



# YARP and iCub code tutorials

# Software installation

- [www.yarp.it](http://www.yarp.it) → Installation
- [http://wiki.icub.org/wiki/ICub\\_Software\\_Installation](http://wiki.icub.org/wiki/ICub_Software_Installation)
- Installation from sources
- Linux virtual machine:  
[http://www.icub.org/download/other/Ubuntu\\_1404\\_64-bit.zip](http://www.icub.org/download/other/Ubuntu_1404_64-bit.zip)



# Code available on github

- <https://github.com/lornat75/Teaching>

- Type:

git clone

git@github.com:lornat75/Teaching.git

# Yarp from command line

# A (very) simple example: read data to/from a port

[on terminal 1] yarpserver

[on terminal 2] yarp read /read

[on terminal 3] yarp write /write /read



```
$ yarp write /write /read
Port /write listening at tcp://127.0.0.1:10012
yarp: Sending output from /write to /read using tcp
Added output connection from "/write" to "/read"
hello yarp
1 2 3
```

```
$ yarp read /read
Port /read listening at tcp://127.0.0.1:10002
yarp: Receiving input from /write to /read using tcp
hello yarp
1 2 3
```



yarp name list

yarp name query /read

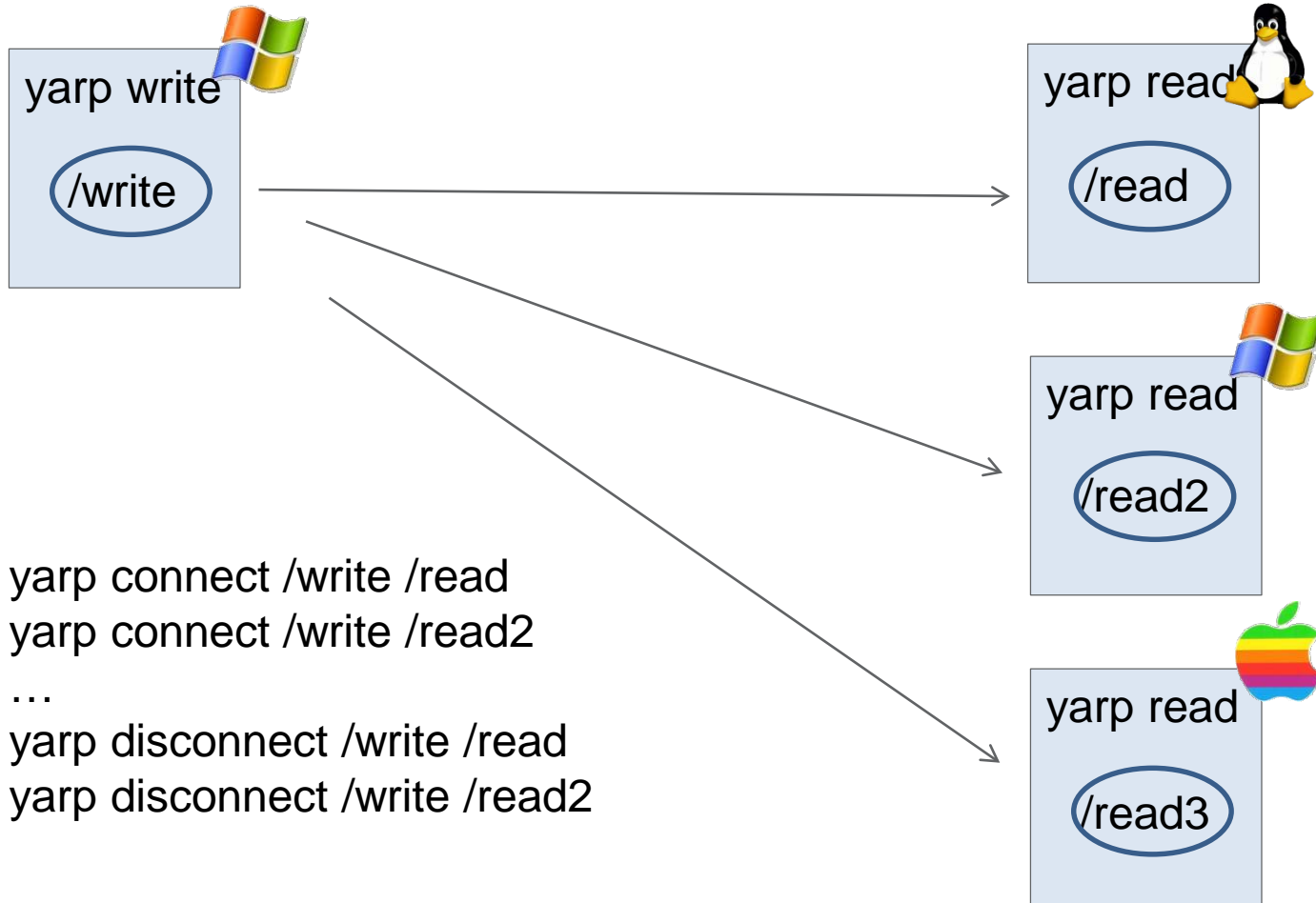
yarp name register PORT CARRIER IP  
NUMBER

yarp name unregister PORT

# how the network grows

It is easy to add, for example, another reader...

Processes can run on different machines, with different OS



# yarpview



```
yarpdev --device test_grabber --name /cam/right  
yarpdev --device test_grabber --name /cam/left  
yarpview --name /view1  
yarpview --name /view2
```

```
yarp connect /cam/right /view1  
yarp connect /cam/left /view2
```





# YARP configuration file

Where is the yarp nameserver?

```
icub@ubuntu-1404-64-vm:~$ yarp detect
```

```
icub@ubuntu-1404-64-vm:~$ yarp conf  
/home/icub/.config/yarp/yarp.conf
```

```
icub@ubuntu-1404-64-vm:~$ cat /home/icub/.config/yarp/yarp.conf  
192.168.59.128 10000 yarp
```

- **yarpserver** by default decides based on the available network card (i.e. eth0) on which adapter/ip to listen
- You can manually modify the yarp.conf file to change adapter/ip.
- **yarpserver** can accept that (--read) or overwrite it (--write).

# Connecting to mjpeg



Make sure these CMake flags are enabled:  
CREATE\_OPTIONAL\_CARRIERS=ON  
ENABLE\_yarpcar\_mjpeg\_carrier=ON

Run ccmake in YARP's build directory

```
yarpview --name /view
```

```
yarp connect /195.67.26.73:80 /view mjpeg+path./mjpg/video.mjpg
```

Alternatively you can register the remote address manually:

```
yarp name register /webcam mjpeg+path./mjpg/video.mjpg 195.67.26.73 80
```

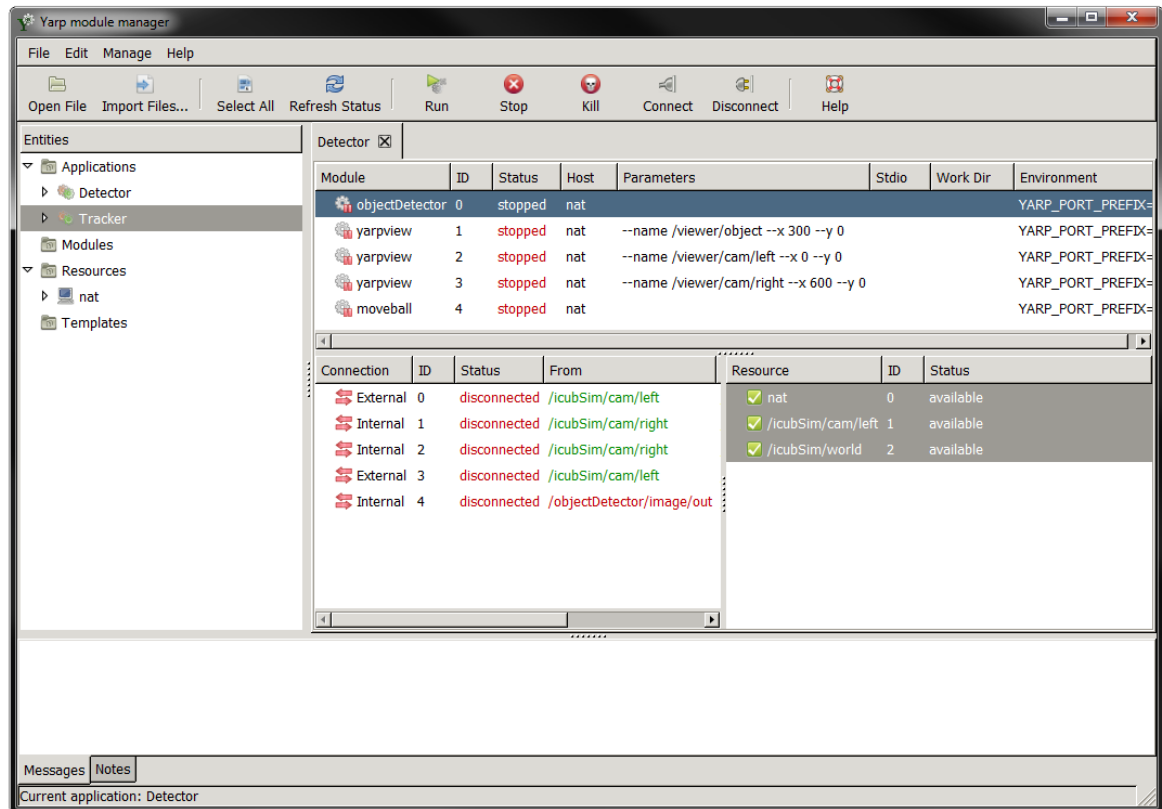
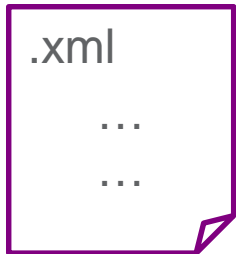
And use /webcam as an alias

```
yarp connect /webcam /view
```

# Automation

# The YARP Manager

- The **yarpmanager** is a graphic interface to monitor processes
- It allows to start/stopping/monitor, redirect i/o
- In addition it automates establishing connections between modules



**Yarp module manager**

File Edit Manage Help

Open File Import Files... Select All Refresh Status Run Stop Kill Connect Disconnect Help

**Entities**

- Applications
  - Detector
  - Tracker
- Modules
- Resources
  - nat
- Templates

**Detector**

Module	ID	Status	Host	Parameters	Stdio	Work Dir	Environment
objectDetector	0	running	nat				YARP_PORT_PREFIX=
yarpview	1	running	nat	--name /viewer/object --x 300 --y 0			YARP_PORT_PREFIX=
yarpview	2	running	nat	--name /viewer/cam/left --x 0 --y 0			YARP_PORT_PREFIX=
yarpview	3	running	nat	--name /viewer/cam/right --x 600 --y 0			YARP_PORT_PREFIX=
moveball	4	running	nat				YARP_PORT_PREFIX=

Connection	ID	Status	From	Resource	ID	Status
External	0	connected	/icubSim/cam/left	nat	0	unknown
Internal	1	connected	/icubSim/cam/right	/icubSim/cam/left	1	unknown
Internal	2	connected	/icubSim/cam/right	/icubSim/world	2	unknown
External	3	connected	/icubSim/cam/left			
Internal	4	connected	/objectDetector/image/out			

Messages Notes

Current application: Detector



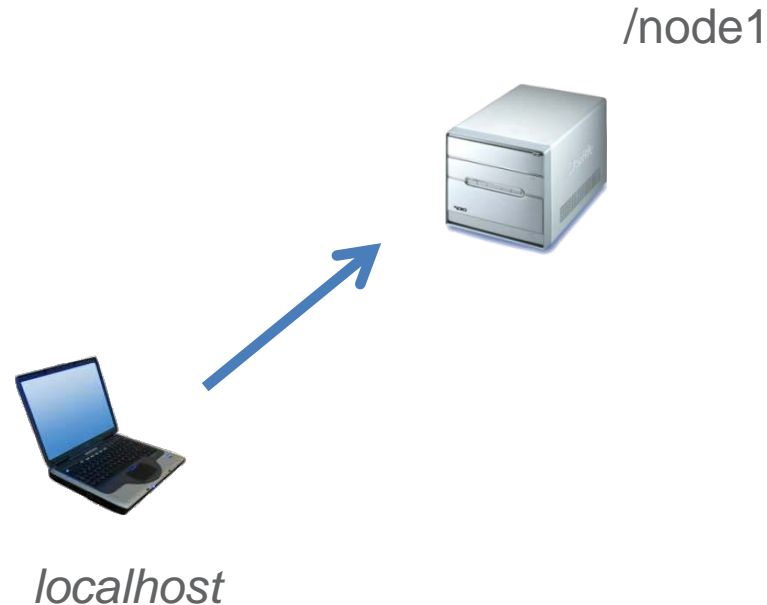
# yarpmanager documentation

- <http://wiki.icub.org/yarpd/doc/yarpmanager.html>

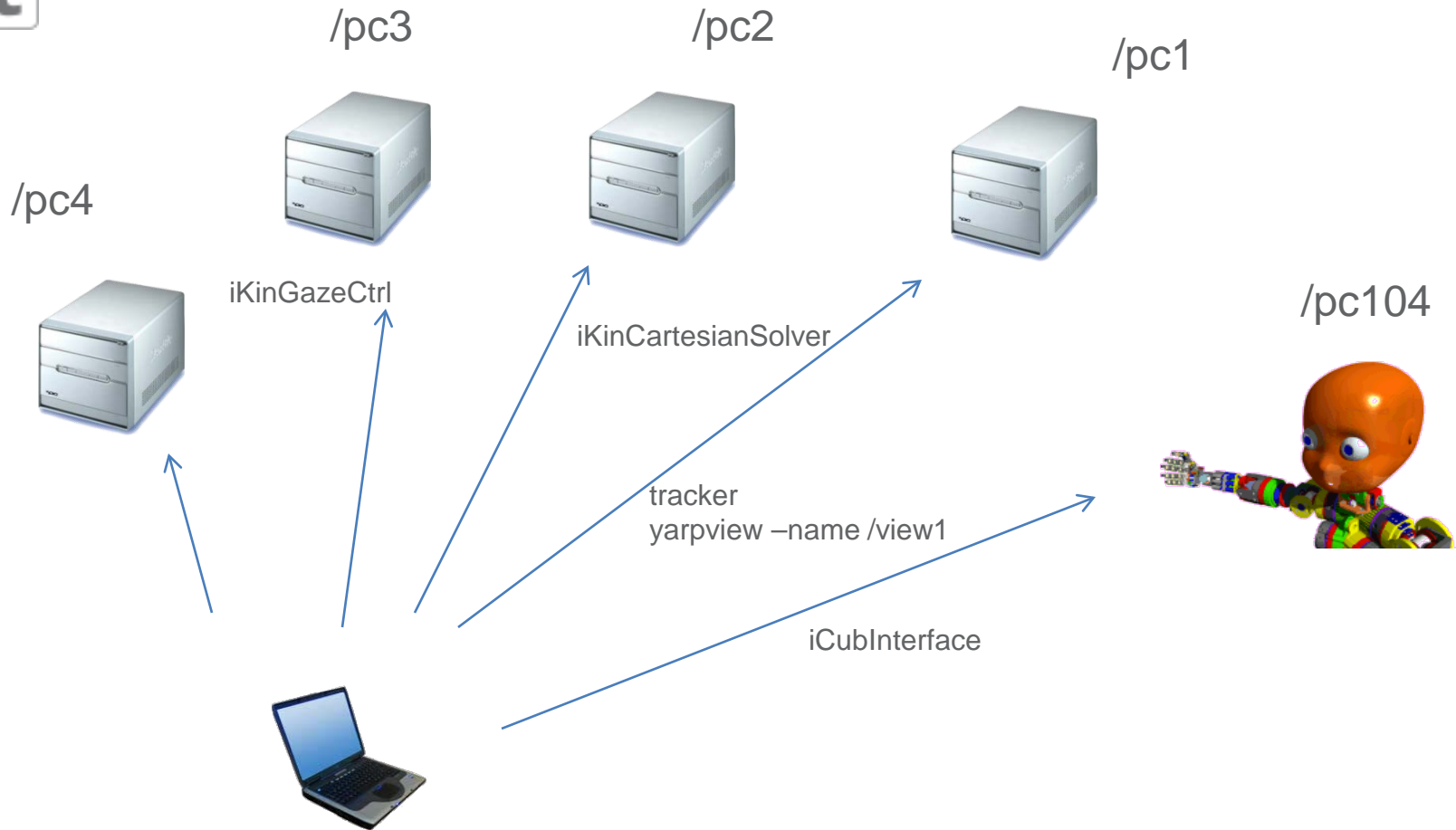
run a server, which will wait for  
commands on /node1

Starting a server

```
$node1: yarprun -server /node1
```



- The manager has two ways to execute processes: locally (localhost) or through yarprun
- yarprun is a server that waits for commands on a port
- start/termination/kill monitor lifecycle







# Syntax

<application>

<name>Name of the application</name> //this can be anything, just a symbolic name

<dependencies>

<port>/port1 </port>

<port>/port2 </port>

</dependencies>

<module>

<name>mymodule1 </name>

<parameters>--threshold 1 --name /myName</parameters>

<node>localhost</node>

</module>

<module>

<name>mymodule2</name>

...

</module>

<connection>

<from>/port1</from>

<to>/otherport</to>

<protocol>udp</protocol>

</connection>

<connection>

...

</connection>



```
<application>
```

```
  <name>Name of the application</name> //this can be anything, just a symbolic name
```

```
  <module>
```

```
    <name>yarpdev</name>
```

```
    <parameters>--device test_grabber --name /cam/right</parameters>
```

```
    <node>localhost</node>
```

```
  </module>
```

```
  <module>
```

```
    <name>yarpview</name>
```

```
    <parameters>--name /view/right</parameters>
```

```
    <node>localhost</node>
```

```
  </module>
```

```
  <connection>
```

```
    <from>/cam/right</from>
```

```
    <to>/view/right</to>
```

```
    <protocol>udp</protocol>
```

```
  </connection>
```

```
</application>
```

or any other node in the network:

/node1, /node2 etc..

E.g. on the iCub: icub14, icub15, icub-b11...

# Other tags

```
<dependencies>
```

```
  <port>/icub/cam/left</port>
```

```
  <port>/icub/cam/right</port>
```

```
</dependencies>
```

```
<module>
```

```
  ...
```

```
  <workdir>C:/mydir</workdir>
```

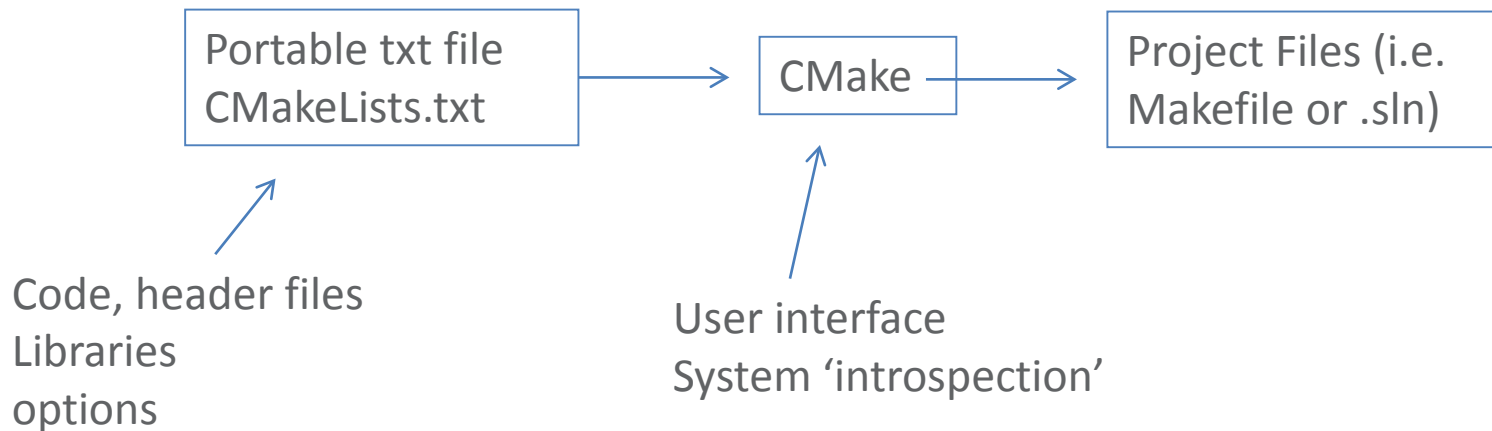
```
  <stdio>node3</stdio>
```

```
</module>
```

# CMake Basics

# Introduction

- Open source build manager
- Specify build parameters in a simple portable text format



# Problems solved by CMake

- Write and maintain project files for multiple platforms
- Optional components?
- Build on more than a single machine:  
different OS have different libraries, same OS  
can be installed differently → automatically  
search for programs libraries header files
- Build directory tree different from source tree
- Handle dependencies
- Static versus Dynamic libraries
- ...

# Basics

- Commands (case insensitive)
- Variables (case sensitive)

```
command(a b c)  
set(FOO a b c)
```

```
command(${FOO})  
command("${FOO}")
```

Consider:

```
set(PATH_TO_MY_FILE C:\program files\myfile)  
command(${PATH_TO_MY_FILE})
```

```
command("${PATH_TO_MY_FILE}")
```



# Hello World with CMake

```
cmake_minimum_required(VERSION 2.8)

project(hello)

include_directories(${CMAKE_CURRENT_SOURCE_DIR})

message(STATUS "--> Hello from CMake")

if (WIN32)
    message("--> Running on windows")
else()
    message("--> Assuming running on Linux")
endif()

if (NOT EXISTS "${CMAKE_CURRENT_SOURCE_DIR}/hello.cpp")
    message(FATAL_ERROR "File hello.cpp not found!")
endif()

add_executable(hello hello.cpp)
```



# How to run CMake

- Source versus build directories
- From command line:
  - mkdir build
  - cd build
  - cmake ../ or ccmake ../
- From gui:
  - mkdir build
  - cmake-gui
  - Set source and build directories
- Hit “c” until you get “g”

When build = source dir: in source build

When build != source dir: out of source build

# Cache

- Some variables are determined only once and cached on disk; CMake will not touch them, only the user can
- E.g. user options or result of system introspection, info that are expensive to determine (compiler to use, system libraries, etc..)
- To do a fresh restart, clean the cache
  - From the gui
  - Remove CMakeCache.txt

# Commands on targets

- `add_executable(name file1.cpp file2.cpp header1.h header2.h)`
- `target_link_libraries(name libname)`
- `add_library(name file1.cpp file2.cpp header1.h)`
- `include_directories(dir1 dir2)`
- `add_definitions(-DFOO -DBAR)`

# Example:

```
#if _ENABLE_DEBUG_  
    printf("Value of variable v is %d", v);  
#endif  
  
option(ENABLE_DEBUG "Enable debugging messages"  
      FALSE)  
  
if (ENABLE_DEBUG)  
    message(STATUS "Debugging messages are enabled")  
    add_definitions(-D_ENABLE_DEBUG_)  
endif()
```

# Installation

- In some builds include an installation step
- You can add installation rules using CMake

```
install(TARGETS myExe RUNTIME DESTINATION <dir>)
```

```
install(FILES files DESTINATION <dir>)
```

<dir> can be:

- Absolute path
- Relative path, in this case it will be CMAKE\_INSTALL\_PREFIX\<dir>
- The user can customize CMAKE\_INSTALL\_PREFIX



# Hello World with CMake (2)

...

```
add_executable(hello hello.cpp)
```

```
install(TARGETS hello
```

```
    RUNTIME DESTINATION
```

```
    ${CMAKE_CURRENT_SOURCE_DIR}/../bin)
```

# Finding libraries

- The problem
- You want to compile an executable that links libraries from another package, e.g. YARP
- Naïve way:

```
cmake_minimum_required(VERSION 2.8)
```

```
project(hello)
```

```
include_directories("C:\Program files\yarp\include")
```

```
add_executable(hello hello.cpp)
```

```
target_link_libraries(hello "C:\Program files\yarp\lib\libYARP_OS.lib")
```

# Finding libraries

- The problem
- You want to compile an executable that links libraries from another package, e.g. YARP
- Naïve way:

```
cmake_minimum_required(VERSION 2.8)
```

```
project(hello)
```

```
include_directories("C:\Program files\yarp\include")
```

```
add_executable(hello hello.cpp)
```

```
target_link_libraries(hello "C:\Program files\yarp\lib\libYARP_OS.lib")
```

Os dependent (.a in Linux)

Installation dependent



# Finding libraries...

- CMake has a few commands that can be used to find directories, executables and libraries inside a computer

```
find_file(<var> name dir1 dir2)
```

```
find_library(<var> name dir1 dir2)
```

```
find_path(<var> name dir1 dir2)
```

- However there is a better interface...

# find\_package()

- A package should provide you:
  - Paths to libraries
  - Paths to header files
  - Linker flags (if any)

`find_package(<PACKAGE> [VERSION])`

This function attempts to locate the package called <PACKAGE> and will return a set of variables:

`<PACKAGE>_FOUND`

`<PACKAGE>_INCLUDE_DIRS`

`<PACKAGE>_LIBRARIES`

`<PACKAGE>_VERSION`

`<PACKAGE>_VERSION_MAJOR`

`<PACKAGE>_VERSION_MINOR`

# Example:

```
find_package(YARP)
```

```
YARP_FOUND
```

```
YARP_INCLUDE_DIRS
```

```
YARP_LIBRARIES
```

- How does find\_package() work?
  - Looks for system directories
    - C:\Program files\<package>
    - /usr/<package>
    - /usr/local/<package>
    - ...
  - Look for environment variables, very popular  
<PACKAGE>\_DIR
- CMake does not enforce a particular set of variables  
each package set different variables
- Other examples:
  - <PACKAGE>\_INCLUDE\_DIR
  - <PACKAGE>\_LIBS
  - etc..



# Hello YARP

```
cmake_minimum_required(VERSION 2.8)
```

```
project(myproject)
```

```
find_package(YARP)
```

```
include_directories(${YARP_INCLUDE_DIRS})
```

```
add_executable(hello hello.cpp)
```

```
target_link_libraries(hello ${YARP_LIBRARIES})
```

```
add_executable(hello2 hello.cpp)
```

```
target_link_libraries(hello2 ${YARP_LIBRARIES})
```



# Hello yarp

```
#include <stdio.h>
#include <yarp/os/Time.h>

int main()
{
    printf("Starting the application\n");
    int times=10;

    while(times-->0)
    {
        printf("Hello iCub\n");
        yarp::os::Time::delay(0.5);    //wait 0.5 seconds
    }
    printf("Goodbye!\n");
}
```

# Ports

# A (very) simple example: read data to/from a port

[on terminal 1] yarpserver

[on terminal 2] yarp read /read

[on terminal 3] yarp write /write /read



```
$ yarp write /write /read
Port /write listening at tcp://127.0.0.1:10012
yarp: Sending output from /write to /read using tcp
Added output connection from "/write" to "/read"
hello yarp
1 2 3
```

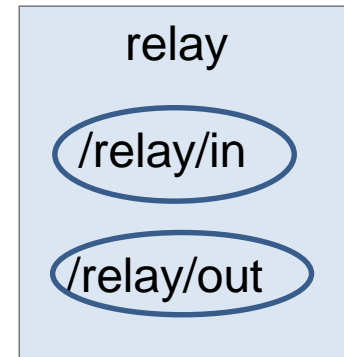
```
$ yarp read /read
Port /read listening at tcp://127.0.0.1:10002
yarp: Receiving input from /write to /read using tcp
hello yarp
1 2 3
```



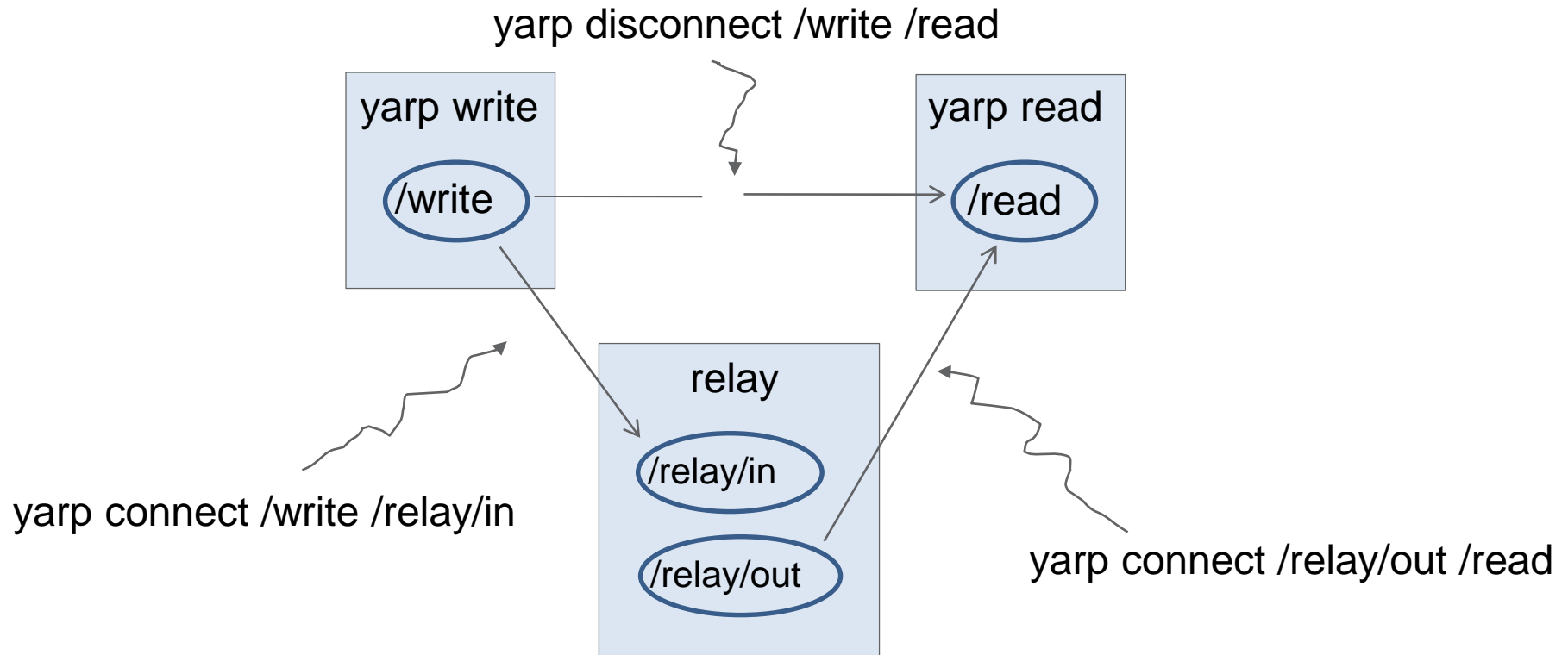
# How do we get this?

Let's now write a simple "relay" executable which takes whatever comes from a port and forwards it to another one.

```
int main(int argc, char *argv) {  
    Network yarp;  
    Port inPort;  
    inPort.open("/relay/in");  
  
    Port outPort;  
    outPort.open("/relay/out");  
  
    while (true) {  
        cout << "waiting for input" << endl;  
        Bottle input,output;  
        inPort.read(input);  
        output=input;  
        cout << "writing " << output.toString().c_str() << endl;  
        outPort.write(output);  
    }  
    return 0;  
}
```



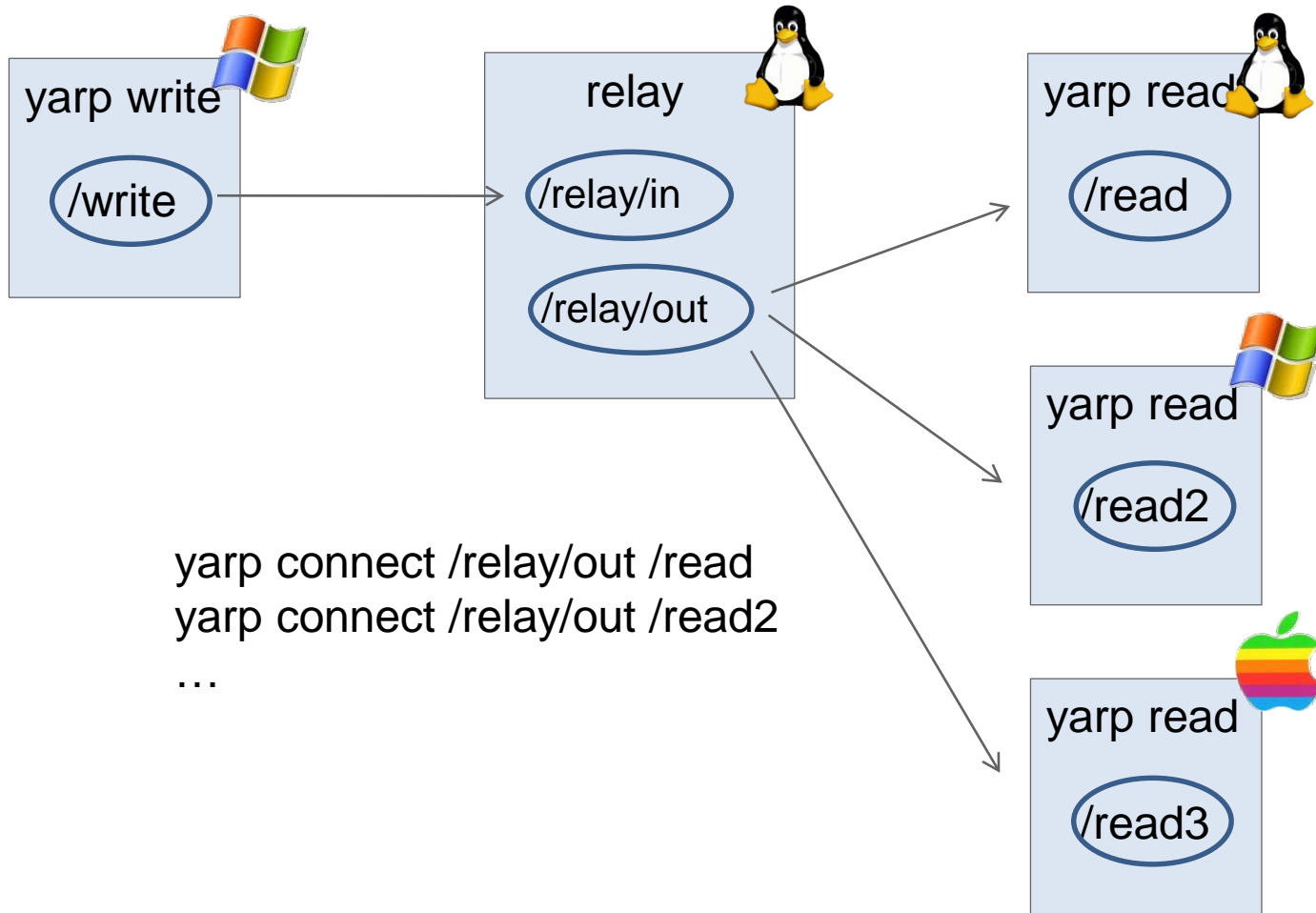
# Connect the new module to our network



# how the network grows

It is easy to add, for example, another reader...

Processes can run on different machines, with different OS



# BufferedPort

- In the previous example timing between ports is coupled:
  - The reader waits until data arrives to the port
  - The writer waits until data is transmitted
- Buffered ports allow decoupling time:
  - non blocking read
  - non blocking write
- May lose messages

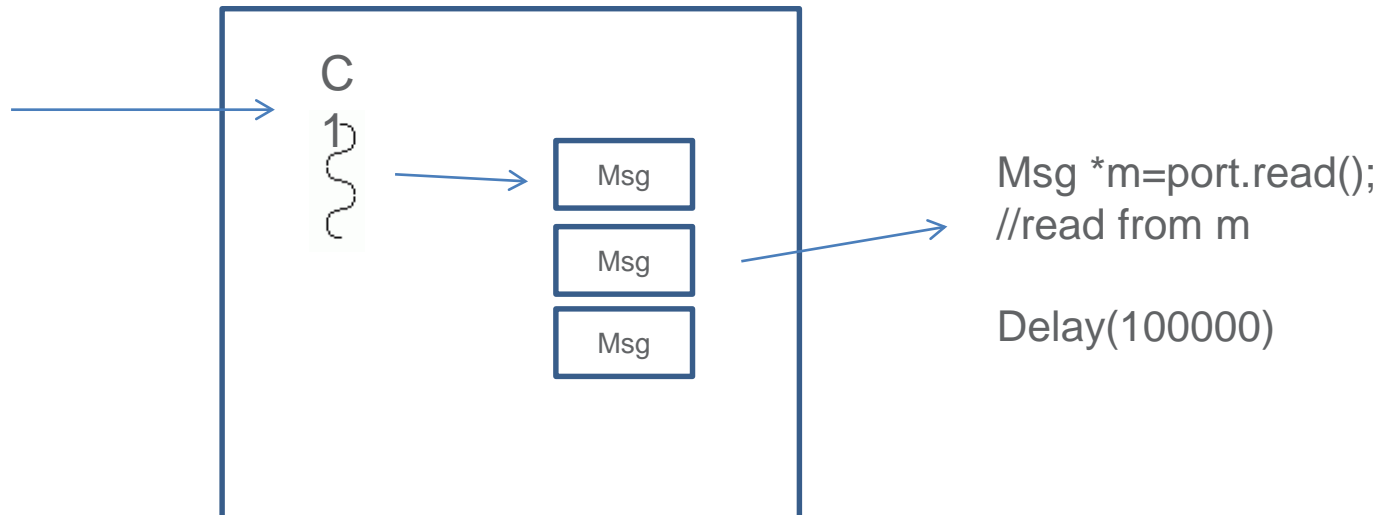
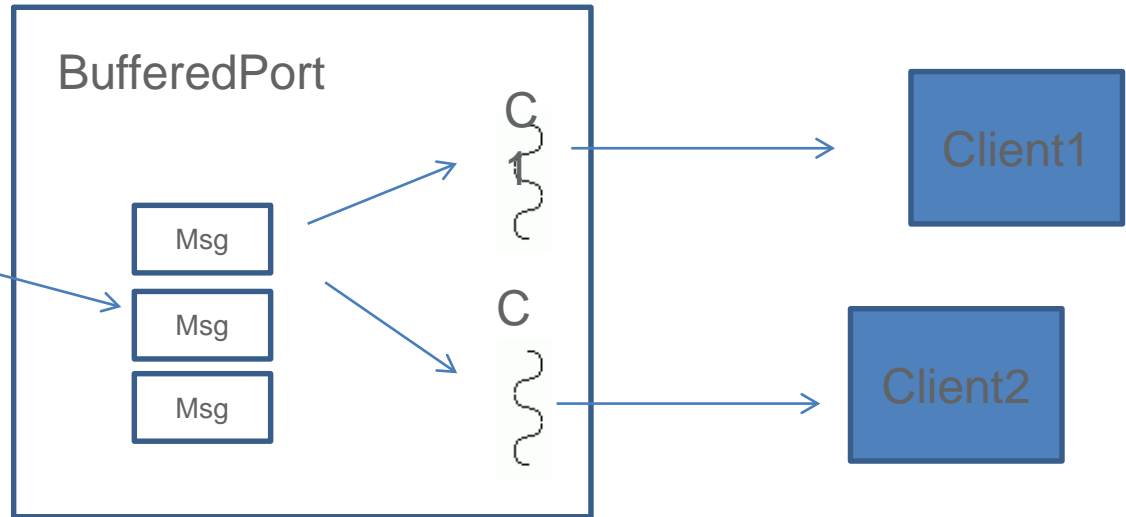
- Read:

```
BufferedPort<Bottle> p;      // Create a port.
p.open("/in");               // Give it a name on the network.
while (true) {
    Bottle *b = p.read();     // Read/wait for until data arrives. ...
    // Do something with data in *b
}
```

- Write:

```
BufferedPort<Bottle> p;      // Create a port.
p.open("/out");              // Give it a name on the network.
while (true) {
    Bottle& b = p.prepare();  // Get a place to store things. ...
    // Generate data.
    p.write();                // Send the data.
}
```

```
Msg m
=port.prepare();
//fill m
Port.write()
```



- Polling: when you do not want to wait for input data:

```
BufferedPort<Bottle> p;
```

```
...
```

```
Bottle *b = p.read(false);
```

```
if (b!=NULL) {
```

```
    // data received in *b
```

```
}
```

# Getting callbacks



- Callbacks: useful if you want to be notified when data arrives
- Easy to do with BufferedPorts

```
class DataPort : public BufferedPort<Bottle> {  
    virtual void onRead(Bottle& b) {  
        // process data in b  
    }  
};  
...  
DataPort p;  
p.useCallback(); // input should go to onRead()  
callback  
p.open("/in");
```

## Things are a bit more complicated with normal ports

```
class DataProcessor : public PortReader {  
    virtual bool read(ConnectionReader& connection) {  
        Bottle b;  
        bool ok = b.read(connection);  
        if (!ok) return false;  
        // process data in b  
        return true;  
    }  
};  
  
Port p;  
p.open(..)  
DataProcessor processor;  
...  
p.setReader(processor); // no need to call p.read() on port any more.
```

# Replies in a callback

```
class DataProcessor : public PortReader {
    virtual bool read(ConnectionReader& connection) {
        Bottle in, out;
        bool ok = in.read(connection);
        if (!ok) return false;
        ...    // process data "in", prepare "out"
        ConnectionWriter *returnToSender = connection.getWriter();
        if (returnToSender!=NULL) {
            out.write(*returnToSender);
        }
        return true;
    }
};

DataProcessor processor;

...
p.setReader(processor); // no need to call p.read() on port any
more.
```

# Bidirectional communication: Getting replies

# Client side

```
RpcClient p;           // Create a port.
p.open("/out");        // Give it a name on the network.
while (true) {
    Bottle in,out;      // Make places to store things.
    ...                 // prepare command "out".
    p.write(out,in);    // send command, wait for reply.
    ...                 // process response "in".
}
```

# Server side

```
RpcServer p;                // Create a port.
p.open("/in");              // Give it a name on the network.
Bottle in, out;             // Make places to store things.
while (true) {
    p.read(in,true);         // Read and warn that we'll be replying.
    ...                      // Do something with data, prepare
    reply
    p.reply(out);            // send reply.
}
```

# YARP modules: RFModule

# The RFModule class

- You create a new module by deriving a new class from RFModule

```
class MyModule:public RFModule
```

```
{
```

```
public:
```

```
    bool configure(ResourceFinder &rf)
```

```
    { //module configuration }
```

```
        bool close()
```

```
        { //code executed at shutdown }
```

```
};
```

← get parameters form RF and  
configure the module, return true  
on success, false otherwise

← perform cleanup, close ports, delete memory

```
MyModule module;
```

```
ResourceFinder rf;
```

```
//configure resource finder
```

← We skip this

```
module.runModule(rf);    //if configure returns true block here until the module  
closes
```



- What does a module do?
- Nothing, really...

- What does a module do?
- Nothing, really...
- Wait for termination signal (message or ctrl-c)
- Can be configured to receive messages from a port/keyboard
- Can perform periodic activities
- It is a container for active objects (threads)

# Attach callbacks

```
class MyModule::RFModule
{
    Port handlerPort;

    ...
    bool configure(ResourceFinder &rf)
    {
        // use rf to configure your module

        handlerPort.open("/myModule");
        attach(handlerPort);

        ...
    }
    ...
}
```

- Now add a respond message to catch data from terminal or/and the handler port

```
// Message handler. Just echo all received messages.  
bool respond(const Bottle& command, Bottle& reply)  
{  
    cout<<"Got something, echo is on"<<endl;  
    if (command.get(0).asString()=="quit")  
        return false;  
    else  
        reply=command;  
    return true;  
}
```

# Periodic Activities

- In MyModule overload:

```
double getPeriod()  
{ return 1; }
```

← define period in seconds

```
bool updateModule()  
{  
    // place here code that will be  
    // executed every "getPeriod"  
seconds  
    return true;  
}
```

← this function will be executed until termination

- You can interrupt blocking reads on ports in the interrupt method:

```
bool interruptModule()  
{  
    port1.interrupt();  
    port2.interrupt();  
    ...  
    return true;  
}
```

# Threads

```
#include <yarp/os/Thread.h>
```

```
Class yarp::os::Thread
```

```
{
```

```
public:
```

```
    virtual bool start();
```

```
    virtual bool stop();
```

```
    virtual bool threadInit();
```

```
    virtual bool threadRelease();
```

```
    virtual void run();
```

```
    bool isStopping();
```

```
};
```

yarp::os::Thread is  
the class that  
provides thread  
support in YARP





```
#include <yarp/os/Thread.h>
```

```
Class MyThread: public Thread  
{  
public  
    void run()  
    {  
        while(!isStopping)  
            //thread body  
    }  
};
```

```
MyThread thread;  
thread.start();  
...  
thread.stop();
```

You can implement  
your own thread by  
deriving a class from  
Thread

```
Class MyThread: public Thread
{
public
    bool threadInit()
    {
        //perform init tasks, memory allocation...
        //return true if successful false otherwise
    }
    bool threadRelease()
    {
        //cleanup memory, release resources...
    }
    void run() {..}
}
```

Override  
threadInit() and  
threadRelease() to  
perform initialization  
and cleanup:

```
#include <yarp/os/RateThread.h>
```

```
Class yarp::os::RateThread  
{  
public:  
    RateThread(int period); //periodicity,  
ms  
    virtual bool start();  
    virtual bool stop();  
  
    virtual bool threadInit();  
    virtual bool threadRelease();  
    virtual void run();  
  
};
```

Very often you want  
a thread to perform  
periodic activities  
(e.g. control loop)

RateThread supports  
periodic threads



```
#include <yarp/os/RateThread.h>
```

```
Class MyRateThread: public  
RateThread  
{  
public:  
    MyRateThread(int p=50):  
    RateThread(p){}  
  
    void run()  
    {  
        ...  
    }  
};  
MyRateThread rthread;  
rthread.start();  
...  
rthread.stop();
```

# Getting images

- YARP defines an image class
- ImageOf<...> is a template class that provides:
  - basic methods for image manipulation
  - support for remotization (i.e. images can travel across Ports/the network)
- data format is opencv compatible
- See: [YARP image class online documentation](#)

- Images from cameras are streamed from two ports:
  - /icub/cam/right
  - /icub/cam/left
- Easily read:

```
BufferedPort<ImageOf<PixelRgb> > imagePort;  
imagePort.open("/imageProc/image/in");  
ImageOf<PixelRgb> *image = imagePort.read(); //read an  
image;
```



```
BufferedPort<ImageOf<PixelRgb> > imagePort;
```

```
imagePort.open("/imageProc/image/in");
```

```
//read an image:
```

```
ImageOf<PixelRgb> *image = imagePort.read();
```

```
//do something with the image, for example cycle through all pixels
```

```
int ct=0
```

```
for (int x=0; x<image->width(); x++) {
```

```
    for (int y=0; y<image->height(); y++) {
```

```
        PixelRgb& pixel = image->pixel(x,y);
```

```
        // very simple test for blueishness
```

```
        // make sure blue level exceeds red and green by a certain factor
```

```
        if (pixel.b>pixel.r*1.2+10 && pixel.b>pixel.g*1.2+10) {
```

```
            xMean += x;
```

```
            yMean += y;
```

```
            ct++;
```

```
        }
```

```
    }
```

```
}
```

```
if (ct>0) {
```

```
    xMean /= ct;
```

```
    yMean /= ct;
```

```
}
```

```
printf("Best guess at blue target: %g %g\n", xMean, yMean);
```



- Complete tutorial shows a program example that control the gaze of the robot to fixate a blue ball:
  - [http://wiki.icub.org/iCub/dox/html/icub\\_basic\\_image\\_processing.html](http://wiki.icub.org/iCub/dox/html/icub_basic_image_processing.html)



# Controlling the simulator

- Set of ports for parts {head} {left\_arm} {torso} etc...
- Ports:

/icubSim/head/rpc:i

/icubSim/head/command:i

/icubSim/head/state:o

```
icub@ubuntu-1404-64-vm:~$ yarp rpc /icubSim/head/rpc:i
```

```
>>get encs
```

```
Response: [is] encs (-0.000015 0.000004 -0.000004 -0.0 0.0 -0.0) [tsta] 1 1434026836.655992 [ok]
```

```
>>set pos 0 -10
```

```
Response: [ok]
```

```
>>set pos 1 20
```

```
Response: [ok]
```

```
>>set poss (0 0 0 0 0 0)
```

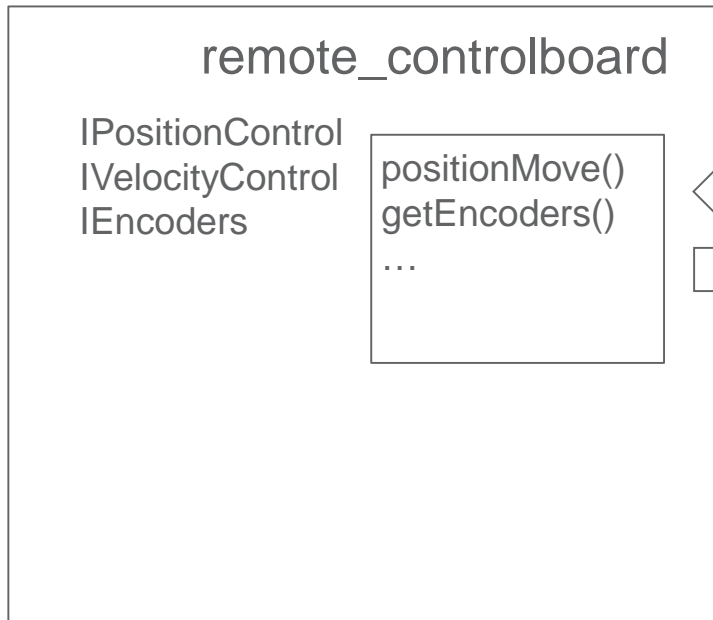
```
Response: [ok]
```

```
>>get encs
```

```
Response: [is] encs (-0.0005 0.000971 -0.000004 -0.0 0.0 -0.0) [tsta] 2 1434026858.553787 [ok]
```

```
>>
```

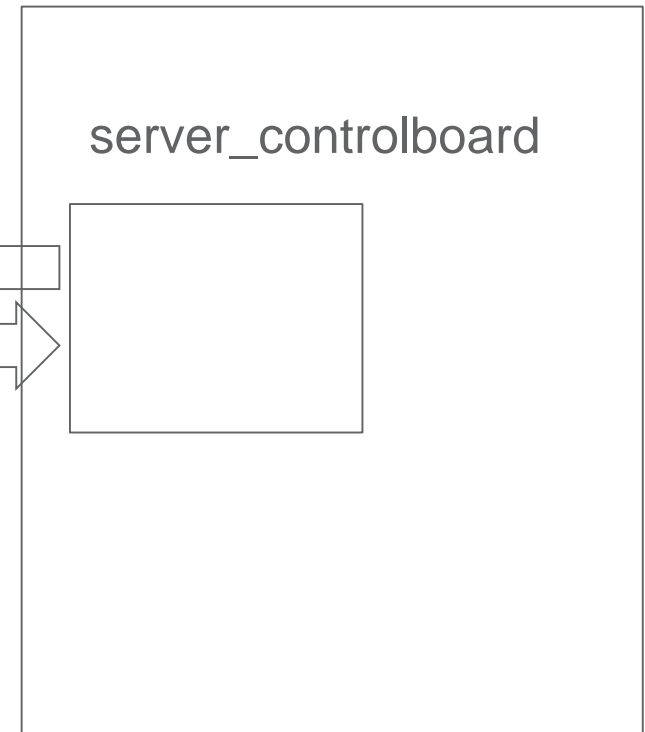
## User code



```

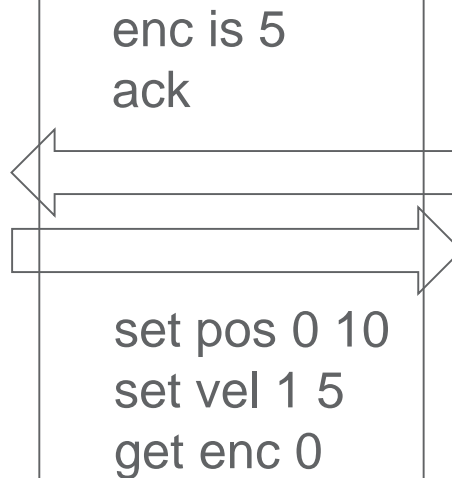
/client/left_arm/rpc:i
/client/left_arm/state:i
/client/left_arm/command:o
  
```

## Simulator/iCub



```

/iCubSim/left_arm/rpc:i
/iCubSim/left_arm/state:i
/iCubSim/left_arm/command:o
  
```



- [http://wiki.icub.org/yarpdoc/classyarp\\_1\\_1dev\\_1\\_1IPositionControl.html](http://wiki.icub.org/yarpdoc/classyarp_1_1dev_1_1IPositionControl.html)
- [http://wiki.icub.org/yarpdoc/classyarp\\_1\\_1dev\\_1\\_1IVelocityControl.html](http://wiki.icub.org/yarpdoc/classyarp_1_1dev_1_1IVelocityControl.html)

# PolyDriver

