Stream a Joint Angle Manual

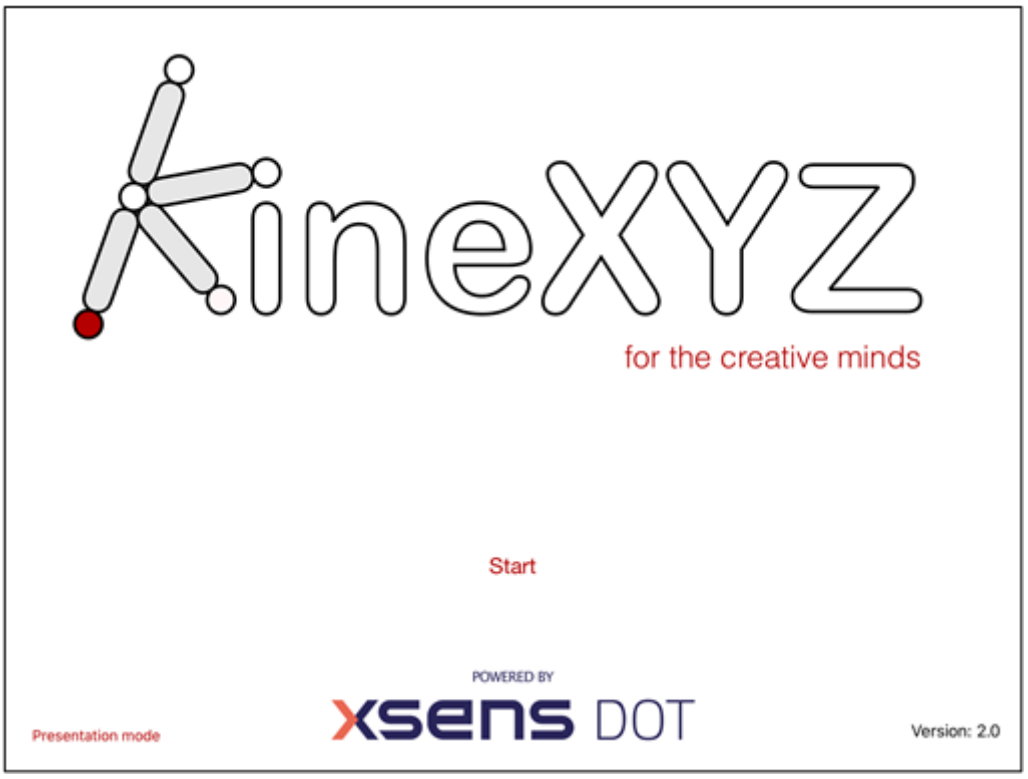

Table of contents

Introduction

When considering for example biomechanical measurement systems, often the joint angle is required between two segments of the body.
Without requiring any prior knowledge, this document describes the complete process of using the KineXYZ app on the iPad to:

1. Create a kinematic model of an arm with an elbow joint.
2. Attach and align the wireless motion tracking sensors.
3. Create a mathematical model to calculate the joint angle.
4. Stream the joint angle to the Xsens DOT Plot app.

To get going, find the KineXYZ icon on the iPad home screen and start the KineXYX app (see figure 1).

Figure 1: Start the KineXYZ app on the iPad.

Create kinematic model

Overview

The first step is to create a *kinematic model* of an arm. This involves

1. Adding objects to the scene.
2. Create a kinematic chain by connecting the segments.
3. Set anchor-point.

These steps are described in more detail in the following sections.

Adding objects to the scene

To be able to create simple models of objects and segments, KineXYZ comes with a number of standard geometric primitives[1].

To create a model of an arm, two *capsules* can be used.

To add an object to the scene, press the Objects button (see figure 2) which will show all the available geometric primitives.

Figure 2: Objects button.

[1] https://en.wikipedia.org/wiki/Geometric_primitive

Select the capsule and drag it into the scene. The capsule is added as a partially transparent

object (see figure 3).

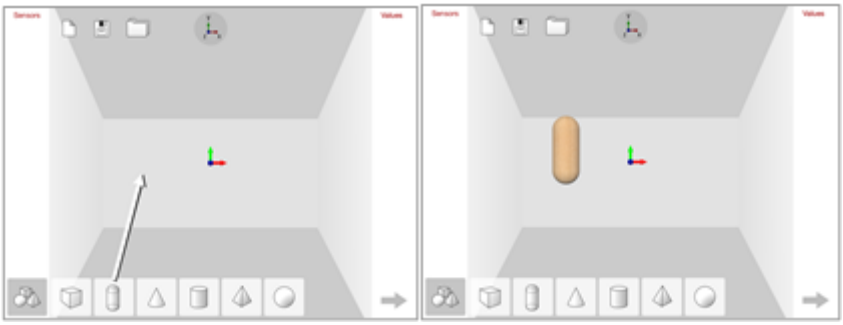

Figure 3: Objects-button pressed (left) and dragged into scene.

Newly added objects are given a default name like “Object_1”.

To give the added capsule in the scene more meaningful names, select the capsule by tapping it and then press the Properties button (see figure 4).

Figure 4: Properties button.

Next, the properties-window appears showing all variables that can be altered, starting with the object’s name (see figure 5).

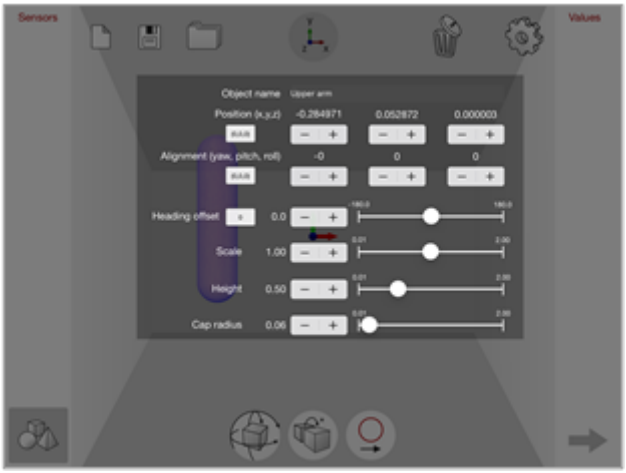

Figure 5: Properties-window for a capsule object.

Change the name to “Upper arm” and set the length to 0.50 (or any other length that seems realistic). When done, dismiss the properties window by tapping anywhere outside it.

While the object is still selected, a copy can now be made by pressing the Copy button (see figure 6). The copy will appear in the origin.

Figure 6: Copy button.

Change the name of this capsule to “Lower arm”.

Connect segments

The two added capsules are still not part of a kinematic chain. For this to be the case they must be connected by a *joint* obviously.

In KineXYZ, a *joint* is a connection between two objects. It is represented in the scene as a visible object and can be selected and manipulated.

To make it very easy for the user, each geometric primitive has 6 *connection-points* at predefined locations. For a capsule these are located at the top and bottom (y-axis) and mid-way on both sides of the x- and z-axis.

As the name suggests, a connection-point can be used to connect to a connection-point of another object, whereby effectively a new joint is created.

Connection-points are not shown by default. Pressing the joints-button (see figure 7) will make the connection-points of all objects in the scene visible.

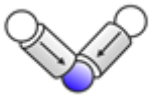

Figure 7: Joints button.

First select the upper connection point of the lower arm and then select the lower connection point of the upper arm. This will create the joint, i.e. the elbow.

Hiding the connection-points is done by deselecting the Joints button.

The process is illustrated in figure 8.

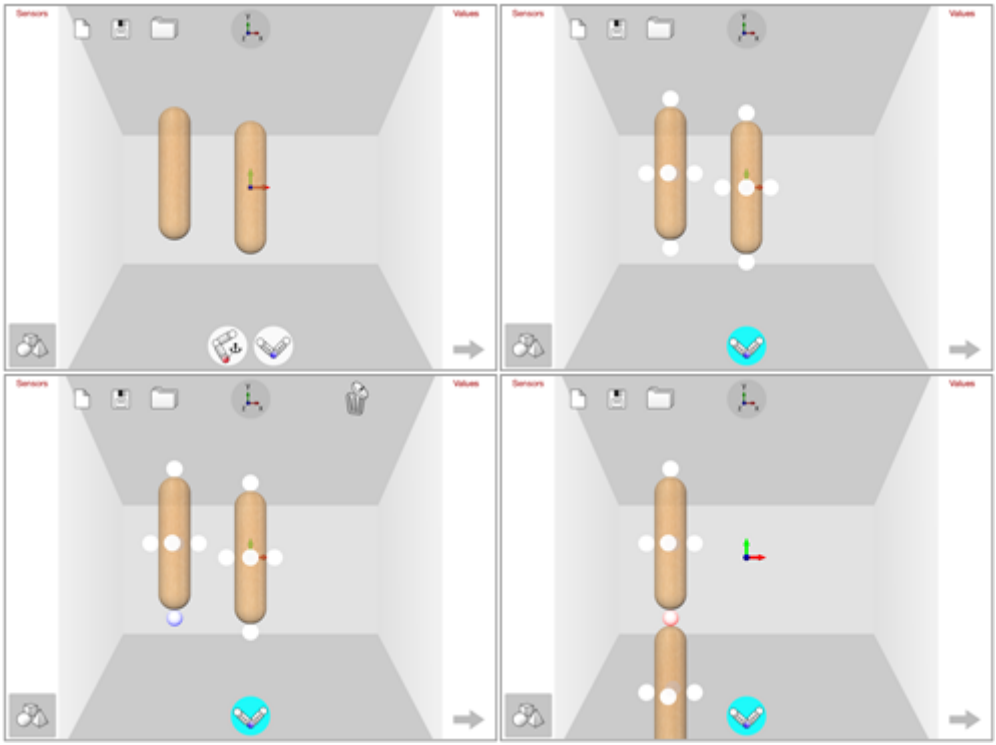

Figure 8: Creating joints by connecting objects.

Note that the scene can be zoomed out by making a pinching movement (see figure).

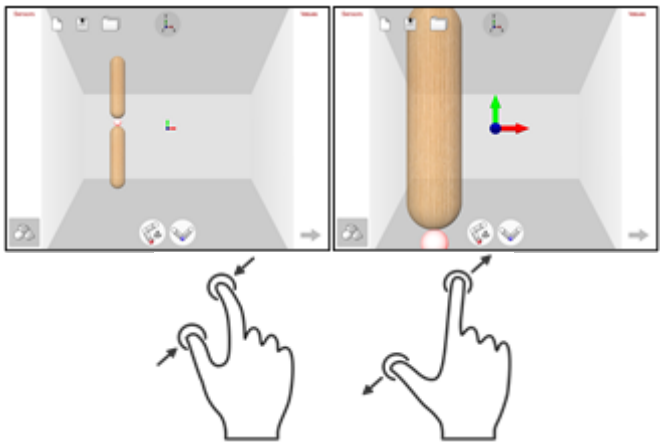

Figure 9: Pinching movements to zoom out (left) and zoom in (right).

Set anchor point

A kinematic chain is built by connecting objects, whereby the objects become segments. Such a structure of objects always has a fixed point which is the starting point in determining the *pose* (i.e. the positions and orientations of the segments) of the kinematic chain. This fixed point is called an *anchor-point*. By default, an anchor point is created as soon as two objects are joined together. However, in KineXYZ you can also adjust the anchor point of a kinematic chain.
To be able to change the anchor-point, first press the Anchor-point button (see figure 10).

Figure 10: Anchor-point button.

Pressing the Anchor-point button will unhide the connection-points of all objects in the scene, with the anchor-point visualized as a red coloured connection-point.
Now an anchor point can be defined by selecting one of the connection points on the kinematic chain. For the created model of an arm, the anchor point will be the connection point at the top of the upper arm, thereby modeling the shoulder. This process is illustrated in figure 11.

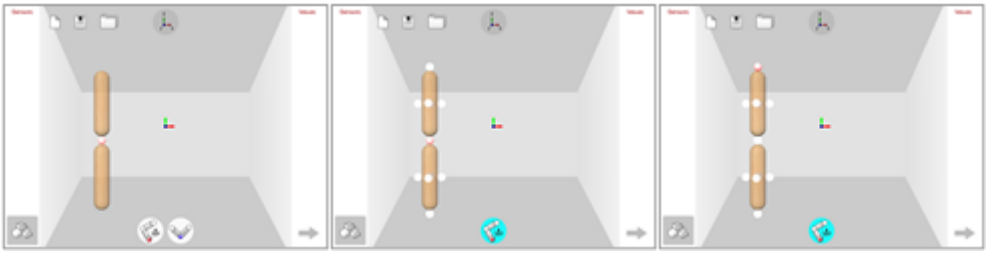

Figure 11: Changing the anchor point of a kinematic chain.

Now that the kinematic model of an arm is created, the next step is to use the motion tracking sensors to manipulate the arm. This process is described in the next chapter.

Connecting and aligning sensors

Overview

Now that the kinematic model is created, the next step is to use the motion tracking sensors to control the orientations of the segments. This is done by performing the following steps:

1. Start scanning and connect sensors.
2. Assign the sensors to the proper segments.
3. Align the sensors to their segments.

These steps are described in detail in the following sections.

Connecting sensors

To enable the detection of Bluetooth sensors, the device must start listening, or *scanning* as it is called in Bluetooth jargon. This can be done by dragging the “Sensors” column on the left of the screen towards the right (see figure 12).

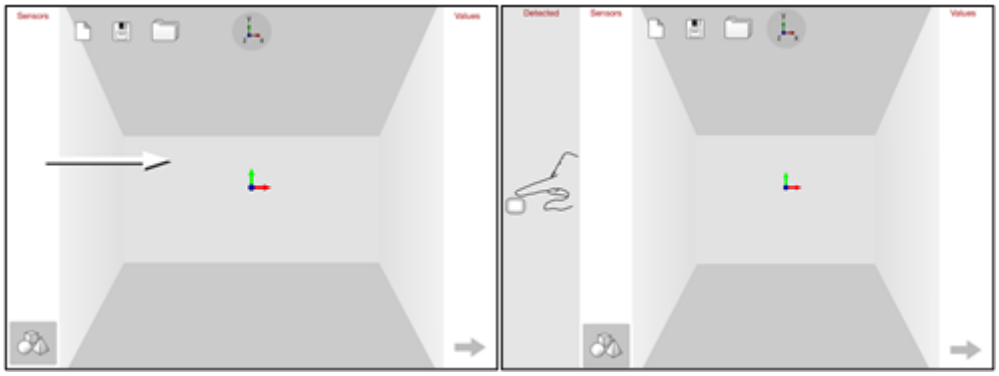

Figure 12: Dragging the Sensors-column to the left, enables scanning.

When the sensors are switched-on, they will connect and appear in the column. If sensors are switched-on, but don’t properly connect, power-cycle them and try again.
By rotating the sensor around its vertical axis, the sensor can be identified in the interface.
Because only two sensors are required in this scenario, as soon as all required sensors are connected, scanning can be stopped by sliding the “Sensors” column back to the left.
*Note: it is preferable to **stop scanning** when working with connected sensors.*

Orientation reset

The Xsens DOT offers the possibility to reset the horizontal orientation, i.e. the *heading*. This functionality *must* be used to realign sensors because when sensors are switched-on and are connected to the app, their heading can be different.
The process to do this is illustrated in figure 13.

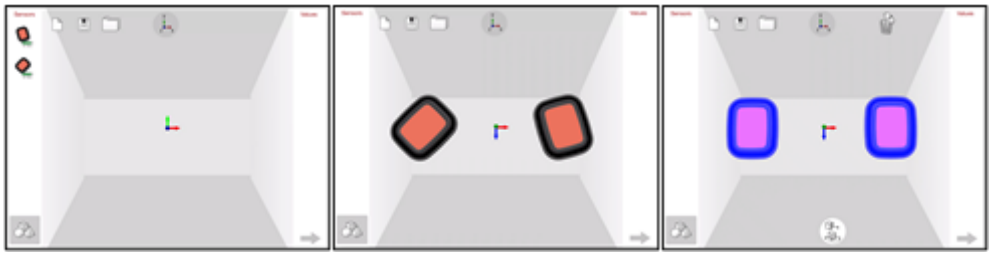

Figure 13: Selecting sensor objects and pressing the reset orientation button.

The steps illustrated in figure 13 are as follows.

1. Place the connected sensors on a flat surface (with no metal) and make sure that the sensors are aligned, i.e. pointing in the same direction. One can use the charging case for example to ensure the alignment.
2. Drag the sensors into the 3D view.
3. To be able to monitor the result, rotate the viewport by swiping down.

4. Select all 3D sensor objects. This will make the heading reset button (see figure 14) appear.
5. Press the heading reset button.

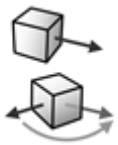

Figure 14: Heading reset button.

As a result, the sensors will now all point to the same heading.

Assign sensors

The next step is to *assign* the sensors to the segments.

To do this, choose the sensor that is to be attached to the upper arm and drag it onto that segment.

When the sensor is dragged over the object, the object will be selected.

Now drop the sensor and the object will become opaque, indicating that the orientation of the object is now controlled by a sensor.

The steps are illustrated in figure 15.

Figure 15: Dragging a sensor over an object will turn the object into a model.

Repeat the same for the sensor that is to be assigned to the lower arm.

Using the straps, the sensors can now be attached to a person.

When looking at the pose, it can be seen that the arm is not behaving as expected because the sensors still need to be *aligned* to their segment.

This “sensor-to-segment-alignment” can be performed as described in the next section.

Vertical alignment

For the created situation, it is sufficient (and required) to perform a *vertical alignment*. To do this, first select the whole kinematic chain by double tapping on of the segments.
This will make the Alignment button appear (see figure 16).

Figure 16: Alignment button.

Pressing this button will show the specific alignment action buttons. For the purpose of the arm model the *Upright* vertical-alignment button can be used (see figure 17).

Figure 17: Upright vertical alignment button.

By holding the arm in an n-pose position (hanging to the side) and then pressing the upright vertical alignment button, the local y-axis of the selected object is automatically aligned with the direction of the virtual gravitational acceleration of the scene (i.e. also the y-axis).
The movements of the kinematic model now mimic that of the arm of the person wearing the sensor.
The next step is to calculate the joint angle of the elbow. This is done by constructing a dataflow diagram as discussed in the next section.

Create dataflow diagram

Overview

To calculate the joint angle between the two segments of the created kinematic chain, the following steps are to be performed:

1. Select the kinematic values of the segments.
2. Create a dataflow diagram using default functions.
3. Connect the kinematic values.
4. Observe the joint angle.

These steps are described in detail in the following sections.

Method

There are several methods to calculate a joint-angle between two segments. In this example, the joint-angle will be determined by calculating the angle between the longitudinal axes of the two segments.
In the screenshot in figure 18, no sensors are attached, so there is no rotation placed on the

segments yet. In this situation the longitudinal axis is aligned to the vertical axis, i.e. the y-axis (green arrow).

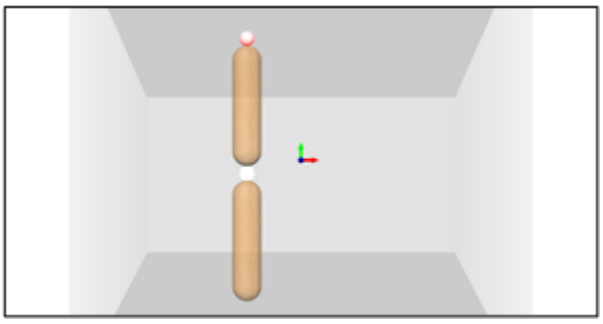

Figure 18: Unrotated segments.

To get the direction of this vector when the segment is rotated, we can rotate the vector using the orientation of the segment. For this access to the orientation is required as will be explained in the next section.

Kinematic values

Each object in the scene has a *sensing-point*. By default, this is located in the origin of the object. This sensing-point of an object has the following predefined *kinematic values*.

- q = orientation (quaternion)
- p = position (x,y,z coordinate)

These values are updated with each movement of the corresponding object and can be used in calculations. To be able to do this, first select the whole kinematic chain by double tapping on one of the segments and then select the Values button (see figure 19).

Figure 19: Values button.

Now the orientation and position values appear (see figure 20) and can be used in the dataflow diagram.

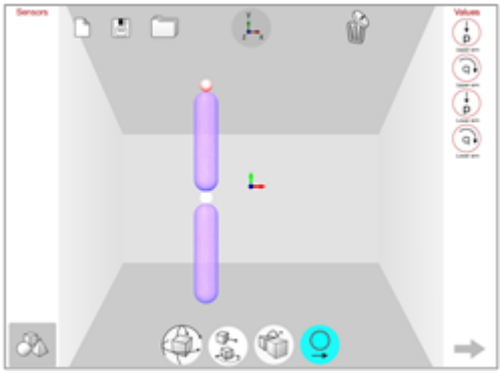

Figure 20: Selected kinematic values of the upper and lower arm.

Go to the dataflow diagram by pressing the arrow.

Rotated vectors

To calculate the direction of the rotated y-axis of a segment, the “Rotate vector” function node can be used, which takes a vector and quaternion as input. Adding this function node can be done by dragging the Functions button (see figure 21) onto the canvas.

Figure 21: Functions button.

The location can be chosen where the function must be placed (see figure 22). This can be changed later as well by dragging the function around.

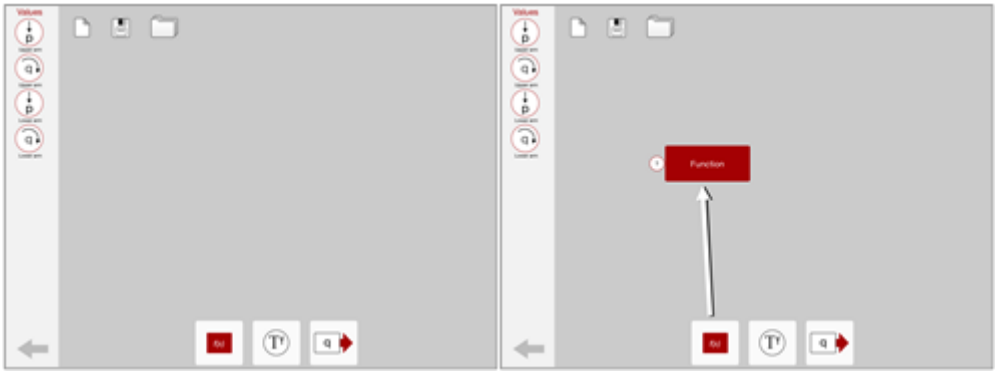

Figure 22: Dragging function node into the diagram.

Now a window will appear, showing all functions to choose from. Scroll down to find the function titled “Rotate vector” and press on it to select it to add it to the dataflow diagram as illustrated in figure 23.

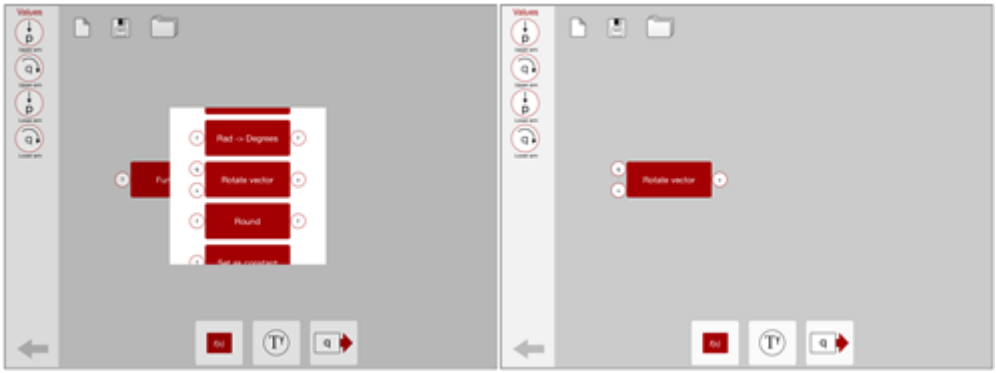

Figure 23: Selecting a function.

Since the vector rotation must be performed for both segments, repeat the same steps to add another “Rotate vector” function node.

Adding value nodes

The added “Rotate vector” function node requires a quaternion and vector as input, with the quaternion being the orientation of the corresponding segment and the vector the y-axis in this case.

Therefore the orientations of the two segments are to be added to the dataflow diagram by dragging the orientation nodes (see figure 24) onto the diagram.

Figure 24: Orientation node.

When dragged into the dataflow diagram, the *orientation node* can be connected to the input of function nodes, allowing the orientation to be processed (see figure 25).

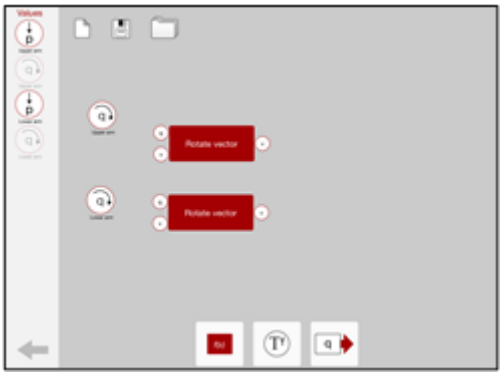

Figure 25: Both orientations are added to the dataflow diagram.

Adding the vector constant node

Since the new direction of the y-axis needs to be determined, the “Rotate vector” function needs the vector (0,1,0) as input. This can be added as a *constant* node to the dataflow diagram. Adding a constant node can be done by dragging the Constants button (see figure 26) onto the canvas.

Figure 26: Constants button.

Now a window will appear, showing all constants to choose from as illustrated in figure 27.

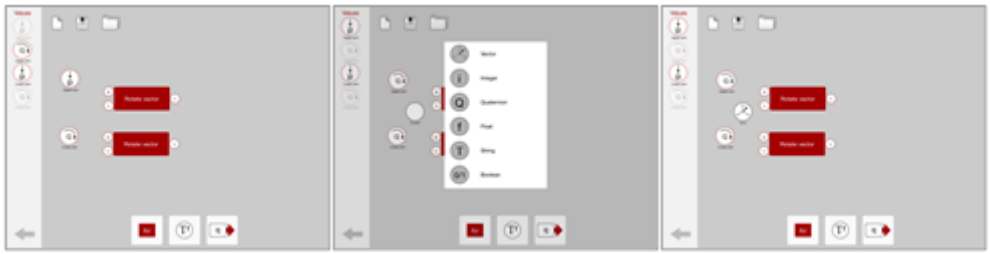

Figure 27: Adding a constant node.

By default the vector is (0,0,0). To customize it, select the added vector constant and press the properties button. Next, a window will appear showing the values that can be customized for the selected constant. Change the y-value to 1.

This process is illustrated in figure 28.

Figure 28: Changing the properties of a constant.

Connecting input nodes

Now the orientation node of a segment can be connected to the orientation input of the corresponding “Rotate vector” function and the y-axis vector node connected to vector input. Connecting nodes can be done by selecting the two points that need to be connected as illustrated in figure 29.

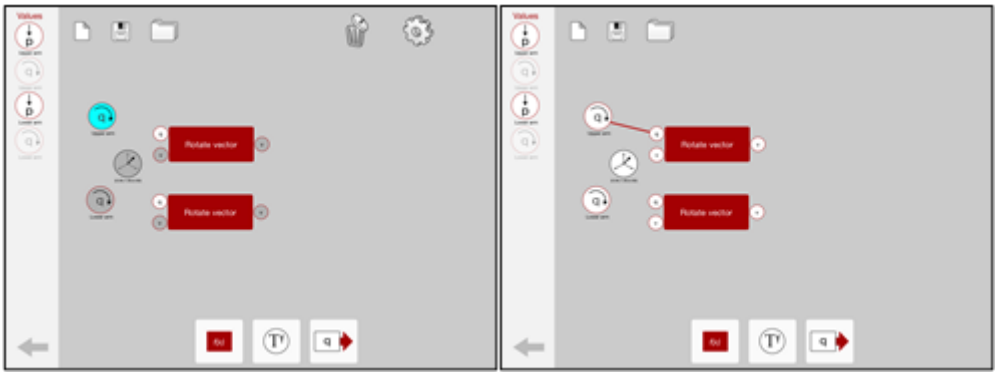

Figure 29: Selecting a node disables all locations that cannot be connected.

With everything connected, the resulting diagram is given in figure 30.

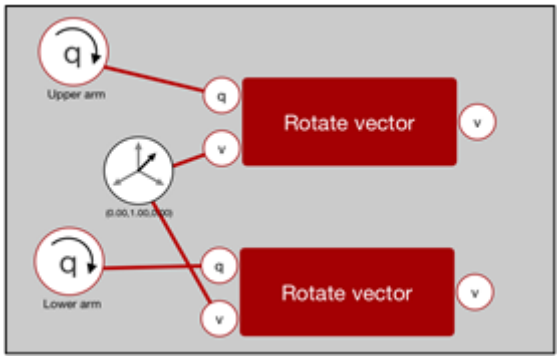

Figure 30: The vertical axis (0,1,0) is rotated by the orientation of the arms.

Calculating the angle

To determine the angle α between two vectors v_1 and v_2, the definition of the dot-product can be used:

$v1\ v2 = v1v2\cos\alpha$

To get the dot-product, the corresponding function can be used as illustrated in figure 31.

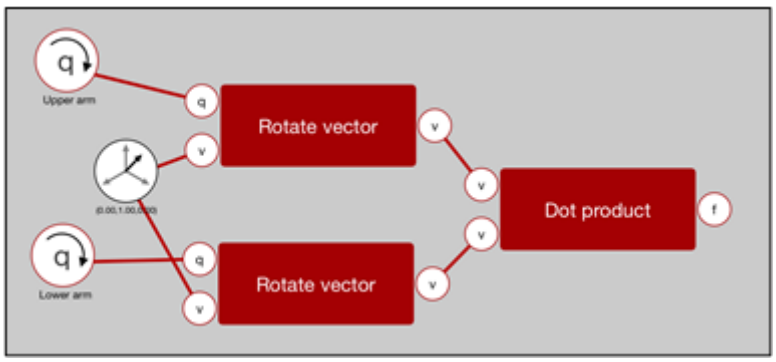

Figure 31: Determining the dot product of the two rotated longitudinal vectors.

Since the length of the vectors is 1, the angle in degrees can now be calculated by taking the

inverse cosine of the dot-product and converting it to degrees as illustrated in figure 32.

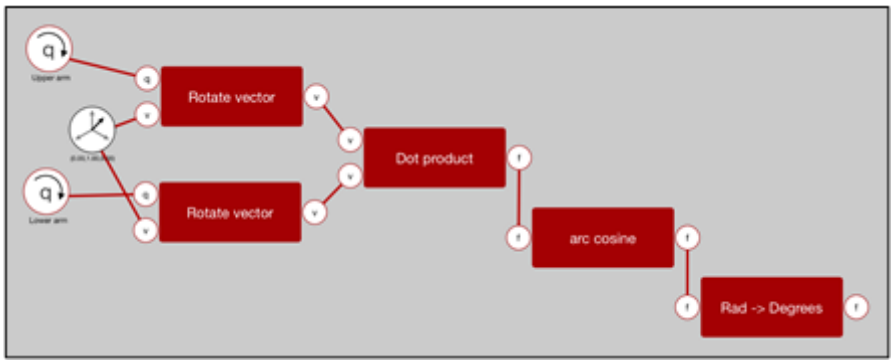

Figure 32: Complete dataflow diagram for creating the joint angle.

By dragging the nodes on top of each other, this can be placed in a single container (see figure 33).

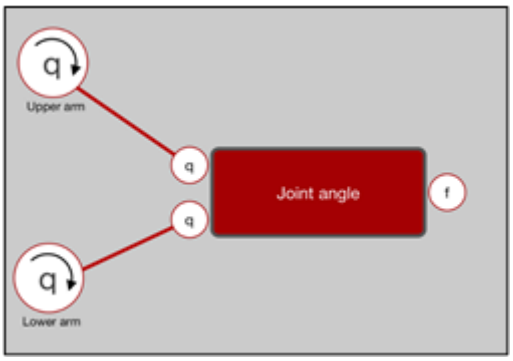

Figure 33: Created container for the joint angle.

By attaching a value label, the calculated joint angle can be seen in the scene. The next step now is to stream the value.

Streaming

Overview

To be able to plot the created joint angle value in a real-time graph, KineXYZ offers the possibility of streaming data as an XML or JSON string inside a UDP packet to an external device or companion app on the iPad itself, using UDP packets.
To do this, the following steps are to be taken:

1. Set the object and property name of the joint angle.
2. Extract the timestamp.
3. Set the object and property name of the timestamp.
4. Add a Network Streamer node to the dataflow diagram.
5. Configure the Network Streamer.
6. Connect the nodes.

These steps are described in detail in the following sections, but first some background information is given on the format of the UDP packet.

Background

The values generated in a dataflow diagram can be streamed on the network. Consider the example of the “Upper arm”.The position and orientation of this object can be streamed to the network using the network stream as given in figure 34.

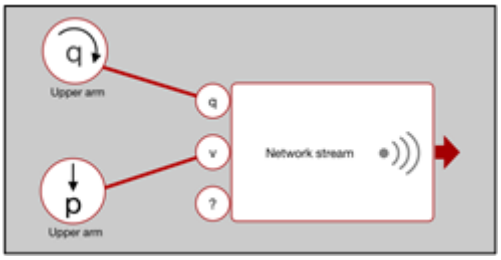

Figure 34: Streaming the kinematic values of a segment.

The data can be sent as an XML or JSON string inside a UDP packet.

The format of the XML string for this example is as follows:

```
<UpdateObject><Object Name="Upper arm"><Position><x>-0.1112</x><y>-0.0934</y>
<z>0.0200</z></Position><Orientation><x>-0.1511</x><y>0.9944</y><z>-0.0623</z><w>-0.0101</w></Orientation></Object></UpdateObject>
```

The bold red are the scalars of the actual kinematic values. By default these values already are associated with an object and have a property name (“Position” and “Orientation”). However, this is only the case for these default values. For all other values the object name and property name must be specified explicitly.

How this is described in the following section.

Setting the object and property name

The problem with streaming a calculated value like a joint angle to an external application is that the generated value itself is not associated with a specific single object and no property name is specified either.

This means that if the output would be connected to a network stream, nothing would be transmitted. It is therefore that an object name and property name must be specified for the calculated value.

To give some background first, the following describes how the object and property name are used to create the XML string.

Note first that all values in the diagram have the following properties:

Field	Type	Description
timestamp	float	The system time at which the value was received or generated, expressed in us.
objectName	string	The name of the object associated with the value.
propertyName	string	The name of the property that is represented by the value.

The XML string starts with the following:

```
<UpdateObject><Object Name="OBJECT_NAME">
```

Next, for each property associated with the object, the XML representation of the property will be added. This starts with:

```
<PROPERTY_NAME>
```

The XML representation of the property ends with:

```
</PROPERTY_NAME>
```

When the XML representation of all the properties are added, the XML string ends with:

```
</Object></UpdateObject>
```

Now that we know how KineXYZ generates an XML string, assume that we want to stream a joint angle and that the XML-string must be as follows:

```
<UpdateObject><Object Name="Joint">
<angle>angle</angle>
<timestamp>timestamp</timestamp>
</Object></UpdateObject>
```

To explicitly set the object name, the “Set object name” function is used. Likewise, to set the property name, the “Set property name” function is used. These functions take a string as input and a value for which the name needs to be set.
The diagram to set the object name to “Joint” and the property name to “angle” is given in figure 35.

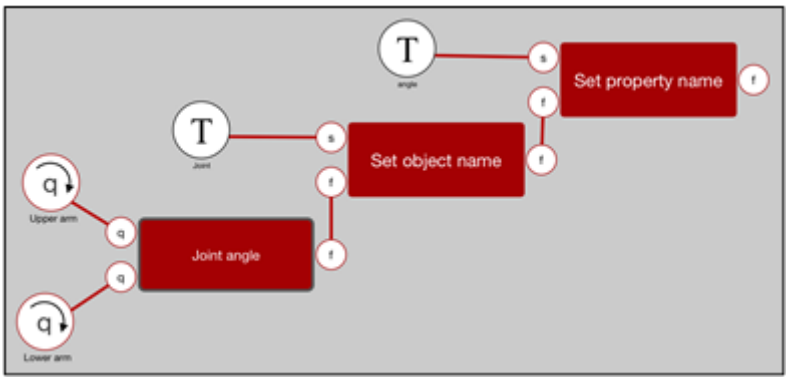

Figure 35: Setting the object name to “Joint” and the property name to “angle”.

The output of this diagram now generates a float value with the object name and property name filled in.

Extract the timestamp

The next step is to extract the timestamp as a value. This can be done by using the “Timestamp” function and extending the diagram as given in figure 36.

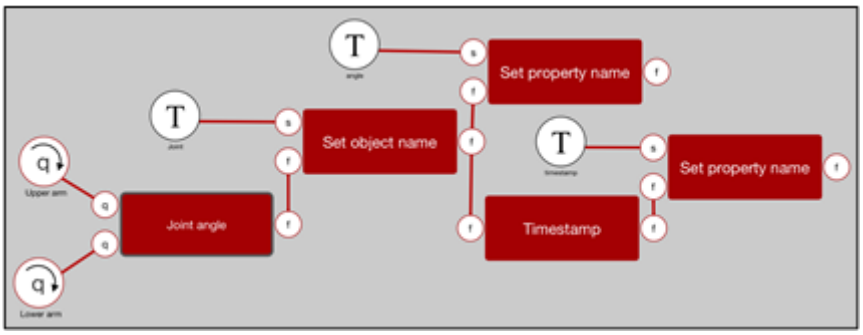

Figure 36: Extracting the timestamp and adding it as a property.

Both outputs can now be connected to a network stream as illustrated in figure 37.

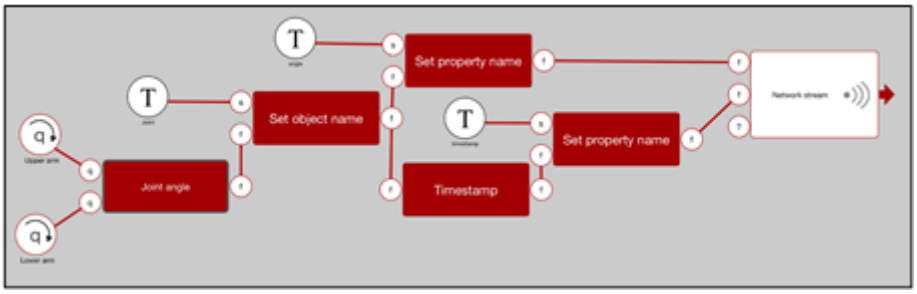

Figure 37: Streaming the two properties.

Since both properties now have the same object name, they are packed as a single packet.

Configure the network stream

Note that the default *format* of the network stream is specified as "JSON". If the XML format is required, this must be changed in the properties of the network stream (see figure 38).

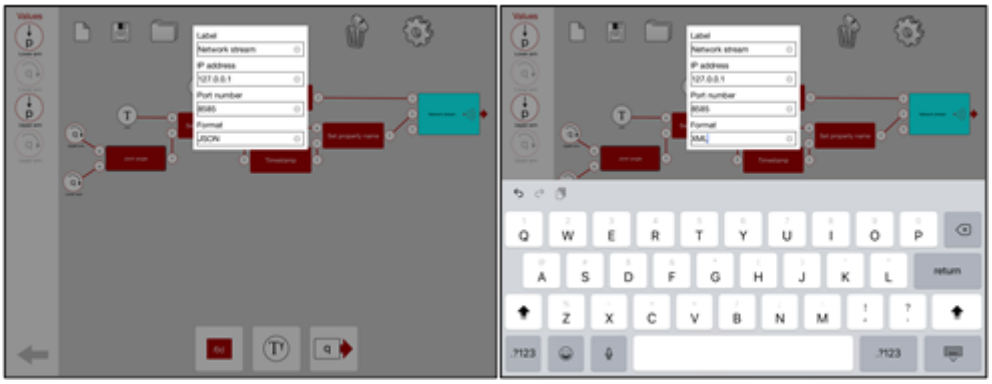

Figure 38: Properties of a Network Stream node.

Note: the displayed IP address 127.0.0.1 belongs to the host name "*localhost*", so on the device itself. This host name can also be typed explicitly if so required.

***Note**: because KineXYZ works on updates, a Network stream must have received updates for* all values *for the object. Otherwise no UDP packet will be generated.*

Receiving

As XML

You can now build any app on any device in the network that will listen for UDP packets (i.e. is a UDP server) received on port 8585 and containing an XML string.

The diagram as developed in the previous chapters, generates a UDP packet containing for

example the following string:

```
<UpdateObject><Object Name="Left elbow">
<angle>93.0</angle>
<timestamp>123131.319</timestamp>
</Object></UpdateObject>
```

As JSON via Node.js

Alternatively, when the JSON format is used, a small Node.js[2] script could be written that creates a UDP server:

```
const PORT = 8585;
const udp  = require('dgram'); // Standard module, no install
required.
const ip   = require( 'ip' );  // Install this package via "npm install ip"

var server = udp.createSocket( 'udp4' );

server.on( 'message', function(msg, info)
{
   var data = JSON.parse( msg );

   // Process data here...

});

server.on( 'listening', function()
{
   console.log(`UDP server listening on ${ip.address()}:${PORT}`);
});

server.bind(PORT);
```

Since the data is available as JSON it can easily be processed, e.g. stored in a CSV file.

---

[1] https://en.wikipedia.org/wiki/Geometric_primitive

[2] https://nodejs.org/