YARP hands-on

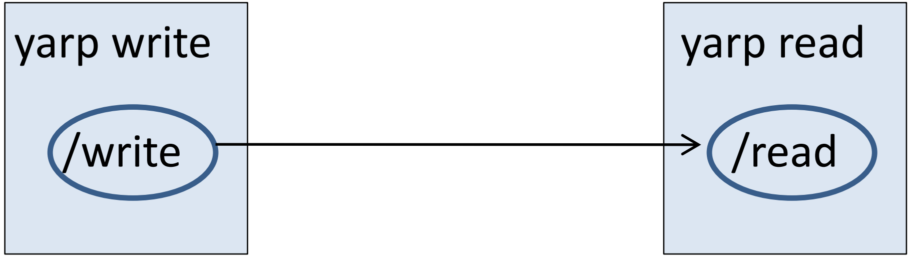
Lorenzo Natale

iCub Facility

Istituto Italiano di Tecnologia, Genova, Italy

YARP from command line

```
[on terminal 1] yarpserver
[on terminal 2] yarp read /read
[on terminal 3] yarp write /write /read
```

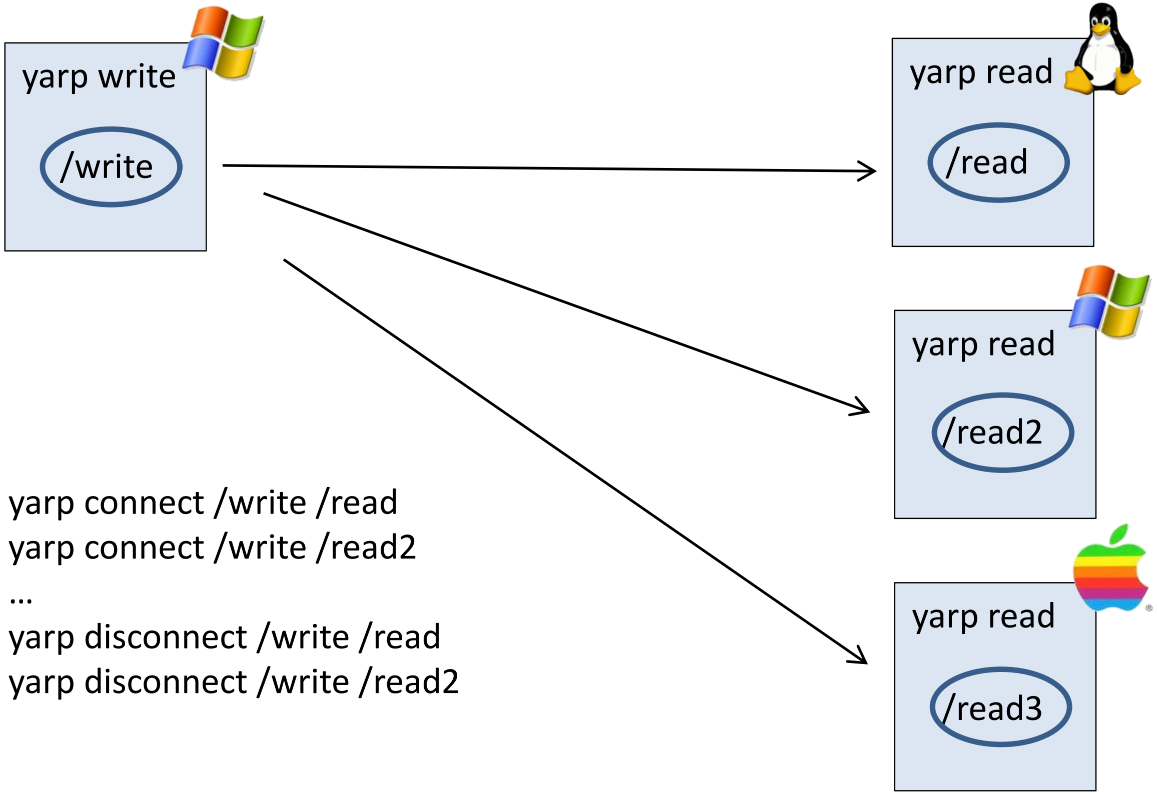


```
$ yarp write /write /read
Port /write listening at tcp://127.0.0.1:10012
yarp: Sending output from /write to /read using tcp
Added output connection from "/write" to "/read"
hello yarp
1 2 3
```

```
$ 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
```

It is easy to add, for example, another reader…
Processes can run on different machines, with different OS



yarp write

/write

yarp read

/read

yarp read

/read2

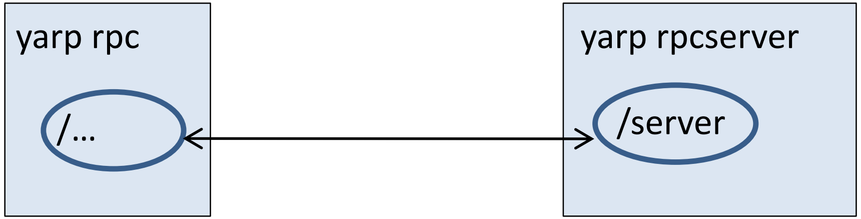
yarp read

/read3

```
yarp connect /write /read
yarp connect /write /read2
…
yarp disconnect /write /read
yarp disconnect /write /read2
```

YARP RPC

[on terminal 1] yarpserver
[on terminal 2] yarp rpcserver /server
[on terminal 3] yarp rpc /server



```
$ yarp rpc /server
yarp rpc /server
hello
Response: hello to you
from client
Response: from server
^C
```

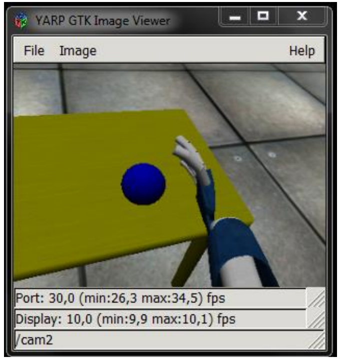
```
$ yarp rpcserver /server
Waiting for a message...
Message: hello
Reply: hello to you
Waiting for a message...
Message: from client
Reply: from server
```

YARP configuration file

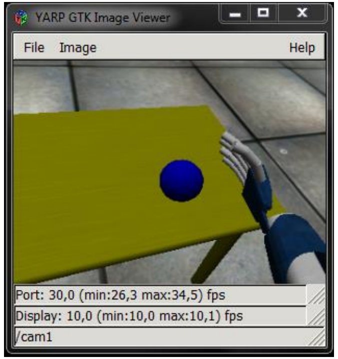
Where is the nameserver?

```
$ yarp detect

$ yarp conf
/home/icub/.config/yarp/yarp.conf

$ 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).



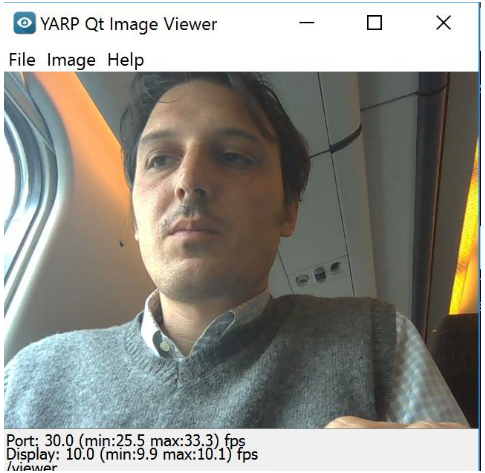
```
Port: 30,0 (min:26,3 max:34,5) fps
Display: 10,0 (min:9,9 max:10,1) fps
/cam2
```



```
Port: 30,0 (min:26,3 max:34,5) fps
Display: 10,0 (min:10,0 max:10,1) fps
/cam1
```

```
$ 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
```

Use your webcam!



```
Port: 30.0 (min:25.5 max:33.3) fps
Display: 10.0 (min:9.9 max:10.1) fps
/viewer
```

Run ccmake in YARP’s build directory

Make sure these CMake flags are enabled:

```
CREATE_DEVICE_LIBRARY_MODULES=ON
ENABLE_yarpmod_opencv_grabber=ON
```

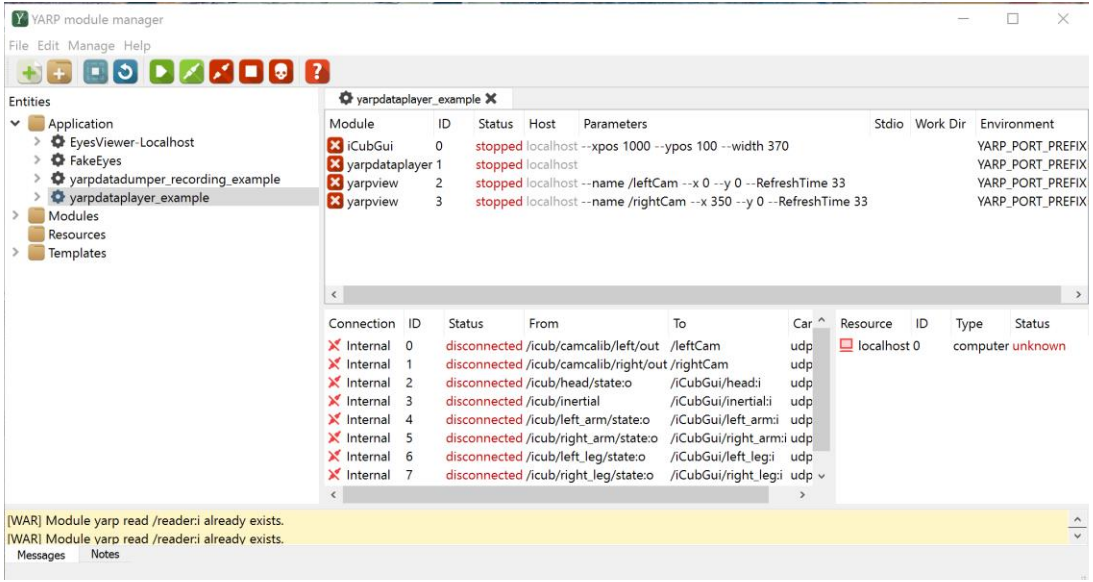
Rebuild (and install):

```
$ make
$ sudo make install
```

```
$ yarpview --name /viewer
$ yarpdev --device opencv_grabber --camera 0
$ yarp connect /grabber /viewer
```

Play recorded sequence

```
$ wget http://www.icub.org/download/software/datasetplayer-demo/testData_20120803_095402.zip
$ unzip testData_20120803_095402.zip
$ yarpmanager
```

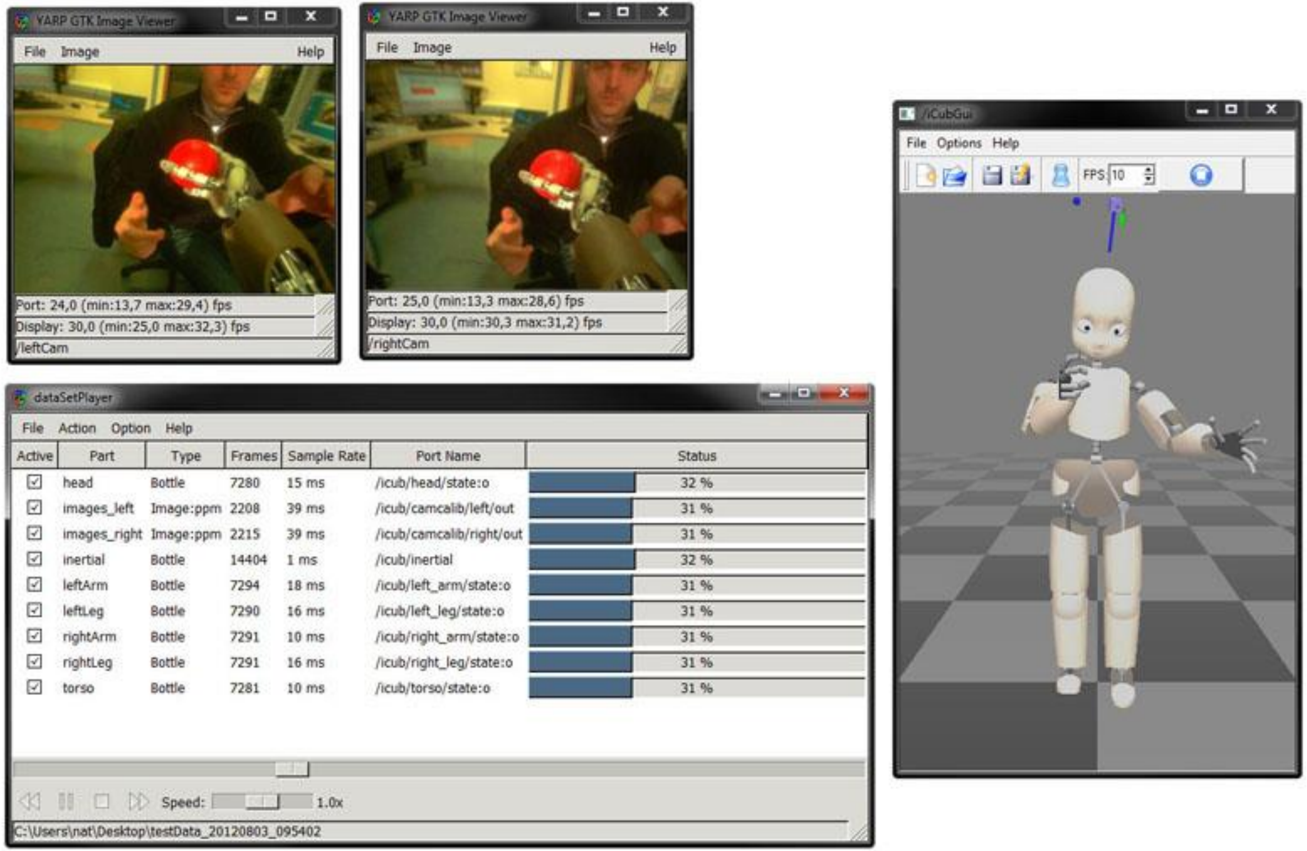


Module	ID	Status	Host	Parameters	Stdio	Work Dir	Environment
☒ iCubGui	0	stopped	localhost	--xpos 1000 --ypos 100 --width 370			YARP_PORT_PREFIX
☒ yarpdataplayer	1	stopped	localhost				YARP_PORT_PREFIX
☒ yarpview	2	stopped	localhost	--name /leftCam --x 0 --y 0 --RefreshTime 33			YARP_PORT_PREFIX
☒ yarpview	3	stopped	localhost	--name /rightCam --x 350 --y 0 --RefreshTime 33			YARP_PORT_PREFIX

Connection	ID	Status	From	To	Car
✗ Internal	0	disconnected	/icub/camcalib/left/out	/leftCam	udp
✗ Internal	1	disconnected	/icub/camcalib/right/out	/rightCam	udp
✗ Internal	2	disconnected	/icub/head/state:o	/iCubGui/head:i	udp
✗ Internal	3	disconnected	/icub/inertial	/iCubGui/inertial:i	udp
✗ Internal	4	disconnected	/icub/left_arm/state:o	/iCubGui/left_arm:i	udp
✗ Internal	5	disconnected	/icub/right_arm/state:o	/iCubGui/right_arm:i	udp
✗ Internal	6	disconnected	/icub/left_leg/state:o	/iCubGui/left_leg:i	udp
✗ Internal	7	disconnected	/icub/right_leg/state:o	/iCubGui/right_leg:i	udp

Resource	ID	Type	Status
localhost	0	computer	unknown

```
[WAR] Module yarp read /reader:i already exists.
[WAR] Module yarp read /reader:i already exists.
```



Port: 24,0 (min:13,7 max:29,4) fps
Display: 30,0 (min:25,0 max:32,3) fps
/leftCam

Port: 25,0 (min:13,3 max:28,6) fps
Display: 30,0 (min:30,3 max:31,2) fps
/rightCam

Active	Part	Type	Frames	Sample Rate	Port Name	Status
☑	head	Bottle	7280	15 ms	/icub/head/state:o	32 %
☑	images_left	Image:ppm	2208	39 ms	/icub/camcalib/left/out	31 %
☑	images_right	Image:ppm	2215	39 ms	/icub/camcalib/right/out	31 %
☑	inertial	Bottle	14404	1 ms	/icub/inertial	32 %
☑	leftArm	Bottle	7294	18 ms	/icub/left_arm/state:o	31 %
☑	leftLeg	Bottle	7290	16 ms	/icub/left_leg/state:o	31 %
☑	rightArm	Bottle	7291	10 ms	/icub/right_arm/state:o	31 %
☑	rightLeg	Bottle	7291	16 ms	/icub/right_leg/state:o	31 %
☑	torso	Bottle	7281	10 ms	/icub/torso/state:o	31 %

C:\Users\nat\Desktop\testData_20120803_095402

Or run the iCub simulator

```
$ iCub_SIM
$ yarpmotorgui
```

Controlling the simulator with the command line

- Set of ports for parts {head} {left_arm} {torso} etc…
- Ports:

/icubSim/head/rpc:i

/icubSim/head/command:i

/icubSim/head/state:o

```
$ 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]
>>
```

YARP from code

YARP Data types: Value

Value is a container able to store in a uniform way a single instance of different basic data types.

Data can be extracted in its native format with *asXXX ()* function.

```
class yarp::os::Value : public Portable
{
   Value(int x);                // Create an integer data.
   Value(double x);             // Create a floating pointdata.
   Value(ConstString &str);     // Create a string data.
   Value(void *data, int len);  // Create a binary data.

   bool isInt();
   bool isDouble();
   bool isString();
   bool isBlob();

   int asInt();               // Get integer value.
   double asDouble();         // Get floating point value.
   ConstString asString();    // Get string value.
   char* asBlob();            // Get binary data value.

   …
}
```

YARP Data types: Bottle

Most flexible type of data.

Can hold variable number of Value.

Bottle can be appended or nested one into another.

Bottle can be accessed using:

- Index, *Size* is the number of element you can *get()*
- *Key-Value* pair, *find(“key”)*

```
Bottle bot;
bot.clear();

bot.addInt(5);
bot.addString("hello");

Bottle& b1 = addList();
b1.addDouble(10.2);


Value &v0 = bot.get(0);
Value &v1 = bot.get(1);

Property &prop = bot.addDict();
prop.put(“pib”, “Help me”);

bot.find(“pib”).asString();
```

ImageOf<PixelType>

```
ImageOf<PixelRgb> yarpImage;
yarpImage.resize(300,200);
PixelRgb  rgb;
rgb = yarpImage.pixel(10, 20);

unsigned char *r=getRow(0);

ImageOf<> is OpenCV compatible!
```

Port

Ports are identified by their name.

Constraints:

- Names must be **unique**
- Names must **start with '/'** character
- No **'@'** character allowed

Ideal for client/server pattern

```
yarp::os::Port myPort;
myPort.open(“/port”);

Bottle b;
port.read(b);
int n = b.get(0).asInt();
n++;
b.clear();
b.addInt(n);
myPort.write(b);

myPort.close();
```

BufferedPort

- Write and Read operations in a Port are blocking
- Buffered ports allow decoupling time:
  - non blocking read
  - non blocking write
- May loose messages!

```
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
 }

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
    // write inside b
    p.write();                // Send the data.
}
```

Buffering policy

- By default BufferedPort drops old messages (Oldest Package Drop)
- You can change buffering policy to FIFO

```
BufferedPort<Bottle> p;
p.open("/in");
p.setStrict(true);   // received messages are queued and never dropped
while (true) {
     Bottle *b = p.read();
 }

BufferedPort<Bottle> p;
p.open("/out");
while (true) {
     Bottle& b = p.prepare();
     // Generate data.
     p.write(true); //wait for previous pending write to complete
}
```

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

```
BufferedPort<Bottle> p;
...
Bottle *b = p.read(false); // returns immediatly

if (b!=NULL) {
  // data received in *b
}
```

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 }
};

MyModule module;
ResourceFinder rf;
//configure resource finder

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

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

perform cleanup, close ports, delete memory

We skip this

Attach callbacks

```
class MyModule::RFModule
{
     Port handlerPort;
      …
     bool configure(ResourceFinder &rf)
     {
          // use rf to configure your module

          handlerPort.open(“/myModule”);
          attach(handlerPort);
       …
     }
     …
}
```

Attach port interface and handle messages

```
// 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

define period in seconds

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

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

this function will be executed until termination