Ikaros: A framework for controlling robots with system-level brain models

Christian Balkenius®, Birger Johansson® and Trond A Tjøstheim

Abstract

Ikaros is an open framework for system-level brain modeling and real-time robot control. Version 2 of the system includes a range of computational components that implements various algorithms and methods ranging from models of neural circuits to control systems and hardware interfaces for robot. Ikaros supports the design and implementation of large-scale computation models using a flow programming paradigm. Version 2 includes a number of new features that support complex networks of hierarchically arranged components as well as a web-based interactive editor. More than 100 persons have contributed to the code base and over 100 scientific publications report on work that has used Ikaros for simulations or robot control.

Keywords

Robotics middleware, system-level brain modeling, robot control

Date received: 1 June 2019; accepted: 11 March 2020

Introduction

Ikaros is an open infrastructure for system-level brain modeling and robot control. The project started already in 2001 and has been running continuously since then. To date, more than 100 persons have contributed to the code base and over 100 scientific publications report on work that has used Ikaros for simulations or robot control. The system has recently been substantially updated, and this article describes the addition in latest version as well as some of the projects using the system.

The systems perspective has in recent years been gaining influence as a fruitful way of organizing and understanding scientific theories. This perspective implies an integrative and holistic view of science and emphasizes computer modeling in particular as a tool both for increasing human understanding, as well as a way of extending cognitive reach. Implicit also in the systems view is the notion of hierarchy and that component parts are required to cohere. This can be contrasted with the more traditional reductionist approach to science. Ikaros aims at supporting the development of biologically motivated computational models from a systems perspective.

Evidence from neuroscience supports the notion that the brain is organized to minimize energy expenditure in general, which tend to favor functional localization and usage of energy optimal neural codes. In practice, this translates into a central nervous system that favors modular processing stages, where long-distance communication is penalized. Hence, to adequately and realistically model brain regions, Ikaros is geared toward a modular and systemic modeling approach, that is, the modeling work flow...
implies going from a network of brain regions to a network of computational processing modules.

No matter what level of detail is targeted for neural modeling, whether it be the cellular level, the level of neuronal populations, or the level of functional systems, a modeler needs to determine the following: Firstly, what the components of the system are; secondly, how the components are connected, that is, what the structure of the system is; thirdly, what the function of each of the components is; and finally, what kind of information flows between the components and how that information is coded. Given this information, a graph can be constructed which represents the desired model. The Ikaros system is flexible enough to model such graphs at any level and also supports multilevel models by means of grouping of components. The framework has been used to model learning in classical conditioning, reflex pathways, dynamic memory systems, developmental mechanisms, and learning in the perceptual system.

Several frameworks for simulating neural and cognitive processes now exist, focusing on different levels of detail, and using different approaches. At the cellular level, NEURON, Brian, and Nest allow simulation of a variety of neuron models. The Emergent framework allows modeling of cognitive modules using spiking neurons, while ACT-R uses a symbolic approach. Nengo is somewhat similar to Emergent, using populations of spiking neurons that can be configured to perform functions ascribed to particular brain areas like the basal ganglia. A more abstract approach is used by Cedar, which uses dynamic neural fields instead of populations of spiking neurons. Outside of computational neuroscience and cognitive science, machine learning frameworks such as TensorFlow have been applied to do practical image classification and object recognition. There are also robot frameworks that do not aim to implement biologically motivated models such as ROS. Although all these systems have their merit, none of them address real-time control using system-models in the way that Ikaros does.

Ikaros supports modeling of a broad category of biological, neural, and brain systems, but it was primarily designed to facilitate real-time cognitive control of robot systems. There is presently available a wide selection of frameworks for off-line modeling tools focusing on every component and how that information is coded. Given this information, a graph can be constructed which represents the desired model. The Ikaros system is flexible enough to model such graphs at any level and also supports multilevel models by means of grouping of components. The framework has been used to model learning in classical conditioning, reflex pathways, dynamic memory systems, developmental mechanisms, and learning in the perceptual system.

Several frameworks for simulating neural and cognitive processes now exist, focusing on different levels of detail, and using different approaches. At the cellular level, NEURON, Brian, and Nest allow simulation of a variety of neuron models. The Emergent framework allows modeling of cognitive modules using spiking neurons, while ACT-R uses a symbolic approach. Nengo is somewhat similar to Emergent, using populations of spiking neurons that can be configured to perform functions ascribed to particular brain areas like the basal ganglia. A more abstract approach is used by Cedar, which uses dynamic neural fields instead of populations of spiking neurons. Outside of computational neuroscience and cognitive science, machine learning frameworks such as TensorFlow have been applied to do practical image classification and object recognition. There are also robot frameworks that do not aim to implement biologically motivated models such as ROS. Although all these systems have their merit, none of them address real-time control using system-models in the way that Ikaros does.

Ikaros supports modeling of a broad category of biological, neural, and brain systems, but it was primarily designed to facilitate real-time cognitive control of robot systems. There is presently available a wide selection of frameworks for off-line modeling tools focusing on everything from single cells, to neuronal populations on the one hand, to statistical models using deep networks on the other. Like these systems, Ikaros can be used to train neural networks and perform off-line modeling, but it is somewhat unique in that it is built from the ground up to support low latency processing and enable realistic response times in various physical robot systems. Makers of particular robot systems like the Pepper system or iCub supply frameworks that enable model building, but they are naturally tied to the particular system in question. Ikaros currently supports controlling popular hardware like robots using Dynamixel servos, as well as the e-puck, but can easily be extended to support any system of sensors and actuators that have a C/C++ API.

Because it was initially thought that Ikaros should continue to operate over many years, it was decided to build a system that was as platform independent as possible. One consequence of this decision is that the data format used for most of the files in the system is XML. This also meant using a standard language, in this case C++, and a small set of external libraries.

Although Ikaros primarily targeted Linux systems, the framework was developed in parallel for Linux, Mac OS, and Windows. This turned out a good idea because it made any platform-specific solutions harder to use and as a consequence, Ikaros does not depend on any peculiarities in either operating system. The most important aspect of this is that Ikaros does not need to be changed every time the OS is updated.

Ikaros was initially intended for all kinds of computer hardware, ranging from microcontrollers to large clusters. Since the first version, the scene has changed substantially and today, even the smallest systems are likely to contain a complete Linux kernel. Some of the initial design constraints, such as not depending on any external libraries, is no longer valid and this has made several solutions much simpler in the present version. This also meant dropping Windows support since that version was not used to the same extent as the others.

The Ikaros framework has been designed to support a wide range of computational systems, depending on the requirements of the model in question. Hence, Ikaros can run well on small form factor computers like the Raspberry PI that are used primarily for control of robot platforms. But due to its highly threaded design, Ikaros is also well suited to take advantage of the massive parallelism of supercomputers or of recently available hardware like multi-GPU servers for off-line simulation and deep neural network training.

Version 2 of Ikaros builds on the previous versions but is substantially updated with many new features, including an interactive web-based editor and a more efficient XML-based language for the specification of large hierarchical network models. The rest of this article describes the different components of the framework and their current features.

System overview

Ikaros uses a dataflow programming paradigm. This approach can be traced back to work by Sutherland in the 1960s and 1970s but has recently seen a surge in interest as it may be well suited for future highly parallel processors. The paradigm implies the formation of a directed computational graph, where elements can function as sources or sinks but can also transform or process data. Hence, dataflow programming intersects with the functional programming paradigm in that it...
has a focus on data and transformations of data, rather than, for example, objects, data structures, or algorithms. This emphasis has made flow programming particularly popular in applications like signal processing, audio processing, and image manipulation. The implicit graph-like structure of flow programming lends itself also naturally to visualization, and most visual programming languages employ flow paradigms under the hood.

With the advent of multicore processors and specialty hardware like graphical processing units (GPUs), flow programming has enjoyed something of a renaissance. Nodes in the computational graph are usually independent enough that they can map to individual processing units or can execute concurrently. When data are shaped as tensors, the graph can be optimized to run efficiently on GPUs, yielding significant time savings.

Atomistic modules are meant to act as nodes in a network, and such a network is specified in a group file. Coarsely, a group file consists of a minimum of one module, but usually consists of a number of modules, parameter information for each module, and connections between the modules.

A single module input can have an arbitrary number of connections. Under the hood, this means that each connection’s array is concatenated, making up the input’s total associated array. In cases where only a subset of a source module’s output is to be used for a particular target module, the connection between them can be specified with a source and target offset value, along with the size of the subset.

In the brain, timing of spike trains is of the utmost importance, particularly when modeling processes having to do with learning reward value of actions. Ikaros supports specification of timing by means of connection delays and delay ranges. In its simplest form, this means that a change in the output is kept in a buffer for the length of the delay, so that it takes the specified number of time steps, or ticks, to reach the target module. By using delay ranges, the signal can be delayed by more than a single value of time steps. For example, if the range is set from 1 to 3, the signal will be delayed with 1, 2, and 3 time steps.

Groups can also be used to encapsulate a network of modules so they perform a specific function. In this case, both inputs and outputs can be specified for the group, and the group will appear and act as a module to an external network that embeds it. Depending on the topology of the group network, the group will have an intrinsic signal delay from input to output that must be taken into account when embedding it in larger network.

Unlike earlier versions, Ikaros 2 support a dot notation for accessing part of the system. For example, an address like

```
target="A.B.C.INPUT"
```

would target the input of the module C within B, which in turn is within A. Outputs and parameter of C can also be addressed in this way. The module C can also address its

In short, module classes are specified in terms of inputs, outputs, and class parameters. Both inputs and outputs are restrained to be tensors that are maximally of dimension 2. This means $n \times m$ matrices, where $n$ and $m$ are arbitrary, but constrained by system resources. Input and output sizes can be set explicitly or can be inferred at run time from connections. Inputs can also be specified to be optional but are set as required by default. This means that the model will not run if a required input has no connection. Parameters can be of type float, integer, boolean, or an enumerated list. A module can also have a default graphical user interface, the particulars of which will be described later in the section “WebUI.”

Figure 1. Example of an ikg-file in XML format. Here a module called Trigger is defined using the class MotionSequencer. The value of a parameter called speed is set. The module sends its output to the input INPUT of another module called RobotGripper of class Dynamixel. One of the connections has an explicit delay set to 2 time steps. A view for the WebUI is defined that consists of a single bar graph that shows the output ACTIVITY of the module Trigger.

Systems are described in an XML-based language (Figure 1). Models in Ikaros are built using two primary file types: class files (with extension ikc) that specify atomic modules and group files (with extension ikg) that specify networks and hierarchies of these modules.

In the brain, timing of spike trains is of the utmost importance, particularly when modeling processes having to do with learning reward value of actions. Ikaros supports specification of timing by means of connection delays and delay ranges. In its simplest form, this means that a change in the output is kept in a buffer for the length of the delay, so that it takes the specified number of time steps, or ticks, to reach the target module. By using delay ranges, the signal can be delayed by more than a single value of time steps. For example, if the range is set from 1 to 3, the signal will be delayed with 1, 2, and 3 time steps.

Groups can also be used to encapsulate a network of modules so they perform a specific function. In this case, both inputs and outputs can be specified for the group, and the group will appear and act as a module to an external network that embeds it. Depending on the topology of the group network, the group will have an intrinsic signal delay from input to output that must be taken into account when embedding it in larger network.

Unlike earlier versions, Ikaros 2 support a dot notation for accessing part of the system. For example, an address like

```
target="A.B.C.INPUT"
```

would target the input of the module C within B, which in turn is within A. Outputs and parameter of C can also be addressed in this way. The module C can also address its

In short, module classes are specified in terms of inputs, outputs, and class parameters. Both inputs and outputs are restrained to be tensors that are maximally of dimension 2. This means $n \times m$ matrices, where $n$ and $m$ are arbitrary, but constrained by system resources. Input and output sizes can be set explicitly or can be inferred at run time from connections. Inputs can also be specified to be optional but are set as required by default. This means that the model will not run if a required input has no connection. Parameters can be of type float, integer, boolean, or an enumerated list. A module can also have a default graphical user interface, the particulars of which will be described later in the section “WebUI.”

Atomistic modules are meant to act as nodes in a network, and such a network is specified in a group file. Coarsely, a group file consists of a minimum of one module, but usually consists of a number of modules, parameter information for each module, and connections between the modules.

A single module input can have an arbitrary number of connections. Under the hood, this means that each connection’s array is concatenated, making up the input’s total associated array. In cases where only a subset of a source module’s output is to be used for a particular target module, the connection between them can be specified with a source and target offset value, along with the size of the subset.

In the brain, timing of spike trains is of the utmost importance, particularly when modeling processes having to do with learning reward value of actions. Ikaros supports specification of timing by means of connection delays and delay ranges. In its simplest form, this means that a change in the output is kept in a buffer for the length of the delay, so that it takes the specified number of time steps, or ticks, to reach the target module. By using delay ranges, the signal can be delayed by more than a single value of time steps. For example, if the range is set from 1 to 3, the signal will be delayed with 1, 2, and 3 time steps.

Groups can also be used to encapsulate a network of modules so they perform a specific function. In this case, both inputs and outputs can be specified for the group, and the group will appear and act as a module to an external network that embeds it. Depending on the topology of the group network, the group will have an intrinsic signal delay from input to output that must be taken into account when embedding it in larger network.

Unlike earlier versions, Ikaros 2 support a dot notation for accessing part of the system. For example, an address like

```
target="A.B.C.INPUT"
```

would target the input of the module C within B, which in turn is within A. Outputs and parameter of C can also be addressed in this way. The module C can also address its
parent using the address $B$. This allows modules to easily address other modules in its near context. At the same time, it makes it possible to target modules deeply nested within the graph with a direct connection.

As in earlier versions, an inside module inherits parameters from its enclosing group, that is, the module $C$ is able to access a parameter even if it is only declared in $A$ or $B$. In Figure 1, that means that the parameter $x$ is available in both of the modules within the group. In addition, version 2 introduces variables. These can be used to efficiently distribute parameter values across module instances or specify connection targets, for instance. Variables can be used in any assignment and will access a parameter set in any enclosing group. For example

$$a = "@b"$$

will set the parameter $a$ to the value of $b$. In the example in Figure 1, this feature is used to set the parameter $torque$ of the module $RoboGripper$ to the value 0.1. Similarly, using variables in connections makes networks more flexible since targets can be set at a higher level in the hierarchy. For example

$$source = "@aOUTPUT"$$

sets the module from which the output is taken using the value of parameter $a$.

These new features are very helpful in larger models. Currently, the largest models run with Ikaros consists of several hundred modules and several thousand connections. Only the available memory puts limits on the sizes of models.

**The kernel**

The kernel is the central component of the Ikaros framework. It is responsible for the initial allocation of resources for before execution and the scheduling during execution.

**Start-up**

At start-up, the kernel first performs a topological search on the network graph based on connections with no delays. The goal is to find an execution order that will make sure that a module is not executed before its input data are available. The sorting algorithm also looks for cycles that prevent a model from running. Note that cycles are allowed but require that there is a delay in the connections between the modules.

After the topological sort, each module is assigned a thread. Modules that have zero-delay connections between them are assigned to the same thread group while other modules run in their own thread. This allows such modules to share memory which makes the copying of data from output to input unnecessary.

In the second step, the kernel goes through all connections and calculates the amount of memory needed to hold the delayed signal between the modules. Since many delays can be specified for a single connection, the kernel sometimes needs to allocate memory for outputs several time steps backward in time.

Subsequently, the kernel allocates memory for all inputs and outputs for the modules. It also sets up memory sharing between modules where possible.

The final step is the initialization of the modules. An initialization function of each module is called. The role of this function is to bind inputs, outputs, and parameters to local variables in the module. It can also do additional initialization such as allocate internal memory or connect to external hardware.

The start-up time in Ikaros 2 is substantially decreased compared to earlier versions. For a typical use case with a robot controlled by 320 modules and 615 connections, start-up is 500 ms which is approximately 50 times faster that with the previous version.

**Running**

Once the start-up is completed, the kernel will start the run loop. The loop is divided into four steps: First, the tick function of each module gets called to allow the modules to update their states. Second, data are propagated between the modules from outputs to inputs. Third, data requested by the user interface are copied from the modules. Fourth, the kernel waits for the next tick to start. This timing is done at millisecond resolution.

Conceptually, all modules run synchronously and the order of execution of each module is guaranteed. This makes it possible to use delays in a predictable way to synchronize data from different parts of the system.

In real-time mode, which is the default, a time base is set at start-up that sets the frequency of module execution. This base is also used for connections delays. For a time base $t$ (ms) and a delay $d$, the actual delay in a connection will be $t - d$ (ms).

The kernel can also run in step mode controlled by the user interface. In this case, each tick is executed remotely from the user interface. This mode allows every time step to be visible in the user interface but precludes real-time processing. This mode is useful when real-time execution is not desired or for debugging.

**Timing, lags and profiling**

When running in real-time mode, two possible errors can occur. The first is when the three steps described above takes longer than the allotted time. This means that the system has not been able to adhere to the required real-time constraints. As a consequence, the execution will start to lag. The kernel will try to compensate for this by waiting a shorter time, or not at all, after the next iteration if possible. This will in general put the execution on track again if it is caused by an unexpected operating system event.
However, if the execution keeps lagging, it means that there are not sufficient computational resources to keep up with the desired execution frequency or that some module is slowed down or blocked, for example, by external communication. In this case, the model or individual modules have to be changed. Alternatively, a slower execution frequency needs to be chosen.

Another possible error occurs if the communication with the user interface does not finish before it is time to the next iteration. In this case, the processing of user data is interrupted and an empty package is sent to the user interface. This keeps the interface responsive even though the displays are not updated. This priority is set so that the user interface should never be able to slow down execution.

To help the user tune the system, the kernel continuously measures potential lag as well as the time requirements of all modules. It also measures the delays and latency of the communication with the user interface. Furthermore, the processor utilization and the idle time between ticks are also calculated and shown in the user interface (see below).

**Modules**

Native modules are implemented in C++. In the minimal case, each module must implement only two functions. The init-function is run during start-up and must perform all initialization needed by the module. In the simplest case, the init-function will obtain handles for inputs and outputs of the module as well as bind parameters to local variables. The second required function is called ‘tick’ and will be called repeatedly during execution. It should perform whatever function the module has and compute its outputs based on the inputs and parameters.

A new function in Ikaros 2 is the command function that allows the user interface to directly execute a command in a module. This function takes a string and two values as its input. These parameters can be set in the user interface to select a particular function and to pass arguments, for example, the location where the user clicked in the user interface. This new functionality greatly simplifies the design of more advanced user interfaces with many controls, and although limited is sufficient for our current use cases.

In addition to the C++ code, each module is defined in an ikc-file that uses XML format to declare the inputs, outputs, and parameters of a module. This file is used by Ikaros during initialization to know what functionality is implemented by the module.

New modules can also be defined by combining already existing modules. In this case, the groups in ikg files are used for encapsulation and additional input and output elements are used to export the connections of the module.

**Standard modules**

The Ikaros framework comes with an extensive selection of modules for building computational pathways and models. In this section, we present the major categories of modules and give some illustrative examples of modules.

**Neural network modules**

A perceptron module is available, as well as a multiresolution network module, an autoassociator, and a module for backpropagation. The multiresolution module can be used to transform any complex input, like images, into a representation of discrete levels of detail.

**Brain model modules**

To this category belongs both module complexes, like a model of pupil control, a model for many parts of the brain, including the amygdala, parts of cortex, hippocampus, and thalamus, but also a module for modeling single nuclei, as well as populations of spiking neurons.

**Coding modules**

Modules that implement various forms of signal coding and decoding are available, including coarse coding, interval coding, population coding, and spectral timing. Included here is also a module for a tapped delay line.

**Control modules**

Among the control modules are two modules that implement classic control algorithms. The first is a Kalman filter and the second is a PID controller.

**Environment modules**

Models of systems that interact with an environment, like robots, can also interact with simulated models of environments. For this purpose, Ikaros contains the following modules. The GridWorld module offers a simple 2D environment of discrete blocks where a model might move or encounter an obstacle. The MazeGenerator module similarly generates a 2D maze which can be used to simulate path finding, spatial memory, prospection, and similar algorithms. The PlanarArm module is also a 2D environment but includes an arm effector and a target for the arm to reach. The GLRobotSimulator allows specification of a simple 3D environment containing a robot with a single arm effector and blocks that the arm can pick up and move.

**IO modules**

For getting information into and out of the system, there are modules that can read and write CSV files, image files in JPEG, PNG, and Raw format. Video data in the various
formats can be read, and 3D data can be captured from the
Kinect sensor. There is also an interface module for the Eye
Tribe eye-tracker. Network communication is supported in
the form of IPC client and server, and a YARP port. There
is also support for playing back sound files.

**Learning modules**

Aside from neural network learning, there are also modules
that implement machine learning algorithms like the Delta
rule, K-nearest neighbor, a linear associator, a local
approximator, and Q learning for reinforcement
learning.

**Robot modules**

In this category are modules that interface with various
kinds of robot systems, motors, and other effector hard-
ware. This includes Dynamixel motors, the e-puck system,
the FadeCandy system, a gaze controller, the
SSC-32 servo, and the Phidget system. Also included are
utility modules, like the MotionGuard module that protects
against potentially harmful, sudden movements, and a
MotionRecorder module that allows recording movement
by moving effectors.

**Utility modules**

This category contains the most modules, many of which
are basic mathematical operations like add, average, multi-
ply, max and min, sum, counter, and so on. There is also a
module which allows specifying mathematical expressions
in text, and an arbiter module which allows choice of an
element in a tensor according to rules such as winner takes
all, or max amplitude. The output of the arbiter module can
be specified to be the smoothed average of the inputs, and
switching between values may also be smoothed in time to
avoid abrupt changes. This is useful for interpolating
between different behavior in a robot.

**Vision modules**

The vision category contains modules for attentive selec-
tion, depth selection, a face detector, a color classifier and
matcher, and a background subtractor. Various image oper-
ations like change detection, curvature detection, edge
detection, and a selection of filters are available. Depth
processing modules like depth blob list, depth contour
trace, transformation, and segmentation are available to use
on point clouds, for example, from Microsoft Kinect
sensor.

**Simulation environments**

Ikaros has support for three types of environment for run-
ing simulations in 2D or 3D.

**GridWorld.** The grid world module simulates an agent on
a 2D plane. In discrete mode, the environment consists
of a grid of discrete states and actions allow the agent to
move between these states. In addition to the agent
itself, the environment can contain various objects such
as obstacles and walls or object of positive or negative
valence, for example, rewarding and punishing objects.
This environment is suitable for reinforcement learning
experiments.

The grid world simulation also has a mode where the
agent can move freely in two dimensions. Here, the system
simulates a mobile robot with differential steering. There is
still an underlying grid that can be used for navigation or to
explore the environment.

**Box2D.** To simulate dynamic environments in two dimen-
sions, Ikaros has a module that uses the open source library
Box2D to simulate the physics of simple scenes. Since
Box2D is developed for games, the physics is not com-
pletely accurate, but it is still useful as a test environment.

**3D.** Simulated robots can also be visualized in a 3D envi-
ronment. There is currently no physics simulations and this
environment is currently only used to show a 3D view of a
controlled or simulated robot. We discuss this functionality
further below in relation to the WebUI.

**Using external libraries**

Ikaros modules are designed to be self-contained comput-
ing units that do not necessarily depend on the Ikaros API
for their function. This means that a module can import and
use any kind of external library, as long as it can be found in
the path and linked. The easiest way is to use libraries
implemented in header files, where no additional effort is
required for the module to compile and run.

Using specialty hardware like GPUs or spiking proces-
sors have become common in recent years, to speed up
computation in the context of training or running neural
networks. Frameworks such as Nengo or Emergent provide
GPU back ends for this purpose. For Ikaros, we chose
to not use back ends like this, but rather leave it up to
module programmers to implement suitable computational
pathways for desired hardware. This has a cost in terms of
programmer time when new modules are to be implemen-
ted, but it means that it is possible to run models on hybrid
architectures, and utilize all hardware simultaneously. The
decision to not offer GPU or other hardware support intrin-
scically was also based on our current use cases which
mostly involves running models in real time on small
form-factor hardware like the Raspberry PI. In addition,
adding back-end support for GPU hardware would imply
a major redesign of the Ikaros system, which we found
difficult to justify.
**WebUI**

The user interface for Ikaros is run as a web application in the browser. This allows the user interface to either run on the same computer as Ikaros or on a remote computer. The latter is useful when Ikaros is controlling robots. Ikaros provides a way for specifying graphical user interfaces for both networks and for single modules. These interfaces can be composed of a number of plotting and interaction elements, the particulars of which will be specified below (see also Figure 2). Elements can be added in the class or group files, or can be built in the WebUI directly.

The WebUI has been completely redesigned from earlier versions. The most important addition is the interactive editor and an interface that can be completely redesigned by changing the style sheet for the interface (Figure 3). The views in the WebUI are built from a set of WebUI elements. These elements are written as JavaScript objects and can easily be extended with new types.

The bar graph element (Figure 2(e)), as the name indicates, can be used to show relative magnitudes between output elements and the plot element plots outputs as functions of time (Figure 2(d)).

The grid element (Figure 2(c) and (f)) is suitable for showing low resolution images or heat map like data. It consists of a grid of pixel elements which can be colored by means of color maps like grey scale or other color ranges. It can also be used to imitate control indicators (Figure 2(f)). Similarly, the image element can be used to show both image-like data from modules and can also load images from disk.

The marker elements maps input data to \(XY\) coordinates. The principal use of these elements are as overlays to images for showing various dynamic spatial properties. It can also be used for scatter plots (Figure 2(a)). This element is versatile since the markers can consist of simple geometrical elements but also of text and coordinate data. The element can also be used to show 2D point clouds, for example, from physical sensors.

The path element can draw arbitrary paths based on coordinate data in a matrix output (Figure 2(b)). There are options to set line and fill attributes as well as an option to draw arrows at the end of lines. This can be used, for example, to plot vectors.

The text elements allow flexible positioning of labels and longer texts in the UI. The table element (Figure 2(g)) is useful for showing explicit numerical values. Positive and negative values can be color coded.

Four UI elements afford interaction; these are the button, switch, drop down menu, and slider (Figure 2(h)). The drop down menu allows selecting an element in a list, while the slider can be used to set continuous or interval values depending on configuration. The controls are bound to variables in the modules using the Bind-function that is called by a module during initialization.

Figure 2. Gallery of some of the graphical elements of the WebUI. (a) Scatterplot, (b) trace, (c) grid, (d) plot, (e) bar graph, (f) grid with circular representation, (g) table, (h) button, drop down menu, and slider.

Most of the above UI elements can be drawn in a coordinate system or as a stand-alone plots. There are also option to stack several graphs in subsequent rows. The colors and other properties of the lines can also be set.
There is also an element for 3D rendering based on Three.js. This element is used to visualize robots during simulations.

**Robots using Ikaros**

During the development of the Ikaros framework, it has been used to control a large number of different robots. Figure 4 gives an overview of some of these robots. The different robot projects using Ikaros has worked with both commercially available platforms and custom built robots.

**Boe-Bot**

One of the first projects in which Ikaros was used to do modeling, included an experiment considering the effect of...
anticipation on robot navigation.\textsuperscript{49} By equipping small two-wheeled robots with ability to plan and anticipate, as well as behaving randomly or reactively to each other, the authors showed that anticipation and planning is not necessarily dominant, when the goal is for the robots to trade places with each others. The experiments used the Boe-Bot (Parallax, Inc.). The robots were controlled remotely from a computer cluster running Ikaros over wireless connection. Since there were no sensing on the robots, an overhead camera was used to track the positions of the robots using color makers that were identified and tracked using the processing modules in Ikaros (ColorTransform, ColorClassifier and SpatialClustering).

**e-puck**

Later experiment used the e-puck instead since it offers a better wireless interface. These were used in a number of experiments on anticipation and attention in groups of robots.\textsuperscript{50} The models used in this experiments were later extended to also support color marker tracking and the use of an overhead camera.\textsuperscript{51}

**Laser Turret**

Ikaros has also been used to control robots based on standard radio control servos through the SCC-32 Servo Controller (Lynxmotion, Inc.). The real-time features of Ikaros in combination with the position interpolation of the servo controller allowed a laser turret to smoothly track moving objects such as a toy car on a track even at high speed. The LinearAssociator module in Ikaros was central to the setup and learned a model that was used to predict the motion of the car half a second into the future. This system was developed as part of the EU-financed project MindRaces.

**Mobile Arm**

A similar control strategy was used in a mobile arm robot that again used radio control servos together with the SCC-32 controller to pursue moving objects using a toy fishing rod. The robot learned the temporal dynamics of the target and the inverse kinematics of the arm at the same time using Ikaros’ modules for population coding together with a linear associator.

**Goaliebot**

A robot with two arms controlled by two Dynamixel AX12 was built to test models of anticipatory motor control. The robot learned to predict the movements of an approaching ball and the robot selected the appropriate action to stop the ball from passing.

**Nao**

Ikaros has also been interfaced with the NAO robot (Aldebaran). Ikaros was used with Nao to study the learning of inverse kinematics.

**LUCS Robotics Kit**

To allow students to interact with Ikaros, the LUcS Robotics Kit was developed. It consists of a small robot arm with three degrees of freedom together with a Kinect sensor and a set of cubes with fiducial markers. The robot arm uses three AX-12 servos (Robotis) that are connected to a computer over USB. These robot system have been used in a large number of student projects involving Ikaros and the actual robot arm has been modified in many ways to include a head or a hook or even the cosmetics to make it look like a rooster.

**BioloI**

Ikaros was used to produce walking and climbing movement in the small humanoid robot BioloI Premium (Robotis). The real-time features of Ikaros were used to produce precise movements controlled at 50 Hz.

**LUcS Robotics Hands I–III**

A number of robot hands were used to study haptic perception.\textsuperscript{50,52,53} The robot hands use Ikaros to control the movements of the fingers and to analyze sensor data using self-organizing maps. The robots were able to sense hardness, texture, and shape of grasped objects. The models aimed to represent human neurophysiology but limited the shapes that the system could categorize to spheres and cubes. Johnsson et al.\textsuperscript{52} built on these experiments and used a robotic arm system with 12 degrees of freedom and proprioceptive sensors into self-organizing maps (SOM) to map up neural representations of object shapes.

**Builder Robot**

The Builder Robot was constructed as part of the project Goal Leaders and used Ikaros to control its Mecanum wheels as well as two movable cameras and an arm with a gripper. The robot used an on-board Mac Mini to run Ikaros that interfaced with a number of Dynamixel servos and a Phidgets interface to proximity sensors and IMU (Phidget Inc.).

**Epi Head**

The Epi Head robot consists of Dynamixel motors to allow a head to flex, extend, and rotate sideways. The head also has eyes with controllable LED irises and pupils that can dilate and contract. This simple setup allows experiments to be conducted that involves, for example, social reactions to
pupil dynamics and head movements. But it allows also

testing of models of brain systems that control such
dynamics and of systems that are not necessarily depen-
don manipulators. Examples might be memory sys-

tems or models of visual attention. The robot was used
with Ikaros to study emotional displays and interaction
with children in a tutoring situation.

**Epi Full**

The most advanced robot to be controlled by Ikaros is the
Epi robot which is an upper body humanoid with a torso
with two arms with five degrees of freedom each and a head
with two degrees of freedom. There is one additional
degree of freedom for each hand and one for each of the
eye that can move sideways. In addition, the irises of the
eyes can be animated. Much of the current work on Ikaros
target this robot. The goal is to design a complete control
system for a humanoid robot based on human neurophys-
ology. Current work focuses on low-level reflex systems,
in particular eye reflexes and startle reflexes.

**Discussion**

Ikaros 2 constitutes a large step forward compared to earlier
versions. The new system includes all the functionality
needed to build very large system-models of the nervous
system and use them to control robots in real time.\textsuperscript{10,56–58}
The new functionality of the XML language used to specify
modules and connections has become more powerful by
including variables and dot notation for direct access to
modules deep down in a hierarchy. The user interface has
been completely rewritten to support interactive editing of
networks and graphs. Taken together, these additions make
Ikaros a much more efficient tool. There are also a large
number of small changes that enhances performance, like
the partially rewritten kernel. The new version incorporates
the experiences from using Ikaros to control robots for
more than 18 years.

We expect there to only be minor changes to the Ikaros
kernel and user interface in the future although many new
modules will be written to implement various aspects of the
nervous system. Nevertheless, a few additional functions
are planned. The thread management will be fine tuned to
allow more important threads controlling robot hardware
higher priority. Tests are ongoing with this functionality
that increases the robustness of the communication with
servos. Furthermore, we will allow modules to run asyn-
chronously in the future. This allows the inclusion of mod-
ules that cannot guarantee their maximum execution time.
This will be useful when modules use external libraries or
algorithms whose execution time is unpredictable.

A final addition will be the inclusion of higher dimen-
sional matrices for inputs and outputs. This functionality is
mainly ready but not included in Ikaros yet. Multidimen-
sional arrays will simplify some operations that now
requires several outputs, such as color images or layers of
neural networks.

**Authors’ note**

The source code for Ikaros can be downloaded through the project
website: http://www.ikaros-project.org.

**Declaration of conflicting interests**

The author(s) declared no potential conflicts of interest with
respect to the research, authorship, and/or publication of this
article.

**Funding**

The author(s) disclosed receipt of the following financial support
for the research, authorship, and/or publication of this article: This
work was funded in part by eSSENCE.

**ORCID iDs**

Christian Balkenius
https://orcid.org/0000-0002-1478-6329

Birger Johansson
https://orcid.org/0000-0002-9834-4279

**References**

1. Mobus GE and Kalton MC. Principles of systems science.
Berlin: Springer, 2015.

2. Hasenstaub A, Otte S, Callaway E, et al. Metabolic cost as a
unifying principle governing neuronal biophysics. *Proc Natl
Acad Sci* 2010; 107(27): 12329–12334.

3. Markman AB and Otto AR. Cognitive systems optimize
energy rather than information. *Behav Brain Sci* 2011;
34(4): 207–207.

4. Felleman DJ and Van DE. Distributed hierarchical processing
in the primate cerebral cortex. *Cereb Cortex (NY 1991)*
1991; 1(1): 1–47.

5. Balkenius C, Morén J, and Johansson B. System-level cogni-
tive modeling with Ikaros. *Lund Univ Cognit Stud* 2007;
133: 1–11.

6. Morén J and Balkenius C. Emotional learning: a computa-
tional model of the amygdala. *Cyber Syst* 2001;
32(6): 611–636.

7. Johansson B and Balkenius C. A computational model of
pupil dilation. *Connect Sci* 2016; 30: 1–15.

8. Balkenius C, Tjostheim TA, Johansson B, et al. From focused
thought to reveries: a memory system for a conscious robot.
*Front Robot AI* 2018; 5: 29.

9. Balkenius C and Johansson B. Anticipatory models in gaze
control: a developmental model. *Cognit Proc* 2007;
8(3): 167–174.

10. Tjostheim TA and Balkenius C. Cumulative inhibition in
neural networks. *Cognit Proc* 2019; 20(1): 87–102.

11. Hines M, Davison AP, and Muller E. Neuron and python.
*Front Neuroneinf* 2009; 3: 1.

12. Goodman DF and Brette R. The brain simulator. *Front Neu-
rosci* 2009; 3: 26.

13. Eppler JM, Helias M, Muller E, et al. PyNEST: a convenient
interface to the NEST simulator. *Front Neuroneinf* 2009;
2: 12.
14. Aisa B, Mingus B, and O’Reilly R. The emergent neural modeling system. *Neural Netw* 2008; 21(8): 1146–1152.
15. Anderson JR, Bothell D, Byrne MD, et al. An integrated theory of the mind. *Psychol Review* 2004; 111(4): 1036.
16. Bekolay T, Bergstra J, Hunsberger E, et al. Nengo: a python tool for building large-scale functional brain models. *Front Neuroinform* 2014; 7: 48.
17. Lomp O, Zibner SKU, Richter M, et al. A software framework for cognition, embodiment, dynamics, and autonomy in robotics: cedar. In: *International conference on artificial neural networks*, Sofia Bulgaria, 10–13 September 2013, pp. 475–482. Berlin: Springer.
18. Abadi M, Barham P, Chen J, et al. Tensorflow: a system for large-scale. In: *OSDI*, Savannah, Ga, USA, 2–4 November 2016, vol. 16. pp. 265–283.
19. Quigley M, Conley K, Gerkey B, et al. ROS: an open-source robot operating system. In: *ICRA workshop on open source software*, Kobe, Japan, 12–17 May 2009, vol. 3, p. 5.
20. Tanaka F, Ishikawa T, Takahashi F, et al. Pepper learns together with children: development of an educational application. In: *2015 IEEE-RAS 15th international conference on humanoid robots (humanoids)*, Seoul, 3–5 November 2015, pp. 270–275. New York: IEEE.
21. Sandini G, Metta G, and Vernon D. The iCub cognitive humanoid robot: An open-system research platform for enactive cognition. In: *50 years of artificial intelligence*. Berlin: Springer, 2007, pp. 358–369.
22. Mensink A. Characterization and modeling of a Dynamixel servo. Technical report, University of Twente, Enschede, 2008.
23. Mondada F, Barham P, Chen J, et al. The e-puck, a robot designed for education in engineering. In: *Proceedings of the 9th conference on autonomous robot systems and competitions*, Castelo Branco, Portugal, 7 May 2009, vol. 1, pp. 59–65.
24. Sutherland WR. The on-line graphical specification of computer procedures. PhD Thesis, Massachusetts Institute of Technology, Cambridge, MA, 1966.
25. Gilles K. The semantics of a simple language for parallel programming. *Inf Proc* 1974; 74: 471–475.
26. Dennis JB. First version of a data flow procedure language. In: *Programming symposium*. Berlin: Springer, 1974, pp. 362–376. Paris, Fance: Institut de Programmation, University of Paris.
27. Bergstra J, Breuleux O, Bastien F, et al. Theano: a CPU and GPU math expression compiler. In: *Proceedings of the Python for scientific computing conference (SciPy)*, Austin, TX, 28–30 June 2010, vol. 4.
28. Hari R, Levänen S, and Raji T. Timing of human cortical functions during cognition: role of MEG. *Trend Cogn Sci* 2000; 4(12): 455–462.
29. Hari R and Parkkonen L. The brain timewised: How timing shapes and supports brain function. *Philos Trans R Soc B Biol Sci* 2015; 370(1668): 20140170.
30. Balkenius C, Morén J, Johansson B, et al. Ikaros: building cognitive models for robots. *Adv Eng Inf* 2010; 24(1): 40–48.
31. Rosenblatt F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychol Rev* 1958; 65(6): 386.
32. Bourlard H and Kamp Y. Auto-association by multilayer perceptrons and singular value decomposition. *Biol Cybern* 1988; 59(4-5): 291–294.
33. Hecht-Nielsen R. Theory of the backpropagation neural network. In: *International Joint Conference on Neural Networks (IJCNN)*. Washington, DC, USA, 1989, pp. 593–605. New York: IEEE.
34. Balkenius C and Morén J. A computational model of emotional conditioning in the brain. In: *Proceedings of workshop on grounding emotions in adaptive systems*, Zurich, 21 August 1998. Neural information processing systems foundation, Inc.
35. Balkenius C and Morén J. A computational model of context processing. In: Meyer JA, Berthoz A, Floreano D, et al. (eds) *From Animals to Animats 6: Proceedings of the 6th international conference on the simulation of adaptive behaviour*. Cambridge, MA: The MIT Press.
36. Izhikevich EM. Simple model of spiking neurons. *IEEE Trans Neural Netw* 2003; 14(6): 1569–1572.
37. Sutton RS. Generalization in reinforcement learning: successful examples using sparse coarse coding. In: Mozer MC, Jordan MI, and Petsche T (eds) *Advances in neural information processing systems*, 1996, pp. 1038–1044.
38. Piper J. Efficient implementation of skeletonisation using interval coding. *Pattern Recogn Lett* 1985; 3(6): 389–397.
39. Georgopoulos AP, Schwartz AB, and Kettner RE. Neural population coding of movement direction. *Science* 1986; 233(4771): 1416–1419.
40. Harvey AC. Forecasting, structural time series models and the Kalman filter. Cambridge: Cambridge University Press, 1990.
41. Rivera DE, Morari M, and Skogestad S. Internal model control: PID controller design. *Ind Eng Chem Process Des Dev* 1986; 25(1): 252–265.
42. Widrow B and Hoff ME. Adaptive switching circuits. Technical report, Stanford Electronics Labs., Stanford University, CA, 1960.
43. Fukunaga K and Narendra PM. A branch and bound algorithm for computing K-nearest neighbors. *IEEE Trans Comput* 1975; 100(7): 750–753.
44. Anderson JA, Golden RM, and Murphy GL. Concepts in distributed systems. In: *Optical and hybrid computing*. International Society for Optics and Photonics, vol. 634, pp. 260–277.
45. Tsybakov AB. Robust reconstruction of functions by the local-approximation method. *Problemy Peredachi Informatsii* 1986; 22(2): 69–84.
46. Watkins CJ and Dayan P. Q-learning. *Mach Learn* 1992; 8(3–4): 279–292.
47. Greenberg S and Fitchett C. Phidgets: easy development of physical interfaces through physical widgets. In: *Proceedings of the 14th annual ACM symposium on user interface
software and technology, Orlando, Florida, November, 2001.
New York: ACM, pp. 209–218.
48. Bååth R and Johansson B. Making a map-making robot: using the Ikaros system to implement the occupancy grid algorithm. In: Roland B (ed) The Swedish AI Society Workshop May 20-21; 2010, Uppsala University. Linköping: Linköping University Electronic Press, pp. 27–34.
49. Johansson B and Balkenius C. An experimental study of anticipation in simple robot navigation. In: Martin BV, Olivier S, Giovanni P, and Gianluca B (eds) Workshop on anticipatory behavior in adaptive learning systems, Rome, Italy, 2006. Berlin: Springer, pp. 365–378.
50. Johnsson M and Balkenius C. Experiments with artificial haptic perception in a robotic hand. J Intell Fuzzy Syst 2006; 17(4): 377–385.
51. Johansson B and Balkenius C. A multi-robot system for anticipatory experiments. LUCS Minor 2007; 11: 1–21.
52. Johnsson M and Balkenius C. Experiments with proprioception in a self-organizing system for haptic perception. Proc TAROS 2007; 2007: 239–245.
53. Johnsson M and Balkenius C. Neural network models of haptic shape perception. Robot Auton Syst 2007; 55(9): 720–727.
54. Lindberg M, Sandberg H, Liljenberg M, et al. The expression of mental states in a humanoid robot. In: International conference on intelligent virtual agents, Stockholm, Sweden, 27–30 August 2019. Berlin: Springer, pp. 247–250.
55. Lindberg M, Månsson K, Johansson B, et al. Does a robot tutee increase children’s engagement in a learning-by-teaching situation? In: International conference on intelligent virtual agents, 27–30 August 2019, Stockholm, Sweden. Berlin: Springer, pp. 243–246.
56. Tjøstheim TA, Johansson B, and Balkenius C. A computational model of trust-, pupil-, and motivation dynamics. In: Proceedings of HAI 2019, Kyoto Japan, October, 2019. New York: ACM, pp. 179–185.
57. Balkenius C, Fawcett C, Falck-Ytter T, et al. Pupillary correlates of emotion and cognition: a computational model. In: 2019 9th international IEEE/EMBS conference on neural engineering (NER), San Francisco, USA, 20–23 March 2019. Piscataway: IEEE, pp. 903–907.
58. Balkenius C, Tjøstheim TA, and Johansson B. Arousal and awareness in a humanoid robot. In: A Chella, D Gamez, P Lincoln, et al. (eds) Towards conscious AI systems, CEUR workshop proceedings, Stanford, CA, 25–27 March 2019, vol. 2287.