This thesis was typeset with LaTeX, using a modified version of the University of California PhD dissertation class file, `ucthesis.cls`. Unless otherwise noted, all figures in this thesis were created by the author using IDL® CorelDraw® or the GNU Image Manipulation Program (GIMP).

OPTIMISATION OF THE INSTRUMENTAL PERFORMANCE OF IRMA

Regan Eugene Dahl

B.Sc. Computer Engineering, University of Alberta (2004)

A thesis
submitted to the School of Graduate Studies
of the University of Lethbridge
in partial fulfilment of the
requirements of the degree

MASTER OF SCIENCE

Department of Physics
University of Lethbridge
Lethbridge, Alberta, Canada

To Jillian and David
for your patience and support during my Master's Program.

Abstract

The Infrared Radiometer for Millimetre Astronomy (IRMA) is a passive atmospheric water vapour monitor developed at the University of Lethbridge. It is a compact, robust, and autonomous instrument, which is capable of being operated remotely. The latest model is based on a PC/104 running an AMD 133 MHz SC520 processor, which allows for more flexible control of the unit. The modifications and upgrades to the software required for the transition to this new platform are discussed in this thesis. In addition to software optimisation, a new calibration method has been developed as the unit has become better understood. This method has been verified through test campaigns carried out in Lethbridge and Chile. The results of these tests are included in this thesis.

Acknowledgements

This thesis is the result of contributions by many people. It would never have been completed without the assistance of numerous others. Some have helped me directly, while others I have stood on the shoulders of their pervious work, in order to reach a little higher. I would especially like to thank my supervisor, David Naylor, for welcoming me into the Astronomical Instrumentation Group to complete my Masters degree.

I would like to acknowledge those who have made IRMA what it is today.

Graeme Smith, for starting the ball rolling with the construction of the IRMA prototype.

Ian Chapman, for his work on atmospheric modelling, allowing us to take IRMA anywhere in the world.

Ian Schofield, for all his work on the control and communication software.

Robin Philips, for his work as IRMA Project Manager during the first year of my Masters, and for all the help and guidance he gave me.

Richard Querel, for working with me on the IRMA project for these past two years.

Greg Tompkins, for all the insight and entertainment he brought to the group, as well as his electronics expertise.

Frank Klassen, for his machining work supporting the IRMA project.

Brad Gom, for his mechanical work as well as the additional time he took from his other projects to help with IRMA.

Matthias Schöck for acting as Project Manager for a month helping to calibrate the TMT units. As well as the entire TMT site selection team.

Karim Ali, Scott Jones, Dan Sirbu, and Amy Smedes for the summers they worked on IRMA.

Locke Spencer for always being willing to help out with LaTeX or IDL.

Thanks to Drs. Peter Ade and Carole Tucker, University of Cardiff, for supplying us with specially designed IR filters. Thanks to Fluke for loaning us the Ti20 Thermal Imager. Thanks also to NSERC, HIA and NRC for funding the IRMA project.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

The Infrared Radiometer for Millimetre Astronomy (IRMA) is an instrument used to measure the amount of water vapour in the atmosphere, usually expressed as precipitable water vapour (PWV). It was developed as a collaboration between the University of Lethbridge and the Herzberg Institute of Astrophysics to be a fast, relatively simple, robust radiometer which could be used to determine PWV at high altitude sites around the world. Central to IRMA is a sensitive infrared photoconductive detector, used to measure the atmospheric emission of a carefully selected spectral band centred at $\sim$20 μm. The total radiant infrared flux received from the atmosphere is converted into PWV by use of an atmospheric model, BTRAM (Blue Sky Transmittance and Radiance Atmospheric Model) [2], which has been developed by our research group.

1.2 Precipitable water vapour

Water vapour is simply water in the gaseous phase. Precipitable water vapour (PWV) is a measure of the columnar abundance of water vapour in the atmosphere. PWV refers to the depth of liquid water present upon condensing a column of atmosphere. PWV is a linear parameter referred to in units of mm. For example, 1 mm PWV condensed over an area of 1 m^2 would correspond to 3.34×10^{25} molecules of water in the atmosphere.

1.2.1 Importance of Measuring PWV

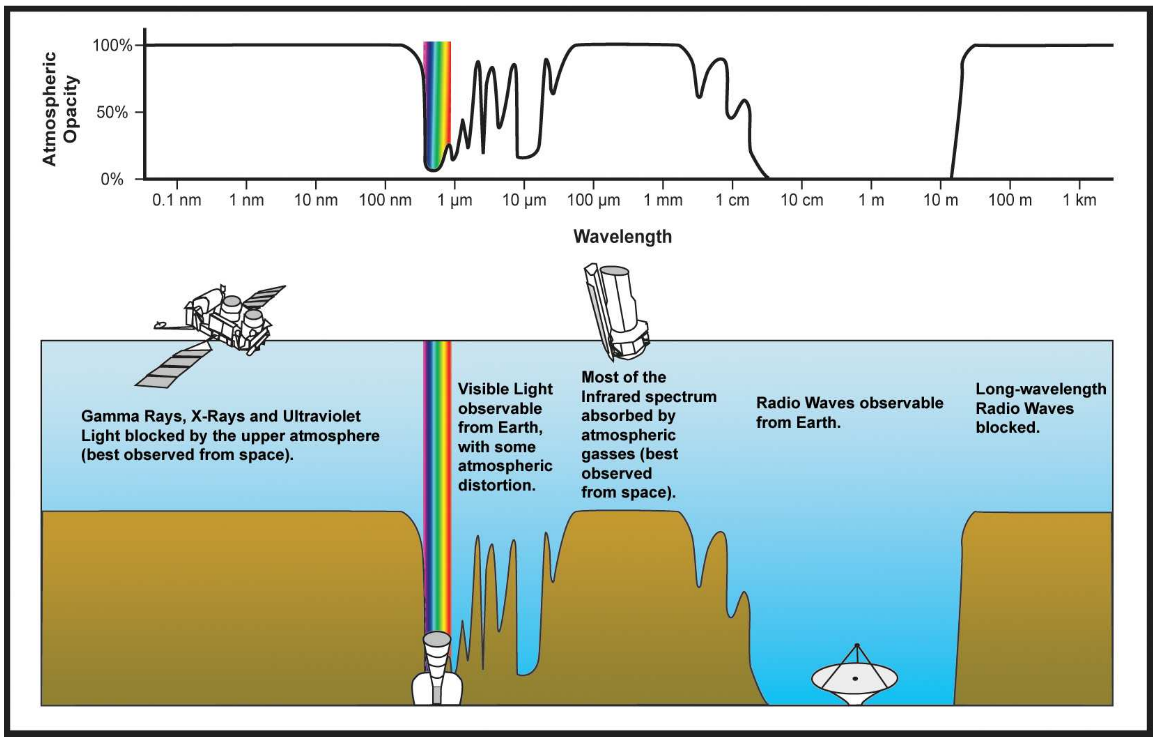


Figure 1.1: The atmospheric opacity as a function of wavelength. The graphs show with regions of the electromagnetic spectrum are visible from ground based sites. Courtesy of NASA/JPL-Caltech.

Astronomical objects emit light at all wavelengths of the electromagnetic spectrum. By and large, this light travels vast distances through space relatively unimpeded until it reaches the Earth's atmosphere. While our atmosphere is transparent to some forms of radiation, it is opaque to many wavelengths as shown in Figure 1.1 [3]. The dominant source of opacity for infrared wavelengths is due to atmospheric water vapour. Therefore, the best infrared observing sites require low column abundances of water vapour. This is why the world's best observatories are located at high and dry mountain tops, which place them above most of the atmospheric water vapour.

Since water vapour is the primary source of atmospheric opacity, quantifying its abundance, usually expressed in terms of PWV, is critical for the calibration of astronomical data at observatories around the world. For this reason it is also key factor in deciding the location of future ground based observatories. IRMA has been used as both an opacity meter for existing observatory sites (Figure 1.2), as well as in a site testing role (Figure 1.3).

Water vapour is also a cause of phase delay in submillimeter interferometric arrays. These submillimeter arrays are ground-based observatories which are able to synthesize a massive receiving antenna whose diameter equals the length of the maximum baseline of the array. The maximum baseline is the distance between the two farthest-separated antennas in the array.

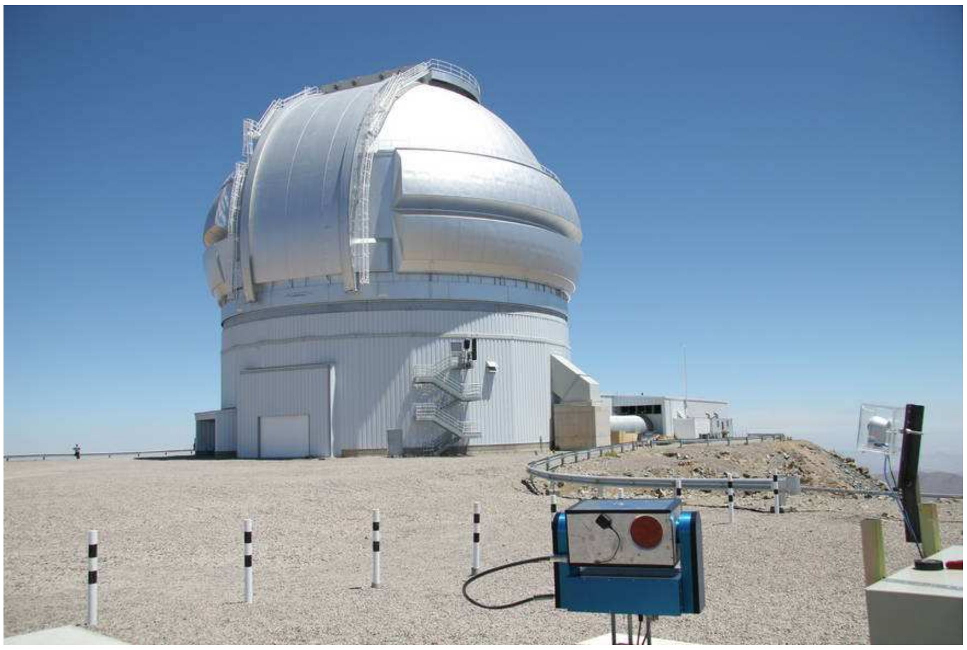


Figure 1.2: A photograph showing an IRMA unit deployed in front of the Gemini South telescope in Chile.

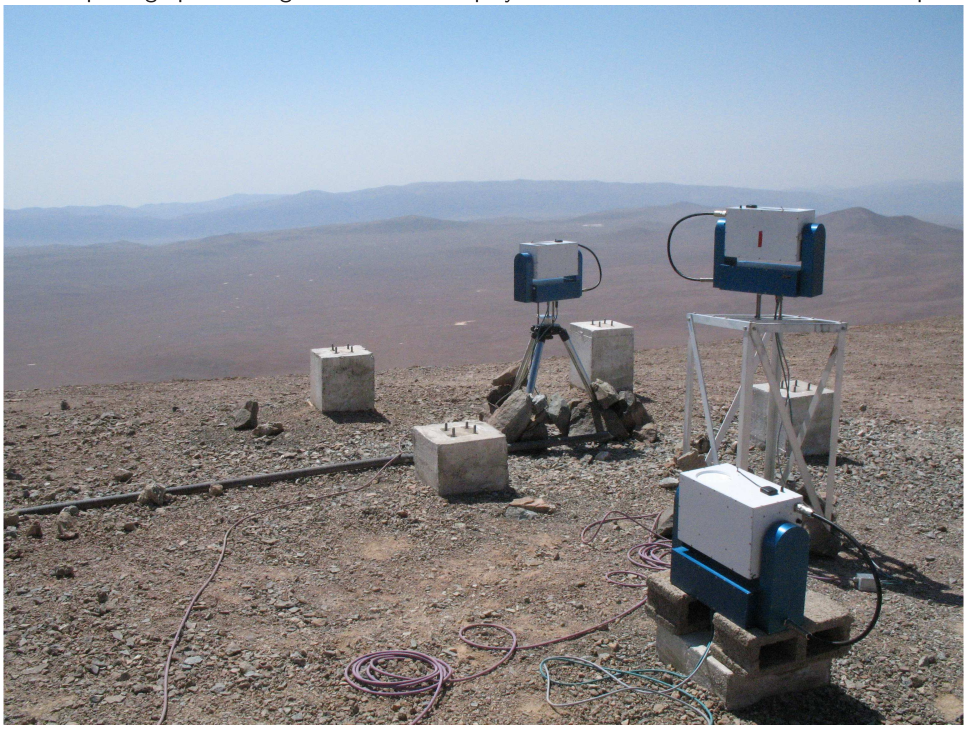


Figure 1.3: A photograph showing three IRMA units deployed on one of the Chilean sites being considered as part of the Thirty Meter Telescope (TMT) project.

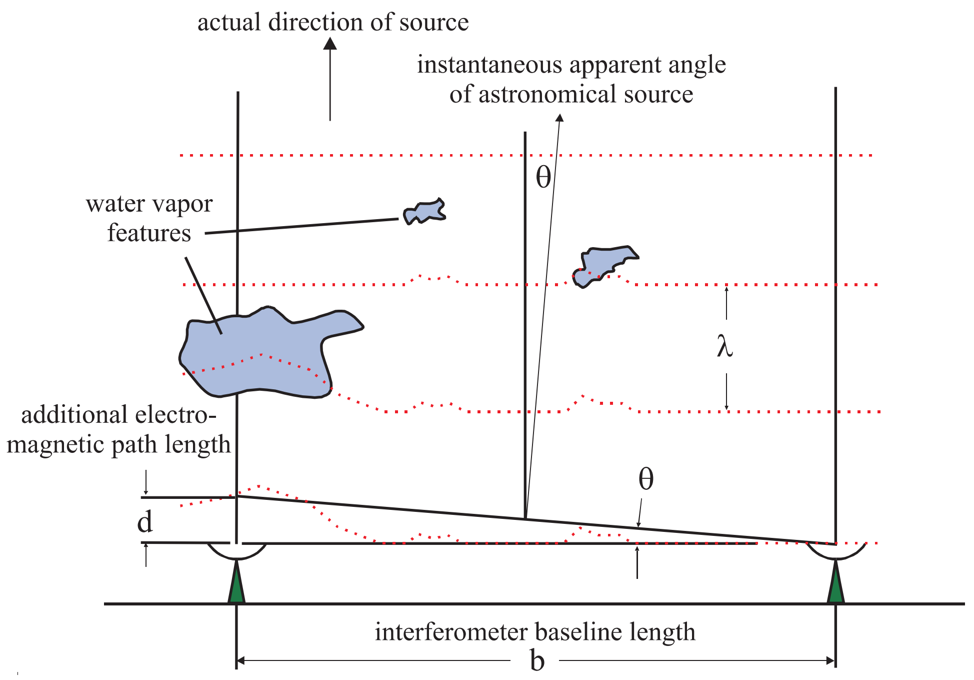


Figure 1.4: A schematic of the cause of atmospheric phase distortion of a celestial signal. [1].

At millimeter wavelengths, water vapour found in the Earth's atmosphere is present in sufficient amounts to slow down the incoming wavefront of a celestial signal. Studies have shown that for every 1 mm of precipitable water vapour along the line of sight of a millimeter telescope, the optical path length through the atmosphere increases by 6 mm [4]. Moreover, the distribution of water vapour in the atmosphere is neither spatially nor temporally homogeneous inside the column projected from the antenna's receiving dish. For example, with a wind speed of 20 m/s, which is not uncommon at the altitude of many observatories, the mass of air in the column above the telescope changes every ∼0.5 s. Thus, in general the amount of water vapour along the line of site of each receiving antenna will be different.

The effect of atmospheric phase distortion is illustrated in Figure 1.4, which shows an interferometric array with two antennas. The antenna pair observe the same object, whose wavefront appears planar in the upper atmosphere. The wavefront above the left hand antenna passes through a region of water vapour, which adds excess optical path length, d, to the incoming signal. Interferometry requires the precise measurement of the time the wavefront was received at each antenna. The apparent direction of the observed object is perpendicular to the planar wavefront. A slight phase error manifests itself as a slight change in the immediate apparent angle of the astronomical source, decreasing the spatial resolution of the interferometer. Real-time knowledge of the PWV above each antenna would allow for a partial correction of this phase delay.

IRMA is well-suited for phase correction. It has been designed such that it samples the same patch of sky as a 10 m antenna, as described in §2.2. Importantly, IRMA is a passive radiometer and therefore, does not require any radio frequency (RF) components which may interfere with the RF instrumentation of a radio telescope. Figure 1.5 shows IRMA mounted to one of the antennas at the Smithsonian Millimeter Array (SMA) on Mauna Kea in Hawaii during phase correction testing in June 2004.

1.2.2 Measuring Water Vapour

Many methods have been described in literature to measure the water vapour content of the atmosphere. These include: 183 GHz radiometer [5], 225 GHz Radiometer (Caltech Submillimeter Observatory (CSO) Taumeter) [6], tropospheric delay measurements using GPS signals [7], and infrared radiometer [8]. IRMA is the only method to employ a passive infrared radiometer. Moreover, it is also compact, robust, and able to be operated remotely.

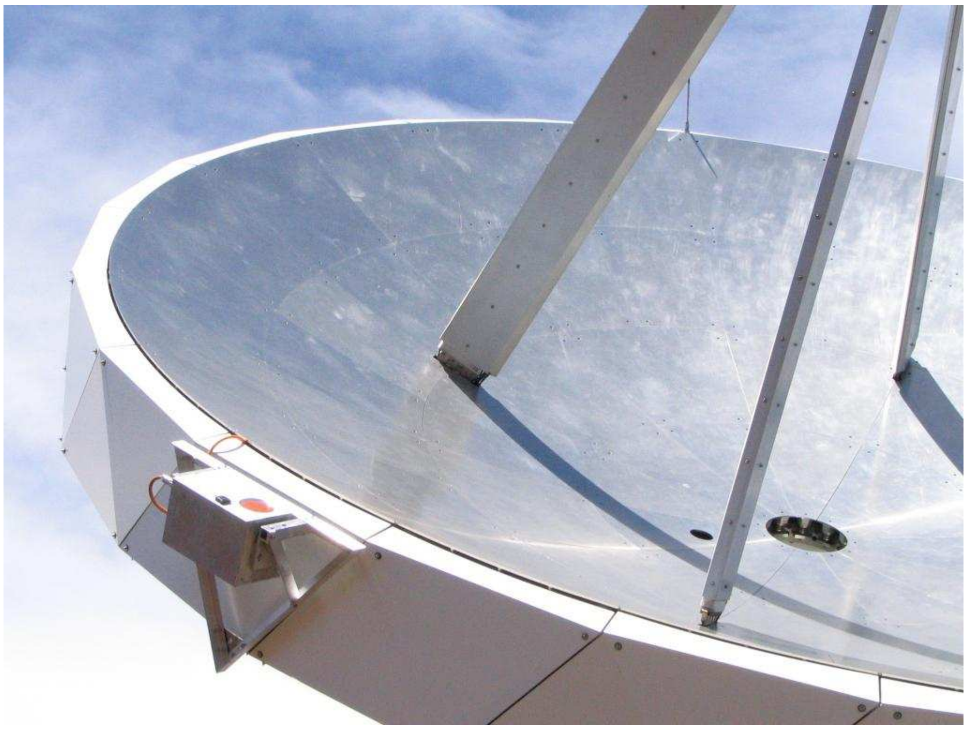


Figure 1.5: A photograph showing IRMA attached to the side of one of the antennas of the Smithsonian Millimeter Array telescope on Mauna Kea, Hawaii.

1.3 The IRMA Solution

1.3.1 The Instrument

The IRMA instrument will be the main focus of my thesis. IRMA is essentially a very sensitive infrared thermometer, which measures the emission from the atmosphere in a narrow spectral band where only water vapour contributes to that emission. The IRMA device consists of an optical system which focuses the emission onto a photoconductive detector, which will be discussed in §2.2. Finally, this flux can then be converted into PWV using an atmospheric model, BTRAM, which has been developed by the Astronomical Instrumentation Group at the University of Lethbridge and is briefly described in the next section. A mechanical system, described in §2.3, has been designed such that IRMA can be pointed in any direction, and gives it the ability to protect itself from the elements with a weather shutter which doubles as a calibration source. The electronics, discussed in §2.4, facilitate the control of the instrument through software, which will be discussed in Chapter 3. Other software, discussed in Chapter 4, has also been designed which allows for remote access and control of the IRMA unit in addition to data reduction routines.

1.3.2 The Atmospheric Model - BTRAM

To determine the relationship between the detected flux and PWV, IRMA employs the atmospheric model BTRAM (Blue Sky Transmittance and Radiance Atmospheric Model). BTRAM [9] (originally known as ULTRAM [2]) is a line-by line, layer-by-layer radiative transfer model used to simulate transmission, emission, or opacity of a user-definable atmosphere.

The impetus for developing BTRAM, was that IRMA could be deployed in locations where standard models do not apply. The main goal in developing BTRAM was to provide the user with the flexibility to model radiative transfer under local conditions. It was originally designed as a customisable GUI with a simplified subset of geometries available in Fast Atmospheric Signature Code (FASCODE) [10]. FASCODE was difficult to customise due to the volume of code, causing modifications to one aspect of the model would have unpredictable consequences in other parts of the code. BTRAM has been developed into an independent, easy to use, fully customisable atmospheric model using the HITRAN 2004 database of spectral lines [11].

Within BTRAM there are nine pre-built atmospheric profiles: Antarctic Summer, Chajnantor Winter, Mauna Kea, Mid-Latitude Summer, Mid-Latitude Winter, Sub-Arctic Summer, Sub-Arctic Winter, Tropical, and U.S. Standard Atmosphere 1976. A customised atmospheric model is built by either modifying one of the pre-built profiles, or creating one based on radiosonde data when available. A radiosonde is a suite of meteorological instruments carried by a weather balloon, which makes *in situ* measurements of pressure, temperature, wind speed, and dew point or relative humidity. These data are then used to determine inputs for the creation of a site-specific atmospheric model, such as the scale height and the adiabatic lapse rate.

Astronomical observations must account for the variable transmission through the earths atmosphere. While IRMA measures emission from of the Earth's atmosphere, by application of Kirchoff's Laws the transmission can be derived and a suitable correction applied to the astronomical observation. Previous studies have shown that, at a wavelength of 20 μm, water vapour is the primary source of emission in the atmosphere [12]. Thus, by measuring the emission at these wavelengths, the detected flux can be converted to water vapour abundance through the use of an atmospheric model.

From any site there are natural variations of temperature and pressure, which must be taken into account when generating the model. Using the range of local temperatures and pressures common to the desired site, BTRAM is run in a batch mode to generate a series of model files. These files are combined to form a lookup table relating flux to PWV for the given range of local meteorological conditions. The operator uses the atmospheric flux values obtained by IRMA, along with the concurrent temperature and pressure, to determine the corresponding PWV value from the lookup table. The final accuracy of the retrieved PWV value depends not only on instrumental uncertainties, but also on uncertainies in the atmospheric model. For this reason, where possible, nearby radiosonde data are used to generate the models used by IRMA. Error analysis studies have also been performed on the models and show that the contribution to the total error budget is driven by the accuracy of the scale height, and the adiabatic lapse rate. A detailed analysis of how the key parameters affect the resulting PWV output from the model has been performed. The scale height of water vapour was found to be the parameter most affecting PWV sensitivity. While scale height of water vapour is critical to the accuracy of the model, it remains the most difficult to measure in real-time [13].

1.4 Summary

Water vapour is an important component of the atmosphere. Due to its interference with astronomical signals, understanding the quantity of water vapour in the atmosphere is especially important for astronomy. IRMA provides a method of measuring the columnar abundance of water vapour, PWV, for astronomical applications. This thesis describes the work that I have done in optimising the instrumental performance of IRMA, specifically improving the performance of the control and data acquisition software and refining the calibration schema of the radiometer.

Chapter 2 gives a brief history as well as an overview of the current optical, mechanical and electronics design of the IRMA Hardware. Having a knowledge of the physical instrument is required for an understanding of the system software. Chapter 3 presents an overview of the previous version of the IRMA software. My work involved implementing and improving the previous distributed version of IRMA onto a new platform. Chapter 4 describes the control and data analysis software, which allow the unit to be controlled remotely and the calibrated data products to be reduced. Chapter 5 introduces an improved calibration schema, which was verified through test campaigns both in our laboratory in Lethbridge and on a remote mountain top in Chile. In the second part of Chapter 5, a case study is presented demonstrating how the software was able to be reconfigured to acquire scientific data in the event of an unexpected hardware failure. Chapter 6 concludes the thesis with the future directions of the IRMA project as well as some final thoughts.

Chapter 2

IRMA Hardware

2.1 Overview

The IRMA concept has evolved from a barebones prototype into a compact, autonomous, remotely operated instrument. During this evolution, most of the original prototype was redesigned. The IRMA hardware can be divided into three main categories: Optics, Mechanical Design, and Electronics. A brief history, along with the current design, will be presented in this chapter.

2.2 Optical System

The IRMA optical system is the most important part of the unit, since it enables the unit to observe the sky, as well as calibration sources. The prototype, IRMA I, shown in Figure 2.1, was developed by a previous graduate student, Graeme Smith [1]. The unit mounted on a three-legged optical table and consisted of a plane scanning mirror, which directed light to an off-axis parabolic mirror, which subsequently focused the light at the

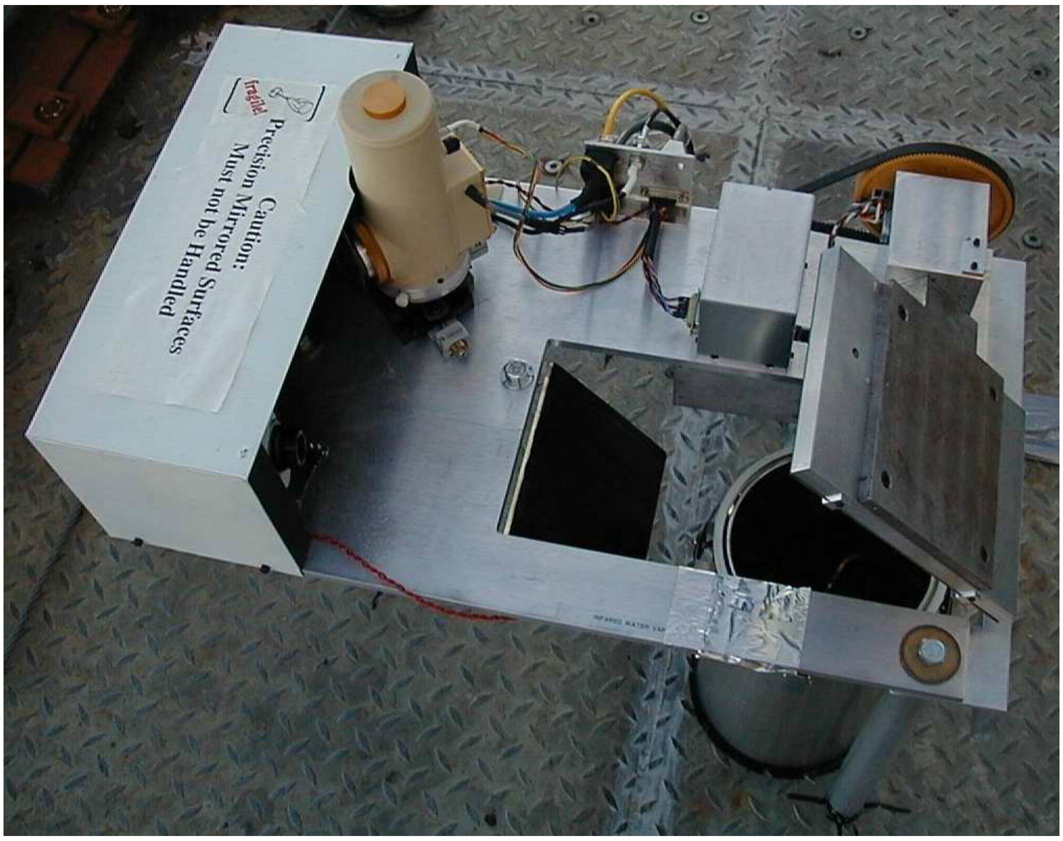


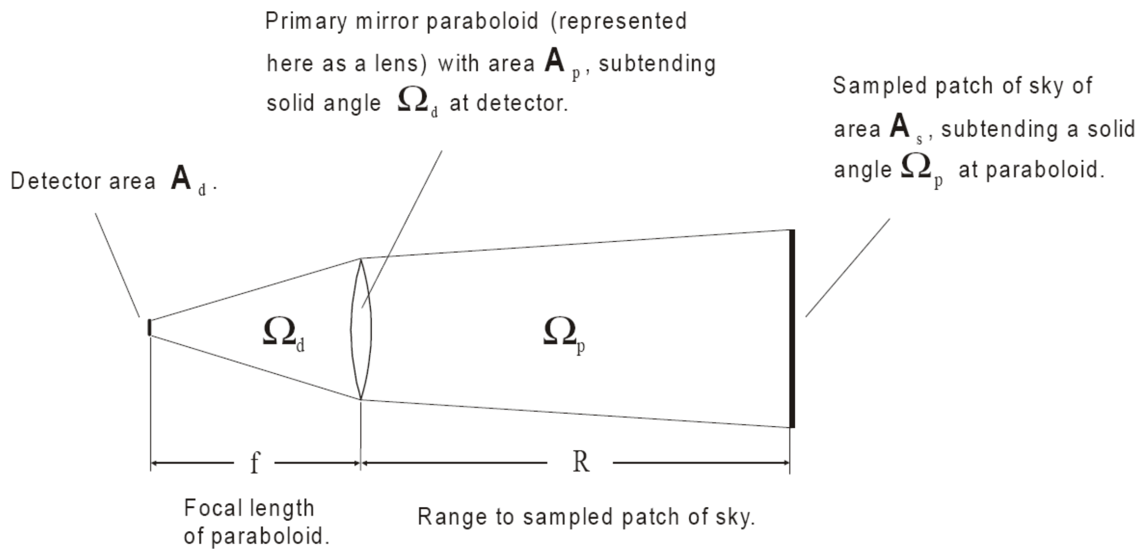
Figure 2.1: A photograph of the original IRMA instrument.

detector. The scanning mirror provided a range of observable zenith angles from 0 to 70.38°, which represent an airmass range from 1 to 3. The prototype was unable to move in the azimuth direction and therefore, was fixed when the instrument was deployed.

The initial design was improved by obtaining a detector with higher responsivity and using a bandpass filter that more closely matched the desired spectral range. Furthermore, a complete mechanical and electrical redesign was undertaken to make the unit more compact, autonomous, and robust. As part of this redesign, the scanning mirror was removed and the parabolic mirror was mounted in a compact box, such that the system could be pointed in any arbitrary direction by use of an Alt-Az mount, as shown in Figure 2.3. By replacing the original 45° off-axis parabolic mirror with a 90° off-axis parabolic mirror, the optical system became more compact.

2.2.1 Parabolic Mirror Design

A 90° off-axis paraboloid was chosen as the focusing optic to avoid obscuring part of the optical beam by a secondary mirror assembly, and therefore maximize the optical throughput. The focal length of the optical system was determined by the size of the detector element, 1 × 1 mm, and the atmospheric sampling criterion, which was chosen to sample a ∼10 m patch of sky at an altitude of ∼1 km, thus defining a field of view of about 1/100 radian.



Primary mirror paraboloid (represented here as a lens) with area A_p, subtending solid angle Ω_d at detector.
Sampled patch of sky of area A_s, subtending a solid angle Ω_p at paraboloid.
Detector area A_d.
Ω_d
Ω_p
f
R
Focal length of paraboloid.
Range to sampled patch of sky.

Figure 2.2: A schematic of the equivalent optical system of IRMA. [1]

Referring to the schematic of the optical system in Figure 2.2, conservation of the throughput of the optical system requires that

$$A_d \Omega_d = A_p \Omega_p \tag{2.1}$$

where A_d is the area of the detector, Ω_d is the solid angle viewed by the detector, A_p is the area of the paraboloid, and Ω_p is the solid angle subtended by the 10 m diameter patch of atmosphere viewed at a distance of 1 km. The solid angle subtended by the sampling area A_s at the parabolic mirror at a distance R can be approximated as $\Omega_p = A_s/R^2$, and, similarly, the solid angle subtended by the paraboloid at the detector is $\Omega_d = A_p/f^2$. Equation 2.1 can then be rearranged to specify a focal length

$$f = R\sqrt{\frac{A_d}{A_s}} = R\sqrt{\frac{d_d^2}{d_s^2}} \tag{2.2}$$

where d_d is the diameter of the detector element and d_s is the diameter of the source area. The focal length of the paraboloid is the determined by d_d, d_s, and R, which have been previously determined, and is independent of the diameter of the paraboloid. Inserting the values into equation 2.2 gives a focal length of

$$f = (1000)\sqrt{\frac{0.001^2}{10^2}} = 0.1\ m \tag{2.3}$$

Thus, a 10 cm diameter f/1 off-axis parabolic mirror was chosen as a good compromise between collecting area and compact design.

2.2.2 IR detector and filter design

The incident infrared radiation is focused by the parabolic mirror onto a Mercury-Cadmium-Telluride (MCT) photoconductive detector, supplied by Kolmar Technologies [14], which is cooled to 77 K. Since the development of IRMA I, improved detectors have been obtained, which have better detectivity and responsivity, helping to increases the observed signal-to-noise ratio. The resistance of the detector changes as a function of infrared radiation which falls upon it. When biased with a constant current source, this resistance change is detected as a change in voltage across the detector. This voltage signal can then be amplified and digitized by an analog-to-digital converter. While initially cooled with liquid nitrogen, to allow for remote operation, the IRMA detector is now cooled by a closed cycle Stirling cooler.

In order to restrict the incoming radiation to the desired spectral region, $450 - 500$ cm^{-1}, a bandpass filter is placed in front of the detector. The second generation IRMA II included a filter, provided by Professor Peter Ade of Cardiff University, Wales, which uses resonant capacitive and inductive micro-elements [15]. These filters provide superior performance over commercially available filters.

2.2.3 Chopper wheel

A five-blade reflective chopper modulates the incoming optical beam at ∼450 Hz. The detector alternates between views of the radiation from the sky and from an unfocused view of itself, which produces a chopped signal of high stability over a wide range of temperature variations. This signal is then detected using a standard lock-in amplifier.

2.3 Mechanical Design

As mentioned earlier, the prototype IRMA I was constructed on a three legged instrument platform. This platform contained all of the optical components, two blackbody references and an LN$_2$-cooled infrared detector dewar assembly. This configuration worked

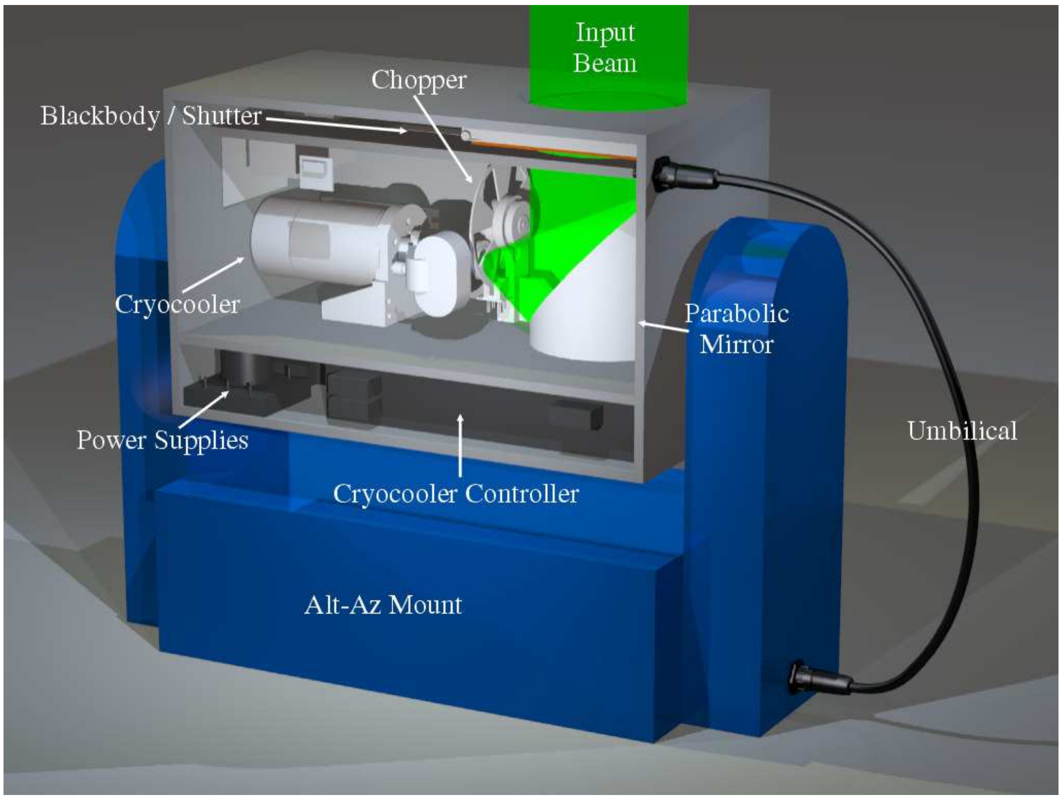


Figure 2.3: A rendered model of IRMA in its Altitude-Azimuth fork mount.

well for initial testing and proof-of-concept, but a more compact, and robust design was required for autonomous remote operation. The IRMA unit was designed to fit into a compact box such that it could be mounted on the side of a radio antenna when used in a phase correcting role, see Figure 1.5. An Alt-Az mount was also designed to allow IRMA to be pointed in any direction when not attached to an antenna. The wet cryostat was a major stumbling block to remote operation since it required an operator to refill the dewar every 4 to 6 hours. Moreover, the wet cryostat could not be attached to the side of an antenna because had to be kept vertical. To address this problem and allow for remote operation, the wet cryostat was replaced with a Stirling cycle cooler, which was able to achieve liquid nitrogen temperatures without the use of cryogens.

2.3.1 IRMA Module

The main IRMA module can be divided into four main compartments: optical, electronics, weather shutter, and power supplies. The module is designed to require only mains power and an ethernet connection. The mains power is routed to the bottom of the module where the power supplies provide 24 V and 5 V DC to the unit. The 24 volt supply powers the Stirling-cycle cooler, the lid motor and the temperature controlled calibration source, while the 5 volt supply powers the IRMA computer and the other electronics. Also mounted in the bottom compartment is the Stirling cooler controller, which regulates the temperature of the cold finger of the cooler.

On one side of the module is the optical compartment, which is visible in Figure 2.3. This occupies the majority of space in the IRMA module. The optical system is discussed in section 2.2. The IR flux from the atmosphere enters the unit through a 117 mm diameter aperture, which is directly above the parabolic mirror. The 90° off-axis parabola directs radiation to the detector, which is mounted horizontally to the end of the cold finger of a Hymatic NAX025-01 Stirling-Cycle Cooler. A vacuum chamber surrounds the cold finger which is evacuated to 1×10^{-4} mbar. This vacuum is designed to last five years and must have a leak rate no greater than 1×10^{-15} mbar cm^{-2} s^{-1} in order to allow the cryo-cooler to operate at its target temperature of 70 K. Also mounted inside the vacuum chamber on the detector block are the bandpass filter and a cold stop. The window of the vacuum chamber is made of anti-reflection coated ZnSe [16]. The cold stop was designed to restrict the viewing angle of the detector to the size of the mirror. However, during tests in the lab it was found that there was some spill-over of the beam, §5.1.2, which was corrected by adding a second aperture to the exterior of the ZnSe window. The optical chopper is mounted such that the blades pass in front of the exterior aperture.

Due to the fast optics of the IRMA, if the unit is pointed directly at the Sun, the focused light is sufficiently intense to burn a hole in the infrared filter and damage the detector. Unfortunately, this occurred twice on previous IRMA models. To prevent this from recurring, a solenoid operated shutter was incorporated into the design such that if the unit moves within 15° of the Sun it blocks the beam. This Sun shutter is activated independent of software, by a phototransistor mounted in a small hole on top of the unit on axis with the IRMA field of view; it may also be controlled via software.

The electronics are mounted on the side opposite the optical system and include the PC/104, the IRMA motherboard, and the pre-amplifier. The electronics are described in more detail in §2.4.

A narrow compartment at the top of the IRMA unit contains a weather shutter. The shutter can be positioned via a lead screw to cover the main aperture in case of inclement weather. The shutter serves a dual purpose, as a blackbody calibration source is mounted on its underside. When the shutter is closed, the detector views the blackbody, which is used to provide radiometric calibration.

2.3.2 Alt-Az mount

An Alt-Az mount is provided to allow full range of motion when the IRMA is not attached to the side of a radio antenna. In this case, power and Ethernet are connected to the bottom of the Alt-Az mount and then provided to the main IRMA box through an umbilical cable. Alt-Az articulation is powered by two Maxon EC167129, low-noise, 50 W brushless DC motors, each coupled with a Maxon 1QEC50V digital motor control unit. Axis rotation for each direction is geared down substantially by a 1621:1 gear head. An additional 8:1 gear reduction is provided by belts connecting the motors to their respective axes.

For the azimuth direction, the motor is mounted in one arm of the mount and connected via a belt to a fixed central gear about which the unit rotates. The unit is capable of rotating $\sim 370°$ about its azimuth axis; this range, being defined by optical limit switches, restricts the rotation to prevent damage to the wiring connecting the articulating parts. The altitude motor turns a gear in the arm of the mount which is fixed to the main IRMA box and allows rotation of $\sim 185°$. To prevent rotation beyond these limits and prevent the main IRMA module from contacting the Alt-Az mount, optical limit switches are again used. The motors are disabled when the switch is interrupted by one of two metal tabs attached to the rotating housing.

Limit detection is independent of software in order to eliminate the risk of runaway axis movement damaging the mount if the software were to fail. Attached to each of the axis are US Digital E6M optical encoders, which provide positional data to a resolution of 8192/360 counts/degree as discussed in §2.4.7.3.

2.4 Electronics

2.4.1 Overview

The IRMA electronics have evolved significantly since the original prototype. The original system was very basic, being controlled through the parallel port of a laptop com-

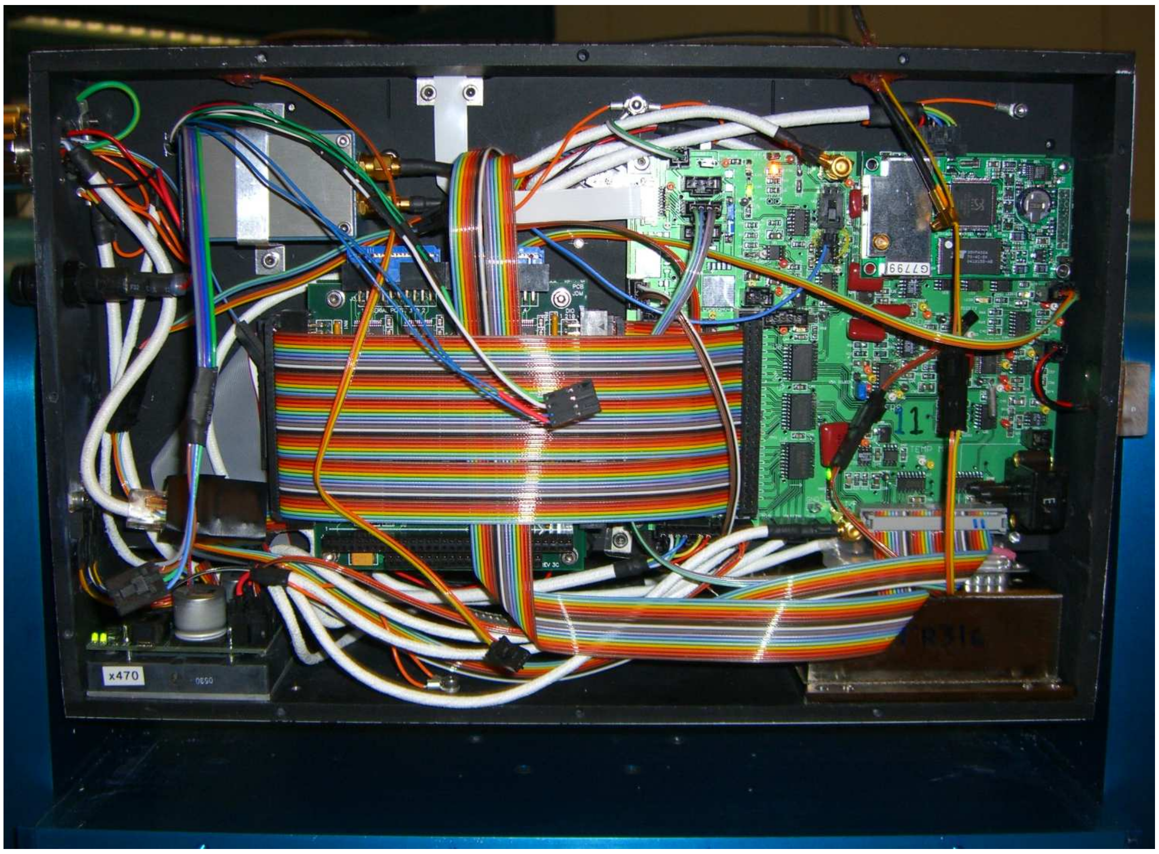


Figure 2.4: A photograph of the electronics side of the IRMA module. The optical compartment is located immediately on the other side of the vertical wall shown in the image.

puter. This system was first upgraded to allow commands to be sent to the laptop over the internet for remote operation. With the new mechanical design, the electronics were integrated into the IRMA unit. This system was based on an 8-bit Rabbit Semiconductor RCM2100 microcontroller [17], which could be controlled remotely over the internet but still required a command processor (CP) computer for the issuance of commands. If the network link between the RCM2100 in the IRMA and the CP were severed, the unit would cease to function. Furthermore, in order to upgrade the software, it was necessary for an operator to reprogram the microcontroller manually on site. To address these issues, in the latest version of IRMA, the CP and the 8-bit microcontroller have been replaced by a 32-bit PC/104 inside the IRMA unt. A significant part of my thesis has involved migrating and optimising the code to run efficiently on the PC/104.

2.4.2 PC/104

At the heart of the electronic system (Figure 2.5) is a Winsystems PCM-SC520 PC/104 running an AMD 133 MHz SC520 processo [18]. PC/104s are true IBM PC compatible computers capable of running a standard desktop operating system. The PC/104 is responsible for the communication, control and data acquisition of the IRMA unit. The PCM-SC520 is robust and designed for extreme conditions and temperatures of -40 °C to 85 °C. It contains an onboard Intel 82551ER 10/100 Ethernet controller and an onboard Compact Flash socket, which is used for data storage.

2.4.3 Diamond Systems - Emerald-MM-DIO

The PCM-SC520 lacks sufficient I/O lines to be able to control the IRMA unit. A Diamond Systems Emerald-MM-DIO [19] connects through the 16-bit PC/104 bus and provides 48 bi-directional digital I/O (DIO) lines as well as four serial ports. The Emerald-MM-DIO is also rated for operating temperatures of -40 °C to 85 °C.

2.4.4 Pre-Amplifier

The measured signal from the MCT detector is very small, having a noise floor of $\sim$ 10 nV, and first needs to be amplified in order to make efficient use of the full range of the ADC. The detector signal is connected to a low noise pre-amplifier, with a gain of 10,000, before passing to the IRMA motherboard.

2.4.5 IRMA motherboard

The IRMA motherboard is a custom electronics board, designed and built in-house in our electronics fabrication laboratory. It contains most of the electronics required to control the IRMA unit. The PCM-SC520 interfaces with the IRMA motherboard though the Emerald-MM-DIO board. The 48 digital I/O lines are divided between two connectors (JP5 and JP6), each containing 24 I/O lines, which plug into the IRMA motherboard. In order to simplify the configuration, JP5 was used for input and JP6 was used for output. The motherboard contains the electronics for the data acquisition system including signal conditioning, lock-in amplifier, temperature sensor multiplexing, weather shutter control, blackbody control, chopper control, and GPS.

2.4.6 Cryocooler Controller

The cryocooler controller communicates with the PC/104 through a serial port on the Emerald-MM-DIO board. It is a specialized controller from Hymatic designed to regulate the temperature of the NAX025-001 cryocooler [20].

2.4.7 Alt-Az Electronics

The Alt-Az controller (AAC) is responsible for pointing the IRMA within its Alt-Az mount. The AAC, a custom built electronics board designed around the Rabbit Semiconductor RCM2010 [21] (Figure 2.6), connects to the PC/104 over a serial connection through the umbilical cable. It acts as a slave processor to the main computer and its primary function is motor control, which consumes the majority of DIO lines on the AAC Rabbit.

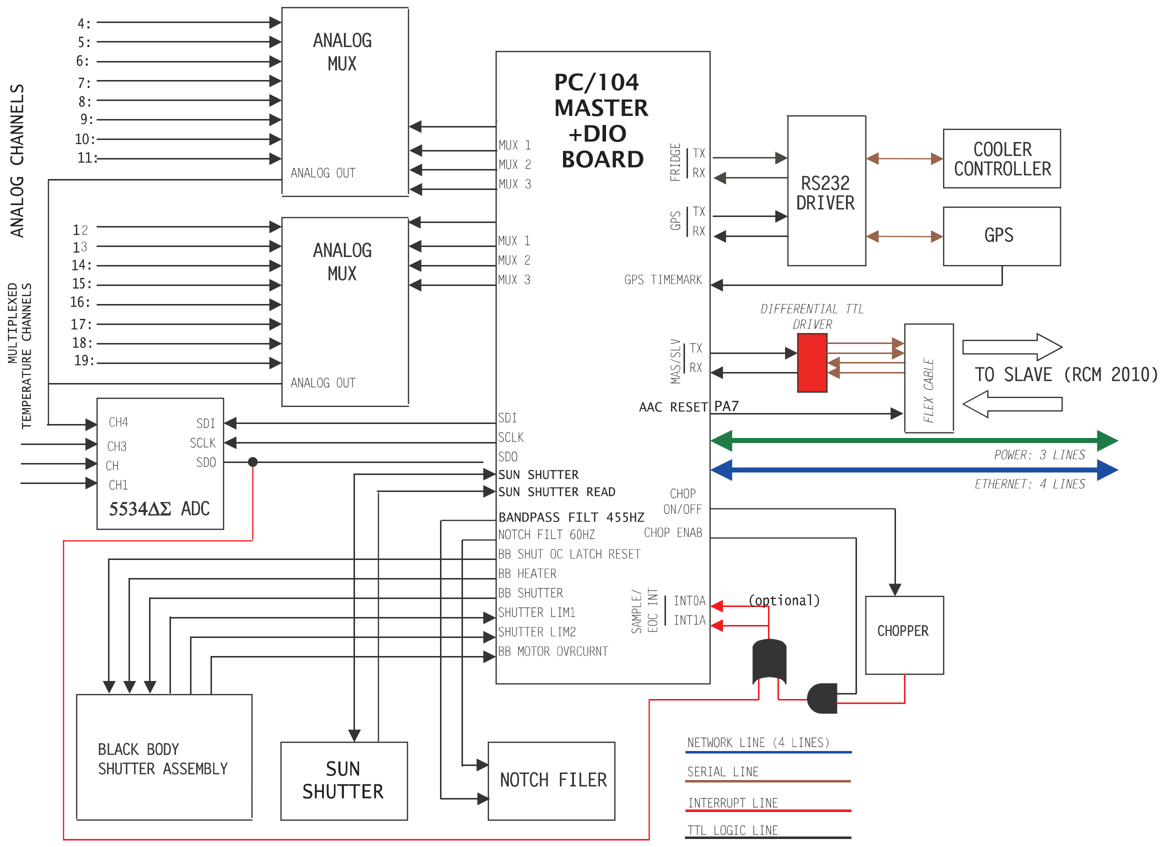


Figure 2.5: A Block Diagram of the new configuration of the IRMA Master Controller showing how the PC/104 interfaces with the various subcomponents of the radiometer.

2.4.7.1 Rabbit RCM2010

The RCM2010 controller module is the control computer that handles motion control and communication for the AAC. It is fitted onto the main Alt-Az electronics board through two 2x20 pin dual-in-line headers. These headers allow the DIO of the RCM2010 to interface with the motor controllers, limit switches, and optical encoders. A block diagram of these connections is shown in Figure 2.7.

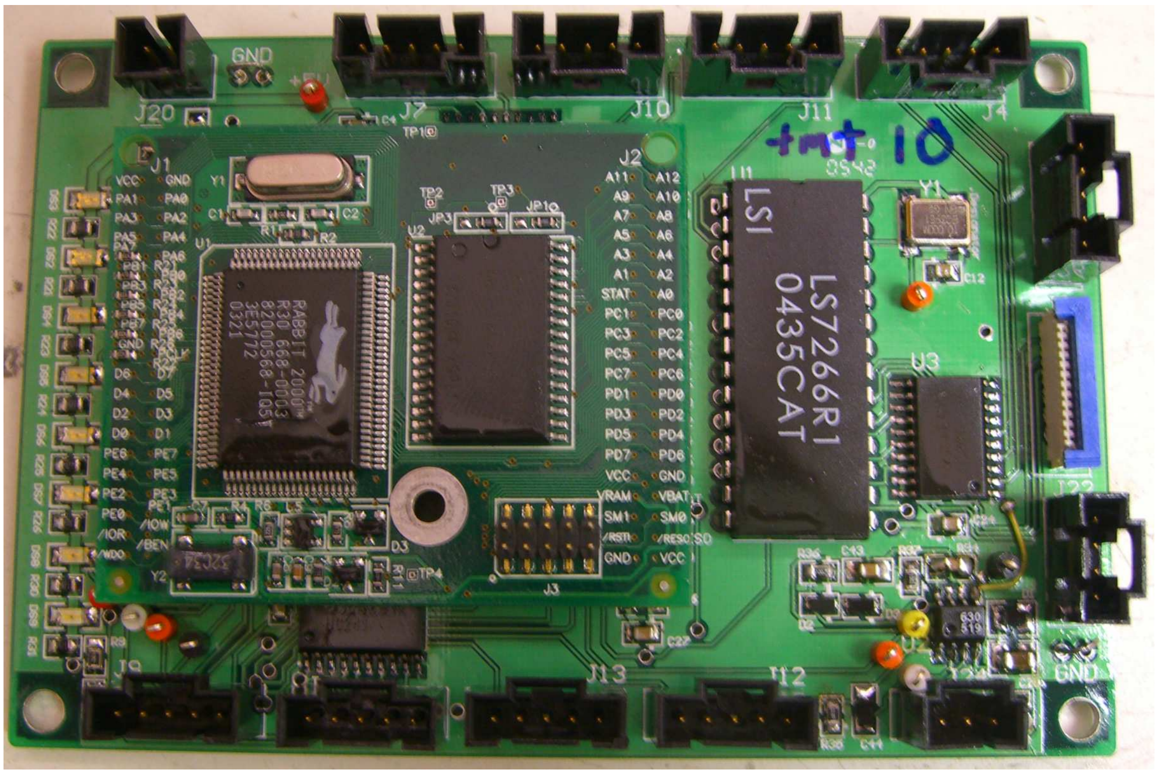


Figure 2.6: A photograph Alt-Az electronics board with Rabbit RCM 2010 microprocessor attached (centre left).

2.4.7.2 Maxon Motor Controllers

As seen in Figure 2.7, the RCM2010 does not connect directly to the Alt-Az motors. Instead, a Maxon 1QEC50V digital motor control unit is used for each motor. Using the control units allows for digital control of the 24 V motors. Altitude and azimuth motor controller enable lines are mapped to output lines 6 and 7, respectively, on parallel port B. Motor controllers are enabled by setting these lines, while clearing these lines disables the controllers. A braking system is also available and used to hold the axes stationary. Braking is applied by setting bits 4 and 5 (for altitude and azimuth respectively) on parallel port A. Altitude and azimuth motor direction is controlled by DIO output lines 6 and 7, respectively,

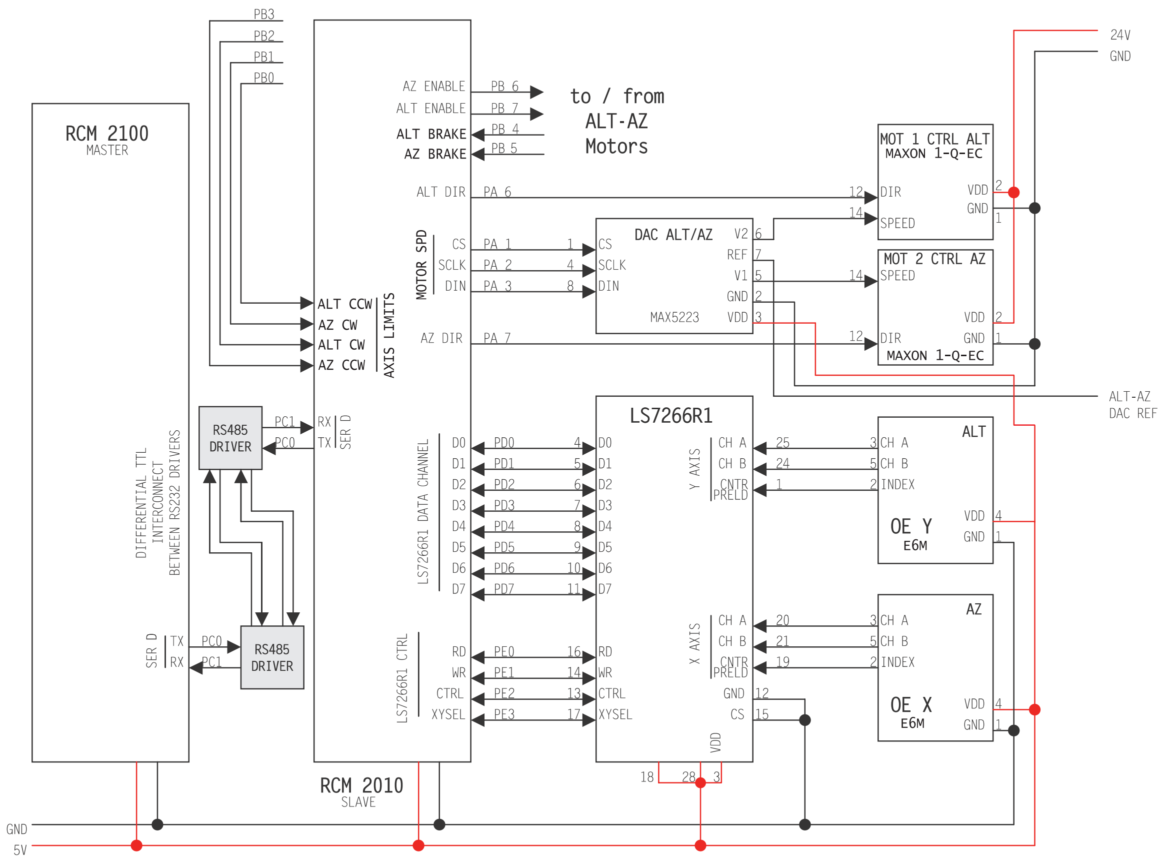


Figure 2.7: A Block Diagram of the IRMA Alt-Az Controller showing how the Rabbit RCM2010 interfaces with motor control system.

on parallel port A. Setting either of these two lines sets the corresponding axis into clockwise (CW) rotation, while clearing puts the corresponding axis into counterclockwise (CCW) rotation. Altitude CW and CCW limits are respectively mapped to input DIO lines 2 and 0 on parallel port B. Azimuth CW and CCW limits are respectively mapped to input DIO lines 1 and 3 on parallel port B. When a limit line is set, a limit has been encountered, otherwise, the axis angle is within safe rotational limits.

The speed of the motor is set by supplying an analog voltage of 0 to 2.5 V to the controller. An 8-bit, 2-channel, Maxim 5223 serial digital to analog converter (DAC) is used to generate the required analog voltage. The 5223 has a 3-wire serial communications interface involving a chip select line (CS), a serial clock line (SCLK) and a data input line (DIN). All the communication lines are mapped to Rabbit parallel port A: CS (active low), SCLK, and DIN, are mapped to pins 1, 2, and 3 respectively. Voltage is individually adjustable on each of the two analog outputs of the 5223, A and B. Voltage can be set between 0 V to full scale (the input reference voltage) in 256 equal steps. Analog output channel A is mapped to the azimuth motor controller, while analog output B is mapped to the altitude motor controller.

Using dip switches on the motor controllers, the speed is limited to a range of 500 to 12,500 RPM. However, the maximum manufacturer recommended rotational speed of the gear box is 8000 RPM, a limit which is set within the software.

2.4.7.3 Optical Encoders

The AAC determines the position of the Alt-Az mount through the use of two US Digital E6M optical encoders [22], which are interfaced with the RCM2010 through a US Digital LS7266R1 encoder chip [23]. The optical encoders employ 2048 lines per revolution optical encoder wheels. However, the LS7266R1 can obtain 8192 lines of resolution per revolution, or 2 arc seconds per encoder step, by operating in quadrature mode. To operate in quadrature mode, a filter clock with a frequency in the range 10 to 35 MHz must be connected to the LS7266 (LS7266 pin 2). The filter clock is supplied by a 10 MHz oscillator. The LS7266R1 detects and counts ticks from the encoders (in either mode), where the count is relative to a software determined location.

The LS7266 optical encoder chip is capable of reading the position from two encoders and interfaces to the RCM2010 using 12 DIO lines. Data sent to, and received from, the LS7266 is carried over an 8-bit bidirectional data bus, mapped to pins 0 through 7 on parallel port D. The software must set the appropriate data direction depending on whether data are being written or read. Since the chip select (active low) line is connected directly to ground, there are four pins used for controlling the LS7266: read, write, control/data and X/Y axis select, connected to pins 0 to 3 on parallel port E respectively. The control/data line (LS7266 pin 13) selects whether data registers (low) or control registers (high) are selected. Read and write (LS7266 pins 16 and 14 respectively) are active low, and are used for enabling reading or writing to the chip. Similarly, the X/Y axis line (LS7266 pin 13) determines whether the azimuth axis counter (low) or altitude axis counter (high) is selected.

The LS7266 requires three input lines from each encoder, two analog inputs, labeled A and B, and an index marker. The azimuth axis encoder is connected to the X axis inputs of the LS7266, while the altitude axis encoder is connected to the Y axis inputs. The A/B inputs for both axes are enabled by setting the A/B input enable lines (LS7266 pins 18 and 28) high. These lines are permanently tied high on the AAC main board. Each encoder contains a unique index mark, which generates a pulse signal when detected. This signal from the altitude and azimuth encoders are connected to pins 1 and 19 of the LS7266 respectively, and can be used to either reset the counter or load a preset value. While the index mark could be used for initialisation, as the mechanical limit switches are currently preferred.

2.5 Summary

The IRMA hardware can be divided into three main components: optics, mechanics, and electronics. Each of these areas have evolved significantly since the first prototype. The optics have been fitted into a compact box, which can either be mounted to the side of a radio antenna or in a specially designed Alt-Az mount. Through the use of custom electronics, embedded computer systems now allow for remote, autonomous control of the IRMA unit.

Chapter 3

IRMA System Software

3.1 History

As the IRMA hardware has evolved, so has its software. In the beginning, the software consisted of a simple C++ program, running on a laptop, which controlled IRMA through its parallel port. Subsequently, a Common Gateway Interface (CGI) was incorporated into the original program so that it could be operated over the Internet with a web browser. Though it was still controlled through the parallel port, this enabled remote operation. However, with the mechanical redesign, IRMA became a more complex system, necessitating a complete software redesign.

3.2 Rabbit based IRMA

With the mechanical redesign, the IRMA became a distributed system running over three processors. As shown in Figure 3.1, these were the Command Processor (CP), the Master Controller (MC), and the Alt-Az Controller (AAC). The CP was a standard desktop PC running Linux, which provided the operator access to the system. The MC, based on an 8-bit Rabbit Semiconductor RCM2100 [17], was located inside the instrument itself. It handled the communication with the CP and interfaced directly with the hardware to control the instrument. The Alt-Az tasks, however, were off-loaded to another Rabbit processor (RCM2010), the AAC, which acted as a slave to the MC. It was located in the Alt-Az mount and was connected to the MC via a serial link through the umbilical cable, which can be seen in Figure 2.3. This system was programmed by a previous graduate student, Ian Schofield, and formed the basis of his Master's thesis [24]. Though most of my work involved the PC/104 based system, it was necessary for me to understand the Rabbit based system, in order to maintain the Rabbit based IRMA units currently deployed. I was also able to make some improvements to this system, the most significant of which was to add a software package [25], which allowed for remote reprogramming of the RCM2100. The Rabbit based IRMA is briefly described below to aid the reader in understanding the current design.

3.2.1 Command Processor

The Command Processor was built around a standard Linux installation. At the core of the CP was the *IRMAscript* interpreter `irmaExec.pl`. Whether commands are sent to the IRMA unit directly through the command line (§3.2.1.3), through the Control Interface GUI (§4.1), or programmed to execute automatically using *AutoTasks* (§3.5), all commands ultimately get sent through `irmaExec.pl`. *IRMAscript* is a custom scripting language used to control the unit. It provides a highly flexible, fine-grained control mechanism for the instrument. A comprehensive description of the *IRMAscript* language can be

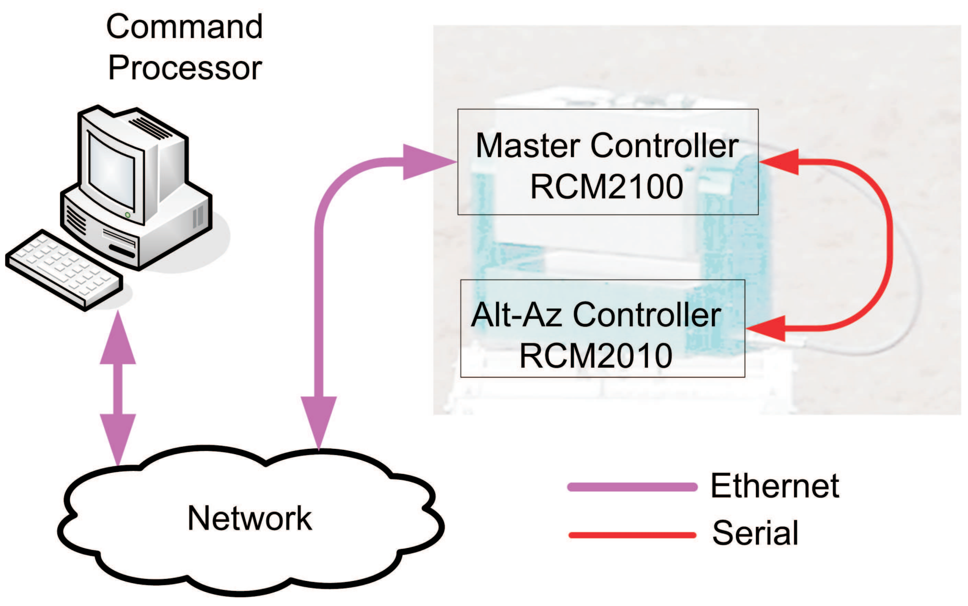


Figure 3.1: A System diagram for the Rabbit-based IRMA.

found in Appendix A.

3.2.1.1 irmaExec Software Structure

`irmaExec.pl`, written in Perl [26], a popular systems programming language, is the interpreter for *IRMAscript*. Interpreters and compilers translate instructions from one form to another. Interpreters typically translate statements into actions as they are encountered, while compilers translate source code into machine language instructions to be executed on the target machine at a later time.

Compilers and interpreters are built using similar mechanisms. A typical compiler consists of a scanner, a parser, a scope checker, and code generation. An interpreter, such as `irmaExec.pl`, is also built using these components, except that it will typically execute the statements rather than generating code.

The scanner is responsible for recognising tokens or strings of a language statement. `irmaExec.pl` accomplishes this by reading in the source file, ignoring white space and comments, and separates each line into tokens. Tokens may be either literals, strings, variables, constants, or reserved words, words which are commands of the language. The statements of each line are then stored in an associative array, or hash table, using the generated line number as the key to access the statements.

The parser is responsible for determining if a sequence of tokens conform to a language statement. Since *IRMAscript* uses a finite automaton, or finite state machine, and does not allow for nested statements, the parser in `irmaExec.pl` is greatly simplified. Basic syntax for hardware commands consists of 3 tokens:

[Command] [Modifier field 1] [Modifier field 2]

These are followed by n parameters, as needed. This structure helps make the language more readable and also allows the commands to be divided into families. For example, the commands relating to the chopper wheel are shown below.

```
CHOP READ STATE
CHOP STATE ON
CHOP STATE OFF
```

The three token hardware commands are converted into 3 digit command codes and embedded in network command packets. The command set is stored in an Excel spreadsheet which is parsed using the Perl module `Spreadsheet::ParseExcel`. The parser stores variables in a hash table and also handles nested loops.

Scope checking and type checking are generally combined in a compiler. Since *IRMAscript* is typeless, and all variables are considered global, that is, visible throughout the program, this stage is not included in the interpreter.

Table 3.1: pseudo code description of the the IRMAscript interpreter.

```
initialize command code hash table

open irmascript file (read)
do
   read line from file
   split line into fields, put into array
   put array in source code hash table with key = line count
   increment line count
until reach EOF

initialize program counter "pc" to 0

do
   get statement from source code hash table using key = pc
   pattern match statement on command, modifier1 and modifier2
   look up command code using keys command, mod1 and mod2
   make command packet
   send command packet to MC according to network comm protocol
while pc < total lines in program
```

A compiler translates the verified language statements into lower level, machine readable codes. This code may then be executed on the target machine at a later time. By contrast, interpreters execute the statements on the target machine immediately after they are parsed. For `irmaExec.pl` this involves either generating a binary packet to send over the network to the MC, or, in the case where a statement does not command the IRMA hardware, such as a variable assignment or flow control, the interpreter will execute the statement directly within the `irmaExec.pl` process.

The basic functionality of the *IRMAscript* interpreter can be described by the pseudo code in Table 3.1.

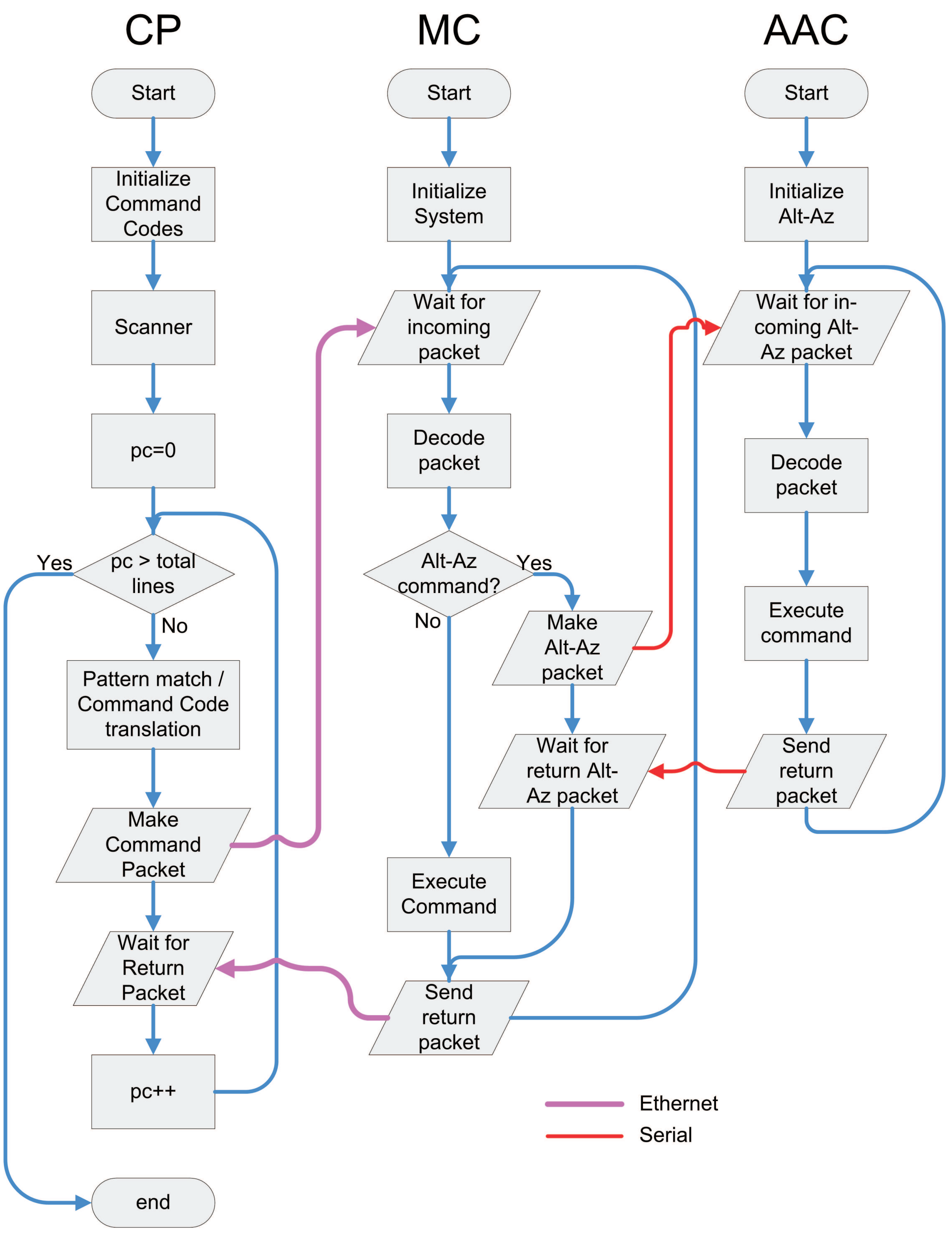


Figure 3.2: A flowchart of Rabbit-based system software.

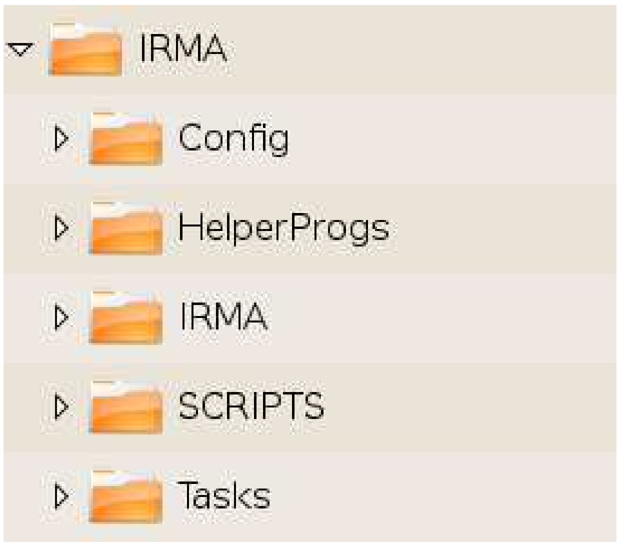


Figure 3.3: An overview of the IRMA directory structure.

3.2.1.2 Directory Structure and Configuration Files

The software requires the defined directory structure shown in Figure 3.3. The base directory for the software is named `IRMA` and located in the users home directory, usually denoted `~/`. Within the base IRMA directory, there are four subdirectories: `~/IRMA/IRMA/`, where the custom IRMA Perl modules are located, `~/IRMA/Config/`, where configuration files are stored, `~/IRMA/SCRIPTS/`, scripts to be executed with `irmaExec.pl` must be located in this directory, and finally `~/IRMA/HelperProgs/`, where `irmaExec.pl` is located.

The interpreter is initialized via configuration files. The first is the command set, which is stored as an Excel file and shows the association of each command with is corresponding numeric code. By default, is is named `IRMAscript.xls` and is located in the `~/IRMA/HelperProgs/` directory.

In addition, each IRMA unit has a unique configuration file, `box_<box number>.cfg`, stored in the `~/IRMA/Config/` directory. The configuration file shown in Table 3.2 provides

Table 3.2: An example unit configuration file which is used to initialise instrument parameters.

```
**************************************************
2006-01-01T00:00:00
IPaddress 128.171.116.72
Data_port 10072
Cooler TR282
Board 1
# Dummy calibration data for this time period
CalibrateLow 77_7.74e6
CalibrateHigh 319_6.82e6
**************************************************
2006-08-09T15:00:00
# Unit returned from Hawaii
IPaddress 142.66.41.40
ElevGearReduction 128
AzimGearReduction 128
BeltReduction 8
MinMotorRPM 500
MaxMotorRPM 25000
MaxGearRPM 8000
elev_kProp 10.0
elev_kInteg 1.0
elev_kDeriv 1.0
azim_kProp 1.0
azim_kInteg 1.0
azim_kDeriv 1.0
**************************************************
```

the CP with the IP address and data port of the master controller, and supplies the gear reduction ratios, servo parameters and detector calibration constants to the MC. The file is broken into parameter blocks, which are delimited by lines of repeating asterisks. A time stamp appears at the head of the block, which establishes the date/time when the parameters, immediately following, took effect. The parameters within a block include all lines following the time stamp, up to but not including the next block delimiter line. A parameter line consists of a label followed by a value, and is terminated with a carriage return. A whitespace separates the label from the value.

When `irmaExec.pl` is executed it reads in the box file specified by the box number command line parameter, accepting parameter fields whose time stamp is closest to the current time/date. For example, using the configuration file in Table 3.2, if `irmaExec.pl` were executed on 2006-09-01, it would accept the IP address values from the command block dated `2006-08-09T15:00:00`, because this is the most recent entry.

For a complete list and description of parameters that can be defined in a CP configuration file see Appendix C.1. If parameters are not defined, default parameters are assigned.

3.2.1.3 CP software requirements and execution

As described earlier, the system is designed to run on a desktop or server PC running Linux. All CP software has been designed to run with Perl 5.8.6 and later. In addition to having the Perl interpreter installed, additional Perl modules are also required. Appendix C.2 lists the modules, and installation instructions.

Scripts are executed on an IRMA unit by issuing a command following the form:

```
./HelperProgs/irmaExec.pl <box number> <IRMAscript name> [IRMAscript.xls]
```

where `<box number>` is the assigned unit number for the IRMA, and `<IRMAscript name>` is the name of the script to be executed. This script must be located in the `~/IRMA/SCRIPTS/` directory. The `IRMAscript.xls` is shown in square brackets to show that it is optional. It directs the interpreter to the Excel file which defines the formal language of *IRMAscript*. Unless the file is renamed it is not necessary to include it as a parameter, since the interpreter will use the default filename.

As an example, if an operator wanted to open the weather shutter on IRMA unit 1 and had written a script named shutterOpen.irma, he would copy the script into the `~/IRMA/SCRIPTS` directory of the CP for IRMA Unit 1. He would then execute the following command from the base IRMA directory `~/IRMA/`:

```
./HelperProgs/irmaExec.pl 1 shutterOpen.irma
```

3.2.2 CP - MC Communication

The binary network packets generated by `irmaExec.pl` are sent over the network to the MC. Due to the fact that the CP - MC communication is over Ethernet, transmission could not be guaranteed. For this reason a communication structure based on the European Space Agency (ESA) Packet Telecommand Standard [27] was used. This protocol is shown in figure 3.4. After the command is received, the MC responds by sending an acknowledgment packet, a packet indicating the requested activity has begun, a data packet (if applicable), and finally a packet indicating the requested activity has concluded. For the in-

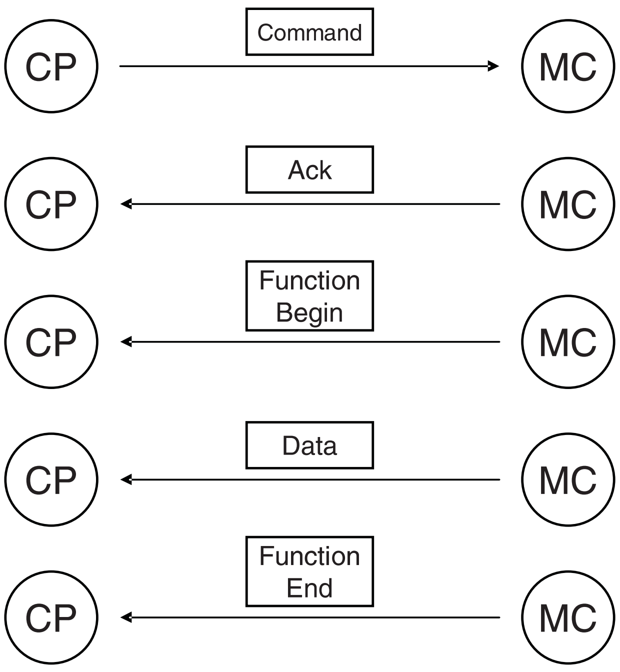


Figure 3.4: A summary of the IRMA network communications handshaking sequence.

terpreter, this funtionality was encapsulated in a separate Perl module `IRMA::PacketComm`.

3.2.3 Master Controller

The Master Controller (MC) is a real-time multitasking program using the MicroC/OS-II real-time kernel. In this type of program, high-priority tasks perform their tasks in short bursts, then sleep for a defined interval or block on an event, in order to open up time in which the next lower-priority tasks can execute. The order of execution continues down the priority hierarchy until all the tasks have completed.

Task priority was assigned according to the degree to which a task can tolerate being preempted. In priority-based preemptive multi-tasking, tasks can only preempt other tasks having lower priorities than themselves. It is important to determine which tasks can be preempted and for how long, giving the most critical task the highest priority. All tasks are subject to preemption if an interrupt service routine (ISR) is present. This should not pose a problem, however, since ISRs, should execute and return to the interrupted process as fast as possible.

The task structure of the MC is shown in Figure 3.5. The dispatcher task is the most active task within the MC, and is primarily concerned with receiving commands from the CP. Since the majority of commands are classed as short-duration, meaning they take less than a second to execute, they are allowed to execute within the dispatcher task. Long duration commands such as scans, however, must be executed outside the context of the dispatcher task, otherwise the dispatcher would be unable to accomplish its primary duty of listening for incoming commands until the task was completed. When a scan is requested, the dispatcher task forks the scan task (as shown in Figure 3.5), then returns to wait for

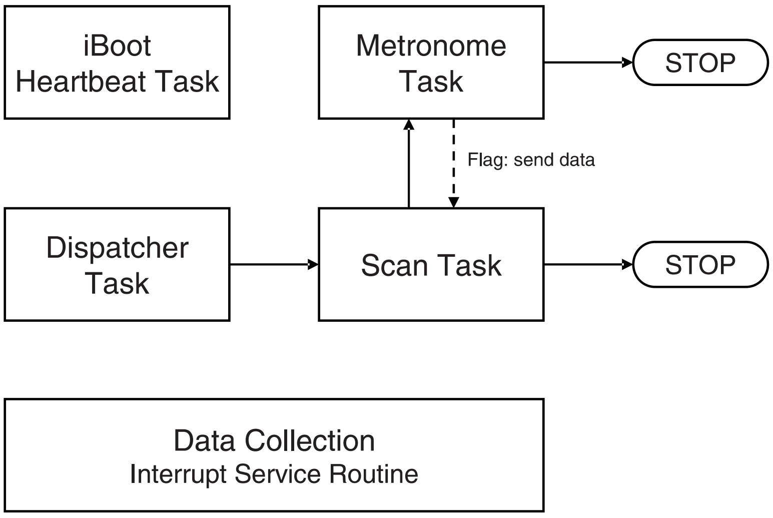


Figure 3.5: A schematic of the IRMA master control software task structure during scanning.

incoming commands. This allows short duration tasks to run concurrently with the long duration scan task.

The scan task is responsible for the data collection process. It also handles the construction of data packets and their transmission to the CP. The scan task starts the metronome task, and enables the ISR to trigger on the external interrupt provided by the chopper blade. The metronome, whose priority is greater than the scan task, counts the data points collected by the ISR, fetches the current Alt-Az coordinate from the AAC for each data point, and signals the scan task (by means of an event flag) to construct and transmit the data packet when a full frame of data are collected. Once the scan task has successfully transmitted the data packet, it returns to wait on the data transmit event flag, shown as a dotted arrow in Figure 3.5. When data collection is terminated, the scan task and metronome task are both instructed to terminate themselves.

3.2.4 MC - AAC Communications

If the dispatcher task of the MC receives an Alt-Az command, it forwards the command onto the slave AAC. The MC communicates with the AAC over a 2-wire serial link operating at 19.2 kbit/s, which travels through the umbilical cable linking the IRMA to the Alt-Az base. While it was originally planned to use a similar communication protocol as the CP - MC Communications, this was not possible due to the limited bandwidth of the serial communications. Therefore, the communication protocol needed to be significantly simplified. Serial packets were transmitted as a 6-field colon delimited ASCII character string, consisting of a command field, four data fields, and a CRC checksum field, encapsulated between STX (start transmission) and ETX (end transmission) characters. An example of a serial communications packet string appears in Figure 3.6. The MC - AAC Communication has remained the same for the PC/104 based IRMA.

STX	CMD	:	Field A	:	Field B	:	Field C	:	Field D	:	CRC	ETX

Figure 3.6: The description of the IRMA serial communications packet string.

The MC - AAC communication protocol uses simple handshaking. The MC sends the serial communications packet to the AAC, which then converts it from ASCII to binary, and performs the function. Once the task was complete, the AAC responds with the same packet, except with data fields populated, if applicable.

3.2.5 Alt-Az Controller

The AAC software exists as a bootable software image that resides in the flash memory of the Rabbit microcontroller. Similar to the MC, MicroC/OS-II real time kernel

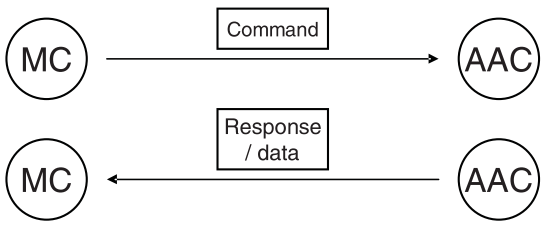


Figure 3.7: IRMA serial communications protocol between the Master Controller and the slave Alt-Az Controller.

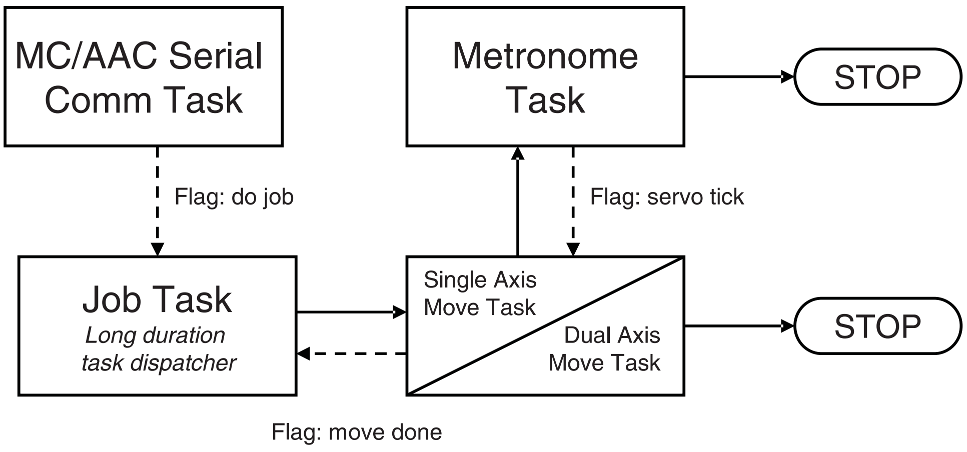


Figure 3.8: A schematic of the IRMA Alt-Az controller task structure.

was used to provide a real-time, multitasking environment, which is required for the servo control loop. The task structure of the AAC, shown in Figure 3.8, is very similar to that of the MC. The serial communication task, analogous to the MC's dispatcher task, waits for commands from the MC. It is a real-time task devoted to monitoring serial data traffic. When it reads a STX character, it proceeds to read up to 80 characters or when an ETX character is encountered. If the packet is received without error, the command is decoded and executed. For a complete list of the possible AAC commands and their associated integer codes see Appendix B. Most AAC commands are short-duration, such as querying the current position, and therefore, are executed within the communication task. Long-duration tasks, such as axis movements, must be run in their own task, otherwise serial communication would not be possible until the task was complete. The job task waits for an event flag to be set by the serial communication task. Once the flag is received, it carries out the desired task in parallel with the communication task, such that short duration task can still be handled. If axis movement is requested, the job task starts either the single axis move task or the dual axis move task depending on the request.

To ensure accurate speed control, a proportional-integral-derivative (PID) servo tracking loop is used. The axis move task waits on a 10 Hz servo tick event flag, set by the metronome task, the highest priority task of the AAC to ensure accurate timing, which signals the move task to update the servo loop calculations. The servo loop must be updated exactly at this rate for the servo control algorithm to function correctly. The ability to meet deadlines in a multi-tasking environment is the defining attribute of real-time programming [28]. Once the movement has completed, the move task sets an event flag, which signals to the job task that the axis rotation has completed. The metronome and move axis tasks then suspend themselves, waiting on another event flag to signal a new request.

It was necessary to introduce a second mode of axis movement to handle extremely slow movement, that is, movement slower than 500 RPM, which is the minimum speed which can be selected with the motor controller (§2.4.7.2). This second movement mode is referred to as slewing, meaning long range motion around the altitude or azimuth axis. Slews have the ability to perform periodic steps over a long period of time, thus lengthen the time to rotate from the initial to destination angle. The serial communication task signals the job task to wake up and start the appropriate axes control tasks. Each axis movement is controlled in its own task: one exists for altitude movement, and another for azimuth. Both axis tasks can be run concurrently, which requires that they each have unique priority levels. Since both tasks cannot have the same priority, the slew elevation task has a slightly higher priority than the azimuth slew task. Given that skydip operations, discussed in §5.2, which involve slewing the altitude axis, are performed more often than azimuth movements, preference was given to elevation movements. Slew tasks and the servo move tasks have priority levels that place them below the metronome task priority, but above the serial communications task, and the job task.

When a task is completed successfully, the AAC will respond with the same packet requesting the task, except with the command field populated with the MSCOMM_SUCCESS code (integer value 100) and, where applicable, data is returned in the four data fields, and a checksum value is calculated and placed in field 6. If an error occurs in the communications the command packet is populated with the MSCOMM_FAILURE code (integer value 101), and each of the four data fields are populated with the associated error codes.

3.3 PC/104 based system

Though the Rabbit based system worked well, it was limited by its requirement of the CP. If the network connection between the CP and the MC was lost, the MC would be unable to receive commands and would not be operational. To overcome this problem the 8-bit Rabbit microcontroller of the MC was replaced by a 32-bit PC/104. The PC/104 is a true IBM PC compatible computer capable of running a standard desktop operating system. IRMA uses the Slackware 10.2 Linux distribution. With the increased computing power of the PC/104, a separate CP was no longer necessary, and was merged with the MC.

3.3.1 Porting the Master Controller

The first step to running IRMA with a PC/104 was to rewrite the MC for execution on the PC/104. Ian Schofield started the task of porting the original code, written in Dynamic C® [29], a proprietary C variant by Rabbit Semiconductor, to ANSI C. The code was organized by having a separate file encapsulate the functions for each command family. These functions were called by the dispatcher function which was responsible for interpreting the binary packets sent by the CP. The main function simply initialized the system and opened a socket to listen for incoming command packets and forward them to the dispatcher. This code was compiled into a single executable named `irmamc`.

Originally, the PC/104 based MC was a direct port of the Rabbit based MC, and the CP software was simply installed on the PC/104. This meant that the *IRMAscript* interpreter was still encapsulating the commands in binary packets and sending them to the MC which was now located on the same machine. Also, the same communication protocol was still used even though the packets were not being sent over the network. The complexity of the distributed system was being carried over to the PC/104 based system.

Since the CP and MC were merged into one processor, it was no longer necessary to encode commands with the interpreter, send them over a TCP socket, and decode the commands and execute them on the MC. Additionally, PC/104 based systems do not have the same resources of a standard desktop PC so the additional overhead caused delays on the order of $\sim$10 seconds in executing commands.

Running a distributed system on a single processor is inefficient as shown by the execution delay. To optimise the performance of the embedded system, I ported `irmaExec.pl`, which was written in Perl and designed to run on a powerful desktop PC, to C and then incorporated it into `irmamc`. The C version of the interpreter followed the same design as described in section 3.2.1.1. The scanner was written as a separate function called `readScript`. Receiving a string containing the name of the script to be tokenized, the function read the script from the `~/IRMA/SCRIPTS/` directory and removed any comments and additional whitespaces and returned a two dimensional array, where one dimension was the line number and the other was the individual tokens in the line. This array was then passed to the parser (`runScript`),which is identical in functionality to the parser in `irmaExec.pl`, except that instead of generating a network command packet when a command was to be issued, `runScript` simply executes the function to perform the desired task directly.

One difficulty in porting `irmaExec.pl` into C, was the handling of typeless variables. As mentioned *IRMAscript* is a typeless language. In the original implementation this was trivial since the underlying language, Perl, is also typeless. To handle this in C additional checking had to be performed to determine the type of the variable. The underlying variables were then stored as either strings or double precision floating point numbers, which is similar to the method Perl uses for typeless variables.

Merging `irmaExec.pl` into `irmamc` dramatically reduced the complexity of both programs as the network packet handling functions could be removed. Also, since the Perl interpreter was no longer need, the compiled C code executed much more quickly. As a result of these improvements scripts now start executing over an order of magnitude faster than the previous configuration, which brings an important real-time response to the Operator's

Console. Moreover, a reduction of ∼11,200 lines of code of the original 68,000 lines of code was achieved.

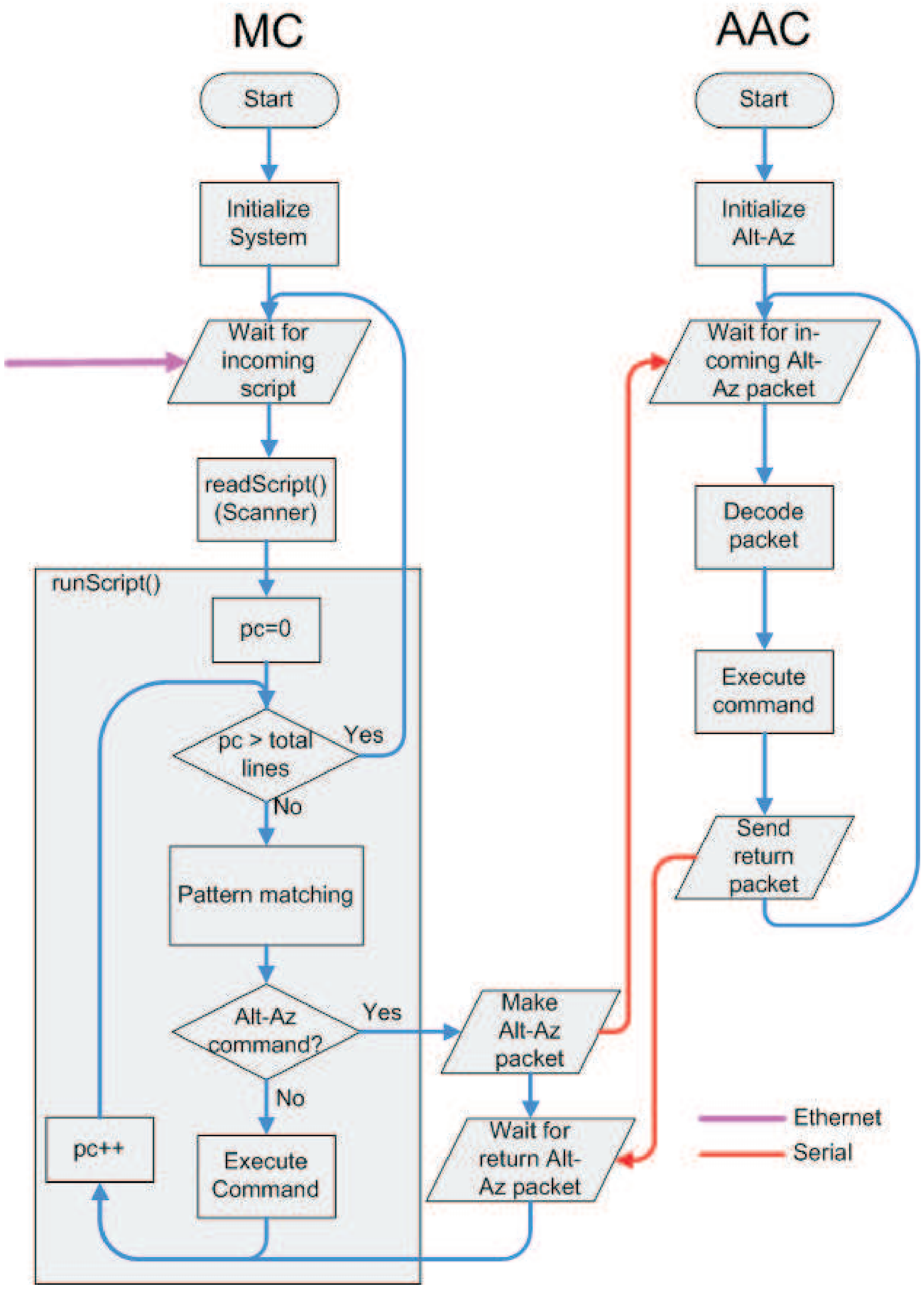


Figure 3.9: A Flowchart of PC/104-based system software.

Scripts are still executed in the same form as described in §3.2.1.3. However, the new `irmaExec.pl` simply passes the script name to `irmamc`, and outputs the returned results.

3.3.2 Autonomous Control

Running embedded Linux allows for increased autonomous control of the IRMA. When IRMA is operating at remote sites it is important for it to be able to recover from power failures without intervention. Through the startup file, `/etc/rc.d/rc.local`, the system is initialised. First it verifies if the Alt-Az mount is initialised, and if not it performs the initialisation. For the case where the PC/104 is simply rebooted the AAC will not be affected and therefore will remain initialised. After initialising the mount, the unit moves to the `Park` position while it checks the current weather conditions. If favourable conditions exist, the unit will point up to zenith and resume normal operations. Additional autonomous control is handled through *AutoTasks* §3.5.

3.3.3 AAC Updates

Though the MC processor was replaced with a PC/104, the AAC remained Rabbit based. Therefore, much of the original AAC software remained intact. However, I identified several upgrades, which made the AAC run more efficiently.

3.3.3.1 Faster Initialisation

During initialisation, the previous version of the AAC would seek both the counter-clockwise, and the clockwise limits. This made initialisation a time consuming task; taking up to 10 minutes to complete. Since the physical distance between the limits remains constant once the unit is built, it is only necessary to seek one limit and load the predetermined distance to the other limit. Also, the original software was only capable of initialising one axis at a time. An additional function was written to allow for a dual axis initialisation, which further improved the startup time.

3.3.3.2 Software Initialisation

In some cases, such as after a power cycle, the position of the Alt-Az is known to the operator, but the unit has not been initialised. In these cases it would be desirable to be able to load the position directly without having to perform an initialization. This additional functionality was added to the latest version of the AAC software.

3.3.3.3 Uninitialised movements

Previously, the AAC was designed to accept movement commands only after the Alt-Az mount was initialised. This was due to the fact that the input parameters to these commands were required to be an absolute position, i.e. an elevation of 30°. However, if a limit switch fails, the IRMA unit will be unable to complete the initialisation. For this reason, functionality was added which would allow the operator to move to a relative position, i.e. increase elevation 20°. These additional commands provided invaluable flexibility, and proved useful when problems were encountered in the field, see §5.2.

3.3.3.4 Offset handling

The AAC is capable of applying offsets to each direction to allow the unit to adjust the value to North in the azimuth direction and the horizon for elevation. The position data are then returned in accurate altitude and azimuth coordinates. In the Rabbit based system, the values were set in the AAC, but could not be queried. A copy of the values were stored in the MC, and it was this copy that was returned when the value was queried. In the PC/104 based unit, the MC is not as closely coupled to the AAC. Since `irmamc` is simply a program running under Linux on the MC, it may restarted or the computer may be rebooted without affecting the AAC, which would cause incorrect offset values to be returned. The MC was modified to forward Alt-Az offset queries to the AAC, which would respond with the current value.

3.4 Queue Server

The Rabbit based IRMA could only handle one TCP connection at a time. If a second command was sent while the first was executing, the second command would receive a `"Connection Refused"` error. This problem was solved by writing a non-proirty queue server. Programs which wanted to send a command to an IRMA unit would first submit the request to the associated queue server. The queue server would then notify the program when it was free to execute the *IRMAscript*. This was only of limited use, however, because any commands executed directly from the command line would bypass the queue server and potentially cause `"Connection Refused"` errors.

With the increased flexibility of the PC/104, `irmamc` automatically queues up to 10 simultaneous commands in the TCP stack, making the functionality of the queue server redundant. At present, the queue server is still included on the PC/104 based system because it was very tightly integrated into some of the software such as *AutoTasks*, which will be discussed in the next section.

3.5 AutoTasks

AutoTasks helps the unit run more autonomously, by monitoring the system and executing tasks.

3.5.1 Software Structure

AutoTasks is written entirely in the perl scripting language. It is designed to run in the background as a service. As shown in Figure 3.10, *AutoTasks* is simply a loop which performs various tasks at defined intervals. These tasks include: weather monitoring, cryocooler logging, daily tasks, and skymaps. Each of these tasks are described below.

3.5.1.1 Weather Monitoring

The IRMA unit must be able to operate over a wide variety of environmental conditions. The weather must be continually monitored. If the weather becomes unfavourable, as determined by the relative humidity, the unit will close the weather shutter and move to a parked position. The relative humidity threshold is set in the `autoTasks.conf` file with the `humidity humid` parameter. If an external weather station is available at the site a special function can be added to the `IRMA::Weather` module to parse the output. This information is then read at a specified interval. If external weather sources are unavailable the unit can be configured to use its own on board humidity sensor.

3.5.1.2 Cryocooler Logging

The cryocooler must be observed constantly to ensure that it is maintaining a proper temperature. It is also important to record the amplitude of the cooler drive as

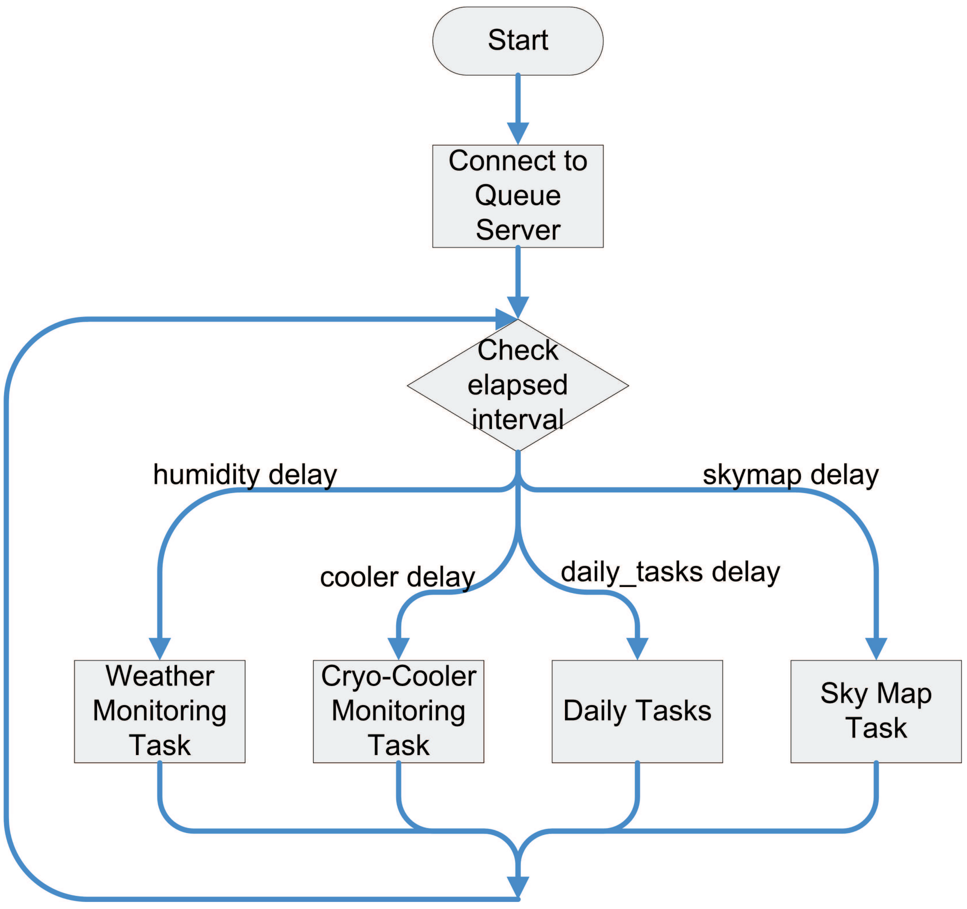


Figure 3.10: A flowchart of the AutoTasks execution path.

this may lead to early detection of a vacuum leak. It is possible to monitor the cooler temperature, drive amplitude and oscillator frequency with the cooler monitoring function by setting their respective parameters within `autoTasks.conf`. The cryocooler logging function is executed at a time interval specified in `autoTasks.conf`, and the information is be stored in a cooler statistics file.

3.5.1.3 Daily Tasks

AutoTasks also allows for the scheduling of scripts. This provides functionality similar to `cron` [30] on linux systems. At a regular time interval set by `daily_tasks delay` in `autoTasks.conf`, *AutoTasks* will check if a scheduled task is ready to run. Task file must be placed in the `~/IRMA/Tasks/.box_<box no>/` directory. *AutoTasks* will first check if a special task file for the current day is present, which is determined by the filename, i.e. `2007-05-12.task`. If no such file is present, the file `default.task` will be used. Any changes to the task file while *AutoTasks* is running will be detected automatically and the updated scheduled tasks will be loaded. A sample daily task file is included in Appendix C.3.

3.5.1.4 Sky Maps

Sky maps are obtained by scanning the entire sky with the IRMA unit to provide an all-sky map of the PWV. Though skymaps have been performed manually in the lab, the process has not yet been automated in *AutoTasks*. It is present in *AutoTasks* as future development.

3.5.1.5 autoTasks.conf

The main configuration file for *AutoTasks* is `autoTasks.conf`. The different tasks within *AutoTasks* can be activated by setting their respective `on` parameter to `1`. How often the task is performed is given in seconds by the `delay` parameter. A sample configuration file is shown in Appendix C.4.

3.6 Housekeeping Tasks

There are various other smaller scripts which assist IRMA. As mentioned in §1.3.2, to determine PWV the local temperature and pressure are required. Using the same Perl module used by the weather monitoring task, `IRMA::Weather`, a script, `logger.pl`, was written to generate a weather file, which records the local temperature, pressure, and humidity. This file is stored with the data files. The default directory is

```
/IRMAdata/IRMA_<Box_number>/<year>/<year>-<month>-<day>/
```

Due to the limited amount of storage space available on the Compact Flash drive, 1 GB, data must be copied to a remote server and removed from the IRMA unit before the available storage space is exhausted. The program `data_sync.pl` was written to handle this task. It synchronizes the data to a remote server using `rsync` [31]. Then, if the `rsync` was successful, it removes any IRMA data more than one day old.

Both of these tasks are executed on a regular basis using the `cron` [30] scheduler.

3.7 Summary

The IRMA control and communication software has evolved significantly over the course of the project. From its humble beginnings, when IRMA was controlled through the parallel port of a laptop, the system has evolved into a complex distributed system involving three processors, a PC and two 8-bit Rabbit Semiconductor microprocessors. I joined the project and was able to successfully deploy the latest generation of IRMA based on a PC/104. This latest system added significant computing power and flexibility to the unit by integrating a 32-bit processor, capable of running a standard Linux operating system. At the same time the complexity of the overall system was reduced by eliminating the need for an external PC. Improvements were also made to the software to make it more robust and autonomous.

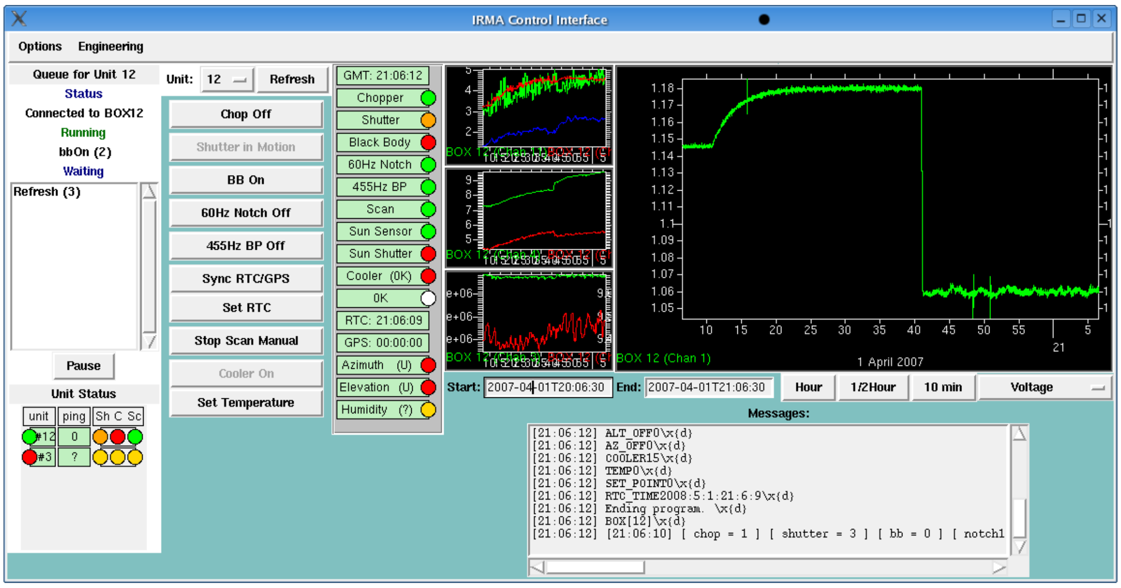
Chapter 4

IRMA Control and Data Analysis Software

The IRMA control and data analysis software resides on a remote computer. This software allows the operator to control any IRMA unit and to view and interpret the collected data.

4.1 IRMA Control Interface

In order to control the IRMA unit, the operator has the option to either connect to the unit remotely via `ssh` and run desired commands from the command line, as described in §3.2.1.3, or use the *IRMA Control Interface*. The *Control Interface*, shown in Figure 4.1 provides a simple, intuitive graphical user interface (GUI) to the unit. The GUI was written by a previous undergraduate student, Ms. Amy Smedes, and is able to provide real-time status information for multiple IRMA units. It also allows for basic control of the unit. For more complex controls, however, the operater must write the desired functionality in



Start: 2007-04-01T20:06:30 End: 2007-04-01T21:06:30

```
[21:06:12] ALT_OFF0\x{d}
[21:06:12] AZ_OFF0\x{d}
[21:06:12] COOLER15\x{d}
[21:06:12] TEMP0\x{d}
[21:06:12] SET_POINT0\x{d}
[21:06:12] RTC_TIME2008:5:1:21:6:9\x{d}
[21:06:12] Ending program. \x{d}
[21:06:12] BOX[12]\x{d}
[21:06:12] [21:06:10] [ chop = 1 ] [ shutter = 3 ] [ bb = 0 ] [ notch1
```

Figure 4.1: A screen shot of the IRMA Control Interface GUI.

IRMAscript and execute it from the command line §3.2.1.3 or through *AutoTasks* §3.5. After Ms. Smedes finished her work term, I learned the code and was responsible for maintaining it; I subsequently improved the performance such as improving the data handling routines for the graphs, and allowing for more flexible cryocooler monitoring.

4.1.1 Software Structure

The *IRMA Control Interface* is a high level control and monitoring program written in Perl/Tk. Tk is a powerful graphical toolkit which has been ported to Perl, a popular systems scripting language, for the rapid development of graphical interfaces. Various preprogrammed graphical elements, called widgets, are included, such as control buttons, text boxes, radio buttons, etc. Like most GUI programs, Perl/TK programs are *event driven*; meaning a main loop waits for *events*, such as clicking on a button, to occur and then

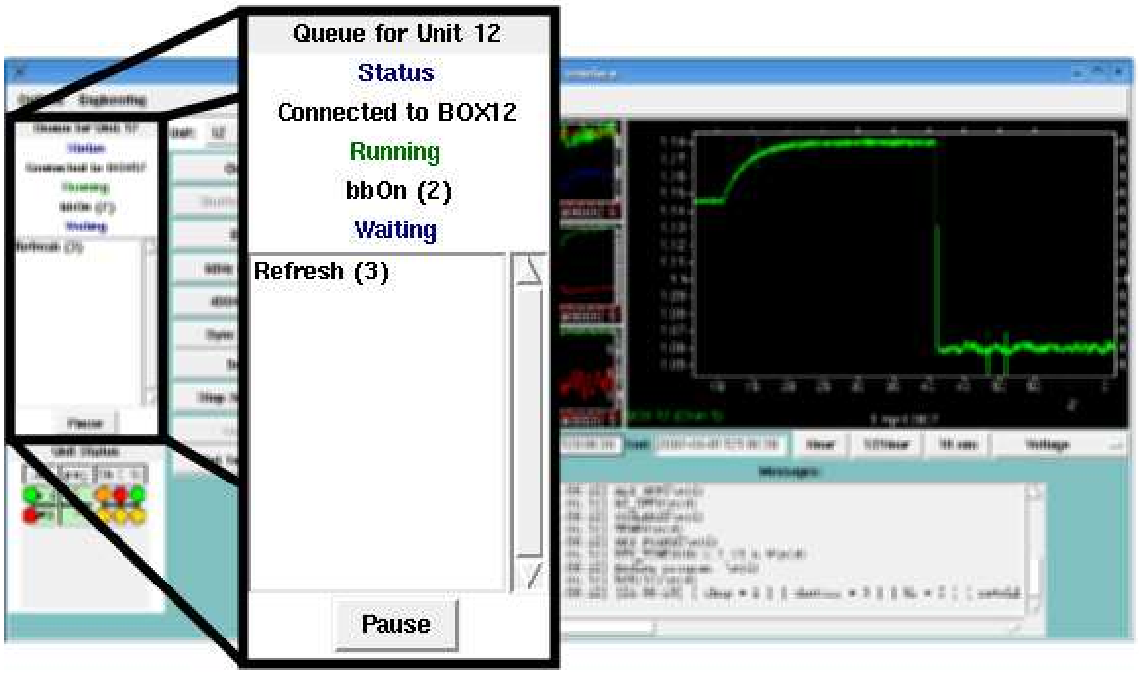


Figure 4.2: A screenshot of the Queue Status area of the IRMA Control Interface showing the blackbody ON command running with the Refresh command waiting in the Queue.

executes associated funtions called *callbacks*. Some events are handled automatically within the Tk core, while others are associated with custom written callback routines [26].

The primary purpose of the IRMA Control Interface is to provide remote, real-time status monitoring and control of the IRMA unit. The *IRMA Control Interface* can also be used to monitor the data from the unit, and setup and control *AutoTasks* remotely. The functionality of the *Control Interface* is described in this section.

4.1.1.1 Queue Status

The *Queue Status* area is shown in Figure 4.2. Here, the operator is able to view the connection status of the selected unit. Also shown, is the script currently executing, as well as, a list of scripts waiting to execute. For example, in Figure 4.2, the executing

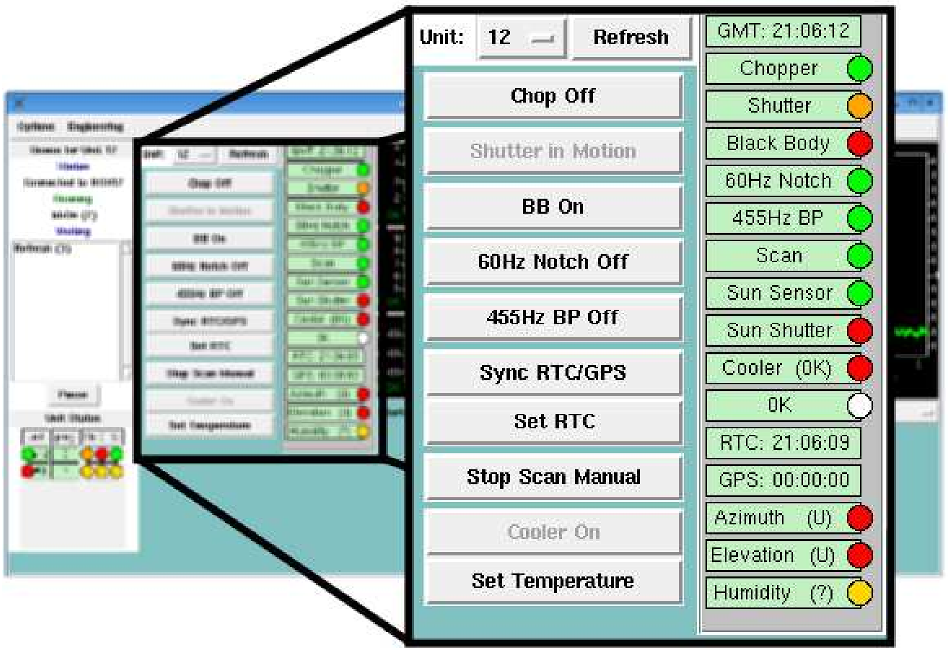
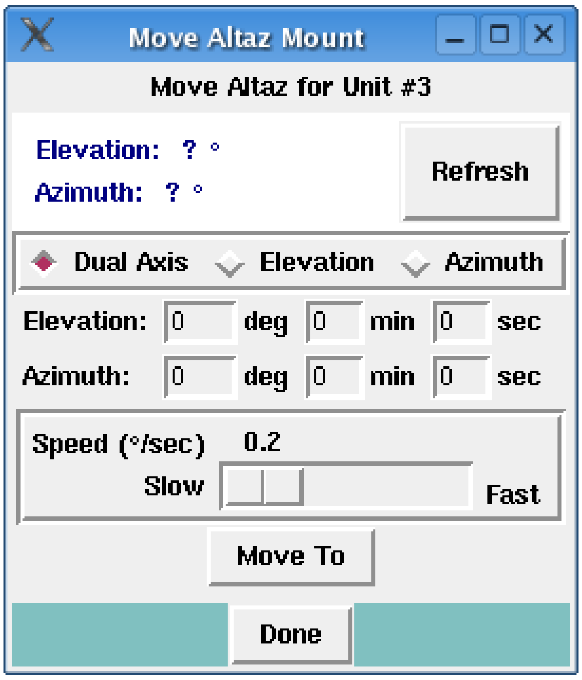


Figure 4.3: A screenshot of the Current unit status area of the IRMA Control Interface showing the current status unit showing the chopper on, the shutter moving and the blackbody off.

script is `bbOn`, and `Refresh` is waiting to execute. The list of scripts waiting to execute includes scripts which were initiated by the current operator, an operator at another location controlling the same unit, or scripts executed automatically by *AutoTasks*. IRMAscripts executed directly from the command line, however, will not be listed. The operator is able to remove any queued items by right clicking on the item and selecting `remove`.

4.1.1.2 Current Unit Control and Status

The *Current Unit Control and Status* is located to the right of the Queue Monitor as shown in Figure 4.3. The unit is selected from the drop-down list located at the top of this section. This selection also affects the Queue Status and Data Monitoring sections. Control of the unit is accomplished by use of a column of command buttons; one for each function. The function of the button is determined by the status of the Unit. For example,



Move Altaz for Unit #3

Elevation: ? °
Azimuth: ? °

[✓ Dual Axis] Elevation Azimuth

Elevation: 0 deg 0 min 0 sec
Azimuth: 0 deg 0 min 0 sec

Speed (°/sec) 0.2

Figure 4.4: A screenshot of the dialog box that the operator uses to point IRMA.

if the chopper wheel is currently turned on, the button will display `Chop Off` indicating that the chopper wheel can be turned off by clicking the button. If the status of the unit is unknown, the button will be disabled. In order to enable it, the user simply clicks the `Refresh` button to force a query of the unit status. Since Alt-Az motion cannot be accomplished through a simple control button, Alt-Az functions are accessed though a menu option under `Engineering`. The `Move Alt-Az Mount` dialog box is shown in figure 4.4. The Alt-Az mount can also be initialised by selecting the appropriate menu item.

The sun shutter is also controlled through the `Engineering` menu. Under normal operating conditions, it will only be controlled by the hardware when the sun sensor is illuminated however, provision is made for manual control during testing.

To send a command to the IRMA unit, the *Control Interface* executes the process

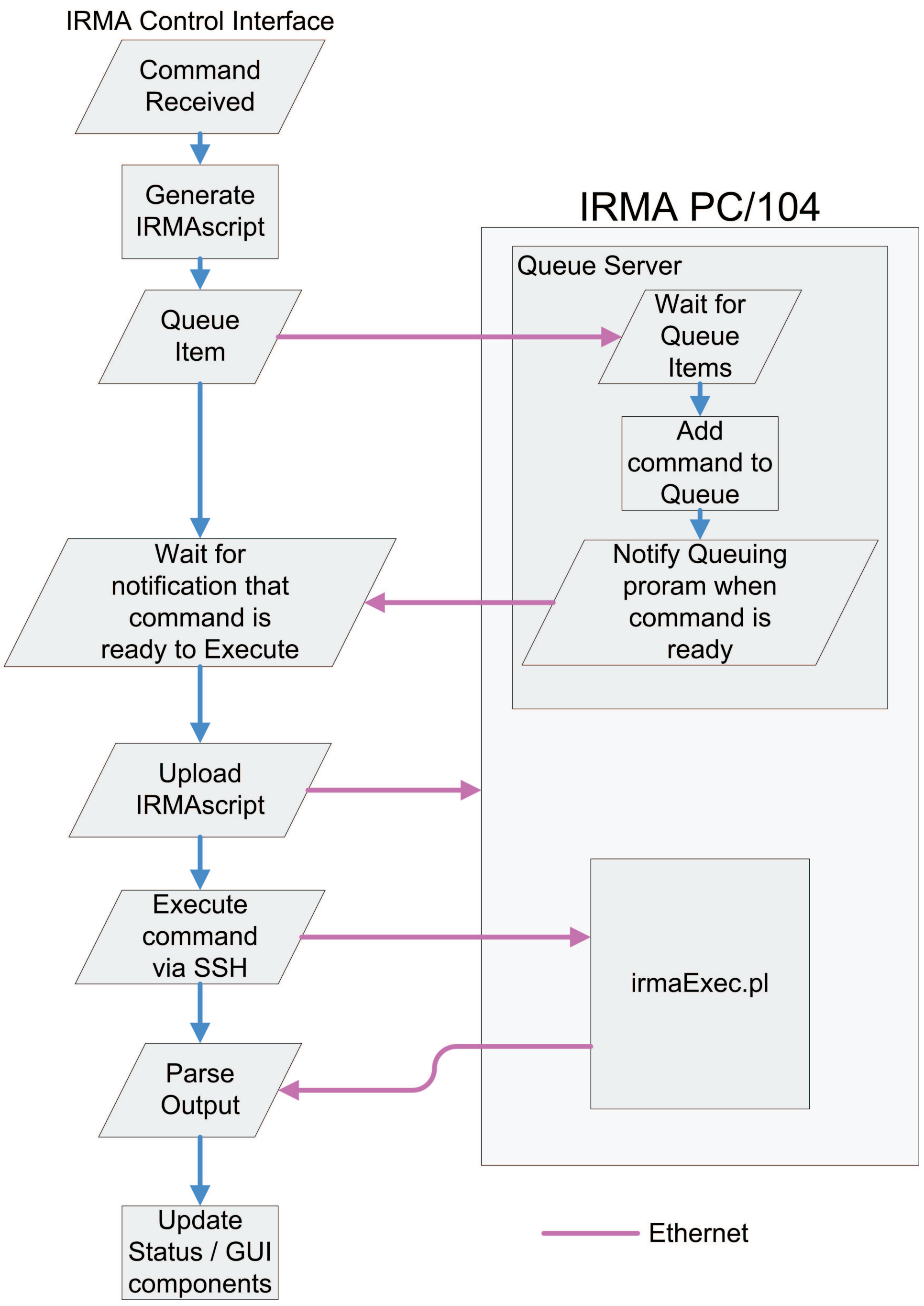


Figure 4.5: A flowchart showing the execution path of the IRMA Control Interface.

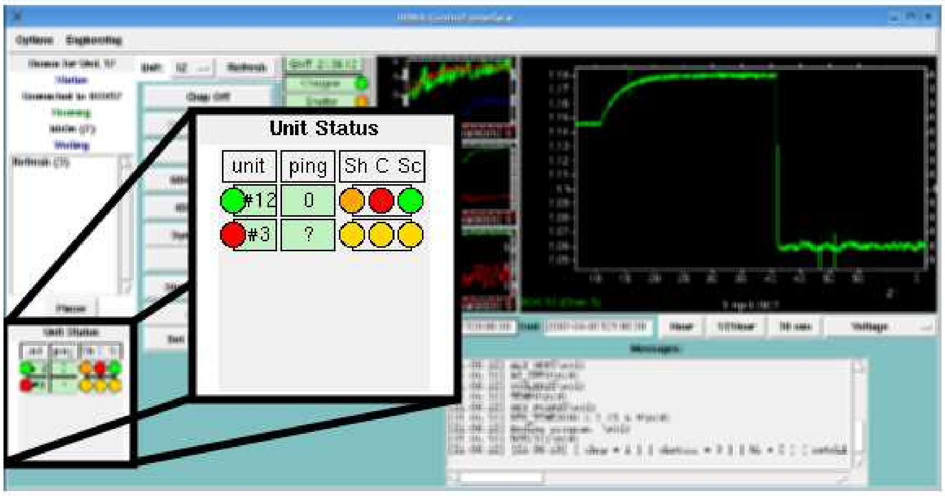


Figure 4.6: A screenshot showing a zoomed region of the Unit Stat area of the GUI showing that unit 12 is connected while unit 3 is disconnected.

shown in Figure 4.5. Immediately to the right of the control buttons, the status of the current unit is displayed. Each aspect of the IRMA unit is listed with a corresponding status indicator light. The indicator lights will be green for 'on' or 'open', yellow for 'unknown status', and red for 'off' or 'closed'. The status will be updated automatically at regular intervals if this option is selected in `GUI options` under the `Options` menu. Otherwise, the status can be updated manually at any time.

4.1.1.3 Unit Status

The *Unit Status* is located below the Queue Window (Figure 4.6). It provides an at-a-glance overview of all the units being monitored by the Control Interface. For each unit it shows whether or not it is connected to the units associated Queue Server, what the ping latency is for the unit, and the status if the shutter, cooler and scan. Again green indicates 'on' or 'connected', yellow indicates 'unknown status', and red indicates 'off' or 'not connected'. Units may be added or removed through `'GUI options'` under the

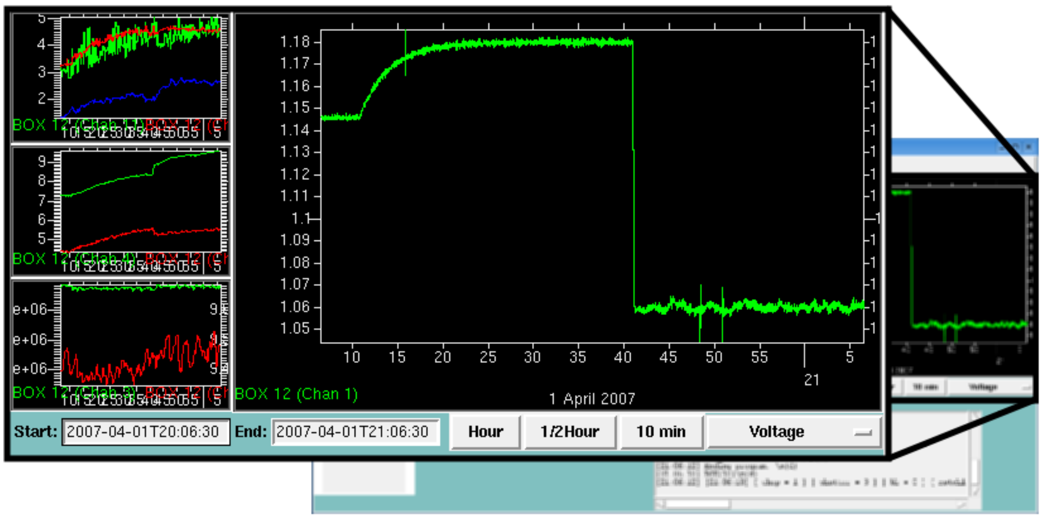


Figure 4.7: A screenshot showing a zoomed region of the graphical display area of the IRMA Control Interface. The right hand graph shows a calibration cycle followed by sky observation.

`‘Options’` menu.

4.1.1.4 Data Monitoring

The *Data Monitoring* section is capable of graphing any data channel from an IRMA unit. Channels are selected though a dialog box which is displayed by right clicking on the graph. This is the same for the smaller graphs. The time range is selected by entering the start and end times in ISO 8601 format [32] in their respective text boxes. Conveniently, time ranges can also be set by clicking on the `‘Hour’`, `‘1/2 Hour’`, or `‘10 min’` buttons which will select the corresponding most recent time interval. For example, the last 10 min. The graph will be displayed in the units selected from the drop-down list. Selecting ‘Spectral Power’ or ‘PWV’ for a channel other than channel 1 (the detector signal) will result in the units being shown in the appropriate calibrated units (°C, mbar, % humidity). Double clicking on a small graph will cause this graph to exchange places with the graph currently in the large graph area.

The data from the cryocooler can also be monitored using the *IRMA Control Interface*. This is accomplished by selecting `Display Cooler Stats` under the `Engineering` menu,which opens a new window and displays the cryocooler temperature and amplitude. Time ranges are selected in the same manner as the main graphing windows.

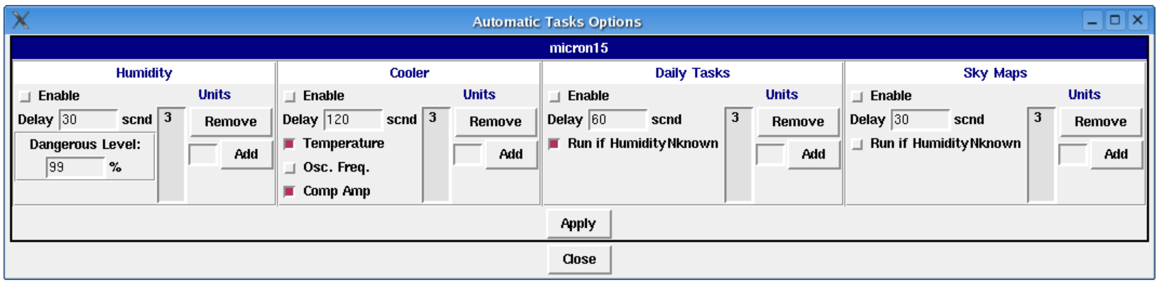
4.1.1.5 Message Window

The message window is located at the bottom right of the GUI, seen in Figure 4.1. It displays information such as connection attempts, raw output from IRMAscripts and other debug messages.

4.1.1.6 Managing AutoTasks

As discussed in §3.5, *Autotasks* may be configured manually on the IRMA unit. *Autotasks* can also be managed from within the *IRMA Control Interface*. The options for the `autoTasks.conf` file, described in §3.5.1.5 can be set with the *IRMA Control Interface* by selecting `AutoTask Options` under the `Options` menu. Once selected, the *Control Interface* connects to the unit and retrieves the current `autoTask.conf` file and displays the information in the dialog box shown in Figure 4.8. At this point, the options may be modified. The new options are then written to the remote `autoTask.conf` file when the operator selects `‘Apply’`.

The Daily Tasks application within *AutoTasks* may be managed through the `Daily Tasks` sub-menu, located under the `Engineering` menu. There are two items under the `Daily Tasks` sub-menu. The first, `Edit Task`, allows the operator to create or edit `.task` files without requiring any knowledge of the file format described in §3.5.1.3. The `.task` files may then be loaded to the current unit through the second option, `Organize Tasks`.


Automatic Tasks Options

micron15

Humidity
- Enable
- Delay 30 scnd
- Dangerous Level: 99 %
- Units: 3
- Remove
- Add

Cooler
- Enable
- Delay 120 scnd
- 🟥 Temperature
- Osc. Freq.
- 🟥 Comp Amp
- Units: 3
- Remove
- Add

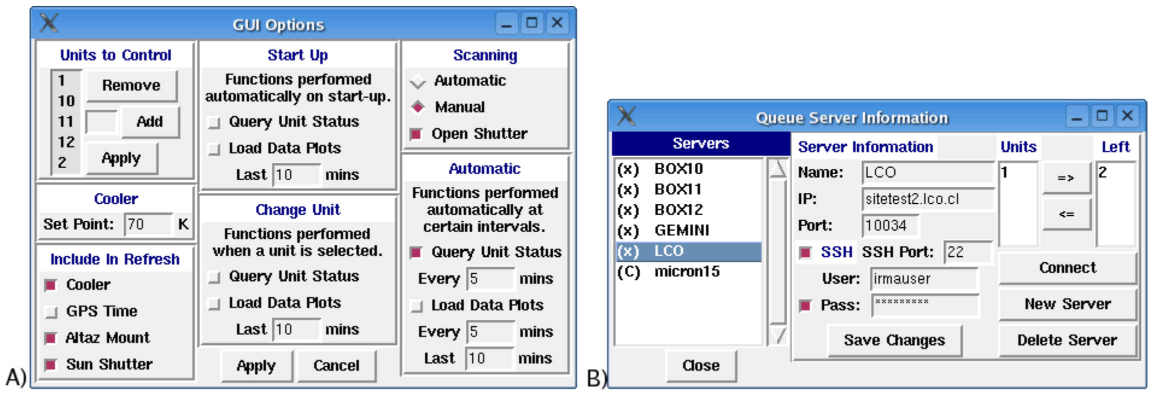
Daily Tasks
- Enable
- Delay 60 scnd
- 🟥 Run if HumidityNknown
- Units: 3
- Remove
- Add

Sky Maps
- Enable
- Delay 30 scnd
- Run if HumidityNknown
- Units: 3
- Remove
- Add

Apply

Close

Figure 4.8: A screenshot of the dialog box that the operator uses to configure AutoTasks.


A) GUI Options

Units to Control
- 1, 10, 11, 12, 2
- Remove
- Add
- Apply

Start Up
- Functions performed automatically on start-up.
- Query Unit Status
- Load Data Plots
- Last 10 mins

Scanning
- Automatic
- ◆ Manual
- 🟥 Open Shutter

Cooler
- Set Point: 70 K

Include In Refresh
- 🟥 Cooler
- GPS Time
- 🟥 Altaz Mount
- 🟥 Sun Shutter

Change Unit
- Functions performed when a unit is selected.
- Query Unit Status
- Load Data Plots
- Last 10 mins
- Apply
- Cancel

Automatic
- Functions performed automatically at certain intervals.
- 🟥 Query Unit Status
- Every 5 mins
- Load Data Plots
- Every 5 mins
- Last 10 mins

B) Queue Server Information

Servers
- (x) BOX10
- (x) BOX11
- (x) BOX12
- (x) GEMINI
- (x) LCO
- (C) micron15
- Close

Server Information
- Name: LCO
- IP: sitetest2.lco.cl
- Port: 10034
- 🟥 SSH SSH Port: 22
- User: irmauser
- 🟥 Pass: *********
- Save Changes

Units: 1

Left: 2

=> <=

Connect

New Server

Delete Server

Figure 4.9: A screenshot of the dialog box that the operator uses to select GUI options(A) and Queue Server information(B).

Tasks may either be setup to run on a specific day of as the default task file. Clicking `Apply` will upload the `.task` file as well as any associated IRMAscript files.

4.1.1.7 IRMA Control Interface Setup

Additional dialog boxes are provided under the `Options` menu to allow the operator to customize the IRMA Control Interface. The `GUI Options` dialog (Figure 4.9A) allows the operator to select which units to control as well as other options such as automatic status queries and graphing.

The `Queue Server Options` dialog (Figure 4.9 B) is used to associate unit numbers to the Queue Servers. This dialog is also used to setup the Queue Server information, which includes the IP address, username and password for remote login and SSH port. The standard SSH port is 22, however, some IRMA units are setup with non-standard ports.

4.1.2 Requirements and Execution

As mentioned earlier, the control GUI is written in Perl/Tk. Therefore, in order to use the program the Perl interpreter must be installed. A complete list of Perl modules required by the *IRMA Control Interface*, along with installation instructions, is found in Appendix C.2.

As with the IRMA software installed on the PC/104, the IRMA software running on other machines must follow the same directory structure listed in §3.2.1.3. The *IRMA Control Interface* is executed from the command line from the `~/IRMA/` directory with the `./IRMA.pl` command.

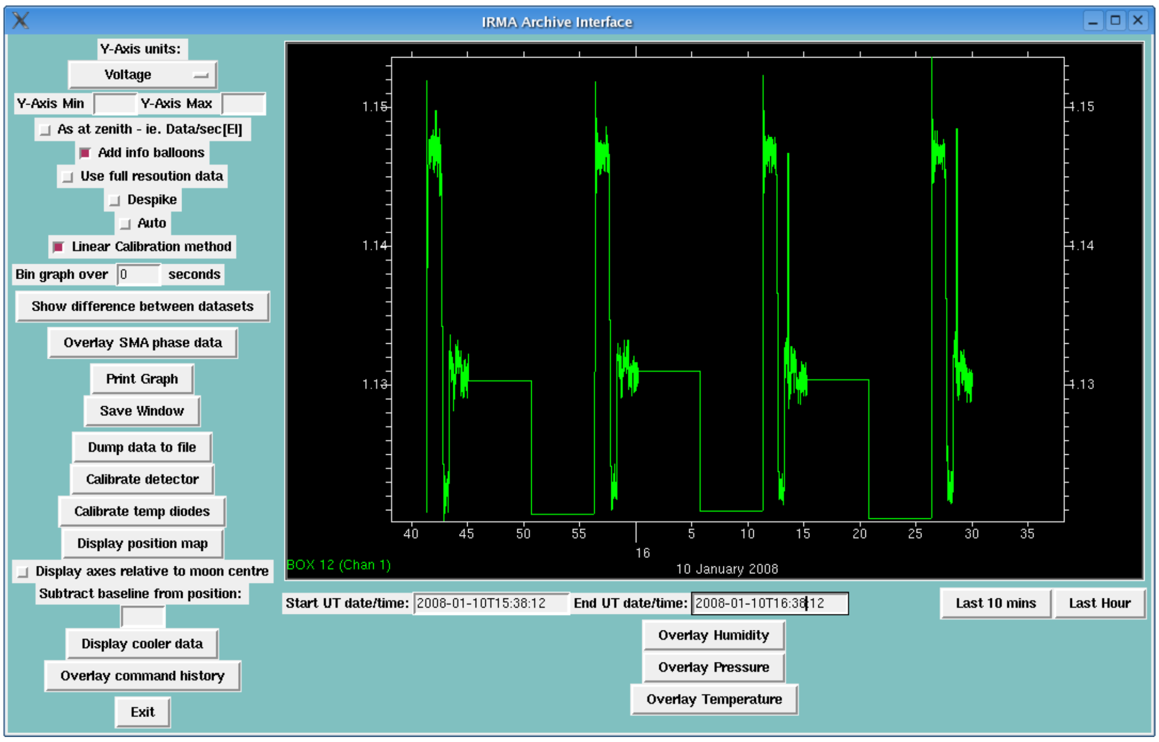
4.2 IRMA Archive Interface

The *IRMA Archive Interface* shown in Figure 4.10 is used for viewing data from the IRMA units. It is similar to the data monitoring portion of the IRMA Control Interface. In fact, through modular programming, the graphing functions written by Ms. Amy Smedes for the *IRMA Control Interface* are re-used in the *IRMA Archive Interface*. It also has many features which are not included in the *Control Interface*. These additional features relate to data processing. The *IRMA Archive Interface* has no menu items and all options and controls are selected through the checkboxes and buttons on the GUI. The *IRMA Archive Interface* has been a continually evolving program. As different data reduction fuctions were needed, they have been added to the interface. For example, after completing a testing campaign at the Smithsonian Millimeter Array (SMA), where IRMA was used in a phase correction role (see Figure 1.5), a button was added which would overlay the SMA phase correction data. Some of the other key features of the *Archive Interface* include, binning data, taking the difference of two data sets, adjusting data for elevation angle of the unit, saving and printing the graph, and saving the processed data to a file. There are also functions to overlay other relevant data on the graphs, such as weather data or command history.

In addition to maintaining this program, I made several additions and improvements, such as the ability to overlay the weather, selectable axis ranges, and an `auto` mode, which would automatically update the graph every minute with the last 10 minutes of data. The improved data handling routines written for the *IRMA Control Interface* were also used by the *Archive Interface*, since the module is accessed by both programs.

The *IRMA Archive Interface* also shares some funtionality with the *IRMA Control Interface*, namely the ability to display the cryocooler data, and calibrate the temperature diodes.

As will be discussed in Chapter 5, as the performance of the radiometer has become better understood, the calibration method for the IRMA units has been modified and improved, and with it the algorithm for converting detector signal voltage to PWV has become more complex. The new algorithm was not incorporated into the *IRMA Archive interface*, but written in IDL® and is discussed in §4.3.


Y-Axis units: Voltage

Y-Axis Min Y-Axis Max

☐ As at zenith - ie. Data/sec[El]

☑ Add info balloons

☐ Use full resoution data

☐ Despike

☐ Auto

☑ Linear Calibration method

Bin graph over 0 seconds

☐ Display axes relative to moon centre

Subtract baseline from position:

Start UT date/time: 2008-01-10T15:38:12 End UT date/time: 2008-01-10T16:38:12

Figure 4.10: A screen shot of the IRMA Archive Interface GUI.

4.2.1 Requirements and Execution

As with the other Perl based programs which have been discussed, the *IRMA Archive Interface* requires that additional Perl modules be installed, as well as custom IRMA modules to be present. The *IRMA Archive Interface* depends on many of the same Perl modules as the *IRMA Control Interface*, a complete list of required modules, and installation instructions are given in Appendix C.2.

The Archive Interface is located in the same directory as the *IRMA Control Interface*, `~/IRMA/`, and is executed with the `./viewirma.pl` command.

4.3 IRMA PWV

As will be discussed in §5.1, the IRMA units were once calibrated using a simple, two point calibration method. However, it was discovered that the output voltage of the detector is also dependent on other factors such as internal box temperature. Therefore, changes were made to the calibration method, which modified the conversion algorithm, and therefore, new calibration software was required. IRMA PWV was written in IDL®(Interactive Data Language) because it is well suited for large arrays of data, and its integration of complex mathematical functions such as multivariable linear regression and interpolation functions. Built in plotting functions, greatly simplify the visualisation of IRMA data.

IRMA PWV is still in the early stages of developement. It is currently a simple program, which performs the specific task of converting the IRMA signal voltage to PWV. The conversion process performed by the software is described in Figure 4.11. It is envisioned that this software will form the basis of the next generation of data reduction and control software, replacing both the *IRMA Control Interface* and the *IRMA Archive interface*. The proposed future development is discussed in §6.3.2.

4.4 Summary

The front end and visualisation software allows the user to control the IRMA unit and analyse data remotely. The IRMA Control Interface provides an intuitive GUI for real-time unit status and data monitoring, in addition to remote AutoTasks configuration. The IRMA Archive Interface is used for viewing and analysing the IRMA data. While

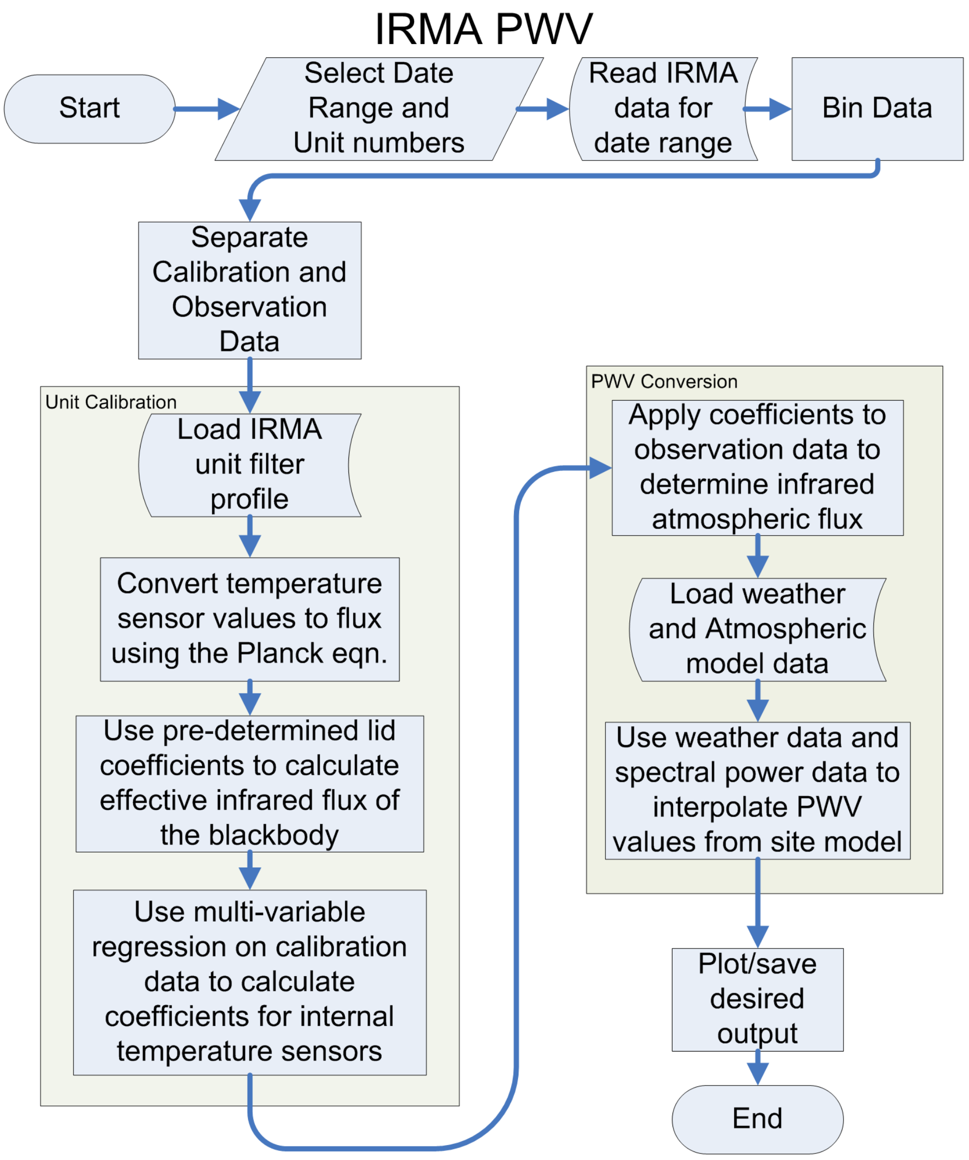

IRMA PWV
Start
Select Date Range and Unit numbers
Read IRMA data for date range
Bin Data
Separate Calibration and Observation Data
Unit Calibration
Load IRMA unit filter profile
Convert temperature sensor values to flux using the Planck eqn.
Use pre-determined lid coefficients to calculate effective infrared flux of the blackbody
Use multi-variable regression on calibration data to calculate coefficients for internal temperature sensors
PWV Conversion
Apply coefficients to observation data to determine infrared atmospheric flux
Load weather and Atmospheric model data
Use weather data and spectral power data to interpolate PWV values from site model
Plot/save desired output
End

Figure 4.11: A flowchart describing the steps involved in calculating the PWV from the raw data.

these programs were written by a previous student, I was able to learn the code structure, maintain and make bug fixes in the code. I also improved the performance of the programs and added new functionality as needed.

With the development of a new calibration method, new calibration software was written. The new calibration program *IRMA PWV*, was written in IDL® because of its efficient handling of large arrays of data, and built in graphical display functions. Though *IRMA PWV* is currently only used for conversion of IRMA signal voltage to PWV, future plans are to expand its development into a full suite of IRMA control and data processing function, thereby replacing both the *IRMA Control* and *Archive Interfaces*.

Chapter 5

Instrument Calibration and Field Performance

5.1 Calibration

IRMA detects the radiated infrared flux from the atmosphere and, by use of an atmospheric model, this value is converted to precipitable water vapour (PWV). The accuracy of the final measurement is determined by sources of error in the instrumental measurement and the accuracy of the model. This section addresses the instrument calibration. For more details on the atmospheric model see [2], [13], and [33].

5.1.1 Calibration Basics

To first order, the response of the IRMA infrared detector is linear with respect to infrared flux, Φ. A flux-to-voltage relationship is established by measuring the signal voltage while observing two calibration sources at different temperatures. This can subsequently be used to convert any signal voltage to a measured flux. The calibration method described in this section was adopted for the prototype, IRMA I. Although it still forms the basis of the calibration used today, in the most recent IRMA design, which is an enclosed system, a more complex calibration procedure is required. Improvements to this calibration method are discussed in §5.1.3.

In the basic calibration method, the detector observes two blackbody calibration sources of different temperature. For a blackbody of temperature T in Kelvin, the spectral radiance L(σ,T) is given by the well known Planck equation:

$$L(\sigma,T) = \frac{2\,h\,c^2\,100^4\,\sigma^3}{\exp\left(\dfrac{h\,c\,\sigma}{k_B\,T}\right)-1} \qquad [W\,m^{-2}\,sr^{-1}\,(cm^{-1})^{-1}], \tag{5.1}$$

where σ is the wavenumber in cm^{-1}, h is Planck's constant in Js, k_B is Boltzmann constant in J/K and c is the speed of light in m/s. Calculating the radiant flux, Φ, detected by IRMA required knowledge of the instrumental throughput, $A\Omega$, and the spectral response function of the detecting system, F_σ. The flux is calculated by integrating over the normalised spectral bandpass of the detection system, 450 to 575 cm^{-1} (equivalent to 17 to 22 μm), shown in Figure 5.1, and is given by:

$$\Phi(\sigma,T) = \int_{450}^{575} \frac{2\,h\,c^2\,100^4\,\sigma^3}{\exp\left(\dfrac{h\,c\,\sigma}{k_B\,T}\right)-1} A\,\Omega\,F_\sigma\;d\sigma \qquad \text{[W]}. \tag{5.2}$$

The measured signal voltage is related to the calculated flux by an equation of the form

$$\Phi = A\;V + \Phi_0 \qquad \text{[W]}. \tag{5.3}$$

After observing the two calibration sources, the gain, A, and offset, Φ_0, of the voltage to flux relationship are established by the following formulas.

$$A = \frac{\Phi_2 - \Phi_1}{V_2 - V_1}, \tag{5.4}$$

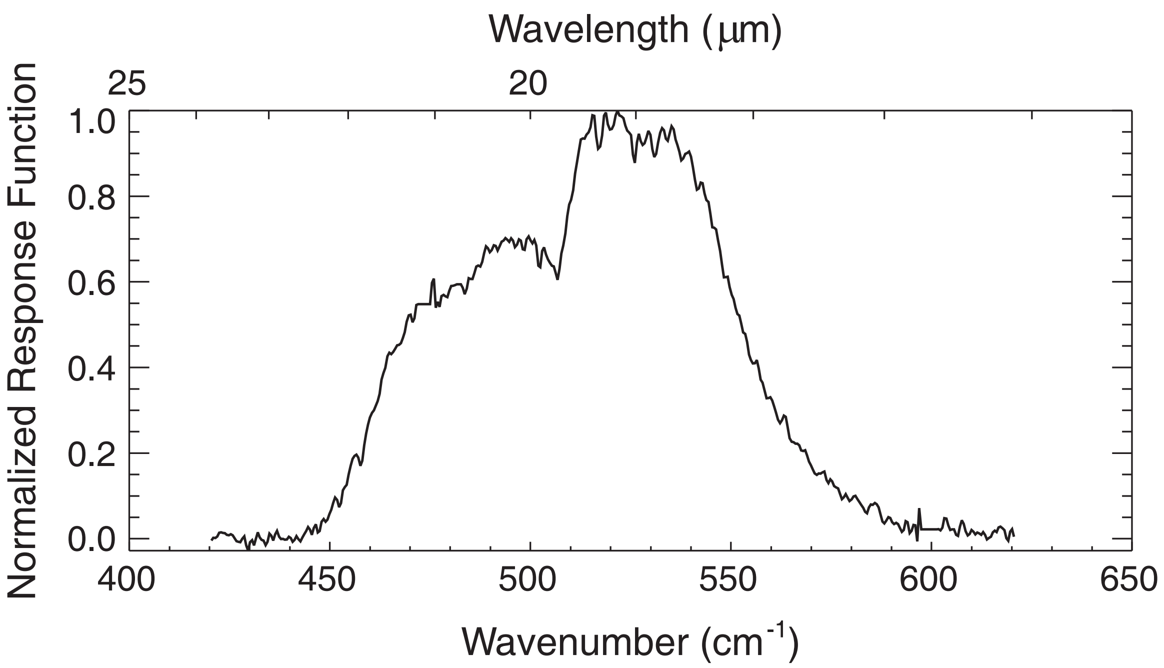


Figure 5.1: Normalised IRMA instrument response function as measured at 70 K using an ABB Bomem FTS, model MB102. The profile is the end-to-end instrument response (the convolution of the filter transmission profile, transmission of the anti-reflection coated ZnSe window, and the photodetector response over the given spectral range).

$$\Phi_0 = \Phi_1 - \frac{V_1\,(\Phi_2 - \Phi_1)}{V_2 - V_1} \qquad [\mathrm{W}]. \tag{5.5}$$

Once calibrated, these parameters can subsequently be applied to the measured signal voltage during sky observations using Equation 5.3 to determine the infrared flux radiated by the atmosphere. Finally, the atmospheric flux can be related to PWV by using an appropriate atmospheric model as described in §1.3.2.

5.1.2 Calibration Issues

In the IRMA I prototype, the lower temperature blackbody source consisted of a large dewar of liquid nitrogen (LN_2), ∼77 K (∼73 K at the summit of Mauna Kea). The second calibration blackbody was a large aluminum plate coated with carbon black doped

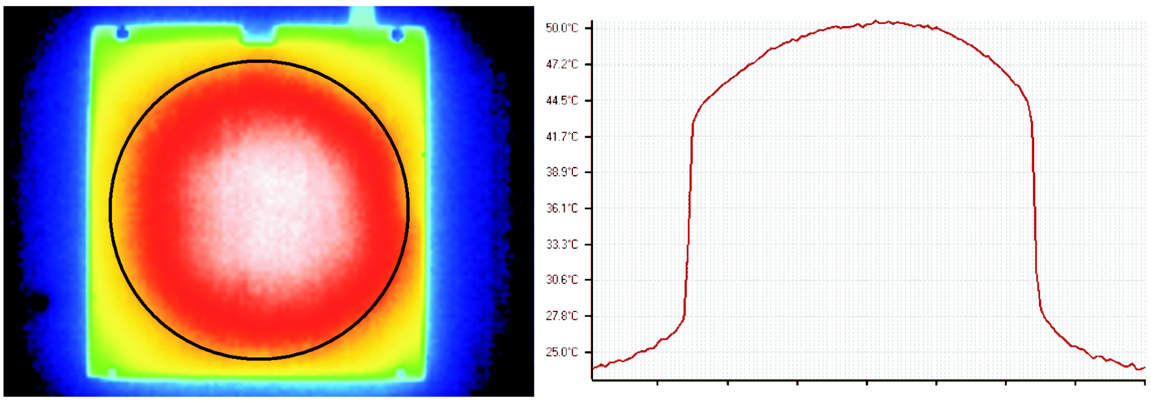


Figure 5.2: A thermal image of an internal lid blackbody taken with a Fluke Ti20 7-14(left). A cross-sectional temperature profile of the image taken through the middle. µm camera.

thermally conductive epoxy, which tracked the ambient temperature. Since the effective sky temperature falls between these two extremes, it allowed for a direct interpolation of the measured atmospheric flux from the signal voltage. As discussed in Chapter 2, the IRMA unit was redesigned to provide robust, remote, and autonomous operation. This redesign necessitated removing the dependence on LN_2, which is not available at remote sites, both in the wet cryostat and the calibration blackbody. As has been discussed (§2.3.1), the wet cryostat has been replaced by a Stirling cryocooler. The original calibration sources were replaced by a blackbody attached to the underside of the weather shutter (§2.3.1), which could be heated to a temperature of 20 K above ambient. The two-point calibration was then accomplished by observing the blackbody at ambient and elevated temperatures. Unfortunately, in this method the temperature of both calibration points lies above the effective temperature of the sky and an extrapolation of the calibration data is needed to determine the atmospheric flux. Modeling has shown that in order to measure PWV to an accuracy of 10% the effective temperature must be know to a precision of ± 0.1 K [13].

In the original design, two temperature sensors were embedded in the blackbody

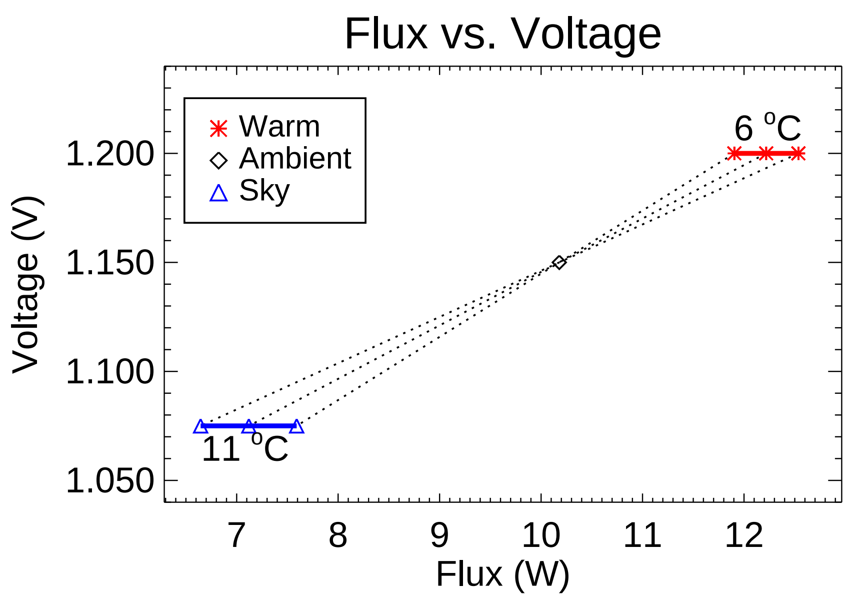


Figure 5.3: Potential error in atmospheric flux introduced through a two-point extrapolation. The flux calculation for the ambient reading is taken to have small error bars and acts as a fulcrum point. Warm blackbody flux variance equivalent to ±3 K, results in an extrapolated error equivalent to ±5.5 K at typical sky temperature.

to determine the effective temperature; one sensor in the center and the other located near one of the edges. Under ambient conditions the blackbody temperature is uniform across its surface, but when heated, a temperature gradient was found to exist, as shown in Figure 5.2, which shows a thermal image obtained with a Fluke Ti20 infrared camera kindly made available by Fluke Electronics Canada. The temperature profile shows that the centre of the blackbody was measured at 50 ±0.2 °C (323 K), while the edge was 46 ±0.2 °C (319 K).

Initially, it was unclear how the two temperature readings from the heated blackbody, related to the effective temperature of the surface. Since the blackbodies on individual IRMA units are custom-made, each has unique gradients, which lead to errors in their respective calibration. This results in different flux values being reported by two IRMA units

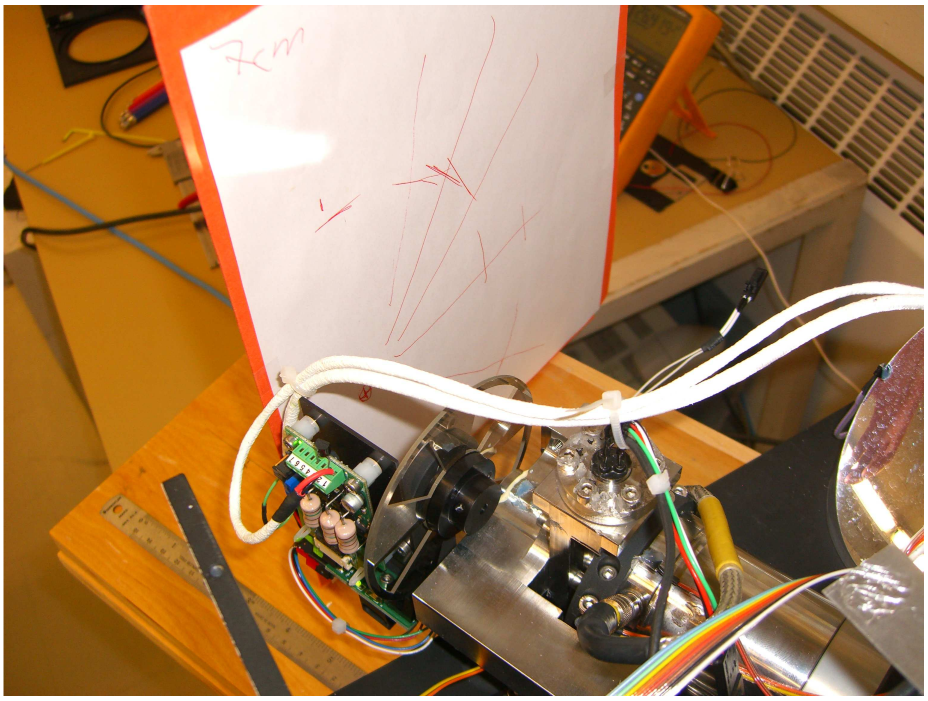


Figure 5.4: A photograph showing how the IRMA field of view was mapped using a soldering iron.

observing the same atmosphere, which is clearly unacceptable. Moreover, this error is further amplified by the extrapolation process described above. To illustrate this point, Figure 5.3 shows the resultant extrapolation error due to a ±3 K error in the effective temperature of the hot calibration source. At ambient temperatures, the blackbody is assumed to be in a state of thermal equilibrium, and therefore, accurately represented by the internal temperature sensors. The smaller error of the ambient measurement causes this point to act as a fulcrum for the extrapolation. The ±3 K results in an error equivalent to ±5.5 K at typical sky temperatures, which is the key driver for the ±0.1 K accuracy requirement for the heated blackbody.

During rigorous laboratory testing, it was determined that a difference in lid gra-

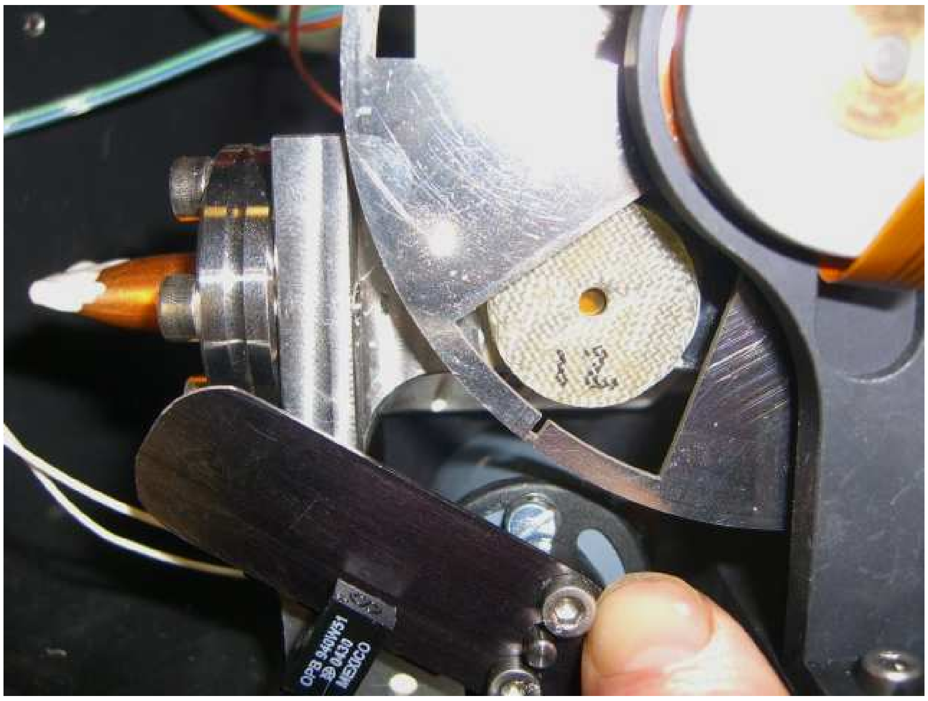


Figure 5.5: A photograph of the additional aperture which was attached to the cryo-cooler window to better define the field of view.

dients was not the only source of the offset observed between IRMA units when viewing the same atmosphere. To explore this further, it was decided to view a dewar of liquid nitrogen, LN_2, since at this temperature there is essentially zero emission at 20 μm. The IRMA unit was carefully turned upside-down and held above an LN_2 dewar. A signal, indicative of a stray radiation component, was detected, prompting further investigation. A hot soldering iron was slowly moved around the expected field of view of the detector to further investigate this phenomenon (Figure 5.4). These tests confirmed that the detector was viewing past the primary mirror. To correct for this over-illumination an additional aperture was mounted onto the cooler window, as shown in Figure 5.5. Though reducing the aperture size lessened the effect of the stray internal box radiation, it did not eliminate it.

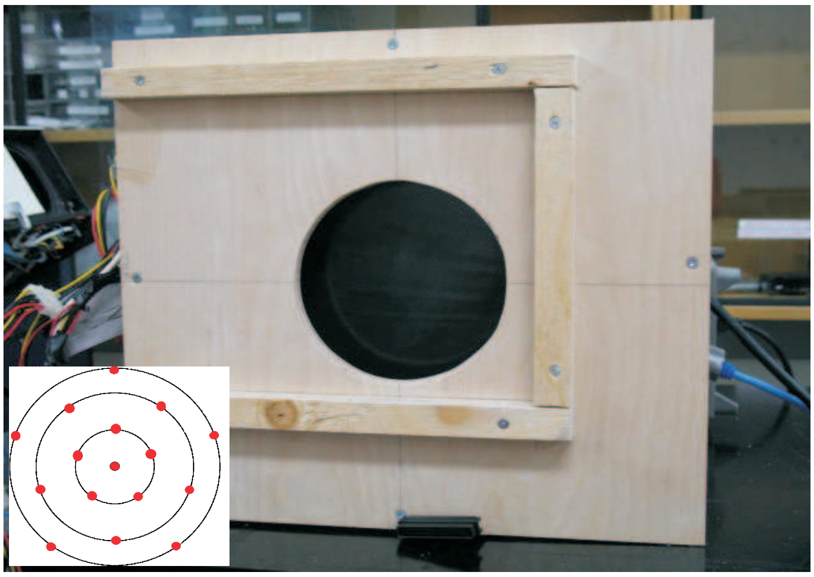


Figure 5.6: A photograph of the the large reference blackbody (LBB), which is mounted in a wooden frame that can be accurately positioned atop an IRMA unit. Inset in the image is a representative mapping of the embedded temperature sensors.

In the next phase of the investigation, 11 of the 16 temperature sensors were placed throughout the optical cavity to study the temperature distribution inside the IRMA unit in an attempt to determine the cause of the stray radiation. At the same time, a new calibration procedure was developed.

5.1.3 Improved Calibration Method

To address the issues arising from the box-to-box variance in blackbody emission, a new process was adopted, which involved the development of a large external black body (LBB). The LBB would serve as a primary reference to which all of the individual IRMA blackbody sources would be calibrated. To overcome the limitations of the internal calibration sources which, by virtue of the available space, were small, found to exhibit unexpectedly large temperature gradients, and only contained two temperature sensors, the reference blackbody was designed with a higher power heater and a much larger surface to avoid edge effects. The large blackbody had 16 temperature sensors embedded into its surface, arranged in the pattern shown in Figure 5.6, to allow for the accurate mapping of its surface temperature profile. The sixteen sensors were calibrated by placing the entire large blackbody in a convection oven and referencing each sensor to a calibrated Lakeshore temperature diode [34]. Initially, and creatively, the temperature sensors of the LBB were read using a PC/104 and a spare IRMA motherboard. The IRMA software required only minor modifications to acquire the temperature data from the 16 sensors. This was later replaced with a dedicated Data Translation DT9803 [35] high-performance USB data acquisition module. A wooden housing was built to allow for repeatable positioning atop the IRMA viewing port, shown in Figure 5.6. Several of the internal IRMA temperatures were also incorporated into the calibration in an attempt to identify and account for the effects of stray internal radiation measured by the detector.

The calibration was performed by taking a series of LBB measurements interspersed with internal lid blackbody measurements as shown in Figure 5.7. A typical calibration scheme consists of viewing the LBB at four distinct temperatures of approximately 22 °C, 35 °C, 60 °C and 90 °C.

Once the calibration data were obtained, a linear, least-squares fit was performed between the effective LBB temperature being viewed by IRMA, the internal IRMA temperatures to account for stray radiation, and the photodetector signal voltage. This fit was then used to calibrate the internal blackbody temperature against the LBB such that it could be used as a secondary calibration source in the field. The complete calibration procedure was done in three steps.

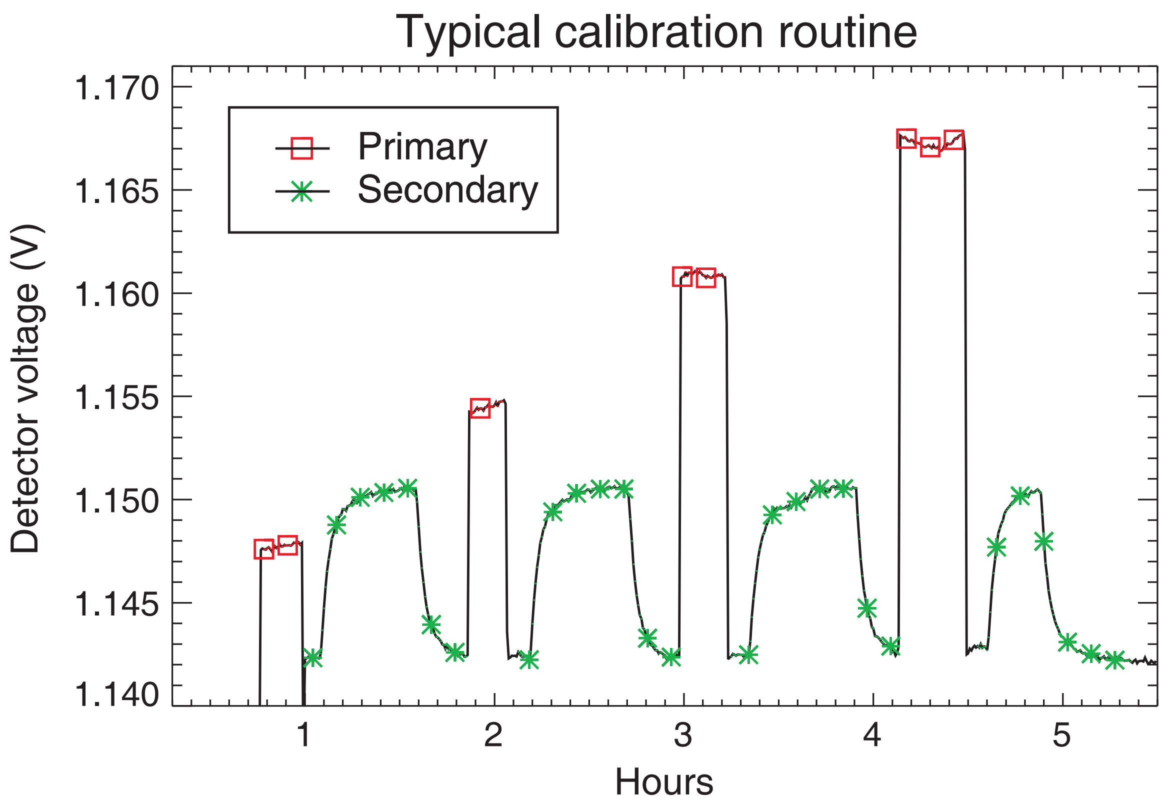


Figure 5.7: A typical calibration sequence performed in the laboratory. The black line is detector voltage, the red (square) highlighted sections correspond to measurements of the primary calibrator (LBB), the green (asterisk) highlighted sections correspond to the measurements of secondary calibrator (lid blackbody).

5.1.3.1 Primary calibration

The primary calibration is responsible for determining the influence of internal unit temperatures on the signal voltage. Originally, a relationship involving the effective temperature of potentially contributing components of the signal voltage was used in the analysis. However, the signal voltage is linear with respect to radiant flux, not effective temperature. Therefore, a special scaling function was required to linearise the signal voltage-to-temperature relationship. Once the effective temperature of the sky was calculated, this value could be converted to flux using Equation 5.2. The scaling function was potentially a source of error in the flux calculation, because it had to be pre-calculated for a range of temperatures. If the actual sky temperature was found to be outside the pre-calculated range, extrapolation errors would be present when the scaling function was applied.

The primary calibration is accomplished by observing the LBB shown in red in Figure 5.7. Due to stray radiation within IRMA, the detector voltage is not simply a result of the flux from the LBB, Φ_{LBB}, but also contains some additional flux. The detector voltage can, in general, be expressed as a linear combination of all possible sources of emission:

$$V = V_0 + c_{\mathrm{LBB}}\,\Phi_{\mathrm{LBB}} + \sum_{i=0}^{n} c_i\,\Phi_i \qquad [\mathrm{V}] \quad , \tag{5.6}$$

where V is photodetector voltage, V_0 is the offset term, c_{LBB} is the coefficient associated to the flux emitted by the LBB, and c_i are the n coefficients associated with the flux emitted from the n areas inside the IRMA unit used to account for stray internal radiation, Φ_i. This equation can be rearranged to solve for Φ_{LBB}:

$$\Phi_{\mathrm{LBB}} = \frac{V - V_0 - \sum_{i=0}^{n} c_i\,\Phi_i}{c_{\mathrm{LBB}}} \qquad [\mathrm{W}] \quad . \tag{5.7}$$

The offset and coefficients were calculated by performing a linear, least squares fit to the flux emitted by the LBB.

Selecting the internal temperature sensors to use in the fit was a non trivial task. Any of the 11 temperature sensors located in the optical compartment can potentially be used to account for the stray internal radiation. Originally, a statistical approach was taken to determine which of the 11 sensors should be used. All relevant combinations of sensors were fitted to a set of primary calibration data using Equation 5.6. The standard deviation, σ, of the difference between the fits and the measured $\mathrm{T_{LBB}}$ were computed. The combination of temperature sensors resulting in the lowest σ was chosen. Later, a more general approach was taken. An initial fit was performed using all available temperature sensors. The temperature sensors with the largest coefficients from the fit were selected. In each case, the dominant sensor was the same one selected using the statistical method. However, the secondary sensor differed in each case.

5.1.3.2 Secondary calibration

A secondary calibration is performed to relate the internal blackbody to the LBB. Calibrating the internal blackbody to the LBB allows for periodic verification of the calibration while the unit is operating remotely. For this secondary calibration, the data from the internal blackbody during the calibration sequence are used (shown in green in Figure 5.7). Once the primary calibration has been completed the infrared flux of the lid is determined similar to Equation 5.7 by:

$$\Phi_{\mathrm{effLID}} = \frac{V - V_0 - \sum_{i=0}^{n} c_i \, \Phi_i}{c_{\mathrm{LBB}}} \qquad [\mathrm{W}] \ . \tag{5.8}$$

The correlation between the flux associated with readings from the embedded temperature sensors and the effective flux of the internal blackbody is determined by fitting to the equation:

$$\Phi_{\mathrm{effLID}} = \Phi_0 + c_1 \, \Phi_1 + c_2 \, \Phi_2 \qquad [\mathrm{W}] \ , \tag{5.9}$$

where Φ_0 is an offset term, Φ_1 and Φ_2 are the calculated flux associated with the lid diode temperatures, and c_1 and c_2 are their respective fit coefficients.

5.1.3.3 Calibrated sky measurement

When the IRMA units are operating at remote sites, periodic calibrations are performed using the internal blackbody, which acts as a secondary standard. By applying

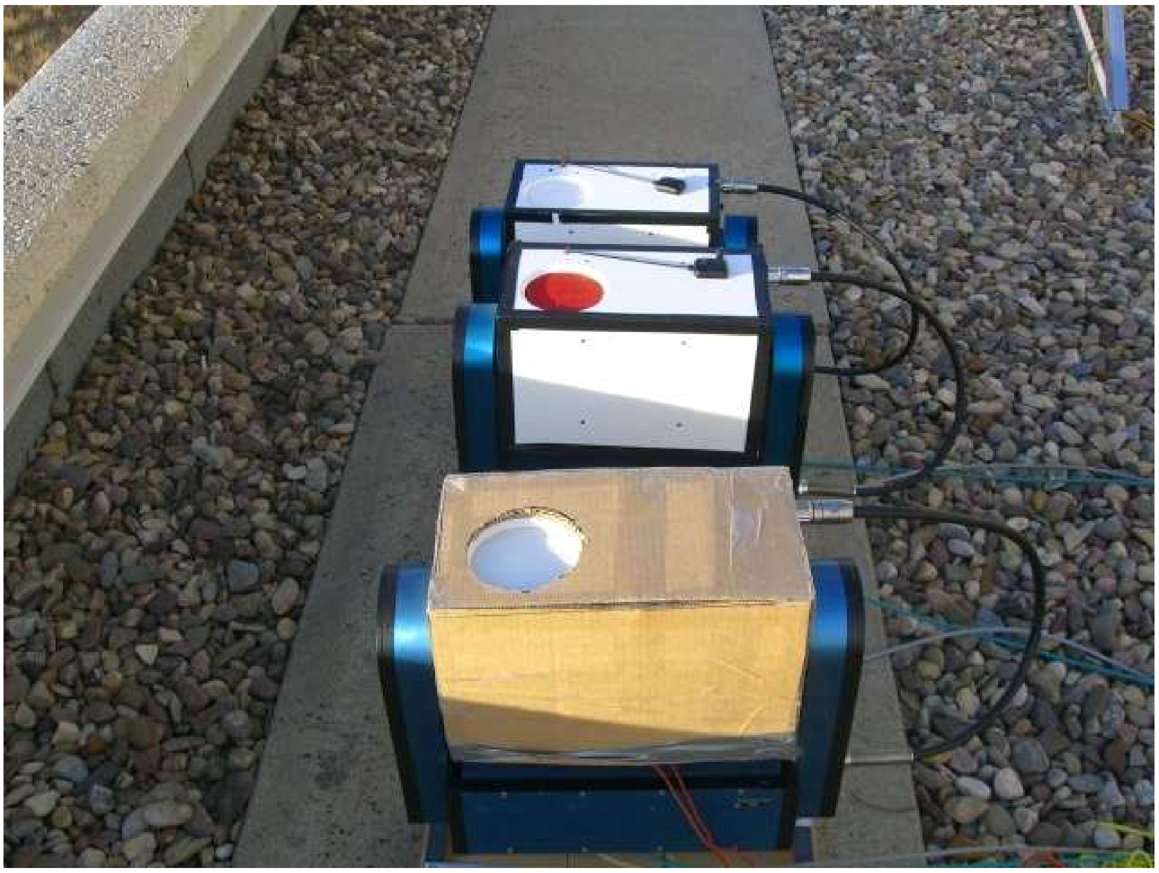


Figure 5.8: Three IRMA units on the roof above our laboratory at the University of Lethbridge. The lower unit is fitted with a heating cable and insulating jacket (cardboard) to test the ability of the fitting routine to correctly account for and remove the systematic offset due to heating.

the coefficients determined in Equation 5.9 the effective flux seen at the detector can be calculated. A new set of internal temperature coefficients are then calculated by applying a linear, least squares fit similar to Equation 5.7 except replacing Φ_{LBB} with Φ_{effLID}. Once these new coefficients are calculated the infrared flux from the atmosphere can be determined by equation:

$$\Phi_{\mathrm{Sky}} = \frac{V - V_0 - \sum_{i=0}^{n} c_i \, \Phi_i}{c_{\mathrm{effLID}}} \qquad [\mathrm{W}] \quad . \tag{5.10}$$

The flux associated with this temperature is found using equation 5.2. This flux value, along with the local temperature and pressure, can be converted to PWV using a pre-calculated

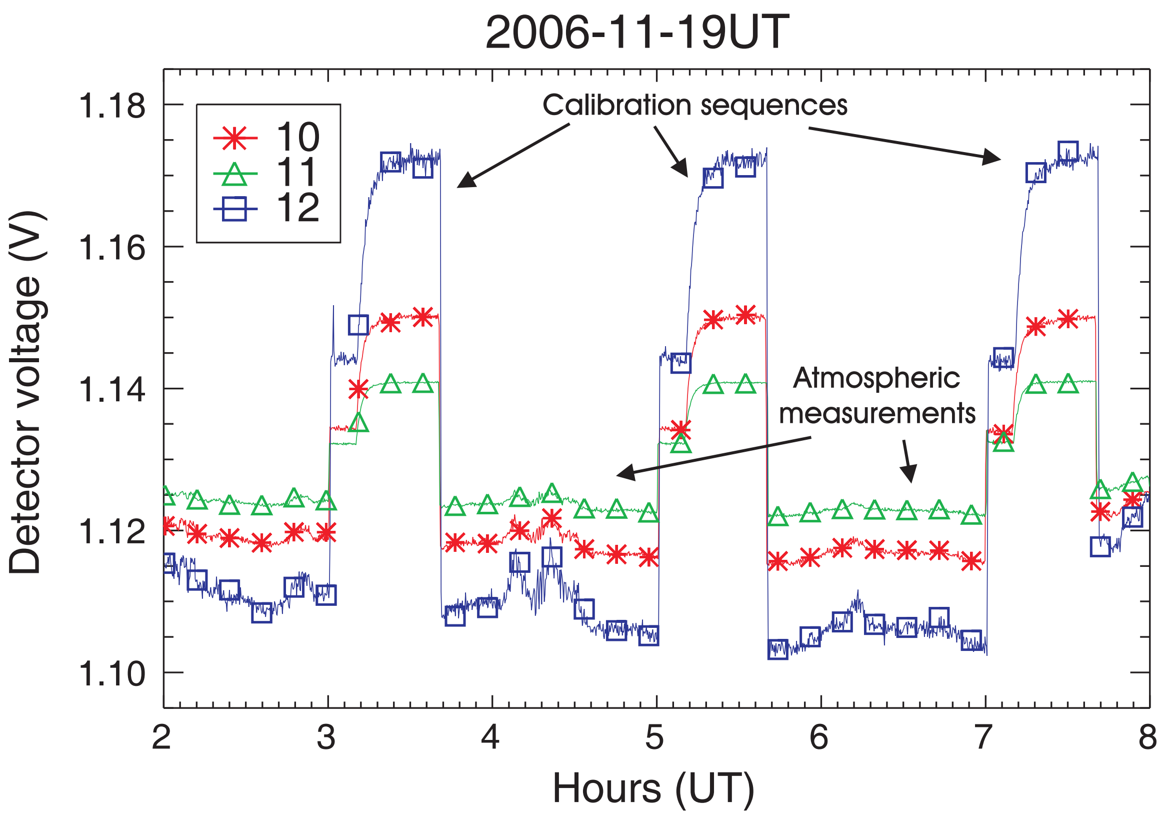


Figure 5.9: Raw data for the three co-located IRMA units operating in Lethbridge, shown in Figure 5.8. Interspersed with measurements of the atmosphere are three calibration sequences.

atmospheric model, as discussed in §1.3.2.

5.1.4 Results

To verify the performance of the improved calibration method, three IRMA units were calibrated independently using the procedure described in §5.1.3. These units were then placed side-by-side on the roof of University Hall at the University of Lethbridge,

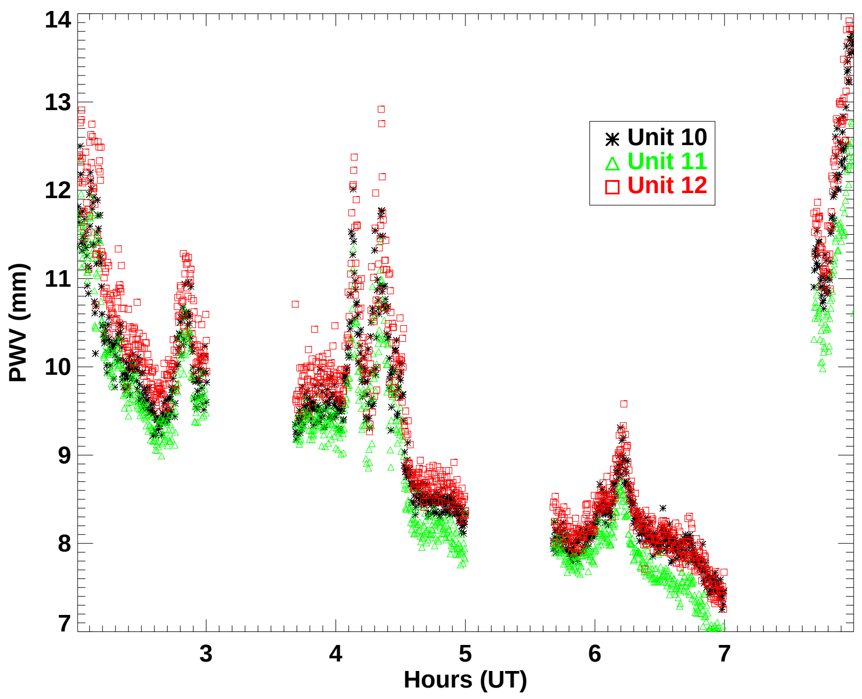


Figure 5.10: Derived PWV values for the three co-located IRMA units in Lethbridge, shown in Figure 5.8.

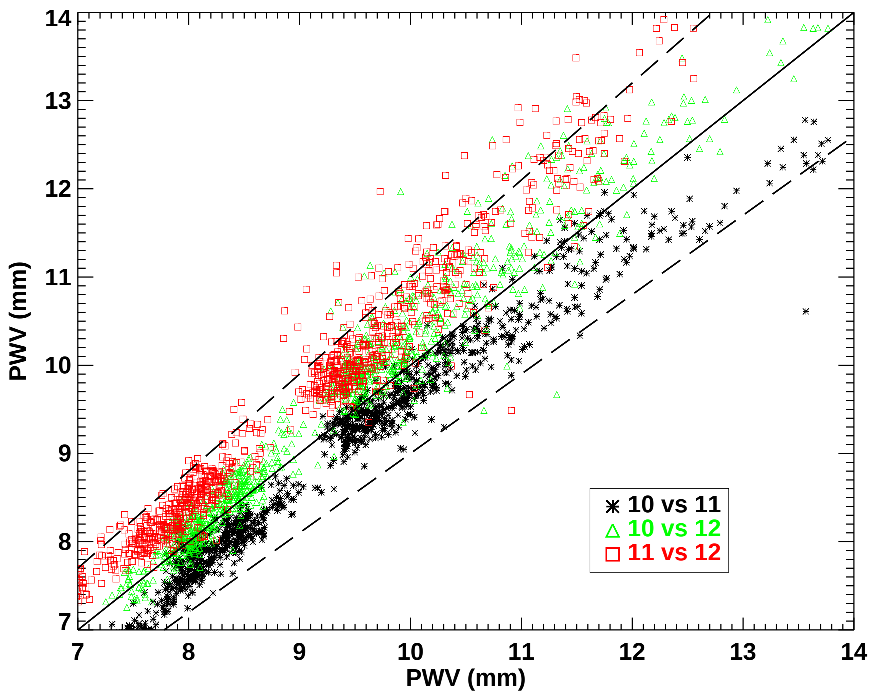


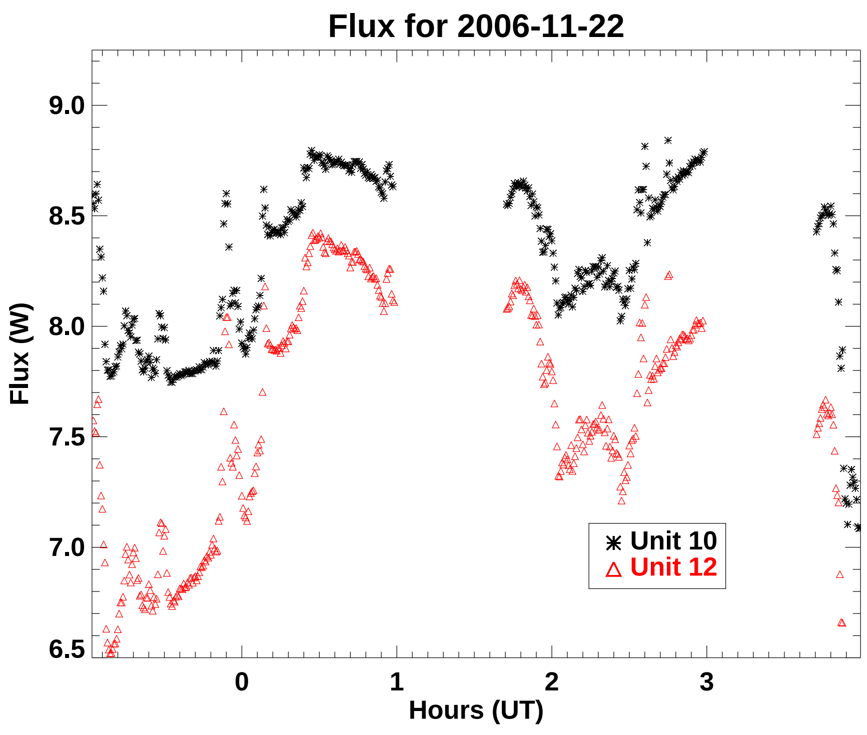
Figure 5.11: Inter-comparison scatter plot of PWV data from three co-located IRMA units in Lethbridge, showing a pairwise comparison. The solid line is the ideal unity slope reference line, while dashed lines are the $\pm10\%$ tolerance limits.

Table 5.1: Correlation coefficients for the pairwise comparison of Lethbridge test data shown in Figure 5.11.

Data Set 1	Data Set 2	Correlation Coefficient
Box 10	Box 11	0.9768
Box 10	Box 12	0.9774
Box 11	Box 12	0.9652

directly above our laboratory, as shown in Figure 5.8 and allowed to view the same atmosphere. The raw, unprocessed voltage data from each of the units is shown in Figure 5.9. From this figure, it can be noted that each unit has different offset and gain characteristics, which are due mainly to differences in the gain and offset of the pre-amplifier of each unit. However, despite these differences, once the initial calibration coefficients were applied to the data the resulting PWV measurements were well correlated, as can be seen in Figure 5.10. To better illustrate the correlation, resulting PWV from each unit was plotted against the PWV from the other two (Figure 5.11). The near unity linear relationship, with correlation coefficients listed in Table 5.1, demonstrated that the calibration method was successful, and the units were ready for testing under more optimal conditions.

Once in the field, the units are subjected to a wide variety of environmental conditions. To demonstrate that the IRMA calibration was independent of environmental conditions, one on the boxes was fitted with a heater while the units were observing on the roof in Lethbridge, the lower unit in Figure 5.8. The heater caused the internal temperature of IRMA to rise $\sim$10 K above the ambient operating conditions. Despite the large difference in unit temperature, the calibration routine was able to successfully remove the systematic effects due to the heating. The results are shown in Figure 5.12.



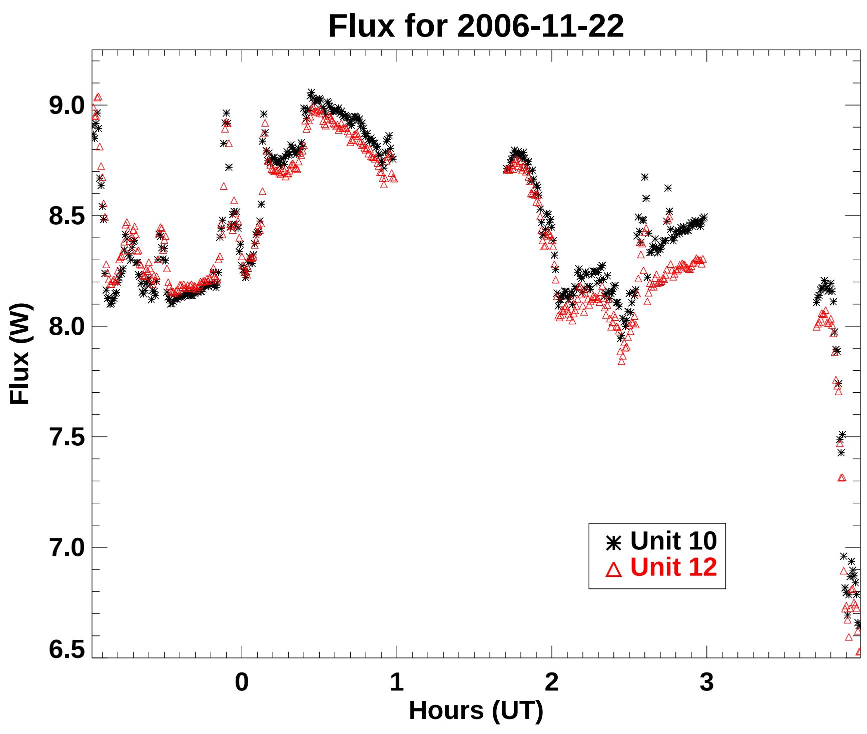


Figure 5.12: Flux values for two co-located IRMA units in Lethbridge while unit 10 was heated 10 K above ambient temperature. The upper plot does not account for the effective stray light. The lower plot includes a correction for stray light based on the method described in the textwhile the bottom plot does.

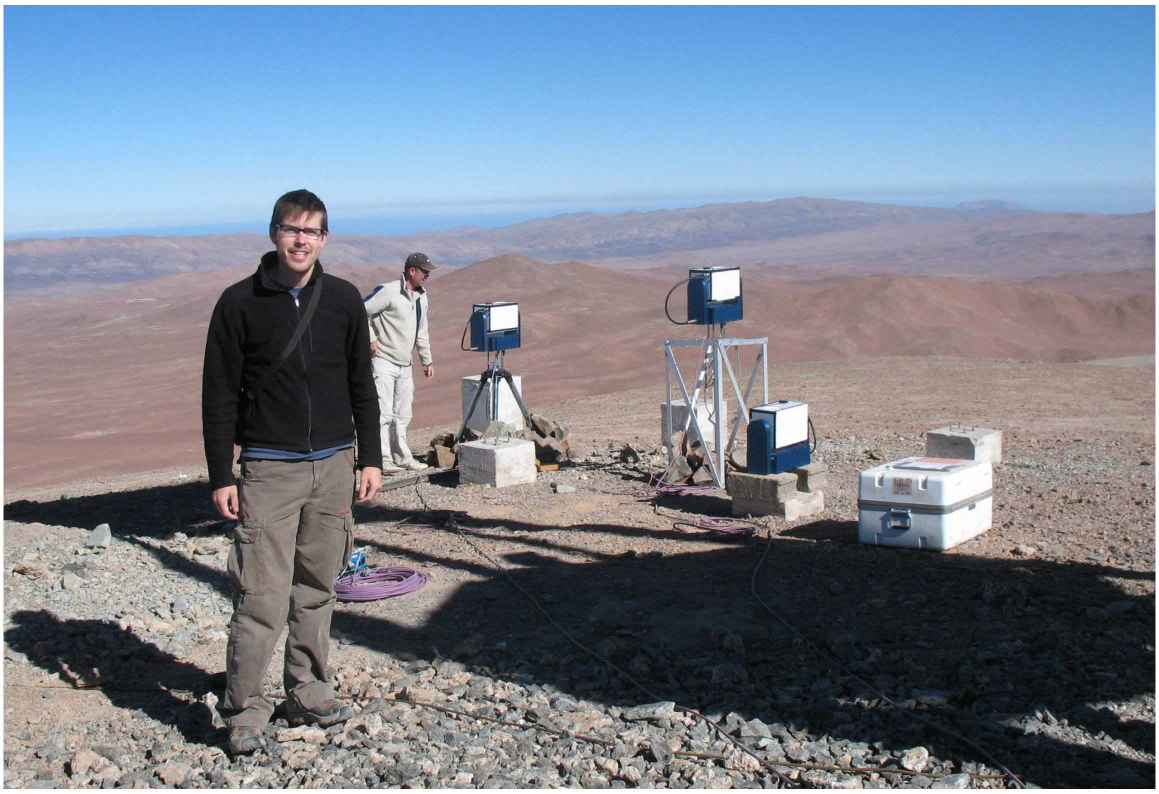


Figure 5.13: A photograph showing three IRMA units deployed on one of the Chilean sites being considered as part of the Thirty Meter Telescope (TMT) project with myself in the foreground. Mr. Greg Tompkins can be seen in the background checking one of the Units.

5.1.4.1 Field Tests in Chile

The sensitivity of IRMA to PWV measurements is a non-linear function of the altitude of the observing site and the amount of water vapour in the atmosphere. IRMA is designed to have maximum sensitivity at high altitude sites. In fact, it is not expected to perform well at the lower altitude, and thus wetter site above our laboratory.

For this reason, in Lethbridge, it is not possible to evaluate the performance of IRMA in the operating conditions for which it is designed. Therefore, to test the units under more optimal conditions, the three units were shipped to Chile. In January 2007, I had the opportunity to travel to Chile where we carefully drove the IRMA units across the

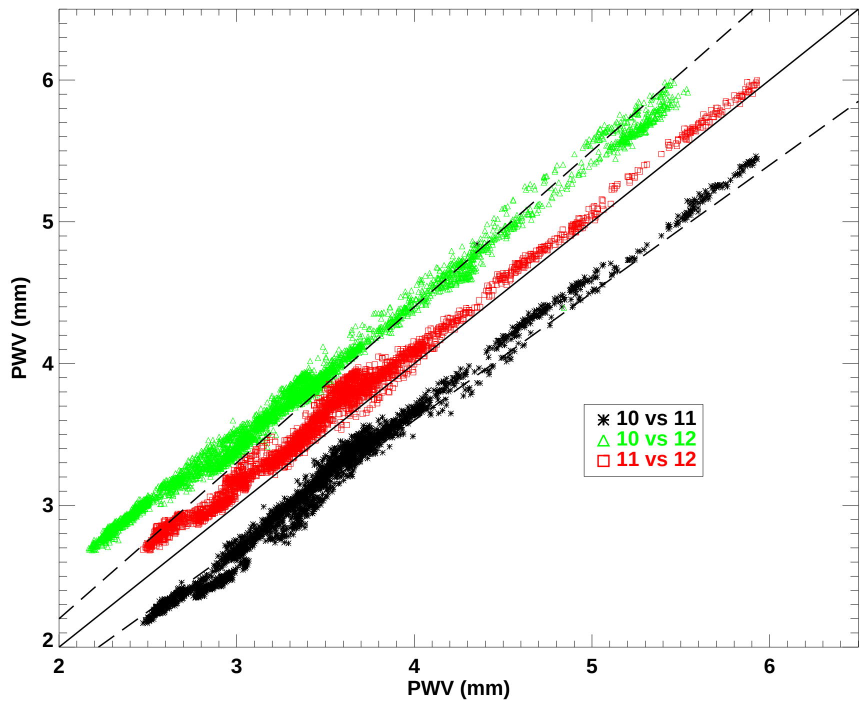


Figure 5.14: Inter-comparison scatter plot of PWV data from the three co-located IRMA units in Chile, analagous to Figure 5.11. These data were derived using the original calibration coefficients calculated during the original test campaign in Lethbridge. As can be seen and as expected this site is significantly drier than Lethbridge. It can also be seen that the systematic errors are outside our error budget target of 10% for PWV.

desert in 4x4 trucks to a 3000 m high mountain. The units were unpacked and set up side-by-side, as shown in Figure 5.13. Here, the testing procedure was repeated, and the three units were once again operated co-located observing the same atmosphere. While on site, the correlation between the units was verified. However, a small offset was evident between the units shown in Figure 5.14. It was assumed that this was because a thorough study of the internal box temperatures had not yet been completed. Upon my return to Lethbridge, the statistical study of the contributing temperature sensors was used to determine new coefficients. The results of the test campaign in Chile, using the new coefficients, are shown in Figure 5.15 [36]. Figure 5.16, with corresponding correlation coefficients listed in Table 5.2, shows that the data points are more closely correlated than with the Lethbridge data.

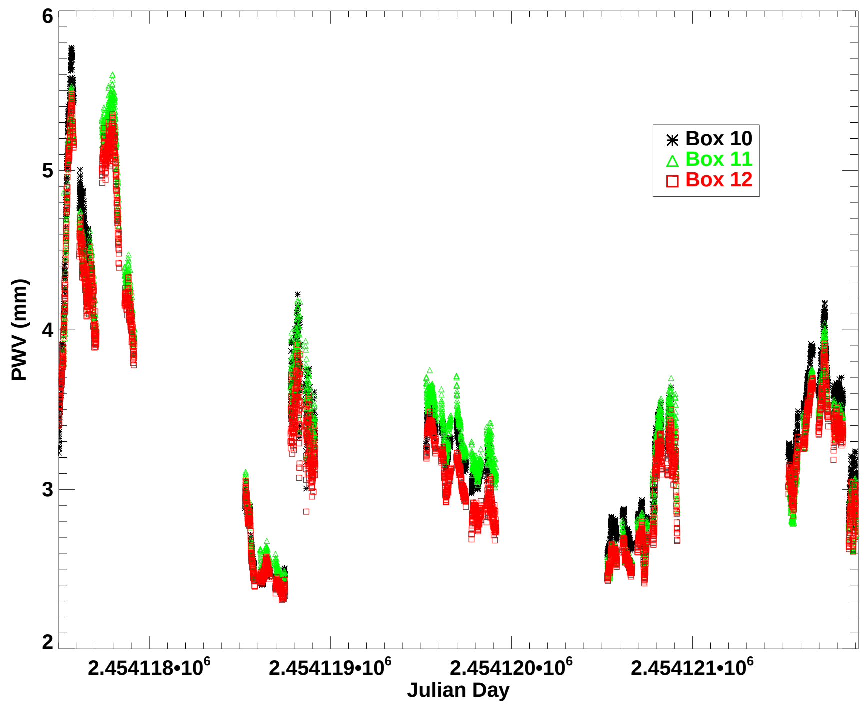


Figure 5.15: A time series plot of the three co-located IRMA units in Chile. The gaps in the data correspond to calibration sequences.These data were derived using the calibration coefficients calculated using the statistical approach described in §5.1.3.1.

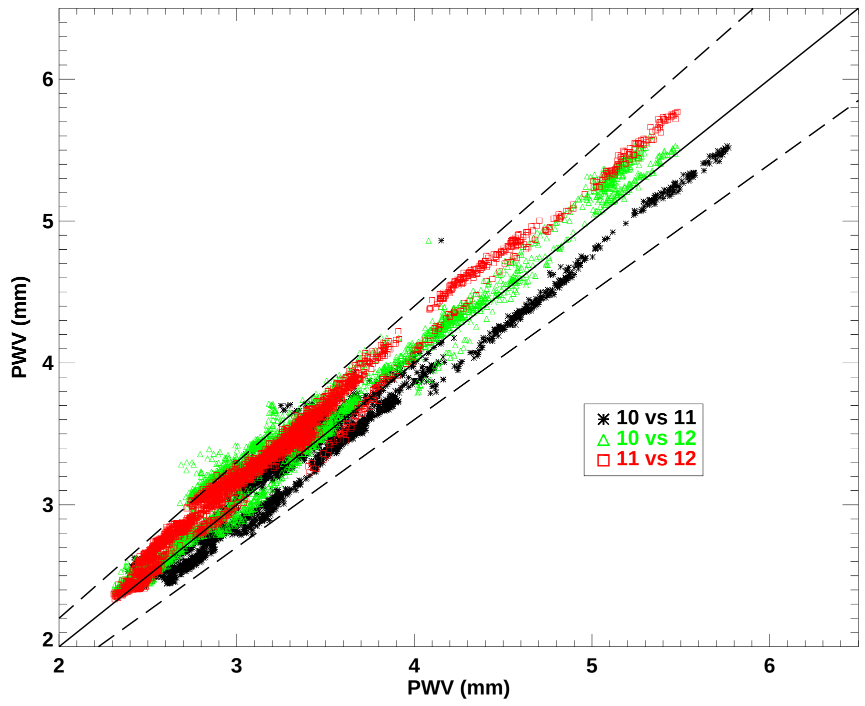


Figure 5.16: Inter-comparison scatter plot of PWV data from the three co-located IRMA units in Chile, analagous to Figure 5.11. These data were derived using the calibration coefficients calculated using the statistical approach described in §5.1.3.1.

Table 5.2: Correlation coefficients for the pairwise comparison of the Chile test data, shown in Figure 5.16, using the calibration coefficients calculated using the statistical approach described in §5.1.3.1.

Data Set 1	Data Set 2	Correlation Coefficient
Box 10	Box 11	0.9836
Box 10	Box 12	0.9939
Box 11	Box 12	0.9886

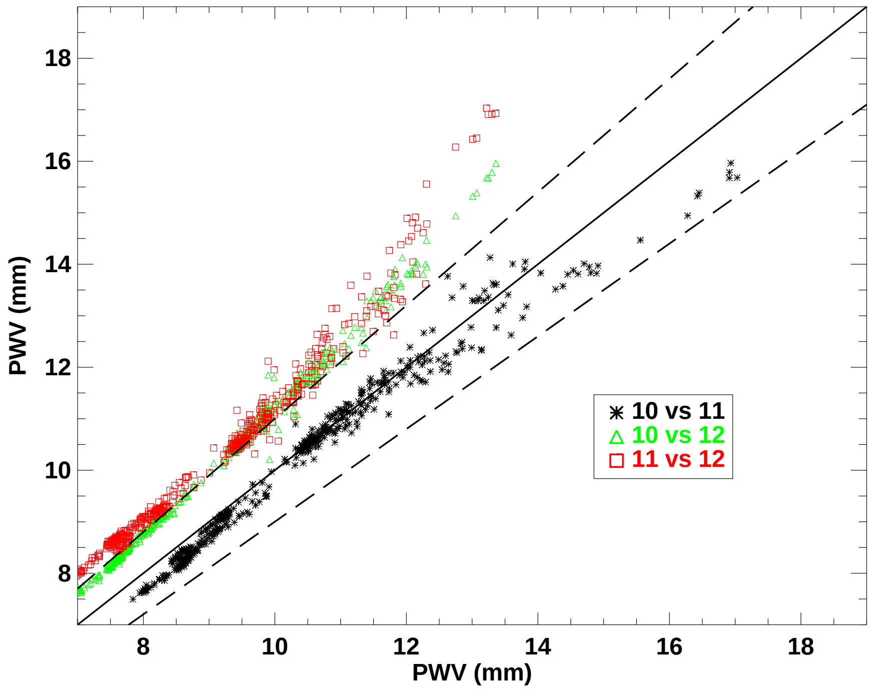


Figure 5.17: Inter-comparison scatter plot of PWV data from the three co-located IRMA units in Lethbridge; compare to Figure 5.11. These data were derived using the calibration coefficients calculated using the statistical approach described in §5.1.3.1. It can also be seen that the systematic errors are outside our error budget target of 10% for PWV.

However, when these new calibration coefficients were applied to the Lethbridge data an offset was present, as seen in Figure 5.17.

The difficulty in obtaining a single set of coefficients, which would adequate calibrate both the Lethbridge and Chilean testing campaigns, raised questions about the accuracy of the primary calibration method. While the method is fundamentally sound, there may be issues with the design of the large blackbody. When the large blackbody is placed on top of an IRMA unit during calibration, it completely encloses the unit. As a result,

Table 5.3: Correlation coefficients for the pairwise comparison of the Chile test data, shown in Figure 5.18, using the calibration coefficients calculated using the Chilean sky relative to Unit 10 as the calibration source.

Data Set 1	Data Set 2	Correlation Coefficient
Box 10	Box 11	0.9896
Box 10	Box 12	0.9869
Box 11	Box 12	0.9953

when it is heated to temperatures up to 90 °C, there is significant internal heating of the IRMA unit, which may cause exaggerated contributions to the signal voltage from some areas of the box, resulting in errors in the calculated coefficients.

In an attempt to calibrate the units under normal operating conditions, we decided to use the Chilean sky as a calibration source. Since the actual atmospheric flux is unknown, a relative calibration was performed where Unit 10 was selected as the reference, and used to determine the radiated flux from the atmosphere. The internal blackbody was then calibrated using the method described in §5.1.3, except the flux from the atmosphere, as determined by Unit 10, was used in place of the large blackbody. A set of lid coefficients was generated for each day of the testing campaign. The average value for each of the coefficients was determined, and used to process the data. A time series plot of the Chilean data is shown in Figure 5.18, while Figure 5.19 shows the correlation of the boxes with the correlation coefficients listed in Table 5.3. The correlation coefficients do not show a significant improvement over those calculated for the other calibrations because the coefficients do not account for an absolute calibration.

The coefficients were then applied to the data obtained in Lethbridge, and are shown in Figures 5.20 and 5.21. Reassuringly, there was found to be excellent correlation between the boxes (Table 5.4) and virtually no offset.

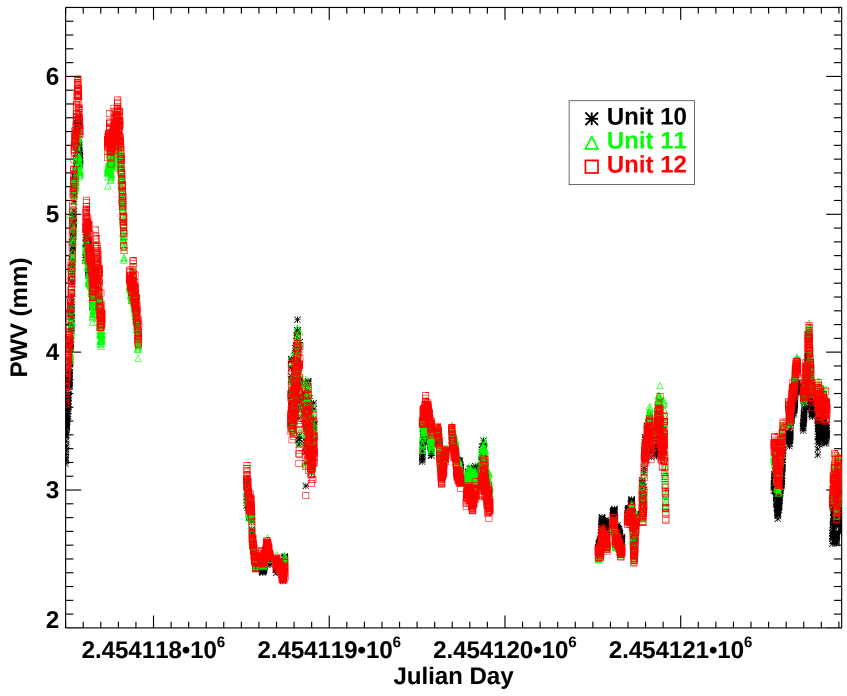


Figure 5.18: A time series plot of the three co-located IRMA units in Chile. The gaps in the data correspond to calibration sequences. These data were derived using the calibration coefficients calculated using the Chilean sky relative to Unit 10 as the calibration source.

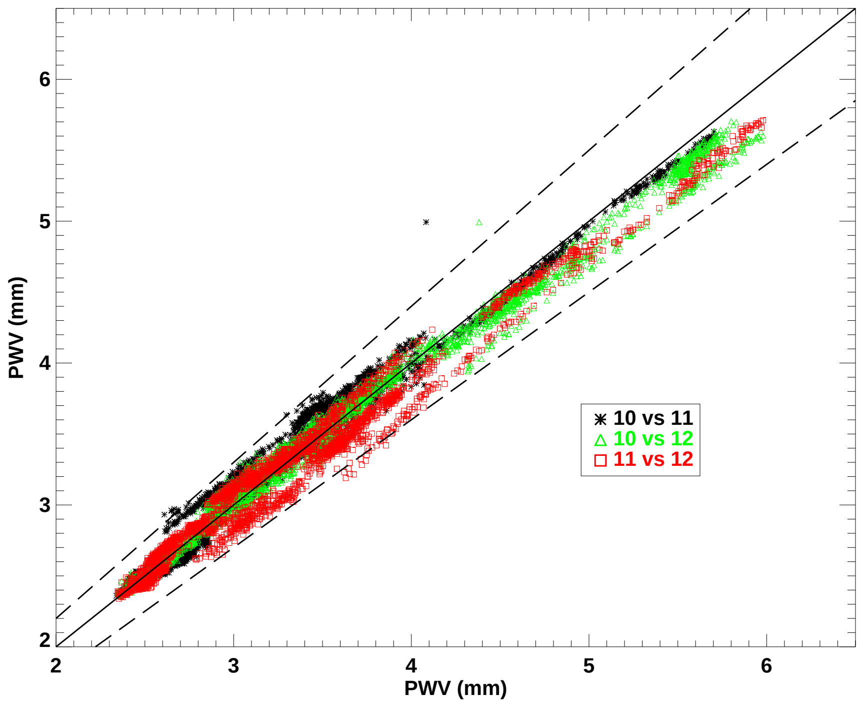


Figure 5.19: Inter-comparison scatter plot of PWV data from the three co-located IRMA units in Chile, analagous to Figure 5.11. These data were derived using the calibration coefficients calculated using the Chilean sky relative to Unit 10 as the calibration source.

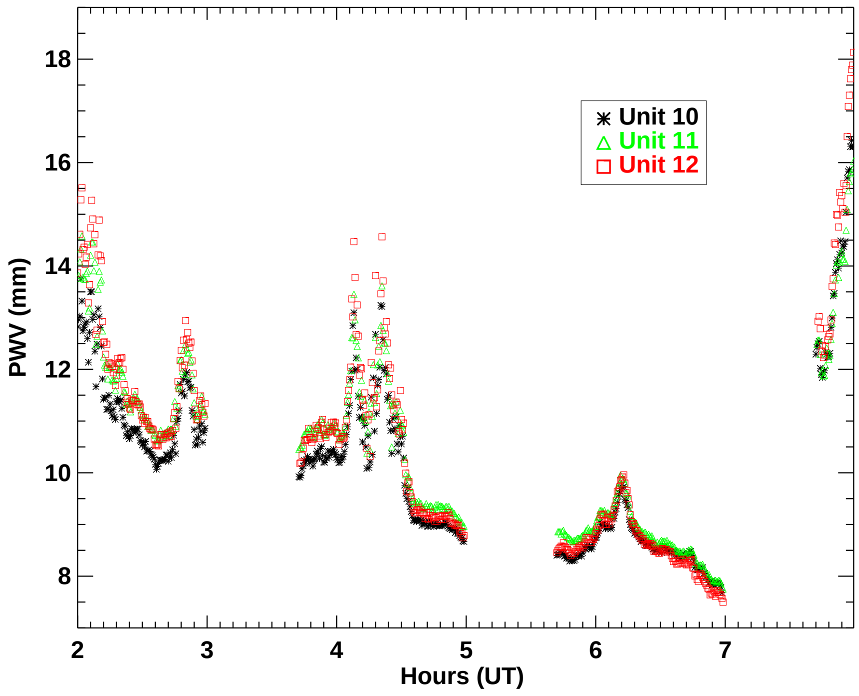


Figure 5.20: A time series plot of the three co-located IRMA units in Lethbridge. The gaps in the data correspond to calibration sequences. These data were derived using the calibration coefficients calculated using the Chilean sky relative to Unit 10 as the calibration source.

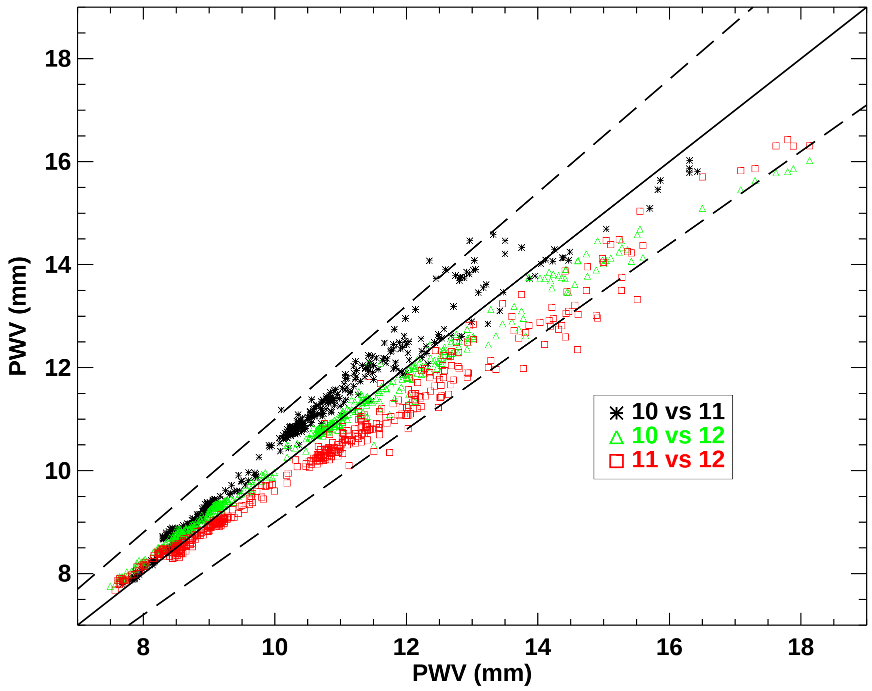


Figure 5.21: Inter-comparison scatter plot of PWV data from the three co-located IRMA units in Lethbridge, analagous to Figure 5.11. These data were derived using the calibration coefficients calculated using the Chilean sky relative to Unit 10 as the calibration source.

Table 5.4: Correlation coefficients for the pairwise comparison of the Lethbridge test data, shown in Figure 5.21, using the calibration coefficients calculated using the Chilean sky relative to Unit 10 as the calibration source.

Data Set 1	Data Set 2	Correlation Coefficient
Box 10	Box 11	0.9766
Box 10	Box 12	0.9918
Box 11	Box 12	0.9935

5.2 Field Performance

IRMA is often deployed at remote sites where there is little physical interaction with the unit. For this reason, the units must be robust, but it is possible that a mechanical failure may occur in the field. IRMA is designed to minimise single point failures, such that when such mechanical failures occur, it may still be possible to operate the unit and obtain meaningful data.

In previous versions, it was only possible to move the Alt-Az mount to an absolute position, i.e. Elevation: 70°, Azimuth: 350° West of North. However, thanks to recent updates to the Alt-Az Controller (AAC) (§3.3.3), commands were added which allowed the Alt-Az mount to make relative movements, i.e. move clockwise 30° in azimuth.

In April 2007, while a unit was deployed atop Mauna Kea in Hawaii, failures occurred which previously would have made normal operation of the unit impossible. First, a problem with the elevation drive was discovered, which was traced to a failure in an optical switch. During an initialisation routine, the unit would not recognize one of the elevation limits. This meant that the elevation initialisation routine could not be completed, and therefore, the standard move commands could not me used. At this time, the initialisation involved moving one direction, assigned counter-clockwise (CCW), until the limit was reached, then rotating in the other direction, assigned clockwise (CW), to the CW limit.

With the CW limit switch not responding, the initialisation routine would continue to drive the motor seeking the CW limit. Using the relative movement commands it was possible to verify that the elevation drive still had a full range of motion $\sim 190°$, and it was pointed to zenith using the new commands. However, the weather protection routine uses absolute movements and therefore requires a fully initialised unit.

After a standard elevation initialisation, the unit stops in the CW limit and sets its position to 0 degrees elevation even though it is actually pointing below the horizon. An offset is applied to correct this. Before a unit is shipped, the offset between the limit and the true 0 degrees is calculated and stored in a file on the unit. It was determined that we could simulate an initialisation by using an offset value to compensate for the uninitialised value, 90,000 optical encoder (OE) counts. By applying an offset of 90,000 plus the original offset, the unit would appear initialised. This solution corrected the Alt-Az problems until a repair could be scheduled.

During this same period, and before the unit could be repaired, the motor on the weather shutter failed in the open position. With the weather shutter not functioning, IRMA was seriously compromised, as it was no longer possible to calibrate the unit. Had the shutter failed in the closed position it would have been impossible to view the atmosphere at all. Although we were not able to obtain calibration measurements, we could still observe the atmosphere and so during this time, a sequence of skydips were performed.

Skydipping consists of taking measurements at increasing zenith angles and inverting the measurements to retrieve the column abundance of water vapour. In essence, a skydip is a method of increasing the number of observed water molecules in a controlled manner. Assuming a homogeneous parallel-plane atmosphere with an effective height h, the effective path length along the line of sight at the zenith angle θ is given by

$$l = \frac{h}{cos\theta} \qquad [\mathrm{m}] \quad . \tag{5.11}$$

Airmass is the path length relative to the height of the atmosphere at zenith, which under the same assumptions can be approximated as:

$$airmass = \frac{1}{cos\theta} \; = \; sec\theta \tag{5.12}$$

For example, at a zenith angle of 60° the airmass = 2, therefore IRMA is observing the emission of twice the amount of water molecules. As the airmass increases so should the observed flux, in a well defined manner.

For a single spectral water vapour line, the peak emission increases with the number of molecules, until the peak reaches the Planck envelope. At this point the increased emission is due solely to the broadening of the line. IRMA observes many lines in its spectral window (450 – 575 cm$^-$1), a portion of which can be seen in Figure 5.22 plotted at various airmasses. From this Figure it can be seen that the amplitude of the emission spectra increases in a complex manner. Integrating the spectra with respect to wavenumber, and then plotting the radiant flux against the respective airmass gives the curve shown in Figure 5.23, which is known as a curve-of-growth. The atmospheric opacity, τ, can be calculated from this curve, and is proportional to the precipitable water vapour in the atmosphere.

On the summit of Mauna Kea, the Caltech Submillimeter Observatory (CSO) Tau Meter also measures PWV by performing skydips with a 225 GHz radiometer [6], at a fixed azimuth of 316 degrees, every 10 minutes to determine the atmospheric opacity, τ_{CSO}. IRMA was pointed to approximately the same azimuth as the CSO Tau Meter while the skydips were performed. 149 skydips were performed over the course of 19 hours. Following this

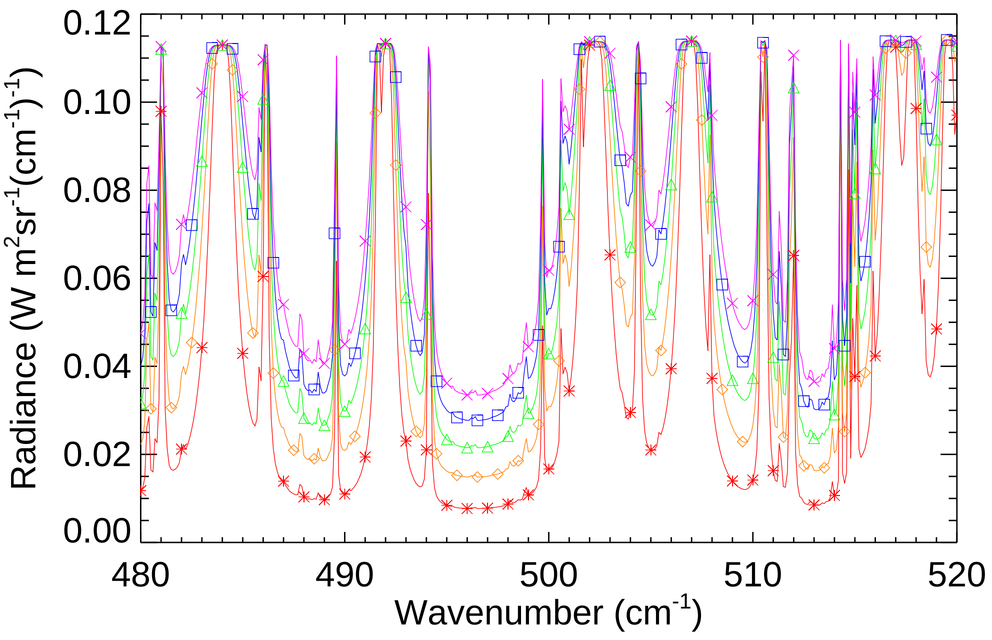


Figure 5.22: Theoretical emission spectra of the atmosphere in the 20 μ spectral region, produced by BTRAM [9]. These spectra correspond to 1 mm PWV and span an airmass range from 1 to 5.

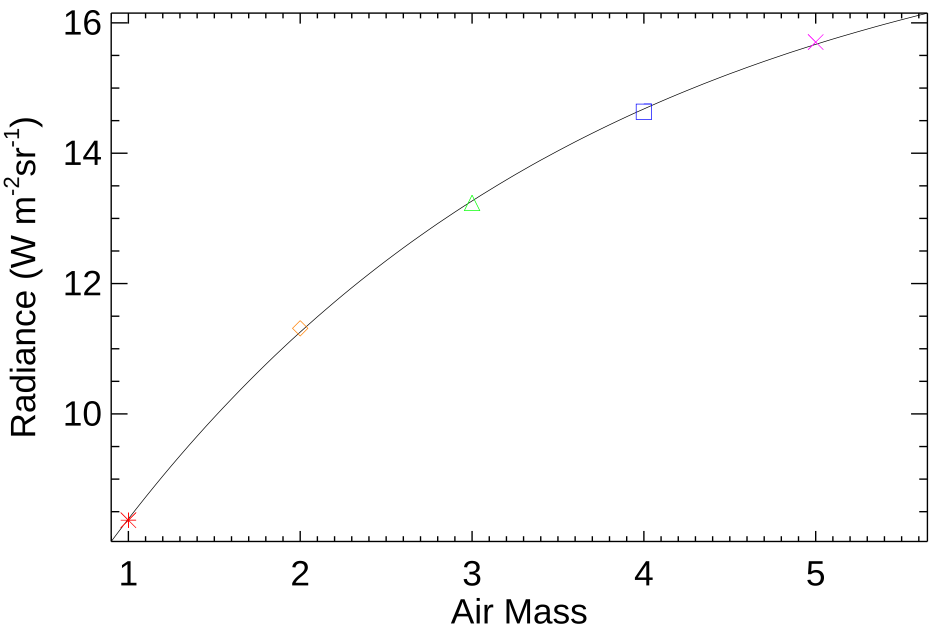


Figure 5.23: A theoretical curve-of-growth calculated by integrating flux underneath the spectra shown in Figure 5.22 as a function of airmass. The symbols represent the 5 discrete integrals and the solid line represents the continuous curve of growth.

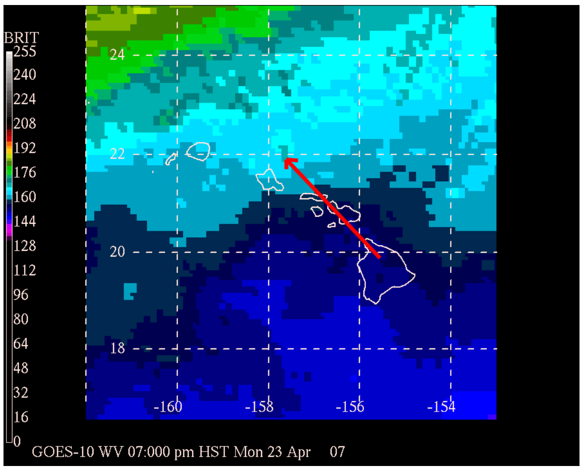
the unit was powered down and covered to await repair.

The data from the 149 skydips were compared with the CSO tau data. The $\tau_{\rm CSO}$ values were first converted to PWV using a relationship which has been published in the literature for Mauna Kea, given by [37]:

$$PWV = 20(\tau_{\rm CSO} + 0.016) \qquad [\rm mm] \ . \tag{5.13}$$

A curve-of-growth, similar to the one shown in Figure 5.23, was then generated by BTRAM using the meteorological data, and the PWV, as determined from $\tau_{\rm CSO}$. This curve of growth was then fitted to the IRMA skydip data. Although, as mentioned above it was not possible to calibrate IRMA due to the weather shutter/calibration source failed in the open position, it is instructive to compare the IRMA skydip data with the that derived from CSO tau. During these complemetary observations, a storm front was approaching the Hawaiian islands from the North-West direction. An example of an IRMA skydip and corresponding curve-of-growth for $\tau_{\rm CSO}$=0.032 (PWV=0.74 mm) taken when the storm front is distance, is shown in Figure 5.24 together with a satellite PWV image showing the location of storm front . The asterisks represent the raw IRMA skydip data, while the smooth curve represents the modeled, scaled curve-of-growth based upon the independent CSO tau data. Figure 5.25 shows the corresponding data three hours later when the storm from was approaching the Mountain. At this time the $\tau_{\rm CSO}$=0.062 (PWV=1.37 mm), and the IRMA skydip deviates from the modeled curve-of-growth.

Generally, there is seen to be good agreement between the IRMA measured and the CSO tau derived curves-of-growth when the storm front was far away, but as the storm approached Mauna Kea there was an increasing discrepancy between the two data sets. These differences can be attributed to the fact that CSO and IRMA radiometers view



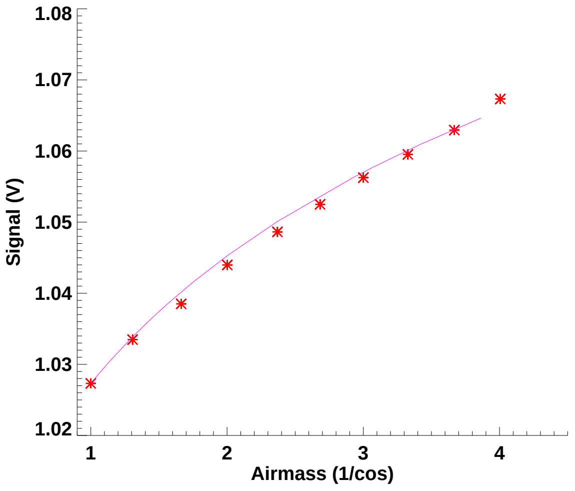
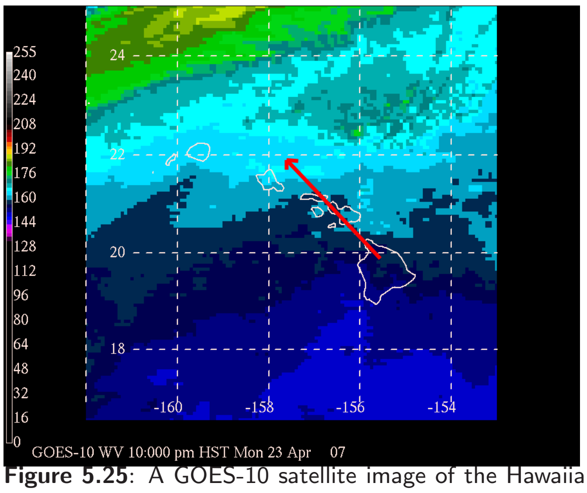


Figure 5.24: A GOES-10 satellite image of the Hawaiian Island chain taken at 05:00 UT April 24, 2007 (left). The IRMA skydip (asterisks) and the modeled curve-of-growth derived from the independent CSO tau data taken at the same time (right). There seem to be good agreement between the IRMA and the CSO tau data.



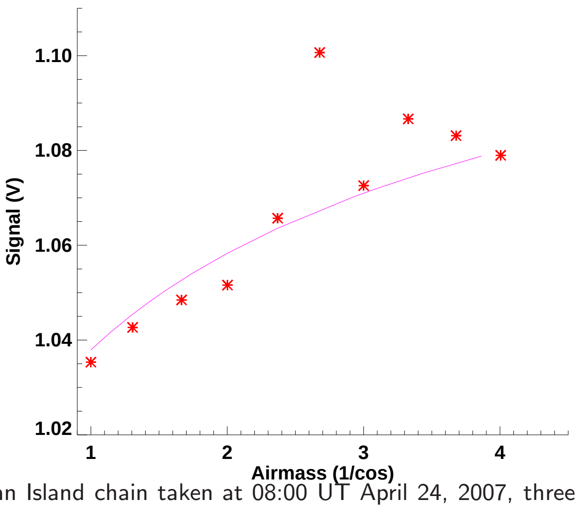


Figure 5.25: A GOES-10 satellite image of the Hawaiian Island chain taken at 08:00 UT April 24, 2007, three hour after Figure 5.24 (left). The IRMA skydip (asterisks) and the modeled curve-of-growth derived from the independent CSO tau data taken at the same time (right). It can be seen that as the storm has advanced there are now significant differences between the IRMA and CSO tau data.

different parts of the sky and have different beam sizes, which become more pronounced as the storm front approaches Mauna Kea. Although the uncalibrated IRMA data are of limited scientific value Figures 5.24 and 5.25 shows the effectiveness of IRMA in sensing impending inclement weather.

5.3 Summary

The prototype IRMA used a simple, two-point, linear calibration method. However, the latest mechanical designs posed some problems for this method. With the elimination of wet cryogens, the two calibration points moved from liquid nitrogen, LN_2, and ambient temperatures to ambient and $\sim$20 K above ambient temperature. As a result, the calibrated sky measurements need to be extrapolated, rather than interpolated, from the calibration points, increasing the required precision of the thermometry. A gradient was found to be present across the internal blackbody calibration source when heated making it more difficult to accurately determine its effective temperature. In addition, it was found that the signal voltage was also sensitive to stray internal radiation.

To address these challenges, an additional aperture was attached to the window of the cryocooler, which helped reduce, though not eliminate, the sensitivity to stray radiation. A large reference blackbody was constructed to act as a primary calibration source. A new calibration method was developed, which associated the internal blackbody (secondary) to the reference blackbody (primary) and determined the signal dependence on stray radiation.

Once calibrated, three units were tested in Lethbridge side-by-side, viewing the same atmosphere. It was shown, using the new calibration method, that there was very good agreement between the units. In order to evaluate the performance of the units under the operating condition for which IRMA was designed, the units were sent to Chile where they again observed the same atmosphere while on a mountain at an altitude of $\sim$3000 m. As expected, the units showed an even better correlation, with measured inter-unit correlation coefficients of 0.99 under these conditions.

The second section of this chapter demonstrates the flexibility of the IRMA unit.

During a time of instrumental failure it was possible to modify the operation of the unit remotely such that useful data could be obtained. With the Alt-Az mount damaged and the weather shutter stuck open, skydips were performed in the same direction as the Caltech Submillimeter Observatory (CSO) taumeter, an independent measure of water vapour. These results were compared and showed a good correlation.

Chapter 6

Future Directions

6.1 Overview

Since the development of the prototype, evolution has occurred in all aspects of the IRMA design. The latest improvements have been discussed in my thesis. This chapter outlines some of the areas where further improvements will allow IRMA to better perform its task of measuring atmospheric water vapour. Improvements in software will make the unit more autonomous, reliable, and robust. Modifications to the mechanical and optical design will improve the signal quality, reduce noise, and also contribute to the robustness of the instrument.

6.2 Future Mechanical and Optical Development

Work is currently underway to build the next generation of IRMA, which features a redesign of the mechanical and optical systems (Figure 6.1). Fortunately, the electronics and control software are largely unaffected by these changes.

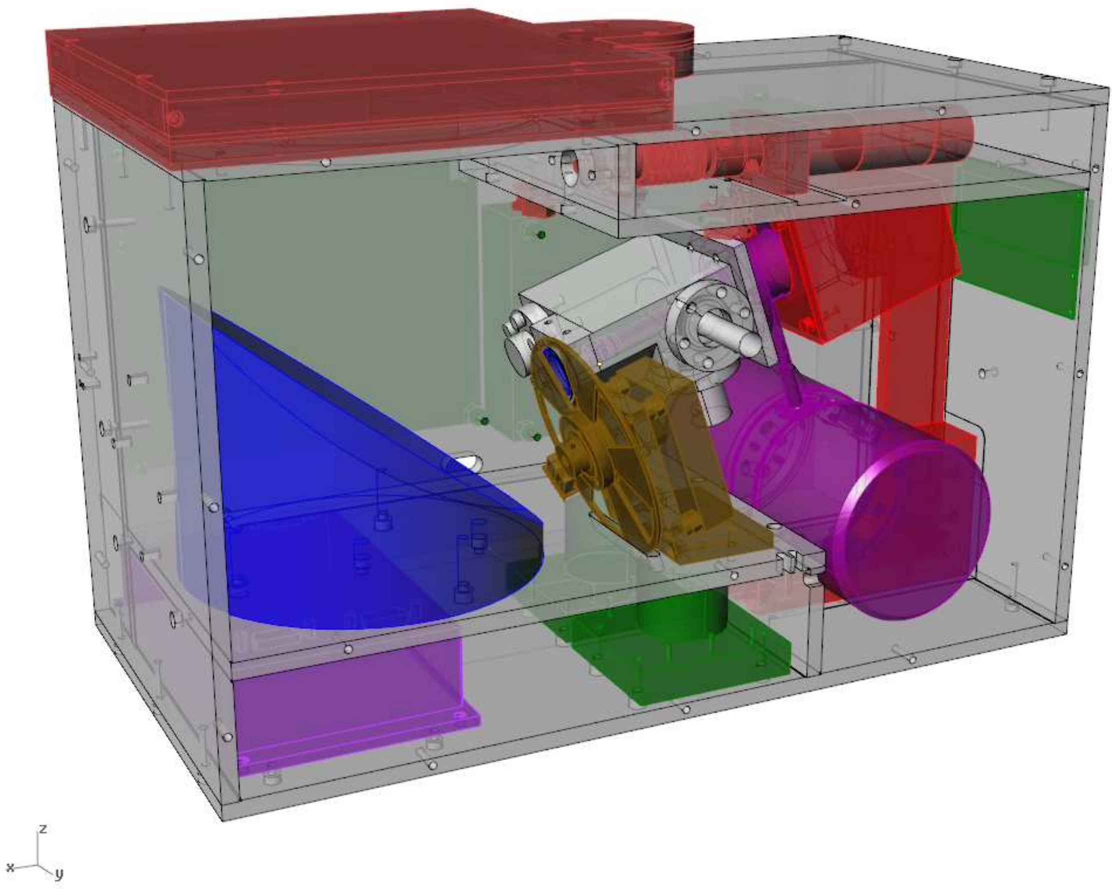


Figure 6.1: New IRMA mechanical/optical design. Courtesy of Brad Gom.

6.2.1 New Cryocooler

The redesign was driven by the the decision of Hymatic, the makers of the cooler, to discontinue the cryocooler model NAX025-001, used in the previous design. The new cryocooler from Hymatic, model SAX101 was chosen for the next generation IRMA. The new cryocooler provides many improvements over the previous model. It made use of two dual opposed linear compressors mounted on a central block. The dual opposed pistons minimize the vibration and acoustic noise from the cooler, while doubling the cooling power.

Reducing the vibrations should also improve the signal to noise ratio of the detector. The SAX101 cryocooler has the advantage of using the same cooler controller as the current version of IRMA, so the hardware and software interfaces to the cooler will be identical to the present model of IRMA. Unfortunately, however, the SAX101 cryocooler is larger than the previous model, which necessitates mounting the cooler such that the detector is at an 45° angle to the mirror, as shown in Figure 6.1. Mounting the cooler in this way may pose challenges to aligning the detector.

6.2.2 Larger Primary Mirror

As mentioned, fitting the new cooler into the restricted dimensions of the IRMA module, necessitated installing it at an angle. While mounting the cooler in this way poses some challenges, it allows for the installation of a larger, 12.5 cm diameter, primary mirror. Having a larger primary mirror increases the collection area and therefore increases the amount of flux incident on the detector. The increase in flux will increase the signal from the detector and thereby increase the signal to noise ratio.

6.2.3 New Weather Shutter Design

The 12.5 cm diameter mirror requires a larger viewport for IRMA. Thus, the design of the weather shutter needed to be modified. The redesign provided an opportunity to find solutions to some of the problems with the shutter design. The current weather shutter has been the source of some failures such as the incident mentioned in §5.2. In the new design, rather than retracting inside the unit, the weather shutter rotates out of the beam, which will result in less heating of the optical cavity. The blackbody, on the underside of the new weather shutter, will contain an array of sixteen temperature sensors, rather than only

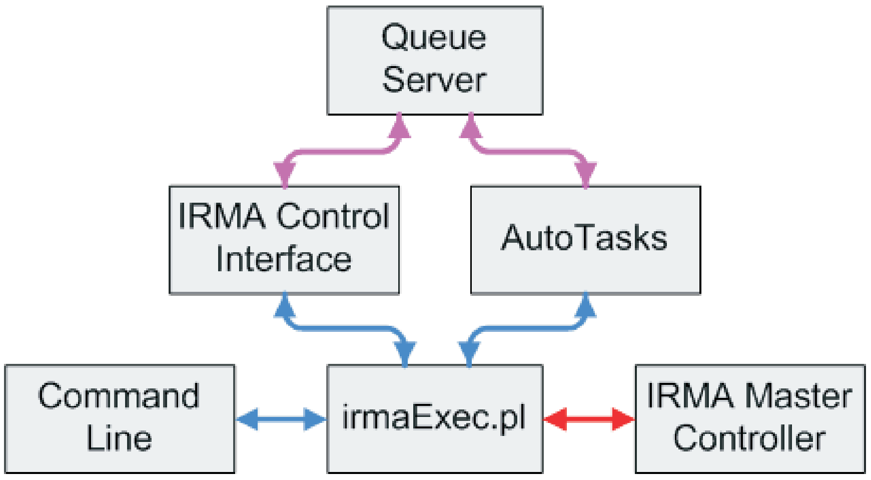


Figure 6.2: Block diagram of original Queue Server functionality.

two, which will allow for a more accurate mapping of any temperature gradients access the blackbody surface and a more accurate calculation of the effective temperature of the blackbody calibration source.

6.3 Future Software Development

The IRMA related software will continue to evolve. This includes enhancements to both the system software, resident on the PC/104 discussed in Chapter 3, and the front-end software discussed in Chapter 4.

6.3.1 System Software Enhancements

6.3.1.1 Queue Server Integration

As mentioned in §3.4, the *Queue Server* is not required to prevent two scripts from attempting to access `irmamc` on the PC/104 based IRMA since the TCP/IP requests

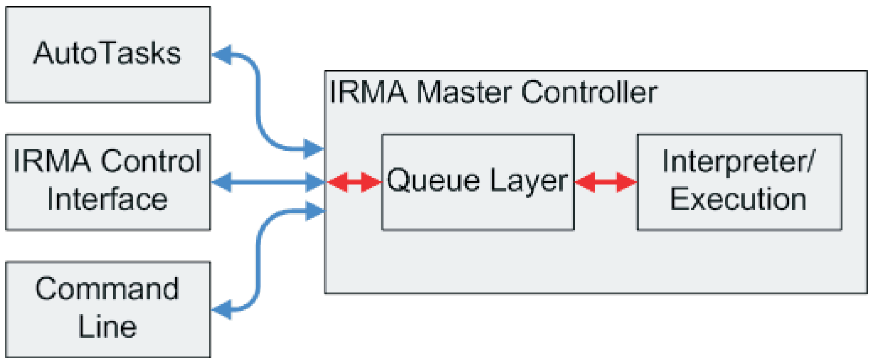


Figure 6.3: Block diagram of proposed Queue functionality.

are queued on the TCP stack. However, there are some features of the *Queue Server*, not related to conflict resolution, which would still be desirable on the system. These include the ability to view scripts waiting to execute, removing items from the queue, and modifying the priority of a queued item. Rather than simply continue to use the current implementation, the *Queue Server* should be integrated into `irmamc`. Currently the *Queue Server* simply acts as a "take-a-number box", yet has no direct interaction with the IRMAscript interpreter, as shown in Figure 6.2. This adds complexity to the programs which use the *Queue Server*, such as, *AutoTasks* and the *IRMA Control Interface*, because they must include the communication protocols for both the *Queue Server* as well as the IRMAscript interpreter. Additionally, the current *Queue Server* implementation does not allow for scripts to be added through the command line, and therefore these scripts are not visible from the *Queue Server*.

If the *Queue Server* is integrated into the *Master Controller*, all executed scripts, regardless of how they are executed, would be inserted into the queue, shown in Figure 6.3. Unlike the original *Queue Server*, the queue functionality will be completely abstracted from the user, or programs sending commands to `irmamc`. Whenever a IRMAscript is received, it will automatically be entered into the queue. Additional commands will be added to query the status or modify the queue. Having the *Queue Server* tightly integrated into `irmamc`, will greatly simplify the design of programs which will interact with `irmamc`.

6.3.1.2 Improved Autonomous Operation

Currently, `irmamc` is only capable of executing IRMA scripts. Autonomous control is carried out by additional programs which generate scripts and send them to `irmamc`, such as *Autotasks* (§3.5). With the adoption of the PC/104 based platform, all of the system software is executed on the same processor, so these programs can be integrated into `irmamc`. This would reduce overhead and allow for more efficient operation. In addition, with the integration of the *Queue Server* and *AutoTasks* into the *Master Controller* the system would be written almost entirely in C; perl, which runs slowly on the PC/104 due to the increased overhead of the Perl interpreter, would be used only for very simple housekeeping tasks.

6.3.1.3 IRMAscript Modification

IRMAscript allows for flexible control of the radiometer. However, in most cases instrument control is handled by short scripts which handle an individual task. This is due to the fact that only one script can be executed at a time; if a large complex script were executing, all other scripts would be required to wait. The subsequent delay could be catastrophic if, for example, the command in the queue is a weather protection command. Future software development will likely see IRMAscript reduced from a full scripting language with variables and flow control to a series of instrument control commands. The flow control would then be handled by the program interfacing with the IRMA unit. Reducing

IRMAscript in this a manner would not only reduce the complexity of the IRMA code, but make it easier for an operator to write custom interfaces with the unit.

6.3.2 Front End Software Development

As discussed in §4.3, improvements in the calibration method required writing a new data reduction program, *IRMA PWV* which was written in IDL® and performed some of the functions of the *IRMA Archive Interface* (§4.2). Future development of *IRMA PWV* will include adding a GUI interface, as well as a complete suite of data reduction and viewing functions rendering the *IRMA Archive Interface* obsolete. It is also foreseeable that IRMA control commands will be added as well, which would incorporate the functionality of the *IRMA Command Interface*. As a result, all of the data reduction and control functions would be contained in one convenient package which could be run on any platform using the IDL® Virtual Machine.

6.4 Final Thoughts

Since the development of the IRMA prototype in 1999, the IRMA project has been continually progressing. To date, the IRMA project has been the subject of 4 theses from the Astronomical Instrumentation Group at the University of Lethbridge [1] [33] [24] [13]. Two of these theses focused on the instrumentation and software design. I joined the group as the IRMA project was entering its latest design change; moving from a distributed Rabbit based system (§3.2) to the current PC/104 based system (§3.3). To support all of the existing systems, it was necessary to become familiar with both IRMA designs, in addition to the graphical interfaces which supported them. In all, this involved familiarising myself with over 68,000 lines of code.

As I gained greater understanding of the system, I was able to make improvements to the operation, from minor bug fixes to the design changes mentioned in this thesis. The most significant update, was the porting of the *IRMAscript* interpreter from Perl to C, and incorporating it into the Master Controller (§3.3.1). This allowed for over an order of magnitude improvement in the execution time and reduced the complexity of the system by eliminating the need for a special communication protocol between the Command Processor and Master Controller. Approximately 2,800 lines of code were added for this change, while ∼14,000 lines of code were eliminated. During this time, we increased our understanding of the IRMA instrument, which lead to the development of an improved calibration method (§5.1.3). IRMA has now been deployed around the world and will continue to improve and evolve.

Fin.

Appendix A

IRMAscript

A.1 Overview

An IRMAscript statement is structured simply. For commands that directly control IRMA, a command statement consists of a command type, followed by two modifiers, and zero to fifteen arguments. The command type and its two modifiers define a unique command. The arguments are provided in order to pass information pertinent to the command to IRMA. Most command statements, with the exception of the Alt-Az moveto/slewto commands, have zero or one argument. In addition to IRMA commands, IRMAscript provides variables, data assignment, arithmetic, system commands (such as reading system time), looping mechanisms, lists, and flow control, and console I/O. They do not follow the same command structure described above.

Whitespace is used to delimit, or separate, each of the elements (command type, modifiers and arguments) that make up an IRMAscript statement. Whitespace can consist of spaces or tabs. Each statement must terminate with a carriage return. Only one statement can appear on one line, which precludes IRMAscript from being a free form language, such as C or C++. IRMAscript is caseless; it does not matter whether IRMAscript statements are written in upper or lower case letters. Within the interpreter, all statements are converted to uppercase.

A.2 IRMAscript Language Summary

The following table lists all IRMA system commands addressable within the IRMAscript language. Non-system commands, such as flow control commands, are not listed.

Command	**Modifier 1**	**Modifier 2**	**Arguments**
STARTPROG	SOCKET	OPEN	
ENDPROG	SOCKET	CLOSE	
CRYO	STATE	ON	
CRYO	STATE	OFF	
CRYO	SET	MANUAL_MODE	
CRYO	SET	AUTO_MODE	
CRYO	SET	STOPPED_MODE	
CRYO	SET	SET_POINT	tempKelvin
CRYO	READ	COMP_AMP	
CRYO	READ	SET_POINT	
CRYO	READ	MODE	
CRYO	READ	CURR_TEMP	
CRYO	READ	OSC_FREQ	
CRYO	SERIAL	OPEN	
CRYO	SERIAL	CLOSE	
GPS	READ	DATE_TIME	
GPS	READ	EPOCH_TIME	
GPS	READ	LAT_LON	
GPS	SERIAL	OPEN	
GPS	SERIAL	CLOSE	
ADC	INIT	RESYNCH	
ADC	INIT	RESET	
ADC	INIT	RW_TEST	
ADC	SET	CSR	chan,gain,wordRate,polarity
ADC	SET	GAIN	channel, gainValue
ADC	SET	OFFSET	channel, offsetValue
ADC	SAMPLE	NO_INT	channel
ADC	SAMPLE	ON_INT	channel

ADC	READ	CSR	channel
ADC	READ	GAIN	channel
ADC	READ	OFFSET	channel
ADC	READ	CONFIG_REGISTER	
SHUTTER	STATE	OPEN	
SHUTTER	STATE	CLOSE	
SHUTTER	READ	LIMIT	
SHUTTER	READ	OVERCURRENT	
SHUTTER	SET	OC_RESET	
CHOP_MOTOR	STATE	ON	
CHOP_MOTOR	STATE	OFF	
CHOP_MOTOR	STATE	MEASURE_RPM_ON	
CHOP_MOTOR	STATE	MEASURE_RPM_OFF	
CHOP_MOTOR	READ	STATE	
CHOP_MOTOR	READ	RPM	
BB	STATE	ON	
BB	STATE	OFF	
BB	READ	STATE	
ALTAZ	MOVE_TO	DMS	elD,elM,elS,azD,azM,azS,spd
ALTAZ	STATE	POSLOG	poslog_enable/poslog_disable
ALTAZ	STATE	HALT	
ALTAZ	STATE	REBOOT	
ALTAZ	INIT	PING	
ALTAZ	INIT	ALTAZ	
ALTAZ	INIT	AXES	ELEVATION/AZIMUTH
ALTAZ	INIT	SERVO	
ALTAZ	INIT	MOTOR	
ALTAZ	SET	ALT_OFFSET	offset
ALTAZ	SET	AZ_OFFSET	offset
ALTAZ	READ	POSITION	
ALTAZ	READ	TASK_STATUS	
ALTAZ	READ	ALT_OFFSET	
ALTAZ	READ	AZ_OFFSET	
ALTAZ	READ	POSLOG_RANGE	
ALTAZ	READ	POSLOG_DATA	
ALTAZ	READ	POSLOG_STATE	
ALTAZ	SERIAL	OPEN	
ALTAZ	SERIAL	CLOSE	
ALTAZ	SLEW_TO	DMS	elD,elM,elS,azD,azM,azS,spd
RTC	SET	DATE_TIME	
RTC	READ	DATE_TIME	
RTC	SET	ARBITRARY_TIME	YYYY-MM-DDThh:mm:ss
SCAN	SIGNAL	ON_INT	
SCAN	SIGNAL	STOP	

SCAN	READ	STATE	
IRMA	STATE	OFF	
IRMA	READ	UPTIME	
SUN_SENSOR	READ	STATE	
SUN_SENSOR	READ	SHUTTER_STATE	
SUN_SENSOR	STATE	SHUTTER_OPEN	
SUN_SENSOR	STATE	SHUTTER_CLOSE	
NOTCH_FILTER	STATE	60HZ_IN	
NOTCH_FILTER	STATE	60HZ_OUT	
NOTCH_FILTER	STATE	120HZ_IN	
NOTCH_FILTER	STATE	120HZ_OUT	
NOTCH_FILTER	READ	60HZ	
NOTCH_FILTER	READ	120HZ	
BANDPASS_FILTER	STATE	IN	
BANDPASS_FILTER	STATE	OUT	
BANDPASS_FILTER	READ	STATE	

A.3 IRMAscript Language Definition

A.3.1 List Manipulation

INITIALIZATION

Construct (initialize) a list with one or more elements.

LENGTH

Return the length of a list.

INDEX

Reference an element of a list, where the index ranges from 0 (the first element) to n.

SUBSTRING

Retrieve a substring from a colon delimited data record. In IRMA commands that return multiple data items, such as `ALTAZ INIT PING`, data is returned as a colon delimited string. This command splits the data string into its constituent data items and returns the desired datum, based on an index value.

A.3.2 Utility Functions

DEG2DMS

Convert an Alt-Az coordinate expressed as floating point degrees into degree-minute-second (DMS) format. The degrees, minutes and seconds must be variables because the deg2dms function places values in these variables. They are not input variables.

STARTPROG SOCKET OPEN / ENDPROG SOCKET CLOSE

Open and close a TCP/IP stream socket connection to the IRMA master controller. If a script contains instructions to execute on the IRMA master controller, a network socket must be established to the IRMA MC, as low-level IRMA system commands and data flow over this connection. If a script does not contain IRMA hardware control commands, it is not necessary to wrap a script with these statements.

LOCALHOST

This command handles system functions performed by the host computer's operating system.

> **localhost log open**
> Open the log file. A log file name must be created using the `new log filename` command before logging can commence.
>
> **localhost log close**
> Close the log file.

NEW

The `new` family of functions creates new data items of various types, such as filenames and time stamps.

> **new log filename**
> Automatically generate and return a filename, and create a directory path for the new file. Filenames generated by this function follow the ISO time format:

YYYY-MM-DDTHHmmSS.dat

and end with the `.dat` extension. File paths follow the structure:

/IRMAdata/IRMA_<boxNumber>/YYYY/YYYY-MM-DD

where `/IRMAdata/` is a link (or filesystem shortcut) to some directory where IRMA data is stored, **<boxNumber>** is the IRMA unit's identifier number, **YYYY** is the year in which the data/log file was created, and **YYYY-MM-DD** is a year-month-day time stamp. This directory format organizes data files chronologically according to the particular unit.

new iso timestamp
Create a time stamp string conforming to the ISO date-time format:

YYYY-MM-DDTHH:mm:SS.sss

Where **YYYY** refers to year, **MM** to month (1-12), **DD** to day (1-31), **HH** to hour (0-23), **mm** to minute (0-59), **SS** to second (0-59), and **sss** to milliseconds (0-999). The symbols **-**, **T**, and **:** are delimitation symbols.

A.3.3 Variable Manipulation

ASSIGN

Assign a value to a variable. The source of the assignment can be literal or another variable. Literal values can be numeric or strings. Strings can be defined with or without enclosing double quotes. When quotes are used, it is permitted to include whitespace in the string.

INCR / DECR

Increment or decrement a value contained in a variable. This operation does not work with literals, as literals cannot have values assigned to them.

EVAL

Perform arithmetic operations and assign results to a variable. This command precedes a simple arithmetic statement involving two operands and one operator. The operations available are addition, subtraction, multiplication, division, modular division, and exponentiation. The operands can be literals or variables, but the result must be assigned to a variable.

A.3.4 Delays

WAIT

Delay execution of the script by N seconds. N can be a real value, ranging from 0 to some arbitrary value.

A.3.5 Flow Control

DO .. WHILE

While loops repeatedly execute a block of statements while some arbitrary condition is logically evaluated to be true. With **do .. while** statements, the condition is tested at the end of the block, as opposed to the beginning of the block, which occurs in **while** loops. A **do .. while** loop in IRMAscript opens with a **do** statement, and closes with a **while** ***condition*** statement. Any number of IRMAscript statements, including other **do .. while** loops, can be included in this block. There is no limit to the number of **do .. while** loops that can be nested within one another.

The condition can take two forms: a simple comparison involving two operands, or a compound conditional statement that logically ANDs or ORs two comparisons. For example, a simple conditional statement takes the form `$x < $y`, while a compound conditional is structured `$x < $y or $a = $b`.

Four kinds of comparison are available: less than (`<`), greater than (`>`), equality (`=`), and inequality (`!=`). Logical ANDing and ORing can be specified in a compound conditional using the symbols `and` and `or`. Do not use the symbols `&&` or || to perform logical evaluations.

REPEAT .. ENDLOOP

Repeat execution of a block of statements. This structure is equivalent to a **for** loop that increments from 0 to n.

GOTO

The most basic flow control mechanism is the goto statement. When the IRMAscript interpreter executes **goto *label*** statement, program control jumps to the IRMAscript statement immediately following the **label *labelName*** statement. Using GOTOs as a form of program flow control can lead to unstructured, unmanageable code. However, in the context of IRMAscript, whose scripts tend to be quite short (less than a printed page long), the issue of structured GOTO-less programming is not important. Given the relatively primitive flow control mechanisms available in IRMAscript, GOTO allows the programmer to develop sophisticated flow control within an IRMA script. With labels, the use of a colon after the label name is optional.

A.3.6 Input / Output Commands

PRINT

Feedback from an executing IRMAscript can be directed to the console (or shell) by means of the `print` command. The argument to the print command can be a literal or a variable. In its most simple form, `print` can accept bare literals, either text or numeric, which is inconsequential to IRMAscript, as it is a typeless language. If a string literal enclosed in double quotes is passed as the parameter, the user can format the output, mixing variables and literals together. The only stipulation is that each item in the string, whether literals or variables, must be separated by commas, and there must not be any whitespace between the quotes. The reason for the prohibition on whitespace is that the IRMAscript interpreter divides statements into their constituent parts (tokens) along whitespace divisions. Two special literals can be used within printf strings: the **s** symbol defines a single whitespace, while the **n** symbol defines a linefeed, and is often called a newline character.

Output can be directed to an open log file by including the `log` modifier immediately after the `print` command. The methods for defining the string format is identical to the standard `print` command. The `localhost` command has methods to open and close logfiles. Furthermore, the `assign` command can be used to define strings that can be assembled using the `print` command.

A.3.7 System Commands

The following group of commands are responsible for controlling and/or reading data from IRMA's hardware components, which includes the AAC.

NOTCH_FILTER

The **notch_filter state [*filter*]** commands enable or disable the 60 Hz notch filter. The filter is enabled with the `60hz_in` parameter, and disabled with the `60hz_out` parameter. Reading 60 Hz notch filter state can be done with the **notch_filter read 60hz** command. A return value of 0 (zero) indicates that the filter is not enabled, while a return value of 1 indicates that the filter is enabled.

BANDPASS_FILTER

The **bandpass_filter state** [***in/out***] is used to enable or disable the 455 Hz bandpass filter, whose job is to filter out all frequencies above and below the 455 Hz chopper wheel frequency. The filter is enabled with the `in` parameter, and disabled with the `out` parameter. The state of the bandpass filter can be read with the **bandpass_filter read state** command. A return value of 0 (zero) indicates that the filter is not enabled, while a return value of 1 indicates that the filter is enabled.

SHUTTER

The command **shutter state** [***open/close***] signals the shutter control circuitry to respectively open or close the shutter. Once this command is issued, it cannot be aborted. The shutter will open or close until it has reached its destination position. Shutter condition during actuation can be read with the **shutter read limit** command. The following integer codes are returned: 3 - shutter is in the process of moving during shutter movement, 2 - shutter is closed (covering the optical aperture), and 1 - shutter is in the open position (optical aperture is exposed). Shutter jams can be detected by looking for an increase in the amount of current going to the shutter motor. The shutter overcurrent bit is set when this condition occurs. Calling the **shutter read overcurrent** statement returns the value of the overcurrent bit: 1 when the overcurrent condition exists, and 0 when it does not. When the overcurrent condition bit has been set, it must be reset to zero by calling the **shutter set oc_reset** command.

BB

The **bb state** [***setting***] command enables or disables the blackbody shutter heater. The heater is turned on by calling this command with the argument **on**, while **off** turns the blackbody heater off. The state of the heater can be read by calling the command **bb read**

state. The return value 1 indicates that the blackbody heater is on, while a return value of 0 (zero) indicates that it is off.

CHOP_MOTOR

The 450 Hz chop wheel is controlled and monitored by means of the **chop_motor** family of commands. The chop wheel is turned on or off by the **chop_motor state** ***setting*** command, where ***setting*** can be set to **on** or **off**. **chop_motor read state** reads the chop wheel status, returning the value 1 if the chop wheel motor is on, and 0 if it is off.

To read the chop wheel's angular speed in revolutions per minute (RPM) IRMA counts the number of interrupt pulses from the chop wheel over a selected period of time. Consequently, one cannot simultaneously perform a data collection scan and measure chop wheel angular speed. To perform a measurement, one turns the chop wheel on, then issues the command **chop_motor state measure_rpm_on**. The next step is to wait for a period of time, using the **wait** ***seconds*** command. The longer the time spent in angular speed measurement mode, the more accurate the average angular speed value will be. Wait periods ranging between 30 and 60 seconds provide adequate results. After the wait period has passed, one takes IRMA out of measurement mode with the command **chop_motor state measure_rpm_off**, then reads the resulting value with the command **chop_motor read rpm**.

RTC

Current time on the MC's RTC is read with the command **rtc read date_time**. Returned is a colon-delimited string containing the current date time: ***year*** **:** ***month*** **:** ***day*** **:** ***hour*** **:** ***minute*** **:** ***second***. As an example, February 12, 2005, at 3:37:49 PM would be returned as **2005:2:12:15:37:49**. Months range from 1 to 12, days range from 1 to 31, hours range from 0 to 23, and minutes and seconds range from 0 to 59.

Current time can be read using the **rtc read epoch_time** command. This command is convenient for timing events within an IRMA script, as it returns a 32-bit unsigned integer number representing the current time as the number of elapsed seconds since midnight of January 1, 1980.

User-defined date-time can be set using the the command **rtc set arbitrary_time** ***ISOtimeString***. An ISO formatted date time string has the following format: ***YYYY-MM-DD*****T*****hh*****:*****mm*****:*****ss***, where **YYYY** is a four-digit year, **MM** is month (1 - 12), **DD** is day-of-month (1-31), **hh** is hour (24-hour format), **mm** is minute, and **ss** is second. The punctuation contained in this format (the **T** and dashes **-**) must be left as shown.

The second method of setting date-time, using the GPS receiver, requires that the serial channel to the GPS board be opened. Not doing so will result in the call to set the RTC to timeout and fail. Once the serial channel has been opened, the command **rtc set date_time** will read the current date time from the GPS receiver, convert it to 1980 epoch format, and write it to the RTC. The GPS emits a time synchronization signal every second. Date-time is written to the RTC as soon as this time synch signal goes high. One concludes the RTC setting session by closing the serial channel to the GPS.

GPS

The GPS family of commands involves the reading of time-date and location information from the IRMA MC's GPS receiver board. The GPS is interfaced to the MC by means of a 4800 bps serial channel. Consequently, all commands to the GPS must be preceded by issuing the command to open the GPS serial channel: **gps serial open**. After the transaction with the GPS has been completed, the GPS serial channel should be closed using **gps serial close**.

Datetime is read from the GPS receiver using the command **gps read date_time**. The data returned is contained in a colon-delimited string: ***year* : *month* : *day* : *hour* : *minute* : *second***. Epoch time, returned in 1980 epoch format, is read by calling **gps read epoch_time**. Latitude-longitude data is read with the command **gps read lat_lon**. Data is returned as a colon-delimited string.

IRMA

This family of commands is used to perform system-level activities on the IRMA system as a whole. The statement **irma state off** forces the IRMA MC software to reboot. The statement **irma read uptime** returns the number of elapsed seconds since the IRMA MC was powered up or last rebooted. The value returned by this command is represented in floating-point seconds.

SUN_SENSOR

The solenoid-controlled shutter protecting the filter and IR detector can be controlled by the software using the **sun_sensor** commands. The state of the sun shutter is read using **sun_sensor read shutter_state**. A return value of 0 indicates that the shutter is closed (covering the filter and detector), while a value of 1 indicates the shutter is open. A photo cell coupled with discrete logic automatically closes the sun shutter when IRMA's line of sight comes within $\pm$ 15° of the sun (or any bright light source), is read using the command **sun_sensor read state**. A return value of 1 indicates that the sun sensor is detecting a bright light source in its line of sight. A value of zero indicates the opposite.

CRYO

The Stirling engine (cryo cooler) that cools the IR detector is controlled by the **cryo** family of commands. Before attempting to send commands to the cyro cooler, the serial communication channel to the cooler must be opened with the command **cryo serial open**. Likewise, the channel is closed with the command **cryo serial close**.

cryo read comp_amp
Returns the compressor amplitude value as a floating point value.

cryo read set_point
Returns the cryo cooler's set point temperature in Kelvin. The return value is a floating point number.

cryo read mode
Returns an integer code representing the operational mode of the cryo cooler controller.

cryo read curr_temp
Returns the current temperature in Kelvin of the cryo cooler's cold finger. The return value is a floating point number.

cryo read osc_freq
Returns the cryo cooler's oscillation frequency, which is the frequency of the piston inside the cold finger. The oscillation frequency is expressed in cycles per second (Hz).

cryo set manual_mode
This command sets the cryo cooler into manual mode, which powers the cryo cooler down.

cryo set set_point *temperature*
This command sets the desired temperature of the cryo cooler's cold finger. This command will successfully execute only when the cryo cooler is in **manual_mode**.

cryo set auto_mode
The cryo cooler begins to cool when this command is received. Cooling is a gradual process, taking roughly 30 minutes according to the cryo cooler controller's internal configuration settings. When target set point temperature is reached, the controller will maintain this temperature as long as it is in **auto mode**.

ADC

Control of the Cirrus CS5534 Delta-Sigma ADC is handled by the **adc** family of commands. Before A/D conversions can be performed, the ADC must be first initialized using the

resynch command, **adc init resynch**, then reset using the **reset** command, **adc init resynch**. The last step involves configuring each of the ADC's four channels with the command: **adc set csr *arguments***. The following list describes each of the CS5534 IRMAscript functions in depth.

adc init resynch
Calling this command puts the ADC's serial port into a known state. When using the ADC for the first time, it is recommended that this command be called in order to ensure that the ADC will successfully accept serial commands. At low level, this command serially writes 15 bytes of the value 0xFF, followed by 1 single byte valued 0xFE.

adc init reset
This command resets the ADC and sets its fundamental parameters. At low level, the reset command sets the **RS** bit in the CS5534's configuration register, which has the effect of forcing a system reset.

adc set offset *channel value*
Set offset command allows the user to configure each of the CS5534's four input channels' offset registers. Channels 1 through 4 can be specified, while the value field can accept offset values ranging between -2^{23} and 2^{24}. The offset value represents the fraction of the input span that must be applied to the output value of the ADC to shift it up or down. Offset values must be defined in ADC units. For example, an offset of 255 refers to a positive offset of $255/2^{24}$ of the ADC 's input span. ADC channels configured for taking unipolar samples have an input span of 2^{24}, while channels configured for bipolar mode have an input span of 2^{23} [38]. The CS5534's default offset setting is 0.

adc set gain *channel value*
Similar to the **set offset** command, set gain allows the user to manually set a gain value, ranging from 64 to 2^{-24}, to channels 1 through 4. When a channel's gain register is set, the offset is subtracted from the A/D sample value, after which this result is multiplied with the gain value. IRMA currently does not use custom gain settings. Instead, gain and offset are applied to the data in post processing. The CS5534's default gain value is 1.

adc read gain *channel*
adc read offset *channel*
These two commands read the current gain and offset values from the CS5534 ADC.

adc set csr *channel gain word-rate polarity*
Each of the CS5534's four input channels can be configured in terms of signal gain, accuracy (word rate) and input span (polarity). Channel settings are

stored in the CS5534's four channel setup registers (CSR). Gain as defined in the CSR is separate from the gain contained in the channel gain registers described earlier. Gain values can be defined with the IRMAscript constants or their respective numeric values, as shown in Table A.2.

Table A.2: CS5534 ADC gain settings in IRMAscript.

Gain	Value
CS5534_GAIN_1	1
CS5534_GAIN_2	2
CS5534_GAIN_4	4
CS5534_GAIN_8	8
CS5534_GAIN_16	16
CS5534_GAIN_32	32
CS5534_GAIN_64	64

Resolution refers to the number of noise free bits contained in the A/D sample value. The longer the ADC integrates the analog signal, the greater the accuracy (or resolution) of the digitized sample. Table A.3 lists the different sample resolutions in terms of noise-free resolution bits, integration time (in milliseconds), and word-rate. Input span of digitization can be either unipolar, where A/D values contain values ranging from 0 to 2^{24} - 1, or bipolar, which allow signed values ranging from -2^{23} to 2^{23} -1. Table A.4 lists constants and their respective numeric values that can be applied to the polarity field.

Table A.3: CS5534 ADC sample resolution settings in IRMAscript.

.

Resolution	Bits	Integration	ms	Word rate
CS5534_RES_23	23	CS5534_INTEG_538	538	7
CS5534_RES_22_SLOW	22	CS5534_INTEG_269	269	15
CS5534_RES_22_FAST	22	CS5534_INTEG_136	136	30
CS5534_RES_21_SLOW	21	CS5534_INTEG_69	69	60
CS5534_RES_21_FAST	21	CS5534_INTEG_35	35	120
CS5534_RES_18	18	CS5534_INTEG_19	18.2	240
CS5534_RES_17_SLOW	17	CS5534_INTEG_10	9.9	480
CS5534_RES_17_FAST	17	CS5534_INTEG_6	5.7	960
CS5534_RES_16	16	CS5534_INTEG_4	3.6	1920
CS5534_RES_13	13	CS5534_INTEG_2	1.5	3840

adc read csr *channel*
The contents of each of the CSR channels can be read using this command. A colon-delimited string having the following format is returned:

channel* : *gain* : *word-rate* : *polarity

Table A.4: CS5534 ADC polarity settings in IRMAscript.

.

Polarity	Value
CS5534_UNIPOLAR	1
CS5534_BIPOLAR	2

adc init rw_test
Primarily used for troubleshooting and verification, the read-write test command tests the ADC to ensure that the IRMA software can communicate with it. An arbitrary value is written to one of the CS5534's offset registers, then that value is read back from the offset register. If the two values are identical, the test is deemed a success, and a value of 1 is returned. A failed read-write test returns a 0 (zero). This command should be followed with an ADC system reset in order to clear the dummy value in the offset register.

adc sample *sample_type channel*
This command initiates an A/D sample on a given ADC channel. The returned value is given in ADC units, thus it must be interpreted according to the channel's polarity setting: bipolar or unipolar. Two types of samples can be taken: those synchronized to the 450 Hz chop wheel, specified with the **on_int** parameter, or samples not synchronized to the chop wheel, specified with the **no_int** parameter. When the **on_int** parameter is specified, the A/D sample commences when the chop wheel signal reports a logic level of 1. The **channel** parameter is mapped to 19 separate channels (only 11 channels are present in the older Rabbit based implementation).

SCAN

The scanning process involves repeatedly sampling the IR signal and temperature / pressure / humidity channels at some interval. Scanning differs from reading an ADC channel directly in that the A/D sampling process is contained separate real-time task, and uploads the data to a separate network port on the IRMA CP. This allows the MC to service other commands while the data collection process is executing, such as moving the Alt-Az mount, or querying the status of the cryo cooler. The Alt-Az serial communications channel must be opened before executing the scan command. Additionally, it is vital that the Alt-Az channel be left open for the duration of the scan. Closing the channel during a scan will lead to scan failure, which results in the scan terminating itself.

scan read status
Returns the value 1 if a scan is currently executing on the MC, otherwise the value 0 is returned.

scan signal on_int
Forks the data collection process task, where the IR signal is sampled on the positive edge of the notch notch interrupt signal. Temperature, pressure and humidity channels are each sampled following one IR signal sample in a round-robin fashion.

scan signal no_int
IR signal, temperature, pressure and humidity are sampled, but the IR signal A/D conversion is not synchronized to the notch interrupt.

ALTAZ

Movement and control of the altitude and azimuth axes is handled by the **altaz** family of commands. Given that the AAC is connected to the MC over a serial communications link, the AAC-MC serial connection must be opened before any altaz command can be sent. Failing to open the serial port when sending AAC commands produces subtle errors that are hard to track down. Each of the altaz command groups will be examined in detail.

Commands destined for the AAC are sent over the MC/AAC serial link using the serial packet communications protocol.

altaz serial open
altaz serial close
These commands respectively open and close the serial channel from the MC to the AAC.

altaz init altaz
Once the Alt-Az serial channel has been opened, the first command that should be sent to the AAC is the **init altaz** command. This command has the effect of initializing the AAC's two-channel optical encoder chip that is responsible for digitizing axis encoder positions. Upon initializing the optical encoder chip, altitude and azimuth axis positions are set to 90,000 encoder units. There are 8192 encoder units in one revolution.

altaz init axes *axis*
Upon using the AAC for the first time, or where re-initialization is required, the axes should be sent to their default positions. The **init axes** command performs a homing operation, whereby it determines the clockwise and counter-clockwise optical limits on both axes. When axis homing has completed, altitude and azimuth positions are set to position 0 (in encoder units). The axis parameter can be defined in three ways: ***altitude***, ***elevation***, or ***azimuth***.

altaz init motor
The gearboxes and motor controllers used in the IRMA AAC differ from unit to unit. In order to deal with these variations, gear ratios, motor RPM values, and other configuration information unique to the given IRMA unit is contained in a configuration file. By issuing this command, the CP uploads motor configuration information, stored in the particular IRMA unit's configuration file, into the AAC. Without this information, the AAC cannot calculate motor speeds or slewing times. Therefore, it is vital that **init motor** be called before any axis movement is attempted.

altaz init servo
The AAC uses a servo loop, based on proportional-integral-derivative (PID) motion control algorithm, to control axis movement. The PID servo control algorithm has three constants, **P**, **I** and **D**, which are unique to each Alt-Az mount. The **init servo** command loads the PID constants for the altitude and azimuth axes into the AAC. If the servo-controlled movement **move_to** command is going to be used, servo parameters must be loaded into the AAC beforehand. The **slew_to** non-servo movement command does not require servo parameters to be set. The command **init servo** may be called while the AAC is idle (not moving) as many times as required, which is particularly helpful if the user is "tuning" the servo algorithm.

altaz init ping
The Alt-Az **ping** command is used to check if the AAC is on-line, ready to receive commands. If the AAC is alive and on-line, it returns a three-field, colon delimited string of the form:

987654321 : ***123456789*** : ***uptime***

If the AAC is not on-line or unresponsive, the three fields will contain the code **999999999**. The **Uptime** field indicates the number of elapsed CPU ticks since the IRMA MC was booted. Each CPU tick is 1/64 seconds, therefore to convert this value to elapsed seconds, divide it by 64. Uptime can also be read using the command **altaz read uptime**.

altaz set alt_offset *offset_value*
altaz set az_offset *offset_value*
The **set *offset*** commands are provided in order to allow the user to define virtual fiducial points, thus avoiding the necessity of physically orienting IRMA's

fiducial (the axis limits) to external physical references, such as zenith for elevation, or North for azimuth. By providing an offset value defined in optical encoder units, IRMA's AAC calculates all axis moves relative to the offset position instead of the default physical limit. Axis offsets is the angle between the physical limit and the position where the physical reference is determined to be. The default offset value for both axes is zero.

altaz state poslog ***action***
AAC position logging is controlled using this command. Three separate activities can be performed: log initialization, log enabling and log disabling. Upon AAC start-up, the position log is allocated, zero-filled, and its index pointer is pointed to the first element in the position log array. This action should be explicitly called before using the position log by using the **log_clear** constant in the action parameter. One begins logging an axis movement by calling this command using the **log_enable** constant. Logging is stopped by using the **log_disable** constant.

altaz state halt
Stop movement immediately in both axis.

altaz state reboot
Perform a soft reset (or reboot) of the AAC software running on the Alt-Az controller. The master controller software is not affected.

altaz move_to ***axis alt_d alt_m alt_s az_d az_m az_s speed***
Servo-controlled movements, which track a theoretical velocity versus position profile, are performed using the **move_to** command. Three parameters must be provided: the axis to be moved, the destination angle, and the axis rotation speed, specified in degrees per second. Options available for **axis** include: **altitude**, **azimuth**, and **dualaxis**.

The destination angle is defined in degree-minute-second format, where altitude degrees, minutes and seconds occupy fields 4, 5 and 6 respectively (assuming field 1 refers to the "**altaz**" symbol). For single-axis movement, altitude or azimuth destinations should be written to fields 4, 5 and 6, while fields 7, 8 and 9 should be zero-filled. For dual-axis movements, altitude should occupy fields 4, 5 and 6, and azimuth should occupy fields 7, 8 and 9. Field 10 is populated with the desired axis speed. In the case of dual-axis movement, the speed refers to the diagonal speed between the two moving axis, or rather, the speed required for both axes to meet at the final altitude/azimuth coordinate.

Since this is a servo-controlled move command, movements are continuous, and are consequently limited to the speed options provided by the given Alt-Az mount's gearing. The slowest speed possible with this command occurs when the axis motor is driven at 0 volts, which corresponds to 500 motor RPM. The axis will rotate considerably slower than the minimum motor RPM, due to the motor's gear box and drive belt. Be aware that it is impossible to perform movements slower than the minimum motor RPM. For performing movements slower than the minimum achievable speed, there is the **slew_to** command.

altaz slew_to dms *axis alt_d alt_m alt_s az_h az_m az_s speed*
Usage of the **slew_to** command is identical to **move_to**. What differs is the range of speeds available, and the fact that movement is not servo controlled. When a speed less than the minimum achievable speed is selected, **slew_to** goes into stepping mode, where the given slew path is broken up into sub-degree, one encoder unit steps. The axis (or axes) rotate for the duration calculated from the slew path length and the requested speed.

Mention should be made about the relationship between offset angles and axis moves. Offset angles for each axis are measured from the counterclockwise limit switch in the clockwise direction. The AAC rotates to the requested angle, to which is added the currently defined offset angle. The offset angle should be considered as zero degrees. Altitude angles less than the offset angle are reported as negative angles, while azimuth angles less than the offset wrap at 360 degrees, because the azimuth axis has the ability to rotate a full 360 degrees.

Full rotational movement allows for the possibility of destination angles that lie beyond the far (clockwise) limit. In such cases, the AAC rotates the azimuth axis in the opposite direction to the target angle lying beyond the far limit. In the case of low-speed, small-distance movements that result in destinations crossing the rotational limit, the AAC drives the axis to the destination angle in the opposite direction at high speed in order to eliminate the annoyance of slewing nearly 360 degrees as low speed.

altaz read position
This command returns a three-value colon-delimited string containing altitude and azimuth values respectively. The third field contains scan status: 1 when a scan is executing, and 0 when no scan is running.

altitude : ***azimuth*** : ***scan_status***

read position is the most common query request to the AAC because during scans, the MC requests axis positions for each data point collected.

altaz read task_status
In order to remain responsive to incoming commands, the AAC executes axis movements separate from the main dispatcher task. **read task_status** allows external processes, such as an executing IRMA script, to check up on an ongoing AAC movement, and determine when the operation has completed. Task status is returned as one of three codes: code 0 indicates there is no axis movement task operating, while code 2 indicates a task is executing. Code 1 is returned when the AAC is dispatching a long-duration job to one of its available tasks. It is rare that this code would be encountered, and should be considered simply as a running task.

altaz read alt_offset
altaz read az_offset
These two commands respectively return the currently defined altitude and azimuth offset values in optical encoder units. There are 8192 units per revolution.

altaz read poslog_state
The **poslog** commands are used primarily for Alt-Az servo tuning. They allow the user to collect axis motion data necessary for tuning the AAC's PID servo control loop. The **read poslog_state** command returns the current operation mode of the position log, the table in the AAC that is used to store servo and position data. Three states can be reported: code 1 indicates the position log is enabled. Code 0 indicates the position log is disabled. Code 2 is returned if the position log was not initialized during AAC start-up. This can happen if an extended memory allocation failure occurred on board the AAC Rabbit processor.

altaz read poslog_range
Calling this command returns the dimensions of the position log, a memory array aboard the AAC containing position and servo data. A four field colon-delimited string is returned:

min array index : ***max array index*** : ***curr array index*** : ***NULL***

The range of data readable from the AAC's position log is found between the minimum array index and the current array index inclusive. Reading values beyond the maximum array index will result in a memory read error on the AAC.

altaz read poslog_data *index*
Given some index value, this command returns the position log entry at that index. A four field, colon-delimited array is returned:

DAC val : ***rel pos*** : ***theor pos*** : ***error val***

DAC val contains the 8-bit unsigned integer that is written to the AAC's DAC, which in turn controls axis speed. **Rel pos** refers to the actual position of the axis relative to its start position, and is given in optical encoder units. **Theor pos** is the calculated theoretical axis position, also given in optical encoder units. It is this theoretical displacement path that the PID servo must track. The last field, **error val**, contains the PID algorithm error value. All four data are necessary in the servo tuning process.

Appendix B

AltAz Commands

Table B.1 contains a list of the command codes and their respective aliases used in communication between the IRMA Master Controller and the Alt-Az Controller.

Code	Alias
1	ALTAZ_READ_CURRENT_POSITION
2	ALTAZ_MOVETO
3	ALTAZ_HALT
5	ALTAZ_PING
6	ALTAZ_SET_ALT_OFFSET
7	ALTAZ_SET_AZ_OFFSET
8	ALTAZ_SET_RTC
9	ALTAZ_SLEW_STATUS
10	ALTAZ_MOVE_AXIS
11	ALTAZ_INIT
12	ALTAZ_INIT_AXES
13	ALTAZ_INIT_SERVO_ELEV
14	ALTAZ_INIT_SERVO_AZIM
15	ALTAZ_INIT_MOTOR
16	ALTAZ_SLEWTO
17	ALTAZ_RD_POSLOG_RANGE
18	ALTAZ_RD_POSLOG_DATA
19	ALTAZ_INIT_POSLOG
20	ALTAZ_POSLOG_STATE
21	ALTAZ_REBOOT
22	ALTAZ_READ_ALT_OFFSET
23	ALTAZ_READ_AZ_OFFSET

Table B.1: IRMA AAC command codes sent over MC AAC serial link.

Appendix C

Configuration and Installation

C.1 CP configuration file options

A complete list and description of parameters that can be defined in a CP configuration file is given below.

Data_Port
The TCP/IP socket port that is used by the MC to send scan data to.

Antenna
The identification number of the antenna that the given IRMA unit is associated with. This parameter is not always used.

ElevGearReduction
The gear reduction ratio of the gear box driving the elevation axis.

AzimGearReduction
The gear reduction ratio of the gear box driving the azimuth axis.

BeltReduction
The gear reduction ratio caused by the drive belt. The total gear reduction ratio of a given gear is the sum of its gear box reduction ratio and the belt reduction ratio.

MaxMotorRPM
This is the vendor-specified maximum motor rotational rate, generated when full scale voltage is applied to the motor controller unit.

MinMotorRPM
This is the vendor-specified minimum motor rotational rate, generated when zero volts is applied to the motor controller unit.

MaxGearRPM
The maximum recommended rotational rate of the gear head (not the motor). This value is provided by the motor vendor.

elev_kProp, elev_kInteg, elev_kDeriv
Servo constants for the elevation axis motor. The three constants refer to the proportional, integration and derivative constants (PID), which must be determined by the user by tuning the servo algorithm.

azim_kProp, azim_kInteg, azim_kDeriv
Servo constants for the azimuth axis motor. The three constants refer to the proportional, integration and derivative (PID) constants, which must be determined by the user by tuning the servo algorithm.

Location
This refers to the name of the site where this given IRMA unit is located.

Cooler
The model number of the cryo cooler associated with this given IRMA unit

Board
An ID number which identifies the IRMA motherboard associated with this given IRMA unit.

CalibrateLow
This is the ADC count when the IR channel measures the unpowered shutter blackbody calibration source (cold). Calibration of the calibration target in hot and cold states (powered and unpowered) relates the IR measurement with a temperature reading from the same target.

CalibrateHigh
This is the ADC count when the IR channel measures the powered-up (hot) shutter calibration source. See the description of CalibrateLow for calibration details.

C.2 Perl Module Installation

Additional Perl modules must be installed for the IRMA software to run properly. Perl modules can be installed using the Comprehensive Perl Archive Network (CPAN) using the following command

```
sudo perl -MCPAN -e shell
```

This will bring up a `cpan>` prompt. At this prompt, simply enter the command

```
cpan> install <module_name>
```

To install custom IRMA modules, the files simply need to be copied into the `~/IRMA/IRMA/` directory.

C.2.1 System Software Perl Modules

The following modules must be installed on the PC/104 for the system software to operate properly.

```
IO::Handle
IO::Socket
Net::Ping
Socket::IO
SpreadSheet::ParseExcel
Time::Local
Time::Piece

IRMA::CCIT16_CRC
IRMA::commander
IRMA::packetComm
```

C.2.2 Control and Visualisation Software Perl Modules

```
Time::Piece;
Time::Seconds;
List::Util qw{max min};
Tk;
Tk::Balloon;
Math::Round qw(:all);
Math::Trig;
Expect;

IRMA::Data;
IRMA::scripter;
IRMA::queue;
IRMA::BoxInfo;
IRMA::gui_graph;
```

C.3 Sample Daily Tasks File

Shown below is a typical daily tasks file, which monitors the status of the unit every 10 minutes, and performs a calibration routine every two hours.

```
| 1 accuracy allowance function id label repeat reptime time
At|5|script status_check.irma|5699|status|1|00:10:00|00:04:00
At|5|script shutterOpen.irma|5700|shutterOpen|1|02:00:00|00:40:00
At|5|script shutterClose.irma|5703|shutterClose|1|02:00:00|00:00:00
At|5|script bbOn.irma|5704|bbOn|1|02:00:00|00:10:00
At|5|script bbOff.irma|5706|bbOff|1|02:00:00|00:41:00
```

C.4 AutoTasks.conf

Shown is a typical `autoTasks.conf` file. It must be located in the `~/IRMA/Config/` directory.

```
daily_tasks on 1
daily_tasks delay 30
daily_tasks boxes 1
daily_tasks unkhumidrun 1
skymap on 0
skymap delay 20
skymap boxes 1
skymap unkhumidrun 1
humidity on 1
humidity humid 70
humidity delay 60
humidity boxes 1
cooler on 1
cooler comp_amp 1
cooler temp 1
cooler delay 30
cooler boxes 1
cooler osc_freq 0
```

References

[1] Graeme J. Smith. *An Infrared Radiometer For Millimeter Astronomy*. MSc thesis, University of Lethbridge, Lethbridge, AB (2000).

[2] Ian M. Chapman and David A. Naylor. *Fourier Transform Spectroscopy/ Hyperspectral Imaging and Sounding of the Environment*. In (edited by editor). Technical Digest (CD), paper HTuD2, OSA (2005).

[3] "Infrared Windows." Infrared Processing and Analysis Center. URL `http://www.ipac.caltech.edu/Outreach/Edu/Windows/irwindows.html`.

[4] O. P Lay. "MMA Memo 209: 183 GHz Radiometric Phase Correction for the Millimeter Array." *ALMA Memos* (1998). URL `www.alma.nrao.edu/memos/html-memos/abstracts/abs209.html`.

[5] M. C. Wiedner and R. E. Hills. *Imaging at Radio through Submillimeter Wavelengths*. In Mangum, J. G. and Radford, S. J. E. (edited by editor), **217**:327–+ (2000).

[6] S.J.E. Radford and R.A. Chamberlin. "MMA Memo 334: Atmospheric Transparency at 225 GHz over Chajnantor, Mauna Kea, and the South Pole." *ALMA Memos* (2000). URL `www.alma.nrao.edu/memos/html-memos/abstracts/abs334.html`.

[7] J. Duan, M. Bevis, P. Fang, Y. Bock, S. Chiswell, S. Businger, C. Rocken, F. Solheim, T. van Hove, R. Ware, S. McClusky, T. A. Herring, and R. W. King. "GPS Meteorology: Direct Estimation of the Absolute Value of Precipitable Water." *Journal of Applied Meteorology*, **35**:830–838 (1996).

[8] D. A. Naylor, I. M. Chapman, and B. G. Gom. *Proc. SPIE Vol. 4815, p. 36-45, Atmospheric Radiation Measurements and Applications in Climate,*. In Shaw, J. A. (edited by editor), **4815**:36–45 (2002).

[9] "Blue Sky Transmittance and Radiance Atmospheric Model." Product information from Blue Sky Spectroscopy, Lethbridge, AB, Canada. URL `http://www.blueskyinc.ca/`.

[10] H. J. P. Smith, D. J. Dube, M. E. Gardner, S. A. Clough, F. X. Kneizys, and L. S. Rothman. "FASCODE: Fast Atmospheric Signature Code (Spectral Transmittance and Radiance)." Technical Report AFGL-TR-78-0081, Air Force Geophysics Laboratory, Hanscom AFB, Massachusetts, U.S.A. (1978).

[11] L.S. Rothman, D. Jacquemart, A. Barbe, D. Chris Benner, M. Birk, L.R. Brown, M.R. Carleer, Jr., C. Chackerian, K. Chance, L.H. Coudert, V. Dana, V.M. Devi, J.-M. Flaud, R.R. Gamache, A. Goldman, J.-M. Hartmann, K.W. Jucks, A.G. Maki, J.-Y. Mandin, S.T. Massie, J. Orphal, A. Perrin, C.P. Rinsland, M.A.H. Smith, J. Tennyson, R.N. Tolchenov, R.A. Toth, J. Vander Auwera, P. Varanasi, and G. Wagner. "The HITRAN 2004 molecular spectroscopic database." *Journal of Quantitative Spectroscopy and Radiative Transfer*, **96**(2):139–204 (2005).

[12] D. A. Naylor, R. T. Boreiko, T. A. Clark, R. J. Emery, B. Fitton, and M. F. Kessler. "Atmospheric emission in the 20-micron window from Mauna Kea." *Publications of the Astronomical Society of the Pacific*, **96**:167–173 (1984).

[13] Richard Querel. *IRMA As A Site Testing Instrument.* MSc thesis, University of Lethbridge, Lethbridge, AB (2007).

[14] MCT infrared detector #KMPC19-1-SP, Kolmar Technologies, Inc. URL `http://www.kolmartech.com`.

[15] C. Lee, P. A. R. Ade, and C. V. Haynes. "Self Supporting Filters for Compact Focal Plane Designs." *ESA SP-388*, p. 81 (1996).

[16] Robin R. Phillips, Vic Haynes, David A. Naylor, and Peter Ade. "Simple method for antireflection coating ZnSe in the 20 μm wavelength range." *Appl. Opt.*, **47**(7):870–873 (2008). URL `http://ao.osa.org/abstract.cfm?URI=ao-47-7-870`.

[17] Rabbit Semiconductor, Inc., Davis, CA. *RCM2100 RabbitCore Data Sheet* (2005). URL `www.rabbitsemiconductor.com/products/rcm2100/rcm2100.pdf`.

[18] WinSystems, Inc. 715 Stadium Drive, Arlington, Texas, 76011, USA. URL `http://www.winsystems.com/`.

[19] Diamond Systems Corporation. *Emerald-MM-DIO Quad RS-232 + 48 Digital I/O PC/104 Users Manual.* Newark, CA (2002). URL `www.diamondsystems.com`.

[20] Honeywell Hymatic, Redditch, Worcestershire, UK. URL `http://www.hymatic.co.uk/`.

[21] Rabbit Semiconductor, Inc., Davis, CA. *RCM2000 RabbitCore Data Sheet* (2005). URL `www.rabbitsemiconductor.com/products/rcm2000/rcm2000.pdf`.

[22] US Digital Corporation, Vancouver, WA. *E6 Optical Kit Encoder* (2005). URL `www.usdigital.com/data-sheets/E6 Data Sheet.pdf`.

[23] US Digital Corporation, Vancouver, WA. *LS7266R1 Encoder to Microprocessor Interface Chip* (2004). URL `www.usdigital.com/products/ls7266/`.

[24] Ian Sean Schofield. *The IRMA III Control and Communication System.* MSc thesis, University of Lethbridge, Lethbridge, AB (2005).

[25] SHDesigns Download Managers, SHDesigns. URL `http://shdesigns.org/rabbit/download.shtml`.

[26] Larry Wall, Tom Christiansen, and Jon Orwant. *Programming Perl*. 3rd edition. O'Reilly Media, Inc.: Sebastopol, CA, U.S.A (2000).

[27] European Space Agency. "Packet Telecommand Standard." PSS-04-017 Issue 2 (1992).

[28] Z World, Inc., Davis, California. *MicroC/OS-II User's Manual* (2003). URL `www.zworld.com/documentation/docs/manuals/DC/DCModules/ModUcos.pdf`.

[29] Z World, Inc., Davis, California. *Dynamic C User's Manual* (2004). URL `www.zworld.com/documentation/docs/manuals/DC/DCUserManual/DCPUM.pdf`.

[30] crontab, The Open Group Base Specifications Issue 6,IEEE Std 1003.1, (2004). URL `http://www.opengroup.org/onlinepubs/009695399/utilities/crontab.html`.

[31] Andrew Tridgell. "Rsync, remote file syncronization system." URL `http://rsync.samba.org`.

[32] International Organization for Standardization. *ISO 8601:2004. Data elements and interchange formats — Information interchange — Representation of dates and times*. International Organization for Standardization: Geneva, Switzerland (2004). URL `http://www.iso.ch/cate/d26780.html`.

[33] Ian Myles Chapman. *The Atmosphere Above Mauna Kea At Mid-Infrared Wavelengths*. MSc thesis, University of Lethbridge, Lethbridge, AB (2003).

[34] Lake Shore Cryotronics, Inc. 575 McCorkle Blvd, Westerville, Ohio, 43082, U.S.A. URL `http://www.lakeshore.com/`.

[35] Data Translation, Inc., Marlboro, MA. *DT9800 Series User's Manual* (2007). URL `ftp://ftp.datx.com/Public/DataAcq/DT9800Series/Manuals/um9800.pdf`.

[36] Regan Dahl, Richard Querel, and Matthias Schöck. *Proc. IR, THz and MMW Applications in Astronomy, Atmospheric and Environmental Science*. In (edited by editor) (2007).

[37] G. R. Davis, D. A. Naylor, M. J. Griffin, T. A. Clark, and W. S. Holland. "Broadband Submillimeter Spectroscopy of HCN, NH_3, and PH_3 in the Troposphere of Jupiter." *Icarus*, **130**:387–403 (1997).

[38] Cirrus Logic, Inc., Austin, Texas. *CS5531/32/33/34 Product Data Sheet* (2004). URL `www.cirrus.com/en/pubs/proDatasheet/CS5531323334_F1.pdf`.