# A remote lab implementation using SAHARA

N.M. Wareing, P.J. Bones, S.J. Weddell

**Abstract**—A remote laboratory system has been designed to allow students to test microcontroller code on model helicopters and view its performance without physically being in the laboratory. They access a web based system, running open-source *SAHARA* software and are able to upload a compiled program onto a *Stellaris LM3S1968 microcontroller*. Students view the helicopter on a live video feed and have virtual button presses sent to pre-defined pins of the microcontroller, for their program to respond to.

**Index Terms**—Remote Labs, SAHARA, Micro-controller, Embedded systems.

## 1 Introduction

The central premise of remote labs is that universities around the world can install and host physical experiments which can be accessed by students over the Internet [1][2][3][4]. The idea holds great promise because it can enhance collaboration between universities (sharing of expensive lab resources between universities)[5], and moreover, allow students to work from wherever they are, at whatever time of day suits. They can thus help flatten the demand for expensive lab resources and software is able to impose limits on the safe use of equipment.

As public access is not required to a remote laboratory, they are more secure, reducing problems associated with theft, vandalism and damage to equipment through inappropriate use. Student safety is also improved through physical isolation from hazardous equipment. These factors may have economic benefits in the form of reduced (occupational health and safety (OSH) compliance costs and reduced insurance premiums.

Until recently, the technological challenges involved in real-time Internet communication have limited the growth of remote laboratories. However, recent improvements and innovations in web technologies have rapidly begun to remove these impediments. The University of Technology, Sydney (UTS)[1] has already implemented a fully featured remote laboratory system.

In 2012 the Electrical and Computer Engineering department at Canterbury University offered a course, ENCE361[2], titled Embedded Systems. One of the projects, 'Fun with Avionics', required students to write an embedded program to control the flight of a small model helicopter. The helicopter was fixed in a custom built stand which allows it to rise, fall, and turn a range of motion (see Fig. 1). Students purchased a *Stellaris* development board with a *Cortex-M3* microcontroller, which included a development platform for compilation, programming, and debugging. The helicopter stand output a voltage proportional to the height of the helicopter. Students were required to control the rotor using pulse width modulation (PWM) signals.

This assignment has proved to be both challenging and interesting for students. It provides a platform to apply embedded programming concepts such as interrupts, implementing proportional, integral, derivative (PID) control, and creating a small real time operation system (RTOS). Due to a limited number of helicopters, and associated challenges, unfortunately all students were not able to gain equal access to the laboratory equipment. The remainder of this report discusses how the 'Fun with Avionics' assignment could be converted to a remote laboratory format to help solve some of these issues.

1. http://www.uts.edu.au/

2. http://www.canterbury.ac.nz/courses/

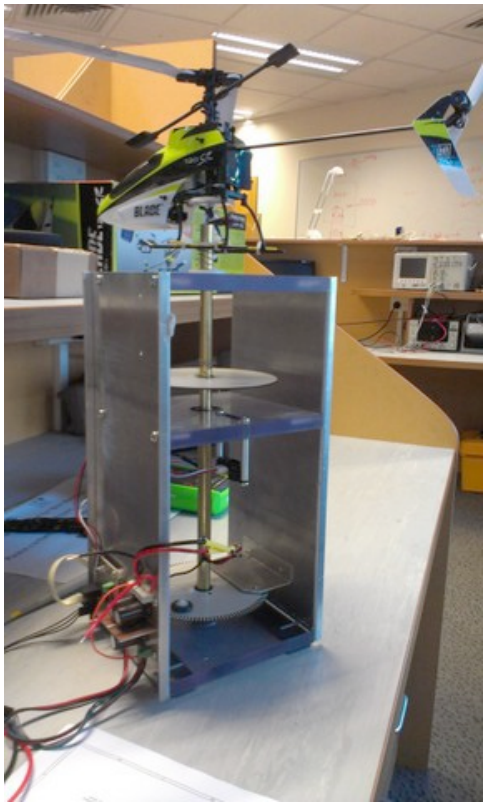

Fig. 1. The helicopter, and stand.

## 2 Early Modification Decisions

One of the difficulties students encountered in writing and testing programs was not breaking the helicopter when the control system code was still in the testing phase! This often involved physically holding the helicopter to stop it swinging wildly from side to side; obviously this would not be possible with the student in a remote location.

We therefore modified the helicopter stand to allow only vertical motion. The tail-rotor was also disconnected. The only real possibility of damaging the helicopter now is if it is run at full power for an extended period of time. This will be protected by the lab server monitoring the length of time the helicopter is at 100% (or near to) altitude.

## 3 System Design

A number of software tools have been created to assist with the development of remote laboratories. MIT's iLab[6] and Lab2Go[7] are two examples. We have chosen the open source *SAHARA Labs* framework[3]. This is a generic platform for designing customised remote labs. It is developed and maintained by a team at the University of Technology Sydney. *SAHARA* consists of three separate components:

1) Web Interface
2) Scheduling Server
3) Rig Client

The interactions between these components in the context of the heli lab is detailed in Fig. 2.

### 3.1 Web Interface

The web interface is the platform presented to students to interact with the physical lab. It is coded in *PHP* (using the *Zend* Framework), along with *HTML/CSS/JavaScript*. It allows students to authenticate with the system using either a *MySQL* database or the university's *LDAP* servers. Queueing and reservation functions are also supported. The *SAHARA* library contains a number of 'elements' which can be placed on a customised *HTML* page to support a newly designed remote laboratory rig.

The main components of the customised heli web interface (see Fig. 3 are the following:

1) A counter showing the remaining time allotted to the student.
2) A 'Program Heli' button which launches a file browser to allow students to pick their compiled program from their computer.
3) A demo button which will upload some model code to the microcontroller which the student can then control. This gives students confidence that the system is working correctly and that it is their code that is at fault!
4) A live video stream, with an option to pick from a number of video formats.
5) Up, down, select and reset virtual buttons to be sent to the GPIO (general purpose input/output) pins of the microcontroller.

### 3.2 Scheduling Server

The scheduling server forms the core of *SAHARA*. It is written in *Java* and needs to connect

3. http://sourceforge.net/projects/labshare-sahara/

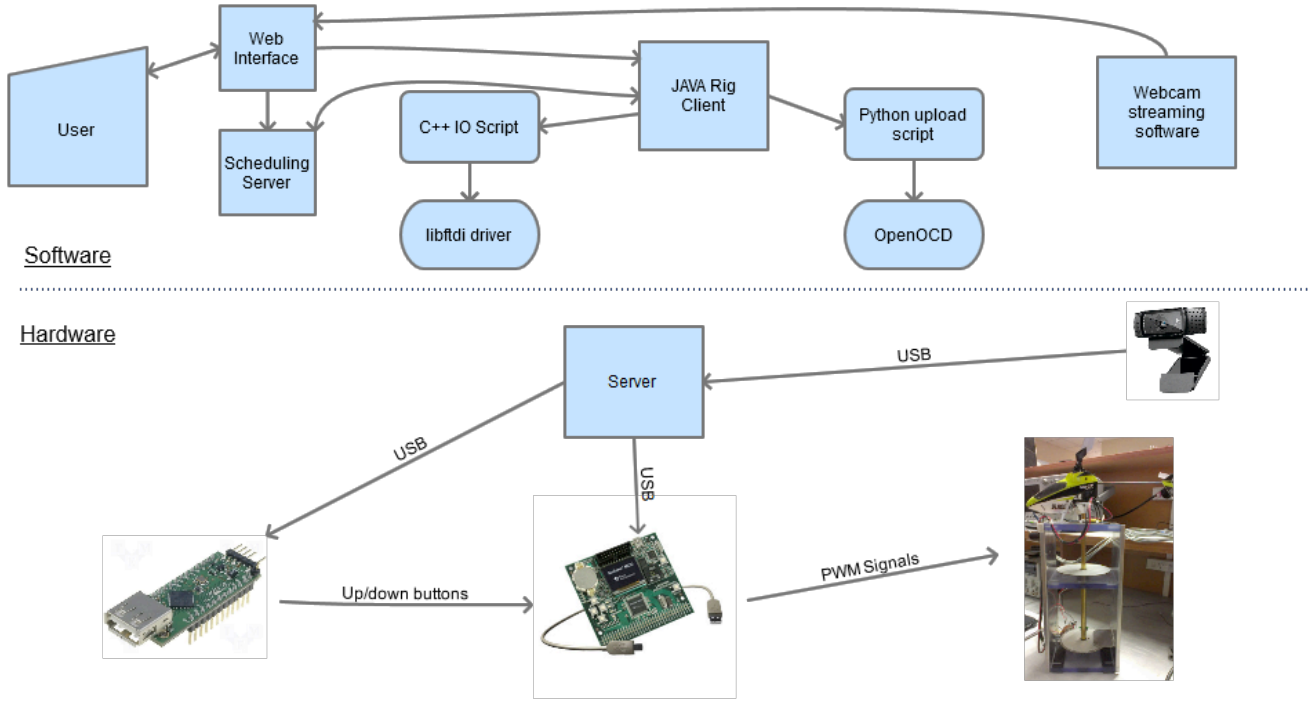


Fig. 2. Interactions between various software components, and hardware.

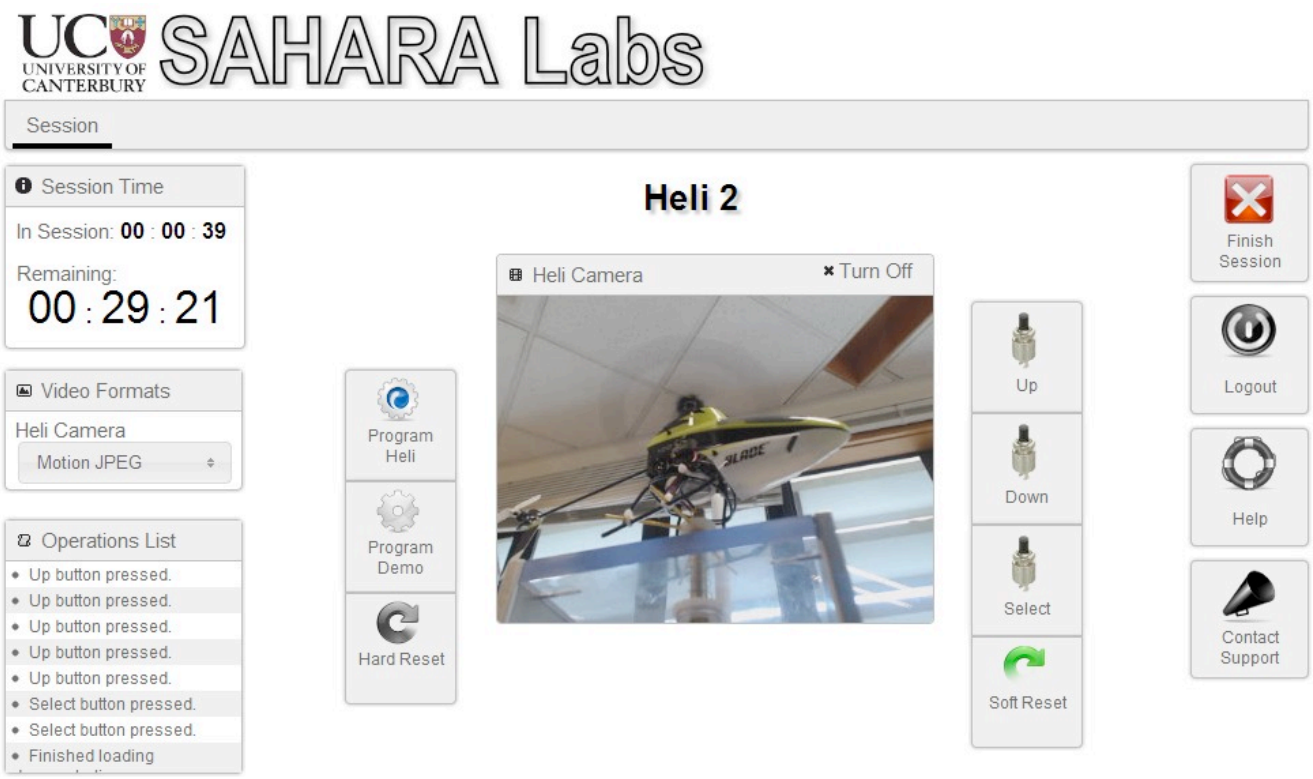


Fig. 3. The web interface in action

to a *MySQL* database. The scheduling server provides a web interface which lab technicians and lecturers can use to allocate access to specific rigs or entire classes of rigs. For the purposes of the heli remote lab, it was sufficient to simply install the scheduling server without any extra customisation.

### 3.3 Rig Client

Three Rig Client control types are provided by *SAHARA* [8]:

1) Peripheral Control: A rig has a controller that is 'outside' of *SAHARA*.
2) Primitive Control: The rig is controlled directly by the Rig Client.
3) Batch Control: The rig operates using batch instructions uploaded by the user. No other intervention is required.

For our purposes, the primitive control rig is most suitable, because ongoing user interaction is required in the form of button presses. Conveniently, this is the same control type used by the 'Nexys' FPGA (field programmable gate array) rig, already developed by UTS[4]. The 'Nexys' rig is a similar concept to ours in that it involved the uploading of student compiled code to a remote target device.

Additionally, it means that the *SAHARA* web interface is used rather than a Remote Desktop Session - this is both a more user friendly solution and offers additional security because no further ports (other than 80) need to be opened on the lab server.

The Rig Client is coded in *Java*, and utilises several third party libraries. In order to design a customised rig, a developer creates a new *Java* class which extends the *AbstractControlledRig* class. The Rig Client program loads in this class at runtime (after specifying its presence in a configuration file). The *HeliRig* class has four main functions:

1) Facility to take the compiled student program from its temporary upload directory (or the demo program) and flash it onto the *Stellaris Microcontroller*.
2) Facility to channel virtual button presses to specific pins of a USB to serial device.
3) Reset functions which the Rig Client runs after a student finishes using the rig, to ensure it is ready for the next user. For example, the students program on the microcontroller is replaced with one which puts the microcontroller in a low power state whilst not in use.
4) Test functions run periodically by the Rig Client to determine if the rig remains in a working order.

#### *3.3.1 Low Level Scripts*

The next step was to investigate how GPIO signals would be sent to the *Stellaris* board. A *FTDI FT245R USB to Parallel-FIFO chip*[5] was chosen for this purpose. A number of different options were considered, including an *Arduino* board[6] and a purpose built USB to GPIO module but the *FTDI* had the best combination of features and price. It meets all our requirements with the following features [9]:

1) Complete USB device mode protocol handled on-chip without custom programming.
2) Complete USB hardware on-chip, including USB resistors.
3) 8 GPIOs available - will allow for future expansion.
4) USB suspend and resume support, to switch device to low-power mode when not in use.
5) Integrated level converter and 5 V, 3.3 V, 2.8 V and 1.8 V totem-pole output, so it can talk to most standard microcontrollers.
6) No additional crystal or oscillator required.

In order to communicate with the *FTDI* chip, the open source *libftdi-dev*[7] library was chosen and installed on the machine. A simple breadboard circuit was constructed where a LED was connected to one of the GPIO pins of the *FTDI* chip, as shown in Fig. 4. A simple program was written and run to test that:

1) The toolchain and *libftdi* library worked.
2) The circuit was feasible.

4. http://www.labshare.edu.au/catalogue/rigtype

5. http://www.ftdichip.com/Products/ICs/FT245R.htm
6. http://www.arduino.cc/
7. http://www.intra2net.com/en/developer/libftdi/repository.php

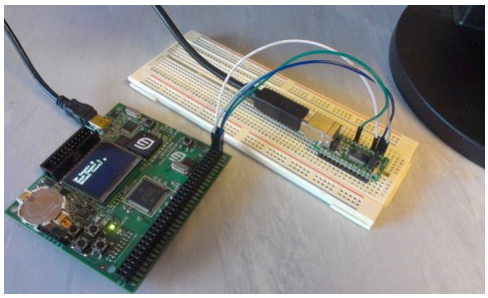

Fig. 4. The GPIO signals are sent to the Stellaris microcontroller using an FTDI device.

To program the *Stellaris*, we have chosen to use *OpenOCD*[8], one of the most popular open-source in-system programmers. Once installed, it is run from the command line with the location of the configuration file given as an argument. It then verifies the *JTAG* scan chain specified, and if successful, starts as a daemon, waiting for connections from telnet clients or the *GDB* debugger (amongst others). The most recent versions have come with configuration scripts for the *Stellaris EKS-LM3S1968*, so it is trivial to get working.

The UTS team opted to use a shell script for their bitstream upload script, however we have chosen to use a simple *Python* program. This is mainly due to our preference for Python, along with the fact that the language has evolved such that there are libraries to accomplish most system level tasks. In this case we are using the *pexpect* module (short for *Python* Expect) [9] to communicate with the *OpenOCD* telnet session. *Expect* is an extension to the *Tcl* scripting language, allowing users to easily create scripts which talk to interactive applications. Here, it is used to start a *Telnet* session to *OpenOCD*, reset the *Stellaris Cortex-M3* and flash the '.bin' file to the microcontroller. Additionally, the *Python argparse* library is used to process the command line arguments sent from the *Java* Rig Client instance. The script has been successfully verified and tested in isolation.

## 4 Video Streaming

One important point to note is that the video streaming and Rig Client have no connection with each other. One must simply ensure that the correct video formats are being continuously streamed, using any suitable software.

We chose to use the *Logitech C920 Webcam* for video capture, and *ffmpeg*[10] as our streaming and transcoding solution. *ffmpeg* streaming can be initiated from the command line. For example, the following invocation starts streaming in the formats described by *ffserver.conf*.

```
ffserver -f ffserver.conf & ffmpeg
    -v quiet -r 5 -s 320x240 -f
    video4linux2 -i /dev/video0 http
    ://localhost:7070/feed1.ffm
```

## 5 Discussion

Latency is one issue that had the potential to cause problems with this project. We have tested the rig (hosted at the University of Canterbury in Christchurch, New Zealand) from Canberra, Australia and have found the delay in the video to be acceptable, at approximately one second.

At the present point in time, the only feedback to the student in regards to the performance of their program is the audio visual webcam stream. This has a number of issues [10]. Firstly, it does not give students any debugging information in terms of program execution. Secondly, and perhaps of greater concern is that the onus is on the student to slow the speed of the helicopter or reset the microcontoller if their program is poorly functioning. For example, due to a poor PID control implementation, the helicopter may run at 100% of its altitude range for an extended period of time. This risks burning out the motor. Considering that one of the main requirements of this remote lab is that it be robust to poorly performing student programs, this is unacceptable. For this reason, a number of extra features are due to be added in future design iterations.

One such feature would involve the Rig Client interpreting the analogue voltage coming from the helicopter stand which is linearly proportional to the helicopters height. If this is read in, perhaps through an Arduino device with an analog to digital converter (ADC), the

8. http://openocd.sourceforge.net/
9. http://www.noah.org/python/pexpect/
10. http://www.ffmpeg.org/

Rig Client could send a signal to a relay circuit to have the power to the helicopter cut, and the student notified that their program need improvement [11]! This could also be implemented with a programmable logic controller (PLC). A second feature being considered is to use the second serial port of the *Stellaris* device to send debug information back to the web interface in a text stream to aid students in the development of their C programs.

At present, the web interface and scheduling server are hosted on an average desktop computer. When the system is fully developed, these services are likely to be moved to virtual machines (VMs) on the department's existing server infrastructure. Using VMs can provide a number of advantages including live migration, reduced electricity usage and better distribution of computational load [12].

## 6 Conclusion

We have presented a new application of remote laboratories, aimed at undergraduate Electrical Engineering students. Utilising the *SAHARA* remote laboratory framework, a complete system has been developed, allowing students to upload compiled code and monitor its ability to control the virtual flight of a small model helicopter. At present, one rig client is in operation (one helicopter). The next stage of the project will add more clients, along with more advanced forms of feedback to help students debug their programs. We are confident that this remote laboratory implementation will be successful and are excited about the experience it will provide to future students.

## Acknowledgments

We would like to thank Michel de la Villefromoy and Michael Diponio from the University of Technology Sydney for their work developing the *SAHARA* software. Their encouragement of our use of the software and time spent corresponding has been invaluable.

## References

[1] D. Ursutiu, C. Samoila, and M. Dabacan, "Cross platform methods in digital electronics engineering education," in *Remote Engineering and Virtual Instrumentation (REV), 2013 10th International Conference on*, 2013, pp. 1–4.

[2] C. F. Riman, A. El Hajj, and I. Mougharbel, "A remote lab experiments improved model," *International Journal of Online Engineering (iJOE)*, vol. 7, no. 1, pp. 37–39, 2011.

[3] K. Jona, R. Roque, J. Skolnik, D. Uttal, and D. Rapp, "Are remote labs worth the cost? insights from a study of student perceptions of remote labs," *International Journal of Online Engineering (iJOE)*, vol. 7, no. 2, pp. 48–53, 2011.

[4] B. Aktan, C. Bohus, L. Crowl, and M. Shor, "Distance learning applied to control engineering laboratories," *Education, IEEE Transactions on*, vol. 39, no. 3, pp. 320–326, 1996.

[5] P. Ordua, L. Rodriguez-Gil, D. Lpez-de Ipia, and J. Garcia-Zubia, "Sharing remote labs: A case study." *International Journal of Online Engineering*, vol. 9, pp. 26 – 27, 2013.

[6] M. I. of Technology, "ilabs: Internet access to real labs," 2009. [Online]. Available: http://icampus.mit.edu/projects/ilabs/

[7] D. Zutin, M. Auer, C. Maier, and M. Niederstatter, "Lab2go: A repository to locate educational online laboratories," in *Education Engineering (EDUCON), 2010 IEEE*, 2010, pp. 1741–1746.

[8] "Sahara Development Handbook Version 1.0."

[9] "FT245R USB FIFO IC datasheet version 2.12," 2010. [Online]. Available: http://www.ftdichip.com/Products/ICs/FT245R.htm

[10] A. Cardoso, M. Vieira, and P. Gil, "A remote and virtual lab with experiments for secondary education, engineering and lifelong learning courses," *International Journal of Online Engineering (iJOE)*, vol. 8, no. S2, pp. 49–54, 2012.

[11] D. A. Samuelsen and O. H. Graven, "Assessment of the quality of low cost data acquisition equipment for remote lab setups," in *Remote Engineering and Virtual Instrumentation (REV), 2013 10th International Conference on*. IEEE, 2013, pp. 1–6.

[12] V. Lasky, "Implementing resilient remote laboratories with server virtualization and live migration," in *Remote Engineering and Virtual Instrumentation (REV), 2013 10th International Conference on*. IEEE, 2013, pp. 1–6.