SCHOOL OF MATHEMATICAL AND COMPUTER SCIENCES

An introduction to using yarp and iCub

Getting Started with YARP

YARP stands for Yet Another Robot Platform.

What is it?

If data is the bloodstream of your robot, then YARP is the circulatory system.

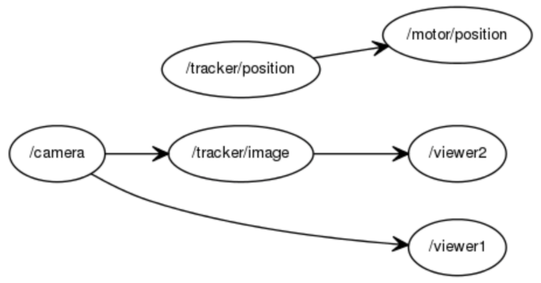
More specifically, YARP supports building a robot control system as a **collection of programs** communicating in a peer-to-peer way, with an extensible **family of connection types** (tcp, udp, multicast, local, MPI, mjpg-over-http, XML/RPC, tcpros, ...) that can be swapped in and out to match your needs. We also support similarly flexible interfacing with hardware devices. Our strategic goal is to increase the longevity of robot software projects.

YARP is *not* :

an operating system for your robot. We figure you already have an operating system, or perhaps several. Nor does it do package management (we like the package managers we have). We're not out for world domination.

A network of ports

Communication in YARP generally follows the ***Observer* design *pattern***. Special *port* objects deliver messages to any number of observers (other ports), in any number of processes, distributed across any number of machines, using any of several underlying communication protocols. Here is a very simple network of ports for a visual tracking application:

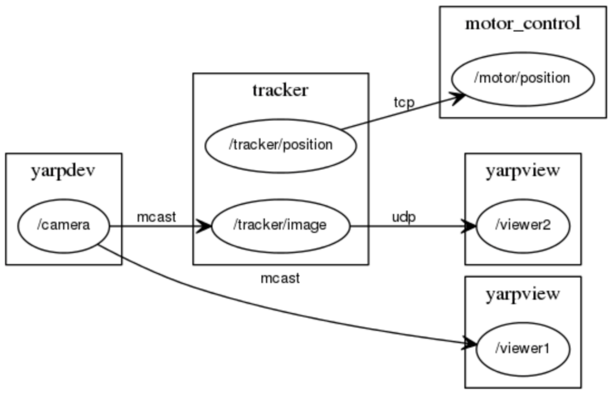


An example of a network of ports

Images are transmitted from a camera ("/camera") port to a viewer ("/viewer1") port and the input of a visual tracker ("/tracker/image"). The tracker annotates the image, for example by placing a marker on a tracked point, and transmits that to another viewer ("/viewer2"). The tracker also sends just the tracked position from a position output port ("/tracker/position") to a input controlling head position ("/motor/position"). Every port belongs to a process. They do not need to belong to the same process. Every connection can take place using a different protocol and/or physical network. The use of several different protocol allows us to exploit their best characteristics:

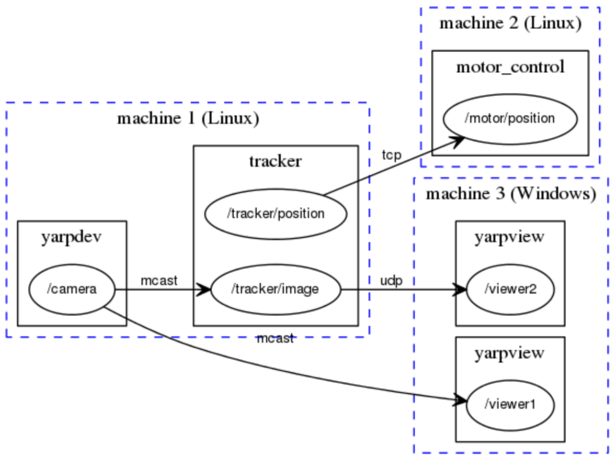
- TCP: reliable, it can be used to guarantee the reception of a message;
- UDP: faster than TCP, but without guarantees;
- multicast: efficient for distributing the same information to large numbers of targets;

shared memory: employed for local connections (selected automatically whenever possible, without the need for programmer intervention); If messages follow YARP guidelines, then they can be automatically converted to and from a "text mode" connection, for easy human monitoring and intervention in the system.



Ports belong to processes, and connections can use different protocols

Processes can be on different physical machines without a problem. They can also run under different operating systems and be implemented using different languages. If the message format follows YARP guidelines, then communication will not be a problem:



Ports can be on different machines and OSes

<u>Exercise 1 Yarp:</u>

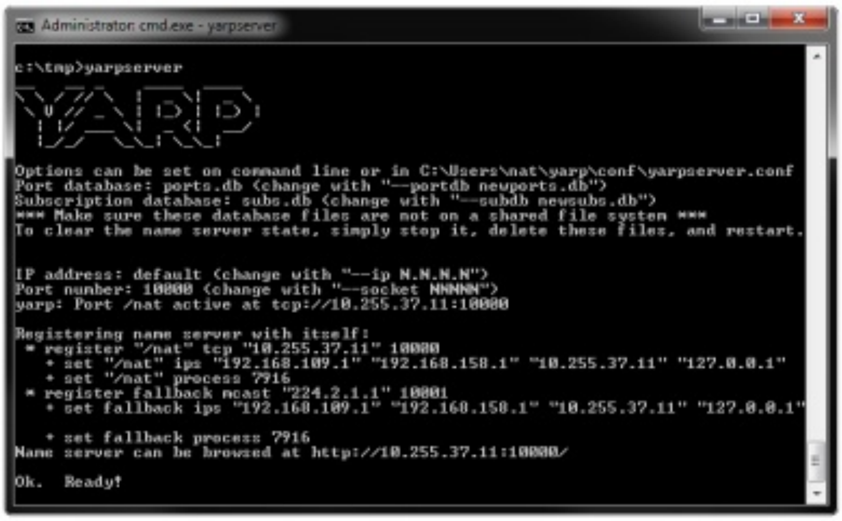
Answer the following questions:

1. What dose YARP stands for?
2. What type of connections is YARP supporting?
3. What is the Observer design pattern?

<u>Exercise 1.1 Use Yarp:</u>
First step is to have a yarpserver running.
At a terminal type:

```
yarpserver
```

Depending on the system the console will look something like this:



```
c:\tmp>yarpserver

Options can be set on command line or in C:\Users\nat\yarp\conf\yarpserver.conf
Port database: ports.db (change with "--portdb newports.db")
Subscription database: subs.db (change with "--subdb newsubs.db")
*** Make sure these database files are not on a shared file system ***
To clear the name server state, simply stop it, delete these files, and restart.

IP address: default (change with "--ip N.N.N.N")
Port number: 10000 (change with "--socket NNNNN")
yarp: Port /nat active at tcp://10.255.37.11:10000

Registering name server with itself:
 * register "/nat" tcp "10.255.37.11" 10000
   + set "/nat" ips "192.168.109.1" "192.168.158.1" "10.255.37.11" "127.0.0.1"
   + set "/nat" process 7916
 * register fallback mcast "224.2.1.1" 10001
   + set fallback ips "192.168.109.1" "192.168.158.1" "10.255.37.11" "127.0.0.1"

   + set fallback process 7916
Name server can be browsed at http://10.255.37.11:10000/

Ok.  Ready!
```

If you type on a web browser **http://127.0.0.1:10000** you get information about the name server (registered ports, info, etc.).

We can just check functionality by running a simple example.
Open another terminal and type:

```
yarp read /portread
```

A third terminal:

```
yarp write /portwrite
```

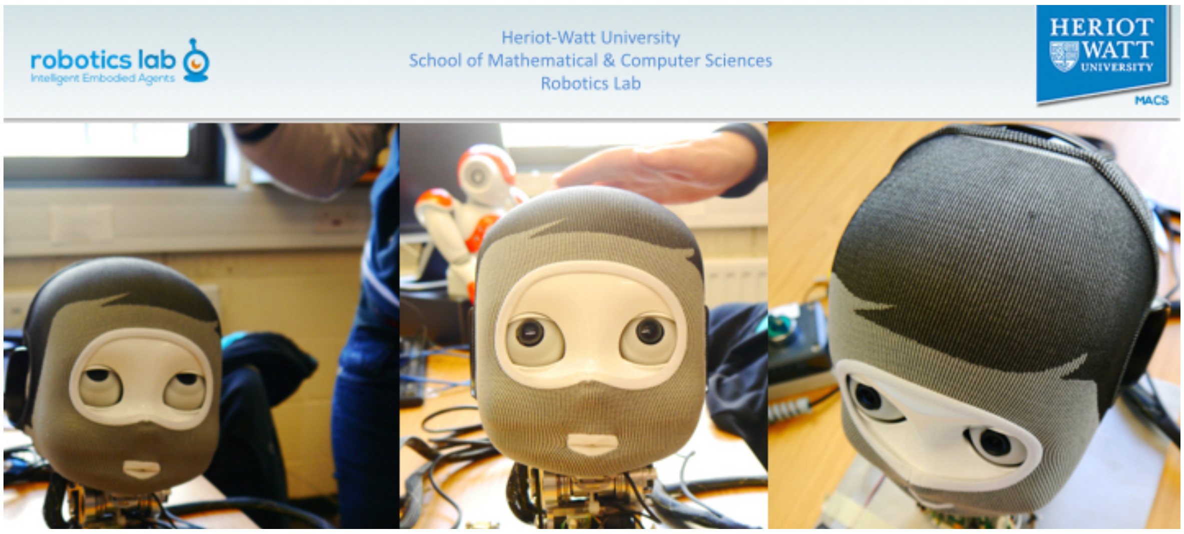
Yet another terminal:

```
yarp connect /portwrite /portread
```

you'll see the effect on the name server:
yarp: registration name /portwrite ip 127.0.0.1 port 10012 type tcp
yarp: registration name /portread ip 127.0.0.1 port 10002 type tcp
Now, anything typed on the yarp write will be sent and printed on the read side.

<u>Getting Started with iCub</u>

The iCub-robotic platform is using YARP as a Middleware to communicated between it's sensors and actuators, as well as to communicate with software components controlling it.
There are several versions of the iCub humanoid robot available. In HWU we have the iCub Talking Head, Nikita.



There are more then 28 iCub's all over the world.
The platform is open hardware and open software, therefore everyone can contributes to it's improvement.

The iCub-software delivers a simulated version of the full body iCub, which we are going to use in the following exercise.

<u>Exercise 1.2 Use iCub Simulator:</u>
First step is to have a yarpserver running (see above).

Open a terminal and type:

```
iCub_SIM
```

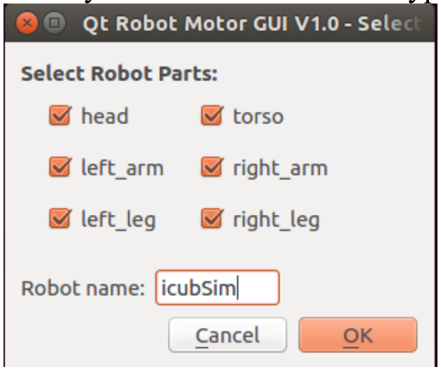
The simulator should open in a new window:

iCub Simulator

Now you can move each joint individually using the *robotMotorGui* tool.
On a different terminal type:

```
yarpmotorgui
```

Modify the name of the robot: type ***icubSim*** in the edit window (see below).

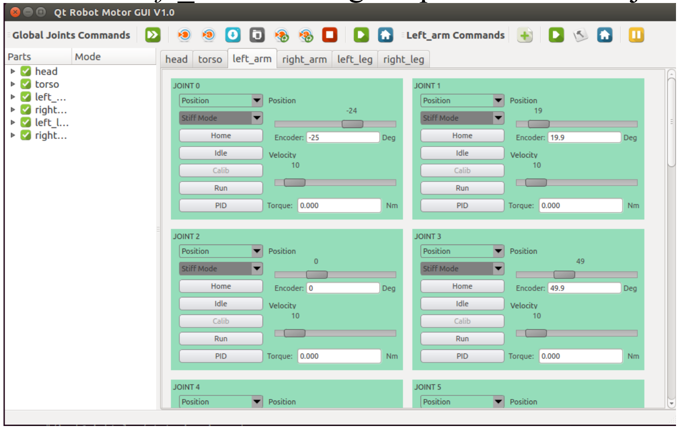


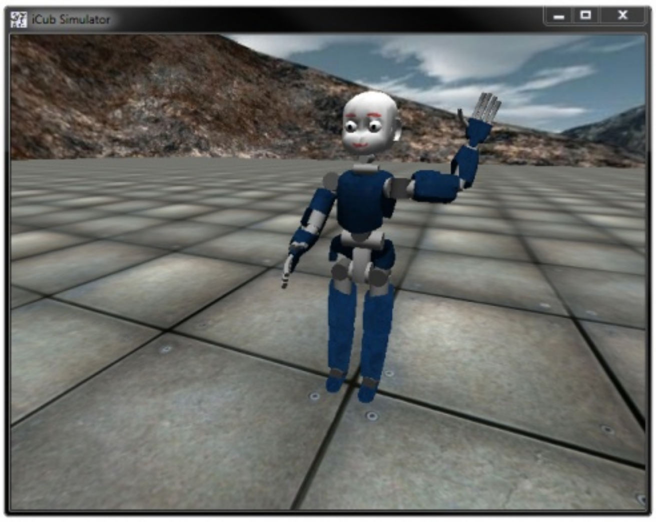
Select Robot Parts:

✓ head ✓ torso
✓ left_arm ✓ right_arm
✓ left_leg ✓ right_leg

Robot name: icubSim

Select the *left_arm* and drag the position of each joint to move the arm:





Exercise 1.3 Use iCub Simulator and simulated cameras:

You can now view the output from the cameras. On different consoles, run two viewers:

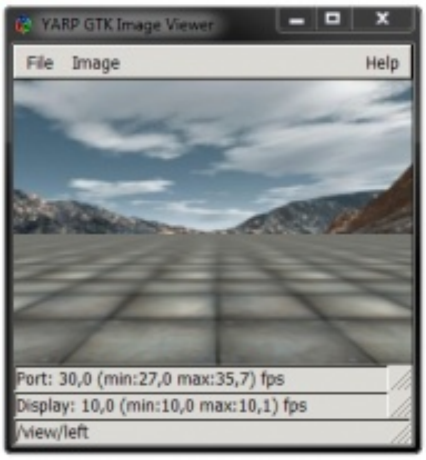
```
yarpview --name /view/left
```

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

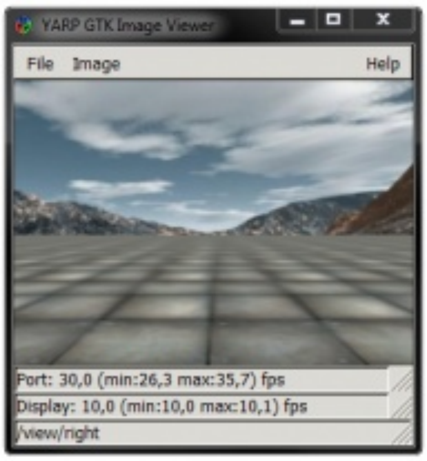
and connect them:

```
yarp connect /icubSim/cam/left /view/left
```

```
yarp connect /icubSim/cam/right /view/right
```



```
Port: 30,0 (min:27,0 max:35,7) fps
Display: 10,0 (min:10,0 max:10,1) fps
/view/left
```



```
Port: 30,0 (min:26,3 max:35,7) fps
Display: 10,0 (min:10,0 max:10,1) fps
/view/right
```