COBOSLAB
Cognitive Bodyspaces: Learning and Behavior

Technical Report No. CoboslabY2011N002
July 12, 2011

Installing the iCub Simulator on Ubuntu

Patrick O. Stalph

COBOSLAB, Psychologie III
Universität Würzburg
Röntgenring 11
97270 Würzburg, Germany

Serena Ivaldi

Institut des Systèmes Intelligents
et de Robotique
Universitè Pierre et Marie Curie
4 place Jussieu, CC 173
75252 Paris cedex 05, France

Installing the iCub Simulator on Ubuntu

Patrick O. Stalph[*] Serena Ivaldi[†]

Abstract

The iCub is an open source platform for research in cognitive and developmental robotics. The humanoid robot is build as a 3.5 year old child. Its mechanical design, as well as its hardware and software specifications, are fully open-source, and released with a GPL license. As the software is a continuously growing research-based project, it is not distributed as a single ready-to-use package, but instead it is based on various other open source projects. Although this is a good thing, it can be very time-consuming to install the iCub simulator and all its dependencies for the first time. This report shall guide through the installation procedure of the iCub simulator on Ubuntu Linux and it was tested on a fresh install of Ubuntu 10.4.

1 Introduction

The guide assumes some basic Linux knowledge, e.g. how to open a terminal, changing directories, the meaning of `sudo` and `apt-get`. Furthermore it is required to manually install some packages, that is, downloading, compiling, and installing manually. All necessary commands are given in this guide, however, it can be troublesome to debug a failed `make` and further expertise is advantageous. Often the main problem is a version conflict: Refactoring in one package leads to compile errors of another one. First, a different version should be tested (not necessarily the latest), if something does not work out of the box.

Root privileges are assumed, as packages are installed system-wide. If root privileges are not available, everything can be installed into the users `$HOME` directory. Finally, a short note about commands within this document. A leading dollar symbol indicates that the subsequent code can be run from within a terminal. Longish commands are broken into several lines with a trailing backslash. This allows to copy&paste a command from the document to a terminal directly. For example, the following two commands have the same effect.

[*]patrick.stalph@psychologie.uni-wuerzburg.de
[†]ivaldi@isir.upmc.fr

```
$ echo Break the line.
$ echo Break \
  the line.
```

This guide is not minimal – some packages might not be necessary for the iCub simulation to work. However, some optional packages allow for quicker computations (e.g. optimized code from alternative packages, multi-threaded computations) or may allow for a broader use of the compiled packages (OpenCV is built with video capturing, decoding, and encoding capabilities).

Two main components are required for the iCub simulation:

- YARP – Yet Another Robot Platform, an open source project for a middleware that mainly focuses on the iCub

  `http://eris.liralab.it/yarp/`

- iCub – the iCub software package itself which includes the simulator

  `http://icub.org/`
  `http://eris.liralab.it/wiki/`

Both require several dependencies. All of them are available in the Ubuntu repositories, but some have to be compiled manually, because the latest versions in the repository is not up-to-date and may not be supported by some iCub modules.

1.1 Compiling with `make` and `cmake`

Often open source projects only provide the source code for download and the user has to compile it manually. Code written in C often comes with a `configure` script and a so called `Makefile`. After the script determined which packages are available on your system and successfully finished without errors, the source code can be compiled with `make`. For large projects this may take quite a while and an immense speedup can be gained by using all cores of the CPU. This is accomplished with `make`'s `-j n` option, where `n` should be the number of available cores, e.g.

```
$ make -j 4
```

for a quadcore CPU. Finally, if a system wide installation is desired, `make install` puts the executables and libraries into the respective system directories. If an installation is not desired, path variables should point to the respective executable and library files.

In order to create platform independent projects, the `cmake` tool is an alternative to the configure script. On Linux, `ccmake` provides a textual user interface that guides through the iterative configuration process and allows to monitor the current options of the project. It is common to create a separate build directory and run `ccmake` from there. The configure step (key: c) has to be repeated until all asterisks are gone. Then, the `Makefile` can be generated (key: g). From now on, it is again the plain `make` and `make install` procedure.

2 Dependencies

The following list briefly outlines the tools and dependencies that can be installed conveniently from Ubuntu's repository.

- build-essential, cmake, and subversion – to checkout and build projects manually
- autoconf, automake, libtool – for ODE
- TBB, avformat, swscale, and python-numpy – for OpenCV
- ACE – communication library for YARP and iCub
- libncurses, GTK and QT (libgtkmm, libglademm, libqt3), Gnu Scientific Library (libgsl), and python-tk – for iCub
- Simple Direct Media Layer (libsdl) and OpenGL (libglut) – for the iCub simulator

The exact package names may vary on your system. For Ubuntu 10.4 the copy-and-paste command to install all of them is as follows.

```
$ sudo apt-get install build-essential cmake \
  cmake-curses-gui subversion autoconf automake libtool \
  libtbb2 libtbb-dev libavformat-dev libswscale-dev \
  python-numpy libace-dev libncurses5-dev libgtkmm-2.4-dev \
  libglademm-2.4-dev libqt3-mt-dev libgsl0-dev python-tk \
  libsdl1.2-dev libglut3 libglut3-dev
```

Two more necessary components are available in the repository, but must be installed on the machine manually for different reasons. One is ODE (Open Dynamics Engine) and the other is OpenCV (Open Source Computer Vision). A detailed description follows.

2.1 Open Dynamics Engine (ODE)

The iCub simulator is based on the Open Dynamics Engine. The latest stable version can be downloaded from

`http://sourceforge.net/projects/opende/`

Extract the file (e.g. into `/opt`) and change to that directory. It is strongly recommended to compile ODE with double precision, thus this option is given to the configure command. Next, the code is compiled and installed.

```
$ ./configure --enable-double-precision
$ make
$ sudo make install
```

More information about ODE can be found at:

`http://ode.org/`

2.2 Open Source Computer Vision (OpenCV)

Although YARP and iCub are fine with OpenCV from Ubuntu's repository, the version is outdated and it is suggested to use the latest release. The latest stable OpenCV release (2.2 or higher) is available at

`http://sourceforge.net/projects/opencvlibrary/`

Extract to a directory of choice (e.g. `/opt`) and change to that directory. Next, create a build folder, and change to the build folder.

```
$ mkdir build; cd build
```

Run `cmake` with the following options.

```
$ ccmake -D CMAKE_BUILD_TYPE=RELEASE -D WITH_TBB=ON ..
```

Optionally, enable the SSE and SSSE support depending on the available instruction set on your CPU. Configure until the asterisks are gone and generate the `Makefile`. Finally, compile and install OpenCV.

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

More information about OpenCV can be found at:

`http://opencv.willowgarage.com/wiki/`

3 Installation of YARP and iCub

We assume an installation into `/opt` and the following structure:

```
/opt
/opt/iCub
/opt/yarp2
```

Typically, the user does not have write access to `/opt` and, consequently, root access is required to create and own the directories. In some cases, it is desirable to install everything in the local user folder, e.g. `/home/user/software`. By changing the installation prefix in the CMake files it is also possible to install all the code (`bin,lib,share,include`) in a specific folder. The following commands have to be modified accordingly, if a different structure is desired (i.e. the installation prefix is different).

```
$ BASE_DIR=/opt           # modify to your needs
$ cd $BASE_DIR
$ sudo mkdir yarp2        # sudo only without write access
$ sudo mkdir icub
$ sudo chown $USER yarp2  # only without write access
$ sudo chown $USER iCub
```

Next, checkout the YARP and iCub source code.

```
$ cd $BASE_DIR
$ svn co https://yarp0.svn.sourceforge.net/\
  svnroot/yarp0/trunk/yarp2
$ svn co https://robotcub.svn.sourceforge.net/\
  svnroot/robotcub/trunk/iCub
```

The remaining directory structure can be created with the following commands.

```
$ cd $BASE_DIR/yarp2
$ mkdir server-conf    # here the yarp configuration resides
$ mkdir build          # here we run cmake and make for yarp
$ cd $BASE_DIR/iCub/main
$ mkdir build          # cmake + make directory for iCub
```

3.1 Path Variables

Path variables can be set at various places in Linux with different effect. A simple way is to put additional variables in the users `.bashrc` file. These variables are only available to one particular user and only within a bash terminal (the standard Ubuntu terminal runs bash). For example, when directly executing a script from desktop or a launcher the variables are not registered. Open the file with your favorite editor, e.g.

```
$ gedit ~/.bashrc
```

and add the following lines (modify accordingly) to the end of the file.

```
export YARP_ROOT=/opt/yarp2
export YARP_DIR=/opt/yarp2/build
export YARP_CONF=/opt/yarp2/server-conf
export ICUB_ROOT=/opt/iCub
export ICUB_DIR=/opt/iCub/main/build
export PATH=$PATH:$YARP_DIR/bin:$ICUB_DIR/bin
```

Save and restart all your terminals to reload the `.bashrc` file.

3.2 YARP

The YARP project is using `cmake` for its configuration. Change to the YARP build directory and run `ccmake` with the following options. Finally, compile and install the project.

```
$ cd $YARP_DIR
$ ccmake -D CMAKE_BUILD_TYPE=Release -D CREATE_GUIS=ON \
  -D CREATE_LIB_MATH=ON -D INSTALL_WITH_RPATH=TRUE $YARP_ROOT
$ make
$ sudo make install
```

To check if YARP is properly installed, type

```
$ yarp
```

It should return "This is the YARP network companion". Then try with

```
$ yarp check
```

It should return something like "YARP seems okay, but there is no name server available". The name server defines the network where YARP ports and modules are connected: By default, it is `/root`. The configuration for the name server – that is, IP and communication port – is stored in `$YARP_CONF/yarp.conf`. The first time you launch the server locally on your PC via

```
$ yarpserver
```

the `localhost` IP with port 10000 are automatically stored in the configuration file.

3.3 iCub

Again, `cmake` is used for the project and the procedure is similar. Apart from the regular install, one more call for the applications is required.

```
$ cd $ICUB_DIR
$ ccmake -D CMAKE_BUILD_TYPE=Release \
  -D ICUB_INSTALL_WITH_RPATH=TRUE $ICUB_ROOT/main
$ make
$ sudo make install
$ sudo make install_applications
```

3.4 Test your Installation

If everything compiled without errors, the iCub simulator should be installed on the system. First, the YARP name server must be running.

```
$ yarpserver
```

The server and corresponding *YARP ports* and *YARP connections* can be monitored from a browser at

`http://localhost:10000/`

In another terminal, the simulation can be started:

```
$ iCub_SIM
```

The robot visualization should come up. In yet another terminal connect to the left arm of the robot and set its position.

```
$ yarp rpc /icubSim/left_arm/rpc:i
> set pos 0 -80
```

The second command is not a shell command, but is run from within the YARP rpc connection (depicted with `>`). The robot should lift its left arm. Now, the iCub simulator is successfully installed. For more information, the interested reader is referred to the YARP and iCub wiki.

`http://eris.liralab.it/yarpdoc/index.html`
`http://eris.liralab.it/iCub/main/dox/html/index.html`