YARP for Windows

Installation manual

Version 1.0

Also derived from:
Configuring YARP for WINNT

Latest version from the CVS repository:
$Id: install.txt,v 1.15 2003/12/16 15:20:49 beltran Exp $

This file is in: %YARP_ROOT%\conf\install\winnt

What is YARP

Hey buddy, I assume you know what you're doing!

Prerequisites

First of all, configure WinCVS (or any other CVS client) as explained in SourceForge. You need to choose a directory (the prospective YARP_ROOT) where to download the whole directory tree. Make sure you check the "prune" flag in CVS so you don't create empty directories.

I use WinCVS 1.3 that solves (apparently) the annoying timezone-daylight saving time problem. Also I used putty (and plink) as a ssh client for WinCVS. I found this configuration very stable and I strongly suggest replicating it if you're not an expert (if that's not the case I wonder why you are thinking on using YARP). If you decide to use putty make sure you download all the small apps that come with the terminal. You will need a few of them in the configuration (for sure, putty, pagent, and plink).

∇ Make sure you don't change the default from the putty terminal application (ssh) because it applies to plink too.

Please, do also the configuration of the public-private key as requested in SourceForge. Make sure your CVS ssh connection is function before moving to the next section. Since we're checking prerequisites also, download ACE 5.3.3 from:

http://www.cs.wustl.edu/~schmidt/ACE.html
It comes as a single zip file. Keep it handy since we're going to use it again.

I've included the .h and DLL for the Intel IPL (optionally used in image processing) in the repository so you no longer need to download and install it separately.

∇ DLLs from Visual Studio 6.0: strangely enough the default win install doesn't have the correct version (e.g. the C terminal library wouldn't normally work. This applies for sure with NT4; I haven't investigated the issue on 2K or XP. If you're running NT4 you might need to install Visual Studio on each machine of the network – only for the DLLs – and perhaps for the remote debugging ability).
Please spend some time reading the SourceForge documentation on the use of CVS and also the WinCVS documentation. You should feel comfortable with the different check-out options and with the WinCVS configuration. It will definitely help later during development.

∇ Directories cannot be removed from the CVS server. Make sure you know what you're doing when adding a new directory to the repository.

Needless to say we assume your machine has the Visual Studio 6.0 installed and properly service packed.

Licensing

YARP is distributed under the **Academic Free License.** The template of the academic free licence can be found in %YARP_ROOT%\conf\licence.template. You can add the template to each file or simply make a link to it.

Downloading components

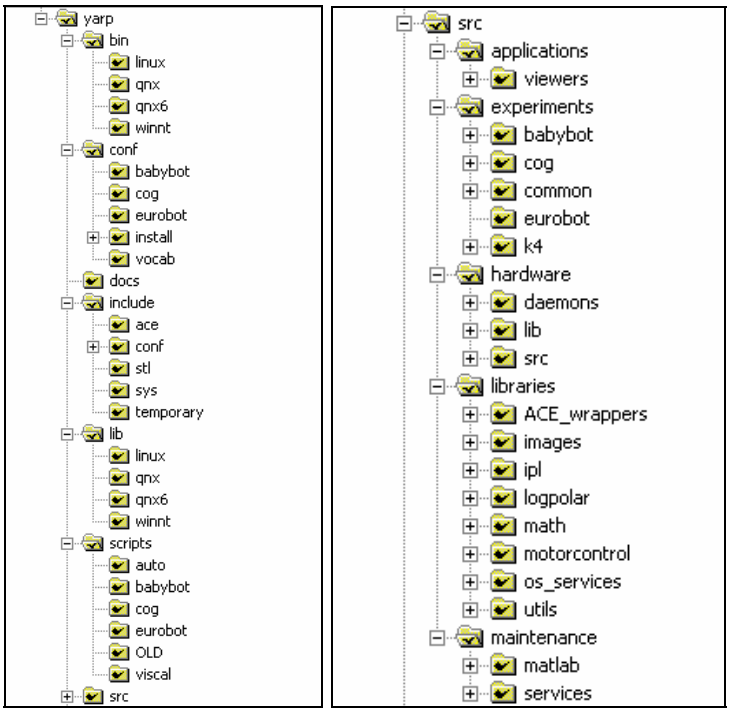
YARP is designed to run on a network of PC's. This short guide first explains how to install a single machine and only later it goes into the details of running on multiple machines. The installation is always one. The "node" PC's are simply running some of the code compiled on the server. And in fact you need to make sure you install (or make available) all device driver files and additional components required for compilation (e.g. .h or .lib files).

For the "network" installation you need also to download the 2K Resource Kit for the additional tools required for Windows (e.g. remote shell and the like), and the WHSRemote scripting. These are all details very specific to Windows. For Linux and QNX different considerations might apply.

Before compiling

First step requires checking out the whole repository from CVS on SourceForge. In the future this whole tree might get packaged into a proper installation. At the moment just manual install. Choose a place where to do the download. This is important since it becomes the root directory of the installation. Run WinCVS now and checkout the module called **yarp**.

If the check out was successful, you should have the following directory tree:



A brief explanation follows.
From the **root** directory (call it **yarp**) you find a **src** directory containing all source code, the usual **bin** for binaries (and DLL), a **conf** directory containing configuration files, a **docs** where documentation is supposed to go, **include** containing all include files after installation, a **lib** directory for static libraries, and a **scripts** directory containing scripts for bringing up your specific system.
In general there are different OS specific directories and robot specific directories. OS directories can be one of the following: **winnt** (for any of the supported MS Windows versions – at the moment of writing NT4, 2K, and XP), **qnx** (for QNX4.25, obsolete, no longer supported), **qnx6** (for QNX version 6.0, 6.1, 6.2), **linux** (only compiled on some RedHat distribution, don't see any plausible issue why it shouldn't compile on any recent Linux distribution). Sometimes there's also a **common** directory with the obvious meaning.
YARP proper libraries are all static; DLL's are only used for IPL and ACE and of course by Windows. Their source code is contained under **libraries** (yarp\src\libraries). Libraries are the **ACE_wrapper** (ACE), **os_services** (communication and OS wrapper code), **math** (simple vector and matrix manipulation library), **images** (for image processing),

logpolar (very specific biologically inspired image manipulation), **motorcontrol** (global library interface to the hardware), and **utils** (miscellaneous).
In addition the interface with the hardware is under **\src\hardware**. You find two components here: i) a tree of directories with specific device drivers (YARP device drivers (called **src** again); they are simple wrappers with a standard interface), and ii) **daemons** (executables to interface to specific robot components).

The latest version of ACE is 5.3.3. The file you should have downloaded from:

http://www.cs.wustl.edu/~schmidt/ACE.html

is called:

ACE-5.3.3.zip

Unzip this file making sure you:

- UNCHECK the option to overwrite the files (this leaves the modified files – as downloaded from CVS – in place).
- UNZIP on top of %YARP_ROOT%\src\libraries\

The UNZIP unzips into ACE_wrappers automatically.

∇ Before continuing:

Define:

- YARP_ROOT env variable: e.g. y:
- add to PATH: %YARP_ROOT%\bin\winnt (location of binaries and DLLs)
- add to PATH: %YARP_ROOT%\scripts (and subdirectories that you need)

Make sure you add to the "system" environment and not to the current user. This is important for correct installation.

Install the IPL by running:

%YARP_ROOT%\src\libraries\ipl\install_ipl.bat

This will copy the .h, lib and dll files in the appropriate directories within the YARP tree.

Now YARP would require a bit of configuration. This is obtained by tuning the appropriate configuration file. A template is provided under:

%YARP_ROOT%\conf\YARPConfigTemplate.h

copy this file to:

%YARP_ROOT%\include\conf\YARPConfig.h

and include the correct config file from within (e.g. Win32 config file):

#include <conf/YARPConfigWin32.h>

So, at the end of this procedure you should have the following file with the correct pointer to the Win32 config file.

%YARP_ROOT%\include\conf\YARPConfig.h

Finally, use the following bat file:

%YARP_ROOT%\src\hardware\src\install_orig_dd.bat

to install all the device driver relevant files. These are lib, headers, and DLL's of the supported hardware devices. This doesn't mean you could then control the devices. There's still the installation of the original device drivers to be done following the hardware manufacturer instructions. These are only the files requested for a successful compilation.

Compile

Now we're ready to start compiling the YARP libraries. Compilation under Windows could be certainly done from within the Visual Studio IDE. Since there are many libraries and certain dependencies we provided a makefile. At the moment of writing what we've produced (Ask Microsoft!) has a potential bug. The only way to know is by actually running the makefile. The issue turns out to be related to the ignominious long/short name of certain DOS compatibility issues. In certain cases the makefile generation includes a system (Visual Studio) header file among the dependencies. Unfortunately the file location tends to change across installations since the short version of the file name is used in the dependency files. The project makefile thus might not compile. If this happens then you have to re-generate the makefiles from within the Visual Studio IDE (when doing it, please make sure you check the flag "generate dependencies"). From within the IDE re-generate all the makefiles for the project called all_libraries.dsw that you find in: %YARP_ROOT%\src\libraries. This project includes pretty much all the libraries and hardware support.

∇ Under "export makefile" select all projects and then check them all, and make sure you also check the write dependencies. Click ok to save makefiles and dependencies.

Once you've got the correct set of makefiles, open a console and CD to:

%YARP_ROOT%\src\libraries\

and type:

make_all_lib.bat debug [or release] full

The first time you might want to issue the **full** flag which starts by compiling ACE. Now, relax and get a coffee, ACE takes a few minutes to compile.

∇ Check continuously the output of the compilation. In case of errors please stop the batch file and try to figure out what might be going wrong. Also, it is useful to have a large buffer into the console window (e.g. 300 or more lines) so that you can scroll it back to the exact point where the problem manifested.

Compilation also installs libraries and copies include files. Eventually you should get:

- The libraries under %YARP_ROOT%\lib\winnt
- Binaries under %YARP_ROOT%\bin\winnt
- Headers under %YARP_ROOT%\include (check also ace, sys, etc.)

The next step is to compile some of the services and install the utilities. Most of these utilities are required to run remote applications (if you're running on a network). Since they're handy, you might want to install them on the server. Also some of the applications internally rely on these utilities.

First of all install the remote scripting host. This is a Microsoft service. You can download it from the Microsoft web site. Also don't rely on the fact it's installed (e.g. somewhere Microsoft says it comes with IE6, but apparently it depends on how you install IE6).

Compiling and installing services

Install remote scripting host, it can be downloaded from:

http://msdn.microsoft.com/scripting

or

%YARP_ROOT%\src\maintenance\scripthost

After installation (and reboot), make sure you give the WSHRemote (a DCOM object) appropriate permissions:

- use **dcomcnfg** to give the **WSHRemote** appropriate permissions
- change the user running remotely to "interactive user" - this is in **dcomcnfg**
- use the enable_remote_scripting.reg to enable remote scripting into the registry

Also, a few of the W2K Resource Kit tools have been used to allow controlling the PC's on the network from the server (the one we're installing now). We wrapped the applications into command line executables following a QNX-like syntax. These tools have been zipped with password and included into:

%YARP_ROOT%\src\maintenance\

The password must be requested to: gmetta@users.sourceforge.net
You should extract the content into the same directories where the zip files are found. Some of the applications have to be installed (together with the corresponding services). Installation has to be done also for the server computer:

- Install **rconsole**
- Install **rkill**
- Extract **shutdown**

Now you can use the make_service.bat script to compile the services:

%YARP_ROOT%\src\maintenance\services\make_services.bat

type: make_services.bat release[debug]; this installs automatically the binaries under %YARP_ROOT%\bin\winnt that should be already in your path variable.

The make_service.bat also compiles the service wrappers (on, slay, etc.), the porter applications (used to connect ports at run-time), copies shutdown (the remote shutdown utility), and the YARP name server.

The **on** command requires the following:
Create an **on.bat** file under %YARP_ROOT%\bin\winnt with the following template:

```
@echo off
_on.exe -u DOMAIN\USERNAME PASSWORD %1 %2 %3 %4 %5 %6 %7 %8 %9
```

replace the domain, user and password fields.

Compiling and installing daemons

The next step is that of compiling the YARP daemons. Depending on your hardware architecture you might require different components and clearly not all of them will actually run on your system. There's no harm in compiling them anyway. To compile the daemons go into:

%YARP_ROOT%\src\hardware\daemons

and use the make_daemons.bat [debug/release]. If you develop/include a new device driver you might want also to develop a new daemon (or modify an existing one). Daemons are executables; they get installed automatically under the bin directory.

Viewers for Windows

There are a couple of general purpose viewer applications: **camview** and **vectviewer**. Clearly this list is meant to increase as more specialized viewers are developed. Viewers interact with the rest of the system through ports and compared to Matlab should sport a much faster interface. Camview is used to display images (and supports also the logpolar remapping), and vectviewer to plot vector data (a graph for each component).
There's no makefile yet for these applications, so you're requested to open the Visual Studio project and manually run the compilation.
Compile the viewers under:

%YARP_ROOT%\src\applications\viewers

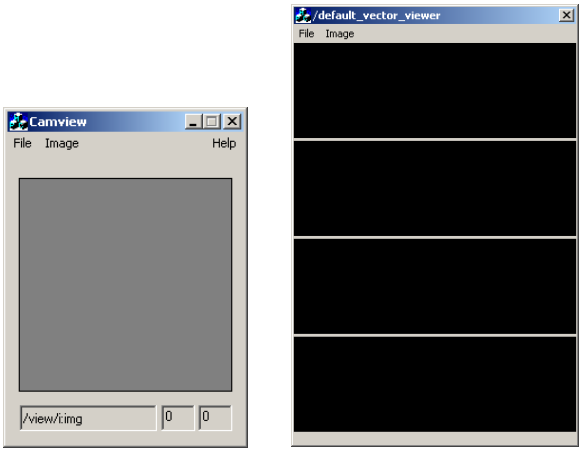


Figure 1: Examples of the viewers

Matlab interface

The interface between YARP and Matlab is accomplished through the MEX interface. This allows creating a DLL that is loaded into Matlab and implements (in C/C++) new commands. The access to ports from Matlab is done by calling the **port(…)** command. The first argument is always the type of command: create, register, etc. and then a variable number of arguments follow. For details on the syntax of the Matlab interface

the best information is either by looking at the code or the handkinematics.m file found in: %YARP_ROOT%\src\experiments\babybot\handkinematics.

Before compiling make sure you have installed Matlab 6.5 (R13) and configured the Matlab MEX support. Open Matlab and type at the command prompt:

mex –setup

and follow the instructions. Further information is found on the Matlab help system. To compile the Matlab interface, open the dsw file directly and compile it. It gets installed under bin. In addition you should add the bin directory to the Matlab search path and save it. The project assumes you installed Matlab on the C drive. If that's not the case, you should consider updating the compiler PATH directories under Tools//Options//Directories both for the include files and libraries.

By using the Matlab interface, any Matlab instance can read/write to ports and in addition script on/slay commands, porter, etc. It becomes a sort of scriptable language for YARP object/ports/processes or simply a way to display data from a running architecture/experiment.

∇ Be careful, blocking calls in YARP cannot be interrupted from within Matlab. If ports are not getting data Matlab would hang. Any other Matlab call is of course interruptible as usual.

Compiling robot specific modules

Robot specific modules are found in:

%YARP_ROOT%\src\experiments\YOURROBOT

and this is also the place where additional subdirectories should be created for any new robot configuration. Since many of the modules are quite general, it might be useful to compile them in spite of the fact you might be developing for something completely different. They are also a good source of examples. Look at the different architectures (only Babybot at the moment). Please note that the **cog** architecture is no longer supported (together with QNX4.25) and directories are maintained only for historical and affective reasons.

Further configuration

Under %YARP_ROOT%\include\conf:

- YARPVocab.h has to be created including the appropriate YARP****Vocab.h file. Depending on your architecture you might require a different set of YARPBottle message types.

Under %YARP_ROOT%\conf:

- See file namer.conf.template: this file configures how the YARP name server sees the network of machines it manages (or where a set of cooperating processes work).

Example:
This is an example of a single machine with a locally running name server.

```
127.0.0.1 10000

// start network description, don't forget to separate "Node=" and
names with space
// Node= name     local nic ip      default nic ip    NetId1 nic ip
      NetId2 nic ip ... NetIdN nic ip
[NETWORK_DESCRIPTION]
Node= hermes      local 127.0.0.1 127.0.0.1     default 5.255.115.211
5.255.115.211     Net0 130.251.4.175 130.251.4.175
[END]
```

Copy the template as namer.conf. This is the file the name server loads at startup. To check the registered names use the utility **nc** (name client).

Description of the directories

This is a (short) description of the various directories of the main installation (extracted from the install.txt file); this information is likely to be obsolete soon!

1. ACE, this copies also the .h, .i, and .cpp files into %YARP_ROOT%\include
    a. use dsw from: %YARP_ROOT%\src\libraries\ACE_wrappers\ace
2. os_services
    a. use dsw from: %YARP_ROOT%\src\libraries\os_services\make_winnt
    b. these two projects do also install the libraries and DLLs
3. compile logpolar library, this is now the actual simulation of one of the CMOS chips. ONLY LogPolarSmallSDK is required to compile images.
4. images
    a. go to $YARP_ROOT\src\libraries\ipl and run script to install include files, DLL's and libs.
    b. compile the fakeipl first. even if you don't use it, just in case.
    c. compile the main img processing lib.
    d. compile the tools.
5. math
    a. dsw in: %YARP_ROOT%\src\libraries\math\math.dsw
6. utils
    a. dsw in: %YARP_ROOT%\src\libraries\utils\utils.dsw
7. hardware
    a. run the install_orig_dd.bat. it copies the libraries+includes in the correct directories.
    b. dsw project in: %YARP_ROOT%\src\hardware\src\alldrivers.dsw
    c. it does the install but...
8. motorcontrol
    a. dsw in: %YARP_ROOT%\src\libraries\motorcontrol\motorcontrol.dsw
    b. compile and install automatically.

Stuff under %YARP_ROOT%/conf

- namer.conf.template: to be compied into namer.conf
- it contains the name and port of the machine running the yarp name service
- ROBOTNAME directories, one for each robot we ever used with yarp (new version)

Network nodes

This is a sequence of steps with useful installation tips for running YARP on additional machines connected to the same network as the server you just installed. A node doesn't require the compilation of all the code. Simply you need to share a disk from the server and make the %YARP_ROOT% on the node to point to the shared drive. You better mount the shared drive appropriately somewhere. I tend to like **Y:** as for **YARP**. Much of these files are under:

%YARP_ROOT%\conf\install\winnt

Step 1: install remote scripting host, it can be downloaded from:

http://msdn.microsoft.com/scripting

As before:

- use dcomcnfg to give the WSHRemote appropriate permissions
- change the user running remotely to "interactive user" - this is in dcomcnfg
- use the enable_remote_scripting.reg to enable remote scripting into the registry
- *** INSTALL from %YARP_ROOT%\src\maintenance\scripthost

Step 2 (optional): Install an **lmhosts** file to facilitate locating the PDC for the NT domain.

- you'll find the example in the install dir: lmhosts.template
- replace the IP, PDC name and domain name
- see also WINNT\system32\drivers\etc

Step 3: Since the node machines are connected to the server (and possibly on a subnet) security can be taken care of by the server (try using an actual Windows NT/2K server). You can apply packet filtering and appropriate routing policies to protect your robot's sub-network. If this is the case then you can plan an automatic logon script on each node so that you're not bothered with logging in on each single node. For what matters a node doesn't even need a mouse/keyboard or display. Keep one handy for debugging though. You can install additionally VNC (http://www.realvnc.com/) to connect remotely and debug. As an alternative if nodes are XP then they have their own native remote desktop (single user) client/server.

There are two steps involved here:

Automatic logon:

- Install autologon.reg key.
- Start from the template provided: automaticLogon.template.reg

- Replace USERNAME, DOMAIN, PASSWD with your specific definitions (the user you'd like to logon automatically at startup).
- It's convenient to define a domain user with the right amount of privileges to install device drivers, start and stop them.

Logon script: e.g. mount net drives etc.

- There's a .bat and a .js file.
- You need to copy the logon.bat from this folder and logon.js in \Winnt\System32\Repl\Import\Scripts\ (what a weird place!).
- In the PDC go to user properties insert logon.bat in the Logon.
- Script name field (this because the USER is actually a domain user).
- The logon.js is a template example (logon.template.js).

Step 4: What you need from the resource kit (resource kit for W2k), and install as for the server.

- rconsole: allows running a remote shell/command
  - Install it using the **rsetup** (that works remotely too)
  - In case of troubles authenticating have a look at \\MACHINE\IPC$, a proper mapping (for a valid user I guess) is required to use the remote tool(s).
  - Try always to use the same user (e.g. the domain one).
  - You might get troubles with credentials depending on the user running it. (damn windows)!
  - Pray!

- Remote kill:
  - Install using rkill /install \\MACHINE.
  - /view to view the process names.
  - /kill to kill a process.
  - *** in case of failure, pls, make sure the Server service is running on \\MACHINE

- rconsole
  - Install using rsetup \\MACHINE
  - It allows opening a remote console.
  - *** make sure the remote registry manipulation service is running on \\MACHINE

- shutdown [installation not required on nodes!]
  - Install, simply copy it to: %YARP_ROOT%\src\maintenance\services\shutdown
  - *** It is then copied automatically during istallation

Step 5: This is exactly as for the server; just define:

- YARP_ROOT env variable: e.g. y:
- Add to PATH: %YARP_ROOT%\bin\winnt (location of binaries and DLLs)

- Add to PATH: %YARP_ROOT%\scripts (and subdirs that you need)

Step 6: unlike the server, node machines might be interfaced to the hardware for real. In that case appropriate device drivers must be installed. It's not just matter of compilation. Install specific device drivers:

- For this you need your original CDs. We don't provide this type of support (of course NOT!)
- Libraries for some of the stuff we're using are included under:
  %YARP_ROOT%\hardware (lib/dll/.h files).
- Libraries are already installed if you've got this far.

Step 7: Disable Windows priority boost for foreground processes.

- Merge noprioritybоost.reg

NOTE for points **3)** and **7)**: after you successfully install and enable remote scripting host you can use mergeRemote.js to include a .reg on a remote machine:

- **example:** mergeRemote remoteMachine noprioritybоost.reg

The scripts looks in %YARP_ROOT%\conf\install\winnt for .regs, remember to define YARP_ROOT on the remote machine before you use the script.

Testing YARP installation

Coming soon!

Further information

YARP is under constant development and it is not mature nor stable yet. No further documentation has been developed apart from the source code itself.