iSpike 2.0 Manual

1. Introduction

iSpike is a C++ library that interfaces between spiking neural network simulators and the iCub robot. It uses a biologically-inspired approach to convert the robot's sensory information into spikes that are passed to the neural network simulator, and it decodes output spikes from the network into motor signals that are sent to control the robot. Applications of iSpike range from embodied models of the brain to the development of intelligent robots using biologically-inspired spiking neural networks. iSpike is an open source library that is available for free download under the terms of the GPL.

A full description of iSpike is available in the paper LazdinsFidjelandGamez11_iSpike.pdf, which is included in the doc folder of the release, which also includes source code documentation. This manual gives instructions on how to build iSpike and use it with your application.

2. Building iSpike

2.1 Windows

iSpike is available as a binary release for Windows consisting of a dynamic library, libiSpike.dll, include files and the third party dependencies. The Windows release also includes mingw32 builds of the Boost libraries regex, system and thread, which are needed to run the library. These libraries (in the 'thirdparty' folder) need to be on the system path for iSpike to work correctly. The easiest way to do this is to place them in the same directory as iSpike.

The source code of iSpike is included in the release or you can download the latest version from http://sourceforge.net/projects/ispike/develop. Note that the latest development version may not be as stable as the version included with the release.

To build iSpike from source you will need Boost and CMake. If you are building using MinGW you can link against the Boost libraries included in the thirdparty folder and download the header files for Boost 1.46. If you are using Visual Studio, download the latest libraries and header files for Boost. Before building, set BOOST_ROOT in src/CMakeLists.txt to the root directory of your Boost installation.

2.2 Linux

The source code of iSpike is included in the release or you can download the latest version from http://sourceforge.net/projects/ispike/develop. Note that the latest development version may not be as stable as the version included with the release.

iSpike depends on Boost and you will need CMake to configure the build. Before building, set BOOST_ROOT in src/CMakeLists.txt to the root directory of your Boost installation.

2.3 Mac OS X

Instructions for building iSpike on Mac OS X will be added shortly.

3. Using iSpike

iSpike consists of the following components:

- *Reader*. Extracts sensory data of a given type from a given location. The current release of iSpike has a FileAngleReader that reads an angle from a file, a FileVisualReader that reads a .ppm image from a file, a

YarpAngleReader that reads joint angles from YARP and a YarpVisualReader that reads images from YARP. Readers run as separate threads that buffer the latest data.

- *Writer*. Outputs data of a given type to a given destination. The current release has a FileAngleWriter that writes joint angles to a file as well as a YarpAngleWriter that sends motor commands to the iCub using YARP.

- *Input Channel*. Receives sensory data from a Reader of a given type and transforms it into a spike representation, which is passed to the neural simulator. The current release has a JointInputChannel that converts the angular data prepared by an AngleReader into spikes and a VisualInputChannel that converts visual data prepared by a VisualReader into spikes. The spike conversion process depends on simulated Izhikevich neurons, which are stepped in synchrony with the neural network simulator.

- *Output Channel*. Receives a spike pattern from the neural simulator, converts it into an appropriate format, and uses a Writer to deliver it to a predefined destination. The current release has a JointOutputChannel that converts neuron spike patterns into joint angle motor commands and uses an AngleWriter to deliver these to the iCub. The spike conversion process depends on simulated Izhikevich neurons, which are stepped in synchrony with the neural network simulator.

Each of these components is constructed using a Factory class - for example a Reader is constructed using a ReaderFactory, an InputChannel is constructed using an InputChannelFactory, and so on. To create YARP readers and writers, you will need to supply the IP address and port of a YARP server to the Reader or Writer factory.

The detailed configuration of each component is carried out by setting properties: each Reader, Writer and Channel has a getProperties and setProperties methods that enable you to access a map containing all of the properities for the component. Some of these properties are read only, which means that they can only be set at initialization time, whereas others can be changed at any time.

Pseudo code for using iSpike is as follows:

```
//Create an input factory
InputFactory inputFactory();

//Get a list of available channels
vector<Description> channelDescriptions = inputFactory->getAllChannels();

//Each channel is compatible with a reader or writer of a particular type
Description& channelDescription = channelDescriptions[1];//Get the description of Channel 1
ReaderFactory readerFactory("192.168.56.101" 10000);//Create a reader factory that can access
YARP
vector<Description> readerDescriptions =
readerFactory.getReadersOfType(channelDescription.getType());

//Configure the properties of the reader
Description& readerDescription = readerDescriptions[0];//Get the description of Reader 0
map<string, Property> propertyMap = readerFactory.getDefaultProperties(readerDescription);
...//Set properties appropriately

//Create the reader
Reader* reader = readerFactory.create(readerDescription, propertyMap);//Create reader

//Configure the properties of the Channel
map<string, Property> propertyMap =
inputChannelFactory.getDefaultProperties(channelDescription);
...//Set properties appropriately

//Create the channel
InputChannel* inputChannel = inputChannelFactory.create(channelDescription, propertyMap);

//Step input channel and get the spikes
inputChannel->step();
vector<int> inputNeuronIDs = inputChannel->getFiring();
//Creation of an OutputChannel proceeds in a similar way to the creation of an InputChannel
```

```
//Step output channel and set spikes
vector<int> outputNeuronIDs;
...//Add firing neuron IDs output from simulator to vector
outputChannel->setFiring(outputNeuronIDs);
outputChannel->step();
```

4. Questions & Problems

Support for iSpike is available at: ispike-user@lists.sourceforge.net.