Development of an open and scalable platform for data acquisition on a low speed wind tunnel

Filipe Roque

October 2012

Abstract

This work studied and documented the original data acquisition system in a wind tunnel, with particular emphasis on some reverse engineering. Suggestions were made for improvement and acquisition of new equipment, with some of these solutions being implemented.

A list of requirements was compiled for the data acquisition program and implemented under the name AeroIST, using the Qt platform, in a Linux environment. The program is open and scalable to new variables through plugins.

Experimental tests were conducted in the wind tunnel with a wing, at two Reynolds numbers, for validation of the program by comparison with the original program, WW6.

1 Introduction

There is a low speed wind tunnel available for tests at IST, with a data acquisition system that was purchased in the 1990's. The system has the capability to read forces and moments in all three axis of reference as well as two angles of rotation, namely pitch and yaw. The data acquisition program is named WW6 and operates under a MS-DOS environment. The model attitude is manually controlled by the operator.

The computer that houses the WW6 software had a malfunction, which stopped data acquisition tests in the wind tunnel. The replacement of the computer was not straightforward because of the hardware and software technology evolution in the last 20 years.

The complete replacement of the data acquisition platform was not possible due to financial constraints. So, the decision was made to implement a new data acquisition system that made use of existing equipment as much as possible.

To operate with the existing equipment it was needed to study and document the components of the data acquisition platform. Some components required reverse engineering to understand how these worked. Several manufacturers were contacted to provide instructions manual.

The original data acquisition platform could also be improved by including more hardware, specially the control for the wind tunnel motor and the control of the angles.

The software to be developed was decided to be open and scalable to accommodate new equipment and variables in the future.

The wind tunnel at the Air Force Academy (AFA) was visited in order to gain familiarity with similar systems and to provide a comparison of functionalities.

2 Components of data acquisition system

2.1 Forces and moments

The forces and moments exerted on the model are obtain by six load cells HBM-Z6. These are excited with a 12 V DC voltage and have a sensibility of 2 mV/V. The multimeter reads the voltage values sequentially and transmits them to the computer via the GPIB protocol.

The multimeter converts the signal from analogue to digital by means of integrations. The time of integrations can be configured from the set of 50 ms, 100 ms, 500 ms, 1 s, 5 s and 10 s [1]. These values constrain the data acquisition rate. Analysing the manual, there is also a time needed to switch channels of 150 ms, as confirmed by some experiments. Switching channels is needed because of the strong coupling between all six load cells. This coupling is accounted for in the post processing carried out by the software.

The multimeter has 6 1/2 digits and measuring ranges of 0.2 V, 2 V, 20 V, 200 V and 1000 V. The measuring range to be used will be the lowest due to the low voltages produced by the load cells, which gives a resolution of 0.1 µV.

Conversion from voltages to forces and moments is done by

$$R = \frac{R_N \times V_{dvm}}{S \times V_{cc}}, \tag{1}$$

where R_N is a nominal force, V_{dvm} is the voltage read by the multimeter, S is the load cell sensibility and V_{cc} is the load cell excitation voltage.

To decouple the interactions between the load cells, the program uses the following second order equation [2]:

$$R = AF + BF^2, \tag{2}$$

where A and B are two matrices provided by manufacturer that are specific to this wind tunnel, R is the result from Eq. 1 and F is the result of interest. F^2 is not the square of matrix F, but $F^2 = [F_1^2 \dots F_1F_6\ F_2^2 \dots F_2F_6 \dots F_6^2]^T$ to follow existing notation. The equation is solved using Newton method's, with the first iteration being obtained by inverting matrix A.

2.2 Angles

The balance has two angles of rotation, which are read by two absolute angle encoders AG661 from Stegmann, with 24 bits resolution. The precision error stated is 10^{-3} rad [3]. The encoders use the SSI protocol to communicate with the serial-to-parallel converters. The converter is the one to specify the data rate, which is set to 125 kHz. This frequency is much higher than that of the forces and moments.

The values obtained from the encoders must be converted to meaningful data. The conversion of angle β is done by $\beta = f_c(x - x_0)$, where $f_c = 2.11 \times 10^{-3}$ is the conversion factor, x is the value obtained from the encoder and $x_0 = 80751$ is the reference to which the measuring is done.

The conversion for the angle α is calculated by $\alpha = tan^{-1}(f_c(x - x_0))$, where $f_c = 9.007 \times 10^{-7}$ and $x_0 = 730303$ [4].

The communication with the serial-to-parallel converters is done using two ISA cards. No information could be found pertaining to these cards, so reverse engineering had to be employed.

Since neither information nor drivers were available for the cards, the communication from the software to these cards is done directly, which requires proper privileges on modern operating systems, unlike MS-DOS [5]. So, to get the program WW6 to work under a Windows 2000 environment it was necessary to use a third-party software, which allows direct communication with the hardware. To get the cards to work in a Linux environment, a kernel module had to be written to expose the device as a file, according to Unix philosophy.

To prevent the problem created by one of the ISA cards breaking down, a replacement was sought. The Dual Encoder USB Converter from BEI Industrial Sensors was purchased, which has a clock rate of 100 KHz and a number of bits per message between 8 and 32.

The control of the angles is done by means of a couple of Hi-T Drive Servo Actuator from Harmonic Drive Systems. These are powered by 24 V DC and manually controlled from the buttons in the +F1 box. These buttons are connected in such a way the changing the direction of rotation is done by changing the polarity on the motors.

A solution to incorporate the motors and their control into the data acquisition system was also sought. To automatize the control it was purchased an Arduino, using a ATmega168 micro-controller, aiming to drive some relays which would act instead of the buttons to close the circuit and actuate the motors.

2.3 Motor velocity

The wind tunnel is driven by a DC Thrige-Titan Lak 160 LA motor of 21.5 kW and 2280 rpm, controlled by a SIMOREG DC-Master unit from Siemens. The wind speed control is done by manually adjusting a potentiometer, with two buttons to start and stop the motor.

The SIMOREG unit has a serial port, which can be used for remote control by the computer, using the protocol USS [6]. While the port is serial, the wiring is not typical and an adaptor had to be constructed. The protocol is used in its simplest version, which consists of sending and receiving two words of 16 bits. To alternate between manual and remote control two parameters must be changed in the SIMOREG unit.

2.4 Temperature

There is an analogue thermometer installed at the wind tunnel, but it cannot be connected to the data acquisition system. An inexpensive digital thermometer was purchased at the same time as the Arduino. Its resolution, accounting for the Arduino analogue-to-digital conversion, is 0.4882 °C.

While the project was ongoing, a digital thermometer AP9512TBLK from APC was found. No information about using it was available, but with relatively small amount of reverse engineering, it was hooked up to the data acquisition system, with an improved resolution of 0.0331 °C.

2.5 Micro-manometer

The wind tunnel laboratory had a micro-manometer FCO12 of 3 1/2 digits from Furness Controls Limited, to measure differential pressures, to a maximum of $\pm 199.9\,\mathrm{mmH_2O}$, with a precision of $\pm 1\%$.

The micro-manometer can be calibrated manually or automatically and it has two gains of 1 and 10 to read 100 % or 10 % of the scale, respectively. An integration time constant from 20 ms to 10 s is available.

There is an analogue output directly related to the pressure, that goes from $-5\,\mathrm{V}$ to $5\,\mathrm{V}$, with relation to the gain used. This output can be connected to an analogue to digital converter of the Arduino, but it can only read positive differential pressures since the Arduino limits are from 0 V to 5 V. The resolution obtained is $0.019\,\mathrm{mmH_2O}$.

A pressure channel selector equipment was also found in the laboratory and included into the project. This equipment consists of two boxes that interconnect by a customized connection. The first box has 5 electro-valves which are controlled by second box, which contains two K2633 circuits from Velleman, an on/off switch and 8 LEDs to indicate which channel is active. The second box has a cable that, after reverse engineering, was connected to the Arduino, and thus allowing control by the computer.

2.6 WW6 software

The software used in the original data acquisition system is the WW6 program, developed to work in MS-DOS. As explained before, to get this program working under Windows a third party application is needed to allow direct access to the hardware.

The source code was made available to study and understand how it accessed and processed data from the hardware. Understanding the configuration used in the wind tunnel was also important.

2.7 Computer

At first, because of the ISA bus requirement, the computer was a Pentium II with a clock rate of 400 MHz. The installed operating systems was Windows 2000 for WW6 use and Xubuntu, a version of Linux, for AeroIST development. After acquiring the SSI to USB converter, a Pentium IV with a clock rate of 2800 MHz was used and the Linux system reused.

The computer requirements are two USB ports for the Arduino and the SSI-to-USB converter, a serial port for the SIMOREG unit and a PCI bus for the GPIB card.

2.8 Pressure scanner

Currently, there is no equipment for reading several pressures simultaneously. The acquisition of such equipment was considered but postponed due to to its cost. The software developed considered that such future expansion was possible and others alike.

One pressure scanner considered was the Pneumatic Intelligent Pressure Scanner of Pressure Systems. This system uses the serial Optomux protocol to interface with a computer.

3 Software development

3.1 Requirements

A list of requirements was compiled after observing the original system, the system at the Air Force Academy and some user cases scenarios. The following requirements are mandatory:

- Data acquisition from hardware;
- Display of raw data, as obtained from hardware;
- Control of possible equipment;
- Allow several models to be studied within the same session;
- Presenting aerodynamic coefficients of interest;
- Data visualization in table and graphic form;
- Data exportation to other programs;
- Allow for future expansion of variables. This requirement allows some technical knowledge;
- Software must run on the provided Pentium IV. No other performance requirement is needed;
- Reusing matrices' files from WW6.

The next items are considered optional:

- Store data in text files;
- Support several platforms;
- Translatable software.

3.2 Concepts

There are some concepts worth explaining. A variable is an abstraction of all variables that the software has access. Some of these variables can be controlled, so these can be said to be input variables, while the rest are output variables.

A reference is a calibration needed by some variables, such as the forces and moments. A measurement is the reading of all variables, associated with a reference.

The control of measurements is automatically done by changing a variable from an initial to final value, for a given step, or manually from the controls available in the graphical user interface (GUI).

A measurement has three further concepts: iterations, measures per iteration and settling time. An iteration is the set of measures done between changing of input variables. The measures per iteration is the number of measures done for any given iteration and the settling time is a period during which the software sleeps, allowing for the settling of the flow, after each iteration.

A reference and a measurement both contain normal data as well as raw data.

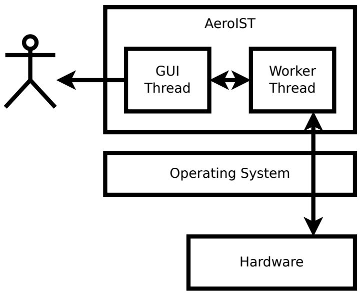

Figure 1: General architecture of AeroIST software.

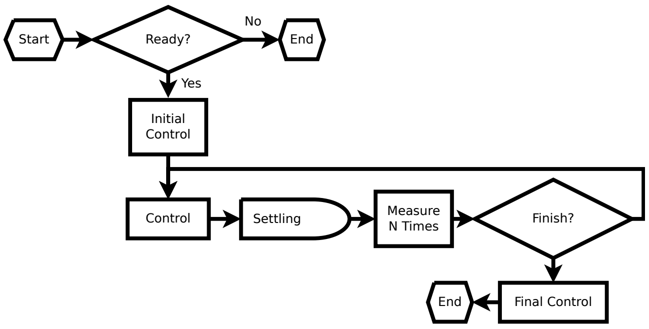

Figure 2: Measurement and control process.

3.3 Development environment

The software was developed in Linux, with a non real time kernel. The software is written in C++ and making use of the Qt library. This library is cross platform and provides many features such as graphical widgets, plugins, XML files and threads management.

The graphical plots are done with Qwt and using muParser for parsing mathematical expressions.

For GPIB, the National Instruments driver could not be made to work. The open project Linux GPIB Package is used, which requires compiling a module for the Linux kernel.

The communication with the other devices is done through serial connection. The implemented solution uses POSIX compatible calls.

3.4 Software architecture

The main components of the program are the worker thread, the MVC models created, the XML file structure and the plugin system.

To separate the communication with the hardware from the interaction with the user two threads are used, as depicted in Fig. 1. The communication between threads is done by means of Qt signals and slots' mechanism.

The worker thread is responsible for direct data acquisition. The algorithm used is represented in Fig. 2.

3.5 Models

Following the model-view-controller principle as implemented by Qt, the AeroIST software uses four models to manage references and measurements. The first model is *ReferenceList* and its purpose is to manage all references. A second model *MeasureList* follows the same idea for measurements.

The third model is *ReferenceModel* and it keeps all data pertaining to the calibration process. The final model is *MeasurementsModel* and it keeps all data concerning the data acquisition process.

The separation of model from its view allows, for example, to reuse the same table in the main AeroIST GUI interface for both the *ReferenceModel* and *MeasurementsModel*.

A proxy model, named *ProxyModel*, acts has a middle man between views and models. The intention of this module is to allow for the selection of normal or raw data to be visualized.

3.6 Plugins

One goal of the software is to be extensible in order to accommodate future variables and equipment. This is implemented by using the plugin system of Qt and the definition of four main classes.

The class *VariableMeta* provides general and constant information about a variable. The class *VariablePreferences* allows for configuration of options specific to each variable. These must be saved using the Qt system of QSettings.

The class *VariableModel* allows for configuration of options specific to a measurement or reference. These options must be saved in the XML file. This is the class that keeps data obtained from the equipment.

The class *VariableHardware* talks to the hardware and lives on the worker thread. This class is constructed by accessing a *VariableModel* and copying relevant information, specific to each variable.

A class *PluginManager* was coded to manage the lower details pertaining to all these classes.

3.7 Graphics

The software can plot mathematical relations between the data obtained from the hardware, thanks to muParser capabilities. The dialogue to make a new plot provides a list of the accepted variables as well as some constants.

For example, if one wishes to make a plot of the lift coefficient one would write for X the expression “Alpha” and for Y the expression “-Fz/(Pressure*mmH20)”, where mmH20 is a constant provided to facilitate conversion of pressure units.

3.8 XML files

A text based file was preferred to keep data on disc, so XML was chosen as the format to use.

The file stores the list of references and the list of measurements. Options specific to each variable are stored in its respective reference or measurement. The plot information is not saved to this file by design.

Table 1: Results for several different integration times in the multimeter.

Integration time (s)	Measures per iteration	Time ratios	F_z average std. deviation (N)	F_x average std. deviation (N)
0.05	400	4.210	0.7426	1.1696
0.10	200	2.589	0.6430	0.4541
0.50	40	1.318	0.1234	0.1331
1.00	20	1.158	0.0657	0.0436
5.00	4	1.032	0.0170	0.0219
10.00	2	1.016	0.0157	0.0125

A XML schema file is available to check the validation of the XML file. Alterations to the plugins must be accompanied by the respective alterations to this file and may break XML compatibility with previous versions.

3.9 Arduino protocol

The communication between the Arduino and the computer requires a protocol. A search was made to find an already existing one, with the better candidate being the Firmata protocol. None of the protocols seemed adequate, so a customized protocol was implemented.

The protocol follows a master and slave architecture, with the computer being the master. The protocol sends and receives eight characters, with the first and last being fixed characters. The second and third characters provide a command for the Arduino to process, with all other characters reserved for data specific to each command. A reply from the Arduino is mandatory.

4 Validation

Some wind tunnel tests were done to validate the program, using a wing based on the airfoil NASA LS(1)-0417. The first wind tunnel test was done to compare the results of using different time values in the multimeter, shown in Table 1. This procedure has been repeated at four different values of α, with an equally amount of time spent in each iteration.

The results indicate that a lower integration time constant has a higher impact on the total time of the test. At lower times, there is higher switching of channels and because changing channels takes some time hence the higher time ratios. The results also indicate that the standard deviation is lower at higher integration time constants.

To compare AeroIST with WW6, two wind tunnel experiments were conducted at Reynolds number $Re = 70\,000$ and $Re = 140\,000$. The lift coefficient for both these tests is represented in Fig. 3, without any correction for the model support.

The minor deviations seen in Fig. 3 are attributed to an aerodynamic nature. At a lower Reynolds number the forces F_z are of lower intensity than the respective forces at a higher Reynolds number. For this reason, the lower forces are subjective to greater influence of errors. However, the results are comparable and thus validate the program developed.

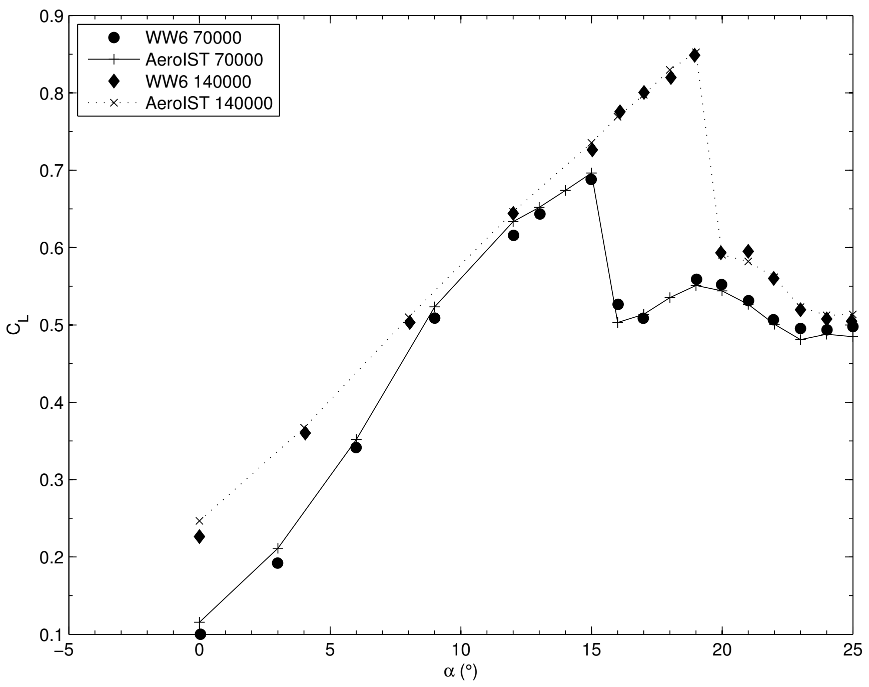

Figure 3: Lift coefficient for both Reynolds numbers.

5 Conclusions

The original data acquisition system of a low speed wind tunnel was studied and documented. Several improvements were laid out with some being carried out.

A software, designated AeroIST, was developed to perform both data acquisition and control of the system. The program satisfies all obligatory requirements and some optional ones as well.

Wind tunnel tests were perform to attest the functionality and quality of the program, with the data results validating the program.

References

[1] PREMA. *Digital Multimeter DMM 5001 and DMM 6001 Instruction Manual.*

[2] Carl Schenck AG. *Compact Wind Tunnel Balance*, 9 1992.

[3] Stegmann. *AG 661 Absolute Angle Encoder.*

[4] Horiba. Program ww6 source code.

[5] Doc: Port i/o with inp() and outp() fails on windows nt, 2012.

[6] Siemens. *SIMOREG DC Master 6RA70 Series Base Drive Instructions*, rev6.0 edition.