

YARP (Yet Another Robotic Platform)

Yarp provides a network interface that allows you pass arbitrary messages between multiple computers. It also provides interfaces for communicating with and controlling robotic hardware. This can be used with a multitude of different robotic platforms, and is possible to add new platforms by simply implementing a set of interfaces.

This document will take you through the basics of using YARP to both control an iCub robot, or to pass messages between multiple processes, possibly running on separate computers.

Controlling a robot

Through yarp you can connect to the various limbs on the robot. This can be done in many different ways including a web-based interface, the command line, or from your own code. From here you can issue basic commands to move joints, or you can obtain the joint positions.

Web based interface

When the YARP nameserver is running, it provides a web-based interface listing all the ports that are open. In a terminal, entering the command “yarp where” will give you the location at which the nameserver is running. For example the output may be:

```
Name server is available at ip 15.255.112.22 port 10000
Name server can be browsed at http://15.255.112.22:10000/
```

Going to this address in a web browser will give you access to the list of the available ports that you can connect to. Clicking on these will open a connection from which you can either read data or send commands. Connecting to the icub/inertial port and reading the data on the port will display regular outputs of the inertial sensors on the iCub

(http://eris.liralab.it/wiki/Simulator_README#inertial_port). On the other hand, connecting to the port /icub/head/rpc:i allows you to send set and get commands to change or obtain the positions of the joints. These commands are covered in more detail in the section on connecting to the robot through a command line terminal.

Command line interface

From the terminal you can connect multiple yarp ports together, access information about the server and get a list of ports available, amongst other things. To obtain a complete list of commands available type “yarp help” into a terminal. Documentation on the various different commands can be found at: <http://eris.liralab.it/yarpdoc/yarp.html>.

To connect to the arms, the following two commands can be used:

```
yarp rpc /icubSim/left_arm/rpc:i
yarp rpc /icubSim/right_arm/rpc:i
```

These will open a connection to these ports from which you can enter commands to control the positions of the individual joints, or obtain the current position. The two main commands here are:

```
get enc [joint no.]
set pos [joint no.] [value]
```

For example: `set pos 0 -10` will move joint 0 to -10°, while `get enc 0` will tell you the current encoder position of joint 0. NOTE: Check the safe joint ranges before randomly entering numbers into here.

`get axes` can also be used to find out how many axes are available from that port. This interface can be closed by pressing `Ctrl+c`.

Robot Motor GUI

The robot motor gui is a very useful tool for quickly gaining access to all the joints on the robot. This can be started by entering “robotMotorGui” on a terminal then selecting which limbs to display.

In the robot motor gui, the ranges of individual joints is fixed. However, some of the extremes are not safe on the real iCub. In particular this applies to joint 1 on the arms, which if allowed to move too close to the body will cause tendons in the shoulder to break. Below is a table of safe joint ranges for the various joints in the arm and hand.

Safe joint ranges:

Joint	Min	Max
0	-95	10
1	10	145
2	-35	75
3	15	105
4	-60	60
5	0	80
6	-20	30
7	0	60

A diagram illustrating the movements of the various joints is given on the next page.

Using the robot motor gui, it is possible to generate simple scripts for moving various joints. These scripts can then be run using the iCubDemoY3 app, and are a good place to start for a straight forward motion demo. An example is given below:

iCub hand demo:

```
cd $ICUB_ROOT/app/demoy3/scripts
$ICUB_ROOT/app/default/scripts/manager.py hannover.xml
```

or:

```
cd $ICUB_ROOT/app/demoy3
iCubDemoY3 --positions hannover.txt
```

To stop the demo, simply press ctrl+c

Code based controllers

YARP works with a variety of different programming languages, however it is written in C++. This gives you the freedom to choose which language you use to implement any processes, without having to worry about other processes running at the same time. Messages can be transmitted between processes running in different languages without even noticing.

There are various examples for connecting to YARP ports and to the iCub on the YARP and iCub wikis. Within YARP, there are a variety of different controllers available for use with the iCub, including PID, position, velocity, torque and impedance controllers. The 'documentation' for the different motor controllers can be found on the website at:

http://eris.liralab.it/yarpdoc/group_dev_iface_motor.html. An example of velocity control can be found at: http://eris.liralab.it/iCub/main/dox/html/dir_f75dd2cb72c5a548a2766ffd926dccc.html

Communicating over YARP

To pass messages from one yarp port to another, a connection needs to be formed. This can be done on the command line using:

```
yarp connect /write /read
```

In a couple of terminals, you can then use the following commands to start sending and receiving data across the connection:

```
yarp read /read
```

```
yarp write /write /read
```

and when no longer needed:

```
yarp disconnect /write /read
```

Within YARP, the main container used for sending messages is a 'bottle'. A bottle can contain an arbitrary amount of data of varying types. Some default types are provided as standard, such as integer, boolean, double and string, and custom types can be defined where necessary. The ports can be buffered or unbuffered and a variety of communication protocols are available to customise the message passing and connections where necessary.

Further information about YARP is available from: <http://eris.liralab.it/yarp/>