Functional product specification

Table of contents

Summary

This chapter contains an overview of the functional product specification for KineXYZ. It gives a definition of what the product does and how it works from a user perspective.

Functionality

KineXYZ is an application development tool that enables all kinds of creative developers to build 3D interactive systems that use combinations of motion tracking sensors to capture and process movements of people, animals or other flexible structures.

KineXYZ is very easy to use. It does not require any specific mathematical or kinematic modeling knowledge. This knowledge will be built-up when working and experimenting with KineXYZ! This means that developers can focus on their strengths and domain knowledge. They don’t have to acquire specialist knowledge or invest a lot of time and money before being able to create innovative applications.

The unique features of KineXYZ compared to more traditional motion capture systems revolves around some concepts that are briefly described in the following and in more detail in their corresponding sections.

3D virtual scene

KineXYZ uses a 3D virtual scene which can be used to build kinematic models as well as getting real-time feedback on the pose.

Wireless orientation sensors

KineXYZ uses wireless sensors that measure their own orientation in the global coordinate system. These sensors use Bluetooth Low Energy (BLE) and inertial measurements. The sensor data is converted into synchronized orientations in the global right-handed coordinate frame of KineXYZ. These orientations are mapped on specified objects by assigning sensors to segments.

Assigning sensors to segments of the kinematic chain is almost a trivial task in KineXYZ. There is no need to have a fixed association between sensors and segments as in most motion capture systems. Although it is possible to configure this. Alignment is done very easily without necessarily requiring the object to take a predefined pose.

Kinematic modeling

KineXYZ enables the user to build a kinematic model of any kinematic chain. This can be done by adding objects to the scene that can be interconnected via joints to create kinematic chains. These chains are internally described as kinematic models that control the pose (position and orientation) of all the objects. The user can build these kinematic models in the virtual scene and can store and load them from the local file system or share them.

Dataflow programming

KineXYZ is the only motion capture tool in which mathematical models or algorithms can be created via a drag-and-drop method and using the concept of dataflow programming, resulting in a dataflow diagram. The immediate feedback can be used by the user to see whether the algorithm is implemented correctly. The kinematic models can generate the kinematic values that can be used by algorithms. These algorithms can be built using dataflow programming to create a dataflow diagram of interconnected function nodes. The dataflow diagram can also contain output-nodes that send the output of algorithms as UDP messages to external networked applications. These applications can use the data to implement a certain functionality. The data can also be visualized in the virtual scene by associated labels.

Content sharing

Users that work with KineXYZ can generate content: kinematic models, data flow diagrams and sessions (include cooperating kinematic models and data flow diagrams). This KineXYZ content can be shared via email or placed online for downloading. This offers the possibility for cooperation, an open learning environment and enables people to offer their work commercially.

Companion apps

Functionality can be easily added to KineXYZ by using KineXYZ companion apps using the multitasking (e.g. multitask split view functionality of iOS). Not all functionality is required by all users and not all required functionality can be foreseen. To keep the UX of KineXYZ intuitive and simple, specific functionality is offloaded to so called KineXYZ companion apps. Examples are storing data and real-time plots. These apps are deployed on the same device and they work together with the KineXYZ app. The data communication is done using output-nodes that stream to localhost.

User groups

For KineXYZ a number of user groups can be defined. These users can be categorized as developers and are identified in this document in order to be able to list the additional requirements that each group might have.

For each group the following is given:

- *A short description of the activities of the user group.*
- *How KineXYZ could be used.*
- *What the group needs.*

For KineXYZ the following user groups were identified:

- *Biomechanical research engineers*
- *Developers of interactive mobile applications*
- *Game developers and animators*
- *Kinetic artists*
- *Developers of virtual reality attractions*
- *Technical students*

Obviously there is not always a sharp border between these groups. Biomechanics researchers could also be interested in studying audiovisual feedback during rehabilitation exercises, game developers could be interested in using the system as a real-time controller in their prototype game, etc.

Biomechanical research engineers

Activities

Biomechanical research engineers are studying the mechanical aspects of the musculoskeletal system of humans and animals. Theoretical and experimental investigations are used and are aimed at reducing the incidence of unintentional injuries in the population. Biomechanical research improves understanding of the human body so that better tools can be built to assess the risk of injury and improving the characterisation of human biomechanical properties.
To perform these studies, movement analysis is performed using motion capture systems. The data measured by these systems are accelerations and orientations of body segments. In some systems this is combined with position data (e.g. is case of optical motion capture systems). All this measurement data is processed to obtain parameters like joint-angles, range-of-motion (ROM), center-of-mass (CoM) and forces. In most cases this processing and the analysis is done offline. Sometimes this data is combined with measurements from other systems such as recorded by force plates or video.

Proposition

The following summarizes the benefits of KineXYZ for biomechanics researchers:

- KineXYZ can be used by biomechanics researchers to easily build measurement systems in which algorithms are implemented that generate and process certain biomechanical values.
- By defining kinematic values and recording them in CSV format, they can perform offline analysis. Using default geometries they can build a kinematic structure.
- While setting-up the system, the output can be seen in real-time.
- Attaching and aligning the sensors is very straightforward. Proper alignment can be visually inspected.
- The measurement values can then be streamed to companion apps where the data can be recorded and analyzed. All this without requiring to write any programming code.
- All data is time stamped and as such it can be combined with data from other measurement systems if they use the same timebase.
- A powerful tool can be to use the companion app that records the data together with video.

Using KineXYZ it also becomes possible for researchers that are knowledgeable in certain areas other than biomechanics to start using the technology in their research.

- Easily build algorithms
- Easy attachment
- Easy alignment

- Visual inspection of alignment
- Synchronized and timestamped data
- Streaming
- Companion apps for recording and real-time graphs

Requirements

The following main requirements can be identified for biomechanical research engineers:

- *Setup evaluation*

Before performing the actual experiment in which the measurements will be done, it is essential that an indication can be obtained as to the quality of the measurements and an indication whether the system is setup correctly.

- *System mobility*

A lot of studies have been performed in the labs of biomechanical research institutes. Measurement systems often are optical motion capture systems. Although these studies give a good basis for hypothesis, the measurements are always limited to the lab environment. There is a great demand for systems that are able to perform measurements outside the lab and also in daily life. For this the system must be really easy to setup and low cost.

- *Sharing*

In the light of collaborating research teams, it must be really easy to share the models and algorithms.

- *Reproducibility*

The results must be reproducible and not depend on the expertise of the researcher.

Developers of interactive mobile applications

There is a great opportunity for developers on mobile platforms to use movements of the users as input controllers in their apps.

Activities

The apps are offered in a digital distribution platform (e.g. App Store or Google Play). The

following phases can be identified:

- Experimenting
- Prototyping
- Development
- Testing
- Deployment

Proposition

The objective of KineXYZ is to offer something for each phase.

- *Experimenting*

Can start experimenting and gaining knowledge on the possibilities of mobile 3D interaction concepts. They can prototype these concepts using the basic functionality in KineXYZ. Alternatively, they could stream the kinematic values to another app running on a device in their local network or locally on the device itself.

- *Prototyping*

A major constraint for the development of mobile 3D interactive applications is the lack of opportunities to experiment and hence the lack of knowledge about the possibilities. With the aid of KineXYZ developers are now starting to experiment with a wide range of concepts. While doing so, they learn about the possibilities and start envisioning the potential of the technology. They use this to create solutions for their professional customers and start building apps.

- *Development and testing*

Part of development will be testing the version that is to be released.

Requirements

The most clear requirement for developers is that KineXYZ offers a means of quickly using the data generated in the first prototypes of their application. This can be done using the streaming functionality of KineXYZ.

Game developers and animators

Although there is a lot of stock animation available online, the animator working on a game or movie can also use KineXYZ to create more lifelike animations quickly as described in this chapter.

Activities

Animation is a large field, but the basic task of an animator is to specify specific motions for objects in a virtual environment (game, movie, cartoon). This does not only include human or animal figures, but also inanimate objects such as doors, planes, cars, etc. It also is not restricted to 3D, but also 2D and even sometimes 1D. These motions can be standalone, combined or stitched together.

Proposition

Although there is a lot of stock animation available online, often the animator working on a game or movie is faced with the problem that the animation needs to be extended with a very specific gesture, e.g. that of an arm or door.

Using KineXYZ, this missing motion can be easily created. This makes the work of the animator much more like the work of a puppeteer. Thus making it possible to generate lifelike movements much quicker and with more quality.

By streaming kinematic values to their 3D game engine, they can use the system to control and record the movements of in-game objects and characters in order to get a more lifelike behavior, i.e. perform puppeteering.

For KineXYZ to be interesting for animators, it is necessary for the data to be available in their animation tool. There are two ways in which this can be achieved:

- *Real-time streaming*

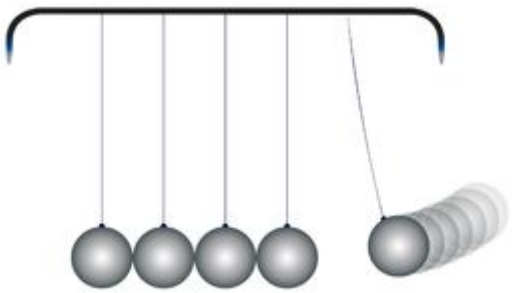
  By streaming to a game engine that takes the JSON data stream and uses it to animate objects in real-time. This animation can then be recorded if so required.

- *KineXYZ companion app for recording*

  By offering animators a companion application that is able to record the motion capture data in an interchangeable format. The file-format of choice is COLLADA which is offered by the Khronos Group as an open XML-based schema to make it easy to transport 3D assets between applications.

Requirements

Clearly, the animators need an easy way in which they can have KineXYZ interface with their animation tool. This is done easily via the KineXYZ output streamers as described in more detail in “KineXYZ Data flow diagram”. To support this group, KineXYZ also supplies default scripts (e.g. C# script for Unity) that gives an implementation example.



Kinetic artists

There is a growing group of artists who engaged in kinetic art. As the name already implies; these artists are interested in using the movement of objects to create art.

Activities

Kinetic artists create kinetic sculptures that generate light or sound.

Proposition

Kinetic artist can use KineXYZ to measure the movements of their kinetic sculptures. They can then use the generated kinematic values from their to control or generate any combination of motion, sound or light.

Requirements

For kinetic artist the tool must be really straightforward. Although most kinetic artist can have a technical background and some are even seasoned programmers.

Developers of virtual reality attractions

Lately, many initiatives are being launched to build attractions in which virtual reality plays a role. Recently virtual reality attractions and rooms are built in which visitors can undergo an immersive experience.

Activities

While developing these attractions there is a lot of focus on the innovative experience. This means that developers need to be able to conceptualize ideas and experiment with the user experience.

Proposition

Although probably for the final product requires some additional engineering, developers can use KineXYZ to experiment with concepts. Especially the possibilities to very easily build algorithms to capture motions like for example walking, climbing, and swimming are interesting.

Requirements

In the settings it is essential to be able to quickly assign and align sensors.

Technical students

Technical students from most subjects can be required to master some level of mathematics that include vector algebra. Also here KineXYZ can be beneficial

Activities

Most of the technical studies at educational institutes involve some level of mathematics that include vector algebra. Students can struggle with the intuitive understanding of this material, which they do require before being able to work with the equations. To obtain this understanding, they need to put in a lot of effort and work with prefabricated examples that not necessarily appeal to them.

Proposition

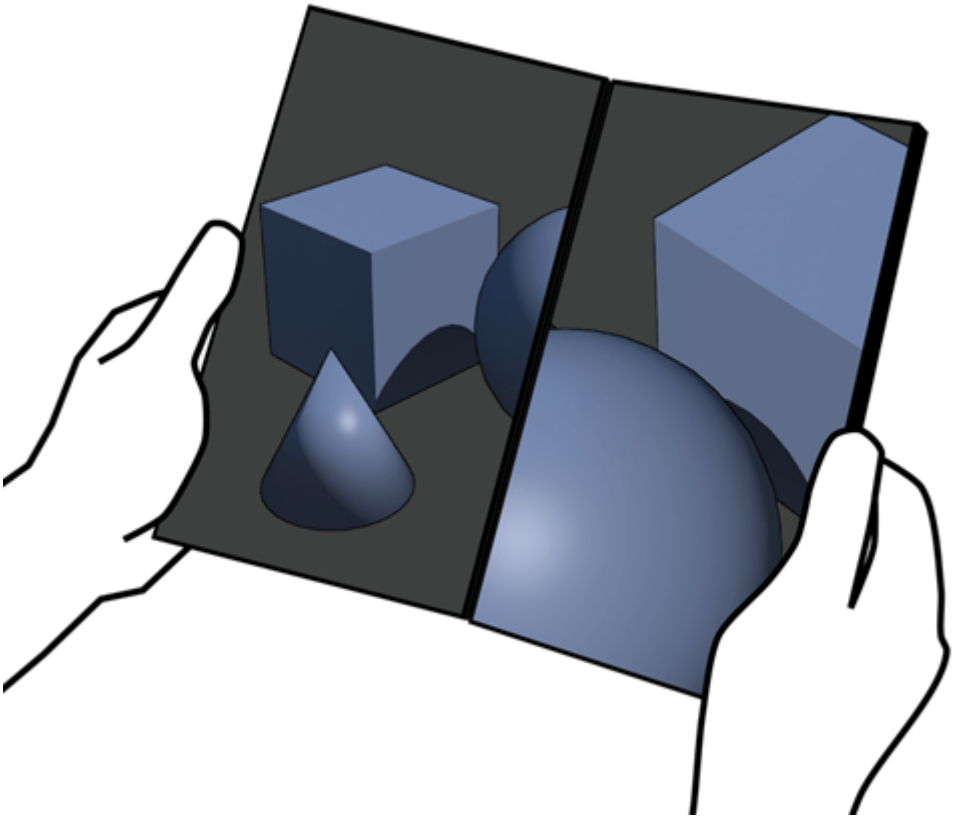
KineXYZ is an ideal tool for technical students to get hands-on experience with the basics of vector algebra. KineXYZ can be used by students to build models and algorithms that use the orientations and positions of the segments and analyse the output. They can make this models according to their preferences and are free to experiment.

This makes it much easier for them to understand concepts behind for example rotation and translation. Furthermore, models and diagrams are easily shared to create a cooperative learning experience.

Requirements

Apart from being really easy and straightforward to use, the most important requirement for having KineXYZ used by technical students is the possibility to share content.

3D scene



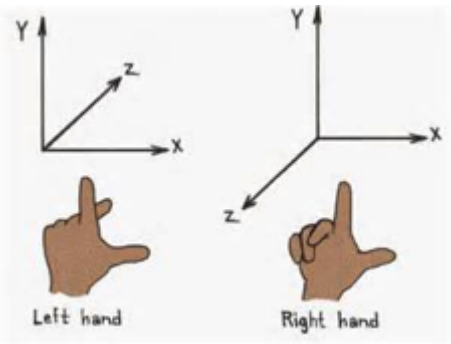
This section describes the specification of the virtual scene as a used in KineXYZ.

Coordinate system

A very important aspect of any 3D software (modeling or animation) is the definition and direction of the coordinate system, more specifically the direction of the axis and the handedness.

Handedness

A 3D coordinate system has a certain handedness. As such there is the left hand and right hand coordinate system as illustrated in the figure below:



Apart from the handedness, the direction of the y-axis is also important. In most cases y is up, but there are also systems that have z-up orientation.

The handedness of the coordinate system also defines the direction of a positive rotation. This is illustrated in the figures below:

	
In the right-handed coordinate system the direction in which your hand closes to make a fist is the direction of a positive rotation around any axis represented by the extended right-hand thumb.	In the left-handed coordinate system the direction in which your hand closes to make a fist is the direction of a positive rotation around any axis represented by the extended left-hand thumb.

Market

Unfortunately, not everybody uses the same handedness for the coordinate system, although since OpenGL uses a right-handed coordinate system, there seems to be a preference for right-handed. On the other hand, Unity3D uses a left-handed coordinate system.

The following lists the coordinate system of some of the mayor 3D animation tools/formats:

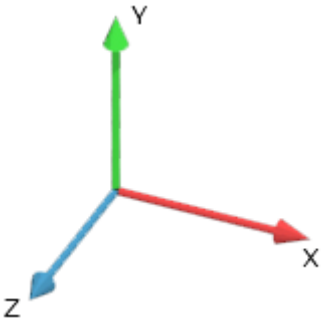
Tool	Handedness	Direction
Unity 3D	Left	y-up
Cinema 4D	Left	y-up
Direct3D	Left	y-up
Unreal	Left	z-up
3D Studio Max	Left	z-up
LightWave 3D	Left	y-up
COLLADA	Left	z-up
OpenGL	Right	y-up
Unigine	Right	z-up
SceneKit	Right	y-up
Maya	Right	Configurable (y-up / z-up)
Houdini	Right	y-up
Blender	Right	z-up
Motionbuilder	Right	y-up
Cheetah3D	Right	y-up

Axis visualization

To give a sense of orientation, it is necessary to visualize the axis.

Coloring

To give a sense of orientation, it is necessary to visualize the axis.
Most 3D software uses the de-facto standard of RGB coloring of the axis: x = red, y = green and z = blue. This is also done in KineXYZ and illustrated below.



Viewport manipulation

The viewport is defined as an area that is visible to the user. This is constructed by the position of the virtual camera in the scene. This chapter describes the possibilities for the user to manipulate this camera and thereby the viewport.

Rotate camera

The user can rotate the viewport camera by making a panning movement with one finger while not touching any other objects in the scene.

Reset camera

As a convenience for the user, it must be possible to undo the rotation and movement of the viewport camera.

Sensing

To actually control objects and segments of a kinematic chain in KineXYZ, the wireless sensors must connect and be associated with the proper objects.
This section contains all the aspects related to this.

Sensor communication

The wireless communication with the sensors is done using Bluetooth Low Energy. This chapter describes the different states of the wireless communication and how the user plays a role in setting-up the wireless connection.

System states

The wireless communication goes through a number of functional states:

State	Description
Scanning	The user enables the scanning. Any sensor that is switched-on and starts advertising itself is detected.
Connecting	Scanning is ongoing and sensors are detected and connected until the maximum number of sensors is reached or until the user disables the scanning.
Connected	All the required sensors are connected and the scanning is disabled.
Disconnected	A connected sensor is disconnected.

Scanning

When a sensor is switched on, it will start broadcasting its presence. To have the device detect the sensors, the user can explicitly start the BLE scanning in the GUI.

Connecting

Detected sensors are automatically connected until the maximum number of sensors is reached. This maximum number is determined experimentally and depends on the capacity of the device, i.e. the number of stable connections that the device can support.

When a sensor is connected this is graphically represented in the GUI.

Connected

When all sensors are connected, the user can disable scanning. Only when the scanning is disabled can the user proceed to the next step.

Once a sensor is connected, it is possible to select a specific sensor.

Inertial Measurement Units (IMUs) sensors need to stabilize their internal filter before they can yield accurate measurements. This sensor can be in the following states. These states are defined by the state of the internal filter and the association with an object in the kinematic model.

The states are given in the following table. Each state requires an action of the user:

State	Description	Action
Calibrating	This instruction means the sensor must be twisted so it passes all possible orientations. The best way to do this is to take the sensor in your hand and start rolling and turning it simultaneously.	
Stabilizing	This instruction means the sensor must be hold still. Keeping it still in the palm of your hand will suffice.	
Magnetic distortion	This instruction means the sensor should be relocated because it is currently too close to a metal object. During actual use, a minimal distance of 150 cm between the sensor and metal objects is advised.	
Ready	The internal filter is ready and the measurements can be reliably used. As a final step the sensor needs to be associated with an object.	
Associated	The sensor is associated with an object in the kinematic model.	None

Disconnecting

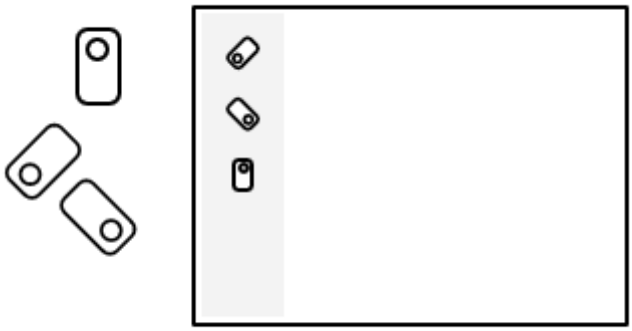
When a sensor disconnects, it is removed from the GUI.

Identification

There are several ways in which sensors can be identified.

Rotation

An easy method to identify sensors is to actual show the rotation of the sensor in the GUI, as illustrated in the following figure:



3D visualization

It must be possible to drag the sensor into the scene whereby a model is created that represents the sensor. In this way the proper orientation of the sensor in the coordinate system can be demonstrated.

Sensor association

When an object is placed in the scene, a connected sensor can be associated to it. This can be done manually or automatically.

Definition

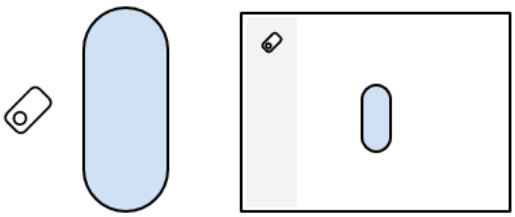
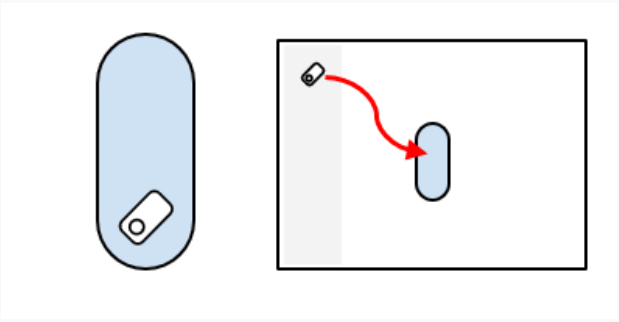
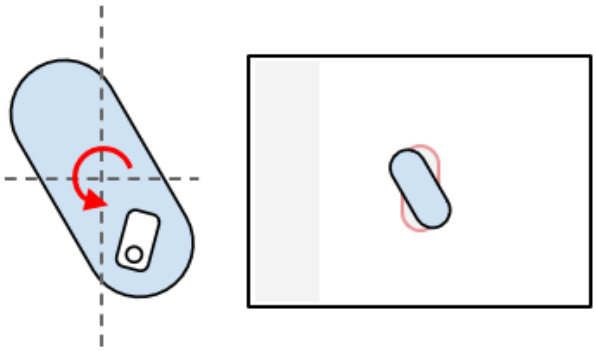
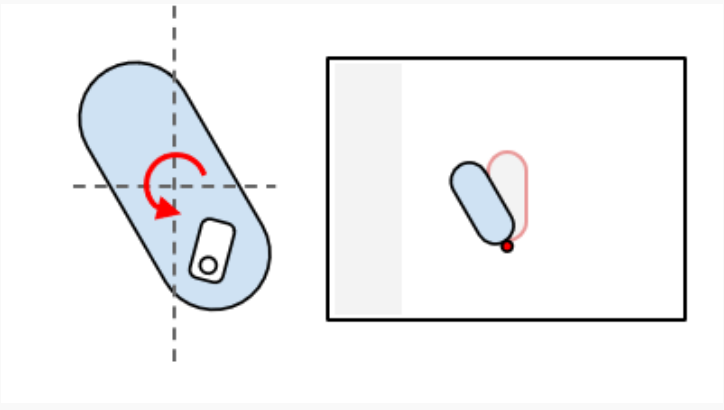
When a sensor is ready to be used, the next step is to associate it with an object in the kinematic model.

When the sensor is associated with an object, the sensor will disappear from the scene. If the object is not a segment in a kinematic chain, this will make the object rotate around its

centerpoint. Otherwise, the object rotates around one of its joints.

Associate sensor

The above is illustrated in the following figures:

The sensor is connected, but not attached to the physical object yet.	
The sensor is attached to the physical object and has to be associated with the corresponding object in the scene.	
After the sensor is aligned properly to the object the rotation of the object is mimicked in the scene by the object rotating around its centerpoint.	
When the object in the scene is part of a segment, it will rotate around one of its joints.	

Association

As stated, when a sensor is ready to be used, the next step is to associate it with an object in the kinematic model. This association can be done manually. There are two locations in which this can be done: the sensor connection window or the virtual scene.

To be able to associate sensors with objects in the sensor connection window, all the object names (not empty strings) are shown. Hereby the user can then indicate which sensor belongs to which object.

The association in the virtual scene is described in “Scene” and involves dragging a sensor on top of an object in the scene.

Disassociate sensor from object

It must be also possible for the user to disassociate the sensor from the object again. To do this, the user selects the object and indicates that the sensor must be disassociated.

The sensor will then reappear in the GUI and can be associated with another object if so required.

The sensor will then reappear in the GUI and can be associated with another object if so required.

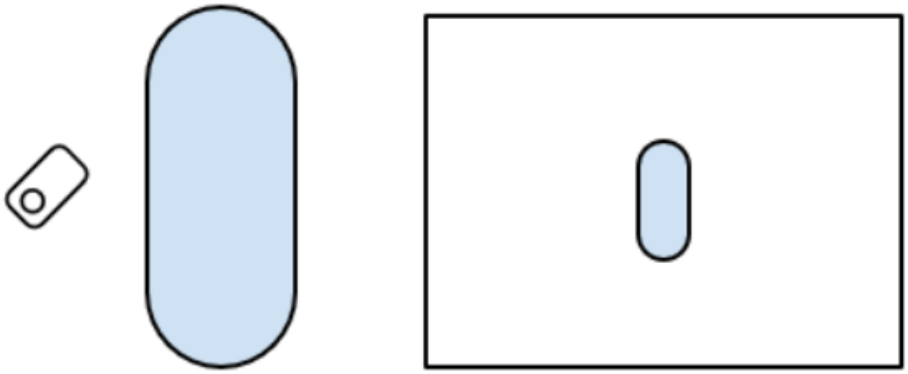
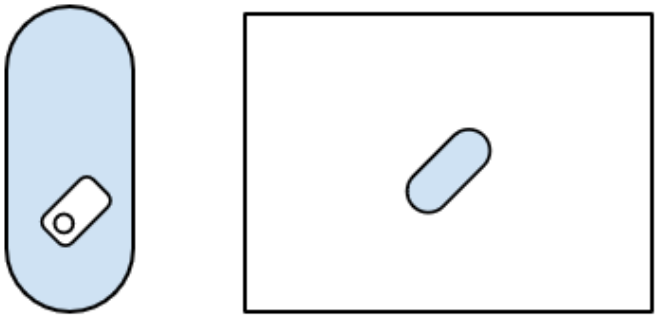
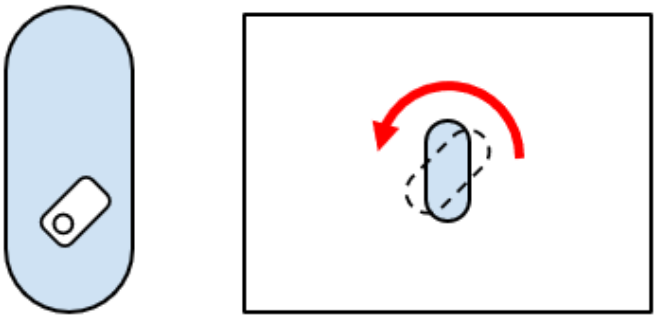
Alignment

For ease of use, when attaching the sensors to segments in KineXYZ it should not necessary to have a predefined location or orientation for the sensor. However, this does mean that the system does not know how the sensor is rotated relative to the segment. A simple alignment step is introduced to be able to determine this.

Object orientation

The orientation of an object is static and is initially determined by the default orientation assigned to the object when it was added to the scene. This default orientation can be changed manually.

This means changing its orientation relative to the scene’s coordinate system.

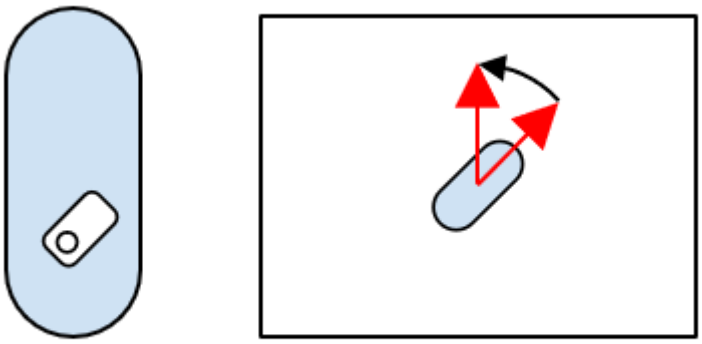
Consider a sensor and an object as illustrated in the figure on the right. When the object is not rotated, the representation of the object in the scene is identical.	
When a sensor is attached to an object, the orientation of the sensor is applied to the object in the scene. Since the sensor is not aligned with the object, the orientation of the object in the scene no longer correspond with the actual orientation of the object.	
To compensate for misalignment, a rotation offset must be applied on the object by rotating it counterclockwise in the example as illustrated on the right. Note that the same result would be achieved if the object was rotated with the proper amount before the sensor was attached.	

This is the basic mechanism in which the alignment can be performed. However, it is clear that

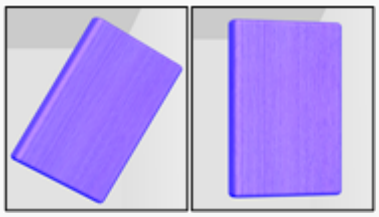
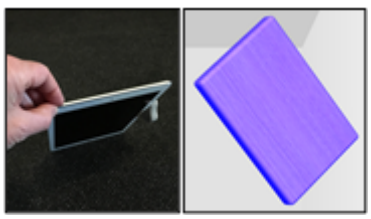
it will be difficult for the user to determine the precise amount of rotation. Especially, since we only shown an example in 2D; in practice, the sensor could for example be placed on the side which would make it even more difficult to determine the rotation offset. It is therefore that the vertical and horizontal alignment functionality is required as described in the following sections.

Vertical alignment

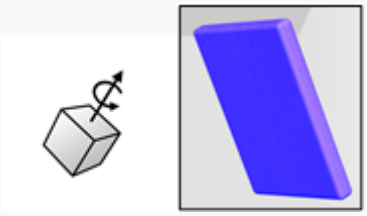
When a sensor is attached to an object, the user can place the object in an upright position. Since the user knows that the object is now in an upright position, this can be indicated to the sensor (e.g. by pressing a button). The system then uses this to determine the required rotation.



Horizontal alignment

Consider a situation as given in the figures on the right showing the object with the attached sensor and the virtual representation of the object in the scene. It is clear that first the vertical alignment as described in the previous section needs to be performed.	
After performing the vertical alignment, the orientation of the object is as illustrated in the figures on the right. At this moment it is clear that the vertical axis are aligned, but it is not clear yet whether that is the case for all axis.	
When the object is tilted forward, it becomes clear that the horizontal axis is not aligned. Since the user knows that the object is now tilted forwards, this can be indicated to the system. This can then be used by the system to perform a horizontal alignment.	

The function can automatically make sure the horizontal axis of the virtual object and the horizontal axis of the sensor are aligned correctly by rotating the virtual object around its vertical axis. The resulting orientation of the virtual representation of an object is corrected as can be seen in the rightmost figure.



Heading offset

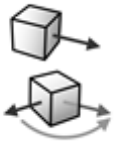
The vertical and horizontal alignment (as described in the previous sections) change the orientation of an object while keeping the orientation of the sensor the same. However, sometimes it can be required to change the heading.

Manual heading offset

As part of the properties of an object in the scene, it's heading offset can be specified. This can be useful in cases where the heading cannot be determined properly.

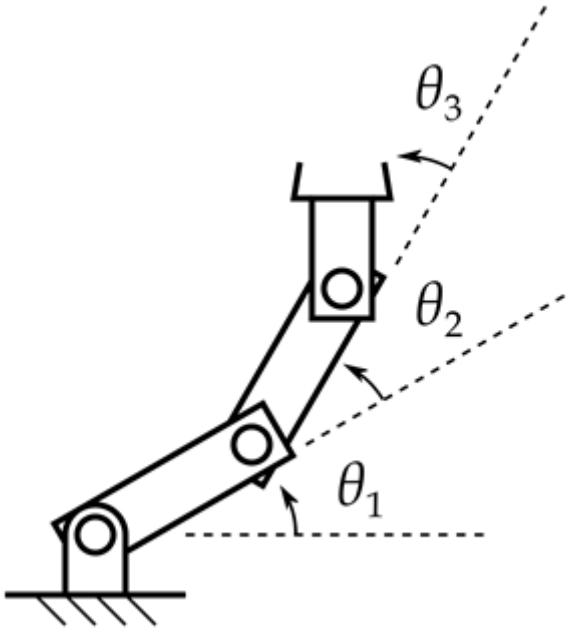
Align to object

There are situations in which the x-axis of one or more objects needs to align to the x-axis of another specific object. To do this, the user selects all the relevant objects and indicated that the alignment needs to be performed, e.g. by pressing a button.



The order of selection is of vital importance. All objects will get the heading of the first selection.

Kinematic modeling



The objective of KineXYZ is to enable developers to build 3D interactive systems that use combinations of motion tracking sensors to capture and process movements of people, animals or objects.
One of the most important functionalities required to do this is being able to model kinematic chains.
This section contains the description of the KineXYZ kinematic modeling functionality.

Kinematic chain

Before describing the method to model a kinematic chain, it is essential to clearly define the components of a kinematic chain and the required functionality.

Segments

Capturing the movements of people, animals and other flexible structures basically means sampling the pose (position and orientation) of all the *segments* from the underlying kinematic chain.

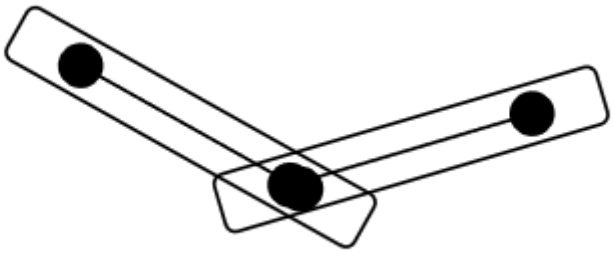
For humans and animals this kinematic chain is the bone structure, but the kinematic models are not restricted to these types of chains only: a user can build a model of any kinematic chain.

Joints

To measure the orientation, motion tracking sensors are attached to the segments and assigned in the model (see “Sensing”). However, in most practical setups it is very difficult to measure the position of the segments reliably and accurately, especially when the setup must be robust and mobile.

> *Note: in a fixed infrastructure of an optical systems, the position of an object can be measured very accurately, but this is not a practical and cost effective technology for mobile or consumer grade systems.*

Additionally, even when the position would be measured reliably we would immediately introduce a scaling problem. This means that the models in the virtual environment would need to have the same relative scaling otherwise the pose would be incorrect. An example is illustrated in the following figure.



This would not only complicate matters, but also would give users less flexibility in setting-up the kinematic model. It is therefore that the segments in this kinematic chain are interconnected by *joints*. Hereby a tree-like structure is created in which the position of a segment is influenced by the orientation of the other segments to which it is attached.

> *Note: this does not mean that position measurements could never be used. Potentially setups can be envisioned where position information could help solving unknown orientations. But the sensors in KineXYZ v2 currently are unable to measure position information*

In a modeled kinematic chain, the movements are already restricted by the range-of-motion (ROM) of these joins and the shape of the segments. It is therefore that in KineXYZ only ball-and-sockets joints are used.

Anchor-point

To determine the pose unambiguously (i.e. solving the kinematic model), a kinematic chain is

hierarchically structured. This means that there is a certain point in the kinematic chain of which the position (momentarily) is fixed. This is denoted the anchor-point and is the starting-point for solving the kinematic model.

> *Note: having one anchor-point means that only tree-like (i.e. so called 'open') kinematic chains can be modeled. Because of the associated complexity (also for the user), closed kinematic chains that have multiple anchor-points are not supported.*

In KineXYZ the anchor-point can be explicitly set by the user.
As stated, the anchor-point only has to be momentarily fixed to be able to solve the kinematic model at that particular time step. Over time, the anchor-point can move or be changed. This can also be done automatically. An example is walking in which case the anchor-point alternates between the feet. To be able to support this in KineXYZ, the anchor-point can also be set programmatically by a control algorithm.

Kinematic modeling

The objective of KineXYZ is to enable developers to build 3D interactive systems that use combinations of motion tracking sensors to capture and process movements of people, animals or objects. One of the most important functionalities required to do this is being able to model kinematic chains. This chapter describes the kinematic modeling functionality.

Objective

The objective is to model a kinematic chain, i.e. creating a kinematic model. Note that it is not the objective of KineXYZ to create a pleasing and anatomically correct virtual character as is the case in 3D animation tools.
In most 3D animation tools, building kinematic models means rigging a virtual character. Often this means manually specifying the interconnections between the segments using some GUI controls or event text based specifications (e.g. XML). This works fine in those tools since rigging does not have to be done very often.
This is in great contrast to the purpose of KineXYZ. The tool is meant for people to experiment with modeling all kinds of kinematic chains and often involves changing the kinematic model.

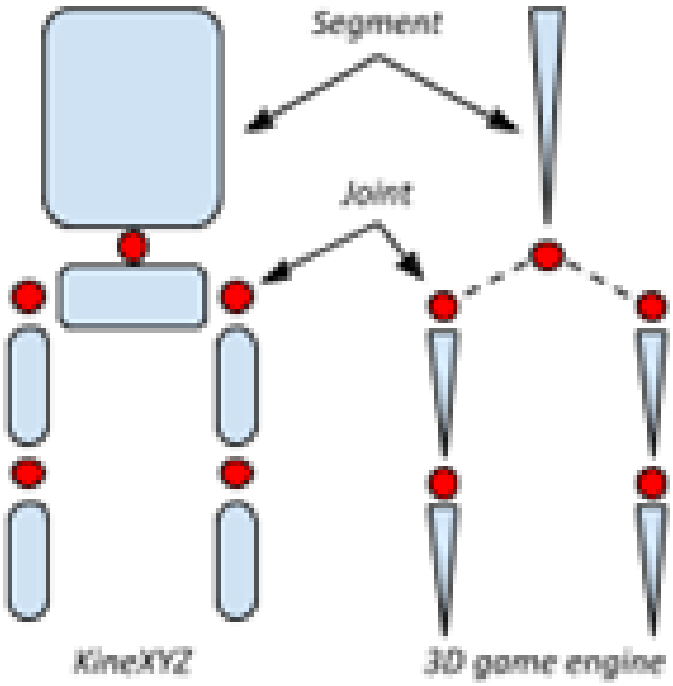
Scene

To create the most intuitive interface, to model the kinematic chain, a 3D scene is created in the

same way as done in 3D animation tools. Similarly, the user can place 3D objects representing and modeling the segments.
Also, position and orientation play an important role. Being able to manipulate these in an intuitive way is essential.
By default objects have no substance, so other objects can pass through them.

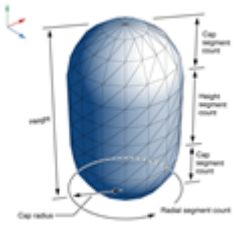
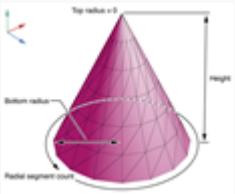
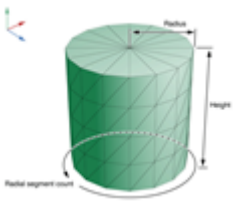
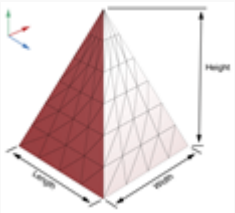
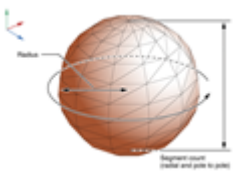
Parameterized primitive objects

The first step in modeling the kinematic chain is representing the segments. In most 3D animation tools, kinematic chains are modeled as ‘stick-figures’. The problem with this approach is that it requires additional effort for a user to model using sticks then with actual geometric objects. This is illustrated in the following figure on the right.



The following geometric primitives are defined:

Primitive	Parameters	Icon	
Cube	Width, height, length, chamfer radius		

Capsule	Cap radius, height		
Cone	Top radius, bottom radius, height		
Cylinder	Radius, height		
Pyramid	Width, height, length		
Sphere	Radius		

Connection-points

The objective of KineXYZ is to make it as easy as possible for the user to model a kinematic chain. After choosing a number of proper geometric primitives that will represent the segments and adding these to the scene, the next step is to connect the segments. This is done by using connection-points.

Predefined locations

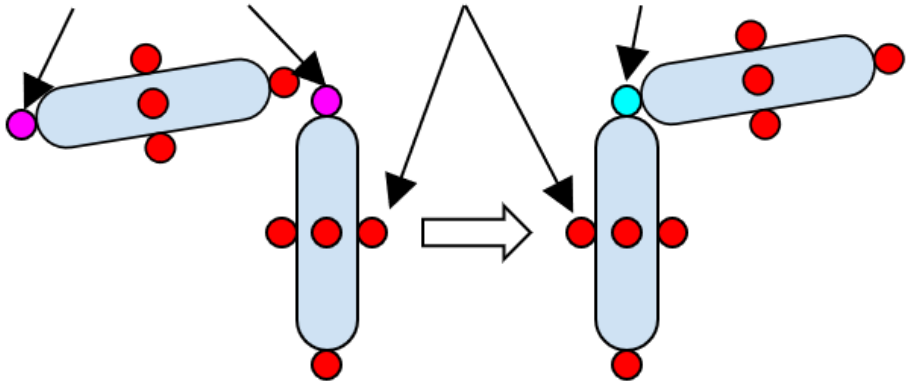
To make it very easy for the user, each geometric primitive has 6 *connection-points* at predefined locations. These locations are specified in the following table:

Primitive	Connection points
Cube	Center point of all faces.

Capsule	Top and bottom and mid-way on both sides of the x- and y-axis.
Cone	Top and centerpoint of the bottom-plane and both sides of the x- and y-axis.
Cylinder	Centerpoint of the top-plane and bottom-plane and mid-way on both sides of the x- and y-axis.
Pyramid	Top and centerpoint of the bottom-plane and on both sides of the x- and y-axis.
Sphere	Poles and mid-way on both sides of the x- and y-axis.

Joining

Using the connection-points, the user can simply connect two segments by selecting one connection-point on each segment. The result is the creation of a *joint* as illustrated in the figure below.



The advantage of creating the kinematic model in this way is that a joint can be created with two clicks and the result is immediately visible.

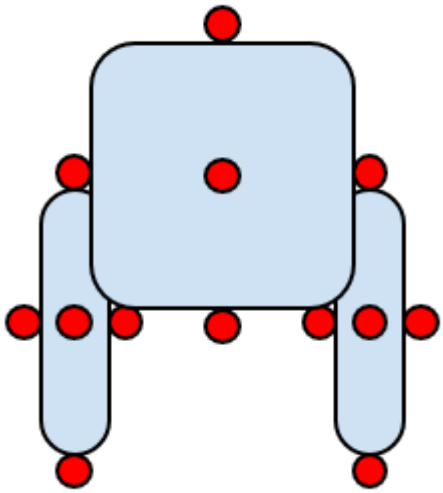
Combining objects

The default geometric primitives and the predefined locations of the connection-points will not

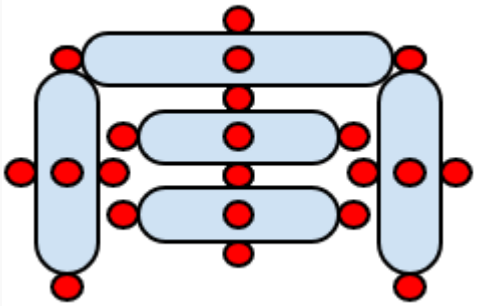
always suffice to create meaningful kinematic models. To solve this, the functionality is introduced that makes is possible to combine objects.

Problem description

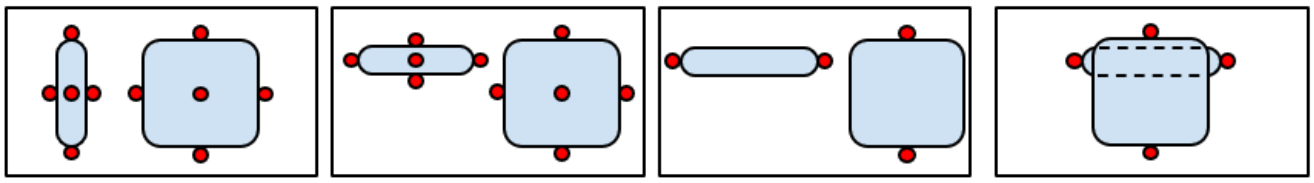
Consider an example in which a simple model must be created of the upper torso of a human and including the upper arms and shoulders. When this would be modeled using the standard primitive objects, the best possible result would be something as illustrated in the figure on the right. Obviously, this is not the result that will be very useful; the upper arms are at the wrong height and also need to be more separated from the torso.



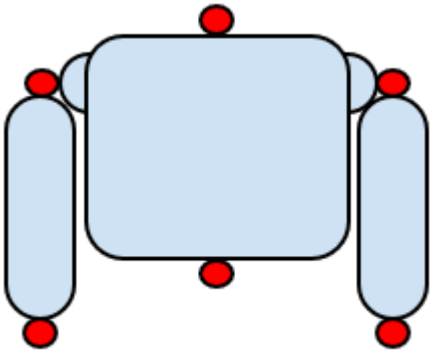
An option could be to use more objects as illustrated in the figure on the right. This obviously is more work and something must be defined for the orientations of the objects that together create the torso.



To solve the problem, geometric primitives can be combined to act as a single object. As can be seen in the examples above, this means that it must be possible for the user to manipulate the orientation of an object. This is discussed in more detail in the next section. It also creates a lot of redundant connection-points. Therefore it is also possible to delete connection-points. An illustrated example in which this functionality is used is given in the following figure. In this figure two objects are placed in the scene (1). The left object must become the shoulders and the right figure the thorax. The orientation of the left object is rotated (2). Next redundant connection-points are deleted (3). Finally, the left object is placed at the correct location and the two objects are combined (4).



The result is a new object that can be manipulated in the same way as the default geometric primitives.
Using this object, a more realistic models can be created very easily as illustrated in the figure on the right.



Not only primitive objects can be combined: also existing combinations can be extended. The resulting object can be manipulated as any other object (see next section) but is loses the geometry parameters of the contained objects.

Manipulating objects

To create combinations it must be possible to rotate and reposition objects and to change their geometric properties. Also, to identify the segments for later use, naming can be useful as well.

Properties

For objects in a kinematic model, the following properties are relevant:

- Object name
- Position
- Rotation offset
- Geometric parameters
- Scale

The possibility of changing these properties is discussed in more detail in the following

sections.

Object name

By default objects are given a name when it is placed in the scene. For convenience, this default name can be changed. The name of an object can be used to identify the object amongst other objects, which is required for several functions that are implemented in KineXYZ.

Position

The 3D position of an object is defined by the x, y and z coordinates. When the object is not part of a kinematic model, the position can be changed freely. However, changing the position of a segment in a kinematic model, can affect the position of other segments and vice versa.
To manually change the position of an object, the user has two options:

- *Visually*

To visually manipulate the position of an object, the user can touch and drag the object around in the scene. When the object is a segment in a kinematic chain, the whole chain will be dragged.

- *Numerically*

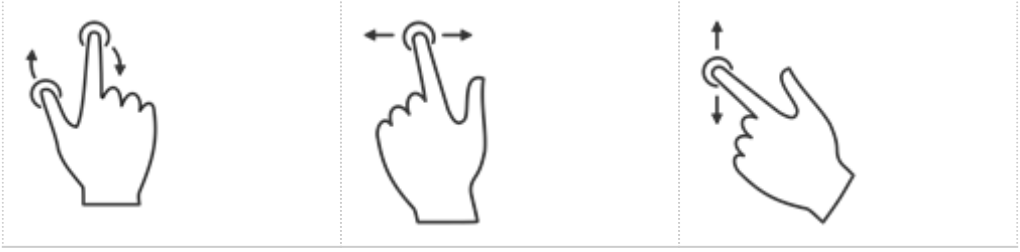
To numerically change the position, the user can select the object and specify the coordinates. Any changes are automatically visible.

Rotation offset

When an object is placed in the scene, it does not have any rotation. When a sensor is associated with it, the object takes over the orientation of the sensor.
When no sensor is associated with the object yet, it must be possible for the user to change the orientation also.
To specify the rotation, the user has two options:

- *Visually*

To visually manipulate the orientation of an object is done by selecting the object and then using the gestures as specified below:



Rotation around an axis perpendicular to the view plane is done by using a two-finger rotation.	Rotation around the axis vertical to the view plane is done by panning sideways with one finger.	Rotation around the axis horizontal to the view plane is done by panning up and down with one finger.

- *Numerical*

To numerical change the orientation, the user can select the object and specify the Euler angles (yaw, pitch, roll) in degrees. Any changes are automatically visible.

This orientation is specified as the *rotation offset* because it is applied on the object before the rotation of an associated sensor. Therefore the rotation offset can be used as a means of aligning the sensor to the object.

Geometric parameters

Depending on the geometric primitive, the object can have a number of geometry parameters. The user can change these parameters numerically by selecting the object and explicitly specify the values.

Scaling

The user can scale an object uniformly. The user can do this by selecting the object and explicitly specify the scale factor. Note that for a geometric primitive uniform scaling is the same as changing all geometry parameters with the same factor. For objects created as a combination of geometric primitive, scaling is the only way of changing the size.

Selecting the kinematic values

The algorithms implemented in the data flow diagram can use the position and orientation of certain objects in the kinematic model. However, not all kinematic values are used for all objects.

Since there can be a lot of objects in a kinematic model, for practical reasons it makes sense to have the user indicate from which object the kinematic values are to be used in the data flow diagram. By doing this, the list of kinematic values is limited in size which making scrolling through this list to find the appropriate kinematic value either not necessary or rather easy.

Kinematic values

The pose information of the objects in the kinematic model can be used as input for the data flow.

Object pose data

The pose information of the objects in the kinematic model can be used as input for the data flow.

For each object in the kinematic model the kinematic values are available. These are the position and orientation and can be used as input for an algorithm.

When the orientation changes, the position does not necessarily have to change as well. However, there are situations in which this is the case as described further in the next section.

Kinematic value updates

Typically, the orientation changes when a sensor is attached to the object. Alternatively, the orientation can also be changed manually by the user.

There are two reasons why a change of the orientation will result in a change of the position: when the object is part of a kinematic chain or when this is implemented to do so in the data flow diagram.

Changing the position will never result in a change of the orientation, unless of course this is implemented in the data flow diagram.

Timing

For an object to which a sensor is attached and that is not part of a kinematic chain, the timestamp of the kinematic value is derived from the timestamp of the incoming sample. However, when the object is part of a kinematic chain, ‘synchronized updating’ must be used. This method ensures that all segments are updated at the same timestamp thereby ensuring a proper pose of the kinematic chain.

Naming

The name-properties of the kinematic values are set by default: the object name corresponds to the name given to the object in the scene and the property name is either “Position” or “Orientation”.

Dataflow programming

In KineXYZ the concept of dataflow programming (or visual programming) is used to develop algorithms.
Dataflow programming is widely used. Some successful examples are:

- *TensorFlow from Google*
- *Mindstorms from Lego*
- *Scratch from MIT*
- *Max from Cycling '74*
- *LabVIEW from National Instruments*
- *SimuLink from MathWorks*

The objective of the algorithms that are implemented in the dataflow diagram is to take the generated data and use it to calculate new data that can be visualized or used in some other manner.
In the KineXYZ Builder algorithms can be built that use the kinematic values of objects to generate new data. As explained in this chapter, this is done using the concept of data flow programming.
The following chapters describe the concepts and components used in a dataflow diagram.

Algorithms

In the KineXYZ algorithms can be built that use the kinematic values of objects to generate new data. As explained in this chapter, this is done using the concept of data flow programming.

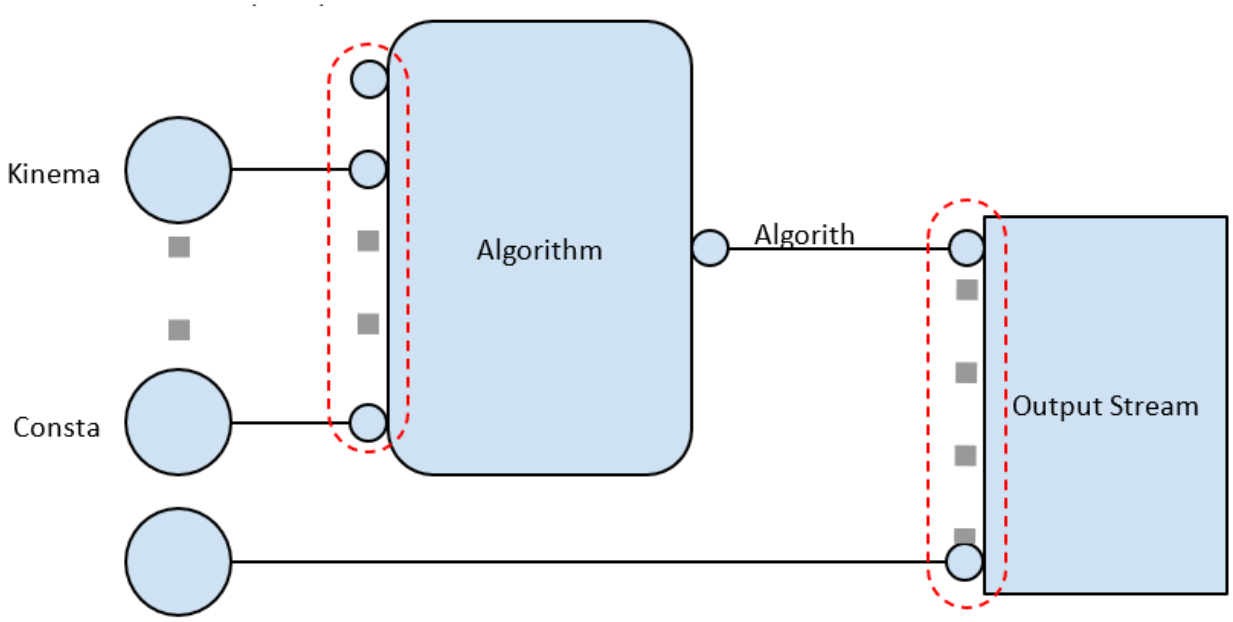
Graphical implementation

In KineXYZ an algorithm can be implemented graphically using the concept of dataflow programming.

Using a number of inputs, the algorithm calculates a single output value. These inputs can be kinematic values that are generated by the kinematic model, constant values (e.g. settings), output values of another algorithm or the output of input streams.

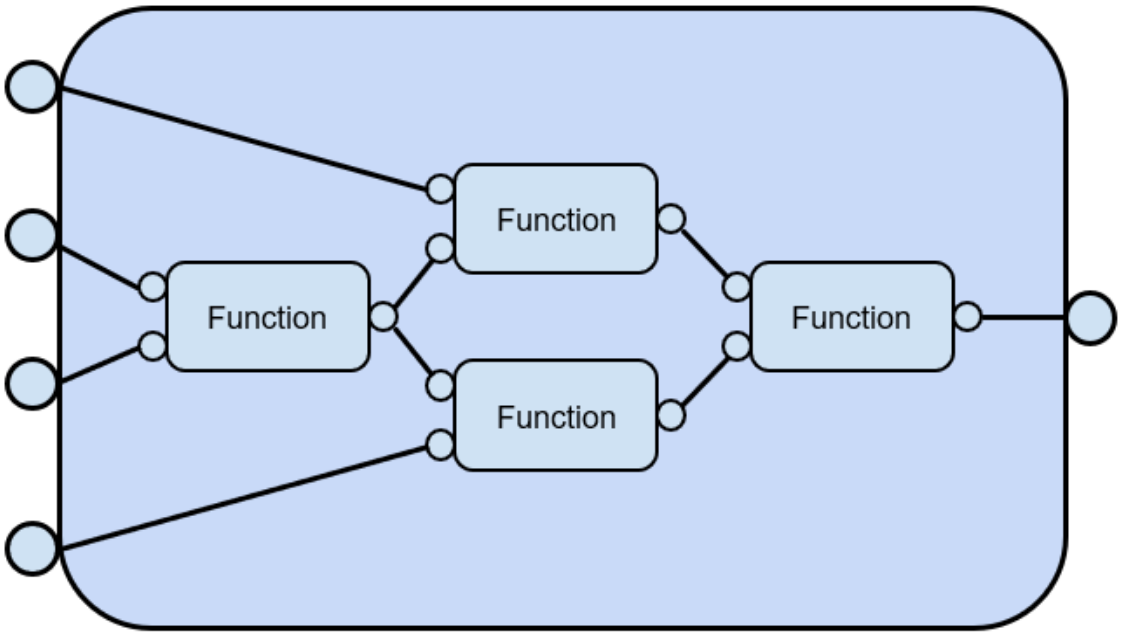
The output of one or more algorithms and/or kinematic values can be connected to an output stream which uses the values in some manner.

The following figure illustrates these possible connections.



Constructing the algorithm

In the data flow programming paradigm an algorithm can be constructed by having the data flow through a number of interconnected functions. This is illustrated in the following figure:



Timestamped values

All values used in the data flow diagram are timestamped values. Several types are defined.

Strict typing

An important consideration is whether or not values in data flow diagram should be dynamically typed like for example JavaScript. This would mean that any output can be connected to any input.

In the first instance this would seem like a good idea as it would offer a great flexibility. However, although there are functions that have inputs for which the type does not matter, most functions are not type agnostic and require a specific type of value on each specific input.

Different types of functions are discussed in the next chapter.

KineXYZ does define the following types:

Name	Description
Vector	A 3 dimensional position in the 3D right-handed coordinate system.
Integer	Integer value can for example be used as a size indicator, index or counter.
Quaternion	Rotation expressed as a (x,y,z,w).
Float	Floating point value. When specifying a value, the text PI can be used.

String	Text that can be used for naming or identifying or specifying an object.
Boolean	True (1) or false (0) value that is used in logical functions.

Timestamping

In the data flow diagram it is important to keep the notion of time. One reason is that the sensor data can be buffered in the local radio stack and not all sensors sample at the same time. This means that incoming data cannot be combined without taking the actual timestamp into account. Therefore, for every value the timestamp must be stored at which the value was generated.

Naming

It is possible to send values to external applications. Within the data flow diagram the meaning of each value can be traced, i.e. to which object it belongs and what it represents. This does not have to be an actual object in the kinematic model; it could also be abstract objects like for example “Left knee angle”. However, this knowledge is lost when the data is transmitted onto the network. Therefore, for each value it must be possible to specify an object and a property name.

In case of a kinematic value, the object name and property are already set.

Properties

Given the discussions in the previous sections, a timestamped value has a number of properties as listed in the following table:

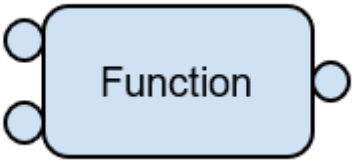
Property	Description
Value	The value obviously depends on the type. For a vector or quaternion the value will be an array of floats.
Timestamp	The system time in ms at which the value was changed.
Object name	The name of the object associated with the value (e.g. “Left upper leg”).
Property name	The name of the property that is represented by the value (e.g. “Position”).

Functions

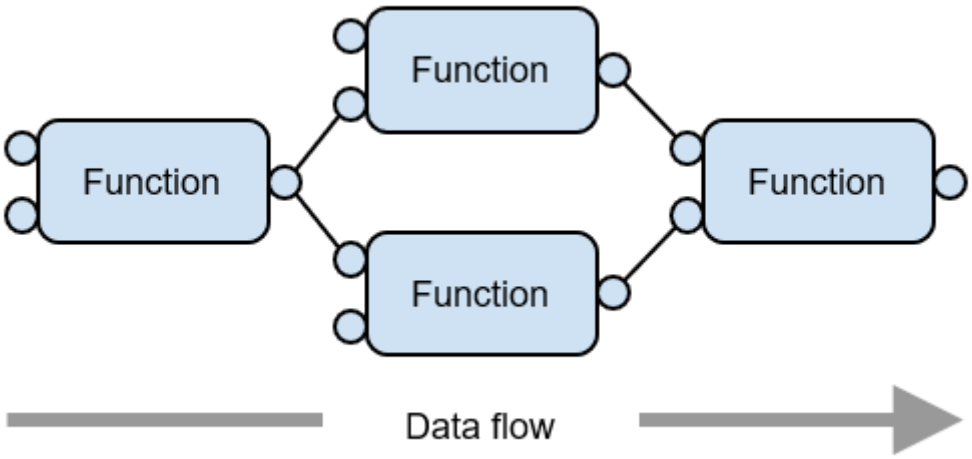
To construct an algorithm in KineXYZ, function nodes need to be interconnected. There are a number of different function types as described in this chapter.

Definition

A function node has a fixed number (one or more) of inputs and a single output.



In the data flow programming paradigm an algorithm can be constructed as a number of consecutive function nodes. The outputs of these nodes are connected to similar typed inputs of function nodes downstream. This is illustrated in the following figure:



Function node properties

To identify what the function does, a function node has a descriptive label. For functions with more than one input the it might be unclear what each input stands for. Therefore, a function node also has a description. The description can be viewed when the function node is selected and a properties window is opened.

Update

Every time one of the inputs changes, the function node needs to determine whether a new output is to be calculated. For most functions this is the case when all inputs have been updated. However, there are a few exceptions:

- In case constant values are used on some of the inputs. These are to be considered as updated whenever one of the other inputs changed.
- In case a gate function is implemented, the gated value needs to be forwarded when either the gate or the gated value changes.
- In case of an interpolation function.

The different function nodes are discussed in the following section.

Function node types

The following table lists the different function node types:

Type	Description
Fixed inputs	The type of all the inputs are defined.
Untyped input	The type of one of the input is undefined. As soon as this input is connected to a specific output, the type is defined.
Untyped correlated inputs	The type of several inputs is undefined, but as soon as one of those inputs is connected to a specific output, the type of these inputs is defined.
Gate function	A gate function is defined as having a boolean input and one or more other inputs. When the boolean input changes, the output is determined even if the input values did not change.
Untyped output	The type of the output depends on one or more of the inputs with undefined types.
Interpolation	Has one input denoting the time at which the interpolated value needs to be performed (tau) and one or two additional inputs that fill the array of data-points required to perform the interpolation. Updates every time when it is possible to determine a new interpolation value.

Required functions

The following classes of functions are required:

Type	Description
Arithmetic	Addition, subtraction, division, multiplication, absolute of float values.
Vector algebra	Creation, length, scalar multiplication, normalize, addition, distance, dot and cross product, negation, rotation.
Quaternion formulas	Creation, rotation angle, rotation vector, inverse, multiplication, normalize
Goniometric functions	Cosine, sine, tangent and their inverses.
Logical functions	Equal, greater than, =, AND, OR, switch and gate.
Naming	Set property and object name.
Timing	Timestamp, updated, timer, latency.
Buffering	Delay, moving average, integration.
Interpolation	One value and two value interpolation of arbitrary values.
Conversion	JSON object parser, Radians to degrees and vice versa, float to integer and vice versa, string to float/integer/boolean/vector/quaternion.

Output value properties

Every timestamped value has a number of properties. Obviously, the value is set by the implemented function, but for the other properties it needs to be determined how these are set. The trivial case is when the function only has one input. In that case the properties of the input values are copied to the output value.

Consider the case in which one or more of the inputs are meant for selecting the other input (e.g. gate function) or setting properties (e.g. naming function). In that case the properties of the ‘most meaningful’ input value are copied onto the output value.

However, for functions with more than one input it needs to be specified if and how the properties of the output value are set. This is discussed in the following two sections.

Output timestamping

The input values can change at different timestamps. However, due to buffering or order of execution, it is not assured that the input value that will change first is always the one with the oldest timestamp. Neither can it be assumed that the timestamp of the newest input value will be close to the current system time.

Therefore, the general rule will be that the timestamp of the output value will be that of the newest input value (i.e. the one with the largest timestamp).

Output naming

In a similar way as the timestamp, the object and property name of the output value are copied from the input value when the function has only one input or when there is only one 'meaningful' input value.

In case there are a number of input values, the object and property names are undefined.

An option could be to specify the name-properties as part of the properties of the function. However, doing that will hide the naming, i.e. the data flow diagram becomes less 'visual'.

Therefore, the approach is taken to have functions that explicitly set the name-properties.

Constant values

For all the timestamped value types a constant value can be added to the data flow diagram.

Definition

The visual programming that is used in the dataflow diagram has a lot of advantages. However, it does mean that everything is a function; even simple statements like $a = 2*b$ must be implemented by function nodes.

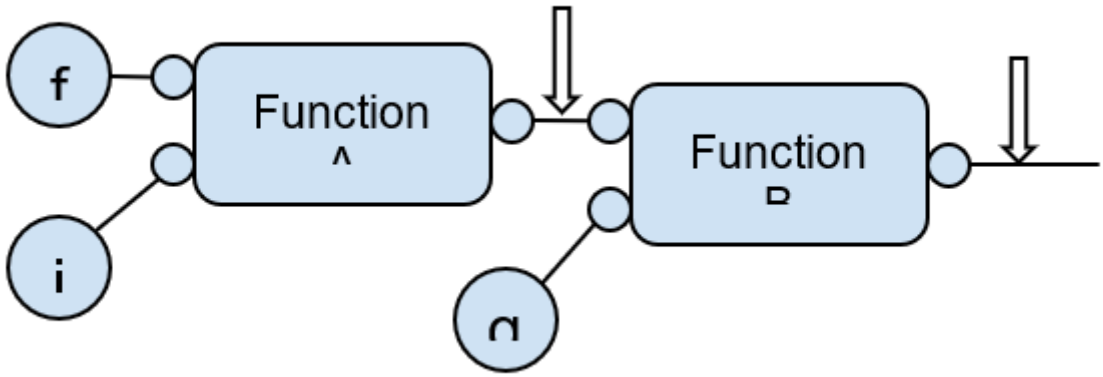
As can be deduced, this does mean that it must be possible to define constant values. As such, constant value nodes for each define value type are defined. These nodes can be added to the diagram and connected to function node inputs.

Properties

Obviously one property of the constant value nodes is the actual value. This value can then be placed as a label under the node. In some cases it can be instructive to actually be able to specify what the constant represents instead of the actual value. Therefore, optionally, the user can specify a label which is then to be placed under the node.

Timing

The following shows a simple illustrative example in which a function (Function A) takes two constants (float and integer) and calculates the output. This output value is calculated every time the user changes either one of the constant values. However, the output value must also be considered a constant. Otherwise Function B will never generate a new output value.



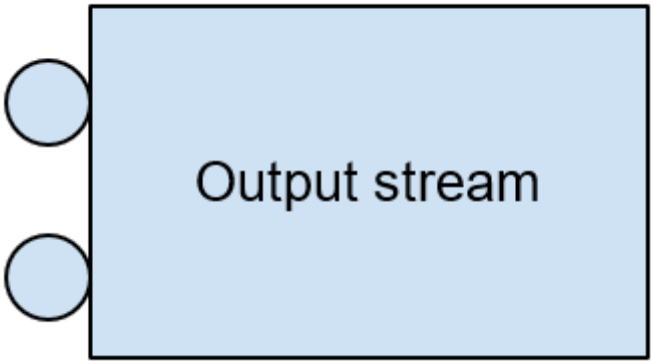
A similar consideration holds for gate functions for example.

Output streams

The objective of building an algorithm in the data flow diagram is to generate values that are to be used outside of the diagram. This is done by connecting values to output streams as described in this chapter.

Description

In most practical applications, values that are generated by the algorithms implemented in the data flow diagram are to be used outside of the diagram. To be able to do this, output stream nodes can be added to the diagram.



Such an output stream node can have one or more inputs. It uses these inputs to stream data

outside of the data flow diagram. As such it has no output into the diagram itself.
The values generated by the algorithms in the data flow diagram can be used in different ways. Therefore, different types of output streams are defined. These are discussed in the following sections.

Value label

The simplest way of using generated values is displaying their value. This can for example be useful when testing algorithms or simply to see whether the generated values are correct. Since in most practical situations the kinematic model generates the inputs for the algorithms, the scene is the desired location to display the value.
To achieve this, the Value label stream is defined. This output stream takes an arbitrary timestamped value and displays the string representation of the value as a label in the scene. Any number of value labels can be placed in the diagram. To distinguish the displayed values, in the properties of the value label stream the user can specify a descriptive name (i.e. label) which is placed in front of the value: <Value name>: <value>

Network stream

In a lot of practical situations the generated values are to be used in another app.
To make streaming to another apps generic, the local WiFi network can be used. This can be done by supplying the IP address and the port on which the data needs to be transmitted. In this way it does not matter whether the data is streamed to a remote app or a local app (using localhost).
Since in most cases a lot of small data packets need to be transmitted, the suitable connection type is UDP as it has the least overhead. This does mean that occasionally data packets will be lost. This seems acceptable, but must be taken into account when developing applications.
A lot of networking applications use JSON or XML as the format to transmit data. Given that there are a lot of libraries available to pack and unpack JSON messages, this is chosen as the default data format next to XML.
The above is implemented in the Network stream. It takes one or more arbitrary timestamped values and stores them in a JSON or XML string using the object name and property name of the values. There is no limit to the number of values that can be connected.

Set orientation

In certain kinematic models it is necessary to control the orientation of an object based on other kinematic values or values calculated by the algorithms. This can be done using a specific output stream that sets the orientation. This stream takes the name of the object and the new orientation as inputs. When either one of the inputs changes, the orientation of the object with the specified name is set in the kinematic model.
Obviously, to prevent an endless loop, setting the orientation of a certain object must be disabled when it would result in a new update of the algorithm.

In case a sensor is attached to the object, setting the orientation means that the orientation offset is set and as such an implicit alignment is performed.
Normally, this alignment is done explicitly by the user when the vertical or horizontal alignment is performed. However, having the ability to define the orientation offset in the data flow diagram as well, gives the possibility to implement different types of alignment procedures.

Set position

In certain kinematic models it is necessary to control the position of an object based on values generated in the algorithm. This can be done using a specific output stream. This “Set position” stream takes the name of the object and the position vector as inputs. When either one of the inputs changes, the position of the object with the name is set in the kinematic model.
Obviously, to prevent an endless loop, setting the position of a certain object must be disabled when it would result in a new update of the algorithm.
Note that when setting the position of a segment in a kinematic chain will reposition the whole kinematic chain.

Building a diagram

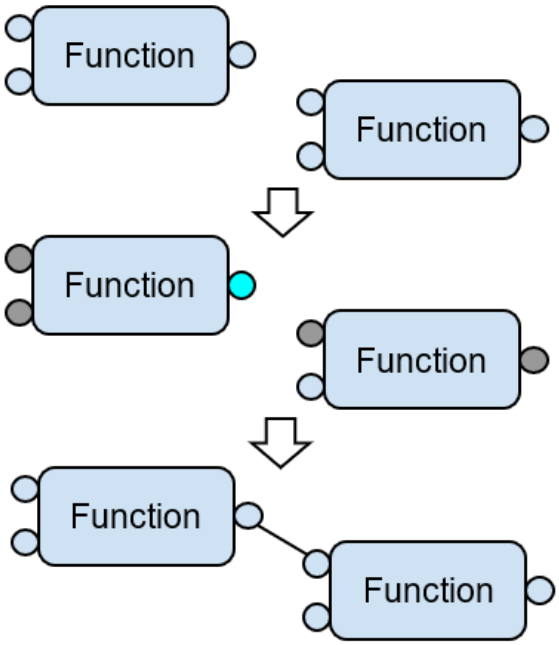
Building a diagram is mainly a visual task, placing and moving nodes, connecting inputs with outputs, etc.

Node placement

In data flow programming the position of nodes is an important factor to consider. Therefore the user has all the freedom to place and reposition nodes anywhere in the diagram. A simple intuitive drag-and-drop method supports this.

Connecting nodes

To add a connection between the output of a function and the input of another function, an output is selected and then an input or vise versa. To make this connection, the output must be the same type as the input.
To simplify the action and to prevent the user from trying to make connections between an output and input with different types, the selectable options are colored when an input or output is selected. This is illustrated in the figure on the right. Only inputs and outputs of the same value type remain available for selection.



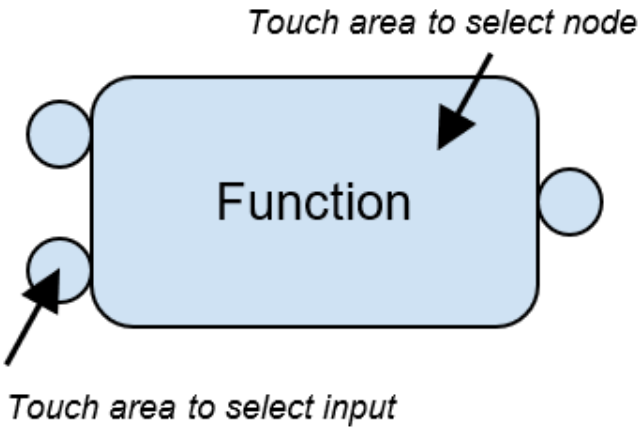
Note that in the figure, also inputs of the selected function are disabled to prevent the user from creating a feedback loop.

A kinematic value and a constant are outputs. This means that selecting such a node (see next section) also means that an output is selected and the described coloring takes place even if the reason for selection was for example to remove the node from the diagram.

Node selection

A node in the diagram can be selected by touching it. The selection is indicated by changing the coloring.

Except for kinematic value nodes and constant nodes, the selection of a node needs to distinguish between selecting one of its inputs or output and selecting the node itself. This means that the input and output indicators must be large enough to make it possible to select these.



To make it possible to perform actions on a number of nodes (deleting, copying), it must be

possible to select more than one node.

Node properties

Not all properties of a node are visible in the diagram. An example is the description of a function. Therefore, for a node that has additional properties that are not shown in the diagram, it must be possible for the user to select a single node and view the properties.
Except for a function node, these properties can be edited also.

Delete nodes

It must be possible to delete a selected node. When a node is deleted, all the connections are deleted as well.

Copying nodes

With the exception of kinematic value nodes, it must be possible for a user to copy a selected node. Especially for container nodes this is a useful function.
When multiple nodes are selected, all nodes are copied with the exception of the selected kinematic values.

Containers

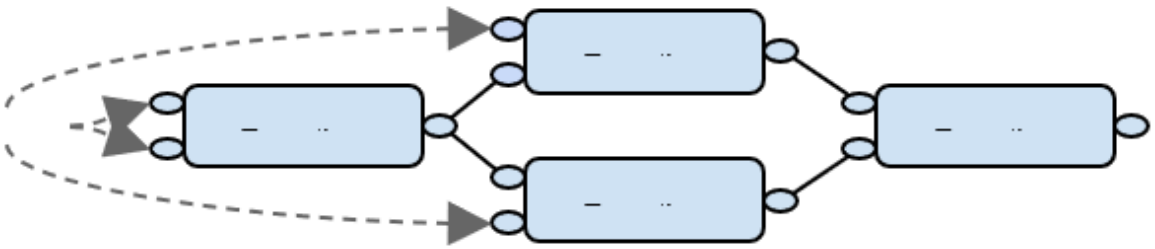
When building a data flow diagram, the size and complexity of the diagram can make it necessary to simplify the diagram to make it easier to understand. This can be done by combining several functions into a single container.

Reducing diagram complexity

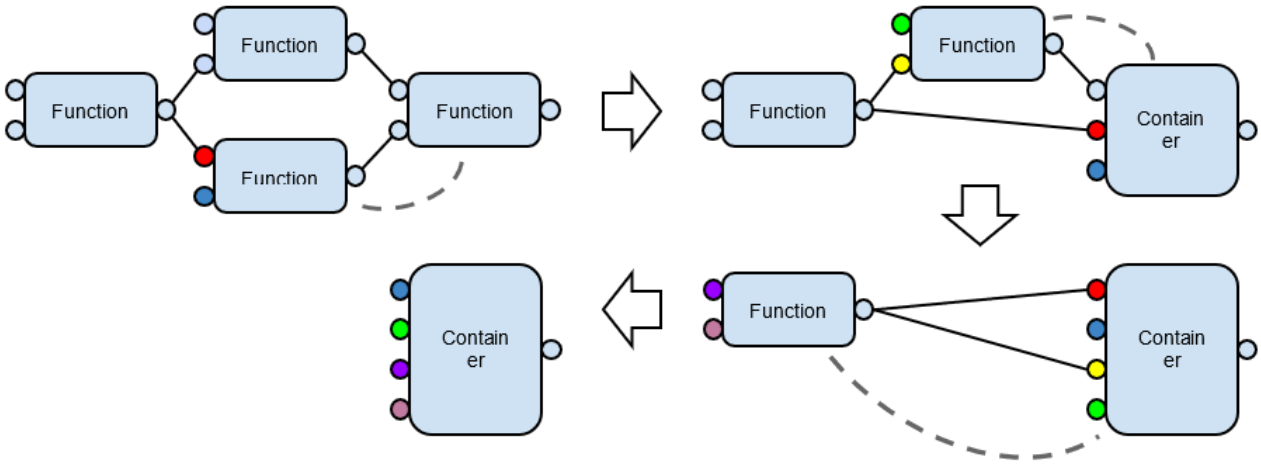
The actual process of building data flow diagrams is relatively straightforward. However, larger diagrams with a certain level of complexity can quickly become difficult to understand as overview is hard to obtain. This is a problem when changing or extending algorithms.
Another possible problem could be that the data flow diagram shows all the details about the implemented algorithms. In case the developer wants to share the diagram, this might be an unwanted situation.
Both problems can be solved by making it possible to hide specific details of the algorithm. This can be done by combining several function nodes into a single container node.
Combining several functions into a single container node also makes it much easier to copy and reuse algorithms.

Combining functions

As described in the previous chapter, building an algorithm in the data flow diagram is done by connecting outputs and constants with the inputs of downstream functions.
An illustrative example of an algorithm is given in the figure below:



In this example, all inputs that are not connected to the output of other function nodes can be considered of the algorithm.

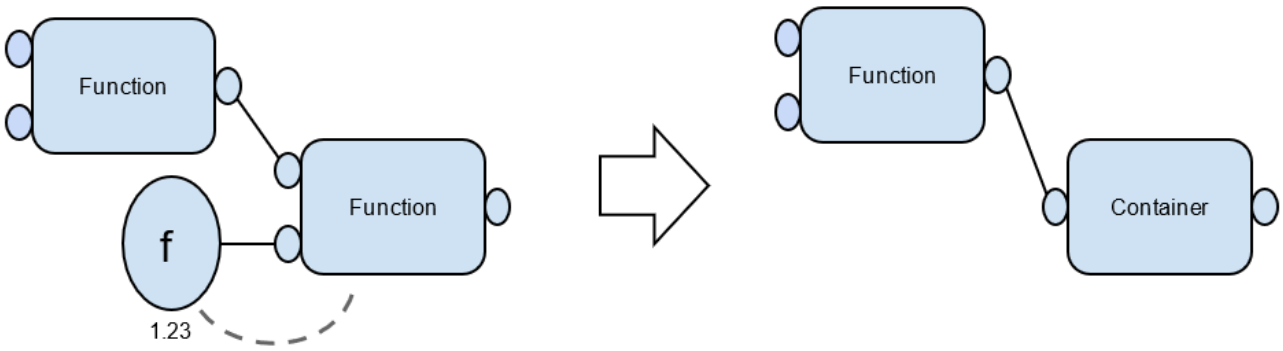


The function nodes of this algorithm can be combined into one container node. Such a container node is similar to a function node with the difference that it was defined by the user. The following figure illustrates how the function nodes of the algorithm can be combined into one container node:

Combining constants with functions

A kind of special case is the combination of a constant with its connected function node. This is illustrated in the figure below. As can be seen, the constant node is no longer visible in the

diagram and the input to which it was connected disappeared.



Container properties

When a container node is constructed, it must be possible to identify the implemented algorithm. Therefore, unlike a function node, the properties of a container node (i.e. name and description) can be edited.

Content sharing

A very important functionality for most user groups is the possibility to save and load content and share it online. This content comprises kinematic models and data flow diagrams. This chapter describes the generic content storing and sharing and the specific aspects for a kinematic model and data flow diagram.

Storing content

In most cases, a user will create several kinematic models and data flow diagrams. This means that functionality is required to store this content and used it later on.

Saving

When a kinematic model or data flow diagram is created, it must be possible for the user to save this on the device.

To make it easy to retrieve the file later on, the following is stored:

- *Snapshot*

A snapshot of the scene or diagram is taken and shown in the list, making it easier to search through a list of saved content files.

- *Date-time*

The date-time information is used to order the content files.

- *Name*

Adding a name to the content file makes it possible to distinguish the files more easily. Note that internally this name does not have to correspond to the actual filename as in most cases the actual files on the file-system are not explicitly exposed to the user or accessible anyway (e.g. as is the case on an iOS device).

- *Description*

By adding an optional description of the content can be instructive when sharing the content later on.

Loading

To make it possible to load a previously saved content file, all the content files stored on the device need to be listed. To make searching easier, this list complies to the following:

- *Snapshot*

The snapshot is displayed as a small image.

- *Name*

The name and date-time information is displayed.

- *Ordering*

Ordering can be by name and by date-time in both descending and ascending order.

- *Description*

It is possible to select a file and see the description.

- *Selecting*

A selected content file can be loaded and the user can indicate whether it needs to be added to the current content or replace it.

- *Deleting*

A selected content file can also be deleted, which will remove it from the file system.

Data flow diagram

In a data flow diagram the algorithms can use selected kinematic values of objects as input. When a data flow diagram is saved, the used kinematic values are not stored in the file and need to be reconnected.

Kinematic model

A kinematic model can be saved with a snapshot of a scene to facilitate the selection when loading a kinematic model.
All properties of the objects are stored in the kinematic model file. This also includes the selection of the kinematic values.

Content sharing

Users that work with KineXYZ can generate content and share that via email.

Generating content

Users that work with KineXYZ can generate content. This content can be a kinematic model or a data flow diagram. This can be shared via email. This offers the possibility for cooperation, creating open learning environment and enables people to offer their work commercially.
To share content, the user select the file in the list and indicates that this file needs to be emailed. A separate window will open belonging to the OS emailing functionality with a prepared concept email with the file already attached.

Receiving

Considering a situation in which a user receives an email to which an KineXYZ file is attached. When this email is received on the same device as on which the KineXYZ app is deployed, the file can be opened by the KineXYZ app.
When the device on which the KineXYZ app is deployed does not have the possibility to receive

the email, the user can place the KineXYZ file online using iCloud, after which it can be downloaded in the KineXYZ app. Alternatively, the file can be placed on a website after which it can be downloaded in the KineXYZ app as well.

Extending the functionality

The functionality of KineXYZ can be extended by using companion apps. This section briefly describes the concept of KineXYZ companion apps which can not only extends the functionality of KineXYZ, but could also be the developer's prototype app to test the interaction using the output of a dataflow diagram. This section describes the concept of companion applications in more detail.

Companion apps

This chapter describes the of companion applications in more detail.

Description

Not all functionality is required by all users and not all required functionality can be foreseen. To keep the KineXYZ app intuitive and simple, specific functionality is offloaded to so called KineXYZ companion apps. When these apps are deployed on the same device they work together with the KineXYZ app.

Generic functionality

The companion app uses the output stream of the KineXYZ app. The output stream is configured to stream to the localhost.

The generic functionality of a companion app is as follows.

1. The companion application opens a port to receive UDP messages.
2. The application can be running on the same or separate system.

To make the companion app receive the stream, the IP address and port number are specified. Specific companion applications expect the data to contain specific information. Any incoming data that is not according to this expectation is to be ignored.

Several companion applications can be running in parallel, but not on the same port.