located at the lower left-hand corner of the screen.

A model can be selected to define any other type of object and can have either a fixed or dynamic position and orientation. A "Texture" or image file and "Color" can be selected for the model; however, a texture will only apply to objects displayed in the Musculoskeletal Animation window. When applied, a Texture will override the Color in the Musculoskeletal Animation window. For more information on the Musculoskeletal Animation window and to get it released on your system, please contact your client support engineer or support@TheMotionMonitor.com. An "Axes" (position and orientation) needs to be provided to control the positioning and movement of the object. For instance, this could be the sensor from a hardware device, rigid body from a rigid body collection, or Axes variable defined using the axes() operator. The scaling can also be applied in the X, Y and Z directions of the model to achieve the appropriate size for the object. For more information on defining the Axes and scaling of models, please see Appendix A. Additionally, an "Initial Object Transformation" in the form of an Axes variable type (position and orientation) can be specified to define an offset, in the reference frame of the Axes assigned to the object. The offset is applied to the mesh file points prior to tracking the object with the assigned Axes. This offset can be applied to initially get the object into the desired position and orientation before tracking. The use of this field will also be described in Appendix A.

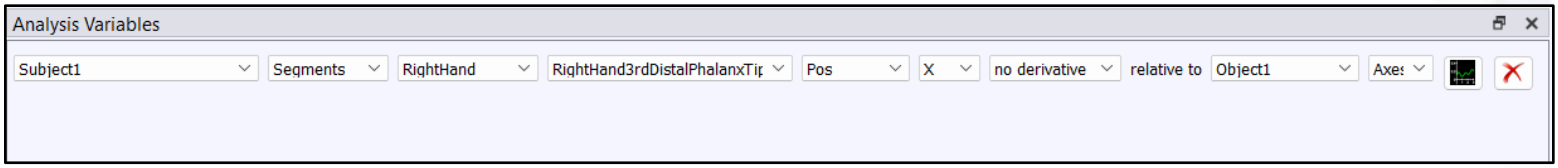


In the example shown above, a marker (sphere) was selected, positioned at the origin, and scaled to have a radius of 10cm. The default radius of the Marker.obj file was 1, so it was scaled by a factor of 0.1 in each direction to accomplish this.

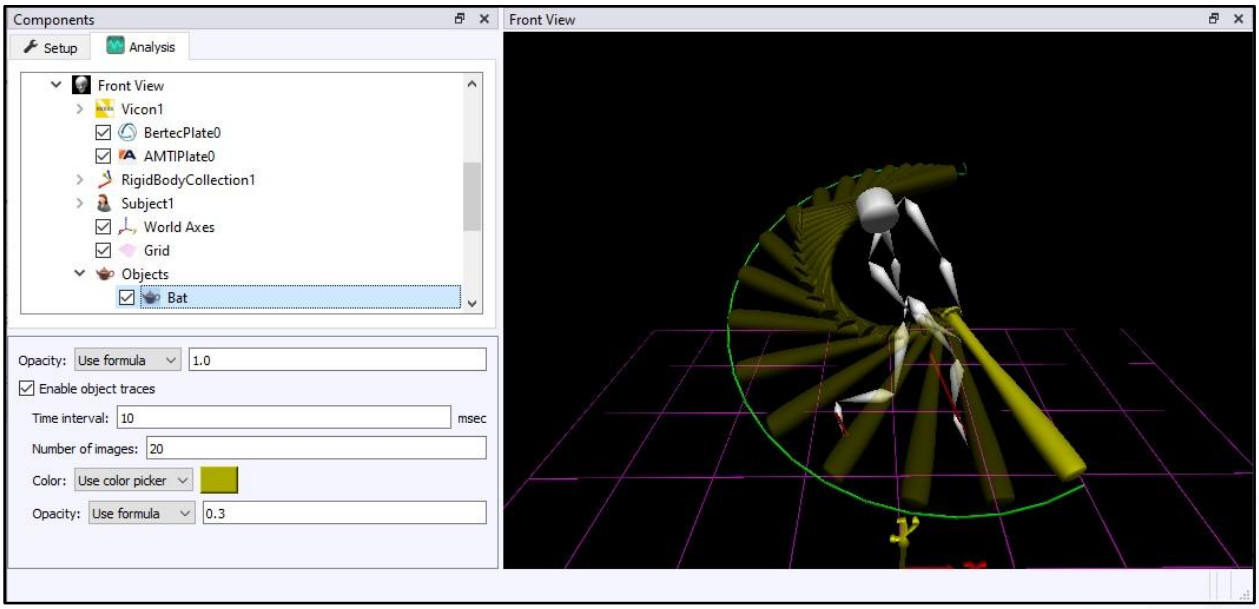
Data from an Object (Plane or Model) can be used in analyses or displayed in graphs. Some sample variable definitions are displayed below.



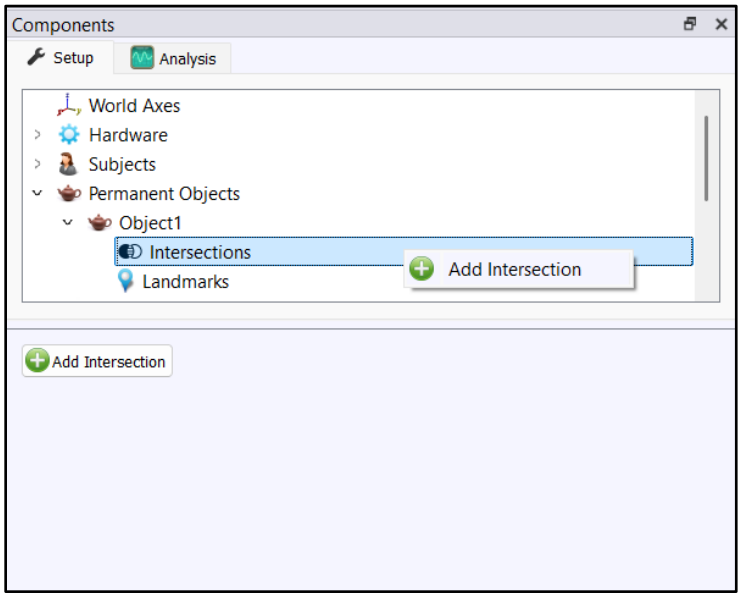
In the case of a Plane, the Axes may also commonly be used as the reference frame that a variable is reported 'relative to', as seen below.



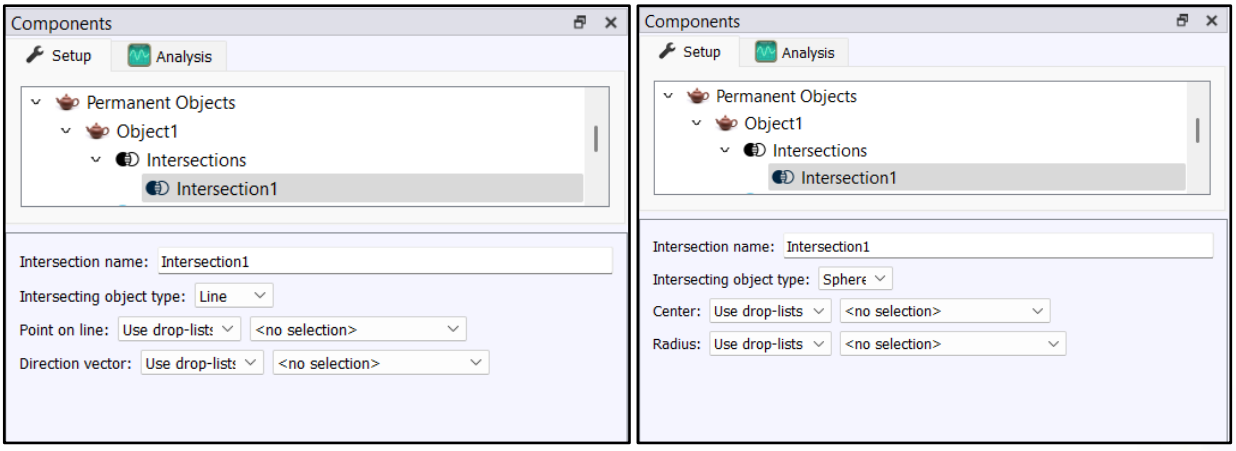
Objects can be enabled or disabled within an Animation window by expanding the Animation node in the Components Analysis tab. When an object is selected, a constant or formula for its opacity can be specified for the display in the Animation window. Object traces can also be enabled to visually track the object's position and orientation over time. The Time interval controls the period in between traces of the object. The number of images indicates the total number of objects to be displayed in the trace. The color and opacity for the object traces can also be specified. An example of this is shown with the Baseball Bat below (Included in the Sample Scholastic Files User).



When defining an Object type as a Plane, intersections can be defined for the automatic detection of intersections with the plane. To add an intersection, go to the Object node of interest and select the Intersections node and click the "Add Intersections" button in the parameters panel at the bottom of the Components window or by right clicking the Intersections node for the Object.



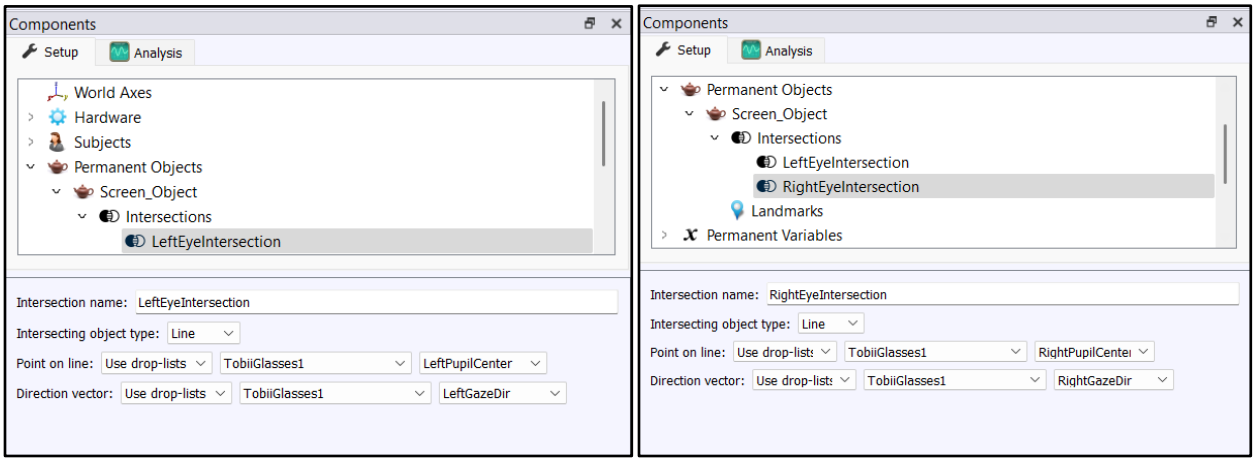
When added, the "Intersection name" can be specified. This will be the name used throughout the software when referencing the object intersection. The "Intersection object type" can be selected as either a "Line" or "Sphere" and multiple intersections can be created for each object with "Plane" selected for the "Object Type".



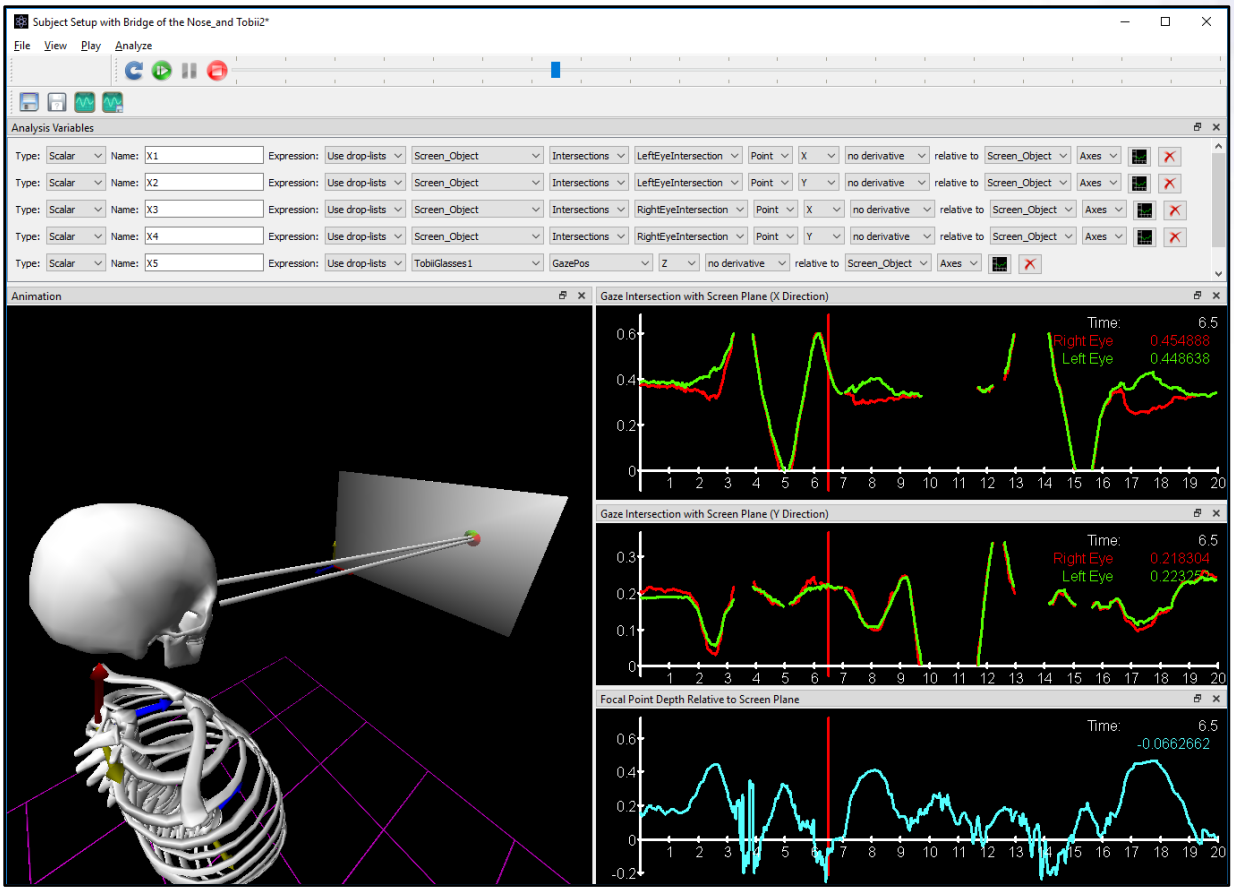
If the plane is defined before data is captured, intersections can be added from within an Activity. However, from an ease-of-use standpoint, it's better to define intersections before data collection to automatically include them in every Activity.

If “Line” is selected as the “Intersecting object type”, the “Point on line” first needs to be specified using the drop-lists or formula method. This point can be any vector position, either a constant value or variable that changes over time. The “Direction vector” is a vector that is specified using the drop-lists or formula method and determines the direction of the line that goes through the previously specified point. The line defined by this point and direction vector will project in both the direction and opposite direction of the specified direction vector from the specified point. Examples for use of the line intersection type could be with an eye angle gaze vector or with finger pointing tasks. In the eye angle example, the Point on line could be defined as the pupil center and the direction vector could be defined as the gaze direction. The result of this would yield information for where the gaze intersects with our Plane. In the finger pointing example, the point on line could be defined as the tip of the finger and the direction vector could be defined as the vector defining the finger. The result of this would yield information for where finger pointing would intersect as a projection onto the plane.

An example for how the eye gaze setup could be configured is displayed below for data that was captured from a Tobii eye tracking system. Line intersections were added to the Plane object for both the left and right eye gaze directions.

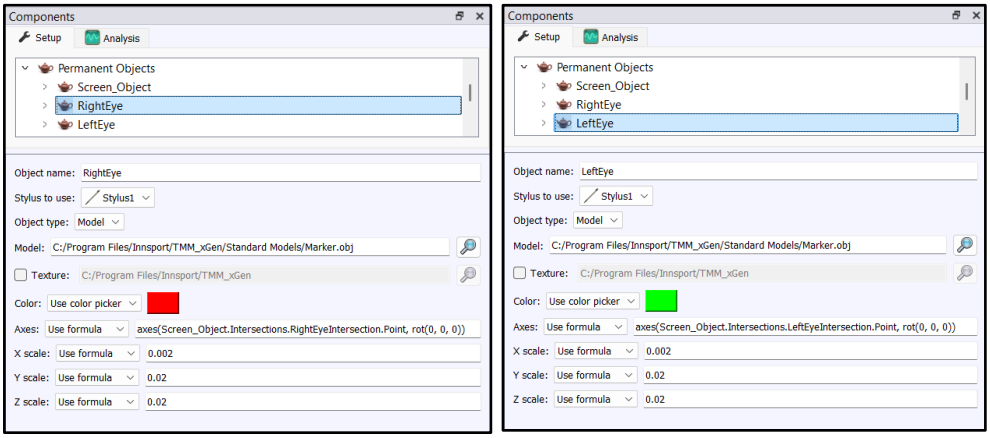


In the Animation window below, the subject, their left and right gaze vectors and a Plane defined for a monitor screen can be seen.



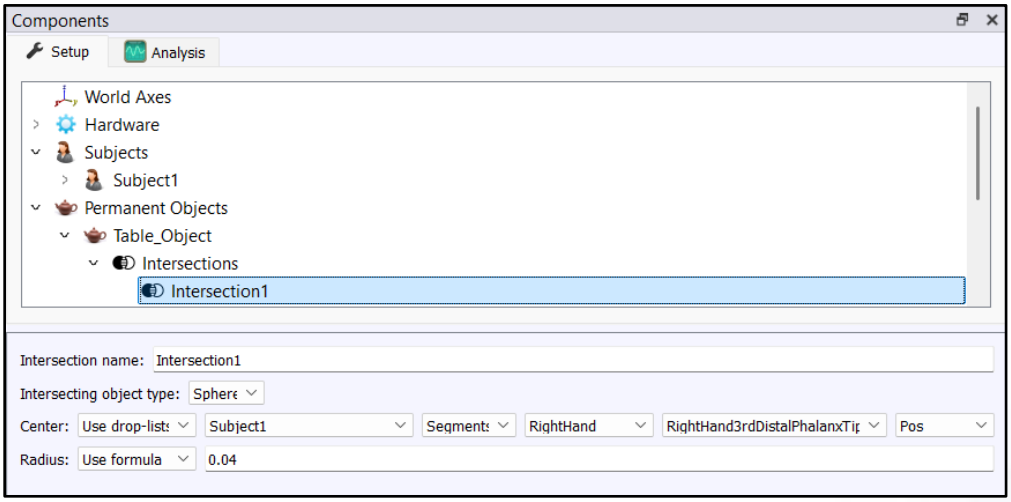
The variables defined in the Analysis Variables window above are also displayed in the 3 graphs. The data represents the location of the right and left eye gaze vector intersections with the Plane, in the reference frame of the Plane in the x and y directions. The focal point depth is also defined relative to the z direction of the Plane axes. Missing data in the plots either represent when the Subject had closed their eyes or when the eye gaze vectors ventured outside of the perimeter of the plane and did not intersect with it.

An additional visualization for where the eye gaze vectors intersect with the Plane were added in the Animation above. The red and green circles displayed on the Plane representing a monitor screen, used the Left and Right eye Object intersection vectors. The X-scale for the objects was also scaled to make the marker more 2-dimensional and to keep it in the plane of the monitor screen in the Animation window.

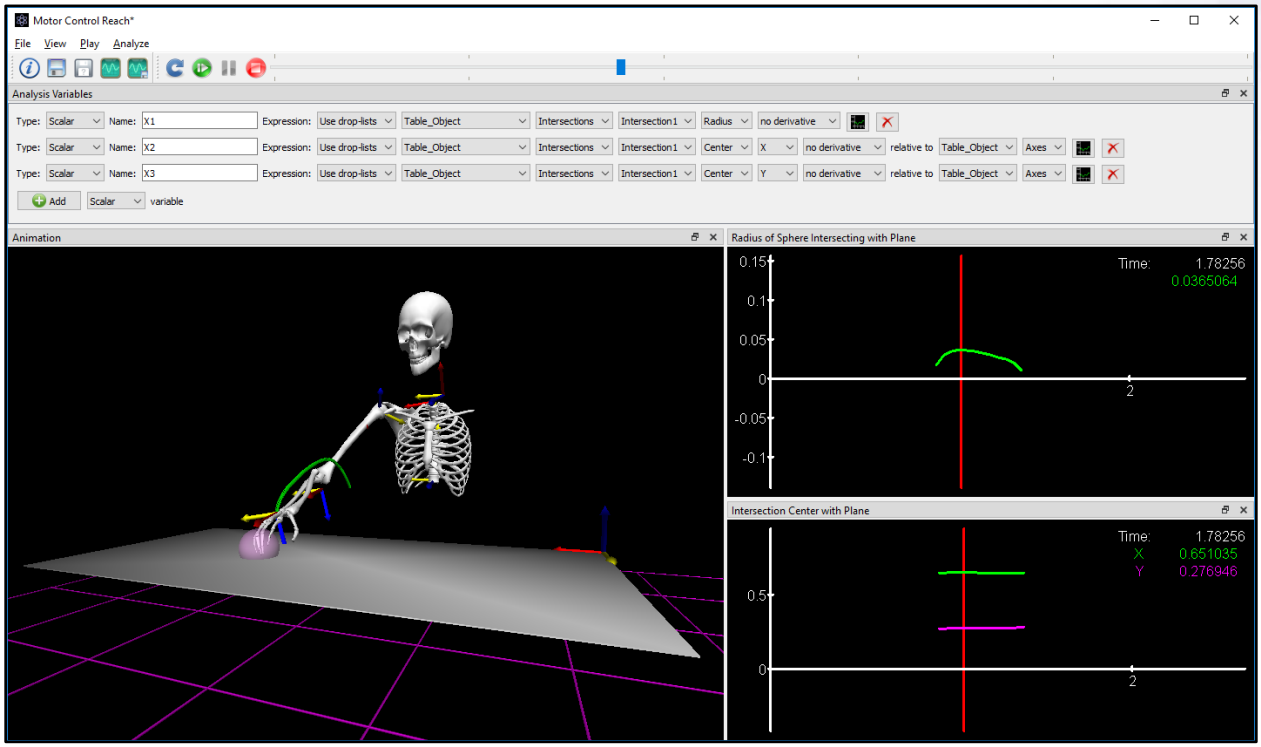


If "Sphere" is selected as the "Intersecting object type", the "Center", a vector position, needs to be specified using the drop-lists or formula method. This will represent the origin of the sphere and can be either a constant value or variable that changes over time. The "Radius" is scalar variable used to define the size of the sphere used in determining the intersection, similarly, it can be a constant or variable that changes. An example for use of the Sphere Intersection type could be for the determination of when and where the hand comes in contact with a table or screen during a reaching paradigm.

In the case of a table, the object would be a plane which represents the table. The sphere intersection could be defined using the fingertip position as the center of the sphere and the radius would represent the acceptable error margin (i.e. 4cm). An example of how this setup could be configured is displayed below.

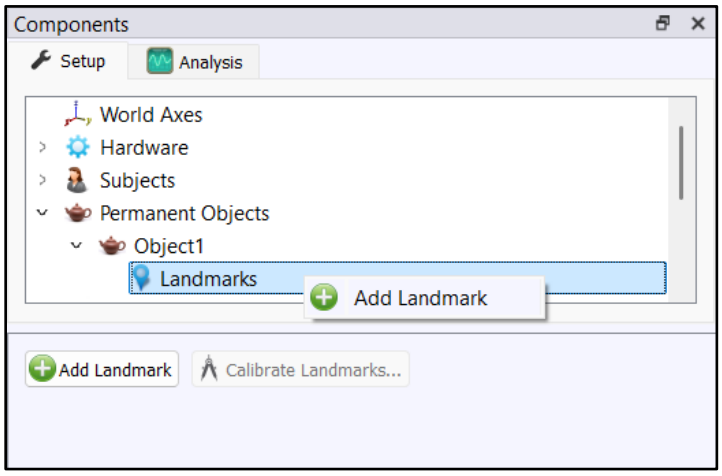


In the Animation below, you can see what this experiment might look like. There is a subject reaching on a table. For purposes of visualization, an additional object has been added to the Animation window representing our sphere intersecting object.

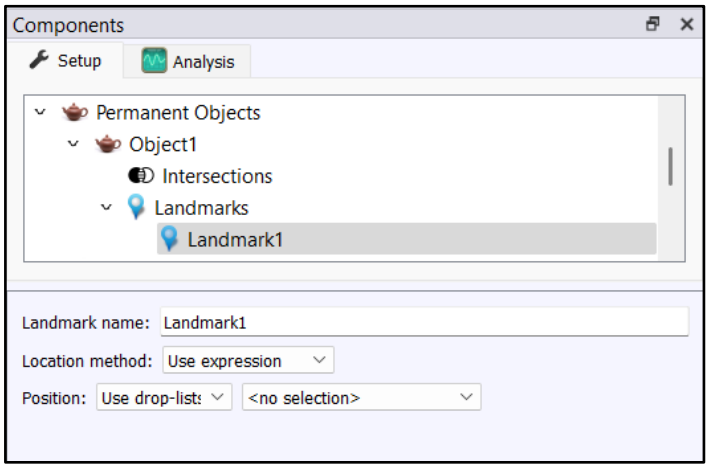


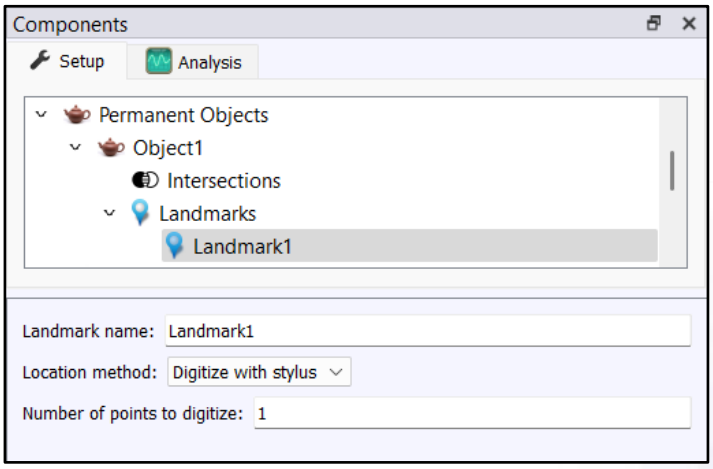
The variables defined in the Analysis Variables window above are also displayed in the 2 graphs. The first variable and top graph shows the variable for the radius of the object that is intersecting with the plane. In the Animation window above, you can see this as the cross-section of the sphere that is intersecting with the plane. It's this radius that is being reported, not the "Radius" specified for the "Sphere" "Intersecting object type". The remaining variables and bottom graph show the position for the center of the section of the sphere that is intersecting with the plane, relative to the table's reference frame. Note that the center reported here is not the center of the sphere, but the center of the cross-section of the sphere that is intersecting the plane. These would only be the same if the exact middle of the spere was intersected with the plane (i.e. half of the sphere is above, and half of the sphere is below the plane). For each of these variables, data is only reported when the sphere intersects with the plane.

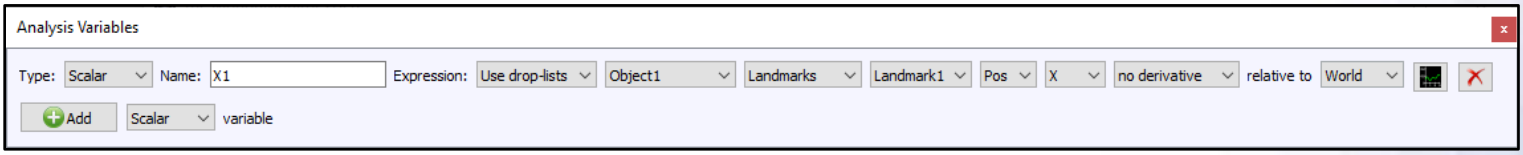
When defining an Object type as either a Plane or Model, landmarks can be tracked similarly to the anatomical landmarks for subject segments. To add a Landmark, go to the Object node of interest and select the Landmarks node and click the "Add Landmark" button in the parameters panel at the bottom of the Components window or by right clicking the Landmarks node for the Object. Landmarks can be calibrated using the "Calibrate Landmarks" button from the parameter panel of the Landmarks node or the Object node.



When added, the "Landmark name" can be specified. This will be the name used throughout the software when referencing the object landmark. The landmarks can be located either with a position expression or by digitizing them with a stylus. Landmarks must be calibrated before data are collected.







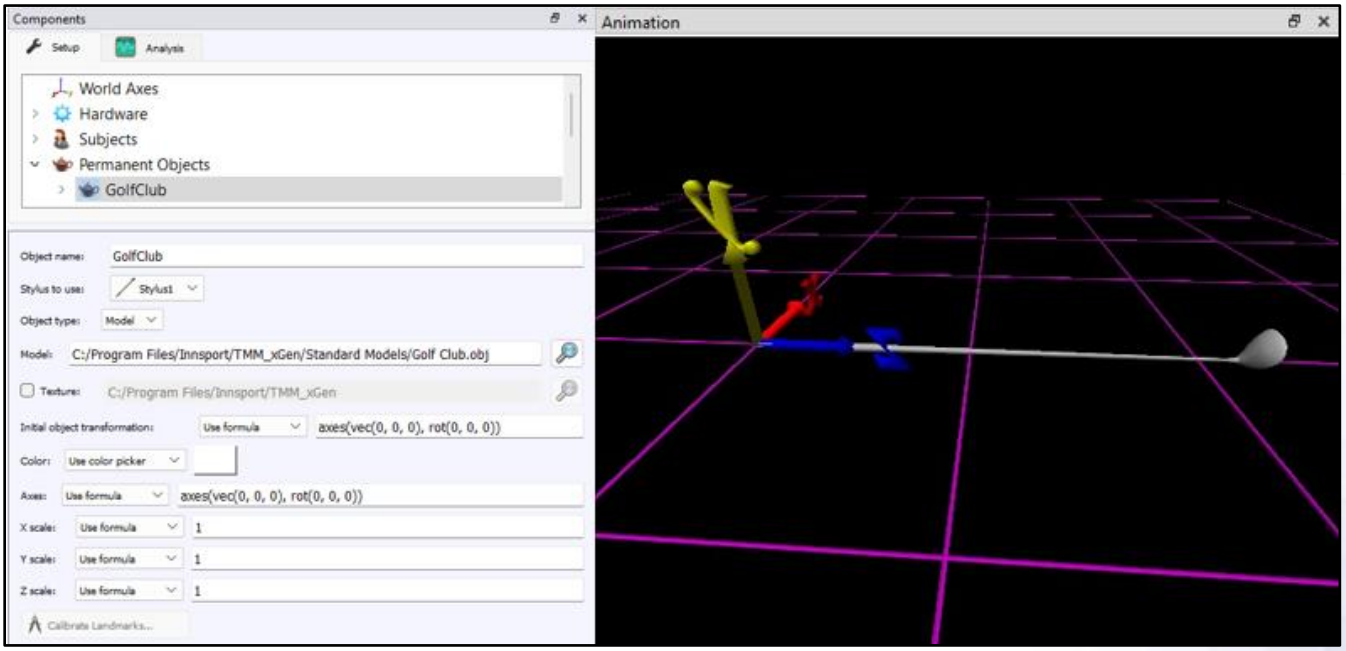
Appendix A

Here, we'll examine the alignment of an object file with the real-world object being tracked. We'll review the object file itself, the axes() used for the position and orientation of the object in the Animation window and the scaling of the object. For the purposes of this example we will use a Golf Club OBJ file, but everything that follows will apply to any OBJ file. We have chosen this file because of its familiarity and non-uniform shape.

Again, an object file (.obj) is an image file that contains data for rendering a three-dimensional object. The file itself does not contain any units of measure, but rather includes vertices relative to some arbitrary reference frame and polygon faces that are used for rendering the object. When reading an OBJ file, The MotionMonitor xGen will render the file as if the units are meters, regardless of the actual units used in the application where the OBJ file was created.

Both the object file and the rigid body assigned to it have their own 6DoF coordinate system. Ultimately, the position and orientation of object file needs to be aligned with the coordinate system of the rigid body used to control the movement of the object in the animation window.

Let us examine an object when it is first loaded in The MotionMonitor xGen. An Axes definition, *axes(vec(0, 0, 0), rot(0, 0, 0))* is the default assignment. This Axes will place the origin of the OBJ file at the origin of our World Axes. The orientation of the object in our World Axes tells us how the orientation of the object is defined within the OBJ file itself, since no rotation is applied to the file. You can also get a sense of the size of the object. In our case, the length is just over 1m, as we can determine because the increments for the grid are 0.5m.



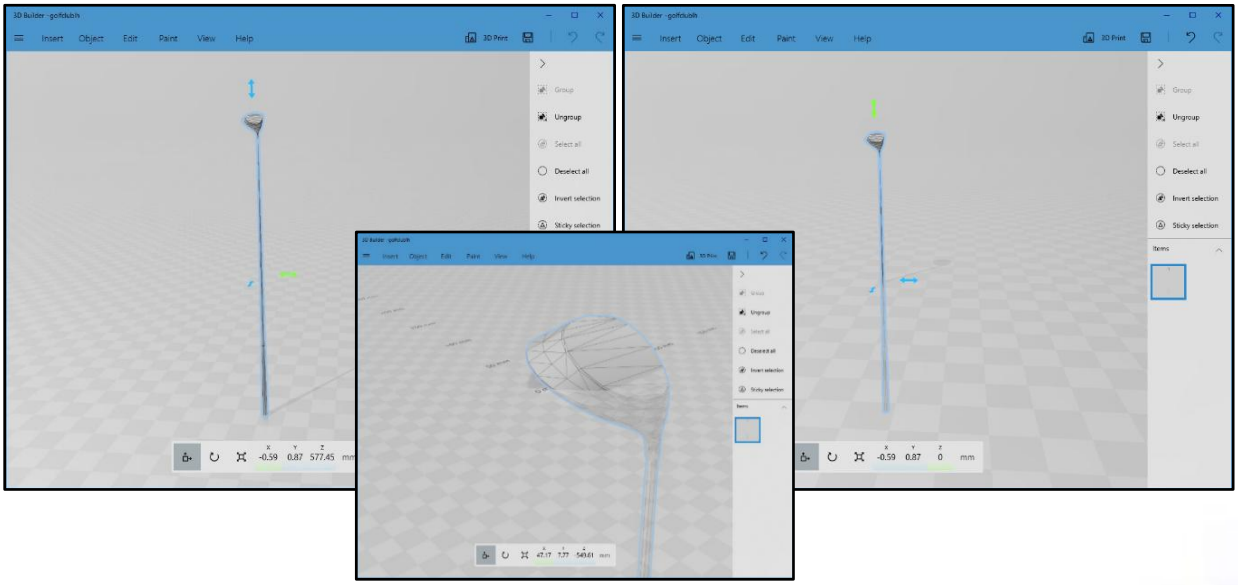
There are many applications that allow for viewing, modifying, and converting between 3D file types. For the purposes of our example here, 3D Builder was used. 3D Builder is an easily accessible application since it is typically included with Windows installations.

Translating the origin location in an OBJ file

As we previously determined from viewing the golf club in The MotionMonitor xGen Animation window, the origin of the OBJ file was located at the proximal end of the club's shaft.

There is no requirement that the origin of the OBJ file be located on the surface or within the structure of the object. However, it will certainly make the process of aligning the object to its analogous real-world position simpler if it is. The recommended placement for the origin of the OBJ file is to select a point that can either be digitized or where a passive optical marker can be affixed to the real-world object. The reference to this marker or digitized landmark can then be used as the position vector for the object Axes in The MotionMonitor xGen. If the origin is located away from the object in the OBJ file, adjustments to the object's orientation can result in large changes for the object's position and make it more difficult to properly align.

The origin in the Object file can be translated to the desired position. The orientation of the Object is the same in 3D Builder as it was in The MotionMonitor xGen, the perspective is just different. In the images below, positive-x is to the right, positive-y is the direction for the perspective of someone reading this document, and positive-z is up.



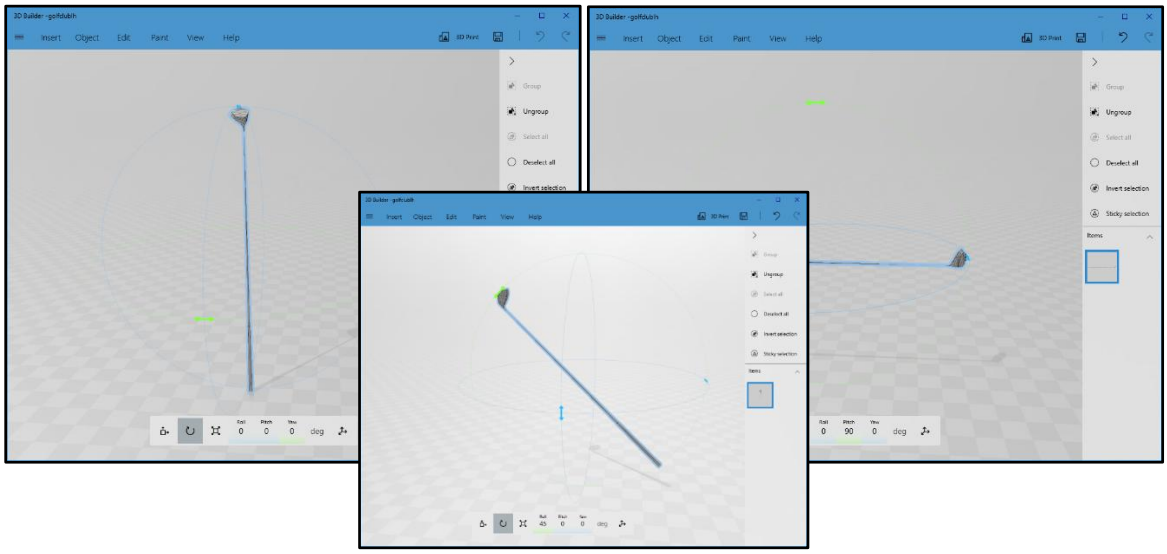
In the images above and to the back, the club was translated along the length of the shaft in the negative z-axis direction. In the image above and on top, the origin was translated from the proximal end of the shaft to the back of the club head – a location where a marker can be affixed to the club.

Orientation of the model in the OBJ file

Ideally, the orientation of the model within the OBJ file will align squarely with the World Axes in The MotionMonitor xGen.

The orientation of the model could be adjusted to have the club head face and club shaft directed along particular axes of the world axes. This can be particularly advantageous when using a passive optical system because the orientation for 6-DoF rigid bodies are aligned to the world axes when they are calibrated. So, if the model's orientation is such that it is the same as the real-world object is when the rigid bodies are calibrated, then the orientation between the two are already aligned.

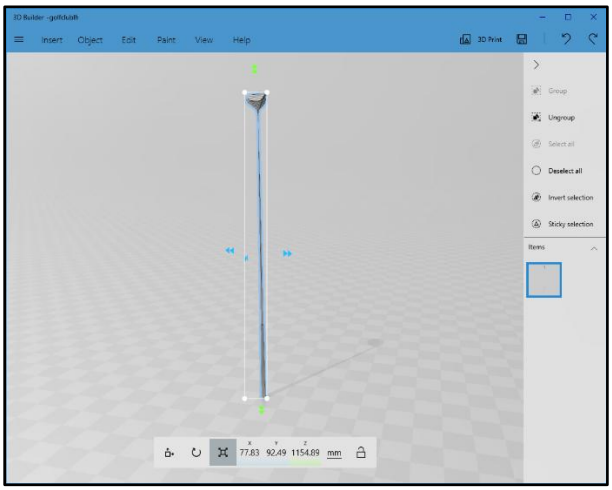
The images below and to the back show the club being rotated about the y-axis. The image below and to the front shows a club that was rotated about the x-axis.

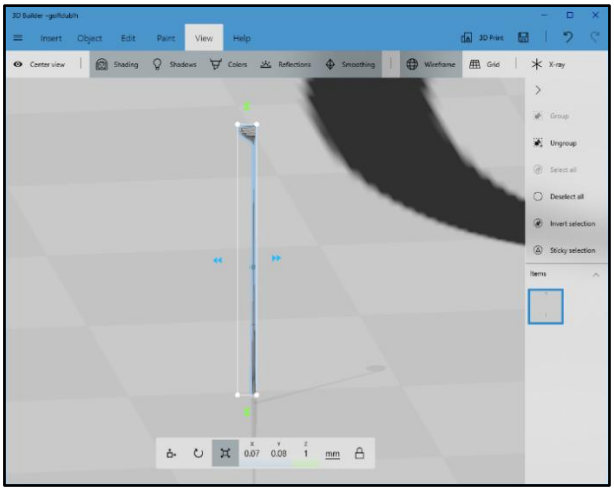


Scaling of the model in the OBJ file

As was previously mentioned, the object file does not contain any units and The MotionMonitor xGen will use meters when rendering the file. If the file was created using cm or inches, for instance, the file may be very large when initially rendered within The MotionMonitor xGen. When this is the case, the Animation perspective may need to be zoomed out in order to see the object. Our recommendation is to scale the object, in this case its length, to 1 or 1 to some power of 10. Also, keep the aspect ratio locked and only adjust the main axis length or scale the size as a percentage of its current size. Scaling the file here will help to make the initial scaling in the x, y and z directions within The MotionMonitor xGen easier and then digitizing the length, for instance, allows us to scale to the proper size for the real-world object along all 3 axes.

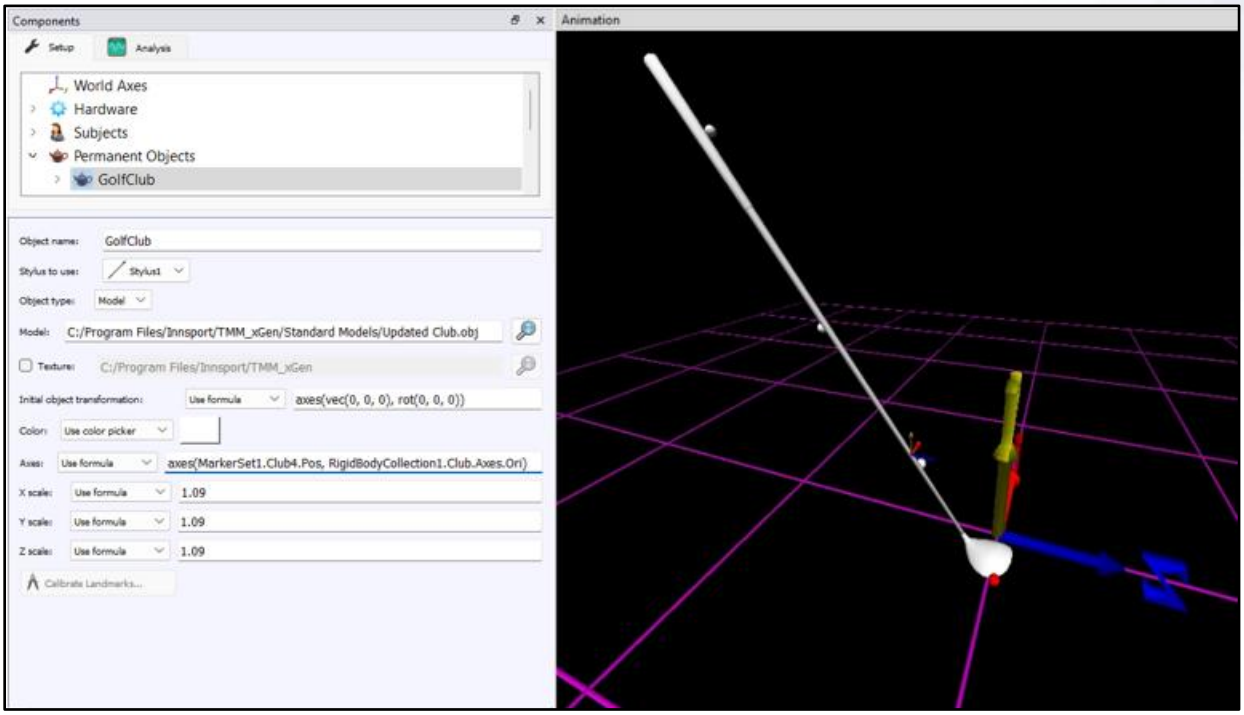
The images below show the scale for the golf club's length being set to 1 while the aspect ratio is locked.





When the origin, orientation and scaling of the object are as desired, save the file as an .obj file to be able to load it within The MotionMonitor xGen.

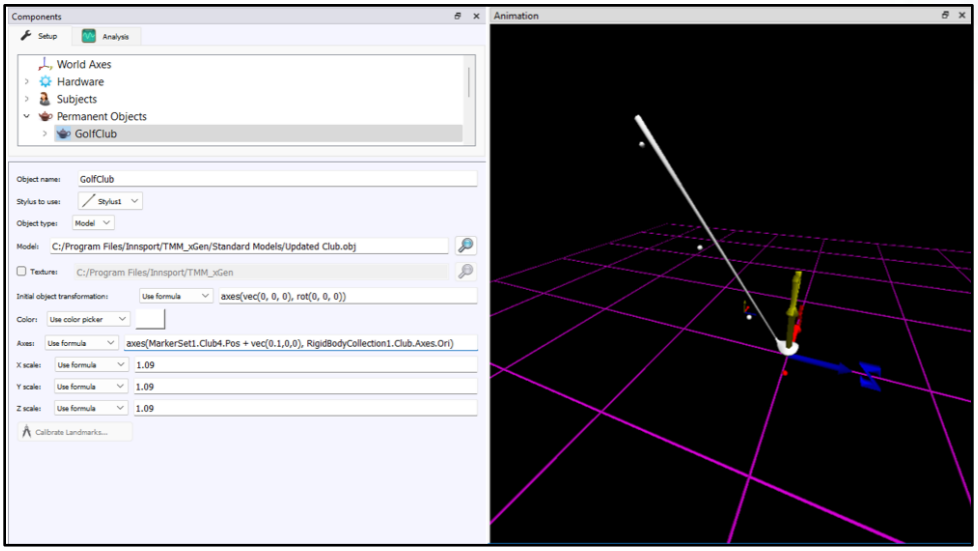
In the image below, the updated model file for the Golf Club was loaded within The MotionMonitor xGen. The model file was configured with the origin at the back of the club head and angled as the club would be at address.



The Axes were defined as *axes(MarkerSet1.Club4.Pos, RigidBodyCollection1.Club.Axes.Ori). MarkerSet1.Club4.Pos* is the marker attached to the back of the club head, highlighted in red above. The *RigidBodyCollection1.Club.Axes.Ori* is the orientation of the rigid body tracking the club. Notice how the smaller rigid body axes for the club are aligned with the World axes, as was discussed previously in the Orientation of the model in the OBJ file section. The x, y and z scales were all set to 1.09, as the length of the club is 1.09m. In this case, our model file perfectly aligns with the placement of the real-world golf club. This is because we defined the model file such that the origin was at a place we could easily define with a marker from the motion capture system, the orientation matched what it would be at address and our length was set to 1. However, this may not always be the case, so now we will look at how adjustments to the model's positioning and orientation can be performed and methods for automating these processes.

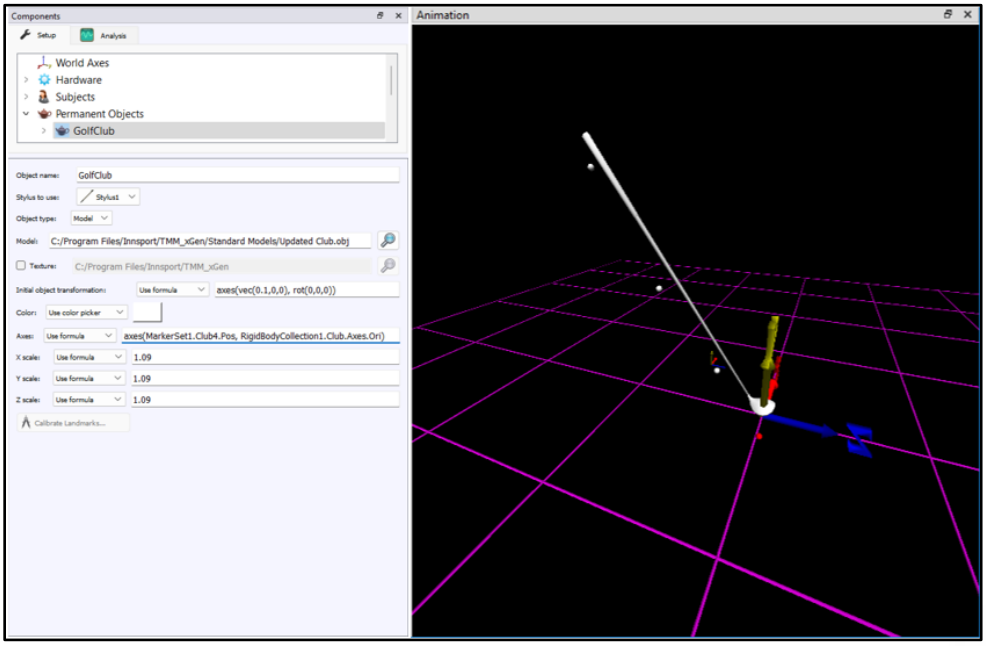
Translating the model files position

The image and formula below demonstrate how the position can be translated.



axes(MarkerSet1.Club4.Pos+ vec(0.1,0,0),RigidBodyCollection1.Club.Axes.Ori)

In the above example, the club was translated 10cm in the positive x-direction of the World axes. Although the image below looks the same as that which is displayed above, the formula utilizes the Initial Object Transformation for translation.

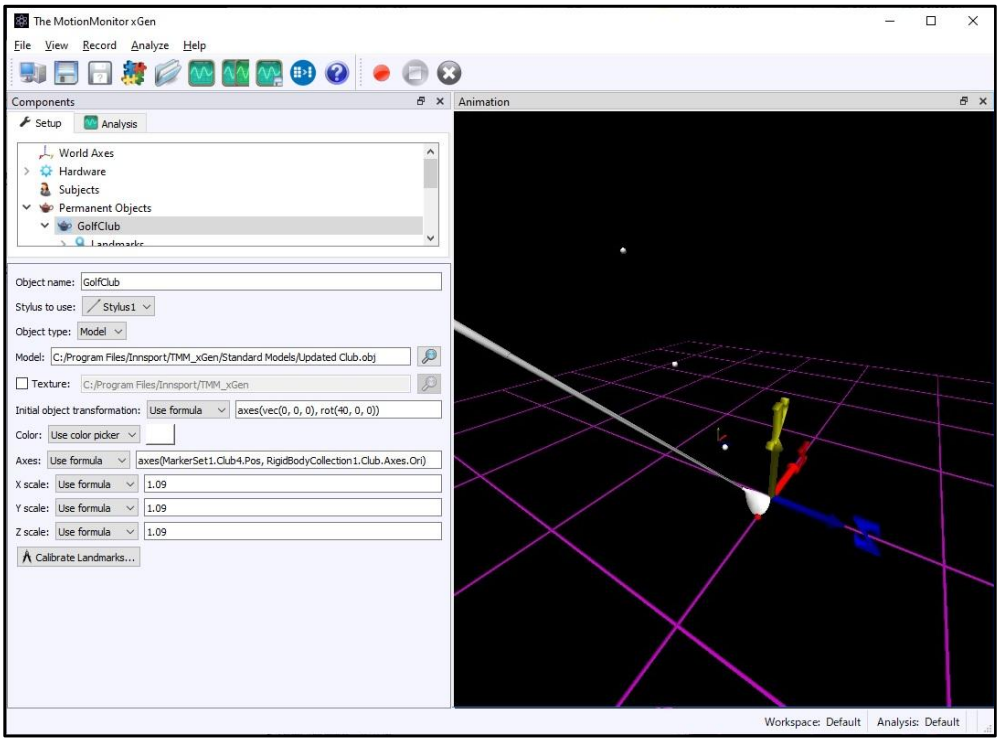


Initial Object Transformation: axes(vec(0.1,0,0), rot(0,0,0))
axes(MarkerSet1.Club4.Pos,RigidBodyCollection1.Club.Axes.Ori)

Instead of adding a vector of vec(0.1,0,0) to our marker position as was done in the first example, we have defined an Initial Object Transformation *axes(vec(0.1,0,0), rot(0,0,0)).* The former definition will result in an offset always being applied relative to the World's reference frame. The latter formula will result in the offset being applied in the club's reference frame, so the position of the offset will rotate with the club as the club rotates. The offset was entered manually here, but landmarks could be digitized and used instead.

Changing the model files orientation

The image and formula below demonstrate how the orientation can be adjusted.



Initial Object Transformation: axes(vec(0,0,0), rot(40,0,0))
axes(MarkerSet1.Club4.Pos,RigidBodyCollection1.Club.Axes.Ori)

In the above example, an Initial Object Transformation was used to rotate the club 40 degrees about the z-axis for the golf club, where the rotation sequence is (Z, Y, X).

It can be tedious to perform rotation adjustments by hand, so we will review a method for automating this process. Although it is not necessary, it is advisable to try to get the orientation of the model file close to how the real-world object can be oriented when calibrating the rigid body used to track the object, as previously described. If the rigid body axes orientation used for the Axes assigned to the object file matches the orientation of the model within the OBJ file, no adjustments may be necessary. Just getting these axes orientations close will make the orientation adjustment process significantly easier.

The automated process consists of 2 steps in a script which will allow us to determine the rotation offset between the orientation of the model in the OBJ file and the orientation of the object in the real-world.

For our Axes assigned to the object file we have:
axes(MarkerSet1.Club4.Pos,RigidBodyCollection1.Club.Axes.Ori)

For our Initial object transformation we have:
axes(vec(0, 0, 0), ObjectOffset.Ori)

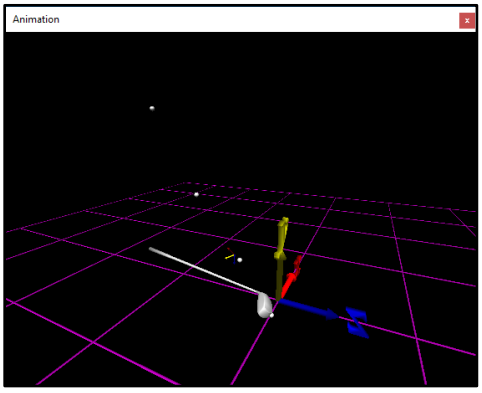
In the Axes assigned to the object, MarkerSet1.Club4.Pos is the marker that coincides with the origin of the Object file, so there is nothing applied in the Initial object transformation for the position. In the Initial object transformation, ObjectOffset.Ori will be our rotation to make the object file align with the real world object. Again, if the rigid body axes orientation used for the Axes assigned to the object file match the orientation of the model within the OBJ file, no adjustments may be necessary

The following variables are used in the script:
ObjectOffset is a script axes variable
ObjectSensorRealAxes is a script axes variable
RigidBodyCollection1.Club.Axes is our axes assigned to the golf club
ObjectSensorAnimationAxes is a script axes variable

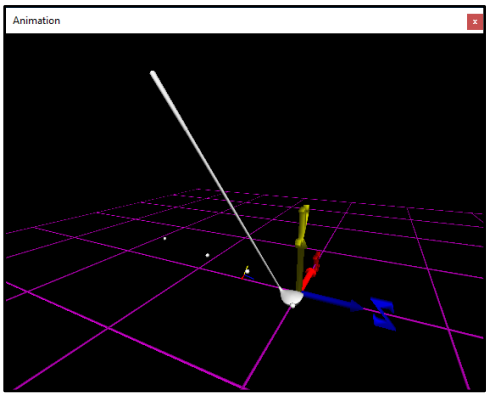
First, place the real-world object in a reproducible 'neutral' orientation. For instance, have the face of the club addressing a particular world axes and the shaft of the club perpendicular to the ground or at address. At this point, the first reading can be taken for the object.

```
OKMessage("Click OK to take the reading of the orientation when real-world object is in the 'neutral' position.");

ObjectOffset = axes(vec(0,0,0), rot(0,0,0));

ObjectSensorRealAxes = RigidBodyCollection1.Club.Axes;
```

In these readings, 1) the offset axes are set to zero and 2) the orientation for the sensor tracking our golf club is obtained during the previously described “neutral” orientation.



Next, physically rotate your real-world object until the animation for the object matches your “neutral” orientation. The face of the club for the model should be addressing the same world axes and the shaft of the club perpendicular to the ground or at address, as they were in the “neutral” reading for the real-world object. When in the proper position, take the second reading. If the rigid body axes orientation used for the Axes assigned to the object file and the orientation of the model within the OBJ file are significantly different, this step can be more difficult.




```
OKMessage("Click OK to take the reading of the orientation when Animation object is in the 'neutral' position.");

ObjectSensorAnimationAxes = RigidBodyCollection1.Club.Axes;

ObjectOffset = rel(ObjectSensorAnimationAxes, ObjectSensorRealAxes);
```

In this step, a second reading of the orientation for the sensor tracking our golf club is obtained and the difference is determined between this reading and our previous “neutral” reading as ObjectOffset.

Scaling

Additionally, instead of manually measuring and entering the length of the club for the use in scaling of the model file, the proximal and distal endpoints of the club could be digitized to capture the club length. If the club is stationary, these points could be taken relative to the world. Otherwise, they could be captured relative to the golf club’s rigid body axes. Ideally, this process would proceed to the orientation adjustments made above in order to ensure valid scaling data so that the object can be visualized in the Animation window. The points used for scaling the object could also be defined as landmarks for the object, instead of as script variables as is about to be demonstrated.

The following variables are used in the script for determining the scaling:

RigidBodyCollection1.Club.Axes is the rigid body axes tracking the golf club
Stylus1.Tip.Pos is the position of the tip of the calibrated stylus
DigitizedGolfClubProximalPos is a script vector variable
DigitizedGolfClubDistalPos is a script vector variable
GolfClubScaleFactor is a script scalar variable

A script could be used to capture the following vector script variables:

```
OKMessage("Click OK to take the reading for the proximal end of the object");

DigitizedGolfClubProximalPos = rel(Stylus1.Tip.Pos, RigidBodyCollection1.Club.Axes);

OKMessage("Click OK to take the reading for the distal end of the object");

DigitizedGolfClubDistalPos = rel(Stylus1.Tip.Pos, RigidBodyCollection1.Club.Axes);

GolfClubScaleFactor = mag(DigitizedGolfClubProximalPos - DigitizedGolfClubDistalPos);
```

If the length of the club in the object file was set to 1, the magnitude of the subtracted vectors, GolfClubScaleFactor, could be used for the x, y & z scales of the object.