# liburbiMatlab

Tutorial

version 0.1

## Introduction

This tutorial is a quick introduction to the use of the Matlab library to work with a URBI server. This tutorial expects the reader to be a bit familiar with the URBI syntax. For information about URBI and for updates on this library, please visit our site :

http ://www.urbiforge.com

For a complete description of the liburbiMatlab library, please see the html documentation located in the doc/html directory of the package. The examples given here are designed to work with an Aibo URBI server.

Please report bugs and request features on our website forum.

## 1 Installation

Unpack the liburbiMatlab package and add the liburbiMatlab/lib directory and its subdirectories to you matlab path.

You can then launch the demo if you have an aibo : go to the examples/ directory and launch :

```
>> urbiDemoSynchronous('myAiboName')
```

or

```
>> urbiDemoCallback('myAiboName')
```

## 2 Connecting to the server

When you work with a URBI server, the first thing you have to do is to connect to this server :

```
>> aibo = urbiConnect('zeus')

aibo =

     1
```

'zeus' is the name of the computer or robot running the URBI server. You have to store the return value as a connection id for future reference. To close this connection after your work, use :

```
>> urbiDisconnect(aibo)
```

If during the use of the Matlab URBI library, you fail to get the expected results, this is perhaps due to data blocked in the connection input buffer. To revert to a clear state , use :

```
>> urbiClearConnection(aibo)
```

# 3 Sending URBI commands

To send a URBI command string, use the function `urbiSend` :

```
>> urbiSend(aibo,'leds=1; wait(3s);leds=0;');
```

This function will do nothing with the messages returned by the server. If you want to see what the server replies, use the `urbiInteract` function instead. Pay attention to the fact that in this case, the messages are displayed and lost. They could not be processed further in matlab, so use only this function when you want a direct visual response of the server :

```
>> urbiInteract(aibo,'headPan;');
[09474076:notag] 0.422671
```

The `urbiSendFile` function allows to send a whole URBI script stored in a file :

```
>> urbiSendFile(aibo,'balltrackinghead.u')
```

You can process the messages coming from the URBI server and get them in a matlab-usable form using the `urbiGet` function :

```
>> [data,timeStamp, tag, type] = urbiGet(aibo)
```

The `data` variable contains the values taken from the message. Its type depends on the message type. For example, if type = numeric, data is the corresponding numerical value (see the html doc of urbiGet for all the details) :

```
>> urbiSend(aibo,'headPan;');
>> [data,timeStamp, tag, type] = urbiGet(aibo)
[00180307:notag] 0.211307
data =
    0.2113
timeStamp =
      180307
tag =
    'notag'
type =
numeric
```

# 4 Synchronous programming

The functions detailed in this section allow you to interact in a synchronous, i.e. blocking, manner with the URBI server. This means that if a message you are not waiting for (for example an alert on the batteries) comes from the server while you are waiting for something else (for example an image), it will be lost. This is therefore generally not a good manner to control the behavior of a robot. Use these functions for simple things or if you want to have very simple action sequences. Prefer the use of the asynchronous functions.

## 4.1 Working with devices and variables

The `urbiSetDeviceField` and `urbiGetDeviceField` functions can be used to set URBI devices or variables and get their values :

```
>> urbiSetDeviceField(aibo,'myVar',10);
>> myvar = urbiGetDeviceField(aibo,'myVar')
myvar =
    10
```

You can also use `urbiSetDevicesFields` and `urbiGetDevicesFields` if you want to do multiple operations at the same time :

```
>> urbiSetDevicesFields(aibo,{'myVar', 'headPan'},[10, -10]);
>> varlist = urbiGetDevicesFields(aibo,{'myVar', 'headPan'})
varlist =
   10.0000  -10.2385
```

The generic function `urbiGet` is able to process messages coming from the server and to output the data in a structure. This function returns several variables : `[data,timeStamp, tag, type]`, with :

- timeStamp : the time stamp of the message
- tag : the tag of the message
- type : the type of the message : 'numeric', 'string', 'system', 'error', 'image', 'sound', or 'otherBIN'
  - if type == numeric, then data is the numeric value of the message
  - if type == string, then data is the string value of the message
  - if type == system, then data is the string value of the system message
  - if type == image, then data is the same as the result of urbiGetImage(aibo) with the fields matrix, width, height, timeStamp
  - if type == sound, then data is the same as the result of urbiGetSound(aibo, length) with the fields samples, Fs, nbits, length
  - if type == otherBIN then data is the vector of bytes of the BIN info

### 4.2 Images and sounds

Images can be recovered using the `urbiGet` function, but can also be recovered using specific functions :

- `[im, width, height, time] = urbiGetImage(aibo)` returns an image taken from the camera device on the URBI server. The image can be displayed using `imshow(im)`.
- `images = urbiGetNImages(aibo)` returns N images taken from the camera device. The result is an array of images, images can be displayed using `imshow(images(i))`.

Sound is also handled by specific functions. `snd = urbiGetSound(aibo, 1s)` returns 1 second of sound recorded from the `micro` device. `snd` is a structure with fields :

- snd.samples : vector of samples
- snd.Fs : sampling frequency used by the server
- snd.channel : number of channels used by the server
- snd.nbits : number of bits used by the server
- snd.length : number of samples in the sound
- snd.duration : duration in ms

This sound structure can be played back by the URBI server using `urbiPlaySound(aibo,snd)` or it can be saved to a wav file using `urbiSound2Wav(snd,'test.wav')`. The sound can also be directly recorded to a wav file using `urbiGetWav(aibo,1s,'test.wav')`.

Finally, a wav file that is on the URBI server file system can be played using `urbiPlayWavLocal(aibo, 'welcome.wav')`, and a wav file that is on the file system of the computer running Matlab can be sent to the URBI server and played using `urbiPlayWavRemote(aibo, 'libUrbi.wav')`.

## 5 Asynchronous programming - Callbacks

The best way to interact with a URBI server is through the use of callback functions. In this programming method, you have to launch a set of commands on the URBI server, that will send messages to Matlab and to associate Matlab functions to these messages that will be automatically launched when a corresponding message is received. In this way, you ensure that all messages coming from the server will be processed and allow parallel processing of various tasks. See the `urbiDemoCallback` for an example.

The first thing you have to do is to link matlab functions to a given message tag :

```
callBack = urbiSetCallback(aibo, @ex_callback_f2,'iseeball');
```

This will allow the library to launch the `ex_callback_f2` matlab function whenever an URBI message with tag `'iseeball'` will be received. For example, the `ex_callback_f2` can simply display the ball position contained in the URBI message :

```
function ans=ex_callback(URBI_message)
ballx = URBI_message.value;
disp(sprintf('I see a ball at %f', ballx)) ;
ans = 1 ;
```

If the return value of your callback function is 0, it is only launched on the first corresponding message reception, if it is 1, the function is called for every corresponding message. See urbiSetCallback html doc for details.

The `URBI_message` variable will be automatically passed to the function. This variable contains all the values taken from the message :

- urbiMessage.timeStamp : the time stamp of the message
- urbiMessage.tag : the tag of the message
- urbiMessage.type : the type of the message : 'numeric', 'string', 'system', 'image', 'sound', or 'otherBIN')
- if type == numeric, then urbiMessage.value is the numeric value of the message
- if type == string, then urbiMessage.value is the string value of the message
- if type == system, then urbiMessage.value is the string value of the system message
- if type == image, then urbiMessage.image is the same as the result of urbiGetImage(aibo) with the fields matrix, width, height, timeStamp
- if type == sound, then urbiMessage.sound is the same as the result of urbiGetSound(aibo, length) with the fields samples, Fs, nbits, length
- if type == otherBIN then urbiMessage.bin is the vector of bytes of the BIN info

Then you have to set up a command on the URBI server that will send the corresponding messages :

```
urbiSend(aibo,'l2:whenever(ball.x != -1) {iseeball:ball.x;wait 300;},');
```

Finally, launch the `urbiProcessEvents` function that will read the messages coming from the URBI server and launch the appropriate callback functions (first argument is the total number of messages to be processed, see html doc for details) :

```
urbiProcessEvents(100, 1000);
```

When you have finished, you can destroy a callback using `urbiDeleteCallBack(aibo, callBack);`. The whole callback system can also be reset using `urbiResetCallbacks`