Preprints of the 8th IFAC Symposium on Advanced Control of Chemical Processes
The International Federation of Automatic Control
Furama Riverfront, Singapore, July 10-13, 2012

GNURadio, Scilab, Xcos and COMEDI for Data Acquisition and Control: An Open Source Alternative to LabVIEW

Jagdish Y. Patil [*] **Balashish Dubey** [**]
Kannan M. Moudgalya [***] **Rakesh Peter** [****]

[*] *Systems & Control Engineering, IIT Bombay, Mumbai 400076, patiljagdish154@gmail.com*
[**] *Chemical Engineering, IIT Bombay, Mumbai 400076, balashish@gmail.com*
[***] *Chemical Engineering and Systems & Control Engineering, IIT Bombay, Mumbai 400076, kannan@iitb.ac.in*
[****] *Dept. of Computer Science, Amrita University, Coimbatore, rakesh.peter@gmail.com*

Abstract: This article describes an attempt to create an open source equivalent to the data acquisition and control software LabVIEW. The proposed solution uses GNURadio, OpenCV, Scilab, Xcos and COMEDI on Linux. GNURadio gives a convenient graphical front end. The computations can be carried out using OpenCV or Scilab and Xcos. The device drivers given in COMEDI are used to access real time data. We also present PyGTK based improved graphical user interface for GNURadio.

1. INTRODUCTION

Engineering institutes and colleges in a developing country need a low cost solution to data acquisition and control applications. The only available solutions are proprietary, and are often expensive, at least to the industry. As a result, the students who are trained only on a propriety software find themselves handicapped on joining industry.

As the required information on the drivers of many popular devices are not available, one has to depend on propriety data acquisition and control systems. The Linux project *Control and Measurement Device Interface* [COMEDI, 2012] has drivers to many popular devices and thus helps partially address this problem. Realising the importance of drivers in the public domain, some governments have also started making this a requirement in their procurement policy [Committee, 2011]. Our group has also made attempts to access hardware through open source software [Arora et al., 2010, Moudgalya and Arora, 2010, Arora et al., 2011]. Nevertheless, to the best of our knowledge there have not been many efforts to find an open source equivalent to a complete data acquisition and control system, such as LabVIEW [NI, 2012].

This paper deals with the creation of an open source alternative for LabVIEW, using GNURadio [2012], Scilab [2012], Xcos [Scilab, 2012], OpenCV [2012] and COMEDI [2012], all of which are open source software. While some software, such as Scilab, can run in many platforms, some others, such as COMEDI (Control and Measurement Device Interface), run only on Linux. By the very nature of open source software, it is possible to integrate many other open source software systems also and further increase the capabilities of the solution proposed in this work.

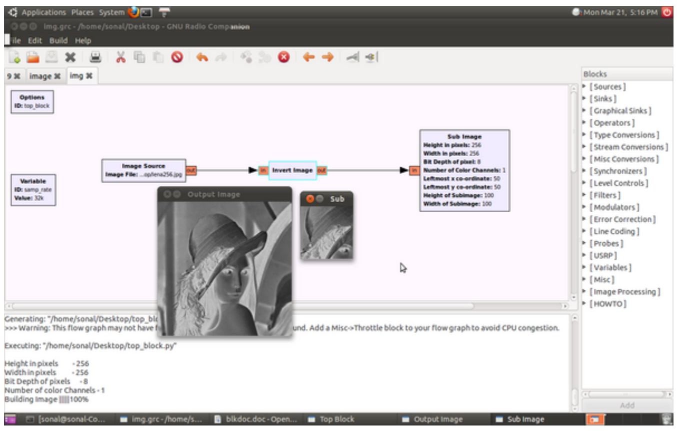
This paper is organised as follows. Section 2 describes the GNURadio in brief. Section 3 talks about how we have integrated the open source libraries and packages into GNURadio. In Section 4, we explain how we interfaced the Single Board Heater System (SBHS) through USB and through Scilab and COMEDI. Section 5 compares the GNURadio solution with LabVIEW and points out the required improvements.

2. GNURADIO

GNURadio is a free software toolkit for learning, building, and deploying software defined radio systems. It is based on C++, Python and its graphical environment resembles LabVIEW. We selected GNURadio in this work because it provides a great flexibility to build GUI (Graphical User Interface). For example, it has knobs and sliders using which, parameter values can be changed at run time, see Fig. 4. Some of these features are more difficult to implement through Scilab or Xcos. GNURadio can be utilised in the development of flexible and scalable, measurement and control applications rapidly and at minimal cost.

An application is created by connecting various blocks. Conceptually, blocks process infinite streams of data flowing from their input ports to their output ports. The attributes of a block include the number of input and output ports they have, as well as the type of data that flow through each one of them. The most frequently used types are short, float and complex. Some important sample blocks such as sources, sinks, simple mathematical operations block, filters block, type conversions block, and variables block come bundled along with GNURadio.

[⋆] This work is partly funded by the National Mission on Education through ICT, http://www.sakshat.ac.in. The early work on GNURadio was done by Ankit, Weissalisa and Sonal.



```
Options
ID: top_block

Variable
ID: samp_rate
Value: 32k

Image Source
Image File: ...op/lena256.jpg

Invert Image

Sub Image
Height in pixels: 256
Width in pixels: 256
Bit Depth of pixel: 8
Number of Color Channels: 1
Leftmost x co-ordinate: 50
Leftmost y co-ordinate: 50
Height of Subimage: 100
Width of Subimage: 100
```

```
Generating: "/home/sonal/Desktop/top_blo
>>> Warning: This flow graph may not have f                         und. Add a Misc->Throttle block to your flow graph to avoid CPU congestion.

Executing: "/home/sonal/Desktop/top_block.py"

Height in pixels       - 256
Width in pixels        - 256
Bit Depth of pixels    - 8
Number of color Channels - 1
Building Image ||||| 100%
```

Fig. 1. Inverting and extracting the input image in GNURadio

We will begin our discussion with blocks that are coded in Python. A unique ID has to be assigned to every block which can be alphanumeric in nature. If two blocks share the same ID, the second one to be generated will overwrite the first.

The graphical user interface of GNURadio is called GNURadio Companion (GRC). Using GRC, one can write GUI for GNURadio. In GRC, applications are represented in terms of flow graphs. The flow graph in Fig. 1 shows the inversion of an image. GRC is written in Python. GRC works only with Python code. GRC can accommodate any GUI development tool written in Python. Python GTK is used to create most of the GUI in GRC [GTK, 2012].

For creating new application we have to connect the appropriate blocks, by clicking which, can set the attributes. The blocks can be placed in the canvas of GRC by simple drag and drop mechanism just like in LabVIEW. After compiling the flow graph, GRC converts it into a python file and stores it in a specified path with a specified name. We can later directly execute this file from the terminal. At the time of execution, all the blocks in the flow graph should be in the python path, meaning that all the blocks should be accessible through an import statement. The scheduler in GRC uses Python built-in module threading to control the start, stop or wait operations of the signal flow graph.

Every block in GRC corresponds to an XML file that describes the block's parameters, inputs, outputs, and other attributes. For each block, there is a separate XML file. GRC has a list of available blocks at the time of start up. In order for a new block written in Python to be in this list, an appropriate XML file should have the above mentioned attributes. In addition, the corresponding Python code should be in an appropriate path.

Next, we will discuss the handling of C++ code, in which most device drivers are written. One should convert the device drivers into Python modules and follow the above mentioned process so that they are available for click, drag and drop operation in GRC. Conversion of C++ code into Python is done through SWIG (Simplified Wrapper and Interface Generator). SWIG generates wrappers around C++ functions and classes, so that they can be called from Python.

3. INTEGRATING SOFTWARE PACKAGES WITH GNURADIO

For online control and signal processing, we need good mathematical tools and the ability to incorporate new algorithms. Scilab and Xcos can meet this requirement. Another important requirement is real time data acquisition, for which we have used COMEDI. An excellent tool for image processing is OpenCV. In this section, we describe a few combinations in which we have used these tools.

3.1 GNURadio with OpenCV

To equip GNURadio with image processing capability, OpenCV (Open Source Computer Vision Library) is used. OpenCV is a library of programming functions mainly aimed at real time computer vision. It focuses mainly on real-time image processing. This library is written in C as well as C++. Image processing is any form of signal processing for which the input is an image, such as a photograph; the output of image processing may be either an image or, a set of characteristics or parameters related to the image. Most image-processing techniques involve treating the image as a two-dimensional signal and applying standard signal-processing techniques to it.

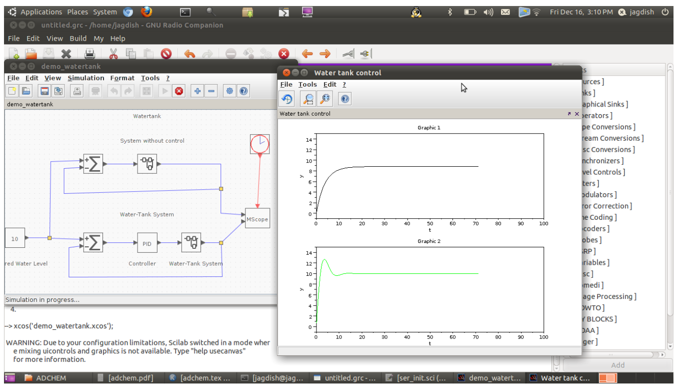
Now we will discuss how to integrate OpenCV with GNURadio. To compile and run OpenCV code in GNURadio, we need to set the path of installed OpenCV in corresponding Makefile of a block in GNURadio. We also need to include the following OpenCV modules in the same Makefile,

(1) cv - Main OpenCV functions.
(2) cvaux - Auxiliary (experimental) OpenCV functions.
(3) cxcore - Data structures and linear algebra support.
(4) highgui - GUI functions.

More details are given in appendix 1.

Fig. 1 shows the inverting and extracting the input image in GNURadio. The application is created by connecting 3 blocks, viz. image source, invert image and sub image. The image source extracts the image from path specified in its block attribute and passes it to the next block. As the name suggests, the invert image block inverts a gray scale image and passes the image pixel values to the sub image block, which extracts the region of interest from an image and displays it in an image window along with the output image. Besides this, we also have added two more blocks i.e. image sink and image file blocks. These display the output image in an image window and create an image file in the specified format, respectively.

OpenCV supported formats are BMP, DIB, JPEG, JPG, JPE, PNG, PBM, PGM, PPM,SR, RAS, TIFF and TIF. OpenCV is widely used in following areas: 2D and 3D feature tool kits, Egomotion estimation, Facial recognition system, Gesture recognition, Human-computer interaction (HCI), Mobile robotics, Motion understanding, Object identification, Segmentation and Recognition, Stereopsis Stereo vision (depth perception from 2 cameras), Structure from motion (SFM) and Motion tracking. We can see that the combination (GNURadio+OpenCV) can be used to acquire data in an image form and analysed.



```
4.

-> xcos('demo_watertank.xcos');

WARNING: Due to your configuration limitations, Scilab switched in a mode wher
   e mixing uicontrols and graphics is not available. Type "help usecanvas"
   for more information.
```

Fig. 2. Demo of executing Scilab and Xcos from GNURadio

3.2 GNURadio with Scilab and Xcos

To improve the functionality of GNURadio, Scilab and Xcos have been integrated with it. Scilab and Xcos can be used for image and video processing, and to carry out mathematical operations. Scilab has toolboxes for control system, communication, system identification, optimization, and many more. Xcos, the open source equivalent of Simulink, is a graphical editor to design hybrid dynamical systems models. Xcos comes with Scilab.

Scilab is included in GNURadio through the call_scilab module. With this module, it is possible to call the Scilab engine from C/C++ code. We also need api_scilab to read/write data from/to Scilab memory. Both of these modules come with Scilab. Details regarding the interfacing of Scilab with GNURadio is given in the appendix. It is also required that Scilab header files such as stack-c.h, call_scilab.h and api_scilab.h are installed in /usr/include/scilab directory.

The most basic functions we need to call Scilab from C/C++ are StartScilab, SendScilabJob, SendScilabJobs and TerminateScilab. StartScilab is used to initialize and start scilab engine in Call Scilab. To send a task from C/C++ code, we use SendScilabJob/SendScilabJobs. These functions are provided in call_scilab. SendScilabJob is used to send a task, while SendScilabJobs can send many tasks for processing by Scilab engine. To stop the Scilab engine in C/C++ code, TerminateScilab is used. Using these functions and others defined in stack-c.h, call_scilab.h and api_scilab.h header files, we have created an application based on Scilab in GNURadio.

The above discussed procedure is to be used to interface Xcos with GNURadio. To execute Xcos through GNURadio, we have used importXcosDiagram and scicos_simulate functions. The importXcosDiagram function imports Xcos files into Scilab. The scicos_simulate function is used to simulate Xcos diagrams in batch mode. It requires the scs_m structure that can be obtained by loading in Scilab the *.cos/*.xcos files. Fig. 2 shows the demo of a PID controller to control liquid level in a tank through GNURadio.

3.3 GNURadio with COMEDI

COMEDI is a free software project that develops drivers, tools, and libraries for various forms of data acquisition: reading and writing of analog signals; reading and writing of digital inputs/outputs; pulse and frequency counting; pulse generation; reading encoders; etc. These drivers work with Linux, and also with Linux combined with the real-time extensions RTAI and RTLinux. The COMEDI core, which ties all the drivers together, allows hardware independent applications to be written. COMEDI has high-level library and integrated real-time support for most hardware. One of the biggest benefits of using a PC-based DAQ device is that one can use software to customize the functionality and visualization of one's measurement system. GNURadio provides an interface to DAQ devices supporting COMEDI. The codes in COMEDI are written in C and Python, making the interfacing with GNURadio easy. The details are in the appendix.

3.4 Graphical User Interface in GNURadio

Good GUIs are critical for user acceptance and subsequent deployment of any software package. User interfaces determine the parameters that a user must enter and display the resultant outputs. While on one hand too many inputs may make the user feel incompetent, too many outputs undermine the importance of the critical information. A good GUI can effectively make the package a favorite amongst the end-users.

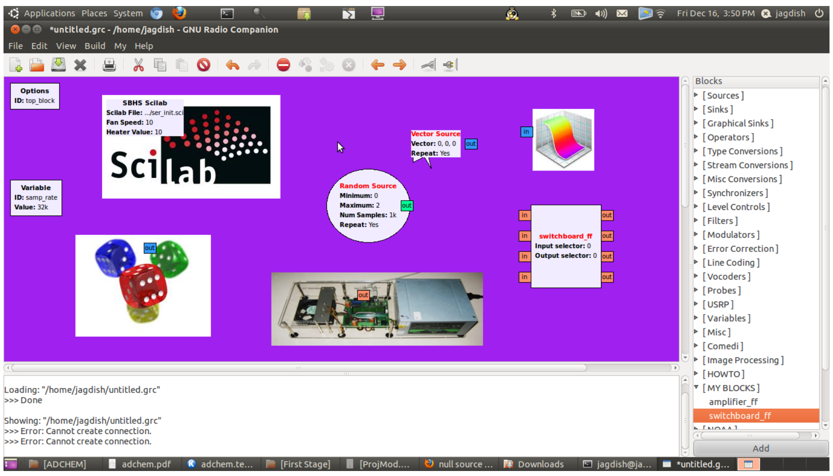
The GUI supplied with the GNURadio package lacks in terms of the simplicity of the design - it supplies the user with a lot of redundant information with a lot of intermediate blocks to process. The default shape of a block is rectangular. The colour of all the blocks are identical. These make the GUIs dry and difficult to use. We explain below how we manipulated the GUI code to suit our objectives better.

The code for the GUI iss python-GTK based. We have modified the same by calling functions from the GTK+ library. We have been able to use images of the actual hardware to replace the standard block supplied in the package. We have also switched off the parameters that were displayed with the block. This helps the end-user appreciate and relate to the experiment better instead of feeling lost trying to make sense of all the blocks displayed. As not all the blocks would have their corresponding physical realisations, we have added a circle and an arc as alternative shapes for source/sinks/intermediate blocks. A colour scheme has also been introduced in the GUI to make it look visually-appealing.

With the above said modifications in place, we have built a GUI capable of supplying our end-users with the appropriate amount of information in a to-the-point manner as shown in Fig. 3. Further modifications of the same can be done on a case to case basis to suit the desired objectives better.

4. DATA ACQUISITION IN GNURADIO

GNURadio can act as a data acquisition system by adding necessary blocks to interact with the data acquisition devices and to set the control parameters. The different ways to interface the SBHS through GNURadio is explained in this section. We also illustrate interfacing a DAQ card with GNURadio.



Options
ID: top_block

Variable
ID: samp_rate
Value: 32k

SBHS Scilab
Scilab File: .../ser_init.sci
Fan Speed: 10
Heater Value: 10

Vector Source
Vector: 0, 0, 0
Repeat: Yes

Random Source
Minimum: 0
Maximum: 2
Num Samples: 1k
Repeat: Yes

switchboard_ff
Input selector: 0
Output selector: 0

```
Loading: "/home/jagdish/untitled.grc"
>>> Done

Showing: "/home/jagdish/untitled.grc"
>>> Error: Cannot create connection.
>>> Error: Cannot create connection.
```

Fig. 3. A typical GNURadio GUI developed in this work

4.1 Interfacing SBHS through USB

The Single Board Heater System (SBHS) is a lab-in-a-box setup suitable for performing numerous control experiments [Arora et al., 2010, Moudgalya and Arora, 2010, Arora et al., 2011]. It is based on ATmega16 microcontroller and it communicates to a PC through a serial interface.

The user can choose the number of data bits, baud rate, stop bits, parity, etc., for serial port communication. The setup is programmed to communicate at 9600 baud rate with 8 bits of data per character frame. The values of 253, 254 and 255 are reserved as command words, respectively to indicate fan speed input, heater power input and the sensed temperature output. The data range 0–252 is used to represent the data.

To access SBHS from GNURadio directly, we have created two blocks namely SBHS source and plot sink. The communication is made through the serial port. The source block is written in C++, while the sink block is written in Python. The source block in GNURadio interfaces with SBHS. It opens the port for communication with SBHS.

To set the fan speed, the source block sends 253 followed by the value taken from the block attribute. This process keeps on executing until one presses stop button on GRC. Because of this, it is possible to alter the input parameters during run time. The inputs can be changed with sliders as shown in Fig. 4. This is the main advantage of using the GNURadio-SBHS interface over the Scilab-SBHS interface. The latter is explained in great detail in Arora et al. [2010]. The source block follows the same process to set the heater value, except that it now sends 254 rather than 253. The source block passes the temperature value to the sink block to display it in a graphical form, in real-time.

4.2 Interfacing SBHS through Scilab and USB

We have also interfaced SBHS through Scilab in GNURadio. Since it is easy to design and implement the control algorithms in Scilab, we propose this as the recommended interface. SBHS is first interfaced with Scilab through serial communication. The serial communication toolbox present in ATOMS module of Scilab is used for communication [Toolbox, 2012]. This toolbox has routines to open the COM port, write to it, read from it, etc. After loading this toolbox, the COM port is opened for Scilab. The fan speed and heater current can be controlled and the temperature of the plate read, as explained earlier.

GNURadio can also access SBHS through Scilab. We have included a block containing the path attribute for this purpose in GNURadio. The path must set to the serial toolbox directory. After executing the flow graph, GNURadio initialises Scilab and loads the serial toolbox in it. In this interface, the code for writing and reading the input-output parameters are partly written in Scilab. The input parameters are set in Xcos file. There is no provision in Xcos to change the input parameters in real time. It is also possible to control inputs from GNURadio directly as we discussed in section 4.1. Fig. 5 shows the execution of Scilab step test code through GNURadio on SBHS.

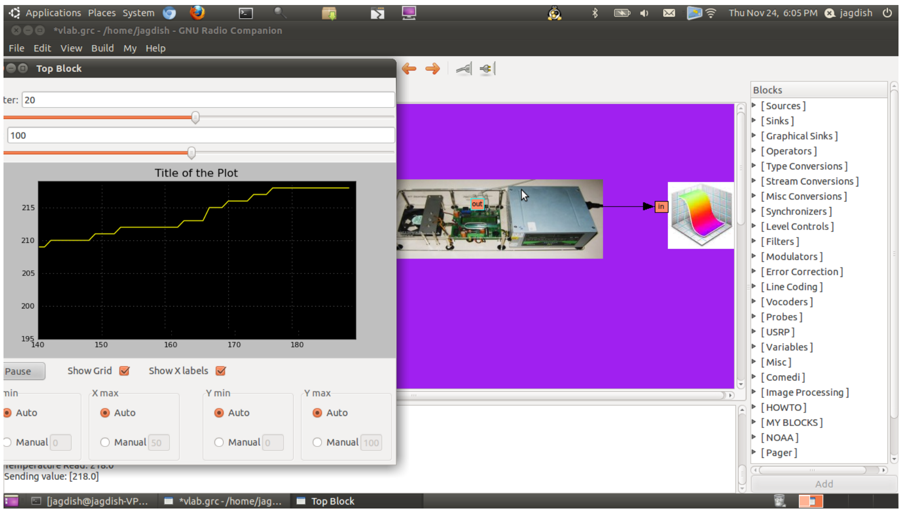
Scilab-SBHS interface is sufficient if we don't need to change the input in run-time. A flaw in Xcos causes it not to be suitable for applications in which inputs need to change during run-time. In GNURadio-SBHS interface, we have overcome this drawback of Xcos. We can set input parameters as variable that can be altered using sliders at any instant.

4.3 Interfacing DAQ Cards through COMEDI

We have discussed GNURadio-COMEDI interface in the previous section. Using a block included in GNURadio, user can access any DAQ card present in the COMEDI list. For this, we need to set the attribute, DAQ card name, as shown in Fig. 6. During run time, this block will load COMEDI library and start accessing the input signal. We interfaced four DAQ cards from the COMEDI list and a locally developed card.

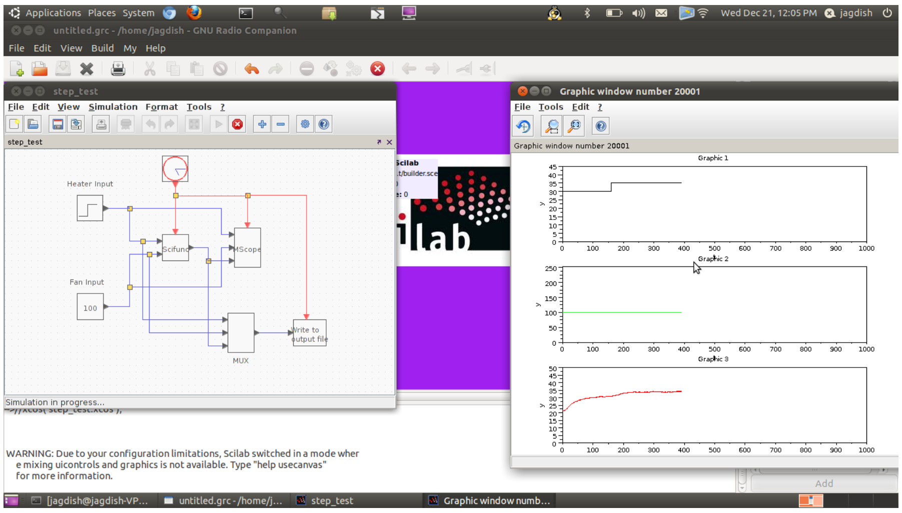
5. COMPARING GNURADIO AND LABVIEW

We have made a survey on LabVIEW and it's functionality, see Table! 1. The first column lists LabVIEW's features and the second lists the capability of the solution proposed in this work. The GNURadio based solution is comparable



```
Sending value: [218.0]
```

Fig. 4. Temperature Plot. It is possible to change the control parameters using the sliders in the left window.



```
Simulation in progress...
WARNING: Due to your configuration limitations, Scilab switched in a mode wher
  e mixing uicontrols and graphics is not available. Type "help usecanvas"
  for more information.
```

Fig. 5. Xcos graph for step change response experiment in GNURadio

Table 1. Comparison of LabVIEW and GNU-Radio on select features

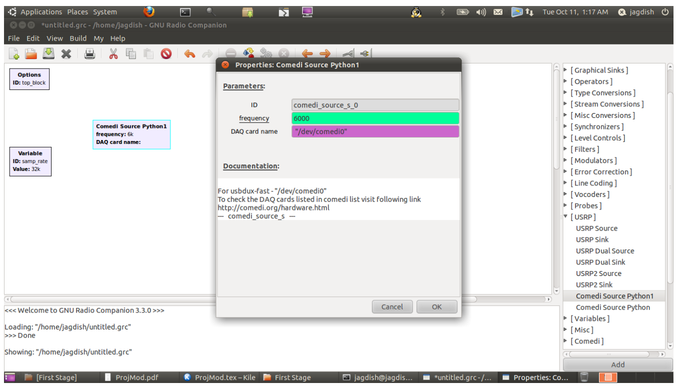
	LabVIEW	GNURadio
1	Buses-MODBus, PROFIBus, FIELDBus, USB, serial poet, devinet, GPIB, VXI, wireless/ethernet communication, NI wifi-DAQ cards, bluethoot, RS-232, plug in boards.	Buses-USB, serial port, bluetooth
2	Signal processing, analysis, mathematical functions	Some of these are present and others can be implemented through Scilab and OpenCV
3	Inbuilt controllers, system identification, control system design	CACSD, a Scilab toolkit, can be used
4	Can develop applications in state chart diagrams	Can be used Hybrid Automata toolbox in Scilab [Najafi and Nikoukhah, 2007]
5	Data acquisition modules present	Use COMEDI and Scilab
6	Virtual Instrumentation	Django based equivalent solution available

to LabVIEW in data acquisition, control, signal conditioning and signal processing.

As the proposed solution is based on a collection of open source software packages, the proposed solution is not as robust. In comparison to LabVIEW, the proposed solution has access to only about 400 DAQ cards. Moreover, the latter runs only on Linux. LabVIEW has a much better GUI and also a lot easier to use.

6. CONCLUSION

The paper introduces a low-cost open source data acquisition system developed as a possible alternative to the proprietary software LabVIEW. The flexibility of this system has been demonstrated through combinations of the open source packages in the field of control, DSP, image processing, etc. As open source software systems, by definition, can work any other similar system. As a result, it is possible to extend the capabilities of the software developed in this work. In contrast, as the propriety software packages are closed, it may not be possible to extend the capabilities and one may have to be at the mercy of the vendors. The solution provided in this work will benefit



Properties: Comedi Source Python1

Parameters:

ID	comedi_source_s_0
frequency	6000
DAQ card name	"/dev/comedi0"

Documentation:

```
For usbdux-fast - "/dev/comedi0"
To check the DAQ cards listed in comedi list visit following link
http://comedi.org/hardware.html
— comedi_source_s —
```

```
<<< Welcome to GNU Radio Companion 3.3.0 >>>

Loading: "/home/jagdish/untitled.grc"
>>> Done

Showing: "/home/jagdish/untitled.grc"
```

Fig. 6. Using USB DAQ Card in GNURadio

the academic institutions and small and medium scale enterprises. It is especially suitable to entrepreneurs and start ups that are strapped for funds, whether they are from developed countries or not. The current focus of work is to make the interfaces robust and to create instructional material to popularise this methodology.

REFERENCES

I. Arora, K.M. Moudgalya, and S.Malewar. A low cost, open source, single board heater system. In *4th IEEE Int. Conf. on E-Learning in Industrial Electronics (ICELIE)*, pages 7–12, Pheonix, AZ, USA, 7-10 November 2010. IEEE.

I. Arora, K.M. Moudgalya, K. Venkata, V. Chakraborty, R. Rokade, and R. Rakhi. A low cost, scalable, virtual control laboratory. In *Proceedings of the 9th IEEE International Conference on Control and Automation (ICCA), 2011*, Santiago, Chile, 19–21 December 2011.

COMEDI, 2012. URL `http://www.comedi.org/`. Seen on 30 April.

Policy Committee. Policy on device drivers for procurement of hardware for e-governance. Technical report, Indian Government, 2011. URL `http://www.public-software.in/node/3414`.

GNURadio, 2012. URL `http://gnuradio.org/redmine/projects/gnuradio/wiki`. Seen on 30 April.

Python GTK, 2012. URL `http://www.pygtk.org/`. Seen on 30 April.

K. M. Moudgalya and I. Arora. A Virtual Laboratory for Distance Education. In *Proceedings of 2nd Int. conf. on Technology for Education, T4E*, IIT Bombay, India, 1–3 July 2010. IEEE.

Masoud Najafi and Ramine Nikoukhah. Implementation of hybrid automata in scicos. *IEEE Int. Conf. on Control Applications*, pages 819–824, 2007.

NI. LabVIEW, Seen on 30 April 2012 2012. URL `http://www.ni.com/labview/`.

OpenCV, 2012. URL `http://opencv.willowgarage.com/wiki/`. Seen on 30 April.

Scilab, 2012. URL `http://www.scilab.org/`. Seen on 30 April.

Serial Toolbox, 2012. URL `http://atoms.scilab.org/toolboxes/serial`. Seen on 30 April.

APPENDIX

For GNURadio + OpenCV - To integrate OpenCV with GNURadio includes two flags AM_CXXFLAGS and AM_CPPFLAGS in the Makefile of GNURadio as follow,

```
AM_CXXFLAGS = -lcxcore -lcv -lhighgui -lcvaux -lml
AM_CPPFLAGS = -I/usr/include/opencv
```

So, while compiling OpenCV based code, GNURadio will look for it's library and modules in above specified path.

For GnuRadio + Scilab - To compile code based on Call Scilab, it is necessary to defines some arguments, variables and path as follows.

(1) CFLAGS - Call Scilab needs to have access to the headers of Scilab core and call Scilab module.
a) In the binary version of Scilab, CFLAGS must be set to /path/to/scilab/include/scilab/core/ and /path/to/scilab/include/scilab/call_scilab/.
b) In the source tree of Scilab, CFLAGS must be set to /path/to/scilab/modules/core/includes/ and /path/to/scilab/modules/call_scilab/includes/.
(2) LD_LIBRARY_PATH - Paths to libscilab.so and libjavasci.so

To compile code based on call scilab used

```
gcc(compiler name) -lscilab -L/path to
/lib/scilab/ -I/path to/include/scilab/fielname.c
```

To run an application based on Call Scilab, there are a few other things to set up. Some global variables must be set:

(1) SCI - Path to Scilab files a) In the binary version of Scilab, SCI will point to /path/to/scilab/share/scilab/ b) In the source tree of Scilab, SCI will point to the root of the source tree /path/to/scilab/source/tree/
(2) LD_LIBRARY_PATH - Paths to libscilab.so and libjavasci.so
(3) LD_LIBRARY_PATH (Java) - Paths to Java native libraries (libjava.so, libjvm.so, libhpi.so). It is usually provided by the operating system or by Scilab distribution.

Series declarations needed to run the code based on Call_Scilab.

```
# With a Scilab source tree:
$ SCI={path to Scilab source tree}

$ export LD_LIBRARY_PATH=$SCI/modules/.libs/:
$SCI/modules/call_scilab/.libs/:
$SCI/modules/api_scilab/.libs

# With a Scilab binary:
$ SCI= {path to Scilab binary}
$ export LD_LIBRARY_PATH=$SCI/lib/scilab/

# Don't forget to update i386 to whatever is necessary
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:
/usr/lib/jvm/java-6-openjdk/jre/lib/i386/:
/usr/lib/jvm/java-6-openjdk//jre/lib/i386/server/

$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:
/usr/lib/jvm/java-6-openjdk/jre/lib/i386/native_threads/
$ export SCI=/path/to/scilab/
```

Add above declarations in Ubuntu's bashrc file.

For GNURadio + COMEDI - To make use of the data acquisition driver in GNURadio set the following path and variable.

(1) Set AM_CXXFLAGS as
-lcomedi -lm -L/path to comedi/lib -I/path to comedi/include
in Makefile of corresponding block for COMEDI in GNURadio.
(2) Add following variables in bashrc file of Ubuntu. SCI={path to comedi}
export LD_LIBRARY_PATH=$SCI/libcomedi.a
export LD_LIBRARY_PATH=$SCI/libcomedi.la
export LD_LIBRARY_PATH=$SCI/libcomedi.so
export LD_LIBRARY_PATH=$SCI/libcomedi.so.0.8.1